
Mobile SDK 開発ガイド

Salesforce Mobile SDK (iOS ネイティブ、Android ネイティブ、React ネイティブ、およびハイブリッド)



本書の英語版と翻訳版で相違がある場合は英語版を優先するものとします。

© Copyright 2000–2024 Salesforce, Inc. All rights reserved. Salesforce およびその他の名称や商標は、Salesforce, Inc. の登録商標です。本ドキュメントに記載されたその他の商標は、各社に所有権があります。

目次

第 1 章: モバイル開発の概要	1
Salesforce モバイルアプリケーションについて	2
Salesforce モバイルアプリケーションのカスタマイズかカスタムアプリケーションの作成か	3
このガイドの内容	4
バージョン	5
フィードバックの送信	5
第 2 章: Salesforce Mobile SDK 開発の概要	6
ネイティブ、HTML、およびハイブリッド開発について	7
クイックスタート	10
第 3 章: Mobile SDK 11.1 の新機能	11
最近のリリースの新機能	14
第 4 章: Mobile SDK 11.1 for iOS および Android の使用開始	23
Developer Edition 環境か Sandbox 環境か	24
iOS および Android の開発の前提条件	25
Salesforce プラットフォームへのサインアップ	26
接続アプリケーションの作成	27
接続アプリケーションを作成する	27
Mobile SDK のインストール	29
Node.js、npm、Git コマンドラインのインストール	29
Mobile SDK npm パッケージ	29
iOS の準備	31
Android の準備	31
Mobile SDK npm パッケージのアンインストール	32
Mobile SDK GitHub リポジトリ	32
Mobile SDK サンプルアプリケーション	34
サンプルアプリケーションのインストール	34
第 5 章: Mobile SDK アプリケーション (5.0 以降) の更新	39
Maven を使用した Android アプリケーションの Mobile SDK ライブラリの更新	43
第 6 章: ネイティブ iOS の開発	45
iOS ネイティブクイックスタート	46
iOS の基本的な要件	46
forceios を使用した iOS プロジェクトの作成	47
カスタムテンプレートを使用したアプリケーションの作成	50

iOS Swift プロジェクトの手動作成	54
Xcode Swift プロジェクトの作成	54
Mobile SDK ライブラリのプロジェクトへの追加	55
プロジェクトのビルド設定の定義	56
オプション 1: Mobile SDK のテンプレートファイルをインポートする	58
オプション 2: Mobile SDK の設定コードを手動で追加する	60
CocoaPods と Mobile SDK の使用	72
Mobile SDK Pod の更新	75
Mobile SDK での Carthage の使用 (サポート対象外)	75
ネイティブ iOS アプリケーションの開発	76
ログインとパスワードについて	76
Salesforce Mobile SDK での Swift の使用	78
iOS ネイティブテンプレートアプリケーション	79
アプリケーションフローの概要	81
SDK マネージャークラス	83
AppDelegate クラス	87
SceneDelegate クラス	89
RootViewController クラス	92
ネイティブ Swift テンプレート	93
Mobile SDK での Salesforce REST API の使用	96
認証エラーの処理	116
Mobile SDK アプリケーションでの iPadOS のサポート	117
Mobile SDK アプリケーションでの Catalyst のサポート	119
Mobile SDK での iOS アプリケーション拡張の使用	120
Signposts を使用したプロファイリング	127
iOS サンプルアプリケーション	129
第 7 章: ネイティブ Android の開発	130
Android ネイティブクイックスタート	131
ネイティブ Android の要件	131
forcedroid を使用した Android プロジェクトの作成	132
カスタムテンプレートを使用したアプリケーションの作成	136
Android Studio でのサンプルプロジェクトの設定	139
Android プロジェクトファイル	140
ネイティブ Android アプリケーションの開発	141
Android アプリケーションの構造	141
ネイティブ API パッケージ	142
ネイティブクラスの概要	142
パスワードの使用	154
リソースの処理	155
REST API の使用	158
Batch 要求および Composite 要求	162
認証されていない REST 要求	164
ネイティブ Android アプリケーションのログインの延期	164

Android テンプレートアプリケーション: 詳細	168
Android サンプルアプリケーション	173
第 8 章: HTML5 およびハイブリッド開発	175
はじめに	176
HTML5 および JavaScript の使用	176
HTML5 の開発要件	176
マルチデバイス戦略	176
Visualforce を使用した HTML5 コンテンツの配信	180
Salesforce データへのアクセス: コントローラーと API	181
ハイブリッドアプリケーションのクイックスタート	184
ハイブリッドアプリケーションの作成	185
ハイブリッド開発について	186
Cordova を使用したハイブリッドアプリケーションの開発	186
ハイブリッドリモートアプリケーションの開発	192
ハイブリッドサンプルアプリケーション	193
ContactExplorer ハイブリッドサンプルの実行	196
モバイルデバイスでのハイブリッドアプリケーションのデバッグ	202
Android デバイスのハイブリッドアプリケーションのデバッグ	202
iOS デバイスで実行されているハイブリッドアプリケーションのデバッグ	202
iOS 7 ハイブリッドアプリケーションでのステータスバーの制御	203
ハイブリッドアプリケーションの JavaScript ファイル	204
バージョン設定および JavaScript ライブラリの互換性	204
例: 適切な JavaScript ライブラリの提供	207
ハイブリッドアプリケーションでのセッション管理	208
ログインの延期	211
第 9 章: ハイブリッドアプリケーションでの Lightning Web コンポーネントの使 用	212
第 10 章: React Native 開発	215
forcereact を使用した React Native プロジェクトの作成	218
React Native プロジェクトでの TypeScript の使用	221
React Native アプリケーションでの Mobile SDK コンポーネントの使用	222
React Native アプリケーション用の Mobile SDK ネイティブモジュール	223
React Native を使用した Mobile SDK サンプルアプリケーション	224
ログインの延期	225
バイナリコンテンツのアップロード	227
第 11 章: オフライン管理	229
SmartStore を使用したオフラインデータの安全な保存	230
SmartStore について	232
ハイブリッドおよびネイティブアプリケーションでの SmartStore の有効化	235
既存の Android アプリケーションへの SmartStore の追加	235

ユーザーベースストアの作成およびアクセス	236
グローバル SmartStore の使用	238
設定ファイルを使用するスーパの登録	240
インデックスパスでの配列の使用	250
スーパの入力	251
スーパからのデータの取得	254
スマート SQL クエリ	263
全文検索クエリの使用	265
クエリ結果の処理	270
データの挿入、更新、および更新/挿入	271
大きいスーパ要素での外部ストレージの使用	275
スーパ要素の削除	276
スーパの管理	278
ストアの管理	285
SmartStore インспекターを使用したテスト	288
擬似 SmartStore の使用	288
Mobile Sync のスーパの準備	289
Swift アプリケーションでの SmartStore の使用	292
Mobile Sync を使用した Salesforce オブジェクトへのアクセス	297
ネイティブアプリケーションでの Mobile Sync の使用	297
ハイブリッドおよび React Native アプリケーションでの Mobile Sync の使用	381
設定ファイルの検証	434
第 12 章: ファイルとネットワーキング	436
アーキテクチャ	437
ファイルのダウンロードと共有の管理	437
ファイルのアップロード	438
暗号化とキャッシュ	438
Android アプリケーションでのファイルの使用	438
要求キューの管理	439
iOS ネイティブアプリケーションでのファイルの使用	440
要求の管理	441
ハイブリッドアプリケーションでのファイルの使用	442
第 13 章: プッシュ通知と Mobile SDK	443
プッシュ通知について	444
ハイブリッドアプリケーションでのプッシュ通知の使用	444
コードの変更 (ハイブリッド)	444
Android でのプッシュ通知の使用	445
FCM (Android) 用の接続アプリケーションの設定	446
コードの変更 (Android)	447
iOS でのプッシュ通知の使用	448
APNS (iOS) 用に接続アプリケーションを設定する	448
コードの変更 (iOS)	449

第 14 章: モバイルアプリケーションでの認証、セキュリティ、ID	453
OAuth の用語	454
OAuth2.0 認証フロー	454
OAuth 2.0 Web サーバーフロー	455
OAuth 2.0 ユーザーエージェントフロー	456
OAuth 2.0 更新トークンフロー	457
Mobile SDK のログインおよび認証フローの詳細	458
範囲パラメーターの値	458
ID URL の使用	462
Android アプリケーションでのカスタムログインサーバーの設定	468
iOS アプリケーションでのカスタムログインサーバーの設定	469
OAuth トークンの取り消し	471
Android ネイティブアプリケーションでの更新トークンの取り消し	471
接続アプリケーション	471
PIN セキュリティについて	472
生体認証	472
OAuth 2.0 サイトおよび Salesforce サイトを使用したポータル認証	473
Salesforce Mobile SDK アプリケーションでの MDM の使用	474
サンプルプロパティリスト設定	477
高度な認証の使用	478
iOS アプリケーションでの高度な認証の設定	481
Android アプリケーションでの高度な認証の設定	482
Google ログイン要件に対応するための Android シングルサインオンアプリケーション のアップグレード	484
外部サービスにアクセスするための OpenID トークンの使用	485
iOS での安全な鍵の保管	488
Android での安全な鍵の保管	490
第 15 章: ログイン画面のカスタマイズ	491
プログラムによる iOS ログイン画面のカスタマイズ	492
第 16 章: ID プロバイダーアプリケーション	497
ID プロバイダー: アーキテクチャ、フロー、および接続アプリケーションの要件	499
Android アーキテクチャおよびフロー	501
ID プロバイダーとしての Android アプリケーションの設定	501
ID プロバイダークライアントとしての Android アプリケーションの設定	503
ID プロバイダーとしての iOS アプリケーションの設定	504
ID プロバイダークライアントとしての iOS アプリケーションの設定	507
Mobile SDK を使用しないモバイル ID プロバイダーアプリケーションの実装	509
実装の詳細とオプション	513
第 17 章: 安全なデータストレージ用のキー - 値ストアの使用	516
第 18 章: ダークモードおよびダークテーマの設定	522

第 19 章: Mobile SDK アプリケーションでの Experience Cloud サイトの使用	524
Experience Cloud サイトと Mobile SDK アプリケーション	525
「API の有効化」プロファイルを設定する	525
権限セットを設定する	526
API アクセス権をユーザーに付与する	527
ログインエンドポイントを設定する	527
Experience Cloud サイトのブランド設定	528
Experience Cloud サイトでのログイン、セルフ登録、およびパスワード管理のカスタマイズ	529
ブランド名が入ったログインページの使用	530
Experience Cloud サイトでの外部認証の使用	530
Salesforce が証明書利用者である認証プロバイダー SSO	531
認証後にユーザーを Experience Cloud サイトに誘導	532
証明書利用者のデータ要求のカスタマイズ	533
Facebook 認証プロバイダーの設定	536
Salesforce 認証プロバイダーの設定	540
OpenID Connect を使用した認証プロバイダーの設定	543
例: Mobile SDK アプリケーションからアクセスできるように Experience Cloud サイトを設定する	546
プロファイルに権限を追加する	547
Experience Cloud サイトを作成する	547
Experience Cloud サイトに API ユーザープロファイルを追加する	548
新しい取引先責任者およびユーザーを作成する	548
新しい Experience Cloud サイトのログインをテストする	549
例: Facebook 認証できるように Experience Cloud サイトを設定する	550
Facebook アプリケーションを作成する	551
Salesforce 認証プロバイダーを定義する	551
Facebook アプリケーションを設定する	552
認証プロバイダーの Apex クラスをカスタマイズする	553
Experience Cloud サイトの設定	553
第 20 章: Mobile SDK でのマルチユーザーのサポート	555
マルチユーザーのサポートについて	556
マルチユーザーのサポートの実装	556
Android ネイティブ API	558
iOS ネイティブ API	564
ハイブリッド API	569
第 21 章: 開発者向け Mobile SDK ツール	570
アプリケーション内開発者サポート	571
第 22 章: ログインと分析	575
iOS のコンパイラーレベルのログイン	576
Android のログインフレームワーク	576

計測およびイベント収集	579
第 23 章: 以前のリリースからの移行	581
11.0 から 11.1 へのすべてのアプリケーションの移行	582
以前のリリースからの移行	584
10.2 から 11.0 へのすべてのアプリケーションの移行	584
10.1 から 10.2 へのすべてのアプリケーションの移行	586
10.0 から 10.1 へのすべてのアプリケーションの移行	588
9.2 から 10.0 へのすべてのアプリケーションの移行	590
9.1 から 9.2 へのすべてのアプリケーションの移行	592
9.0 から 9.1 へのすべてのアプリケーションの移行	594
第 24 章: リファレンス	597
サポート対象の Salesforce API	598
Batch Request	600
ブリーフケースプライミングレコード	601
ブリーフケースプライミングレコード応答	602
コレクションの作成	603
コレクションの削除	605
コレクションの取得	606
コレクションの更新	607
コレクションの更新/挿入	608
コレクション応答	609
Composite Request	611
Create	612
Delete	613
Describe	614
Describe Global	614
メタデータ	615
通知	616
Notification Update	617
通知	618
Notifications Status	619
Notifications Update	620
Object Layout	621
リソース	622
Retrieve	623
SOSL Search	623
Search Result Layout	624
Search Scope and Order	625
SObject Tree	626
SOQL Query	627
SOQL Query All	628
ユーザー情報	628

Update	629
Upsert	630
Versions	631
Files API リファレンス	632
iOS アーキテクチャ	632
サポートされる REST サービス	632
Android アーキテクチャ	635
Android パッケージおよびクラス	635
Android リソース	635
Mobile SDK 11.1 でサポートされるバージョンのツールおよびコンポーネント	635
第 25 章: iOS で現在非推奨のもの	638
第 26 章: Mobile SDK 11.0 で削除された iOS API	639
第 27 章: Android で現在非推奨のもの	641
第 28 章: Mobile SDK 11.0 で削除された Android API	642
第 29 章: 商標の帰属	644
第 30 章: iOS ハイブリッドアプリケーションからの UIWebView の削除	645

第1章

モバイル開発の概要

トピック:

- Salesforce モバイルアプリケーションについて
- Salesforce モバイルアプリケーションのカスタマイズかカスタムアプリケーションの作成か
- このガイドの内容
- フィードバックの送信

即興動画からモバイル地理位置情報、オンラインショッピングに至るまで、あらゆる場所で最新のモバイルデバイスを使用してコンテンツの作成や閲覧が行われています。企業の従業員もスマートデバイスを使用して、顧客と連絡を取ったり、同僚とやりとりをしたり、ソーシャルネットワークで世間の人々と関わったりしています。

エンタープライズ IT 部門では、モバイルでのやりとりの急増に伴い、ソフトウェアサービスでの迅速な対応が必要になっています。Salesforce は、このニーズに対応する Salesforce Platform を提供しています。このクラウドは最もポピュラーなモバイル OS をサポートし、電話、タブレット、ウェアラブルなどさまざまなフォーム要素に対応しています。そのテクノロジーによって、ユーザーはカスタムアプリケーションを構築し、あらゆるシステムからのデータを接続し、エンタープライズをあらゆる場所から管理することができます。

Salesforce Mobile SDK では、Salesforce Platform がサポートされています。Mobile SDK を使用すると、開発者が Salesforce API 経由でプラットフォームにアクセスするスタンドアロンのモバイルアプリケーションを作成できます。

Salesforce モバイルアプリケーションについて

Salesforce Platform では、2つの方法で、エンタープライズ対応のモバイルアプリケーションを構築およびリリースできます。

- Salesforce モバイルアプリケーションは、App Store および Google Play Store から入手可能で、Salesforce のシステム管理者と開発者がアプリケーションを従業員に配布する最も速い方法です。システム管理者向けのシンプルなポイント&クリックツールと、高度な開発者向けの Lightning Web 開発プラットフォームがあります。
- Salesforce Mobile SDK には、独自のユーザーエクスペリエンスを実現するカスタムモバイルアプリケーションを構築するための開発者ツールが用意されています。Mobile SDK では、スタンドアロンのカスタムアプリケーションを作成して、App Store や Google Play Store で配布できます。これらのアプリケーションは、従業員、顧客、またはパートナーを対象にできます。これらのアプリケーションの構築には、ネイティブテクノロジーと Web テクノロジーのいずれかを選択でき、Salesforce モバイルアプリケーションと同じレベルの信頼性とセキュリティを実現できます。

Mobile SDK には次の機能があります。

エンタープライズ ID およびセキュリティ

Mobile SDK で Salesforce のログインフローや認証フローが自動的に実装されます。また、Salesforce 接続アプリケーションポリシーの完全な実装が含まれているため、すべてのユーザーが安全で簡単にデータにアクセスできます。SAML と高度な認証フローがサポートされているため、システム管理者は常にデータアクセスを完全に制御できます。

SmartStore 暗号化データベース

モバイルデータベースは、応答性が高く、どのようなネットワーク状態でも動作するアプリケーションを構築するのに便利です。SmartStore を使用すれば、柔軟なデータモデルに対応し、データをローカルで簡単に保存したり取得したりすることができます。また、AES-256 暗号化が採用され、データが保護された状態で維持されます。

Mobile Sync

Mobile Sync は、オフラインデータベースと Salesforce クラウド間でデータを同期するシンプルな API を提供します。Mobile Sync を使用すると、開発者は、複雑な同期ロジックは Mobile SDK に任せて、アプリケーションの UI とビジネスロジックに集中できます。

モバイルサービス

Mobile SDK は、クラウドでのプッシュ通知、地理位置情報、分析、コラボレーションツール、ビジネスロジックなど、幅広いプラットフォームモバイルサービスをサポートしています。これらのサービスにより、モバイルアプリケーションが大幅に強化されるとともに、開発時間が短縮されます。

Salesforce コミュニティ

Salesforce コミュニティと Mobile SDK を併用すれば、開発者が顧客やパートナー向けのモバイルアプリケーションを構築できます。これらのアプリケーションでも、従業員アプリケーションと同様のエンタープライズ機能と信頼性を利用できます。

iOS および Android

Mobile SDK では、主要な 2 つのモバイルオペレーティングシステムでのネイティブ開発がサポートされています。

ハイブリッドおよび React Native

JavaScript 開発者向けに、Mobile SDK には Cordova ベースのハイブリッドアプリケーションと React Native アプリケーションが搭載されています。これらのアプリケーションは、ネイティブコンテナ経由で Mobile SDK ライブラリにアクセスします。

Salesforce モバイルアプリケーションのカスタマイズかカスタムアプリケーションの作成か

Salesforce 組織ユーザー用のモバイルアプリケーションを開発する場合、2つの選択肢があります。Salesforce モバイルアプリケーションは、Salesforce が開発、構築、配布するカスタマイズ可能なモバイルアプリケーションです。カスタムアプリケーションは、Salesforce Mobile SDK に基づいて最初から構築されたスタンドアロンの iOS または Android アプリケーションです。このガイドでは Mobile SDK 開発のみを取り上げていますが、Salesforce モバイルアプリケーションとカスタムアプリケーションにはいくつかの相違点があります。

Salesforce モバイルアプリケーションのカスタマイズ

- 定義済みのカスタマイズ可能なユーザーインターフェースがある。
- Salesforce 組織にフルアクセスできる。
- Salesforce Platform で開発された機能を使用して統合環境を作成できる。
- アクションバーにより、独自のアプリケーションや機能を含める方法が提供される。
- ポイント & クリックまたはプログラムによるカスタマイズのいずれかを使用して Salesforce モバイルアプリケーションをカスタマイズできる。
- Lightning Web コンポーネントや Visualforce ページを使用して、プログラムで機能を追加できる。
- Salesforce モバイルのカスタマイズまたはアプリケーションは、Salesforce モバイルナビゲーションに準拠する。このため、たとえば Visualforce ページなどをナビゲーションメニューまたはアクションバーからコールできます。
- 既存の Salesforce 開発手法 (ポイント & クリックおよびプログラムの両方) を使用できる。
- すべての Salesforce エディションに含まれ、Salesforce でサポートされている。

カスタムモバイルアプリケーションの開発

カスタムアプリケーションは、Salesforce Mobile SDK で作成する独立型アプリケーションの場合と、プレーン HTML5 および JavaScript (Ajax) を使用するブラウザーアプリケーションの場合があります。カスタムアプリケーションでは、次のことができます。

- カスタムユーザー環境を定義する。
- ネイティブおよびハイブリッドのローカルアプリケーションで REST API を使用するか、JavaScript Remoting を使用するハイブリッドアプリケーションで Visualforce を使用して、Salesforce 組織のデータにアクセスする。HTML アプリケーションでは、jQueryMobile および Ajax を使用して同じ操作を行います。
- 顧客向け公開用のユーザーインターフェースをブランド設定する。

- ネイティブの iOS API または Android API、JavaScript と HTML を併用するハイブリッドコンテナ、React Native (Mobile SDK のみ) のいずれかを使用して、スタンドアロンのモバイルアプリケーションを作成する。
- App Store や Google Play などのモバイル業界チャネルを使用して、アプリケーションを配布する (Mobile SDK のみ)。
- 複雑なオフライン動作を設定および制御する (Mobile SDK のみ)。
- プッシュ通知を使用する。
- 独自の OAuth モジュールを使用して、カスタムセキュリティコンテナを設計する (Mobile SDK のみ)。
- Mobile SDK のその他の重要な考慮事項:
 - オープンソース SDK は、npm インストーラーおよび GitHub から無料でダウンロードできます。
 - アプリケーションの開発とコンパイルは、ブラウザーウィンドウではなく、外部開発環境 (iOS の場合は Xcode、Android の場合は Android Studio) で行います。
 - 開発コストは、アプリケーションやプラットフォームによって決まります。

Mobile SDK は、以下を提供することで Salesforce プラットフォームクラウドアーキテクチャをモバイルアプリケーションに統合します。

- Salesforce 接続アプリケーションポリシーの実装。
- 永続化機能や更新機能を含む、Salesforce 組織のログインおよび OAuth ログイン情報管理。
- SmartStore を使用した安全なオフラインストレージ。
- Mobile Sync による Salesforce Cloud と SmartStore の同期。
- Salesforce コミュニティのサポート。
- 暗黙的なネットワークを利用した Salesforce REST API のラッパー。
- 複数ユーザー間の高速な切り替え。
- ハイブリッドアプリケーション用の Cordova ベースのコンテナ。
- Mobile SDK のネイティブ API に対する React Native ブリッジ。

このガイドの内容

このガイドでは、Salesforce Mobile SDK を紹介し、クラウド用モバイルアプリケーションの設計、開発、管理方法について説明します。各トピックでは、Mobile SDK でサポートされるすべてのプラットフォームについて言及しています。具体的には、次のプラットフォームです。

- ネイティブ iOS
- ネイティブ Android
- React Native
- Cordova ベースのハイブリッド
- HTML と JavaScript

対象利用者

このガイドは、モバイル技術、OAuth2、およびREST APIについて理解している開発者を主に対象としています。特に、Salesforce Platform の一定の実務経験がある開発者に役立ちます。ただし、これに該当しない場合でも心配いりません。上級開発者だけではなく、初心者の方向けのリソースも用意されています。たとえば、初めてモバイルアプリケーションを開発するSalesforce システム管理者や、ベテランのモバイル開発者でSalesforce テクノロジーが初めての方でも理解できると思われる内容になっています。

初心者の方は、Mobile SDK Trailhead モジュールから始めることをお勧めします。その後、このガイドを必要に応じて参照し、知識を広げていってください。Mobile SDK トレイルを次に示します。

Mobile SDK を使用した開発

サポートされるすべてのプラットフォーム、具体的には iOS、Android、hybrid、React Native を紹介し、Mobile SDK のオフライン機能について概説します。

iOS を使用した最新のモバイル開発

Xcode と Swift の基礎、さらにはより上級の Mobile SDK チュートリアルを含めて「iOS 初級」トレイルを強化しています。

より簡潔でカジュアルなチュートリアルもこのガイドのあちこちにあります。

 **メモ:** 『Salesforce Mobile SDK 開発ガイド』には、PDF 形式のものとオンライン形式のものがあります。いずれのバージョンも developer.salesforce.com/docs で利用できます。

バージョン

本書は Salesforce Mobile SDK 11.1 です。

フィードバックの送信

本書に関するご質問またはコメントがある場合や、今後のバージョンで追加するトピックのご提案がある場合、次の方法を利用できます。

- [Mobile SDK Trailblazer Community](#) に投稿する
- Salesforce 開発者ディスカッションフォーラム (developer.salesforce.com/forums) で意見を投稿する
- オンラインドキュメント (developer.salesforce.com/docs/atlas.en-us.mobile_sdk.meta/mobile_sdk/) の各ページの下部にある [フィードバック] ボタンを使用する

Mobile SDK サポートをお探しですか? 次の操作を実行できます。

- [Mobile SDK Trailblazer Community](#) に投稿する
- [Mobile SDK GitHub リポジトリ](#) のいずれかで GitHub の問題を作成する
- < mobile sdk > タグを使用して [Salesforce StackExchange](#) サイトに投稿する
- Salesforce の [開発者フォーラム](#) に投稿する

 **メモ:** Mobile SDK はオープンソースの製品で、Salesforce サポートではサポートされていません。

第 2 章

Salesforce Mobile SDK 開発の概要

トピック:

- ネイティブ、HTML、およびハイブリッド開発について
- クイックスタート

Salesforce Mobile SDK を使用すると、スタンドアロンモバイルアプリケーション内で Salesforce Platform の機能を活用できます。

Salesforce Platform には、Salesforce クラウドコンピューティング用のわかりやすく生産的なプラットフォームが備えられています。開発者は Salesforce Platform を使用して、カスタムオブジェクトとカスタム項目、ワークフロールール、Visualforce ページ、Apex クラス、トリガーなどの Salesforce アプリケーションコンポーネントを定義できます。定義したコンポーネントは、画期的なブラウザーベースのデスクトップアプリケーションに組み込むことができます。

Mobile SDK アプリケーションでは、デスクトップアプリケーションと異なり、ブラウザーではなくモバイルデバイスのネイティブオペレーティングシステム経由で Salesforce データにアクセスします。モバイルユーザーが満足する生産性の高い操作性を確保するために、オンライン/オフライン状態をシームレスに切り替えられるよう Mobile SDK アプリケーションを設定できます。Mobile SDK に着手する前に、モバイル開発のしくみを確認し、重要な Salesforce 開発者リソースについて学習します。

ネイティブ、HTML、およびハイブリッド開発について

Salesforce Mobile SDK には、アプリケーションの開発オプションがいくつか用意されています。ユーザーの開発スキル、デバイスおよびテクノロジーの要件、目的、およびスケジュールに合ったオプションを選択できます。

Mobile SDK を使用すれば、ビジネスニーズとリソースに最適な開発プラットフォームを選択できます。次のようなプラットフォームがあります。

- ネイティブ
- HTML Living Standard
- ハイブリッド
- React Native

各プラットフォームオプションは、iOS と Android の両方に対応しています。次の説明を読んで、どのオプションが自分の要件に最適であるかを把握してください。

ネイティブアプリケーション

ネイティブアプリケーションは特定のモバイルプラットフォーム (iOS または Android) に固有のものです。これらのアプリケーションは、それぞれのプラットフォームでサポートされる開発ツールと言語を使用します。たとえば、iOS アプリケーションは Xcode で Swift または Objective-C を使用して開発します。すべてのアプリケーション種別で、ネイティブアプリケーションは外観とパフォーマンスに最も優れています。ネイティブアプリケーションは、使い勝手、機能、全体的なモバイルの操作性の点において最良です。たとえば、次の機能を提供する場合、ネイティブが最適な選択肢です。

- 高速グラフィック API — ネイティブプラットフォームでは最も高速なグラフィックを実現できます。これは、たとえば、大量のデータを使用しており、すばやい更新が必要な場合にメリットがあります。
- 滑らかなアニメーション — 高速グラフィック API と関連し、滑らかなアニメーションを実現する機能です。この機能は、特にモバイルアプリケーションの UI グラフィックや動画ストリーミングに重要です。
- 組み込みコンポーネント — カメラ、連絡先、地理位置情報、およびデバイスに対してネイティブなその他の機能を、モバイルアプリケーションにシームレスに統合できます。他の重要な組み込みコンポーネントとして、暗号化されたストレージもあります。
- 使いやすさ — ユーザーはネイティブプラットフォームを使い慣れています。この使い慣れた操作感をユーザーが期待するネイティブ機能に追加することで、アプリケーションがさらに使いやすくなります。

ネイティブアプリケーションの開発には、統合開発環境 (IDE) が必要です。IDE には、コードの作成とデバッグ、プロジェクト管理、バージョン管理、およびその他の必要な機能に対応するプラットフォーム固有の豊富なツールが備えられています。他の開発シナリオに比べてより高い経験レベルが要求されますが、制御や自由度のレベルも高くなります。ネイティブ開発のその他の利点は次のとおりです。

- 実証済みの API やフレームワークにより、作業に見合った価値を得られるという確信を持てる
- 定義済みコンポーネントにより、ナビゲーション、タブ、モーダルダイアログなどの標準動作が定義される
- ソース制御により、すべてのコードを安全に一元管理できる

HTML アプリケーション

HTML アプリケーション (HTML5 アプリケーションともいう) では、標準の Web テクノロジー (通常、HTML Living Standard、JavaScript および CSS) を使用し、Web ページをモバイルブラウザに提供します。ユビキタス HTML 標準およびツールは、ネイティブコンパイラーやハイブリッドフレームワークを導入する場合に比べて簡単に開始できます。

HTML 開発では、デバイスにとらわれず、最新のどのモバイルブラウザでも実行できるマルチプラットフォームモバイルアプリケーションが作成されます。反応型 Web 設計ではコンテンツが自動的にさまざまな画面サイズおよび解像度に適応されるため、テスト要件が少なく済みます。HTML の「Write Once, Run Anywhere (一度書けば、どこでも実行できる)」というアプローチにより、配布の労力が最小限に抑えられます。バグ修正や新機能を本番サーバーにプッシュすれば、即座に全ユーザーが使用できるようになります。また、コンテンツが Web に存在するため検索可能です。これは多くのタイプのアプリケーションにとって大きな利点となります。

HTML アプリケーションは、開発とサポートが簡単で、最も広範囲にわたるデバイスを使用できますが、次のような欠点もあります。

- 安全なオフラインストレージの欠如 — HTML ブラウザーでは、オフラインのデータベースとキャッシュをサポートしていますが、標準の暗号化はサポートされません。Mobile SDK には次のすべての機能が備わっています。
- 使いにくいセキュリティフロー — 単純なセキュリティ対策を実装する場合でも、モバイル Web アプリケーションでは複雑な課題となることがあります。また、ユーザーにとっても不便な場合があります。たとえば、認証を必要とする Web アプリケーションでは、アプリケーションを再起動するたび、またはバックグラウンド状態から戻るたびに、ユーザーはログイン情報を再入力する必要があります。
- ネイティブ機能の制限 — カメラ、連絡先、その他のネイティブ機能は、特に旧バージョンのモバイルブラウザでは異なっている場合があります。
- ネイティブのデザインの欠如 — HTML はネイティブの外観しかエミュレートできず、顧客は使い慣れた複合操作を使用できません。

ハイブリッドアプリケーション

ハイブリッドアプリケーションは、HTML コーディングの容易さと、ネイティブプラットフォームの機能とを組み合わせたものです。結果として作成されるアプリケーションはデバイスのネイティブ機能にアクセスでき、App Store から配信されます。Lightning Web コンポーネントまたは Visualforce ページを提供するハイブリッドアプリケーションも作成できます。

ハイブリッドアプリケーションでは、ネイティブプラットフォーム機能へのアクセスを提供する軽量のネイティブコンテナ内で HTML、JavaScript、および CSS コードをラップします。一般に、ハイブリッドアプリケーションは HTML とネイティブの両方の世界の最良の機能を提供します。ハイブリッドアプリケーションは、HTML アプリケーションと同じくらいに開発が容易であり、JavaScript API を通じてネイティブ機能にアクセスできます。また、ハイブリッドアプリケーションでは、Mobile SDK のオフライン機能を使用して、次の操作を行うことができます。

- Salesforce データをモデル化、クエリ、検索、編集する。
- オフラインでできるように Salesforce データを安全にキャッシュする。
- ローカルにキャッシュされたデータを Salesforce サーバーと同期する。

HTML アプリケーションと異なり、ハイブリッドアプリケーションはモバイルデバイスにダウンロードしてインストールします。ハイブリッドアプリケーションのファイルがデバイスとサーバーのどちらに保存されるのか疑問に思われるかも知れません。よい質問です。保存場所は、ハイブリッドアプリケーションがローカルまたはリモートのどちらであるかによって異なります。

ローカル

HTML と JavaScript のコードは、ネイティブアプリケーションと同様の構造で、モバイルアプリケーションバイナリ内でパッケージ化できます。REST API と Ajax を使用して、デバイスとクラウド間でデータを移動します。

リモート

完全な Web アプリケーションをサーバーから実装することもできます (パフォーマンス向上のため任意でキャッシュもできます)。コンテナアプリケーションは、完全なアプリケーションをサーバーから取得し、それをネイティブの Web ビューに表示します。

さらに、ハイブリッドリモートアプリケーションでは Lightning Web コンポーネントをホストできます。ハイブリッドは、現在のところ次の一般的な Salesforce エンティティをサポートしている唯一の Mobile SDK プラットフォームです。

React Native アプリケーション

React Native アプリケーションは、Facebook, Inc. の React Native フレームワークを使用して、JavaScript アプリケーションをネイティブコードとして実行します。ハイブリッドのパラダイムに従うのではなく、React Native では、ネイティブの UI ビルディングブロックを JavaScript コードと組み合わせることができます。このフレームワークでは、ネイティブリソースへの直接アクセスが提供されるため、再コンパイルせずにテストできます。パフォーマンスの点で React Native は純粋なネイティブ実行とほとんど差がありません。

React Native アプリケーションは、特殊なスクリプト言語を使用します。

- JavaScript は ES2015 です。
- CSS は、CSS 構文によく似た JavaScript コードで、通常は JavaScript アプリケーションにインラインで記述されます。
- マークアップは、実際には JSX と呼ばれる特別なフレーバーの XML です。マークアップに JavaScript を埋め込む HTML とは異なり、JavaScript に JSX マークアップを埋め込みます。
- TypeScript のサポートにより、JavaScript コードで非バインド静的型付けを使用できます。

さらに、React Native ブリッジでは、次のような標準の Mobile SDK 機能にアクセスできます。

- SmartStore
- Mobile Sync
- Salesforce のログインと認証
- Salesforce REST API アクセス。

(Swift、Objective-C、または Java 内の) 独自のネイティブオブジェクトに React Native コードで直接アクセスすることもできます。

モバイルアーキテクチャの比較

次の表に、各開発シナリオの比較を示します。

	ネイティブ、React Native	HTML	ハイブリッド
グラフィック	ネイティブ API	HTML、キャンバス、SVG	HTML、SVG
パフォーマンス	最速	高速	中速
デザイン	ネイティブ	エミュレート	エミュレート
配布	App Store、Google Play	Web	App Store、Google Play
カメラ	はい	ブラウザーに依存	はい
通知 (Salesforce から)	はい	いいえ	はい
連絡先、カレンダー	はい	ブラウザーに依存	はい
オフラインストレージ	安全なファイルシステム	安全ではない、共有 SQL、キー-値ストア	安全なファイルシステム、共有 SQL (Cordova プラグインを使用)
地理位置情報	はい	はい	はい
スワイプ	はい	はい	はい
ピンチ、スプレッド	はい	はい	はい
接続	オンライン、オフライン	ほぼオンライン	オンライン、オフライン
開発手段	Swift、Objective-C、Java、Kotlin、JavaScript/TypeScript、マークアップ (React Native のみ)	HTML、CSS、JavaScript	HTML、CSS、JavaScript

クイックスタート

詳細を後で確認したいという方のために、このガイドではネイティブ開発シナリオごとに「クイックスタート」トピックを用意しています。

- [ハイブリッドアプリケーションのクイックスタート](#)
- [iOS ネイティブクイックスタート](#)
- [Android ネイティブクイックスタート](#)

第 3 章

Mobile SDK 11.1 の新機能

トピック:

- [最近のリリースの新機能](#)

Mobile SDK 11.1.0 は、iOS および Android の最新のサポートを含むマイナーリリースです。

多くの場合、中間リリースでは、予定されているメジャーリリースで削除されるネイティブライブラリの項目が非推奨になります。コンパイラログに非推奨の警告がないか必ず確認し、なんらかの変更が実施される前に、変更に対処できるようにしてください。

アプリケーションのアップグレード方法

Mobile SDK アプリケーションのアップグレードについての詳細は、「[以前のリリースからの移行](#)」の手順に従います。

Mobile SDK 11.1 の全般的な更新

これらの変更は、複数のプラットフォームに適用されます。

外部コンポーネントのバージョン更新

Cordova for Android: 12.0.1

Cordova for iOS: 7.0.1

React Native: 0.70.14

Mobile SDK 11.1 for iOS の新機能

iOS 17 の互換性

Mobile SDK の iOS 17 および XCode 15 との互換性については正常にテスト済みです。

「[iOS 17 Release Notes \(iOS 17 リリースノート\)](#)」を参照してください。

Swift Package Manager のサポート

Swift Package Manager のサポートが導入されました。これにより、Mobile SDK をアプリケーションに組み込むことができるようになりました。

- Mobile SDK のバイナリフレームワークは、新しいリポジトリ <https://github.com/forcedotcom/SalesforceMobileSDK-iOS-SPM> でホストされます。
- Swift Package Manager を介して連動関係を取得する `iOSNativeSwiftPackageManager` テンプレートが追加されました。
- 「[Mobile SDK ライブラリのプロジェクトへの追加](#)」、 「[iOS Swift プロジェクトの手動作成](#)」、 「[forceios を使用した iOS プロジェクトの作成](#)」 も参照してください。

非推奨の API

コンパイラーの警告を確認するか、「[iOS で現在非推奨のもの](#)」を参照してください。

Mobile SDK 11.1 for Android の新機能

Android 14 の互換性

Mobile SDK の Android 14 との互換性については正常にテスト済みです。「[Android Version 14 \(Android バージョン 13\)](#)」を参照してください。

Mobile Sync ライブラリの最新化

Android の Mobile Sync ライブラリが最新化されました。

- すべてのソースファイルは Kotlin で記述されるようになりました。
- null が想定されていないかサポートされていない場合、パラメーターとメンバーで null 非許容型が使用されるようになりました。
- Kotlin 構文が必要に応じてサポートされるようになりました。たとえば、文字列テンプレート、?:、let、also、map、forEach、when など。
- コルーチンラッパーが SyncManager の主要メソッドで使用できるようになりました。「[reSync を使用した増分同期](#)」、「[下位同期操作後の「ゴースト」レコードの処理](#)」、「[同期名の使用](#)」を参照してください。

 **メモ:** 通常、Mobile SDK 11.1 などのマイナーリリースではコード変更を使用する必要はありませんが、最新化された Mobile Sync ライブラリでは、場合によってはコード変更を使用する必要があります。たとえば、以前は Java のクラスからインポートされていた定数は、Kotlin コードの使用ではコンパニオンオブジェクトからインポートされるようになりました。

Android テンプレートの更新

- Mobile SDK Android テンプレートは、Kotlin DSL 移行により最新の状態になりました。
- テンプレートは、Maven Central から Mobile SDK アイテムをダウンロードするように設定されるようになりました。その結果、より使いやすいビルド環境になっています。

高度な認証の機能強化

高度な認証に関連する次のバグが修正されました。

- 高度な認証フロー中に Chrome が見つからなかった場合、Android ユーザーにはエラーが表示され、ログインを続行できませんでした。実行時に Chrome を使用できない場合、高度な認証はデフォルトのブラウザを使用する想定された動作に戻るようになりました。
- バグにより、MFA 中にアプリケーションがバックグラウンドになったときに、一部の Android ユーザーのログインフローが初期画面にリセットされていました。この問題が修正され、LoginActivity の起動モードが singleInstance から

`singleTop` に変更されました。LoginActivity を拡張するアプリケーションでも同じ変更が必要になりました。

新しい高度な認証方法が追加されました。これにより、1) アプリケーションが選択するブラウザーを設定し、2) 現在選択されているカスタムタブブラウザーを表示できます。「[Android アプリケーションでの高度な認証の設定](#)」を参照してください。

非推奨の API

コンパイラーの警告を確認するか、「[Android で現在非推奨のもの](#)」を参照してください。

最近のリリースの新機能

次に、最近の Mobile SDK リリースの新機能のお知らせのアーカイブを示します。

⚠ 重要: 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

Mobile SDK 11.0.1

Mobile SDK 11.0.1 はマイナーパッチリリースです。次の変更が含まれています。

- カスタムドメインと拡張ドメインでのログインと更新に関するバグが修正されました。
- ハイブリッドアプリケーションおよび IDP フロー使用時のアクセストークンのリハイドレーション。
- キー - 値ストアの読み取りパフォーマンスが向上しました。

Mobile SDK 11.0

Mobile SDK 11.0 は、いくつかの認証フローを最新ののものにするメジャーリリースです。通常、メジャーリリースでは、今後削除されるために非推奨になっていた項目が削除されます。以下の情報を読んで、新機能や、アプリケーションに影響を与える可能性のある最新の変更を確認してください。すべてのリリースで、コンパイラログに非推奨の警告がないか必ず確認し、これらの変更が実施される前に、変更に対処できるようにしてください。

全般的な更新

これらの変更は、複数のプラットフォームに適用されます。

- iOS と Android のデフォルトの認証でユーザーエージェントフローの代わりに Web サーバーフローが使用されるようになりました。[「OAuth 2.0 Web サーバーフロー」](#) (ページ 455)を参照してください。
- ログインでのデバイスシステムの生体認証。[「生体認証」](#) (ページ 472)を参照してください。
- ID プロバイダーを使用する改良されたマルチアプリケーション SSO フローおよび設定。[「ID プロバイダーアプリケーション」](#) (ページ 497)を参照してください。

外部コンポーネントのバージョン更新

SQLCipher: 4.5.4 (iOS および Android)

SQLite: 3.41.2

Cordova for Android: 11.0.0

Cordova for iOS: 6.3.0

Cordova コマンドライン: 12.0.0

iOS

[「Mobile SDK 11.0 の全般的な更新」](#) も参照してください。

バージョンの更新

リリース対象: 15

ベース SDK バージョン 16

Xcode: 14

削除された API

「[Mobile SDK 11.0 で削除された iOS API](#)」 (ページ 639)を参照してください。

非推奨の API

コンパイラーの警告を確認するか、「[iOS で現在非推奨のもの](#)」を参照してください。

Mobile SDK 10.2

Mobile SDK 10.2 は、非破壊的な API の変更と最新プラットフォームのサポートを含む中間リリースです。

iOS

iOS 16 の互換性

Mobile SDK の iOS 16 との互換性については正常にテスト済みです。「[iOS 16 Release Notes \(iOS 16 リリースノート\)](#)」を参照してください。

バージョンの更新

SQLite: 3.39.2

SQLCipher: 4.5.2

React Native: 0.70.1

ShellJS: 0.8.5 (コマンドラインツール用)

TypeScript: 4.8.3 (React Native 用)

非推奨の API

コンパイラーの警告を確認するか、「[iOS で現在非推奨のもの](#)」を参照してください。

Android

Android 13 の互換性

Mobile SDK の Android 13 との互換性については正常にテスト済みです。「[Android Version 13 \(Android バージョン 13\)](#)」を参照してください。

バージョンの更新

SQLite: 3.39.2

SQLCipher: 4.5.2

OkHttp: 4.10.0

Cordova-android: 11.0.0

React Native: 0.70.1

ShellJS: 0.8.5 (コマンドラインツール用)

TypeScript: 4.8.3 (React Native 用)

Android SDK (ターゲットの API): 33

非推奨の API

コンパイラーの警告を確認するか、「[Android で現在非推奨のもの](#)」を参照してください。

React Native

バージョンの更新

React Native: 0.70.1

Mobile SDK 10.1.1 での変更

Mobile SDK 10.1.1 はマイナーパッチリリースです。Mobile SDK 10.1.1 では、モバイルアプリケーション用の組織の接続アプリケーションの設定から、[次の時間が経過したらアプリケーションをロック]のタイムアウト設定の使用方法が復元されます。これが設定されている場合、モバイルアプリケーションがタイムアウト期間より長くバックグラウンドで実行されているとロックされます。モバイルアプリケーションを有効にするとロックがかかります。アプリケーションのロック解除は従来どおりです。

Mobile SDK 10.1

Mobile SDK 10.1.0 は、バグ修正、パフォーマンスの改善、機能の追加、更新を含むマイナーリリースです。

多くの場合、中間リリースでは、予定されているメジャーリリースで削除されるネイティブライブラリの項目が非推奨になります。コンパイラログに非推奨の警告がないか必ず確認し、なんらかの変更が実施される前に、変更に対処できるようにしてください。

全般的な更新

これらの変更は、複数のプラットフォームに適用されます。

ブリーフケースプライミングレコード用の REST API メソッド (iOS、Android)

ブリーフケースプライミング Salesforce API 用の REST 要求ファクトリーメソッドおよび応答パーサー。 [「ブリーフケースプライミングレコード」](#) を参照してください。

sObject コレクション用の REST API メソッド (iOS、Android、React Native)

次の sObject コレクション操作用の REST 要求ファクトリーメソッドおよび応答パーサー。

- Create — [「コレクションの作成」](#) を参照してください。
- Retrieve — [「コレクションの取得」](#) を参照してください。
- Update — [「コレクションの更新」](#) を参照してください。
- Upsert — [「コレクションの更新/挿入」](#) を参照してください。
- Delete — [「コレクションの削除」](#) を参照してください。

ブリーフケース下位同期ターゲット (iOS、Android)

組織のブリーフケースからレコードをダウンロードしてローカルで同期するための新しい下位同期ターゲット。 [「ブリーフケース下位同期ターゲットの使用」](#) を参照してください。

sObject コレクションを使用したコレクション上位同期ターゲット (iOS、Android)

パフォーマンスを向上するために sObject コレクションを使用する新しい上位同期ターゲット。上位同期ターゲット設定で実装クラス ("androidImpl" または "iOSImpl") を指定していない場合、Mobile SDK では自動的に `CollectionSyncUpTarget` が使用されます。 [「sObject コレクション上位同期ターゲットの使用」](#) を参照してください。

外部コンポーネントのバージョン更新

SQLCipher: 4.5.1 (iOS および Android)

SQLite: 3.37.2

Gradle: 7.2.1

iOS

「*Mobile SDK 10.0 の全般的な更新*」を参照してください。

バージョンの更新

SQLCipher: 4.5.1

SQLite: 3.37.2

非推奨の API

コンパイラーの警告を確認するか、「[iOS で現在非推奨のもの](#)」を参照してください。

Android

「*Mobile SDK 10.0 の全般的な更新*」を参照してください。

MobileSyncExplorerKotlin テンプレート

Kotlin と Jetpack Compose を使用して Mobile Sync のすべての能力を示す新しい Android アプリケーションテンプレート: <https://github.com/forcedotcom/SalesforceMobileSDK-Templates/tree/dev/MobileSyncExplorerKotlinTemplate>

バージョンの更新

SQLCipher: 4.5.4

SQLite: 3.41.2

Gradle: 7.2.1

非推奨の API

コンパイラーの警告を確認するか、「[Android で現在非推奨のもの](#)」を参照してください。

React Native

「*Mobile SDK 10.0 の全般的な更新*」を参照してください。

バージョンの更新

React Native: 0.70.14

SmartStore

バージョンの更新

SQLCipher: 4.5.4 (iOS および Android)

SQLite: 3.41.2

Mobile Sync

「*Mobile SDK 10.0 の全般的な更新*」を参照してください。

ブリーフケース下位同期ターゲット (iOS、Android)

組織のブリーフケースからレコードをダウンロードしてローカルで同期するための新しい下位同期ターゲット。

sObject コレクションを使用したコレクション上位同期ターゲット (iOS、Android)

パフォーマンスを向上するために sObject コレクションを使用する新しい上位同期ターゲット。上位同期ターゲット設定で実装クラスを指定していない場合、MobileSDK では自動的に `CollectionSyncUpTarget` が使用されます。

Mobile SDK 10.0

Mobile SDK 10.0.0 はメジャーリリースです。このリリースでは、破壊的な API の変更、バグ修正、パフォーマンスの強化、細かな機能の追加、アップデートなどが実施されます。

メジャーリリースでは、中間リリースで非推奨になったネイティブライブラリの項目が削除されます。利便性の目的で、非推奨のネイティブ API のリストがコンパイルされています。

全般的な更新

これらの変更は、複数のプラットフォームに適用されます。

キー - 値ストアのバイナリストレージ

ネイティブ iOS および Android のキー - 値ストアで、安全なバイナリストレージ API がサポートされるようになりました。「[安全なデータストレージ用のキー - 値ストアの使用](#)」を参照してください。

外部コンポーネントのバージョン更新

React Native: 0.67.1

Cordova for iOS: 6.2.0

Cordova for Android: 10.1.1

Cordova コマンドライン: 11.0.0

SQLCipher: 4.5.0 (iOS および Android)

SQLite: 3.36.0

node.js:12.0 から最新の LTS バージョンに更新

iOS

キー - 値ストアのバイナリストレージ

キー - 値ストアで、新しい Mobile SDK API を使用する安全なバイナリストレージがサポートされるようになりました。「[安全なデータストレージ用のキー - 値ストアの使用](#)」を参照してください。

MobileSyncExplorerSwift テンプレートアプリケーションのウィジェット

このテンプレートに最近の取引先責任者ウィジェットが追加されました。

バージョンの更新

リリース対象: 14

ベース SDK バージョン 15

Xcode: 13

削除された API

「[Mobile SDK 11.0 で削除された iOS API](#)」を参照してください。

非推奨の API

コンパイラの警告を確認するか、「[iOS で現在非推奨のもの](#)」を参照してください。

Android

キー - 値ストアのバイナリストレージ

キー - 値ストアで、既存の Mobile SDK API を使用する安全なバイナリストレージがサポートされるようになりました。「[安全なデータストレージ用のキー - 値ストアの使用](#)」を参照してください。

バージョンの更新

最小限の API: Android Nougat (API 24)

対象の API: Android 12 (API 32)

ハイブリッドアプリケーションのデフォルトの SDK バージョン: Android 12 (API 32)

削除された API

「[Mobile SDK 11.0 で削除された Android API](#)」を参照してください。

非推奨の API

コンパイラの警告を確認するか、「[Android で現在非推奨のもの](#)」を参照してください。

React Native

バージョンの更新

React Native: 0.67.1

ハイブリッド

バージョンの更新

Cordova for iOS: 6.2.0

Cordova for Android: 10.1.1

Cordova コマンドライン: 11.0.0

SmartStore

WAL for Android

Mobile SDK 10.0 は、SQLCipher for Android にログ先行書き込み (WAL) を実装します。Android では SQLCipher の同時読み取り/書き込みサポートが引き続きブロックされますが、下位レベルの更新により SmartStore のパフォーマンスが大幅に向上します。

機能の非推奨

サードパーティモジュールの向上に伴い、外部ストレージ機能と `SoupSpec` クラスが Mobile SDK 11.0 で削除されるため、非推奨になりました。SmartStore は現在、大量のデータセットを完全に処理できるようになっています。「[Android で現在非推奨のもの](#)」および「[iOS で現在非推奨のもの](#)」を参照してください。

バージョンの更新

SQLCipher: 4.5.0 (iOS および Android)

SQLite: 3.36.0

Mobile SDK 9.2.0

Mobile SDK 9.2.0 は、非破壊的な API の変更と最新の iOS サポートを含む中間リリースです。

多くの場合、中間リリースでは、予定されているメジャーリリースで削除されるネイティブライブラリの項目が非推奨になります。コンパイラログに非推奨の警告がないか必ず確認し、なんらかの変更が実施される前に、変更に対処できるようにしてください。

これらの変更は、複数のプラットフォームに適用されます。

全般的な更新

パスコードの削除

モバイルオペレーティングシステムのセキュリティを優先するため、アプリケーション固有のパスコードが iOS と Android のアプリケーションから削除されました。Mobile SDK では引き続き組織のパスコード要件が優先されますが、パスコードの長さ、パスコードのタイムアウト、接続アプリケーションの生態

認証設定は無視されます。顧客がすでにデバイスのロック画面または生体認証ロック解除を設定している場合、このアップグレードはシームレスです。それ以外の場合、新しいロック画面で顧客に認証モードを設定するよう要求します。顧客がバックグラウンドからアプリケーションを再有効化する場合、アプリケーション固有のパスコードではなくデバイスのパスコードが必要です。[「ログインとパスコードについて」](#) (ページ 76) (iOS) および [「パスコードの使用」](#) (ページ 154) (Android) を参照してください。

外部コンポーネントのバージョン更新

Cordova

- iOS: 6.2.0
- Android: 10.1.0

React Native

0.66.0 (iOS および Android)

SQLite

3.34.1

SQLCipher

4.4.3 (iOS および Android)

Mobile SDK 9.1.0

Mobile SDK 9.1.0 は、非破壊的な API の変更と最新の iOS サポートを含む中間リリースです。

多くの場合、中間リリースでは、予定されているメジャーリリースで削除されるネイティブライブラリの項目が非推奨になります。コンパイラログに非推奨の警告がないか必ず確認し、なんらかの変更が実施される前に、変更に対処できるようにしてください。

全般的な更新

これらの変更は、複数のプラットフォームに適用されます。

キー - 値ストア (iOS、Android)

- キー - 値ストアのバージョン 2 は 9.1 で公開されます。バージョン 2 では、キー - 値ストア API を使用して、ストアからすべてのキーを取得できます。
- [Dev Support (開発サポート)] メニューの [Inspect Key-Value Store (キー - 値ストアを検査)] オプションでは、キー名の一部または全部に一致するすべてのキーを検索できます。
- これらの機能についての詳細は、[「安全なデータストレージ用のキー-値ストアの使用」](#) (ページ 516) を参照してください。

iOS

サンプルアプリケーションでの iPad サポート

- [MobileSyncExplorerSwift](#) テンプレートアプリケーションが iPad 用の Catalyst やマルチウィンドウをサポートするようになりました。
- [RestAPIExplorer](#) サンプルアプリケーションが Catalyst をサポートするようになりました。

REST API ラッパーの更新

- `SFRestApi` (Objective-C) および `RestClient` (Swift) の `requestForQuery` メソッドに `batchSize` パラメーターが追加されました。このパラメーターを使用すると、それぞれの取得で返されるレコード数について、希望の数を指定できます。指定できる値の範囲は、200 ~ 2,000 (デフォルト設定) で

す。実行時のパフォーマンスを調整できるようにするため、Mobile SDK では、要求されたバッチサイズが実際のバッチサイズとなることが保証されていません。

廃止

- コンパイラーの警告を確認するか、[「iOS で現在非推奨のもの」](#)を参照してください。

Android

REST API ラッパーの更新

- `RestRequest.getRequestForQuery` メソッドに `batchSize` パラメーターが追加されました。このパラメーターを使用すると、それぞれの取得で返されるレコード数について、希望の数を指定できます。指定できる値の範囲は、200～2,000 (デフォルト設定) です。実行時のパフォーマンスを調整できるようにするため、Mobile SDK では、要求されたバッチサイズが実際のバッチサイズとなることが保証されていません。

廃止

コンパイラーの警告を確認するか、[「Android で現在非推奨のもの」](#)を参照してください。

SmartStore

スマート SQL では、SELECT または WHERE 句で参照されるすべての項目のインデックスパスが不要になりました。[「スマート SQL クエリ」](#) (ページ 263)を参照してください。

Mobile Sync

SOQL 下位同期ターゲットの機能強化

SOQL 下位同期バッチのサイズを設定できるようになりました。200～2,000 (デフォルト値) までの任意の値を指定できます。[「SOQL 下位同期ターゲットの使用」](#) (ページ 346)を参照してください。

Mobile SDK 9.0

一般情報

これらの変更は、複数のプラットフォームに適用されます。

開発者ツール

(iOS、Android) [Dev Support (開発サポート)] メニューに新しいユーティリティ [Inspect Key-Value Store (キー - 値ストアを検査)] が追加されました。[アプリケーション内開発者サポート](#)

外部コンポーネントのバージョン更新

- SQLCipher (iOS、Android): 4.4.2
- SQLite (iOS、Android): 3.33.0
- yarn: 1.22
- Cordova:
 - iOS: 6.1.1
 - Android: 9.0.0

iOS

iPadOS のサポート

- iPadOS のマルチウィンドウサポートを実装しました。この新機能に対応する場合、iPad での実行を意図した既存アプリケーションを変更する必要があります。[「Mobile SDK アプリケーションでの iPadOS のサポート」](#) (ページ 117)を参照してください。
- iPadOS の横モードサポートを改善しました。(Mobile SDK 9.0 へ更新するだけで、アプリケーションの変更は必要ありません)。

バージョンの更新

- リリース対象: iOS 13
- ベース SDK: iOS 14
- Xcode: 12
- CocoaPods: 1.8.0 (最大の指定なし)

廃止

- コンパイラーの警告を確認するか、[「iOS で現在非推奨のもの」](#)を参照してください。
- Carthage フレームワークマネージャーは、正式にサポートされなくなりました。

Android

バージョンの更新

対象の API: Android 11 (API 30)

廃止

コンパイラーの警告を確認するか、[「Android で現在非推奨のもの」](#)を参照してください。

React Native

TypeScript を新たにサポート

React Native の Mobile SDK の実装では、アプリケーション開発用に標準の JavaScript に加えて、TypeScript を使用できるようになりました。React Native 用の Mobile SDK ライブラリでは、さらにデータ型を使用できるようになりました。TypeScript を使用するには、TypeScript コンパイラーをインストールする必要があります。[React Native 開発](#)

バージョンの更新

React Native バージョン: 0.63.4

Mobile Sync

親子の上位同期用に追加された externalIdField パラメーター

親子の上位同期のための新しい externalIdField パラメーターは、Mobile SDK 8.0 で追加された基本的な同期操作の機能に対応します。[外部 ID による上位同期](#) (ページ 334)。

第 4 章

Mobile SDK 11.1 for iOS および Android の 使用開始

トピック:

- [Developer Edition 環境か Sandbox 環境か](#)
- [iOS および Android の開発の前提条件](#)
- [Salesforce プラットフォームへのサインアップ](#)
- [接続アプリケーションの作成](#)
- [Mobile SDK のインストール](#)
- [Mobile SDK サンプルアプリケーション](#)

カスタムモバイルアプリケーションを作成してみましょう。まだ Salesforce にサインアップして Mobile SDK 開発ツールをインストールしていない場合は、まずこのサインアップとインストールを実行します。

どの開発オプションを選択する場合でも、サインアップすることに加え、接続アプリケーションを定義する必要があります。Mobile SDK for Android または iOS (ハイブリッドおよびネイティブ) をインストールするには、Mobile SDK npm パッケージを使用します。

Developer Edition 環境か Sandbox 環境か

Salesforce では、開発者向けにさまざまな環境を提供しています。最適な環境は、次のような多くの要因によって開発者ごとに異なります。

- 作成しているアプリケーションのタイプ
- アプリケーションの利用者
- 会社のリソース

開発環境の使用目的は、アプリケーションの開発およびテストに限定されています。この環境には、ビジネスに不可欠ではないテストデータが含まれます。開発は、ブラウザー内で行うことも、Visual Studio Code エディター向けの Salesforce 拡張機能で行うこともできます。

開発者環境の種別

Developer Edition 環境は、無料で利用できる全機能が搭載された *Enterprise Edition* 環境のコピーで、ストレージ容量とユーザー数に制限があります。Developer Edition は論理的に独立した環境で、最初の開発環境として理想的です。必要な数の Developer Edition 組織にサインアップできます。そのため、どの Salesforce 本番環境向けに設計されたアプリケーションでも作成できます。

Partner Developer Edition は、無料で利用できる Developer Edition のライセンスバージョンで、より多くのストレージ容量、機能、ライセンスが含まれます。Partner Developer Edition は、登録済みの Salesforce パートナーに無料で提供されます。

Sandbox は本番環境とほぼ一致するコピーであり、Professional Edition、Enterprise Edition、Performance Edition、および Unlimited Edition の顧客が使用できます。Sandbox コピーには、データ、設定、またはその両方を含めることができます。さまざまな目的のために、本番環境のデータとアプリケーションに影響を与えることなく、複数の Sandbox を本番環境で作成できます。

環境の選択

このガイドでは、すべての練習で Developer Edition 組織を使用することを想定しています。ただし、実際には Sandbox 環境でも開発作業を行うことができます。次に、最適な環境の決定に役立つ情報を示します。

- Developer Edition が最適なユーザー
 - 商用 Salesforce アプリケーションの構築を目的に、AppExchange や Trialforce で配布する管理パッケージを作成するパートナー。管理パッケージを作成できるのは、Developer Edition 環境または Partner Developer Edition 環境のみです。
 - Sandbox へのアクセス権のない、Group Edition または Personal Edition の Salesforce 顧客。
 - Salesforce Platform を無料で検証しようとする開発者。
- Partner Developer Edition が最適な場合
 - すべてのソースコードを管理するチーム環境を必要とする開発チームが利用する。この場合、開発者ごとに Developer Edition 環境を設定し、このチームリポジトリ環境との間でコードをチェックインおよびチェックアウトできます。
 - 開発とテストのために 3 人以上の開発者がログインする可能性がある。

- 大量のデータセットに対して多くのユーザーがロバストテストを実行できる大規模な環境を必要とする。
- Sandbox が最適な場合
 - Sandbox が含まれる Professional Edition、Enterprise Edition、Performance Edition、Unlimited Edition、または Salesforce Platform Edition の Salesforce ユーザーである。
 - 本番環境用の Salesforce アプリケーションを開発している。
 - 市販される Salesforce アプリケーションを作成する予定がない。
 - AppExchange に載せたり、Trailforce を使用して配布したりする意図がない。

iOS および Android の開発の前提条件

ある程度の予備知識の習得やシステム設定を経験してから Mobile SDK アプリケーションの作成を始めることをお勧めします。

Salesforce Platform に関するある程度の経験があると役立ちます。Mobile SDK アプリケーションの設計やデバッグを行うには、OAuth、ログインおよびパスワードフロー、Salesforce 接続アプリケーションについて熟知している必要があります。「[モバイルアプリケーションでの認証、セキュリティ、ID](#)」を参照してください。

全般的な要件 (すべてのプラットフォームと環境)

すべての Mobile SDK 開発には、次のソフトウェアが必要です。

- [接続アプリケーション](#)が存在する Salesforce [Developer Edition](#) 組織。

iOS ネイティブ要件

- iOS SDK:
 - リリース対象: iOS 15
 - ベース SDK: iOS 16
- Xcode バージョン: 14 以降。(最新バージョンを推奨)。

Android ネイティブ要件

- Java JDK 11.0.11+9 以降 — www.oracle.com/downloads。
- Android Studio の最新バージョン — developer.android.com/sdk。
- Android SDK — 次の範囲の API をすべて含めます。
 - 最小限の API: Android Nougat (API 24)
 - 対象の API: Android 13 (API 33)
- Android SDK ツール
- Android 仮想デバイス (AVD)

ハイブリッドの要件

- サポートする各モバイルプラットフォームのすべてのネイティブ要件 (forceios および forcedroid npm パッケージを除く)。
- Cordova CLI 12.0.0。
- forcehybrid npm パッケージ、バージョン 11.1。
- HTML5 および JavaScript 言語に習熟している。
- ハイブリッドリモートアプリケーションの場合:
 - Visualforce のある Salesforce 組織。
 - Visualforce の開始ページ。

React Native 要件

- サポートする各モバイルプラットフォームのすべてのネイティブ要件 (forceios および forcedroid npm パッケージを除く)。
- forcereact npm パッケージ、バージョン 11.1。
- JavaScript の習熟 (ES2015)。

Salesforce プラットフォームへのサインアップ

すべての Salesforce 開発者プログラムの豊富なチュートリアル、ブログ、およびサポートフォーラムにアクセスするには、Salesforce Platform に参加します。

1. ブラウザーで <https://developer.salesforce.com/signup> にアクセスします。
2. 各項目にユーザー情報と会社情報を入力します。
3. [Email Address (メールアドレス)] 項目には、Web ブラウザーから簡単に確認できる公開アドレスを使用してください。
4. 一意の [Username (ユーザー名)] を入力します。ユーザー名もメールアドレスの形式にする必要がありますが、メールアドレスと同じにする必要はなく、通常は違うものを入力することをお勧めします。ユーザー名は `developer.salesforce.com` でのログイン情報および ID であるため、担当する作業を説明するユーザー名 (`develop@workbook.org`) や自分自身を表すユーザー名 (`firstname@lastname.com`) を選ぶことで、より有益に使用できます。
5. メインサービス契約を読み、チェックボックスをオンにします。
6. 表示された画像認証文字列を入力し、[Submit Registration (登録を実行)] をクリックします。
7. その後まもなく、ログインリンクを記載したメールが届きます。リンクをクリックし、パスワードを変更します。

接続アプリケーションの作成

モバイルアプリケーションから Salesforce サービスに接続できるようにするには、接続アプリケーションを作成する必要があります。接続アプリケーションには、このガイドのすべての開発シナリオの前提条件となるコンシューマー鍵が含まれます。

接続アプリケーションを作成する

Salesforce システム管理者は、Salesforce サーバーで接続アプリケーションを作成します。Salesforce 接続アプリケーションには、他のモバイル製品 (Salesforce アプリケーションなど) でのみ使用される多くの設定があります。次の手順では、Mobile SDK アプリケーションに適用される設定がカバーされています。

接続アプリケーションを作成する手順は、次のとおりです。

1. Salesforce インスタンスにログインします。
2. [設定] から、[クイック検索] ボックスに「アプリケーション」と入力し、[アプリケーション] を選択します。
3. [接続アプリケーション] で、[新規] をクリックします。
4. 「**基本情報**」の手順を実行します。
5. 「**API (OAuth 設定の有効化)**」の手順を実行します。
6. 該当する場合、「**モバイルアプリケーション設定**」の省略可能な手順を実行します。
7. [保存] をクリックします。

プッシュ通知をサポートする予定の場合、追加の接続アプリケーション設定については、「[プッシュ通知と Mobile SDK](#)」を参照してください。現在必要な情報がない場合は、後でこれらの設定を追加できます。

接続アプリケーションを作成したら、コンシューマー鍵とコールバック URL をコピーして保管します。Mobile SDK アプリケーションのこれらの値を使用して OAuth 設定を行います。後でこれらの値を Lightning Experience で参照する手順は、次のとおりです。

1. Lightning Experience では、[設定] に移動します。
2. [アプリケーション] > [アプリケーションマネージャー] に移動します。
3. 未管理接続アプリケーション ([アプリケーション種別] が [接続済み]) を選択します。
4. 行の右端にあるドロップダウンボタンをクリックし、[表示] を選択します。

メモ:

- 基本的な Mobile SDK アプリケーションの場合、コールバック URL は有効な URL である必要はありません。必要なのは、アプリケーションでこの項目に期待する内容と一致していることです。
「`sfdc://`」など、任意のカスタムプレフィックスを使用できます。
- 高度な認証をサポートするには、コールバック URL がカスタム URI スキームを使用する有効なエンドポイントである必要があります。
- IdP の使用事例の場合、コールバック URL が HTTPS プロトコルを使用する有効なエンドポイントである必要があります。
- 接続アプリケーションの詳細ページに、コンシューマー鍵が表示されます。このキーは後で必要になるため、コピーしておくことをお勧めします。

- 新しい接続アプリケーションを作成したら、トークンが伝搬されるまで数分待ってから、アプリケーションを実行します。

基本情報

このセクションでは、Mobile SDK アプリケーションに関する基本情報を指定します。

1. 接続アプリケーション名を入力し、Enter キーを押します。組織の接続アプリケーション間で一意の名前を入力する必要があります。名前にはスペースを含めることができます。

 **メモ:** Salesforce によって、スペースのない名前が [API 参照名] 項目に自動的に入力されます。

2. 取引先責任者のメールを入力します。

他の基本情報項目は省略可能で、Mobile SDK では使用されません。

API (OAuth 設定の有効化)

1. [OAuth 設定の有効化] を選択します。
2. コールバック URL (エンドポイント) を入力します。Mobile SDK は、この URL を使用して、認証中時にアプリケーションにコールバックします。この値は、アプリケーションのプロジェクト設定で指定した OAuth リダイレクト URI と一致している必要があります。
3. [選択した OAuth 範囲] で、[データへのアクセスと管理 (api)]、[ユーザーに代わっていつでも要求を実行 (refresh_token、offline_access)]、[Web 経由のデータへのアクセスを提供 (web)] を選択します。
4. デバイスのネイティブブラウザ経由で認証を実行する Mobile SDK アプリケーションをサポートするには、[Web サーバーフローの秘密が必要] をオフにします。

モバイルアプリケーション設定

このセクションの大部分の設定は、Mobile SDK では使用されません。次に例外を示します。

1. PIN 保護をサポートするには、[PIN 保護] を選択します。
2. プッシュ通知をサポートするには、[プッシュメッセージングの有効化] を選択します。このセクションの手順は、『Salesforce Mobile プッシュ通知実装ガイド』の「[ステップ 2: 接続アプリケーションの作成](#)」を参照してください。

関連トピック

Salesforce ヘルプの「[接続アプリケーションの作成](#)」。

関連トピック:

[範囲パラメーターの値](#)

Mobile SDK のインストール

Salesforce Mobile SDK には、2つのインストールパスがあります。

- Mobile SDK オープンソースコードを GitHub からダウンロードし、独自の開発環境を設定します。
- Salesforce が提供する Node Packaged Module (npm) スクリプトを使用して、独自の実行可能な Mobile SDK プロジェクトを作成します。

Node.js、npm、Git コマンドラインのインストール

多くのモバイル開発者は、Mobile SDK を「ブラックボックス」として使用することで、できるだけ早くアプリケーションの作成を開始したいと考えています。このような用途では、Mobile SDK で提供されるさまざまな npm コマンドラインスクリプトを使用してアプリケーションを作成できます。これらの npm ツールを使用する場合、まず Node.js をインストールします。Node.js インストーラーにより、npm が自動的にインストールされます。Git コマンドラインの最新バージョンもインストールできます。

Mobile SDK 11.1 ツールでは、次の最小バージョンが使用されます。

- npm 3.10
- Git コマンドライン (最新バージョン)

 **メモ:** Mobile SDK npm ツールのインストールはネイティブ iOS 開発では省略可能ですが推奨されます。その他のすべてのプラットフォームでは、これらのツールを強くお勧めします。

1. www.nodejs.org から Node.js インストーラーをダウンロードします。
2. インストーラーを実行し、インストールの許可を促すメッセージをすべて承諾します。このモジュールでは、node.js と npm の両方をインストールします。
3. コマンドプロンプトで `npm` コマンドを実行し、インストールをテストします。コマンドの使用状況情報のページが表示されない場合、ステップ 2 に戻り、何が欠落しているのかを調べます。
4. コマンドラインで `git` コマンドが認識されない場合は、<https://git-scm.com/> にアクセスし、使用するシステム向けの最新の Git パッケージをダウンロードしてインストールします。

 **メモ:** `git` コマンドラインユーティリティは Xcode に含まれています。

これで、iOS と Android のアプリケーションを作成する Salesforce Mobile SDK npm スクリプトをダウンロードできるようになりました。

Mobile SDK npm パッケージ

ほとんどのモバイル開発者は、Mobile SDK を「ブラックボックス」として使用することで、できるだけ早くアプリケーションの作成を開始したいと考えています。この使用事例の場合、Salesforce は、一連の npm パッケージを提供します。各パッケージでコマンドラインツールがインストールされます。このコマンドラインツールをターミナルウィンドウまたは Windows コマンドプロンプトで使用できます。

Mobile SDK コマンドラインツールには、SDK リリースの静的スナップショットが含まれています。iOS の npm パッケージでは、未コンパイルのソースコードではなくバイナリモジュールがインストールされます。Android

の npm パッケージでは、バイナリではなく SDK ソースコードのスナップショットがインストールされます。npm スクリプトを使用して、Mobile SDK ライブラリにアクセスするだけでなく、プロジェクトも作成します。Mobile SDK では、次のコマンドラインツールが提供されます。

forcedroid

ネイティブ Android プロジェクトを Java または Kotlin で作成します。

forceios

ネイティブ iOS プロジェクトを Swift または Objective-C で作成します。

forcehybrid

iOS、Android、またはその両方のビルドターゲットを使用して、Cordova に基づいてハイブリッドプロジェクトを作成します。

forcereact

iOS、Android、またはその両方のビルドターゲットを使用して、React Native プロジェクトを作成します。

npm パッケージは、<https://www.npmjs.org> にあります。次のコマンドを使用して、Mobile SDK パッケージをグローバルにインストールすることをお勧めします。

- **Windows の場合:**

```
npm install -g <npm-package-name>
```

npm-package-name は、次のいずれかです。

- forcedroid
- forceios
- forcehybrid
- forcereact

- パッケージをローカルにインストールするには、-g を省略します。

- **Mac OS X の場合:**

```
sudo npm install -g <npm-package-name>
```

npm-package-name は、次のいずれかです。

- forcedroid
- forceios
- forcehybrid
- forcereact

- /usr/local/bin/ への「参照・更新」が権限がある場合、sudo を省略できます。

- パッケージをユーザー所有のディレクトリにローカルにインストールするには、sudo と -g を省略します。

- システムに npm が存在しない場合、nodejs.org から Node.js の最新リリースをインストールします。
- npm パッケージではソース制御がサポートされないため、新しいリリース用にインストールを動的に更新することはできません。代わりに、リリースごとに別個にインストールします。新しいバージョンの SDK にアップグレードするには、[npmjs.org](https://www.npmjs.org) の Web サイトにアクセスして、新しいパッケージをダウンロードしてください。

-  **メモ:** forceios npm は必須ではないけれど便利な機能として提供されています。CocoaPods、node.js、npm は forceios には必須ですが、Mobile SDK iOS 開発には必須ではありません。forceios を使用せずに Mobile SDK iOS ネイティブプロジェクトを作成する方法については、「[iOS Swift プロジェクトの手動作成](#)」を参照してください。

iOS の準備

iOS 用の Mobile SDK アプリケーションを作成するには、必須の Apple ソフトウェアをインストールする必要があります。forceios を使用する場合は、CocoaPods もインストールします。

Mobile SDK 4.0 以降、forceios スクリプトは CocoaPods を使用して Mobile SDK モジュールをインポートします。forceios で作成されたアプリケーションは、CocoaPod 駆動型ワークスペースで実行されます。CocoaPods ユーティリティでは、Mobile SDK ソースコードをワークスペースで使えるようにしてデバッグを強化します。また、CocoaPods を使用すれば、新しい Mobile SDK バージョンに簡単に更新できます。プロファイルを更新して、ターミナルウィンドウで `pod update` を実行するだけで済みます。

次の指示にしたがって、iOS での Mobile SDK 開発の準備がすべて整っていることを確認します。

- 開発している iOS アプリケーションの種別に関係なく、「[iOS の基本的な要件](#)」に記載されている要件を十分に理解していることを確認します。
 - (省略可能) forceios を使用する場合は次の手順を実行します。
 - forceios をインストールするには、「[Mobile SDK npm パッケージ](#)」を参照してください。
 - forceios を使用するには、CocoaPods をインストールする必要があります。guides.cocoapods.org の「[Getting Started \(始めましょう\)](#)」を参照してください。
-  **メモ:** forceios npm は必須ではないけれど便利な機能として提供されています。CocoaPods、node.js、npm は forceios には必須ですが、Mobile SDK iOS 開発には必須ではありません。forceios を使用せずに Mobile SDK iOS ネイティブプロジェクトを作成する方法については、「[iOS Swift プロジェクトの手動作成](#)」を参照してください。

関連トピック:

[CocoaPods と Mobile SDK の使用](#)

[Mobile SDK Pod の更新](#)

Android の準備

Android の Mobile SDK アプリケーション (ネイティブ、ハイブリッド、または React Native) の作成を試みる前に、Android ネイティブ開発環境をインストールします。

次の指示に従って、Android での Mobile SDK 開発の準備がすべて整っていることを確認します。

- 「[ネイティブ Android の要件](#)」に記載されている要件を十分に理解していることを確認します。
- forcedroid、forcehybrid、または forcereact をインストールするには、「[Mobile SDK npm パッケージ](#)」を参照してください。

Mobile SDK npm パッケージのアンインストール

npm パッケージをアンインストールする必要がある場合は、npm スクリプトを使用します。

グローバルインストールのアンインストール

グローバルインストールの場合、任意のフォルダーから次のコマンドを実行します。

- **Mac OS X の場合:**

```
sudo npm uninstall <package-name> -g
```

/usr/local/bin/ の「参照・更新」権限がない場合、sudo を使用します。

- **Windows の場合:**

```
npm uninstall <package-name> -g
```

<package-name> を次のいずれかの値で置き換えます。

- forcedroid
- forceios
- forcehybrid
- forcereact

ローカルインストールのアンインストール

ローカルインストールの場合、パッケージをインストールしたフォルダーから次のコマンドを実行します。

- **Mac OS X または Windows の場合:**

```
npm uninstall <package-name>
```

<package-name> を次のいずれかの値で置き換えます。

- forcedroid
- forceios
- forcehybrid
- forcereact

Mobile SDK GitHub リポジトリ

開発者は、GitHub で SDK を探求して、変更を最新に保ち、さらに SDK 開発に貢献するなどを積極的に行うことができます。GitHub を使用すれば、公開されたプレリリースの開発環境でソースコードを監視できます。この場合、アプリケーションと共に構築される SDK ソースコードがアプリケーションに含まれます。

 **重要:** 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

Mobile SDK にアクセスするために GitHub にサインアップする必要はありませんが、このソーシャルコーディングコミュニティに参加することをお勧めします。 <https://github.com/forcedotcom>

最新の Mobile SDK リリースは、次の公開リポジトリからいつでも入手できます。

- <https://github.com/forcedotcom/SalesforceMobileSDK-iOS>
- <https://github.com/forcedotcom/SalesforceMobileSDK-Android>

 **重要:** Mobile SDK プラットフォームのプル要求を送信するには、変更のベースとして **dev** ブランチをチェックアウトします。

GitHub を使用して最新リリースのソースコードの作成のみを行う場合、**master** ブランチをチェックアウトします。

Mobile SDK for iOS GitHub リポジトリのコピー (省略可能)

多くのツールは、ローカル GitHub リポジトリをコピーおよび管理するために存在しています。次の手順は、Mobile SDK をコピーする 3 つの一般的な方法と、コピー後の重要なインストール手順を示しています。

1. リポジトリをコピーします。たとえば、次のいずれかのオプションを使用できます。

- コマンドライン:

- a. OSX ターミナルアプリケーションで、Xcode と共にインストールされる `git` コマンドを使用します。

```
git clone git://github.com/forcedotcom/SalesforceMobileSDK-iOS.git
```

- Xcode:

- a. [Source Control (ソース制御)] > [Clone (複製)] を選択します。
- b. [Save As (別名で保存)] で、デフォルトを受け入れるか、カスタムディレクトリ名を入力します。
- c. [Where (場所)] で、任意の親ディレクトリを選択します。
- d. [Clone (コピー)] をクリックします。

- Mac 用の GitHub Desktop (<https://desktop.github.com/>):

- a. [Clone a Repository from the Internet (インターネットからリポジトリをコピー)] をクリックします。
- b. [URL] をクリックします。
- c. [Repository URL (リポジトリ URL)] または [GitHub username and repository (GitHub ユーザー名およびリポジトリ)] で、<https://github.com/forcedotcom/SalesforceMobileSDK-iOS> と入力します。
- d. [Local Path (ローカルパス)] を任意の対象に設定します。
- e. [Clone (コピー)] をクリックします。

2. OSX ターミナルアプリケーションで、リポジトリをコピーしたディレクトリ (デフォルトでは `SalesforceMobileSDK-iOS`) に変更します。

3. コマンドラインからインストールスクリプト `./install.sh` を実行します。

 **メモ:** `install.sh` スクリプトにより、正しいプラットフォーム開発ツールがインストールされているかどうかチェックされます。次に、Mobile SDK の開発に必要な外部サブモジュールをローカルコピーに同期します。開発マシンに Xcode の無効な Mobile SDK プロジェクトテンプレートが保存されている場合、それらの不要なファイルはスクリプトによって削除されます。

Mobile SDK for Android GitHub リポジトリのコピー (省略可能)

1. ブラウザーで、Mobile SDK for Android GitHub リポジトリ <https://github.com/forcedotcom/SalesforceMobileSDK-Android> に移動します。
2. 次のコマンドを使用して、リポジトリをローカルファイルシステムにコピーします。

```
git clone git://github.com/forcedotcom/SalesforceMobileSDK-Android.git
```

3. コピーしたリポジトリをインストールしたディレクトリで、ターミナルプロンプトまたはコマンドウィンドウを開きます。
4. `./install.sh` (Mac) または `cscript install.vbs` (Windows) を実行します。

 **メモ:** インストールスクリプトにより、Mobile SDK の開発に必要な外部サブモジュールがローカルコピーに同期されます。Windows で `cscript install.vbs` を実行すると、`git status` によって、変更されたファイルおよび削除されたファイルのリストが返されます。この出力は、リポジトリのシンボリックリンクの解決の好ましくない副作用です。これらのファイルは、クリーンアップしたり、元に戻したりしないでください。

コピーされた GitHub リポジトリを使用した Android プロジェクトの作成

コピーされた `SalesforceMobileSDK-Android` リポジトリを使用してネイティブおよびハイブリッドプロジェクトを作成するには、`native/README.md` および `hybrid/README.md` ファイルの手順に従います。

コピーされた GitHub リポジトリを使用した iOS プロジェクトの作成

コピーされた `SalesforceMobileSDK-iOS` リポジトリを使用してプロジェクトを作成するには、リポジトリの `readme.md` ファイルの手順に従います。

関連トピック:

[Node.js](#)、[npm](#)、[Git コマンドラインのインストール](#)

Mobile SDK サンプルアプリケーション

Salesforce Mobile SDK には、主要な機能を示す豊富なサンプルアプリケーションが含まれています。ハイブリッドサンプルとネイティブサンプルは、独自のアプリケーションの基盤として使用したり、参考のために検討したりできます。

サンプルアプリケーションのインストール

GitHub では、サンプルアプリケーションは対象プラットフォームの Mobile SDK リポジトリに存在します。ハイブリッドサンプルでは、Cordova コマンドラインで `SalesforceMobileSDK-Shared` リポジトリのソースコードを使用できます。

GitHub リポジトリからサンプルアプリケーションへのアクセス

Android の場合:

- SalesforceMobileSDK-Android GitHub リポジトリ (<https://github.com/forcedotcom/SalesforceMobileSDK-Android>) をコピーまたは更新します。
- リポジトリのルートフォルダーで、次のインストールスクリプトを実行します。
 - **Windows:** `cscript install.vbs`
 - **Mac OS X:** `./install.sh`
- Android Studio で、SalesforceMobileSDK-Android のローカルコピーがあるフォルダーをインポートします。
- `hybrid/HybridSampleApps` および `native/NativeSampleApps` プロジェクトフォルダーでサンプルアプリケーションを見つけます。



重要: Windows では、必ずシステム管理者として Android Studio を実行します。

iOS の場合:

- SalesforceMobileSDK-iOS GitHub リポジトリ (<https://github.com/forcedotcom/SalesforceMobileSDK-iOS>) をコピーまたは更新します。
- リポジトリのルートフォルダーにある `./install.sh` を実行します。
- Xcode で、SalesforceMobileSDK-iOS/SalesforceMobileSDK.xcworkspace ファイルを開きます。
- NativeSamples および HybridSamples ワークスペースフォルダーでサンプルアプリケーションを探します。

Cordova でのハイブリッドサンプルアプリケーションの作成

Cordova コマンドラインを使用してハイブリッドサンプルアプリケーションを作成するには、「[ハイブリッドサンプルアプリケーションを作成する](#)」を参照してください。

Android サンプルアプリケーション

ネイティブ

これらのサンプルは、[SalesforceMobileSDK-Android/native](#) リポジトリの `NativeSampleApps` ディレクトリにあります。

- **MobileSyncExplorer:** Android 上でのネイティブ Mobile Sync ライブラリの機能を示します。これは、`native/NativeSampleApps/MobileSyncExplorer` の Mobile SDK for Android にあります。
- **RestExplorer:** Mobile SDK の OAuth および REST API 機能を示します。また、REST API アクションをタブレットから調べる場合にも便利です。

ハイブリッド

これらのサンプルは、[SalesforceMobileSDK-Android/hybrid](#) リポジトリの `HybridSampleApps` ディレクトリにあります。

- **AccountEditor:** `mobilesync.js` ライブラリを使用してオフラインデータを同期する方法について説明します。
- **MobileSyncExplorerHybrid:** Mobile Sync プラグインを使用してオフラインデータを同期する方法について説明します。
- **NoteSync:** 非 REST API を使用して Salesforce のメモを取得する方法を示します。

テンプレートアプリケーション

`forcedroid create` または `forcedroid createwithtemplate` によってテンプレートアプリケーションを使用することで、MobileSDK の「スターター」アプリケーションを作成できます。それらのテンプレートは、github.com/forcedotcom/SalesforceMobileSDK-Templates リポジトリで入手できます。

`forcedroid create` による使用

- **AndroidNativeKotlinTemplate:** Kotlin で記述された標準の Android ネイティブテンプレート。`forcedroid create` コマンドのデフォルトのアプリケーション種別です。`forcedroid create` でアプリケーション種別を求められたら、`Return` を押すか、`native_kotlin` と入力します。
- **AndroidNativeTemplate:** Java で記述された標準の Android ネイティブテンプレート。`forcedroid create` でアプリケーション種別を求められたら、`native` と入力します。

`forcedroid createwithtemplate` と共に使用

- **AndroidIDPTemplate:** Android アプリケーションでの ID プロバイダーの設定方法を示したテンプレートです。`forcedroid createwithtemplate` でリポジトリ URI を求められたら、`AndroidIDPTemplate` を入力します。
- **MobileSyncExplorerKotlinTemplate:** Mobile Sync の包括的な使用例を提供します。`forcedroid createwithtemplate` でリポジトリ URI を求められたら、`MobileSyncExplorerKotlinTemplate` と入力します。

iOS サンプルアプリケーション

ネイティブ

これらのサンプルは、[SalesforceMobileSDK-iOS/native](https://github.com/forcedotcom/SalesforceMobileSDK-iOS/native) リポジトリの `SampleApps` ディレクトリにあります。

- **MobileSyncExplorer** は、iOS 上でのネイティブ Mobile Sync ライブラリの機能を示します。これは、`native/SampleApps/MobileSyncExplorer` の Mobile SDK for iOS にあります。
- **RestAPIExplorer** では、ネイティブのすべての REST API ラッパーが実行されます。これは、`native/SampleApps/RestAPIExplorer` の Mobile SDK for iOS にあります。

ハイブリッド

これらのサンプルは、[SalesforceMobileSDK-iOS-Hybrid](https://github.com/forcedotcom/SalesforceMobileSDK-iOS-Hybrid) リポジトリの `hybrid/SampleApps` ディレクトリにあります。

- **AccountEditor:** `mobilesync.js` ライブラリを使用してオフラインデータを同期する方法について説明します。
- **MobileSyncExplorer:** Mobile Sync プラグインを使用してオフラインデータを同期する方法について説明します。

- **NoteSync:** 非 REST API を使用して Salesforce のメモを取得する方法を示します。

テンプレートアプリケーション

`forceios create` または `forceios createwithtemplate` によってテンプレートアプリケーションを使用することで、Mobile SDK の「スターター」アプリケーションを作成できます。それらのテンプレートは、github.com/forcedotcom/SalesforceMobileSDK-Templates リポジトリで入手できます。

`forceios create` による使用

- **iOSNativeSwiftTemplate:** Swift で記述された標準の iOS ネイティブテンプレート。`forceios create` コマンドのデフォルトのアプリケーション種別です。`forceios create` でアプリケーション種別を求められたら、`Return` を押すか、`native_swift` と入力します。
- **iOSNativeTemplate:** Objective-C で記述された標準の iOS ネイティブテンプレート。`forceios create` でアプリケーション種別を求められたら、`native` と入力します。

`forceios createwithtemplate` と共に使用

- **iOSIDPTemplate:** iOS アプリケーションでの ID プロバイダーの設定方法を示したテンプレートです。`forceios createwithtemplate` でリポジトリ URI を求められたら、`iOSIDPTemplate` を入力します。
- **iOSNativeSwiftEncryptedNotificationTemplate:** iOS アプリケーションで、暗号化された通知を受信するための設定方法を示したテンプレートです。`forceios createwithtemplate` でリポジトリ URI を求められたら、`iOSNativeSwiftEncryptedNotificationTemplate` を入力します (またはコピー/貼り付けします)。
- **MobileSyncExplorerSwift:** Mobile Sync の包括的な使用例を提供します。`forceios createwithtemplate` でリポジトリ URI を求められたら、`MobileSyncExplorerSwift` と入力します。

ハイブリッドサンプルアプリケーション (ソースのみ)

Mobile SDK では、ほとんどのハイブリッドサンプルアプリケーションの Web アプリケーションソースコードのみが提供されています。Cordova コマンドラインを使用して、これらのアプリケーションのプラットフォーム固有のバージョンを作成できます。ソースコードを入手するには、[SalesforceMobileSDK-Shared](https://github.com/forcedotcom/SalesforceMobileSDK-Shared) GitHub リポジトリをコピーし、`samples` フォルダを確認します。特定のモバイルプラットフォーム用にこれらのハイブリッドアプリケーションを作成する場合は、「[ハイブリッドサンプルアプリケーションを作成する](#)」の説明を参照してください。

- **accounteditor:** Mobile Sync を使用して Salesforce データにアクセスします。
- **contactexplorer:** Cordova を使用してローカルデバイスの取引先責任者を取得します。また、`force.js` ツールキットを使用して、Salesforce REST API で REST トランザクションを実装します。アプリケーションでは、Salesforce SDK の OAuth2 サポートを使用して OAuth ログイン情報を取得し、JavaScript イベントを送信することによって、そのログイン情報を `force.js` に伝搬します。
- **fileexplorer:** Files API を示します。
- **mobilesyncexplorer:** オフライン同期において、Mobile Sync プラグインではなく、`mobilesync.js` を使用してデモを実施します。
- **notesync:** 非 REST API を使用して Salesforce のメモを取得します。
- **simplesyncreact:** Mobile Sync プラグインを使用する React Native アプリケーションのデモを実施します。

- **smartstoreexplorer:** SmartStore API を探索できます。
- **userandgroupsearch:** グループのユーザーを検索できます。
- **userlist:** 組織のユーザーをリストします。これは、最も単純なハイブリッドサンプルアプリケーションです。
- **usersearch:** 組織のユーザーを検索できます。
- **vfconnector:** ネイティブコンテナ内の Visualforce ページをラップします。この例では、BasicVFTest という Visualforce ページが組織に存在することを想定しています。アプリケーションではまず、SalesforceSDKOAuth2 サポートを使用して OAuth のログイン情報を取得し、そのログイン情報を使用して、Visualforce ページにアクセスするための適切な Web ビュー Cookie を設定します。

React Native サンプルアプリケーション

テンプレートアプリケーション

`forcereact create` または `forcereact createwithtemplate` によってテンプレートアプリケーションを使用することで、MobileSDK の「スターター」アプリケーションを作成できます。それらのテンプレートは、github.com/forcedotcom/SalesforceMobileSDK-Templates リポジトリで入手できます。

`forcereact create` による使用

- **ReactNativeTypeScriptTemplate:** TypeScript で記述された標準の React Native テンプレート。`forcereact create` コマンドのデフォルトのアプリケーション種別です。`forcereact create` でアプリケーション種別を求められたら、`Return` を押すか、`react_native_typescript` と入力します。
- **ReactNativeTemplate:** ES2015 (プレーン JavaScript) で記述された React Native テンプレート。`forcereact create` でアプリケーション種別を求められたら、`react_native` と入力します。

`forcereact createwithtemplate` と共に使用

- **ReactNativeDeferredTemplate:** React Native アプリケーションで遅延ログインを使用する方法を示します。`forcereact createwithtemplate` でリポジトリ URI を求められたら、`ReactNativeDeferredTemplate` と入力します。
- **MobileSyncExplorerReactNative:** Mobile Sync の包括的な使用例を提供します。`forcereact createwithtemplate` でリポジトリ URI を求められたら、`MobileSyncExplorerReactNative` と入力します。

トピック:

- [Maven を使用した Android アプリケーションの Mobile SDK ライブラリの更新](#)

ネイティブアプリケーションと React Native アプリケーションの将来の Mobile SDK のアップグレードがより簡単になります。アプリケーションを作成して、そこにアプリケーションのリソースをエクスポートするのではなく、簡単な設定ファイルを更新してスクリプトを実行することで、新しい SDK ライブラリを使用してアプリケーションを再生成できるようになりました。

ネイティブアプリケーションと React Native アプリケーションの更新

ネイティブアプリケーションと React Native アプリケーションの各ディレクトリには、ルートレベルに `package.json` ファイルがあります。この JSON ファイルには、Mobile SDK ソースパスを示す名前-値ペアのリストが含まれた「dependencies」オブジェクトが含まれています。これらの値を、プラットフォームの Mobile SDK の有効なコピーを参照するローカルパスまたはネットワークパスに設定できます。このファイルを更新した後、次のコマンドを実行して更新を行います。

- `install.js` (Android ネイティブアプリケーション、iOS ネイティブアプリケーション、ネイティブ Swift アプリケーションの場合)
- `installandroid.js` (Android 上の React Native アプリケーションの場合)
- `installios.js` (iOS 上の React Native アプリケーションの場合)

アプリケーションのルートフォルダーに適切なファイルがあります。

たとえば、ネイティブ Android `package.json` ファイルの `dependencies` セクションは次のようになります。

```
"dependencies": {
  "salesforcemobilesdk-android":
    "https://github.com/forcedotcom/SalesforceMobileSDK-Android.git"
}
```

このパスは、SalesforceMobileSDK-Android リポジトリの最新リリースブランチを参照します。

iOS の場合も考え方は同じです。

```
"dependencies": {
  "salesforcemobilesdk-ios":
    "https://github.com/forcedotcom/SalesforceMobileSDK-iOS.git"
}
```

React Native の場合、iOS と Android の両方と React Native バージョンのターゲットを設定できます。

```
"sdkDependencies": {
  "SalesforceMobileSDK-Android":
    "https://github.com/forcedotcom/SalesforceMobileSDK-Android.git",
  "SalesforceMobileSDK-iOS":
    "https://github.com/forcedotcom/SalesforceMobileSDK-iOS.git"
},
"dependencies": {
  "@react-native-community/masked-view": "^0.1.10",
  "@react-navigation/native": "^6.0.2",
  "@react-navigation/stack": "^6.0.7",
  "react": "17.0.2",
  "react-native-force":
    "git+https://github.com/forcedotcom/SalesforceMobileSDK-ReactNative.git",

  "react-native": "0.65.1",
  "react-native-gesture-handler": "^1.10.3",
  "react-native-safe-area-context": "^3.3.0",
  "react-native-screens": "^3.6.0",
  "create-react-class": "^15.7.0"
},
```

❗ 重要: React Native バージョンは、互換性のある Mobile SDK バージョンとペアになっている必要があります。

Mobile SDK リポジトリの開発ブランチ、つまり今後のリリースが開発されるブランチを参照するには、URL に「#dev」を追加します。次に例を示します。

```
"dependencies": {
  "salesforcemobilesdk-android":
    "https://github.com/forcedotcom/SalesforceMobileSDK-Android.git#dev"

  "SalesforceMobileSDK-iOS":
    "https://github.com/forcedotcom/SalesforceMobileSDK-iOS.git#dev"
},
"dependencies": {
  "@react-native-community/masked-view": "^0.1.10",
  "@react-navigation/native": "^6.0.2",
  "@react-navigation/stack": "^6.0.7",
  "react": "17.0.2",
  "react-native-force":
    "git+https://github.com/forcedotcom/SalesforceMobileSDK-ReactNative.git#dev",

  "react-native": "0.65.1",
  ...
}
```

package.json ファイルを変更したら、必ず例で示しているように Mobile SDK の git インストーラースクリプトを実行してください。

 **例:** アプリケーションを別のバージョンの Mobile SDK for iOS にアップグレードする手順は、次のとおりです。

1. アプリケーションディレクトリから、編集のために `package.json` を開きます。
2. 「`sdkDependencies`」セクションで、「`SalesforceMobileSDK-iOS`」の値を、異なるバージョンの `SalesforceMobileSDK-iOS` リポジトリを参照するように変更します。開発ブランチまたはマスターブランチの異なるタグを参照できます (5.x 以降)。
3. リポジトリのルートディレクトリから、アプリケーションに適切なインストーラースクリプトを実行します。
 - ネイティブアプリケーション: `install.js`
 - React Native アプリケーション: `installios.js`

Android での手順は iOS 表示ラベルを除いて同じです。

1. アプリケーションディレクトリから、編集のために `package.json` を開きます。
2. 「`sdkDependencies`」セクションで、「`SalesforceMobileSDK-Android`」の値を、異なるバージョンの `SalesforceMobileSDK-Android` リポジトリを参照するように変更します。開発ブランチまたはマスターブランチの異なるタグを参照できます (5.x 以降)。
3. リポジトリのルートディレクトリから、アプリケーションに適切なインストーラースクリプトを実行します。
 - ネイティブアプリケーション: `install.js`
 - React Native アプリケーション: `installandroid.js`

ハイブリッドアプリケーションの更新

ハイブリッドアプリケーションの場合、Mobile SDK ライブラリは Mobile SDK Cordova プラグインを介して配信されます。ただし、メジャーリリースの場合は、新しいテンプレートアプリケーションで開始することをお勧めします。

1. `forcehybrid create` を実行します。
2. 既存のプロジェクトと同じ名前と同じ種別のハイブリッドプロジェクトを異なるフォルダーに作成します。
3. スクリプトが完了したら、`cd` を実行して新しいプロジェクトフォルダーに移動します。
4. 元のアプリケーションでサードパーティの Cordova プラグインを使用していた場合は、それを追加します。たとえば、アプリケーションで Cordova ステータスバープラグインを使用する場合は、次のように入力します。

```
cordova plugin add cordova-plugin-statusbar
```

5. Web アプリケーションリソース (JavaScript、HTML5、および CSS ファイルなど) を元のプロジェクトから新しいプロジェクトの `www/` フォルダーにコピーします。たとえば、Mac OS X では、次のコマンドを実行します。

```
cp -RL ~/MyProjects/MyMobileSDK50Project/www/* www/
```

6. `cordova prepare` を実行します。

 **メモ:** 特定のリリースで必要な変更の詳細は、「[以前のリリースからの移行](#)」を参照してください。

Maven を使用した Android アプリケーションの Mobile SDK ライブラリの更新

Mobile SDK 9.2.0 以降では、Android ライブラリを Maven Central で使用できるようになります。Mobile SDK ライブラリを使用するには、アプリケーションの `build.gradle` ファイルの `dependencies` セクションに 1 行を追加します。

Gradle を使用してライブラリを [Maven Central](#) からインポートするには、プロジェクトの `build.gradle` ファイルの `dependencies` セクションに `implementation` ステートメントを追加します。Gradle を使用してライブラリを更新するには、`implementation` ステートメント内のバージョン番号を更新バージョンに変更し、ライブラリを再同期するだけです。

詳細

一般的な Gradle の `dependencies` セクションは次のようになります。

```
dependencies {  
    classpath 'com.android.tools.build:gradle:7.0.2'  
}
```

`implementation` ステートメントには次の形式を使用します。

```
implementation '<groupId>:<artifactID>:<version>'
```

Mobile SDK ライブラリの場合:

- `groupId` は「`com.salesforce.mobilesdk`」
- `artifactID` は「`SalesforceSDK`」、「`SalesforceHybrid`」、「`SmartStore`」、または「`MobileSync`」
- `version` は「`x.x.x`」(「`9.2.0`」など)

`implementation` ステートメントでは、指定したライブラリだけではなく、そのすべての連動関係もインポートされます。そのため、たとえば `SalesforceAnalytics` は他のすべてのライブラリと連動するため、明示的にコンパイルする必要はありません。つまり、1つのステートメントだけで必要なすべてを取得できます。

Mobile SDK 9.2.0 ライブラリをインポートするには、次のいずれかの行を追加します。

- `SalesforceSDK` ライブラリの場合:

```
implementation 'com.salesforce.mobilesdk:SalesforceSDK:9.2.0'
```

- `SmartStore` ライブラリの場合 (`SalesforceSDK` ライブラリもインポート):

```
implementation 'com.salesforce.mobilesdk:SmartStore:9.2.0'
```

- `Mobile Sync` ライブラリの場合 (`SalesforceSDK` および `SmartStore` ライブラリもインポート):

```
implementation 'com.salesforce.mobilesdk:MobileSync:9.2.0'
```

- `SalesforceHybrid` ライブラリの場合 (`SalesforceSDK`、`SmartStore`、`Mobile Sync`、および `Apache Cordova` ライブラリをインポート):

```
implementation 'com.salesforce.mobilesdk:SalesforceHybrid:9.2.0'
```

- SalesforceReact ライブラリの場合 (SalesforceSDK、SmartStore、および Mobile Sync ライブラリをインポート):

```
implementation 'com.salesforce.mobilesdk:SalesforceReact:9.2.0'
```

 メモ:

- MobileSDKには、適切な初期化と設定に関するいくつかのコーディング要件があります。使用を開始するには、「[Android アプリケーションの構造](#)」を参照してください。

第 6 章

ネイティブ iOS の開発

トピック:

- [iOS ネイティブクイックスタート](#)
- [iOS の基本的な要件](#)
- [forceios を使用した iOS プロジェクトの作成](#)
- [iOS Swift プロジェクトの手動作成](#)
- [CocoaPods と Mobile SDK の使用](#)
- [Mobile SDK での Carthage の使用 \(サポート対象外\)](#)
- [ネイティブ iOS アプリケーションの開発](#)
- [Mobile SDK アプリケーションでの iPadOS のサポート](#)
- [Mobile SDK アプリケーションでの Catalyst のサポート](#)
- [Mobile SDK での iOS アプリケーション拡張の使用](#)
- [Signposts を使用したプロファイリング](#)
- [iOS サンプルアプリケーション](#)

Salesforce Mobile SDK では、iOS でネイティブモバイルアプリケーションを開発するためのライブラリとサンプル Xcode プロジェクトが提供されます。ネイティブ iOS コードを記述するために、Mobile SDK は Objective-C および Swift をサポートしています。すべての新規および今後の開発には Swift をお勧めします。

forceios ユーティリティを使用してネイティブアプリケーションを作成すると、完全に機能するアプリケーションとしてプロジェクトが開始します。このアプリケーションでは、Salesforce 組織にログインして、使用中の Salesforce データを表示できます。新しく作成した forceios アプリケーションでは多くの操作はできませんが、基本的な SDK の機能が設計どおりに動作することを確認できます。

iOS ネイティブクイックスタート

1. **ネイティブ iOS の要件**を満たしていることを確認します。
2. 希望するスタイルでプロジェクトを作成します。

ウィザードスタイル (推奨)

forceios、CocoaPods、node.js、npm でプロジェクトを作成するには、次のことを実行します。

- node.js (npm を含む) をインストールします — <https://nodejs.org>
- CocoaPods の最新バージョンをインストールします — <https://www.cocoapods.org>。
- npm を使用して forceios をインストールします (npm は node.js と共に自動的にインストールされます)。

```
sudo npm install -g forceios
```

- ターミナルウィンドウで forceios を使用してアプリケーションを作成します。

```
forceios create
```

一部手動

forceios とサードパーティ連動関係を使わずに、Mobile SDK Swift テンプレートのファイル、ライブラリ、設定を Xcode テンプレートプロジェクトに追加する場合は、「[iOS Swift プロジェクトの手動作成](#)」のオプション 1 を参照してください。

全部手動

forceios とサードパーティ連動関係を使わずに、Xcode Swift テンプレートプロジェクトを Mobile SDK プロジェクトとして手動でコーディングし直す場合は、「[iOS Swift プロジェクトの手動作成](#)」のオプション 2 を参照してください。

 **メモ:** 設定およびインストールについての詳細は、Trailhead の「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」を参照してください。

iOS の基本的な要件

すべての Mobile SDK 開発には、次のソフトウェアが必要です。

- **接続アプリケーション**が存在する Salesforce [Developer Edition 組織](#)。

Mobile SDK 11.1 を使用した iOS の開発には、次のソフトウェアも必要です。

- iOS SDK:
 - リリース対象: iOS 15
 - ベース SDK: iOS 16
- Xcode バージョン: 14 以降。(最新バージョンを推奨)。

 **メモ:** Mobile SDK for iOS では、Cocoa Touch 動的フレームワークがサポートされます。

関連トピック:

[iOS の準備](#)

[CocoaPods と Mobile SDK の使用](#)

[Mobile SDK Pod の更新](#)

forceios を使用した iOS プロジェクトの作成

アプリケーションを作成するには、ターミナルウィンドウで forceios を使用します。forceios ユーティリティでは、2つの方法でアプリケーションを作成できます。

- 目的のアプリケーションの種別と基本設定データを指定する。
または
- 既存の Mobile SDK アプリケーションをテンプレートとして使用する。この場合も基本設定データを指定します。

コマンドラインプロンプトを使用する対話型モード、またはコマンドライン引数を使用するスクリプトモードで forceios を使用できます。コマンドの使用方法に関する情報を表示するには、引数を指定せずに「forceios」と入力します。

 **メモ:** 必ず forceios を使用する前に CocoaPods をインストールしてください。「iOS の準備」を参照してください。

forceios のアプリケーション種別

forceios create コマンドは、アプリケーション種別について次の入力値のいずれかを受け入れます。

アプリケーション種別	言語
native_swift と入力 (または RETURN キーを押す)	Swift
native と入力	Objective-C

対話形式での forceios create の使用

対話形式の入力メッセージを使用してアプリケーションを作成するには、ターミナルウィンドウを開いて、「forceios create」と入力します。次に例を示します。

```
$ forceios create
Enter your application type (native, native_swift): <press RETURN>
Enter your application name: testSwift
Enter the package name for your app (com.mycompany.myapp): com.bestapps.ios
Enter your organization name (Acme, Inc.): BestApps.com
Enter output directory for your app (leave empty for the current directory): testSwift
```

このコマンドでは、現在のディレクトリの `testSwift/` サブディレクトリに「testSwift」という名前のネイティブ iOS Swift アプリケーションを作成します。

スクリプトモードでの `forceios create` の使用

スクリプトモードでは、対話形式の入力メッセージなしで `forceios` を使用できます。たとえば、Swift で記述されたネイティブアプリケーションを作成する場合、次のようになります。

```
$ forceios create --apptype="native_swift" --appname="package-test"
--packagename="com.acme.mobile_apps"
--organization="Acme Widgets, Inc." --outputdir="PackageTest"
```

または、Objective-C で記述されたネイティブアプリケーションを作成する場合、次のようになります。

```
$ forceios create --apptype="native" --appname="package-test"
--packagename="com.acme.mobile_apps"
--organization="Acme Widgets, Inc." --outputdir="PackageTest"
```

各コールは「package-test」というネイティブアプリケーションを作成し、現在のディレクトリの `PackageTest/` サブディレクトリ内に配置します。

テンプレートからのアプリケーションの作成

`forceios createWithTemplate` コマンドは、アプリケーション種別ではなく GitHub リポジトリの URI が求められる点を除いて `forceios create` と同じです。この URL を、テンプレートとして使用できる Mobile SDK アプリケーションを含む任意のリポジトリディレクトリを参照するように設定します。テンプレートアプリケーションには、サポートされている任意の Mobile SDK アプリケーション種別を使用できます。スクリプトは、他のパラメーターに指定された値に一致するようにテンプレートの識別子と設定を変更します。

`createWithTemplate` を使用する前に、使用可能なテンプレートを把握しておくのと役立ちます。見つけるには、「`forceios listtemplates`」と入力します。このコマンドにより、Mobile SDK によって提供されるテンプレートのリストが出力されます。各リストには、テンプレートおよびその GitHub URI の簡単な説明が含まれます。次に例を示します。

Available templates:

1) Swift application using MobileSync, SwiftUI and Combine

`forceios createwithtemplate --templaterepour=iOSNativeSwiftTemplate`

2) Swift application using MobileSync, SwiftUI and Combine (pulled in using Swift Package Manager)

`forceios createwithtemplate --templaterepour=iOSNativeSwiftPackageManagerTemplate`

3) Basic Swift application with notification service extension

`forceios createwithtemplate --templaterepour=iOSNativeSwiftEncryptedNotificationTemplate`

4) Basic Objective-C application

`forceios createwithtemplate --templaterepour=iOSNativeTemplate`

5) Sample Swift Identity Provider application

`forceios createwithtemplate --templaterepour=iOSIDPTemplate`

```
6) Sample Swift application using MobileSync data framework
forceios createwithtemplate --templaterepouri=MobileSyncExplorerSwift
```

テンプレートの URI を見つけたら、forceios コマンドラインにその URI を挿入できます。forceios createWithTemplate のコマンドラインでの使用方法に関する情報は、次のようになります。

```
Usage:
forceios createWithTemplate
  --templaterepouri=<Template repo URI> (e.g., MobileSyncExplorerReactNative)]
  --appname=<Application Name>
  --packagename=<App Package Identifier> (e.g. com.mycompany.myapp)
  --organization=<Organization Name> (Your company's/organization's name)
  --outputdir=<Output directory> (Leave empty for current directory)]
```

SalesforceMobileSDK-Templates リポジトリ内のテンプレートでは、templaterepouri のパスを削除できます。テンプレート名だけで問題ありません。例を次に示します。

```
forceios createwithtemplate --templaterepouri=iOSNativeSwiftTemplate
```

対話形式またはスクリプトモードで forceios createWithTemplate を使用できます。たとえば、テンプレートの特定のタグを使用するスクリプトモードのコールは次のようになります。

```
forceios createWithTemplate
--templaterepouri=MobileSyncExplorerSwift
--appname=MyMobileSyncExplorer
--packagename=com.mycompany.react
--organization="Acme Software, Inc."
--outputdir=testWithTemplate
```

このコールでは、MobileSyncExplorerSwift サンプルアプリケーションと同じソースコードおよびリソースを使用するネイティブの Swift アプリケーションが作成されます。forceios は、現在のディレクトリの testWithTemplate/ サブディレクトリに新しいアプリケーションを配置します。また、プロジェクト全体でアプリケーション名が「MyMobileSyncExplorer」に変更されます。

forceios バージョンの確認

インストールした forceios のバージョンを確認するには、次のコマンドを実行します。

```
forceios version
```

新しい forceios プロジェクトの実行

forceios を使用して作成されたアプリケーションは、標準で実行準備が整っています。forceios が終了すると、コマンドラインからプロジェクトを開いて実行するための手順が表示されます。

必要に応じて、コマンドラインを終了し、Xcode で新しいプロジェクトを手動で開くこともできます。

1. Xcode で [File (ファイル)] > [Open (開く)] を選択します。
2. 指定した出力フォルダーに移動します。
3. ワークスペースファイル (<project_name>.xcworkspace) を開きます。

4. Xcode での作成が完了したら、[Run (実行)] ボタンをクリックします。

forceios スクリプトで新しいアプリケーションを生成する方法

- アプリケーションは CocoaPods または Swift Package Manager に基づいています。
- スクリプトは、実行時に GitHub リポジトリからテンプレートをダウンロードします。
- `forceios create` コマンドの場合、スクリプトは [SalesforceMobileSDK-Templates](#) GitHub リポジトリのデフォルトのテンプレートを使用します。
- CocoaPods に基づくテンプレートの場合、スクリプトは実行時に `npm` を使用して Mobile SDK ライブラリをダウンロードします。podfile は、`:path => node_modules/...` ディレクティブでこれらのライブラリを参照します。
- Swift Package Manager で構築されたプロジェクトの場合、スクリプトは <https://github.com/forcedotcom/SalesforceMobileSDK-iOS-SPM> で公開されている Salesforce Mobile SDK Swift Package を使用するように Xcode プロジェクトを設定します。

関連トピック:

[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)

カスタムテンプレートを使用したアプリケーションの作成

独自または他のユーザーのカスタムアプリケーションをテンプレートとして使用できたら良いですね。カスタムテンプレートにより、コードの再利用、迅速な開発、および内部の一貫性が促進されます。Mobile SDK 5.0 以降、`forceios` または `forcedroid` を使用して、カスタムテンプレートでアプリケーションを作成できます。Mobile SDK アプリケーションをテンプレートにするには、いくつかの手順を実行して Mobile SDK で使用するアプリケーションのリポジトリを準備します。

Mobile SDK テンプレートの概要

Mobile SDK は、iOS および Android でサポートするアーキテクチャ別のテンプレートを定義します。これらのテンプレートは github.com/forcedotcom/SalesforceMobileSDK-Templates リポジトリに保持されます。顧客が `forcedroid` または `forceios` の `create` コマンドを実行すると、スクリプトが、リポジトリから適切な組み込みテンプレートをコピーし、このコピーを新しいアプリケーションに変換します。この方法で作成されたアプリケーションは、機能のほとんどない基本的な Mobile SDK アプリケーションです。

実際には、追加の機能、リソース、ブランド設定などを備えた独自のテンプレートを作成したいのではないのでしょうか。この同じ Mobile SDK メカニズムを活用して、独自のアプリケーションをテンプレートに変換することができます。そして、`forcedroid` または `forceios` に、その独自のテンプレートではなく、この変換後のテンプレートを使用するように指示できます。

カスタムテンプレートの使用方法

Mobile SDK は、`forcedroid` および `forceios` の `create` のほか、`createWithTemplate` コマンドも定義します。`forcedroid` または `forceios` の `createWithTemplate` を実行するときは、アプリケーション種別ではなく、テンプ

レートアプリケーションリポジトリを指定し、続いて残りのアプリケーション作成パラメーターを指定します。テンプレートアプリケーションリポジトリには、スクリプトがテンプレートと認識する Mobile SDK アプリケーションが保持されます。このテンプレートから新しい Mobile SDK アプリケーションを作成するために、スクリプトはテンプレートアプリケーションを新しいフォルダーにコピーして、コピーされたコードにパラメーター値を適用します。

template.js ファイル

不明なアプリケーションをテンプレートとして受け入れる場合、forceios および forcedroid で template.js 設定ファイルを定義する必要があります。このファイルを、テンプレートアプリケーションリポジトリのルートに保存します。このファイルはスクリプトに、標準アプリケーションのリファクタリングタスク (ファイルの移動、テキストの置換、リソースの削除および名前変更) の実行方法を指示します。他方、より広範な変更を適用したいと思うことがあります。そのような場合には、標準の範囲を超えるカスタマイズが実行されるように template.js を適応させることができます。たとえば、メインのエントリポイントクラス以外のクラスにアプリケーション名を挿入する場合は、template.js を使用してこれらの変更を実行できます。

template.js ファイルは2つの部分で構成されます。テンプレートから新しいアプリケーションを準備する JavaScript の「準備」関数と、エクスポートの宣言です。

template.js の準備関数

template.js ファイルの大半は「準備」関数で構成されます。デフォルトで、準備関数は次の署名を使用します。

```
function prepare(config, replaceInFiles, moveFile, removeFile)
```

この関数の名前は変更できますが、更新された名前をエクスポートのリストに忘れずに指定する必要があります。Mobile SDK スクリプトは、次の引数を使用してエクスポートする関数をコールします。

- config: プラットフォーム (iOS または Android)、アプリケーション名、パッケージ名、組織、Mobile SDK バージョンを識別する辞書。
- replaceInFiles: ファイル内の文字列を置換するヘルパー関数。
- moveFile: ファイルおよびディレクトリを移動するヘルパー関数。
- removeFile: ファイルおよびディレクトリを削除するヘルパー関数。

Mobile SDK テンプレート内にあるデフォルトの準備関数は、標準テンプレートアプリケーションをパーソナライズするために必要な文字列の置換やファイルの移動および削除を行います。機能を追加する場合は、コードをこの準備関数内に配置します。ただし、準備関数に渡されるヘルパー関数は、標準テンプレートアプリケーションのタスクのみを実行できます。カスタムタスクについては、独自のメソッドを実装してコールする必要があります。

template.js で定義されたエクスポート

各 template.js ファイルは、次の2つのエクスポートを定義します。

appType

次のいずれかの値を割り当てます。

- 'native'

- 'native_kotlin' (forcedroid のみ)
- 'native_swift' (forceios のみ)
- 'react_native'
- 'hybrid_local'
- 'hybrid_remote'

prepare

準備関数のハンドル (引用符なしでリスト)。

template.js ファイルのエクスポートセクションの例を次に示します。このテンプレートは、prepare という名前の準備関数を定義するネイティブアプリケーション用です。

```
//
// Exports
//
module.exports = {
  appType: 'native',
  prepare: prepare
};
```

この場合、準備関数のハンドルは「prepare」です。

```
function prepare(config, replaceInFiles, moveFile, removeFile)
```

template.js のテンプレートアプリケーションの識別 (ネイティブおよび React Native アプリケーション)

ネイティブおよび React Native アプリケーションの場合は、テンプレートアプリケーションの準備関数が、アプリケーション名、パッケージ名、および組織名または会社名を定義します。これらの値は、テンプレートから作成された新しいカスタムアプリケーションではなく、テンプレートアプリケーション自体を識別します。実行時に、Mobile SDK スクリプトがこれらの値を使用して、スクリプトの入力値に置換される文字列を見つけます。これらの iOSNativeTemplate テンプレートアプリケーションの設定の例を次に示します。

```
// Values in template
var templateAppName = 'iOSNativeTemplate';
var templatePackageName = 'com.salesforce.iosnativetemplate';
var templateOrganization = 'iOSNativeTemplateOrganizationName';
```

template.js ファイルの例

Mobile SDK は、github.com/forcedotcom/SalesforceMobileSDK-Templates リポジトリで独自のテンプレートを定義します。各テンプレートディレクトリには template.js ファイルが含まれます。テンプレートに含まれる具体的な内容は次のとおりです。

- iOSNativeTemplate (forceios のみ)
- iOSNativeSwiftTemplate (forceios のみ)
- ReactNativeTemplate
- HybridLocalTemplate
- HybridRemoteTemplate

- `AndroidNativeTemplate` (`forcedroid` のみ)
- `AndroidNativeKotlinTemplate` (`forcedroid` のみ)

これらのテンプレートは、`MobileSDKnpm` スクリプトがアプリケーションを作成するために使用する「ベアボーン」プロジェクトであるため、意図的に複雑性を低くしています。より高度なテンプレートをお探しの場合、次の項目を参照してください。

- `MobileSyncExplorerReactNative`
- `MobileSyncExplorerSwift`
- `AndroidIDPTemplate`
- `iOSIDPTemplate`

リポジトリパスを含めたこれらのテンプレートのリストは、`listtemplates` コマンドで取得できます。すべての `Mobile SDK npm` スクリプト — `forcedroid`、`forceios`、`forcehybrid`、`forcereact` — で、このコマンドがサポートされています。

 **メモ:** スクリプトのコマンドは常にテンプレートと一致させます。`forceios createWithTemplate` のみが設定された iOS 固有のテンプレートと、`forcedroid createWithTemplate` のみが設定された Android 固有のテンプレートを使用します。この制限はハイブリッドおよび React Native テンプレートには適用されません。

基本の `template.js` ファイルの定義

基本の `template.js` ファイルを最も迅速に作成する手順を次に示します。

1. github.com/forcedotcom/SalesforceMobileSDK-Templates リポジトリから `template.js` ファイルを、カスタムテンプレートアプリケーションリポジトリのルートにコピーします。必ずテンプレートで作成するアプリケーションの種別に一致するテンプレートを選択してください。
2. ネイティブまたは React Native アプリケーションのみについては、テンプレートアプリケーションを反映させるようにアプリケーション名、パッケージ名、および組織を更新します。
3. 必要に応じて、`module.exports` オブジェクトの `appType` および `prepare` 設定を前述のとおり更新します。この基本的な例ではこの手順は不要ですが、後々独自の `template.js` ファイルを作成する場合に必要になることがあります。

制限とガイドライン

カスタムテンプレートにはいくつかの制限が適用されます。

- テンプレートアプリケーションは、サポートされているすべてのプラットフォームおよびアーキテクチャを対象とする有効な Mobile SDK アプリケーションにすることができます。
- 主要な要件は、テンプレートリポジトリおよびローカルの Mobile SDK リポジトリを同じ Mobile SDK バージョンにする必要があることです。Git のバージョンタグを使用すると、両方のリポジトリを特定の旧バージョンに同期できますが、この処理はお勧めしません。
- スクリプトのコマンドは常にテンプレートと一致させます。`forceios createWithTemplate` のみが設定された iOS 固有のテンプレートと、`forcedroid createWithTemplate` のみが設定された Android 固有のテンプレートを使用します。この制限はハイブリッドおよび React Native テンプレートには適用されません。

iOS Swift プロジェクトの手動作成

forceios や CocoaPods の使用を希望しない場合は、Xcode で Mobile SDK アプリケーションを手動で作成できます。

❗ 重要: 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

次のチュートリアルで、ネイティブ iOS Swift プロジェクトを作成する 2 通りの方法を示します。

- Mobile SDK Swift テンプレートアプリケーションのソースファイルを転用する。
- Swift Package Manager を使用する。

手順は次の 4 つのセクションに分かれ、以下の順序で実行する必要があります。

1. [Xcode Swift プロジェクトの作成](#) (ページ 54)
2. [Mobile SDK ライブラリのプロジェクトへの追加](#) (ページ 55)
3. [プロジェクトのビルド設定の定義](#) (ページ 56)
4. オプションの選択:
 - [オプション 1: Mobile SDK のテンプレートファイルをインポートする](#) (ページ 58)
 - [オプション 2: Mobile SDK の設定コードを手動で追加する](#) (ページ 60)

どのセクションも Xcode 12.4 と Xcode 13 をサポートしています。

このセクションの内容:

[Xcode Swift プロジェクトの作成](#)

[Mobile SDK ライブラリのプロジェクトへの追加](#)

[プロジェクトのビルド設定の定義](#)

[オプション 1: Mobile SDK のテンプレートファイルをインポートする](#)

このオプションでは、Xcode Swift テンプレートのファイルを Mobile SDK ネイティブ Swift テンプレートのファイルに置き換えます。このオプションを選択しない場合は、forceios でこの処理が実行されます。数回のドラッグアンドドロップ操作で、アプリケーションが Salesforce に接続され、組織データを表示できるようになります。

[オプション 2: Mobile SDK の設定コードを手動で追加する](#)

コード全体をゼロから自分で記述する場合は、Mobile SDK のテンプレートファイルをワークスペースにコピーせずにプロジェクトを作成できます。特定の機能については、テンプレートコードから定型の実装を取得することをお勧めします。

Xcode Swift プロジェクトの作成

❗ 重要: 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

手順の概要

Xcode では、Swift 言語、SwiftUI インターフェース、および UIKit App Delegate のライフサイクルを使用する App プロジェクトを作成します。

空の Xcode プロジェクトの作成 — 詳細

1. Xcode で [File (ファイル)] > [New (新規)] > [Project (プロジェクト)] を選択します。または、Xcode の [Welcome (ようこそ)] 画面から、[Create a new Xcode project (新規 Xcode プロジェクトの作成)] を選択します。
2. [iOS] タブで、[App (アプリ)] を選択し、[次へ] をクリックします。
3. 次の値を割り当てます。
 - **Product Name (製品名):** MyMobileSDKApp
 - **Team (チーム):** Apple 開発者アカウントに関連付けられたチーム
 - **Organization Identifier (組織 ID):** DNS 名を逆にしたもの (*com.acme.apps* など)
 - **Interface (インターフェース):** SwiftUI
 - **Life Cycle (ライフサイクル):** UIKit App Delegate (この手順は Xcode 13 ウィザードに含まれません。)
 - **Language (言語):** Swift
4. デフォルトの [Use Core Data (コアデータを使用)] と [Include Tests (テストを含める)] をそのまま使用して、[次へ] をクリックします。
5. プロジェクトのディレクトリを選択し、[Create (作成)] をクリックします。
6. Xcode で、有効なスキームに最新の iPhone モデルを選択し、[Run (実行)] をクリックします。アプリケーションに、「Hello, World!」以外は空白の白い画面が表示されます。
7. [Stop (停止)] をクリックします。

Mobile SDK ライブラリのプロジェクトへの追加

 **重要:** 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

前提条件

1. [Xcode Swift プロジェクトの作成](#) (ページ 54)

Mobile SDK Swift パッケージの追加

アプリケーションにパッケージ連動関係を追加する一般的な手順については、Apple のドキュメント「[Adding package dependencies to your app \(アプリケーションへのパッケージ連動関係の追加\)](#)」を参照してください。

Mobile SDK Swift パッケージを追加するには、Xcode で次の設定を使用します。

- パッケージリポジトリ URL: <https://github.com/forcedotcom/SalesforceMobileSDK-iOS-SPM>。
- バージョン: 11.0.1 (またはそれ以降のバージョン)。

プロジェクトのビルド設定の定義

❗ 重要: 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

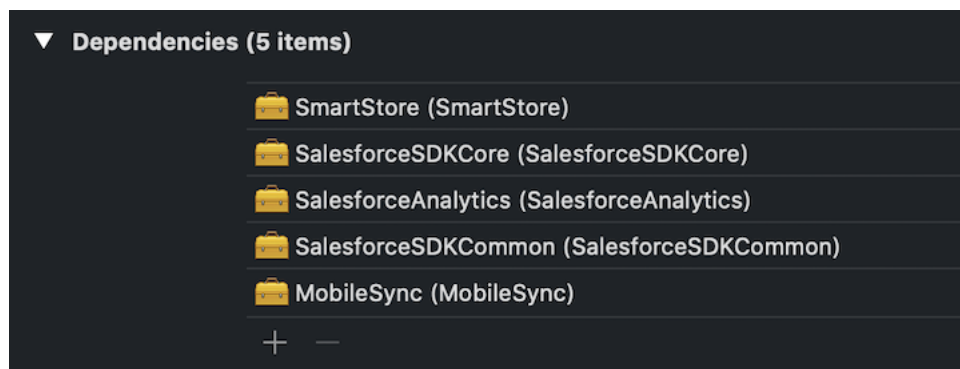
前提条件

1. [Xcode Swift プロジェクトの作成](#) (ページ 54)
2. [Mobile SDK ライブラリのプロジェクトへの追加](#) (ページ 55)

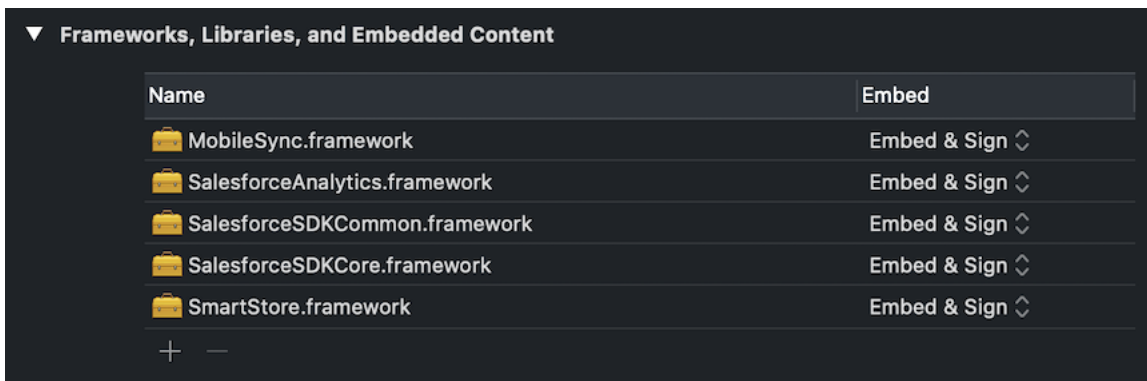
プロジェクトのビルド設定のカスタマイズ

1. プロジェクトナビゲーターで、最上位プロジェクトを選択します。
2. プロジェクトエディターで、[Build Phases (ビルドフェーズ)] を選択します。
3. [Dependencies (連動関係)] の下で [Add Items (項目を追加)] (+) をクリックします。
4. Command キーを押しながら、5 つの Mobile SDK ライブラリの金のブリーフケースアイコンをすべてクリックして複数選択し、[Add (追加)] をクリックします。

作業が終了すると、次のように表示されます。



5. プロジェクトエディターで、[General (全般)] を選択します。
6. [Frameworks, Binaries, and Embedded Content (フレームワーク、バイナリ、埋め込みコンテンツ)] までスクロールします。
7. [Add items (項目を追加)] (+) をクリックして、次のフレームワークを追加します。
 - `SalesforceSDKCommon.framework`
 - `SalesforceAnalytics.framework`
 - `SalesforceSDKCore.framework`
 - `SmartStore.framework`
 - `MobileSync.framework`



❗ 重要: 各フレームワークは必ず1回のみ追加するようにしてください。誤って重複して追加した場合は、余分なフレームワークを選択して [Remove items (項目を削除)] (-) をクリックします。

8. テストビルドを実行します。

メモ:

- このビルドの結果は、機能する Mobile SDK アプリケーションではありません。ここでは、ライブラリが適切にビルドしたかどうかを確認しています。アプリケーションを実行すると、今回も「Hello, World!」以外は空白の白い画面が表示されます。
- 画像ファイルがないという警告が表示された場合は、次の手順を実行します。
 - ターミナルで <選択したローカルパス>/MyMobileSDKApp/SalesforceMobileSDK-iOS/ に移動します。
 - `./install.sh` を実行します。

9. Xcode で、[Stop (停止)] をクリックします。

この時点で、各自のアプリケーションで Mobile SDK を選択可能になりますが、まだ各自のアプリケーションで使用することはできません。Mobile SDK を使用するには、SDK を初期化し、そのサービスを各自のアプリケーションのセッションに統合するコーディングが必要です。Mobile SDK の設定を終える場合、次の2つのオプションを選択できます。

オプション 1: Mobile SDK のテンプレートファイルをインポートする (ページ 58)

このオプションは簡便な方法です。プロジェクトで、特定の Xcode テンプレートファイルを削除し、iOS ネイティブ Swift テンプレートからファイルを追加します。これで完成です。各自のアプリケーションで Mobile SDK を使用できるようになりました。

オプション 2: Mobile SDK の設定コードを手動で追加する (ページ 60)

この実践的なコーディングオプションを選択した場合は、Mobile SDK のインテグレーションに必要なコードの変更を実行しやすくなります。また、その変更の過程で利便性が向上するカスタマイズに気付くこともあります。このオプションでは、テンプレートファイルをプロジェクトにインポートするよう指示されることがありません。ただし、iOS ネイティブ Swift テンプレートにある定型の機能をいくつかコピーしておく役立ちます。

オプション 1: Mobile SDK のテンプレートファイルをインポートする

このオプションでは、Xcode Swift テンプレートのファイルを Mobile SDK ネイティブ Swift テンプレートのファイルに置き換えます。このオプションを選択しない場合は、forceios でこの処理が実行されます。数回のドラッグアンドドロップ操作で、アプリケーションが Salesforce に接続され、組織データを表示できるようになります。

⚠ 重要: 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

前提条件

1. [Xcode Swift プロジェクトの作成](#) (ページ 54)
2. [Mobile SDK ライブラリのプロジェクトへの追加](#) (ページ 55)
3. [プロジェクトのビルド設定の定義](#) (ページ 56)

Mobile SDK コードアセットの追加

1. Xcode で、MyMobileSDKApp/MyMobileSDKApp プロジェクトフォルダーを開きます。
2. Assets.xcassets フォルダーと Preview Content フォルダーを除くすべてのファイルを選択します。
3. Control キーを押しながらクリックし、[Delete (削除)] を選択します。
4. [Move to Trash (ごみ箱に移動)] を選択します。
5. Finder で、SalesforceMobileSDK-Templates のコピーに移動し、/iOSNativeSwiftTemplate/iOSNativeSwiftTemplate/ サブフォルダーを展開します。
6. 次の Swift テンプレートファイルを Xcode のプロジェクトナビゲーターにドラッグします。プロジェクトの MyMobileSDKApp/MyMobileSDKApp/ フォルダーにドロップします。
 - AccountsListModel.swift
 - AccountsListView.swift
 - AppDelegate.swift
 - bootconfig.plist
 - Bridging-Header.h
 - ContactDetailModel.swift
 - ContactDetailsView.swift
 - ContactsForAccountListView.swift
 - ContactsForAccountModel.swift
 - Info.plist
 - InitialViewController.swift
 - main.swift
 - iOSNativeSwiftTemplate.entitlements
 - SceneDelegate.swift
 - userstore.json
 - usersyncs.json

7. 次の設定を選択します。

オプション	値
Destination (追加先):	Copy items if needed (必要に応じて項目をコピーする)
Added folders (追加されるフォルダー):	Create folder references (フォルダー参照を作成する)
Add to targets (ターゲットに追加):	MyMobileSDKApp

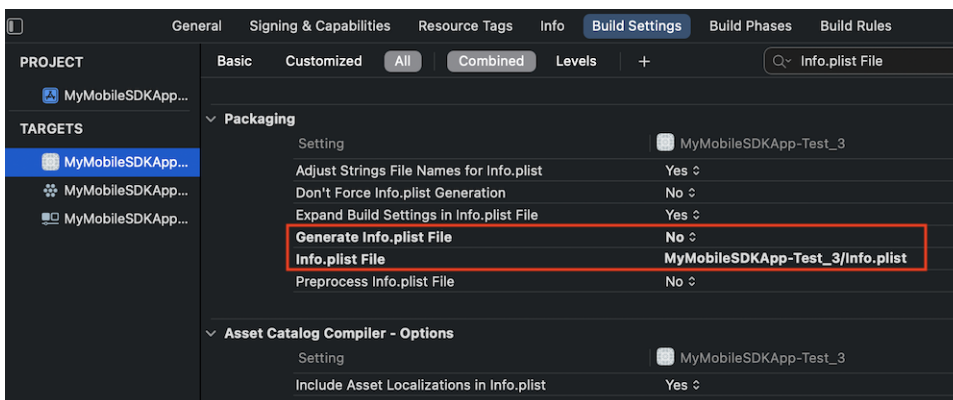
8. Objective-C ブリッジヘッダーを作成する許可を求められたら、[Don't Create (作成しない)] をクリックします。

9. [Project (プロジェクト)] 設定で、[Build Phases (ビルドフェーズ)] を選択します。

10. [Copy Bundle Resources (バンドルリソースのコピー)] で、Info.plist がある場合は、このファイルを選択して [Remove Items (項目を削除)] (-) をクリックします。

11. Xcode 13 のみ:

- [Build Settings (ビルド設定)] で、「Info.plist ファイル」を検索します。
- [Generate Info.plist File (Info.plist ファイルを生成)] を [No (いいえ)] に設定します。
- [Info.plist File (Info.plist ファイル)] を「MyMobileSDKApp/Info.plist」に設定します。



テンプレートファイルについての詳細は、「[アプリケーションフローの概要](#)」を参照してください。

新しい Mobile SDK アプリケーションのビルド

1. 最上位プロジェクトを選択します。
2. プロジェクトはビルドできる状態になっています。[Product (製品)] > [Run (実行)] を選択します。
3. Salesforce のログイン画面が表示されれば完了です。

新しいプロジェクトのカスタマイズが可能になります。独自のアセットを追加したり、AppDelegate.swift と SceneDelegate.swift クラスをカスタマイズしたり、Accounts と Contacts の機能を目的に合わせて使ったりできるようになります。

- 重要:** アプリケーションを App Store に公開する前に、`bootconfig.plist` ファイルの `remoteAccessConsumerKey` と `oauthRedirectURI` を自分の接続アプリケーションの設定に更新する必要があります。「[ネイティブ iOS 開発の開始](#)」を参照してください。

オプション 2: Mobile SDK の設定コードを手動で追加する

コード全体をゼロから自分で記述する場合は、Mobile SDK のテンプレートファイルをワークスペースにコピーせずにプロジェクトを作成できます。特定の機能については、テンプレートコードから定型の実装を取得することをお勧めします。

- 重要:** 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

ここに記載の手順では、AppDelegate - SceneDelegate フローを実装する基本的な Mobile SDK Swift アプリケーションを作成します。

テンプレートファイルをインポートせずに Mobile SDK を iOS プロジェクトに挿入する場合は、Mobile SDK のガイドラインに従って、アプリケーションのライフサイクルを各自の責任で再設定する必要があります。必須タスクの例として次のものが挙げられます。

- SalesforceApp をメインクラスとして設定する
- Mobile SDK を初期化する
- 顧客のログインのプロンプトの表示やユーザーの切り替え処理で Mobile SDK をコールする

省略可能ないくつかの機能については、Mobile SDK に特定のカスタマイズが用意され、コードにドロップできます。

さらに、開発上のニーズに合わせて、次のいずれかを柔軟に選択できます。

- Swift のコード (推奨)

または

Objective-C (古いアプリケーションをサポートするため。ここに記載の手順では説明しません)

- アプリケーションレベルの初期化には AppDelegate を使用し、すべての UI ライフサイクルイベントの処理には SceneDelegate を使用する。推奨の方法です。iPadOS で複数のウィンドウがサポートされ、マルチシーンの処理が明確になり、アプリケーションレベルとシーンレベルのコードが分離されます。

または

AppDelegate のみを使用して、アプリケーションの初期化と UI ライフサイクルイベントを処理する。この方法は、iPadOS で複数のウィンドウをサポートする必要がないシングルウィンドウアプリケーションのみに選択できます。この方法を選択した場合は、ここに記載の手順で SceneDelegate に追加されたすべての Mobile SDK コードの AppDelegate 内の格納場所を確保する必要があります。「[iOS ネイティブテンプレート](#)」で例を確認できます。

- メモ:** ここでは、実行可能な Swift アプリケーションで Mobile SDK を有効にするための最小限の手順について説明します。この手順の末尾に、プッシュ通知や IDP アプリケーションなど、省略可能な機能の設定に関する補足のセクションがあります。

前提条件

1. [Xcode Swift プロジェクトの作成](#) (ページ 54)
2. [Mobile SDK ライブラリのプロジェクトへの追加](#) (ページ 55)
3. [プロジェクトのビルド設定の定義](#) (ページ 56)

Info.plist の追加または編集

Mobile SDK では、プロジェクトのメインの Info.plist ファイルを使用して、デフォルトの Salesforce ログインホストを設定します。Xcode 12.4 の場合は、アプリケーションウィザードで Info.plist ファイルが自動的に作成されます。Xcode 13 の場合は、SwiftUI インターフェースのアプリケーションウィザードでこのファイルが作成されませんが、Mobile SDK テンプレートアプリケーションからコピーできます。

Xcode 12.4

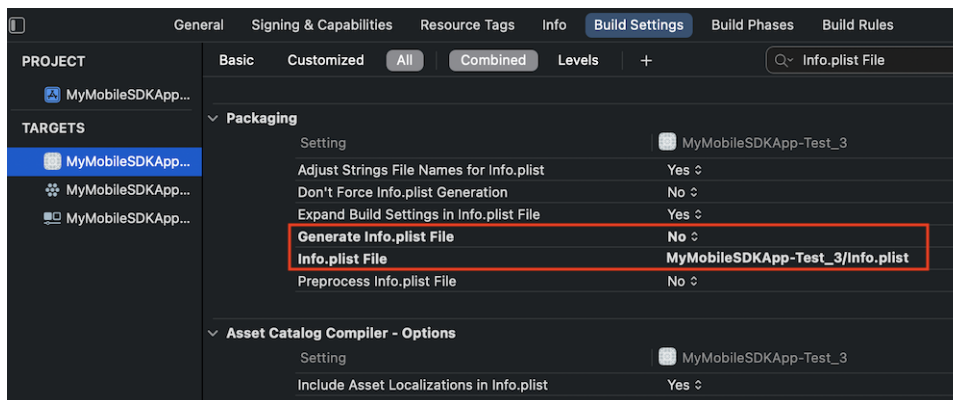
1. プロジェクトナビゲーターで、Ctrl キーを押しながら **MyMobileSDKApp > MyMobileSDKApp > Info.plist** をクリックし、**[Open As (名前を付けて開く)] > [Source Code (ソースコード)]** を選択します。
2. 次のキー - 値ペアを最上位ディクショナリに追加します。

```
<key>SFDCOAuthLoginHost</key>
<string>login.salesforce.com</string>
```

Xcode 13

Info.plist ファイルを [iOS ネイティブ Swift テンプレート](#) からプロジェクトにコピーします。ファイルを配置したら、プロジェクト設定を次のとおり調整します。

1. **[Project (プロジェクト)]** 設定で、**[Build Phases (ビルドフェーズ)]** を選択します。
2. **[Copy Bundle Resources (バンドルリソースのコピー)]** で、Info.plist がある場合は、このファイルを選択して **[Remove Items (項目を削除)]** (-) をクリックします。
3. **[Build Settings (ビルド設定)]** で、「Info.plist ファイル」を検索します。
4. **[Generate Info.plist File (Info.plist ファイルを生成)]** を **[No (いいえ)]** に設定します。
5. **[Info.plist File (Info.plist ファイル)]** をプロジェクトのファイルのパス (例: 「MyMobileSDKApp/Info.plist」) に設定します。



bootconfig.plist の追加

各自のアプリケーションが接続アプリケーションの正しい値を使用して開始できるように、Mobile SDK で bootconfig.plist ファイルが検索されます。このファイルは、Mobile SDK iOS テンプレートプロジェクトから各自のプロジェクトにコピーできますが、必ず値を各自のアプリケーションの設定にリセットしてください (特に remoteAccessConsumerKey と oauthRedirectURI)。

1. 外部のテキストエディターで、空のテキストファイルを作成します。
2. 次のテキストを新しいファイルにコピーします。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>remoteAccessConsumerKey</key>

<string>3MVG9Tu66FKeHhINkBl17xt7kR8czFcCTUhgoA80l2LtflEYHOU4SqQRSEitYFDUpqRWcoQ2.dBv_alDyu5xa</string>

    <key>oauthRedirectURI</key>
    <string>testsfdc:///mobilesdk/detect/oauth/done</string>
    <key>oauthScopes</key>
    <array>
        <string>web</string>
        <string>api</string>
    </array>
    <key>shouldAuthenticate</key>
    <true/>
</dict>
</plist>
```

3. MyMobileSDKApp プロジェクトのローカルディスクディレクトリにこのファイルを bootconfig.plist という名前で保存します。
4. Xcode のプロジェクトナビゲーターで、Ctrl キーを押しながら MyMobileSDKApp > MyMobileSDKApp をクリックし、[Add Files to "MyMobileSDKApp" (「MyMobileSDKApp」にファイルを追加)] を選択します。
5. bootconfig.plist を選択します。
6. 現時点 (接続アプリケーションを使用できる場合) またはアプリケーションをリリースする前に、次の手順を実行します。
 - a. Xcode エディターで [bootconfig.plist] を開きます。
 - b. remoteAccessConsumerKey 値を組織の接続アプリケーションのコンシューマー鍵に置き換えます。
 - c. oauthRedirectURI 値を組織の接続アプリケーションのコールバック URL に置き換えます。

main.swift ファイルの追加

Mobile SDK では、ユーザーイベント通知を監視するために、SFApplication という独自の UIApplication ベースのオブジェクトが必要です。iOS が最初にメインのアプリケーションオブジェクトを初期化するときに、このクラスをインスタンス化する必要があります。デフォルトの UIApplication クラスの代わりにこのクラ

スを使用する場合は、カスタムの `main.swift` ファイルをプロジェクトに追加します。追加後、カスタム引数を指定して `UIApplicationMain` をコールできます。

1. プロジェクトで、Ctrl キーを押しながら **MyMobileSDKApp > MyMobileSDKApp** をクリックし、[New File (新規ファイル)] を選択します。
2. [Swift File (Swift ファイル)] を選択し、「main.swift」という名前を付けて [Create (作成)] をクリックします。Xcode で、次の 1 行のファイルが作成されます。

```
import Foundation
```

3. `SalesforceSDKCore` をインポートします。

```
import Foundation
import SalesforceSDKCore
```

4. `UIApplicationMain` のインスタンスを作成し、次の値を渡します。

- `principalClassName` 引数の「SFApplication」
- `delegateClassName` 引数の「AppDelegate」

```
UIApplicationMain(
    CommandLine.argc,
    CommandLine.unsafeArgv,
    NSStringFromClass(SFApplication.self), //principalClassName
    NSStringFromClass(AppDelegate.self)   //delegateClassName
)
```

Apple 開発者ドキュメントの「[UIApplicationMain\(_::_:\)](#)」を参照してください。

AppDelegate クラスのカスタマイズまたは作成

Xcode 12.4 アプリケーションウィザードに UIKit App Delegate ライフサイクルの作成を指示すると、AppDelegate クラスと SceneDelegate クラスが作成されます。Xcode 13 アプリケーションウィザードにはライフサイクルの選択肢がなく、AppDelegate クラスが作成されません。ただし、次の手順に従って、基本的な最小バージョンをゼロから作成することができます。

1. Xcode 12.4: プロジェクトの AppDelegate から開始し、ライフサイクルの 3 つの標準機能を保持します。

Xcode 13:

- a. プロジェクトの **MyMobileSDKApp > MyMobileSDKApp** フォルダーで、Ctrl キーを押しながら `MyMobileSDKAppApp.swift` ファイルをクリックし、[Delete (削除)] をクリックします。
- b. [Move to Trash (ごみ箱に移動)] をクリックします。
- c. Ctrl キーを押しながら **MyMobileSDKApp > MyMobileSDKApp** をクリックし、[New File (新規ファイル)] を選択します。
- d. [Swift File (Swift ファイル)] を選択し、「AppDelegate」という名前を付けて [Create (作成)] をクリックします。Xcode で、次の 1 行のファイルが作成されます。

```
import Foundation
```

2. AppDelegate ライフサイクルメソッドの次の定型の実装を貼り付けます。

```
import Foundation

@main
class AppDelegate : UIResponder, UIApplicationDelegate {
    var window: UIWindow?

    // MARK: UISceneSession Lifecycle
    func application(_ application: UIApplication, configurationForConnecting
        connectingSceneSession: UISceneSession,
        options: UIScene.ConnectionOptions) -> UISceneConfiguration {
        // Called when a new scene session is being created.
        // Use this method to select a configuration to create the new scene with.
        return UISceneConfiguration(name: "Default Configuration",
            sessionRole: connectingSceneSession.role)
    }

    func application(_ application: UIApplication, didDiscardSceneSessions
        sceneSessions: Set<UISceneSession>) {
        // Called when the user discards a scene session.
        // If any sessions were discarded while the application was not running,
        // this will be called shortly after
        // application:didFinishLaunchingWithOptions.
        // Use this method to release any resources that were specific to the
        // discarded scenes, as they will not return.
    }

    // MARK: - App delegate lifecycle
    func application(_ application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // If you wish to register for push notifications, uncomment the line below.
        // Note that, if you want to receive push notifications from Salesforce,
        // you will also need to implement the
        // application(application, didRegisterForRemoteNotificationsWithDeviceToken)
        // method (below).

        // self.registerForRemotePushNotifications()
        return true
    }
}
```

3. AppDelegate クラス宣言の上にある @main 属性を削除します。この main.swift ファイルがメインの役割を引き受けます。
4. Mobile SDK のすべてのモジュールをインポートします。

```
import Foundation
import MobileSync
```


5. クラスの先頭で、Mobile SDK を初期化する `init()` メソッドを上書きします。たとえば、Mobile SDK のすべてのライブラリを初期化するには、`MobileSyncSDKManager` で `initializeSDK()` をコールします。

```
override init() {  
    super.init()  
    MobileSyncSDKManager.initializeSDK()  
}
```

InitialViewController クラスの追加

Salesforce データにアクセスするには、アプリケーションのユーザーが Salesforce 組織で認証される必要があります。Salesforce から各自のアプリケーションにログイン画面が送信され、ユーザーのログイン情報を収集すると、アプリケーションに OAuth トークンが返されます。

アプリケーションに、Salesforce ログイン画面のホストを唯一の目的とするビューコントローラーが必要です。顧客が誰もログインしていないときは、アプリケーションがこのビューコントローラーをルートビューとして設定します。従来、Mobile SDK アプリケーションでは、このビューコントローラークラスに `InitialViewController` という名前が付けられていました。このクラスに必要なのはスタブ実装のみです。

1. プロジェクトで、Ctrl キーを押しながら `MyMobileSDKApp > MyMobileSDKApp` をクリックし、[New File (新規ファイル)] を選択します。
2. [Swift File (Swift ファイル)] を選択し、「InitialViewController」という名前を付けて [Create (作成)] をクリックします。
3. 次の本体を追加します。

```
import Foundation  
import UIKit  
  
class InitialViewController: UIViewController {  
}
```

SceneDelegate クラスのカスタマイズまたは作成

`SceneDelegate` クラスを使用して、Salesforce ログイン、ユーザーの切り替え、ルートビューのインスタンス化を処理します。ログインが必要になるのは、ログインしているユーザーが1人もいない場合、つまり、アプリケーションが最初に起動されたときや、現在のユーザーがログアウトしたときです。ユーザーの変更は、以前にログインして認証されたままのユーザー間で行われます。Mobile SDK の `AuthHelper` 関数を使用すると、こうしたケースを簡単に処理できます。

- `AuthHelper.loginIfRequired(_:)` は、ログインが必要かどうかを判断します。
- `AuthHelper.registerBlock(forCurrentUserChangeNotifications:)` は、ユーザーの変更イベントをリスンします。

こうした関数を使用して、ログインまたは変更が生じた後の処理を決定できます。通常は、アプリケーションのメインビューをルートビューとして設定するという方法で応答します。

こうした `AuthHelper` コールはそれぞれ、シーンのライフサイクルの特定の時点での使用に適しています。たとえば、シーンが最初に初期化されたときに、ユーザー変更を1回のみ処理するブロックを登録します。ま

た、シーンがフォアグラウンドになるたびに `AuthHelper.loginIfRequired(_:_:)` をコールします。現在のユーザーがログインしたままの場合は、Mobile SDK で完了ブロックのみが実行されます。

アプリケーションの起動時に、`InitialViewController` を最初のルートビューコントローラーとして設定します。このビューは Salesforce ログイン画面をホストし、ログインに成功したら、アプリケーションのルートビューコントローラーに置き換えられます。実行時に、現在のユーザーがログアウトしたら、現在のルートビューコントローラーを閉じ、`InitialViewController` に置き換えます。どちらの場合も、新しいユーザーがログインしたら、`InitialViewController` を閉じ、アプリケーションの最初のビューをルートビューコントローラーとして設定します。

Xcode 13 のみ: Xcode 13 では、SwiftUI インターフェースで `SceneDelegate` が使用されなくなりました。ただし、iOS では引き続きサポートされ、Mobile SDK でも現在使用されています。この作成手順は次のとおりです。

1. プロジェクトで、Ctrl キーを押しながら **MyMobileSDKApp > MyMobileSDKApp** をクリックし、[New File (新規ファイル)] を選択します。
2. [Swift File (Swift ファイル)] を選択し、「SceneDelegate」という名前を付けて [Create (作成)] をクリックします。
3. 次の実装を新しいファイルにコピーします。

```
import UIKit
import MobileSync
import SwiftUI

class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    var window: UIWindow?

    func scene(_ scene: UIScene, willConnectTo session: UISceneSession,
              options connectionOptions: UIScene.ConnectionOptions) {
        // Use this method to optionally configure and attach the
        // UIWindow `window` to the provided UIWindowScene `scene`.
        // If using a storyboard, the `window` property will automatically
        // be initialized and attached to the scene. This delegate does not
        // imply the connecting scene or session are new
        // (see `application:configurationForConnectingSceneSession` instead).

        // Create the SwiftUI view that provides the window contents.
        let contentView = ContentView()

        // Use a UIHostingController as window root view controller.
        if let windowScene = scene as? UIWindowScene {
            let window = UIWindow(windowScene: windowScene)
            window.rootViewController = UIHostingController(rootView: contentView)
            self.window = window
            window.makeKeyAndVisible()
        }
    }

    func sceneDidDisconnect(_ scene: UIScene) {
        // Called as the scene is being released by the system.
        // This occurs shortly after the scene enters the background,
        // or when its session is discarded.
```

```

        // Release any resources associated with this scene that can be re-created
        // the next time the scene connects.
        // The scene may re-connect later, as its session was not necessarily
        // discarded (see `application:didDiscardSceneSessions` instead).
    }

    func sceneDidBecomeActive(_ scene: UIScene) {
        // Called when the scene has moved from an inactive state to an active state.
        // Use this method to restart any tasks that were paused (or not yet started)
        // when the scene was inactive.
    }

    func sceneWillResignActive(_ scene: UIScene) {
        // Called when the scene will move from an active state to an inactive state.
        // This may occur due to temporary interruptions (ex. an incoming phone call).
    }

    func sceneWillEnterForeground(_ scene: UIScene) {
        // Called as the scene transitions from the background to the foreground.
        // Use this method to undo the changes made on entering the background.
    }

    func sceneDidEnterBackground(_ scene: UIScene) {
        // Called as the scene transitions from the foreground to the background.
        // Use this method to save data, release shared resources, and store enough
        // scene-specific state information
        // to restore the scene back to its current state.
    }
}

```

Xcode 12.4 および Xcode 13:

1. プロジェクトの `SceneDelegate` クラスの先頭で、`UIWindow` 変数を宣言していることを確認します。

```

class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    var window: UIWindow?

```

2. `SceneDelegate` クラス本体の下部に、エスケープクロージャを唯一のパラメーターとして取る `resetViewState(__:)` というプライベート関数を実装します。この関数は現在のルートビューコントローラーを閉じ、指定されたクロージャーを実行します。

```

func resetViewState(_ postResetBlock: @escaping () -> ()) {
    if let rootViewController = self.window?.rootViewController {
        if let _ = rootViewController.presentedViewController {
            rootViewController.dismiss(animated: false,
                                       completion: postResetBlock)
            return
        }
        self.window?.rootViewController = nil
    }
    postResetBlock()
}

```

3. アプリケーションのカスタムビューを起動する `setupRootViewController()` というプライベート関数を実装します。この簡単な演習では、ルートビューを、Xcodeテンプレートで提供される `SwiftUI ContentView` に設定できます。

```
func setupRootViewController() {
    self.window?.rootViewController = UIHostingController(rootView:
        ContentView())
}
```

 **メモ:** Xcode テンプレートの `ContentView` は、追加の設定やクリーンアップが必要ありません。複雑な例については、[MobileSyncExplorerSwift](#) テンプレートの `setupRootViewController(_:)` を参照してください。

4. `scene(_:willConnectTo:options)` 関数で、ウィンドウシーンを設定するコードを指定します。既存のコードをすべて次の行に置き換えます。

```
func scene(_ scene: UIScene, willConnectTo session: UISceneSession,
    options connectionOptions: UIScene.ConnectionOptions) {

    guard let windowScene = (scene as? UIWindowScene) else { return }
    self.window = UIWindow(frame: windowScene.coordinateSpace.bounds)
    self.window?.windowScene = windowScene
}
```

5. ユーザー変更イベントのコールバックブロックを登録します。追加したプライベート関数を使用して、ビューステートをリセットし、アプリケーションのルートビューを表示するクロージャールを指定します。

```
func scene(_ scene: UIScene, willConnectTo session: UISceneSession,
    options connectionOptions: UIScene.ConnectionOptions) {

    guard let windowScene = (scene as? UIWindowScene) else { return }
    self.window = UIWindow(frame: windowScene.coordinateSpace.bounds)
    self.window?.windowScene = windowScene

    // Define a response to user change events
    AuthHelper.registerBlock(forCurrentUserChangeNotifications: {
        self.resetViewState {
            self.setupRootViewController()
        }
    })
}
```

6. `InitialViewController` をルートビューとして設定する `initializeAppViewState()` というプライベート関数を実装します。この関数はまず、アプリケーションがメインの UI スレッドにあることを確認します。アプリケーションが他のスレッドにある場合は、メインスレッドに切り替えて、その関数自体を再帰的にコールします。

```
func initializeAppViewState() {
    if (!Thread.isMainThread) {
        DispatchQueue.main.async {
            self.initializeAppViewState()
        }
    }
}
```

```

    }
    return
}

self.window?.rootViewController =
    InitialViewController(nibName: nil, bundle: nil)
self.window?.makeKeyAndVisible()
}

```

7. `sceneWillEnterForeground(_:)` リスナー関数で次の操作を実行します。

- a. ルートビューを `InitialViewController` に設定するには、`initializeAppViewState()` をコールします。
- b. ルートビューコントローラーをアプリケーションのメインビューに設定するクロージャールを指定して、`AuthHelper.loginIfRequired(_:)` をコールします。

```

func sceneWillEnterForeground(_ scene: UIScene) {
    // Called as the scene transitions from the background to the foreground.
    // Use this method to undo the changes made on entering the background.
    self.initializeAppViewState()
    AuthHelper.loginIfRequired {
        self.setupRootViewController()
    }
}

```

新しい Mobile SDK アプリケーションのビルド

1. Xcode で、[Product (製品)] > [Run (実行)] (R) を選択します。
2. Salesforce のログイン画面が表示されれば完了です。

新しいプロジェクトのカスタマイズが可能になります。これで、独自のアセットを追加し、シーンフローを設定できるようになりました。以下に記載の方法で、省略可能なプッシュ通知や IDP 機能を追加することもできます。

❗ 重要: もう 1 つ注意点があります。アプリケーションを App Store に公開する前に、`bootconfig.plist` ファイルの `remoteAccessConsumerKey` と `oauthRedirectURI` を自分の接続アプリケーションの設定に更新する必要があります。「[ネイティブ iOS 開発の開始](#)」を参照してください。

プッシュ通知のサポートの追加

1. Salesforce プッシュ通知をサポートするには、プッシュ通知を登録して処理するコードを追加します。[Mobile SDK の iOSNativeSwiftTemplate アプリケーション](#)の `AppDelegate` クラスから、次のメソッドの定型の実装をコピーできます。

- `registerForRemotePushNotifications()`
- `application(_:didRegisterForRemoteNotificationsWithDeviceToken:)` — 実装コードの 1 行をコメント解除してください!
- `didRegisterForRemoteNotifications(_:)`
- `application(_:didFailToRegisterForRemoteNotificationsWithError:)`

2. Salesforce では通知に暗号化が使用されます。そのため、復号化拡張を実装し、`UNUserNotificationCenter` オブジェクトを使用して承認を要求する必要があります。「[コードの変更 \(iOS\)](#)」(ページ 449)を参照してください。

ID プロバイダーのサポートの追加

Mobile SDK では ID プロバイダー (IDP) アプリケーションの設定がサポートされています。「[ID プロバイダーアプリケーション](#)」(ページ 497)を参照してください。IDP 機能を実装している場合は、`AppDelegate` と `SceneDelegate` をカスタマイズして有効にします。

1. `AppDelegate` クラスで、[Mobile SDK の iOSNativeSwiftTemplate アプリケーション](#)の `AppDelegate` クラスから、次の定型の関数を追加します。
 - `application(_:open:_:)`
 - `enableIDPLoginFlowForURL(_:_:)`
2. `SceneDelegate` クラスで、次の関数を追加します。

```
func scene(_ scene: UIScene, openURLContexts URLContexts: Set<UIOpenURLContext>) {
    if let urlContext = URLContexts.first {
        self.enableIDPLoginFlowForURLContext(urlContext, scene: scene)
    }
}

func enableIDPLoginFlowForURLContext(_ urlContext: UIOpenURLContext,
    scene: UIScene) -> Bool {
    return UserAccountManager.shared.handleIdentityProviderResponse(
        from: urlContext.url, with: [UserAccountManager.IDPSceneKey:
            scene.session.persistentIdentifier])
}
```

ログイン前画面のカスタマイズ

`InitialViewController` クラスを使用すると、ログイン画面が表示される前のログインビューのデザインをカスタマイズできます。以下は、デフォルトの Salesforce 背景色を使用して、ログイン前画面にアプリケーション名を表示してブランド設定する場合の例です。このクラスを使用して再作成可能なリソースを作成する場合は、このリソースを破棄する `didReceiveMemoryWarning()` オーバーライドを必ず実装してください。Xcode の「[UIViewController](#)」ドキュメントを参照してください。

```
import Foundation
import UIKit
import SalesforceSDKCore.UIColor_SFColors

class InitialViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        self.view.backgroundColor = UIColor.salesforceSystemBackground
        let label = UILabel()
        label.translatesAutoresizingMaskIntoConstraints = false
        guard let info = Bundle.main.infoDictionary,
```

```

        let name = info[kCFBundleNameKey as String] else { return }
        label.font = UIFont.systemFont(ofSize: 29)
        label.textColor = UIColor.black
        label.text = name as? String
        self.view.addSubview(label)
        label.centerXAnchor.constraint(equalTo: self.view.centerXAnchor).
            isActive = true
        label.centerYAnchor.constraint(equalTo: self.view.centerYAnchor).
            isActive = true
        // Do any additional setup after loading the view,
        // typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}

```

SmartStore と Mobile Sync の設定ファイルの追加

オフラインの設定ファイルを使用すれば、コードではなく、静的な XML ファイルで SmartStore と Mobile Sync の設定を定義できます。これらのファイルに `userstore.json` および `usersyncs.json` という名前を付け、Xcode プロジェクトで参照されるようにする必要があります。次のメソッドをコールすれば、これらのファイルを読み込むことができます。

```

// Setup store based on config userstore.json
MobileSyncSDKManager.shared.setupUserStoreFromDefaultConfig()
// Setup syncs based on config usersyncs.json
MobileSyncSDKManager.shared.setupUserSyncsFromDefaultConfig()

```

このファイルの主な考慮事項は、必ず各セッションで1回のみ読み込むようにすることです。たとえば、SceneDelegate フローで、次のいずれかに該当する場合に設定ファイルを読み込みます。

- Salesforce ログインが必要な場合
- ユーザーの変更が生じた場合

Mobile SDK ネイティブ Swift テンプレートを使用すれば、ルートビューコントローラーを設定するたびにこの2つの設定関数がコールされ、処理が簡素化されます。このタスクは `setupRootViewController()` というプライベート関数で処理され、次の場所でコールされます。

- `sceneWillEnterForeground(_:)` - Salesforce ログインの発生後に実行されるブロックでコールされます。
- `scene(_:willConnectTo:options:)` - 現在のユーザーの変更通知を処理するブロックでコールされます。

Mobile SDK では、ログインやユーザー変更イベントの内部の可変要素を調整する `AuthHelper` ユーティリティクラスを定義します。この例の全容については、[Mobile Sync Explorer Swift テンプレートの SceneDelegate クラス](#)を参照してください。

- 📌 **メモ:** アプリケーションでログインが延期された場合、`AuthHelper.loginIfRequired(_:)` をいつ、どこでコールするかは各自が決定できます。いずれの場合も、次に示すとおり、ルートビューコントローラーを `IntialViewController` に設定して、必ずコールの前に配置します。

CocoaPods と Mobile SDK の使用

CocoaPods には、Mobile SDK モジュールを既存の Xcode プロジェクトにマージするための便利なメカニズムが備えられています。この記事の手順に従って、MobileSDKiOS アプリケーションの CocoaPods を手動で設定します。forceios 4.0 以降を使用してアプリケーションを作成した場合、CocoaPods は自動的に設定されます。この場合、この記事の手順を実行する必要はありません。CocoaPods ソフトウェアをインストールするだけで、forceios が残りを設定します。他の方法でアプリケーションを作成する場合、この記事を使用して、CocoaPods を介して Mobile SDK モジュールをインポートします。

Mobile SDK 4.0 以降、forceios ではプロジェクトの作成に CocoaPods が使用されます。開発者が CocoaPods を手動で使用して Mobile SDK を既存の iOS アプリケーションに追加することもできます。

CocoaPods に慣れていない方は、www.cocoapods.org にあるドキュメントをまずお読みください。

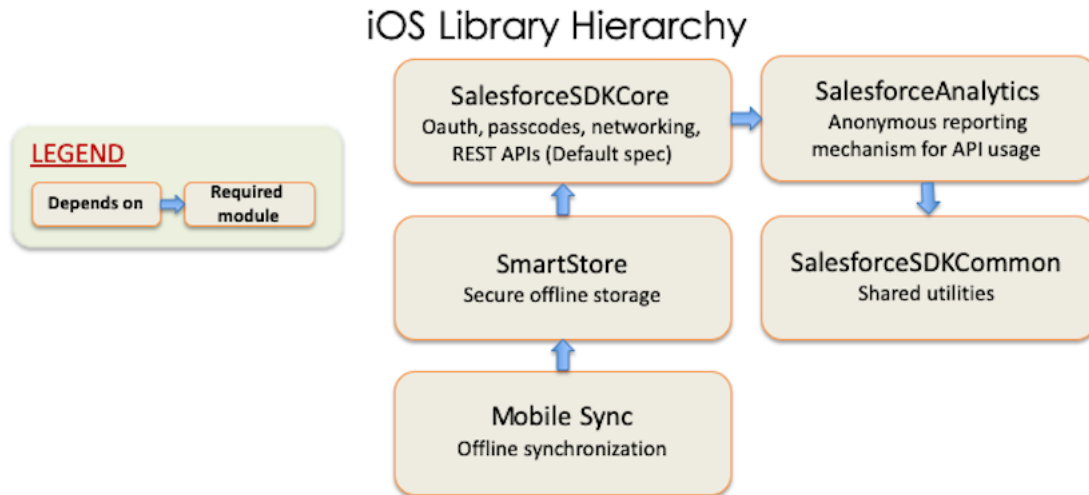
 **メモ:** forceios npm は必須ではないけれど便利な機能として提供されています。CocoaPods、node.js、npm は forceios には必須ですが、Mobile SDK iOS 開発には必須ではありません。forceios を使用せずに Mobile SDK iOS ネイティブプロジェクトを作成する方法については、「[iOS Swift プロジェクトの手動作成](#)」を参照してください。

Mobile SDK には、Mobile SDK モジュールごとに CocoaPods pod の仕様 (*podspecs*) が用意されています。

- SalesforceSDKCore — OAuth、パスコード、ネットワーク、REST API を実装します。
- SmartStore — セキュアなオフラインストレージを実装します。SalesforceSDKCore と連動します。
- Mobile Sync — オフライン同期を実装します。SmartStore と連動します。
- SalesforceAnalytics — Mobile SDK 機能の利用状況と人気度に関する Salesforce 匿名統計情報を送信するレポートメカニズムを実装します。
- SalesforceSDKCommon — SDK 全体で共有されるユーティリティです。

Mobile SDK は、CocoaPods を介して FMDB (外部モジュール) も使用します。

次の図は、仕様間の連動関係を示しています。この図では、連動する側の仕様から連動される側に向かって矢印が伸びています。



pod を宣言すると、その pod の連動関係チェーンのすべてを自動的に取得します。たとえば、Mobile Sync の pod を宣言すると、SmartStore pod と SalesforceSDKCore pod を自動的に取得します。このショートカットは、本番 pod にのみ適用されます。

github.com/forcedotcom/SalesforceMobileSDK-iOS-Specs リポジトリのすべてのバージョンの Mobile SDK podspec にアクセスできます。github.com/forcedotcom/SalesforceMobileSDK-iOS リポジトリから現在のバージョンを取得することもできます。

CocoaPods を通じて現在の Mobile SDK リリースを使用する手順は、次のとおりです。

1. [www.cocoapods.org](https://cocoapods.org) の説明に従って cocoapods Rubygem がインストールされていることを確認します。Mobile SDK 11.1 は pod バージョン 1.8 ~ 最大値 (指定なし) に対応しています。
2. プロジェクトの Podfile で、SalesforceMobileSDK-iOS-Specs リポジトリをソースとして追加します。必ず最初にこのエントリを CocoaPods ソースパスより前に配置します。

```
target 'YourAppName' do
  source 'https://github.com/forcedotcom/SalesforceMobileSDK-iOS-Specs.git' # needs to be
  first
  source 'https://github.com/CocoaPods/Specs.git'
  ...
end
```

3. アプリケーションにマージする Mobile SDK podspec を参照します。たとえば、OAuth モジュールとパスコードモジュールをアプリケーションに追加する場合は、Podfile で SalesforceSDKCore pod を宣言します。次に例を示します。

```
target 'YourAppName' do
  source 'https://github.com/forcedotcom/SalesforceMobileSDK-iOS-Specs.git' # needs to be
  first
  source 'https://github.com/CocoaPods/Specs.git'

  pod 'SalesforceSDKCore'
end
```

この pod 設定は、Mobile SDK アプリケーションの最低要件です。

4. 他のモジュールを追加するには、SalesforceSDKCore を別の pod 宣言で置き換えます。たとえば、Mobile Sync を使用します。

```
target 'YourAppName' do
  source 'https://github.com/forcedotcom/SalesforceMobileSDK-iOS-Specs.git' # needs to be
  first
  source 'https://github.com/CocoaPods/Specs.git'

  pod 'MobileSync'
end
```

MobileSync pod は SmartStore と SalesforceSDKCore と連動するため、これらの pod を明示的に宣言する必要はありません。

5. (代替方法) Mobile SDK の次のリリースで作業する場合は、SalesforceMobileSDK-iOS リポジトリをコピーして、dev ブランチを確認し、そこからリソースを取り込みます。この場合は、リポジトリのパスを示すことができるように、リリース前の各連動関係を明示的に宣言する必要があります。連動関係の宣言を省略すると、CocoaPods には本番バージョンが読み込まれます。
 - a. 目的の commit で github.com/forcedotcom/SalesforceMobileSDK-iOS をローカルにコピーします。
 - b. ターミナルウィンドウで、`git checkout dev` を実行して開発ブランチに切り替えます。
 - c. コピーのルートディレクトリにある `./install.sh` を実行します。
 - d. Podfile の各 pod コールに、コピーを指し示す `:path` パラメーターを追加します。

次のコードは、ローカルコピーからリソースを取り込むように上記の例を再設定したものです。

```
target 'YourAppName' do
  source 'https://github.com/forcedotcom/SalesforceMobileSDK-iOS-Specs.git' # need to be
  first
  source 'https://github.com/CocoaPods/Specs.git'

  # Specify each pre-release pod
  pod 'SalesforceSDKCore', :path => '/<path-to-clone-of>/SalesforceMobileSDK-iOS/'
  pod 'SalesforceAnalytics', :path => '/<path-to-clone-of>/SalesforceMobileSDK-iOS/'
  pod 'SmartStore', :path => '/<path-to-clone-of>/SalesforceMobileSDK-iOS/'
  pod 'MobileSync', :path => '/<path-to-clone-of>/SalesforceMobileSDK-iOS/'
end
```

```
end
```

- ターミナルウィンドウで、プロジェクトディレクトリから `pod install` を実行します。CocoaPodsにより、要求された pod の連動関係のダウンロード、プロジェクトへのマージ、新しくマージされたプロジェクトを含むワークスペースの作成が行われます。

重要: CocoaPods の実行後は必ず `pod install` で作成されたワークスペースからプロジェクトにアクセスします。たとえば、`MyProject.xcodeproj` を開くのではなく、`MyProject.xcworkspace` を開きます。

- マージされたアプリケーションで MobileSDKAPI を使用する場合は、次の重要なヒントを考慮してください。
 - Objective-C アプリケーションでは、二重引用符ではなく山括弧(「<」および「>」)を使用して、ヘッダーファイルをインポートします。次に例を示します。

```
#import <SalesforceSDKCore/SFRestAPI.h>
```

- Swift アプリケーションでは、必ず Podfile で `use_frameworks!` を指定します。また、Swift ソースファイルで、ヘッダーファイルではなく、モジュールをインポートします。次に例を示します。

```
import SalesforceSDKCore
```

Mobile SDK Pod の更新

CocoaPods は、マシンにローカルに保存されているリポジトリにその pod をキャッシュします。pod リポジトリが `forceios` と非同期になった場合、手動で更新できます。

`forceios` でネイティブアプリケーションが作成されると、インストールされた pod とそれらのバージョンのリストが出力されます。次に例を示します。

```
Installing SalesforceSDKCore (8.0.0)
Installing SalesforceAnalytics (8.0.0)
Installing SmartStore (8.0.0)
Installing MobileSync (8.0.0)
```

次のコマンドを入力して、これらのバージョンと自分の `forceios` バージョンを比較できます。

```
forceios version
```

表示された pod バージョンが自分の `forceios` バージョンよりも古い場合、ターミナルウィンドウで次のコマンドを実行します。

```
pod repo remove forcedotcom
pod setup
```

設定が完了したら、`forceios create` を使用してアプリケーションを再作成します。

Mobile SDK での Carthage の使用 (サポート対象外)

Mobile SDK 9.0 以降では、Carthage は正式にサポートされなくなりました。

ネイティブ iOS アプリケーションの開発

ネイティブ iOS の Salesforce Mobile SDK には、Apple モバイルデバイスのアプリケーションを構築するために必要なツールが備えられています。SDK には、次の特性があります。

- Swift および Objective-C API
- Salesforce REST API コールを簡単にするラッパーメソッドを備えたネットワークライブラリ
- 完全に実装された OAuth ログインとパスコードのプロトコル
- ユーザーデータをオフラインで安全に管理するための SmartStore および Mobile Sync ライブラリ

ネイティブ iOS の Mobile SDK を使用するには、iOS SDK に精通している必要があります。Swift または Objective-C のどちらでアプリケーションを記述するかを選択できますが、Swift を使用することを強くお勧めします。また、iOS アプリケーションの開発原則とフレームワークに関する知識も必要です。初めて使用する場合は、developer.apple.com/develop/ で学習を開始することをお勧めします。他の前提条件は、「[iOS の基本的な要件](#)」を参照してください。

一部の Mobile SDK インターフェースでは、いくつかのメソッドとプロパティを上書きする必要があります。SDK ヘッダーファイルには、必須の上書きと省略可能な上書きを示すコメントが含まれています。

ログインとパスコードについて

Mobile SDK アプリケーションから Salesforce オブジェクトにアクセスするには、顧客が Salesforce 組織にログインする必要があります。ログインフローが開始すると、Mobile SDK によってアプリケーションの接続アプリケーション設定が Salesforce に送信されます。Salesforce は、ログイン画面をモバイルデバイスに投稿して応答します。

必要に応じて、Salesforce システム管理者は、ログイン後にパスコードを必要とするように接続アプリケーションを設定することができます。この設定では、たとえばフォアグラウンドに戻るときにバックグラウンドアプリケーションでパスコードの入力を要求することが必須になります。接続アプリケーションでモバイルアプリケーションのパスコードが要求される場合、Mobile SDK 9.2 以降ではデバイスのシステムのパスコードを使用します。

必須のパスコードを確認するために、Mobile SDK では顧客の設定済みの確認モード (たとえば、Touch ID、Face ID、または直接パスコード入力) を使用するロック画面を表示します。デバイスのパスコードが設定されていない場合、Mobile SDK では顧客に iOS 設定インターフェースにアクセスしてパスコードを作成するよう要求します。接続アプリケーションでモバイルパスコードが要求されない場合、Mobile SDK ではパスコードの確認ステップがスキップされます。Mobile SDK では、すべてのログイン画面とパスコードロック画面、および認証ハンドシェイクの処理が行われます。

モバイルパスコードポリシー

各 Mobile SDK アプリケーションでは、特定の Salesforce 組織の接続アプリケーションのコンシューマー鍵と OAuth コールバック URL をハードコード化します。Mobile SDK では、その組織の指定された接続アプリケーションの設定可能なパスコード要件が優先されます。Mobile SDK バージョン 9.2 以降では、PIN 長などの組織設定が無視され、代わりにデバイス設定が利用されます。同様に、不正なパスコード入力はモバイルオペレーティングシステムの標準手順に従って処理されます。

 **メモ:** Mobile SDK バージョン 9.2 以降では、組織の接続アプリケーションの [次の時間が経過したらアプリケーションをロック] 設定が無視され、アイドル状態になった後にデバイスをロックするデバイス設定が利用されました。Mobile SDK バージョン 10.1.1 以降では、接続アプリケーションの [次の時間が経過したらアプリケーションをロック] 設定が再び利用されています。これが設定されている場合、モバイルアプリケーションがタイムアウト期間より長くバックグラウンドで実行されているとロックされます。モバイルアプリケーションを有効にするとロックがかかります。アプリケーションのロック解除は従来どおりです。

顧客がアプリケーションを使用して異なる組織にログインする場合、Mobile SDK では指定された接続アプリケーション設定を取得できません。そのため、その顧客にはパスコードプロンプトが一切表示されません。

ロック画面のマルチユーザー動作

複数のユーザーが同じデバイスの同じアプリケーションにログインしている場合、ロック画面の動作は次のようになります。

1. パスコードが必要なアプリケーションを再開するときに、顧客は最初にロック画面でモバイルオペレーティングシステムを通じて認証を行うよう要求されます。
2. 顧客が認証をキャンセルすると、[ログアウト] ボタンと [ロック解除を再試行] ボタンがロック画面に表示されます。
3. [ログアウト] ボタンは、ロック画面を必須にしている顧客の場合のみ機能します。
4. ロック画面を必須にした最後のユーザーがログアウトすると、Mobile SDK にロック画面が表示されなくなります。

生体認証の使用

デフォルトでは、Mobile SDK では、パスコードの確認に Touch ID と Face ID を使用します。顧客は [設定] で生体認証やその他の iOS 入力モードの設定を定義できます。

生体認証では、次のアプリケーション設定が必要です。

アプリケーション設定

1. アプリケーションの情報プロパティリスト (plist) ファイルを開きます。
2. 次の最上位レベルのプロパティを追加します。
 - キー: `NSFaceIDUsageDescription` (表示ラベルは [Privacy - Face ID Usage Description (プライバシー - Face ID の使用方法の説明)])
 - 値: <任意の文字列を入力> (例: 「PIN の再入力为了避免のために Face ID を使用する」)

Key	Type	Value
▼ Information Property List	Dictionary	(19 items)
Privacy - Face ID Usage Description	String	This can be anything you want.
Localization native development region	String	en
Bundle display name	String	\$(PRODUCT_NAME)
Executable file	String	\$(EXECUTABLE_NAME)
▶ Icon files (iOS 5)	Dictionary	(0 items)
▶ CFBundleIcons~ipad	Dictionary	(0 items)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1.0
Application requires iPhone environment	Boolean	YES
SFDCOAuthLoginHost	String	login.salesforce.com
Launch screen interface file base name	String	LaunchScreen
▼ Required device capabilities	Array	(1 item)
Item 0	String	armv7
▶ Supported interface orientations	Array	(3 items)
▶ Supported interface orientations (iPad)	Array	(4 items)

生体認証の無効化

Mobile SDK 9.2 以降で作成されたアプリケーションでは、Salesforce 接続アプリケーションの生体認証設定が無視されます。代わりに、顧客はデバイス設定で認証モードを自分で設定できます。

関連トピック

- [PIN セキュリティについて](#)
- [OAuth 2.0 認証フロー](#)
- [生体認証](#) (ページ 472)

Salesforce Mobile SDK での Swift の使用

Swift アプリケーションの作成

Mobile SDK Swift プロジェクトを作成するには、手動で要素を組み立てるか、`forceios create` コマンドを使用できます。forceios では、アプリケーション種別に `native_swift` を指定するか、単に `Enter` キーを押します。次に例を示します。

```
forceios create
Enter your application type (native, native_swift, leave empty for native_swift): <press RETURN>
Enter your application name: TestSwift
Enter your package name: com.bestapps.swift
Enter your organization name (Acme, Inc.): BestApps, Inc.
Enter output directory for your app (leave empty for the current directory): TestSwift
```

手動でアプリケーションを作成する手順については、「[iOS Swift プロジェクトの手動作成](#)」を参照してください。

Swift API の名前

Mobile SDK では、Swift のメソッドとクラスは、関連する Objective-C API 署名に密接に従います。ただし、Objective-C API のコールを容易にするために、Mobile SDK では、ほとんどのメソッドとパラメーターに Swift 対応の名前が定義されています。Mobile SDK がカスタム Swift 名を定義すると、その名前は Objective-C ヘッダーファイル内で、`NS_SWIFT_NAME` タグが付けられ、Objective-C メソッド宣言の直後に表示されます。次に、メソッド宣言の例を示します。

```
- (nullable SFSyncState*) reSyncByName:(NSString*) syncName
    updateBlock:(SFSyncSyncManagerUpdateBlock) updateBlock
    NS_SWIFT_NAME (reSync(named: onUpdate:));
```

クラス名とプロトコル名では、宣言がクラス宣言の前になります。

```
NS_SWIFT_NAME (RestClient)
@interface SFRestAPI : NSObject
```

Objective-C API が Swift で使用できない場合、API 宣言の横に `NS_SWIFT_UNAVAILABLE("")` タグが表示されます。

ほとんどの Swift クラス名は、Objective-C 名から「SF」プレフィックスを除いたものですが、変更されたものもあります。たとえば、`SFRestAPI` は Swift では `RestClient` になります。注目すべき例をいくつか挙げます。

Objective-C	Swift
<code>SalesforceSDKManager</code>	<code>SalesforceManager</code>
<code>SFUserAccountManager</code>	<code>UserAccountManager</code>
<code>SFRestAPI</code>	<code>RestClient</code>
<code>SFSmartStore</code>	<code>SmartStore</code>
<code>SFMobileSync*</code>	<code>MobileSync*</code>

iOS ネイティブテンプレートアプリケーション

Mobile SDK 9.0 によって、デフォルトの Swift テンプレートが iOS 14 標準に更新されます。Objective-C テンプレートは前のリリースからのものが維持されます。

重要: 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

forceios デフォルトテンプレートを比較してまとめたものを次に示します。

Swift テンプレート

- [iOSNativeSwiftTemplate](#)
- 2つのレベルの主従ナビゲーションを示す。
- すべてのビューに SwiftUI を使用
- モデル - ビューアーキテクチャを示す
- AppDelegate と SceneDelegate 間でアプリケーションライフサイクルイベントの管理情報を配布

- 各種の Combine Publisher オブジェクトを使用して非同期 REST 応答を処理
- SmartStore と Mobile Sync を使用してオフライン機能を示す

Objective-C のテンプレート

- [iOSNativeTemplate](#)
- 取引先責任者の単一のリストを表示
- AppDelegate を通じてアプリケーションライフサイクルイベントを管理
- SFRestDelegate で非同期 REST 応答を処理

これらのテンプレートは、両方とも github.com/forcedotcom/SalesforceMobileSDK-Templates リポジトリで入手できます。ただし、forceios によるそれらの使用法は多少異なります。

- ネイティブ Swift アプリケーションの種類を選択するには、*Return* キーを押すか、*native_swift* と入力します。

```
$ forceios create
Enter your application type (native_swift or native, leave empty for native_swift):
<Press Return>
...
```

- Objective-C のアプリケーション種別を選択するには、*native* と入力します。

```
$ forceios create
Enter your application type (native_swift or native, leave empty for native_swift):
native
...
```

その他のテンプレートの使用

Mobile SDK では、次のような iOS に特化したテンプレートも提供しています。

[MobileSyncExplorerSwift](#)

MobileSync テクノロジーの高度に拡張されたサンプル。最近の取引先責任者ウィジェットの実装が含まれます。マルチウィンドウや横方向のサポートなどの iPad の機能も示します。

[iOSIDPTemplate](#)

Mobile SDK を使用した ID プロバイダーアプリケーションの作成用テンプレート。

[iOSNativeSwiftEncryptedNotificationTemplate](#)

暗号化された通知の処理方法を示します。

このテンプレートを使用するには、`forceios createWithTemplate` コマンドを使います。たとえば、Swift の暗号化された通知テンプレートを使用してアプリケーションを作成する場合は次のようになります。

```
$ forceios createwithtemplate
Enter URI of repo containing template application:
iOSNativeSwiftEncryptedNotificationTemplate
...
```

 **メモ:** (Mobile SDK 8.0 以降) SalesforceMobileSDK-Templates リポジトリのテンプレートを使用する場合、URI パスを指定せずにテンプレート名のみを指定できます。

アプリケーションフローの概要

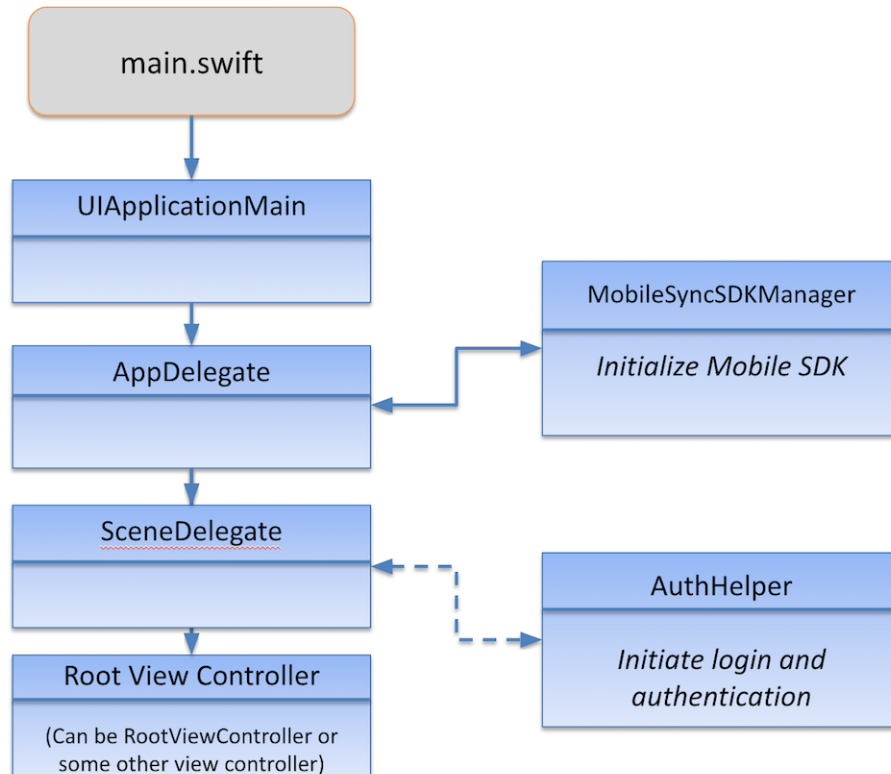
Mobile SDK のネイティブ iOS アプリケーションは、Objective-C と Swift のいずれでも、似たような設計方法になります。main.swift のソースファイル (Objective-C の main.m) では、アプリケーションの残りの部分のルートオブジェクトとなる UIApplicationMain オブジェクトが作成されます。UIApplicationMain コンストラクターは、Mobile SDK の SFApplication オブジェクトをインスタンス化し、アプリケーションのライフサイクルを管理する AppDelegate オブジェクトを作成します。

AppDelegate は、Mobile SDK サービスオブジェクトの SalesforceSDKManager を使用して、SDK オブジェクトを初期化します。正常に初期化した後のフローは Objective-C と Swift で多少異なります。

Swift のフロー

Mobile SDK 用のネイティブ iOS Swift テンプレートでは、AppDelegate、InitialViewController、SceneDelegate、ルートビューコントローラーの4つの主要なクラスが定義されています。AppDelegate オブジェクトが SDK を初期化するために SalesforceSDKManager またはそのいずれかのエクステンションをコールします。次に制御を SceneDelegate インスタンスに渡します。新しいシーンがフォアグラウンドに入るとき、SceneDelegate が InitialViewController (Salesforce のログインと認証画面のコンテナである空の UIViewController オブジェクト) をロードします。SceneDelegate は、必要に応じて AuthHelper をコールして認証プロセスを開始します。認証に成功した後、SceneDelegate はカスタムルートビューコントローラーに制御を渡すことでアプリケーションを起動します。このテンプレートでは、ルートビューコントローラーは AccountsListView クラスです。

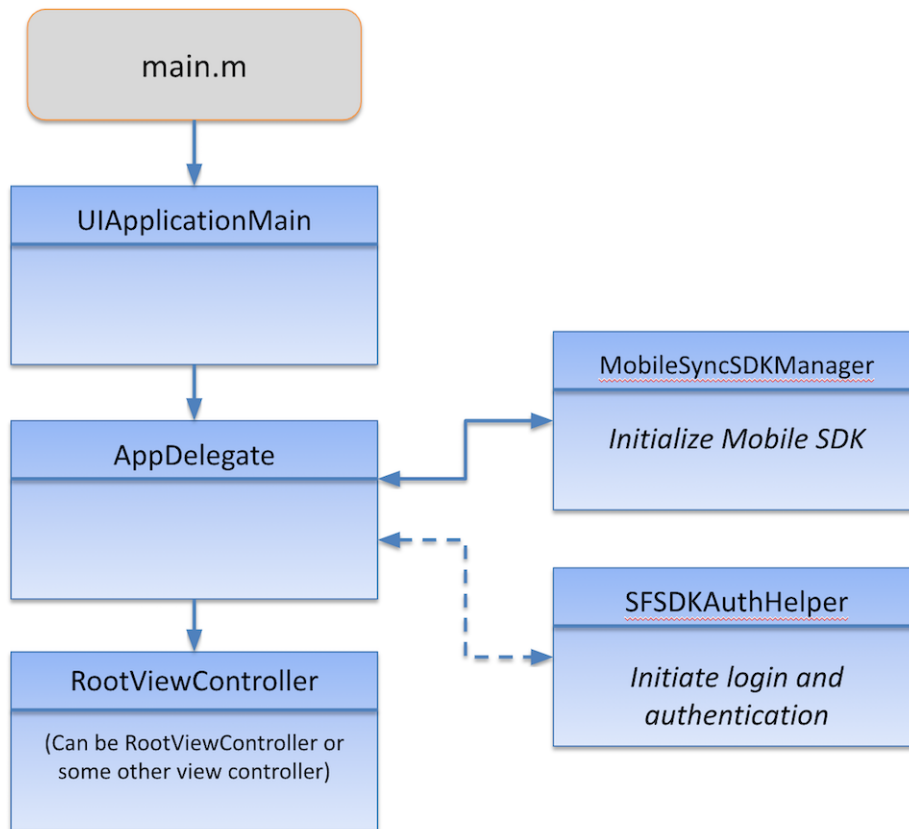
Application Flow (Swift)



Objective-C のフロー

Objective-C のネイティブ iOS テンプレートでは、`AppDelegate`、`InitialViewController`、`RootViewController` の3つのクラスが定義されています。`AppDelegate` オブジェクトは、`SFSDKAuthHelper` をコールして、最初に表示するビューであるログイン画面のために `InitialViewController` を読み込みます。認証が成功すると、`AppDelegate` オブジェクトにより、`RootViewController` に関連付けられたビューがアプリケーションのエントリポイントとして表示されます。

Application Flow (Objective-C)



メモ: テンプレートアプリケーションで示すワークフローは一例にすぎません。AppDelegate およびサポークラスをカスタマイズして、追加機能を有効にしたり、ワークフローを調整したりできます。たとえば、Salesforce 認証は必要になるまで延期できます。REST API コールを使用したデータの取得、データの表示、他のビューの起動、サービスの実行などを行うことができます。MobileSDKのテンプレートには、プッシュ通知の処理やIDPアプリケーションサービスの初期化などのコメント化された定型コードが含まれており、それらを有効化できます。

関連トピック:

[SDK マネージャークラス](#)

SDK マネージャークラス

Swift	Objective-C
SalesforceManager	SalesforceSDKManager
SmartStoreSDKManager	SmartStoreSDKManager
MobileSyncSDKManager	MobileSyncSDKManager
SalesforceHybridSDKManager	SalesforceHybridSDKManager

Swift	Objective-C
SalesforceReactSDKManager	SalesforceReactSDKManager

SalesforceManager クラスは、ブートストラップ設定を実行し、Mobile SDK コンポーネントを初期化します。また、PIN コード、OAuth 設定、その他のブートストラッププロセスを含む、アプリケーションの起動に関与するすべてのプロセスを調整します。つまり、SalesforceManager によって、開発者がアプリケーションの起動を調整する必要がなくなります。

SalesforceManager 以外にも、Mobile SDK には専門化された SDK マネージャークラスがあります。次の SDK マネージャークラスは Mobile SDK の特殊フレーバーをサポートします。それらの名前は Objective-C と Swift で同じです。

- SmartStoreSDKManager — SalesforceManager のサブクラス。SmartStore は使用するが Mobile Sync を使用しないネイティブアプリケーション用。廃止された SalesforceSDKManagerWithSmartStore クラスの代わりに使用されます。
- MobileSyncSDKManager — SmartStoreSDKManager のサブクラス。使用可能なすべての Mobile SDK ネイティブ機能へのアクセスを提供します。
- SalesforceHybridSDKManager — MobileSyncSDKManager のサブクラス。ハイブリッド Cordova アプリケーション用です。すべての使用可能な Mobile SDK ハイブリッド機能が含まれます。
- SalesforceReactSDKManager — MobileSyncSDKManager のサブクラス。React Native アプリケーション専用です。すべての使用可能な Mobile SDK React Native 機能が含まれます。

すべての Mobile SDK アプリケーションは、SalesforceManager またはそのサブクラスを使用して Mobile SDK を初期化します。これらのクラスのいずれかを使用するには、AppDelegate クラスの初期化でその initializeSDK メソッドをコールします。通常、アプリケーションは AppDelegate の外部で SDK マネージャメソッドをコールしません。Mobile SDK ネイティブテンプレートを使用して作成されたアプリケーションは、MobileSyncSDKManager のインスタンスを使用します。この選択の影響として、Mobile SDK は、MobileSyncSDKManager をそのインスタンスクラスとして使用するよう SalesforceSDKManager を内部的に設定します。

Swift の例

AppDelegate クラスで、Swift テンプレートは MobileSyncSDKManager クラスの initializeSDK() をコールします。

```
import Foundation
import UIKit
import MobileSync

class AppDelegate : UIResponder, UIApplicationDelegate
{
    var window: UIWindow?

    override
    init()
    {
        super.init()
    }
}
```

```
MobileSyncSDKManager.initializeSDK()  
...
```

`initializeSDK()` をコールすることで、SmartStore、Mobile Sync、コア SDK 内のすべてのものを使用するために適切な要素の準備が整います。別の SDK マネージャークラスを使用するには、そのモジュールをインポートして、コードの `MobileSyncSDKManager` を該当するマネージャークラスの名前に置き換えます。

Objective-C の例

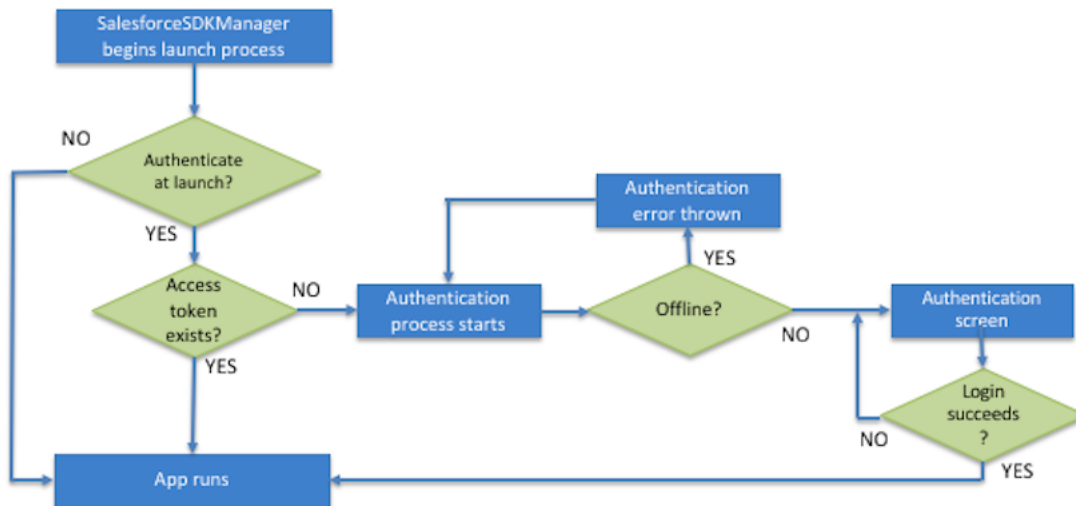
`MobileSyncSDKManager` を `AppDelegate` クラスで使用方法は、Swift アプリケーションで使用方法と同じです。次に、関連するテンプレートアプリケーションコードを示します。

```
#import <SalesforceSDKCore/SalesforceSDKManager.h>  
...  
#import <MobileSync/MobileSyncSDKManager.h>  
...  
  
@implementation AppDelegate  
  
@synthesize window = _window;  
  
- (instancetype)init  
{  
    self = [super init];  
    if (self) {  
        [MobileSyncSDKManager initializeSDK];  
    }  
    ...  
}
```

`[MobileSyncSDKManager initializeSDK]` メッセージにより、SmartStore、Mobile Sync、コア SDK 内のすべてのものを使用するために適切な要素の準備が整います。別の SDK マネージャークラスを使用するには、そのヘッダーファイルをインポートして、コードの `MobileSyncSDKManager` を該当するマネージャークラスの名前に置き換えます。

アプリケーション起動フロー

`application(_:didFinishLaunchingWithOptions:)` メッセージを受信したら、アプリケーションウィンドウを初期化し、ルートビューコントローラーを設定します。必要に応じて、Mobile SDK はログインおよび認証フローを実行します。アプリケーションの接続アプリケーションにパスワードが必要な場合は、ブートストラッププロセスを続行する前に、パスワード確認画面が表示されます。次のダイアグラムに、このフローを示します。



重要なポイント:

- 有効なアクセストークンが検出された場合、フローは Salesforce 認証をスキップします。
- アクセストークンが検出されず、デバイスがオフラインの場合、認証モジュールによりエラーが発生し、ログイン画面に戻ります。SalesforceSDKManager によってこのイベントがアプリケーションに反映されることはありません。

このダイアグラムに表示されている内容の他に、アプリケーションで ID プロバイダー機能とプッシュ通知機能がサポートされている場合、SalesforceSDKManager 起動プロセスはそれらの機能の設定もアプリケーションに委任します。ユーザーがパスコードチャレンジか、Salesforce ログインを失敗またはキャンセルした場合、内部イベントが起動します。その後、制御が AppDelegate に戻されます。

ユーザーのログアウト、ユーザーの切り替え、またはトークンの有効期限切れが発生するまで、SalesforceSDKManager オブジェクトは再表示されません。

認証ヘルパーオブジェクト

Mobile SDK アプリケーションは、ログインおよびログアウトイベントの処理を認証ヘルパーオブジェクトに任せます。

Swift	Objective-C
AuthHelper	SFSDKAuthHelper

認証ヘルパークラスでは、型レベルのメソッドのみを定義します。アプリケーションウィンドウを初期化したら、認証ヘルパーオブジェクトで `loginIfRequired` をコールできます。このコールを実行すると、Mobile SDK は直近のユーザーの状況をチェックします。Mobile SDK は無効な OAuth トークンを検出すると、アプリケーション起動フローを再実行します。

再初期化中の不要なデータ損失を避けるには、認証ヘルパーハンドラーメソッドを使用します。これらのメソッドを使用すると、さまざまなログアウトシナリオの完了ブロック (ハンドラー) を定義できます。1 つの一般的なログアウトハンドラーを定義することも、特定の操作の個別のハンドラーを定義することもできます。

一般的なログアウト

```
handleLogout(completionBlock: (() → Void)!)
```

すべてのログアウトシナリオに適用される操作を実行する場合。このメソッドは、アプリケーションでユーザーの切り替えがサポートされていない場合にのみ使用します。

現在のユーザーの変更

```
registerBlock(forCurrentUserChangeNotifications(completionBlock: (() → Void)!))
```

ログアウトおよびユーザーの切り替え操作に 1 つの完了ブロックを使用する場合。ログアウトのみのメソッド、ユーザーの切り替えのメソッドの順にコールし、完了ブロックを両方のメソッドに渡します。

ログアウトのみ

```
registerBlock(forLogoutNotifications(completionBlock: (() → Void)!))
```

ユーザーの切り替えシナリオのログアウト操作を処理する場合。完了ブロックを `handleLogout` 通知に転送します。

ユーザーの切り替え

```
registerBlock(forSwitchUserNotifications(completionBlock: (() → Void)!))
```

ログアウトの発生後にユーザーの切り替えおよび初期化を処理する場合。

関連トピック

[SFUserAccountManager](#)

AppDelegate クラス

AppDelegate クラスは、iOS アプリケーションの実際のエントリポイントです。Mobile SDK テンプレートアプリケーションでは、AppDelegate によって iOS UIApplicationDelegate インターフェースの一部が実装されます。Mobile SDK Swift アプリケーションの SwiftUI では、AppDelegate によって、次のタスクが実行されます。

- MobileSyncSDKManager オブジェクトによって Mobile SDK を初期化する

- Salesforce ログイン、およびルートビューのインスタンス化の制御を `SceneDelegate` へ渡す
- 必要に応じて、アプリケーションをプッシュ通知用に登録し、ID プロバイダー (IDP) 機能を設定する

初期化

`AppDelegate` の `init()` メソッドでは、`SalesforceManager` オブジェクトが `Mobile SDK` を初期化します。次のコードは、テンプレートアプリケーションで実装される `init` メソッドを示しています。

```
override
init()
{
    super.init()
    MobileSyncSDKManager.initializeSDK
}
```

`init()` から制御が返ると、iOS で `application(_:didFinishLaunchingWithOptions:_)` がコールされます。このコールは、プッシュ通知のためにアプリケーションを登録する行をコメント化しない限り、単純に制御を返します。

iOS では次に `AppDelegate` の `application(_:configurationForConnecting:_)` がコールされます。このメソッドは、UI スレッドで実行されている `SceneDelegate` インスタンスに制御を渡します。`SceneDelegate` がアプリケーションの UI を起動している間、`AppDelegate` はプッシュ通知プロトコルのメソッドをコールし続けます。それらの通知は、通知を許可するか拒否するかを顧客に尋ねるプロンプトで終わります。

アプリケーションがバックグラウンドでアクティブになっている場合、フォアグラウンドでは、iOS がこの初期化プロセスの短縮されたものを実行します。

初期化 (Objective-C)

次に、テンプレートアプリケーションで実装される `init` メソッドを示します。この後に、`application:didFinishLaunchingWithOptions:` メソッドで `SalesforceSDKManager` の `loginIfRequired:` メソッドへのコールが続きます。

```
- (instancetype)init
{
    self = [super init];
    if (self) {
        [MobileSyncSDKManager initializeSDK];

        //App Setup for any changes to the current authenticated user

        [SFSDKAuthHelper registerBlockForCurrentUserChangeNotifications:^(
            [self resetViewState:^(
                [self setupRootViewController];
            )];
        )];
    }
    return self;
}
```

`init` メソッドで、`SalesforceSDKManager` オブジェクトは次のことを行います。

- Mobile SDK を初期化します。
- ユーザー変更通知を処理するブロックを登録します。このブロックは、`setupRootViewController` メソッドをコールして新しいユーザーのためにビューをリセットするようにアプリケーションを設定します。

AppDelegate クラスで、`setupRootViewController` を実装して、認証後にアプリケーションの最初の画面を表示します。`self.window` オブジェクトには、`application:didFinishLaunchingWithOptions:` が完了するまでに有効な `rootViewController` が必要です。Mobile SDK Objective-C テンプレートの実装を次に示します。

```
- (void)setupRootViewController
{
    RootViewController *rootVC =
        [[RootViewController alloc] initWithNibName:nil bundle:nil];
    UINavigationController *navVC =
        [[UINavigationController alloc] initWithRootViewController:rootVC];
    self.window.rootViewController = navVC;
}
```

UIApplication イベントハンドラー

UIApplicationDelegate プロトコルを使用して、UIApplication イベントハンドラーを実装することもできます。実装またはカスタマイズを考慮すべき重要なイベントハンドラーは、次のとおりです。

- **application(_:didFinishLaunchingWithOptions:)**
アプリケーションのライフサイクルの最初にコールされます。テンプレートアプリケーションの実装は単純に制御を返します。必要に応じて、このメソッドを使用してプッシュ通知用のアプリケーションを登録できます。
- **applicationDidBecomeActive(_:)**
アプリケーションがフォアグラウンドに移った後にコールされます。
- **application(_:didRegisterForRemoteNotificationsWithDeviceToken:)**
Salesforce からの受信プッシュ通知の処理に使用されます。テンプレートアプリケーションのスタブ実装のコメントアウトされた手順とコードに従います。

すべての UIApplication イベントハンドラーのリストは、[UIApplicationDelegate](#) のドキュメントを参照してください。

関連トピック:

[iOS でのプッシュ通知の使用](#)

SceneDelegate クラス

SceneDelegate クラスは、シーンの設定とライフサイクル管理を処理します。Mobile SDK Swift テンプレートでは、さらに SceneDelegate で Salesforce のログインを処理し、アプリケーションのルートビューコントローラーを設定できます。現在、Mobile SDK の Objective-C テンプレートでは、SceneDelegate を使用していません。

OAuth リソースは、独立したモジュールに保存されています。この分離によって、Salesforce 認証のオンデマンドでの使用が可能になります。ログインプロセスは、SceneDelegate の実装内から開始することも、実際に必要になるまで (scustom サブビューから OAuth をコールするなど) 延期することもできます。

初期化

次のコードでは、テンプレートアプリケーションで現在のユーザーの変更イベント通知のハンドラーがどのように設定されているかを確認できます。

```
func scene(_ scene: UIScene, willConnectTo session: UISceneSession,
options connectionOptions: UIScene.ConnectionOptions) {

    guard let windowScene = (scene as? UIWindowScene) else { return }
    self.window = UIWindow(frame: windowScene.coordinateSpace.bounds)
    self.window?.windowScene = windowScene

    AuthHelper.registerBlock(forCurrentUserChangeNotifications: {
        self.resetViewState {
            self.setupRootViewController()
        }
    })
}
```

AuthHelper は、Mobile SDK の共有オブジェクトであり、ユーザー変更イベントを処理する閉じたブロックを登録します。このような変更が発生した場合、Mobile SDK は前のユーザーのデータを破棄します。ここでは、登録されたハンドラブロックが resetViewState() をコールして、既存のビュー階層を展開します。最後に、resetViewState() が setupRootViewController() をコールしてアプリケーションを元の状態に戻します。

```
func setupRootViewController() {
    // Setup store based on config userstore.json
    MobileSyncSDKManager.shared.setupUserStoreFromDefaultConfig()
    // Setup syncs based on config usersyncs.json
    MobileSyncSDKManager.shared.setupUserSyncsFromDefaultConfig()

    self.window?.rootViewController = UIHostingController(
        rootView: AccountsListView()
    )
}
```

ここでは、テンプレートがこの実行ポイントを利用して SmartStore と Mobile Sync を設定します。これは、ユーザーごとに1回だけ発生するイベントです。次に、AccountsListView をルートビューに設定し、これを基に UIHostingController オブジェクトを作成します。この新しいオブジェクトがルートビューコントローラーになります。アプリケーションがフォアグラウンドに入るたびに (最初のロード時や、顧客がバックグラウンドから復帰させたとき)、iOS が sceneWillEnterForeground(_:) をコールします。

```
func sceneWillEnterForeground(_ scene: UIScene) {
    // Called as the scene transitions from the background to the foreground.
    // Use this method to undo the changes made on entering the background.
    self.initializeAppViewState()
    AuthHelper.loginIfRequired {
```

```

        self.setupRootViewController()
    }
}

```

`initializeAppViewState()` メソッドは、アプリケーションの最初の画面の状態を初期化またはリセットします。`setupRootViewController()` をコールする前に、ログインページのナビゲーションバーの外観と機能をカスタマイズできます。たとえば、`SalesforceLoginViewControllerConfig` クラスの公開プロパティをカスタマイズするメソッドを作成します。

```

func customizeLoginView() {
    let loginViewConfig = SalesforceLoginViewControllerConfig()

    // Set showSettingsIcon to false if you want to hide the settings
    // icon on the nav bar
    loginViewConfig.showsSettingsIcon = false

    // Set showNavBar to false if you want to hide the top bar
    loginViewConfig.showsNavigationBar = true
    loginViewConfig.navigationBarColor = UIColor(red: 0.051, green: 0.765, blue: 0.733,
alpha: 1.0)
    loginViewConfig.navigationTitleColor = UIColor.white
    loginViewConfig.navigationBarFont = UIFont(name: "Helvetica", size: 16.0)
    UserManager.shared.loginViewControllerConfig = loginViewConfig
}

```

`loginIfRequired` メソッドは、`void` を返すブロック変数を取り、パラメーターを持ちません。上のように、このブロックを使用して、ルートビューコントローラーを設定します。ここでは、`setupRootViewController` メソッドが `self.window` の `rootViewController` プロパティをアプリケーションの最初のカスタムビューに設定します。このため、アプリケーションのカスタム動作が起動されます。このテンプレートでは、カスタムビューは取引先のリストビューです。

遅延ログインについて

Mobile SDK 7.0 以降、ログインダイアログを表示する必要があるタイミングを Mobile SDK が決定するようにできます。`AuthHelper.loginIfRequired` がコールされると、SDK はユーザーが有効な OAuth トークンを持っているかどうかを確認します。トークンがない場合、Mobile SDK はユーザーにログインを求めます。

このメソッドを `SceneDelegate` 実装内でコールする必要はありません。ログインを、アプリケーションの初期化完了後の任意のポイントに延期できます。認証を延期する手順は、次のとおりです。

1. Xcode エディターで、`SceneDelegate` クラスを開きます。
2. `sceneWillEnterForeground(_:)` メソッドで、`loginIfRequired` のコールを削除します。ただし、ブロック内のコードは残します。

```

//AuthHelper.loginIfRequired {
    self.setupRootViewController() // Keep this call in sceneWillEnterForeground(_:)
//}

```

3. ログインを延期する目的のポイントで `AuthHelper.loginIfRequired { }` をコールします。適切な順序で操作が行われるようにするために、認証が必要なコードを `loginIfRequired:` のクロージャータラメーター内に配置します。

RootViewController クラス

ルートビューコントローラーは、アプリケーションが出力した最初のカスタムビューを表示します。Objective-C アプリケーションでは、このビューはビュースタックの一番下またはルートにあるビューです。Mobile SDK Objective-C テンプレートプロジェクトでは、Mobile SDK によって、ルートビューコントローラークラス `RootViewController` に適切な名前が付けられます。このクラスでは、アプリケーションが Salesforce REST API とやりとりするためのメカニズムを設定します。ルートビューコントローラーの定義方法に関係なく、テンプレートアプリケーションのコードを再利用して、REST API 経由で Salesforce データを取得できます。

RootViewController のデザイン

基本的な Mobile SDK アプリケーションの単独のカスタムビューコントローラーとして、`RootViewController` クラスでは重要項目のみが取り扱われます。主要な 2 つのタスクは次のとおりです。

- Salesforce REST API を使用して Salesforce データを照会する
- Salesforce データをテーブルに表示する

このアプリケーションは、SOQL クエリの形式で非同期 REST 要求を発行して Salesforce データを取得します。この場合のクエリは、最大 10 件の取引先責任者レコードから `Name` 項目を取得する単純な `SELECT` ステートメントです。アプリケーションは、静的な参照のみテーブルでクエリ結果を表示します。Mobile SDK は、現在のユーザーの認証済みログイン情報を使用して、REST 要求の作成と送信を行います。

Swift の実装

Mobile SDK 9.0 以降、Swift テンプレートアプリケーションでは、`RootViewController` クラスを定義しません。代わりに、「[ネイティブ Swift テンプレート](#)」で説明されているように、同じ機能を実現するために Combine 拡張機能により SwiftUI のビューとモデルが使用されています。

Objective-C の実装

Objective-C の `RootViewController` クラスは、`UITableViewController` を継承し、`SFRestDelegate` プロトコルを実装します。このアクションは、`UIViewController` の `viewDidLoad` メソッドの上書きで開始します。

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    self.title = @"Mobile SDK Sample App";

    SFRestRequest *request =
        [[SFRestAPI sharedInstance]
         requestForQuery:@"SELECT Name FROM Contact LIMIT 10"
         apiVersion:kSFRestDefaultAPIVersion];
    [[SFRestAPI sharedInstance] send:request delegate:self];
}
```

スーパークラスメソッドをコールしてビューのタイトルを設定した後、アプリケーションはその REST 要求を作成して送信します。`requestForQuery` および `send:delegate:` メッセージが、`SFRestAPI` クラスの単一の共有インスタンスに送信されます。この単一オブジェクトをすべての REST 要求に使用します。

アプリケーションが REST 要求を送信すると、Salesforce は `send:delegate:` メッセージにリストされる代理に状況メッセージとデータ (可能な場合) を渡すことによって応答します。この場合、`RootViewController` 自体が代理オブジェクトになります。

```
[[SFRestAPI sharedInstance] send:request delegate:self];
```

REST 応答を処理するために、Objective-C の `RootViewController` は、`sendRestRequest:failBlock:completeBlock:Block` メソッドの代わりに `SFRestDelegate` プロトコルを使用します。応答は、いずれかの `SFRestDelegate` コールバックメソッドで受信されます。要求が成功した場合、コードで `request:didSucceed:rawResponse:` コールバックの Salesforce データが処理されます。

```
- (void)request:(SFRestRequest *)request
    didSucceed:(id)jsonResponse
    rawResponse:(NSURLResponse *)rawResponse {

    NSArray *records = jsonResponse[@"records"];
    [SFLogger d:[self class]
        format:@"request:didLoadResponse: #records: %lu",
        (unsigned long)records.count];
    self.dataRows = records;
    dispatch_async(dispatch_get_main_queue(), ^{
        [self.tableView reloadData];
    });
}
```

`id` データ型の使用で提示されるように、このコードでは JSON 応答が汎用的な Objective-C 形式で処理されます。`jsonResponse` オブジェクトは `NSDictionary` のインスタンスとして処理され、そのレコードは `NSArray` オブジェクトとして扱われます。`RootViewController` では `UITableViewController` が実装されるため、抽出されたレコードを使用してテーブルを入力するのは簡単です。

ネイティブ Swift テンプレート

iOS のネイティブ Swift テンプレートは、最新の Swift 機能に対応しています。

- SwiftUI
- Combine Publisher
- SceneUI および SceneDelegate フロー

また、このテンプレートは、Mobile SDK オフライン機能も示します。

このテンプレートでは、`AppDelegate` と `SceneDelegate` の 2 つの初期化クラスが定義されています。これらのクラスが対話して、複数のウィンドウを動的に配列できます。

アーキテクチャ上、このテンプレートでは、厳格なモデル-ビュー分離が使用されます。ビュークラスあたりのモデルクラスは 1 つです。モデルクラスは `ObservableObject` プロトコルを実装し、ビュークラスは `View` プロトコルを実装します。

AppDelegate

`AppDelegate` は、高度なセットアップ、つまりアプリケーションの起動と `UISceneSession` ライフサイクルを管理します。Mobile SDK の用語の場合、このテンプレートの `AppDelegate` は、次を行います。

- Mobile SDK の初期化
- プッシュ通知サービスが実装されている場合は、その登録および登録解除
- IDP が実装されている場合は、そのセットアップ

以前の Mobile SDK テンプレートと同様に、AppDelegate の

`application(_:didFinishLaunchingWithOptions:)` メソッドの設定をカスタマイズできます。

SceneDelegate

SceneDelegate は、UISceneSession コンテナのシーンのライフサイクルを処理します。次のイニシャライゼーションメソッドは、シーンを設定して、シーンコレクションに追加します。

```
func scene(_: UIScene, willConnectTo session: UISceneSession, options connectionOptions: UIScene.ConnectionOptions)
```

シーンを UIWindow としてインスタンス化した後、このメソッドは、現在のユーザーの変更用のハンドラーを登録します。CurrentUserChange イベントが発生した場合、テンプレートは、ルートビューコントローラーをセットアップして、ビューの状態をリセットします。

```
AuthHelper.registerBlock(forCurrentUserChangeNotifications: {
    self.resetViewState {
        self.setupRootViewController()
    }
})
```

その後、sceneDidBecomeActive(_:) メソッドで、アプリケーションビューの状態を再初期化し、ルートビューコントローラーをリセットします。ユーザーがSalesforceにログインしていない場合は、このリセットを実行する前に、ユーザーにログインするように要求します。

```
func sceneWillEnterForeground(_ scene: UIScene) {
    // Called as the scene transitions from the background to the foreground.
    // Use this method to undo the changes made on entering the background.
    self.initializeAppViewState()
    AuthHelper.loginIfRequired {
        self.setupRootViewController()
    }
}
```

Model のクラス

テンプレートのビューのモデルは、Salesforce 情報のダウンロード方法の点で異なります。

- AccountsListModel 句は、Mobile Sync SyncManager パブリッシャーを使用して、すべての取引先の情報を下位同期します。応答が返されると、パブリッシャーは、同期とSmartStoreの設定に従って、その情報をSmartStore スープに保存します。

```
func fetchAccounts() {
    _ = syncManager?.publisher(for: "syncDownAccounts")
        .receive(on: RunLoop.main)
        .sink(receiveCompletion: { _ in }, receiveValue: { _ in
            self.loadFromSmartStore()
        })
}
```

```

        self.loadFromSmartStore()
    }

```

取引先情報をビューに送信するために、`loadFromSmartStore()` メソッドは、`SmartStore` パブリッシャーを使用して、`SmartStore` スープからデータを抽出します。

```

private func loadFromSmartStore() {
    storeTaskCancellable =
        self.store?.publisher(for: "select {Account:Name},
        {Account:Industry}, {Account:Id} from {Account}")
        .receive(on: RunLoop.main)
        .tryMap {
            $0.map { (row) -> Account in
                let r = row as! [String?]

                return Account(id: r[2] ?? "", name: r[0] ?? "",
                    industry: r[1] ?? "Unknown Industry" )
            }
        }
        .catch { error -> Just<[Account]> in
            print(error)
            return Just([Account]())
        }
        .assign(to: \AccountsListModel.accounts, on:self)
}

```

- `ContactsForAccountModel` 句は、`RestClient` ファクトリーメソッドを使用して、REST 要求を作成します。

```

let request = RestClient.shared.request(forQuery: "SELECT id, firstName, lastName, phone,
email, mailingStreet, mailingCity, mailingState, mailingPostalCode FROM Contact WHERE
AccountID = '\(acct.id)'", apiVersion: nil)

```

要求を `Salesforce` に送信して非同期応答を処理するために、このモデルは `RestClientsharedCombine` パブリッシャーを使用します。

```

contactsCancellable = RestClient.shared.publisher(for: request)
    .receive(on: RunLoop.main)
    .tryMap({ (response) -> Data in
        response.asData()
    })
    .decode(type: ContactResponse.self, decoder: JSONDecoder())
    .map({ (record) -> [Contact] in
        record.records
    })
    .catch( { error in
        Just([])
    })
    .assign(to: \.contacts, on:self)
...

```

このモデルは、表示されるすべての取引先責任者の詳細を発行済み配列に保存します。[「REST 要求の処理」](#)を参照してください。

View のクラス

ビューには、取引先のリストと、関連する取引先責任者のリストビューおよび詳細ビューが含まれます。取引先リストビューには、取引先名のリストが表示されます。取引先を選択すると、取引先の取引先責任者のリストビューが表示されます。顧客が取引先責任者レコードを選択すると、詳細ビューに取引先責任者の情報が表示されます。

- 取引先
 - Account > Contacts
 - Contact > Details

ビューによって、関連するモデルオブジェクトのデータをインポートして表示する視覚的インターフェースを設定できます。ビューの項目のタップを処理するために、ビューオブジェクトは、対象ビューへのナビゲーションリンクをセットアップして、必要なデータを対象ビューに渡します。取引先リストビューの定義を次に示します。

```
struct AccountsListView: View {
    @ObservedObject var viewModel = AccountsListModel()

    var body: some View {
        NavigationView {
            List(viewModel.accounts) { dataItem in
                NavigationLink(destination:
                    ContactsForAccountListView(account: dataItem)){
                    HStack(spacing: 10) {
                        VStack(alignment: .leading, spacing: 3) {
                            Text(dataItem.name)
                            Text(dataItem.industry).font(.subheadline).italic()
                        }
                    }
                }
            }
            .navigationBarTitle(Text("Accounts"), displayMode: .inline)
        }
        .onAppear{ self.viewModel.fetchAccounts() }
    }
}
```

関連トピック

- [Scenes \(Apple Developer Documentation\) \(Scenes \(Apple 開発者ドキュメント\)\)](#)
- [Ray Wenderlich's MVVM with Combine Tutorial for iOS \(Ray Wenderlich の iOS 向け MVVM with Combine チュートリアル\)](#)
- <https://www.vadimbulavin.com/ios-13-ipados-app-life-cycle-with-uiscene-scene-session-and-scene-delegate/>

Mobile SDK での Salesforce REST API の使用

Salesforce API は、REST エンドポイントを通じて Salesforce オブジェクトにアクセスするためのサービスを提供しています。Mobile SDK は、これらのサービスをサポートするために、次を行う REST クラスを提供しています。

- REST 要求オブジェクトを自動的に作成するファクトリーメソッドを実装する。

- 設定に基づいて認証済み要求を Salesforce に送信する。
- サーバーの応答を捕捉し、Swift オブジェクトまたは Objective-C オブジェクトとしてアプリケーションに返す。

Mobile SDK for iOS では、すべての REST 要求が非同期に実行されます。REST 要求が成功した場合、応答は `Array` オブジェクトまたは `Dictionary` オブジェクトとしてアプリケーションに返されます。要求が失敗した場合、応答には `Error` オブジェクトが含まれます。

関連トピック

- [サポートされる REST サービス](#)
- すべての Salesforce API の概要については、Salesforce ヘルプの「[どのAPIを使用するか?](#)」を参照してください。
- Salesforce API 要求および応答の形式についての詳細は、『[REST API 開発者ガイド](#)』を参照してください。

Mobile SDK REST クライアントインターフェース

Swift	Objective-C
<code>RestClient</code>	<code>SFRestAPI</code>

このクラスは、書式設定を作成し、REST 要求を Salesforce サービスに送信するネイティブインターフェースを定義します。サービスが応答すると、このクラスは非同期応答を `RestClientDelegate` プロトコルの実装、または定義したコールバックブロックまたはクロージャーに転送します。

`RestClient` — `RestRequest` インスタンスのファクトリーとして機能します。サポートされる Lightning Platform エンドポイントごとに要求ファクトリーメソッドを定義します。各ファクトリーメソッドは、要求パラメーター値を使用してカスタマイズされた `RestRequest` インスタンスを返します。

`SFRestAPI` — Swift の名前変更なしに Objective-C の REST 要求ファクトリーメソッドを定義します。Swift `RestClient` オブジェクトでこれらのメソッドをコールするには、Xcode コンパイラーで提供されるオートコンプリート候補表示機能を使用します。たとえば、Mobile SDK アプリケーションで「`RestClient.shared.request`」と入力してから、リストで対象を選択します。

 **警告:** Swift コンパイラーは、メソッドとパラメーターの名前をヒューリスティックに決定します。このため、署名は、Objective-C の署名とは異なる場合があります。

関連トピック

- サポートされる Salesforce Platform API のリストについては、「[サポートされる REST サービス](#)」を参照してください。
- [REST 要求の作成](#)

REST 要求クラス

Swift	Objective-C	
RestRequest	SFRestRequest	

`RestRequest` インターフェースでは、複雑でない方法でネットワークコールを実行できます。このクラスは要求メタデータのコンテナですが、要求オブジェクトを手動で操作することはめったにありません。要求オブジェクト自体は、HTTP で転送できるようにそのメタデータを書式設定します。要求を Salesforce サーバーに送信するために `RestRequest` オブジェクトは、ネットワークパラメーターと REST API 構文を設定します。作成されたオブジェクトが Salesforce リソース用である場合は、クラスでエンドポイントの計算方法がわかります。それ以外の場合は、カスタムパスを設定します。

Mobile SDK で明示的にサポートされない REST 要求を設定するメソッドを提供するのは、`RestClient` ではなく `RestRequest` です。たとえば、Connect REST API または外部サービスを使用するには、`RestRequest` オブジェクトを手動で作成して設定します。

REST 要求の作成

Mobile SDK REST API では、多くの種別の Salesforce REST 要求がサポートされます。Lightning Platform API の場合、`RestClient` は、サポートされるすべてのエンドポイントの `RestRequest` インスタンスを返すファクトリーメソッドを定義します。カスタマイズされた `RestRequest` オブジェクトを取得するには、要求パラメーターを指定して、該当する REST クライアントファクトリーメソッドをコールします。`RestRequest` ファクトリーオブジェクトは、指定された要求種別に特化されたオブジェクトで、入力値を使用して設定されます。

REST 要求の作成方法の概要を次に示します。

- Lightning Platform API の場合、`RestClient` ファクトリーメソッドは、最小限のデータ入力に基づいて、書式設定済みの `RestRequest` オブジェクトを返します。
- (Objective-C のみ) `SFRestAPI (Blocks)` カテゴリメソッドでは、REST 要求を定義して、以下のコールで送信できます。REST 応答を処理するのは、REST 代理オブジェクトではなく、ブロック引数です。
- `SFRestAPI (Files)` カテゴリメソッドは、Salesforce ファイルベースリソースへのアクセスを可能にする `RestRequest` インスタンスを作成します。
- `RestRequest` メソッドでは、非 Lightning Platform および外部 API をコールするためのカスタム設定がサポートされます。
- `SFRestAPI (QueryBuilder)` カテゴリメソッドを使用すると、独自のクエリや検索を手動で書式設定する手間を省くことができます。これらのメソッドは、ユーザーの入力に基づいて SOQL クエリ文字列や SOSL 検索文字列を作成します。

関連トピック

- [REST 要求メソッドの使用](#)
- [認証されていない REST 要求](#)
- [SFRestAPI \(Blocks\) カテゴリ](#)
- [SFRestAPI \(QueryBuilder\) カテゴリ](#)

- [SFRestAPI \(Files\) カテゴリ](#)

REST 要求の処理

Mobile SDK は、実行時に `RestClient` の単一インスタンスを作成します。このインスタンスを使用して、`RestRequest` オブジェクトを取得し、そのオブジェクトを Salesforce サーバーに送信します。

`RestRequest` オブジェクトは、多くの CRUD 操作に使用できますが、最も一般的な使用事例である SOQL クエリを見てみましょう。ほとんどの場合、他の種別の要求では、同じフローで入力値が異なるものが使用されます。デフォルトでは、SOQL クエリ要求では、バッチあたり最大 2,000 レコードが返されます。Mobile SDK 9.1 以降では、`batchSize` 引数を指定して、最大バッチサイズをカスタマイズできます。この引数には、200～2,000 の間の任意の整数値を指定できます。バッチサイズを指定しても、返されるバッチがそのサイズになるという保証はありません。

SOQL REST 要求を作成して Salesforce サーバーに送信する手順は、次のとおりです。

1. 要求オブジェクトを作成するために、SOQL クエリを取得する `RestClient` ファクトリーメソッドをコールします。

Swift

```
let request =
    = RestClient.shared.request(forQuery: "SELECT Name FROM Contact LIMIT 10",
    apiVersion: SFRestDefaultAPIVersion, batchSize: 500)
```

Objective-C

```
SFRestRequest *request = [[SFRestAPI sharedInstance]
    requestForQuery:@"SELECT Name FROM Contact LIMIT 10"];
```

2. REST クライアントインスタンスを使用して新しい REST 要求オブジェクトを送信します。

Swift

オプション 1

`RestClient` 共有 `Combine` パブリッシャーを使用します。たとえば、次の要求を使用するとします。

```
let request = RestClient.shared.request(forQuery: "SELECT id, firstName,
    lastName, phone, email, mailingStreet, mailingCity, mailingState,
    mailingPostalCode FROM Contact WHERE AccountID = '\(acct.id)'",
    apiVersion: nil, batchSize: 500)
```

要求を `RestClient.shared.publisher` に渡します。コールのチェーニングを経験されたことのない方のために、次のコードのコメントでは、一般的なパブリッシャーのシーケンスでどのような処理を実行するかを説明しています。このコードは、Swift テンプレートアプリケーションの `ContactsForAccountModel` クラスに含まれている `fetchContactsForAccount` メソッドのものです。

```
// Use the RestClient publisher to send the request and return the raw REST response
contactsCancellable = RestClient.shared.publisher(for: request)
    // Receive the raw REST response object
    .receive(on: RunLoop.main)
```

```

// Try to convert the raw REST response to Data
// Throw an error in the event of failure
.tryMap({ (response) -> Data in
    // Return the response as a Data byte buffer
    response.asData()
})

// Decode the Data object as JSON to produce an array
// of Contacts formatted as a ContactResponse object
.decode(type: ContactResponse.self, decoder: JSONDecoder())

// Map the JSON array of Contacts
.map({ (record) -> [Contact] in
    // Copy the Contact array to the records
    // member of ContactResponse
    record.records
})

// If tryMap failed, check for errors on the most recent
// object in the chain
.catch( { error in
    Just([])
})

// Store the array of contacts in the model's published contacts
// property for use by ContactsForAccountListView
.assign(to: \.contacts, on: self)

```

Contact および **ContactResponse** は構造体であり、それぞれ取引先責任者レコードと、SOQL クエリへの書式設定された REST 応答を表します。どちらのオブジェクトも「Decodable」として宣言されています。

```

struct Contact : Identifiable, Decodable {
    let id: UUID = UUID()
    let Id: String
    let FirstName: String?
    let LastName: String?
    let PhoneNumber: String?
    let Email: String?
    let MailingStreet: String?
    let MailingCity: String?
    let MailingState: String?
    let MailingPostalCode: String?
}

struct ContactResponse: Decodable {
    var totalSize: Int
    var done: Bool
    var records: [Contact]
}

```

RestClient パブリッシャーを使用する利点を理解できましたでしょうか。単一の論理ブロックで、要求を送信して応答を処理できます。

オプション2

要求オブジェクトを `send(request:_:)` メソッドに渡します。通常、Swift アプリケーションは、次に示すようにトレイリングクローザーで非同期応答を処理します。

```
RestClient.shared.send(request: request) { [weak self] (result) in
    switch result {
    case .success(let response):
        self?.handleSuccess(response: response, request: request)
    case .failure(let error):
        SalesforceLogger.d(RootViewController.self,
            message:"Error invoking: \(request) , \(error)")
    }
}
```

次の別の例では、コール側オブジェクトが `RestClientDelegate` プロトコルを実装して、応答を処理します。したがって、`send(request:delegate:)` オーバーロードがコールされ、それ自体が代理として渡されます。

```
// This class implements the RestClientDelegate protocol to handle responses
RestClient.shared.send(request, delegate: self)
```

Objective-C

要求オブジェクトを `send:delegate:` メソッドの `send:` パラメーターに渡します。通常、Objective-C アプリケーションの場合、コール側オブジェクトは、次に示すように `SFRestDelegate` プロトコルを独自に実装して、非同期応答を処理します。

```
// This class implements the SFRestDelegate protocol
[[SFRestAPI sharedInstance] send:request delegate:self];
```

「[REST 応答の処理](#)」を参照してください。

REST 応答の処理

Salesforce サーバーでは、すべての REST 応答が JSON 文字列として作成されます。Mobile SDK は、この未加工の応答を iOS オブジェクトでラップして、アプリケーションのハンドラーに渡します。応答オブジェクトの種別は、未加工の JSON 構造に従います。

- JSON ルート要素がオブジェクトの場合、応答は Swift Dictionary です。
- JSON ルート要素が配列の場合、応答は Swift Array です。

要求されたペイロードデータを応答オブジェクトから抽出するのは、アプリケーションの責任です。ペイロードの構成が不明の場合は、『[REST API 開発者ガイド](#)』の[リファレンス資料](#)を参照してください。

Mobile SDK では、さまざまな方法で REST 応答を処理できます。次に例を示します。

- (Swift のみ) Combine パブリッシャーを使用します。RestClient.shared.publisher は SOQL 要求を送信し、将来的には非同期応答を受信します。次に、コールをチェーン化して、未加工の応答を解析し、アプリケーションでの使用に合わせて書式設定できます。[テンプレートリポジトリ](#)の `iOSNativeSwiftTemplate` ディレクトリ内の `ContactForAccountModel.swift` を参照してください。
- (Swift のみ) トレイリングクローザー内の `send(request:_:)` コールの後で処理します。この方法は、Swift の「vanilla」Mobile SDK アプリケーションでお勧めします。

- (Swift、Objective-C) Mobile SDK REST 代理プロトコルを実装し、その ID を REST クライアントの `send` メソッドに渡して処理します。この方法は、Objective-C アプリケーションでお勧めします。
- (Objective-C のみ) `send` メソッド内に成功ブロックと失敗ブロックを実装し、`SFRestApi (Blocks)` カテゴリからのメソッドを使用して処理します。

Swift での応答処理

現在、最も新しく効率的な方法は、「[REST 要求の処理](#)」で説明したように、Combine パブリッシャーを使用することです。

REST 代理プロトコルの使用

Swift	Objective-C
<code>RestClientDelegate</code>	<code>SFRestDelegate</code>

通常、REST 代理プロトコルは、Objective-C アプリケーションで使用され、`SFRestApi (Blocks)` メソッドよりもコードが理解しやすくなっています。代理プロトコルは Swift でも使用できますが、パブリッシャーまたはトレイリングクロージャーを使用すると、より効率的かつ明確に同じ結果が得られます。

REST 代理プロトコルを採用したクラスは、Salesforce から送信される非同期 REST 応答の送信先候補になります。REST 要求を送信するときは、REST API クライアントに対し、応答を特定の代理インスタンスに転送するように指示できます。Mobile SDK は、受信した応答を、指定された代理インスタンスの該当するメソッドに転送します。

`SFRestDelegate` プロトコルでは、4 つの可能な応答が宣言されます。

- `request:didLoadResponse:` — 要求は処理されました。代理オブジェクトは JSON 形式の応答を受信します。成功を示すコールバックはこれだけです。
- `request:didFailLoadWithError:` — 要求を処理できませんでした。代理オブジェクトはエラーメッセージを受信します。
- `requestDidCancelLoad` — システム管理者による操作、ネットワーク障害、他の予期しないイベントなど、外部要因により要求がキャンセルされました。代理オブジェクトは戻り値を受信しません。
- `requestDidTimeout` — Salesforce サーバーが時間内に応答しませんでした。代理オブジェクトは戻り値を受信しません。

応答は、いずれかの代理メソッドへのコールで受信されます。応答種別は予測できないため、すべてのメソッドを実装する必要があります。Swift と Objective-C の代理での応答は、同じメソッド名とパラメーター名を使用します。

REST 代理メソッドの実装

成功応答

Swift	Objective-C
<code>request(request:didLoadResponse:[rawResponse:])</code>	<code>- (void)request:(SFRestRequest *)request didLoadResponse:(id)dataResponse rawResponse:(NSURLResponse *)rawResponse;</code>

`request:didLoadResponse:` メソッドは、成功条件を処理する唯一のプロトコルメソッドであるため、Salesforce データを処理するコードをこのメソッドに含めます。次に例を示します。

```
- (void)request:(SFRestRequest *)request
    didLoadResponse:(id)jsonResponse {
    NSArray *records = [jsonResponse objectForKey:@"records"];
    NSLog(@"request:didLoadResponse: #records: %d", records.count);
    self.dataRows = records;
    [self.tableView reloadData];
}
```

要求のデータ型をメソッドで推測できない場合は、イントロスペクションを使用してデータ型を判断します。次に例を示します。

```
if ([jsonResponse isKindOfClass:[NSArray class]]) {
    // Handle an NSArray here.
} else {
    // Handle an NSDictionary here.
}
```

応答を `NSDictionary` オブジェクトとして指定し、そのレコードを `NSArray` オブジェクトに抽出することができます。これを行うには、キー「records」を使用して `NSDictionary:objectForKey:` メッセージを送信します。

エラーによる失敗応答

Swift	Objective-C
<code>request(request:didFailLoadWithError:)</code>	<code>request:didFailLoadWithError:</code>

次のいずれかの条件で、`request:didFailLoadWithError:` コールバックがコールされます。

- 無効な要求パラメーターを使用すると、`kSFRestErrorDomain` エラーコードが返されます。たとえば、このエラーは、`requestForQuery:` に `nil` を渡したり、存在しないオブジェクトを更新しようとしたことを示している場合があります。
- OAuth アクセストークンが期限切れになると、フレームワークで新しいアクセストークンを取得しようとし、成功するとクエリが再試行されます。新しいアクセストークンまたはセッション ID の要求に失敗すると、`kSFOAuthErrorDomain` エラーコードが返されます。例: アクセストークンが期限切れであり、OAuth 更新トークンが無効である。このような状況になることはまずありません。
- 低レベルの HTTP 要求に失敗すると、`kSFIdentityErrorTypeBadHttpResponse` エラーコードが返されます。例: Salesforce サーバーが一時的にアクセスできなくなる。

`requestDidCancelLoad` および `requestDidTimeout` メソッド

Swift	Objective-C
<code>requestDidCancelLoad(request:)</code>	<code>requestDidCancelLoad:</code>
<code>requestDidTimeout(request:)</code>	<code>requestDidTimeout:</code>

「load canceled (読み込みキャンセル)」および「request timed out (要求タイムアウト)」代理メソッドは自己説明的であり、エラーコードは返されません。エラーメッセージの表示、ログへの書き込み、要求の再試行など、結果の処理方法を自由に選択できます。

Batch 要求および Composite 要求

一括および複合 API は、複数の要求を単一のコールで処理するため、特別な処理が必要です。Swift で Mobile SDK 拡張を使用すれば、これらの複雑な要求を簡単に作成して設定できます。

Batch 要求クラスと Composite 要求クラス

Swift

- `BatchRequest`
- `BatchRequestBuilder`
- `BatchResponse`
- `CompositeRequest`
- `CompositeRequestBuilder`
- `CompositeResponse`

Objective-C

- `SFSDKBatchRequest`
- `SFSDKBatchRequestBuilder`
- `SFSDKBatchResponse`
- `SFSDKCompositeRequest`
- `SFSDKCompositeRequestBuilder`
- `SFSDKCompositeResponse`

これらのクラスにより、Batch 要求と Composite 要求を容易に作成できます。これらのクラスを使用する手順は、次のとおりです。

1. ビルダーインスタンスを作成します。Batch 要求では、必要に応じて `haltOnError` プロパティを `true` に設定します。

```
let builder = BatchRequestBuilder()
// Optional; defaults to false
builder.setHaltOnError(true)
```

Composite 要求では、必要に応じて `allOrNone` ロールバックプロパティを `true` に設定します。

```
let builder = CompositeRequestBuilder()
// Optional; defaults to false
builder.setAllOrNone(true)
```

2. REST 要求を作成する場合、REST 要求はビルダーオブジェクトに追加します。

```
builder.add(request as! BatchRequest)
```

Composite 要求では、『[REST API 開発者ガイド](#)』の説明に従って、参照 ID も指定します。ベース `RestRequest` オブジェクトを追加して、参照 ID を明示的に指定します。

```
builder.add(request, referenceId)
```

または、`CompositeSubRequest` オブジェクトを追加します。このオブジェクトは、参照 ID を内部的に保存する `RestRequest` オブジェクトです。

```
builder.add(request as! CompositeSubRequest)
```

3. 準備が整ったら、ビルダーオブジェクトの `build` メソッド (`buildBatchRequest(_:)` または `buildCompositeRequest(_:)`) をコールします。次に例を示します。

```
let request = builder.buildBatchRequest(SFRestDefaultAPIVersion)
```

各 `build` メソッドは、特殊な `BatchRestRequest` または `CompositeRestRequest` オブジェクトを返します。これらのオブジェクトは、共有 `RestClient` インスタンスを通じて送信できます。

 **ヒント:** デフォルトの引数として古いバージョンの API を使用するには、「v42.0」形式のリテラル文字列を渡します。

4. Batch 要求および Composite 要求に対する応答は、応答クラス (`BatchResponse` または `CompositeResponse`) のインスタンスとして返されます。

関連トピック

- [『REST API 開発者ガイド』の「Batch」](#)
- [『REST API 開発者ガイド』の「Composite」](#)

REST 要求メソッドの使用

Salesforce には、Salesforce Platform API とは関係ない製品固有の REST API も一部含まれています。そのような要求にも Mobile SDK リソースを使用して、設定と送信を行うことができます。その場合は、ファクトリーメソッドを利用するのではなく、`RestRequest` オブジェクトを直接作成して設定します。

`RestClient` を使用して非 SOQL および非 SOSL REST の要求を送信する手順は、次のとおりです。

1. `RestRequest` のインスタンスを作成します。
2. `RestRequest` オブジェクトで、必要なプロパティを設定します。
3. `RestClient` オブジェクトで `send` をコールし、作成した `RestRequest` オブジェクトを最初のパラメーターとして渡します。

次の例は、GET 操作を実行して、特定の Chatter フィードにすべての項目を取得します。

```
let request = RestRequest(method: .GET, serviceHostType: .instance,
    path: "/v42.0/chatter/feeds/user-profile/me/feed-elements",
    queryParams: nil)
RestClient.shared.send(request: request { [weak self] (result) in
    switch result {
        case .success(let response):
            // do something with response
        case .failure(let error):
            SalesforceLogger.d(RootViewController.self, message:"Error invoking:
\\(request) , \\(error)")
    }
})
```

4. パラメーターで要求を実行するには、`SFJsonUtils.object(fromJSONString:)` を使用してパラメーター文字列をラップし、`RestRequest` の `queryParams` プロパティに割り当てます。カスタム要求を送信するには、`Dictionary` オブジェクトを作成し、`RestRequest` の `setCustomRequestBodyData(_:contentType:)` メソッドを使用します。

次の例では、カスタムリクエストボディを使用して、Chatter フィードにコメントを追加します。

```
let dataString = "{\"body\" : {\"messageSegments\" : [{\"type\" : \"Text\", \" +
    \"text\" : \"Some Comment\"}]}, \"feedElementType\":\"FeedItem\", \" +
    \"subjectId\":\"me\"}"

if let data = dataString.data(using: .utf8) {

    let request = RestRequest(method: .POST, serviceHostType: .instance,
        path: "/v42.0/chatter/feed-elements", queryParams: nil)
    request.setCustomRequestBodyData(data, contentType: "application/json")

    RestClient.shared.send(request: request { [weak self] (result) in
        switch result {
            case .success(let response):
                // do something with response
            case .failure(let error):
                SalesforceLogger.d(RootViewController.self, message:"Error invoking:
\\(request) , \\(error)")
        }
    })
}
```

5. 要求の HTTP ヘッダーを設定するには、`setHeaderValue(_:forHeaderName:)` メソッドを使用します。このメソッドは、HTML 表示用に事前にエンコードされている Chatter フィードを表示する場合に役立ちま

す。Chatter コメントに不要なエスケープシーケンスが表示されないようにするには、要求の X-Chatter-Entity-Encoding ヘッダーを「false」に設定します。

```
...
request.setHeaderValue("false", forHeaderName:"X-Chatter-Entity-Encoding")
```

認証されていない REST 要求

Mobile SDK アプリケーションの大部分の REST 要求は、認証された Salesforce ユーザーの代わりに安全な Salesforce エンドポイントに移動します。その場合、Mobile SDK は、現在のユーザーの OAuth トークンを要求に埋め込み、古いトークンを自動的に更新して、アプリケーションの認証を処理します。

ただし、Mobile SDK アプリケーションは、認証なしでネットワークサービスを要求することがあります。次に例を示します。

- ユーザーが Salesforce にログインする前
- Salesforce 内の認証されていないエンドポイントを経由
- Salesforce 外の認証されていないエンドポイントを経由

Mobile SDK が提供する REST クライアントには、次の 2 つのバージョンがあります。

- `RestClient.shared` — 現在のユーザーのログイン情報で初期化されます。
- `RestClient.sharedGlobal` — ユーザーで初期化されません。

認証済みおよび未認証のエンドポイントにアクセスするには、`RestClient.shared` インスタンスを使用します。ただし、`RestClient.sharedGlobal` インスタンスを使用した場合は、アクセスできるのが未認証のエンドポイントにのみとなります。認証されていないエンドポイントにアクセスする方法は、サービスが Salesforce ドメインに含まれているかどうかによって異なります。

 **メモ:** 認証されていない REST 要求では、フルパスの URL が必要です。Mobile SDK では、認証されていないエンドポイントの先頭にインスタンス URL は追加されません。

認証されていない Salesforce リソースの要求

未認証のエンドポイントは、OAuth トークンが不要なエンドポイントです。認証されていない Lightning Platform API はほとんどありませんが、その他の製品や独自の Apex エンドポイントは、認証されている可能性があります。認証されていない Salesforce リソースの要求を設定するには、その `requiresAuthentication` プロパティを `false` または `NO` に設定します。

Swift

```
request.requiresAuthentication = false
```

Objective-C

```
request.requiresAuthentication = NO;
```

認証されていない外部リソースの要求

認証されていない外部エンドポイントへの要求をサポートするために、Mobile SDK には、その REST クライアントの共有グローバルインスタンスが用意されています。この REST クライアントでは、OAuth 認証が必要なく、

ユーザーの概念が認識されません。ネイティブアプリケーションは、これを使用して、ユーザーが Salesforce にログインする前または後に、認証されていないカスタム要求を Salesforce 以外のエンドポイントに送信できます。

共有グローバル REST クライアントにアクセスする方法は、次のとおりです。

Swift:

```
RestClient.sharedGlobal
```

Objective-C:

```
[[SFRestAPI sharedInstance]
```

外部エンドポイントをコールするには、

`RestRequest.customURLRequest(with:baseUrl:path:queryParams:)` メソッドを使用します。

Swift

```
let request =
    RestRequest.customURLRequest(with: RestRequest.Method.GET,
                                baseUrl: "https://api.github.com",
                                path: "/orgs/forcedotcom/repos",
                                queryParams: nil)
request.requiresAuthentication = false
RestClient.sharedGlobal.send(request: request,
                              onFailure: {(error, rawResponse) in

}) {(response, rawResponse) in

}
```

Objective-C

```
SFRestRequest *request =
    [SFRestRequest customURLRequestWithMethod:SFRestMethodGET
                                baseUrl:@"https://api.github.com"
                                path:@" /orgs/forcedotcom/repos"
                                queryParams:nil];
request.requiresAuthentication = NO;
[[SFRestAPI sharedInstance] sendRESTRequest:request
 failBlock:^(NSError * e, NSURLResponse * rawResponse) {

} completeBlock:^(id resp, NSURLResponse * rawResponse) {
    NSDictionary *response = resp;
    // response should have forcedotcom repos
}];
```

ユーザーが Salesforce で認証したら、アプリケーションは認証済み REST クライアントに切り替えることができます。認証済みインスタンスを介して任意のエンドポイントをコールできますが、グローバルインスタンスを介して認証済みエンドポイントをコールすることはできません。

非ネイティブアプリケーションでの認証されていないリソースの要求

非ネイティブアプリケーション (ハイブリッドおよび React Native) の場合、Mobile SDK では REST クライアントオブジェクトはサポートされません。非ネイティブアプリケーションで認証されていないエンドポイントにアク

セスするには、次のいずれかのメソッドをコールし、その `doesNotRequireAuthentication` パラメーターに `false` を渡します。

Swift

```
function anyrest(fullUrlPath, returnsBinary, doesNotRequireAuthentication,
params, successHandler, errorHandler)
```

React Native

```
sendRequest(endPoint, path, successCB, errorCallback, method, payload, headerParams,
fileParams, returnBinary, doesNotRequireAuthentication)
```

SFRestAPI (Blocks) カテゴリ

REST API 応答を受信して処理する場合、代行クラスの代わりにインラインコードブロックを使用できます。この代替的な Objective-C アプローチでは、要求の送信とその非同期応答の処理を 1 回のメソッドコールで実行できます。

ネイティブ iOS の Mobile SDK には、1 つの要求、Composite 要求、Batch 要求のブロックメソッドが用意されています。これらのメソッドを使用するときは、成功応答と失敗応答を処理するブロック引数を指定します。Mobile SDK は、応答のネットワーク状況に応じて、非同期応答を成功ブロックまたは失敗ブロックに転送します。

ブロックメソッドと関連する typedef は、[SFRestAPI \(Blocks\)](#) カテゴリで次のとおり定義されます。

```
// Sends a single request you've already built, using blocks to return status.
- (void) sendRequest:(SFRestRequest *)request
    failureBlock:(SFRestRequestFailBlock) failureBlock
    successBlock:(SFRestResponseBlock) successBlock
    NS_REFINED_FOR_SWIFT;

// Sends a composite request you've already built, using blocks to return status.
- (void) sendCompositeRequest:(SFSDKCompositeRequest *)request
    failureBlock:(SFRestRequestFailBlock) failureBlock
    successBlock:(SFRestCompositeResponseBlock) successBlock
    NS_REFINED_FOR_SWIFT;

// Sends a batch request you've already built, using blocks to return status.
- (void) sendBatchRequest:(SFSDKBatchRequest *)request
    failureBlock:(SFRestRequestFailBlock) failureBlock
    successBlock:(SFRestBatchResponseBlock) successBlock
    NS_REFINED_FOR_SWIFT;
```

各 `send` メソッドに次の 2 つのブロックが必要です。

失敗ブロック

失敗ブロックは、タイムアウト、キャンセル、エラー障害などを受信する可能性があります。

- ブロック種別:

```
typedef void (^SFRestRequestFailBlock) (id _Nullable response,
    NSError * _Nullable e, NSURLResponse * _Nullable rawResponse)
    NS_SWIFT_NAME (RestRequestFailBlock);
```

- 応答種別: `NSError`

成功ブロック

ブロック種別と応答種別は、次のとおり要求種別によって異なります。

1つの要求

- 応答種別: 返されるデータに応じて、NSArray または NSData オブジェクト。
- ブロック種別 (要求の戻り値のデータ型に適したテンプレートを使用):

```
typedef void (^SFRestDictionaryResponseBlock) (NSDictionary * _Nullable dict,
    NSURLResponse * _Nullable rawResponse)
    NS_SWIFT_NAME(RestDictionaryResponseBlock);

// Use this block when you request API versions
typedef void (^SFRestArrayResponseBlock) (NSArray * _Nullable arr,
    NSURLResponse * _Nullable rawResponse)
    NS_SWIFT_NAME(RestArrayResponseBlock);
```

Composite Request

- ブロック種別:

```
typedef void (^SFRestCompositeResponseBlock) (SFSDKCompositeResponse *response,
    NSURLResponse * _Nullable rawResponse)
    NS_SWIFT_NAME(RestCompositeResponseBlock);
```

- 応答種別: [SFSDKCompositeResponse](#)

Batch Request

- ブロック種別:

```
typedef void (^SFRestBatchResponseBlock) (SFSDKBatchResponse *response,
    NSURLResponse * _Nullable rawResponse) NS_SWIFT_NAME(RestBatchResponseBlock);
```

- 応答種別: [SFSDKBatchResponse](#)

ブロックを使用した要求の送信

ブロックメソッドを使用して要求を送信する手順は、次のとおりです。

1. 各自のニーズに適した SFRestRequest オブジェクトを作成します。

- 1つの要求の場合:
 - 適切な [SFRestAPI ファクトリーメソッド](#) (ページ 598) をコールして、SFRestRequest オブジェクトを作成します。
 - 要求を送信します。次の例は、NSDictionary 応答を返す要求を処理します。

```
- (void)sendRequestForDictionary:(SFRestRequest *)request {
    // Block to handle failure
    SFRestRequestFailBlock failBlock = ^(id response, NSError *error, NSURLResponse
    *rawResponse) {
        // Do as you wish with the error information
        //...
    };
```

```

// Block to handle success
SFRestDictionaryResponseBlock completeBlock = ^(NSDictionary *data,
NSURLResponse *rawResponse) {
    // Process the dictionary data
    //...
};

// Send the request to Salesforce along with your success and failure blocks
[[SFRestAPI sharedInstance] sendRequest:request
                             failureBlock:failBlock
                             successBlock:completeBlock];

...
}

```

- Batch 要求と Composite 要求の場合:

- a. サブ要求ごとに、適切なファクトリーメソッドをコールして、`SFRestRequest` オブジェクトを作成します。
- b. Salesforce に送信する `SFRestRequest` オブジェクトを作成するには、サブ要求オブジェクトの配列をバッチまたは複合ファクトリーメソッドに渡します。詳細は、[「SFRestAPI.h」](#) を参照してください。

```

// Batch request factory method
- (SFRestRequest *) batchRequest:(NSArray<SFRestRequest *> *)requests
                             haltOnError:(BOOL)haltOnError
                             apiVersion:(nullable NSString *)apiVersion;

// Composite request factory method
- (SFRestRequest *) compositeRequest:(NSArray<SFRestRequest *> *) requests
                                refIds:(NSArray<NSString *> *) refIds
                                allOrNone:(BOOL)allOrNone
                                apiVersion:(nullable NSString *)apiVersion;

```

2. 期待するデータを処理する成功ブロックを設計します。
3. 問題を適切に処理する失敗ブロックを設計します。
4. ステップ1で作成した `SFRestRequest` オブジェクトおよび成功ブロックと失敗ブロックを適切な `send` ブロックメソッドに渡します。

メモ:

- Swift の場合、Mobile SDK で、REST 応答が 1 つの Swift 完了ブロッククロージャに送り込まれるように Objective-C メソッドが微調整されます。[「REST 応答の処理」](#) (ページ 101) を参照してください。
- Objective-C で、ブロックおよび代行メソッドを適切に使用することで、アプリケーションの読みやすさを調整し、メンテナンスを簡単にするのに役立ちます。ブロックの使用に最適な条件は、C++ でのインライン関数や Java での匿名関数を要求する条件と一致することがよくあります。最終的には、各自の要件に基づいて判断します。

SFRestAPI (QueryBuilder) カテゴリ

SOQL クエリまたは SOSL 検索の正しい構文が明確でない場合は、SFRestAPI (QueryBuilder) カテゴリメソッドを利用できます。これらのメソッドでは、指定した基本条件からクエリ文字列が作成され、書式設定された文字列が返されます。返された文字列を SFRestAPI 要求メソッドまたは RestClient 要求メソッドに渡し、クエリや検索に使用できます。

SFRestAPI (QueryBuilder) には、SOQL クエリと SOSL 検索にそれぞれ2つの静的メソッドがあります。1つは最小限のパラメーターを受け入れ、もう1つはすべてのオプションを受け入れます。Swift バージョンのこれらのメソッドは、Mobile SDK で明示的に定義されていません。Swift でこれらのメソッドをコーディングするには、Xcode コンパイラーで提供されるオートコンプリート候補表示機能を使用します。これらのメソッドおよびパラメーター名の候補は、Swift コンパイラーによって決まり、Objective-C とは異なる可能性があります。

SOSL メソッド

SOSL クエリビルダーメソッドは、次のとおりです。

Swift (コンパイラー生成)

```
RestClient.soslSearch(withSearchTerm:objectScope:)
RestClient.soslSearch(withSearchTerm:fieldScope:objectScope:limit:)
```

Objective-C

```
+ (NSString *) SOSLSearchWithSearchTerm:(NSString *)term
                        objectScope:(NSDictionary *)objectScope;

+ (NSString *) SOSLSearchWithSearchTerm:(NSString *)term
                        fieldScope:(NSString *)fieldScope
                        objectScope:(NSDictionary *)objectScope
                        limit:(NSInteger)limit;
```

SOSL 検索メソッドのパラメーターは、次のとおりです。

Swift	Objective-C
withSearchTerm	term

検索文字列。この文字列には任意の値を指定できます。メソッドでは、検索を処理する前に SOSL 予約文字がすべてエスケープされます。

Swift	Objective-C
fieldScope	fieldScope

検索する項目を示します。これは、nil またはいずれかの IN 検索グループ式 (「IN ALL FIELDS」、 「IN EMAIL FIELDS」、 「IN NAME FIELDS」、 「IN PHONE FIELDS」、または 「IN SIDEBAR FIELDS」) になります。デフォルトでは nil の値は 「IN NAME FIELDS」 になります。 [「Salesforce Object Search Language \(SOSL\)」](#) を参照してください。

Swift	Objective-C
objectScope	objectScope

検索するオブジェクトを指定します。有効な値は次のとおりです。

- nil — 範囲に制限はありません。検索可能なすべてのオブジェクトが検索されます。
- NSDictionary オブジェクトポインター — SOSL RETURNING fieldspec に対応します。各キーは sObject 名です。各値は、項目リストおよびキーオブジェクトの省略可能な WHERE、ORDER BY、LIMIT 句を含む文字列です。

NSDictionary オブジェクトを使用する場合、各値には少なくとも項目リストが含まれている必要があります。たとえば、ディクショナリエントリで次の SOSL ステートメントを表すとしてします。

```
FIND {Widget Smith}
IN Name Fields
RETURNING Widget__c (name Where createddate = THIS_FISCAL_QUARTER)
```

この場合は、キーを "Widget__c" に設定し、値を "name WHERE createddate = "THIS_FISCAL_QUARTER" に設定します。次に例を示します。

```
[SFRestAPI
    SOSLSearchWithSearchTerm:@"all of these will be escaped:~{}"
    objectScope:[NSDictionary
        dictionaryWithObject:@"name WHERE
            createddate="THIS_FISCAL_QUARTER"
            forKey:@"Widget__c"]];
```

- NSNull — 範囲が指定されていません。

Swift	Objective-C
limit	limit

返される結果数を制限するには、このパラメーターを、受信する結果の最大数に設定します。

SOQL メソッド

SOQL 文字列を作成する SOQL QueryBuilder メソッドは、次のとおりです。

Swift (コンパイラー生成)

```
RestClient.soqlQuery(withFields:sObject:whereClause:limit:)
RestClient.soqlQuery(withFields:sObject:whereClause:groupBy:having:orderBy:limit:)
```

Objective-C

```
+ (NSString *) SOQLQueryWithFields:(NSArray *)fields
    sObject:(NSString *)sObject
    where:(NSString *)where
    limit:(NSInteger)limit;
```

```
+ (NSString *) SOQLQueryWithFields:(NSArray *)fields
                        sObject:(NSString *)sObject
                        where:(NSString *)where
                        groupBy:(NSArray *)groupBy
                        having:(NSString *)having
                        orderBy:(NSArray *)orderBy
                        limit:(NSInteger)limit;
```

SOQL メソッドのパラメーターは、SOQL クエリの構文に対応します。withFields/fields と sObject を除くすべてのパラメーターは、nil に設定することができます。

Swift	Objective-C
withFields	fields

照会する項目名の配列。

Swift	Objective-C
sObject	sObject

照会するオブジェクトの名前。

Swift	Objective-C
whereClause	where

1 つ以上のクエリ条件を指定する式。

Swift	Objective-C
groupBy	groupBy

結果レコードのグループ化に使用する項目名の配列。

Swift	Objective-C
having	having

グループ化した結果を絞り込むための式 (通常、集計関数を使用)。groupBy でのみ使用されます。

Swift	Objective-C
orderBy	orderBy

結果レコードの並び替えに使用する項目名の配列。

Swift	Objective-C
limit	limit

返されるレコードの最大数。

「[SOQL SELECT Syntax](#)」を参照してください。

SOSL のサニタイズ

QueryBuilder カテゴリには、SOSL 検索語をクリーンアップするためのクラスメソッドも用意されています。

Swift

```
RestClient.sanitizeSOSLSearchTerm(_:)
```

Objective-C

```
+ (NSString *) sanitizeSOSLSearchTerm:(NSString *)searchTerm;
```

このメソッドでは、入力文字列に含まれているすべての SOSL 予約文字がエスケープされ、エスケープされたバージョンが返されます。次に例を示します。

Swift

```
let cleanSOSL = RestClient.sanitizeSOSLSearchTerm(_: "FIND {MyProspect}")
```

Objective-C

```
NSString *soslClean = [SFRestAPI sanitizeSOSLSearchTerm:@"FIND {MyProspect}"];
```

このコールは、“FIND \{MyProspect\}” を返します。

sanitizeSOSLSearchTerm メソッドは、SOSL および SOQL QueryBuilder メソッドの実装でコールされるため、QueryBuilder メソッドに渡す文字列でこのメソッドをコールする必要はありません。ただし、独自のクエリを手動で作成しているなどの場合は、このメソッドを使用できます。SOSL 予約文字は、次のとおりです。

```
\?&|!{}[]()^~*:'"+-
```

SFRestAPI (Files) カテゴリ

SFRestAPI (Files) カテゴリには、ファイル操作要求を生成するメソッドが備えられています。各メソッドは、新しい RestRequest または SFRestRequest オブジェクトを返します。アプリケーションは、このオブジェクトを Salesforce サービスに送信して要求を処理します。

関連トピック

- [ファイルとネットワークング](#)。
- <https://forcedotcom.github.io/SalesforceMobileSDK-iOS/Documentation/SalesforceSDKCore/html/Classes/SFRestAPI.html> の「Files Methods (Files メソッド)」。
- Files REST リソースの説明については、[Connect REST API Resources \(REST API リソースの接続\)](#) > [Files Resources \(ファイルリソース\)](#) (<http://www.salesforce.com/us/developer/docs/chatterapi>) を参照してください。

認証エラーの処理

Swift	Objective-C
UserAccountManager	SFUserAccountManager
UserAccountManagerDelegate	SFUserAccountManagerDelegate

Mobile SDK には、認証エラーが発生したときにメッセージを表示し、アプリケーションフローを迂回させるデフォルトのエラーハンドラーが備えられています。エラーイベントでは、ユーザーアカウントマネージャーは代理を反復処理し、それらがエラーを処理する機会を与えます。

このチェーンに独自のクラスを挿入するには、代理プロトコルを実装し、次のメソッドを上書きします。

Swift:

```
func userAccountManager(_ accountManager: UserAccountManager,
                        didFailAuthenticationWith: Error,
                        info: AuthInfo) -> Bool
{
    // Provide custom error handling
    //...
    return true
}
```

Objective-C:

```
/**
 *
 * @param userAccountManager The instance of SFUserAccountManager
 * @param error The Error that occurred
 * @param info The info for the auth request
 * @return YES if the error has been handled by the delegate.
 * SDK will attempt to handle the error if the result is NO.
 */
- (BOOL)userAccountManager:(SFUserAccountManager *)userAccountManager
    error:(NSError*)error
    info:(SFOAuthInfo *)info {
    // Provide custom error handling
    //...
    return YES;
}
```

戻り値 `true` または `YES` は、メソッドが現在のエラー状況を処理したことを示します。この場合、ユーザーアカウントマネージャーはこのエラーに対してそれ以上のアクションを実行しません。それ以外の場合、代理はこのエラーを処理せず、エラー処理プロセスはリスト内の次の代理に進みます。すべての代理が `false` を返すと、ユーザーアカウントマネージャーは独自のエラーハンドラーを使用します。

 **メモ:** 認証エラー処理の場合、過去には Mobile SDK は `SFAuthenticationManager` 共有オブジェクト内の `SFAuthErrorHandler` オブジェクトのカスタマイズ可能なリストを使用していました。現在は、`SFAuthenticationManager` は非推奨になりました。認証エラーハンドラーリストをカスタマイズしている場合は、`UserAccountManagerDelegate` (Swift) または `SFUserAccountManagerDelegate` (Objective-C) を使用するようにコードを更新してください。

Mobile SDK アプリケーションでの iPadOS のサポート

Mobile SDK 9.0 では iPad のカスタマーエクスペリエンスが強化されています。横モードが正常に機能するようになり、Mobile SDK でマルチウィンドウがサポートされるようになりました。

Swift	Objective-C
AuthHelper	SFSDKAuthHelper

iPad 用の Mobile SDK アプリケーションでは、明示的にマルチウィンドウを有効にする必要はありませんが、この機能を使用するには、SceneDelegate コードの変更が必要です。AuthHelper の新しいメソッドでは、同名の既存メソッドに UIWindowScene パラメーターが加えられています。Mobile SDK では、シーンによってログインフローを明確化できるため、ユーザーのフォーカスの移動に合わせて SDK オブジェクトを調整できます。あるシーンでユーザーがログインまたはログアウトした場合、その変更はすべてのシーンに適用されます。

`loginIfRequired(_:completion:)` および `handleLogout(_:completion:)` メソッドに completion コールバックブロックを定義することで、それぞれのケースを事後に処理できます。

実行時の新しいメソッドを確認するには、[RestAPIExplorer](#) サンプルアプリケーションの更新された SceneDelegate クラスを調べてください。AuthHelper クラスの `registerBlock(forCurrentUserChangeNotifications:)` および `loginIfRequired(_:completion:)` メソッドでは、UIWindowScene 引数を使用できるようになりました。

```
func scene(_ scene: UIWindowScene, willConnectTo session: UIWindowSession, options
connectionOptions: UIWindowScene.ConnectionOptions) {
    guard let windowScene = (scene as? UIWindowScene) else { return }
    self.window = UIWindow(frame: windowScene.coordinateSpace.bounds)
    self.window?.windowScene = windowScene

    AuthHelper.registerBlock(forCurrentUserChangeNotifications: scene) {
        self.resetViewState {
            self.setupRootViewController()
        }
    }
    self.initializeAppViewState()

    AuthHelper.loginIfRequired(scene) {
        self.setupRootViewController()
    }
}
```

このサンプルアプリは、`loginIfRequired(_:completion:)` の completion で、シーンを最初の状態にリセットするコードをコールします。この場合は RootViewController のインスタンスをコールします。

`registerBlock(forCurrentUserChangeNotifications:completion:)` の完了ブロックは、最初に発信ユーザーのビュースタックを破棄し、シーンを RootViewController にリセットします。マルチウィンドウをサポートするには、これらの2つのコールを Swift アプリケーションの SceneDelegate インスタンスの `scene(_:willConnectTo:_:)` メソッドにコピーします。既存の `AuthHelper.registerBlock(_:)` のコールをシーン対応のものに置き換えます。

IDP アプリケーション

AppDelegate と同様に、SceneDelegate は URL を開くためのメソッドを提供します。マルチウィンドウをサポートするには、Mobile SDK IDP クライアントアプリケーションではシーンの代行を使用し、シーンの永続的な識別子を渡す必要があります。

```
func scene(_ scene: UIScene, openURLContexts URLContexts:
    Set<UIOpenURLContext>) {
    if let urlContext = URLContexts.first {
        UserAccountManager.shared.handleIdentityProviderResponse(
            from: urlContext.url,
            with: [UserAccountManager.IDPSceneKey:
                scene.session.persistentIdentifier])
    }
}
```

AuthHelper シーン対応メソッド

次の AuthHelper メソッドは、既存のメソッドに UIScene パラメーターを追加する新しいオーバーロードです。

Swift

```
func loginIfRequired(_ scene: UIScene, completion completionBlock: (() -> Void)?)
func handleLogout(_ scene: UIScene, completion completionBlock: (() -> Void)?)
func registerBlock(forCurrentUserChangeNotifications scene: UIScene completion
    completionBlock: () -> Void)
func registerBlock(forLogoutNotifications scene:UIScene, completion completionBlock: ()
    -> Void)
```

Objective-C

```
+ (void)loginIfRequired:(UIScene *)scene completion:
    (nullable void (^)(void))completionBlock;
+ (void)handleLogout:(UIScene *)scene completion:
    (nullable void (^)(void))completionBlock;
+ (void)registerBlockForCurrentUserChangeNotifications:
    (UIScene *)scene completion:(void (^)(void))completionBlock;
+ (void)registerBlockForLogoutNotifications:
    (UIScene *)scene completion:(void (^)(void))completionBlock;
```

Mobile SDK アプリケーションでの Catalyst のサポート

バージョン 9.1 以降、Mobile SDK のネイティブフレームワークには Catalyst サポートが追加されています。Catalyst は、iPad アプリケーションを Intel Mac およびシリコン Mac で実行できるようにする Apple のクロスプラットフォーム製品です。Mobile SDK iPad アプリケーションを macOS 上で実行するための準備は簡単です。通常は、iOS プロジェクト設定でいくつかの追加設定を行うだけです。

Mobile SDK が Catalyst を通じて macOS をサポートできるのは、次のフレームワークの場合です。

- SalesforceSDKCommon
- SalesforceAnalytics
- SalesforceSDKCore
- SmartStore
- MobileSync

次の Mobile SDK フレームワークでは、Catalyst はサポートされません。

- SalesforceHybridSDK
- SalesforceReact
- SalesforceFileLogger フレームワーク (現在開発中)

 **メモ:** macOS 上で実行するには、アプリケーションで使用されているすべてのサードパーティフレームワークも Catalyst をサポートしている必要があります。

Catalyst の使用開始

iPad アプリケーションを macOS 用に設定するには、Apple の「[Turning on Mac Catalyst \(Mac Catalyst 上での調整\)](#)」チュートリアルに従います。アプリケーションが Mac バージョンの場合は、リリース対象が macOS 11 に設定されていることを確認します。

Mobile SDK では、Catalyst サポートを説明する例も用意されています。

MobileSyncExplorerSwift テンプレートアプリケーション

このテンプレートを `forceios createwithtemplate` コマンドとともに使用して、Catalyst 対応プロジェクトを作成できます。

RestAPIExplorer サンプルアプリケーション

Mobile SDK のセキュリティ機能に関する特別な注意事項

Mobile SDK アプリケーションで次のセキュリティ機能が使用されている場合は、Catalyst に関して特別な考慮が必要です。

パスコード

パスコードは、iPhone や iPad といったモバイルデバイスのセキュリティゲートとして役立ちます。ただし、アプリケーションを Mac 上で実行する場合、パスコードが意味をなすかどうかの判断は、ユーザーにゆだねられています。Mobile SDK のパスコードサポートは、Mac 上でも依然として使用可能で正常に機能します。

スナップショット

スナップショットは、モバイルデバイスに不可欠ですが、その理由は Mac には当てはまりません。アプリケーションが Mac バージョンの場合、Mobile SDK は、この機能を自動的に無効にします。

Mobile SDK での iOS アプリケーション拡張の使用

開発者は、iOS アプリケーション拡張により、アプリケーションの機能をアプリケーションウィンドウ外に拡張できます。Mobile SDK では、設定をわずかに加えるだけでアプリケーション拡張がサポートされます。

Mobile SDK アプリケーションの iOS アプリケーション拡張について

iOS アプリケーション拡張は、プロジェクトの個別のターゲットです。独自のフォルダーに存在し、個別のバイナリユニットとしてアプリケーションバンドルに追加されます。ただし、アプリケーション拡張では、メインアプリケーションと同じリソースとライブラリにアクセスできます。developer.apple.com のアプリケーションの拡張に関する記事で説明されているように、Apple では多数の拡張の種類が用意されています。

拡張を有効にするには、次の 2 つの領域で特別な設定を追加します。

- **プロジェクト設定:** アプリケーション拡張を追加すると、Xcode によってアプリケーション拡張構築ターゲットが作成されます。既存のメインターゲットおよび新しい拡張ターゲットが、同じアプリケーショングループおよびキーチェーン共有グループに結合されるように設定します。両方のアプリケーションで同じプロビジョニングプロファイルにサインインする必要があります。
- **アプリケーションコード:** 実行時に、2 つのアプリケーションで同じアプリケーショングループ情報を使用する必要があり、メインアプリケーションではユーザー認証ステータスを拡張に伝える必要があります。

すべてを適切に設定すると、アプリケーション拡張で、拡張タイプに適した Mobile SDK コードを実行できるようになります。

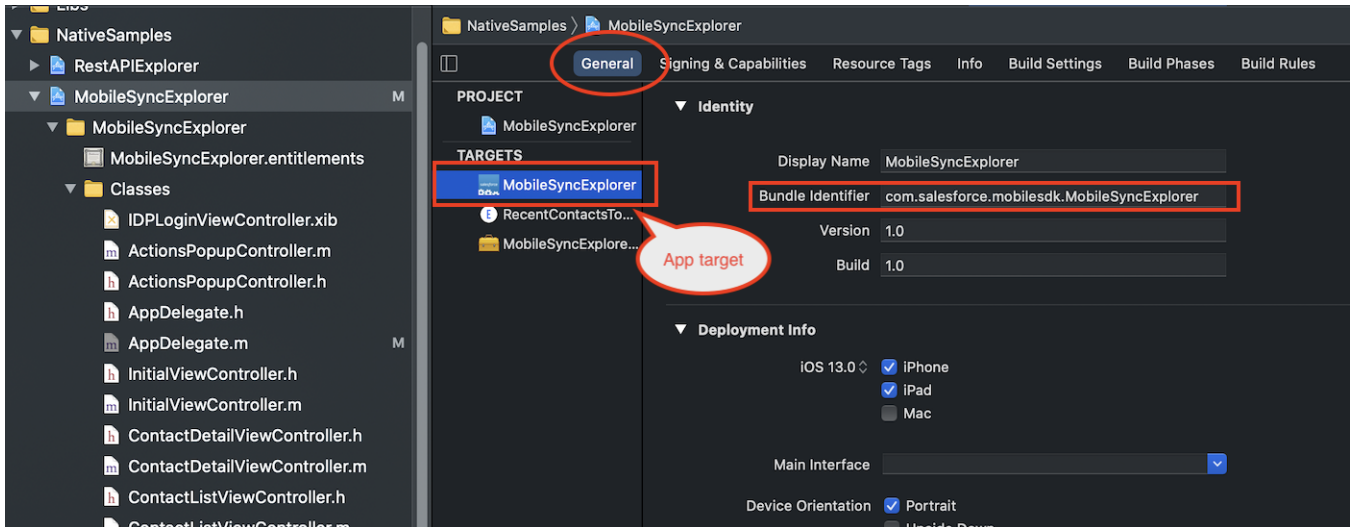


メモ: アプリケーション拡張は物理デバイスでのみ確実にテストできます。シミュレーターテストでは予測できない結果が得られる場合があります。

プロジェクト設定

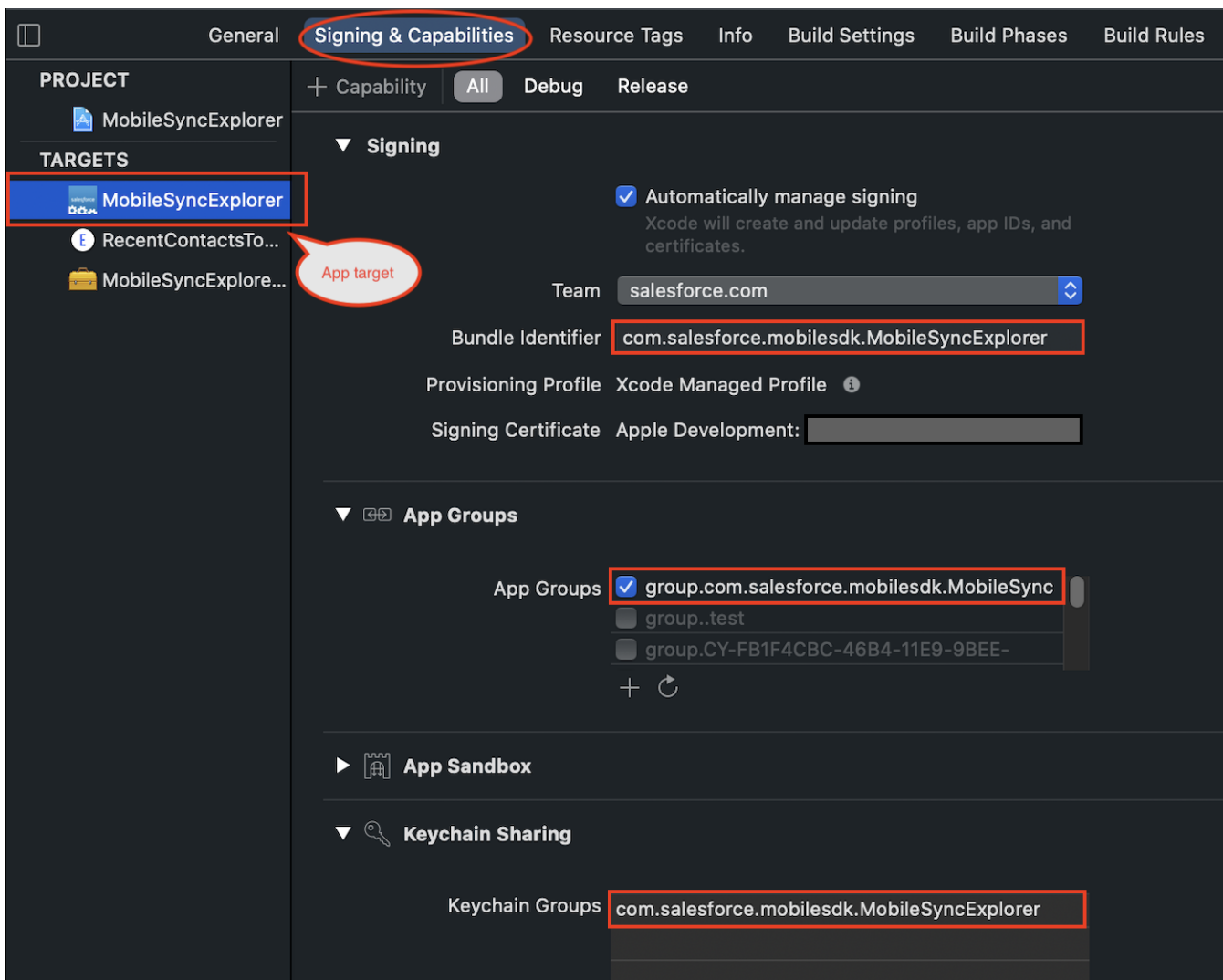
1. Mobile SDK アプリケーションで、Apple 開発者向けドキュメントの説明に従って拡張ターゲットを作成します。この手順の処理方法と拡張の種類は、iOS との関係で決まります。
2. ターゲットの作成後、Xcode プロジェクトナビゲーターで Mobile SDK ワークスペースの最上位ノードを選択します。この手順によって、プロジェクトエディターでプロジェクトが開きます。
3. 両方のアプリケーションと拡張ターゲットで、同じ開発者チームにサインインします。

4. [General (全般)] をクリックし、アプリケーションターゲットの一意のバンドル識別子を指定します。Xcode 12.4 では次のようになります。

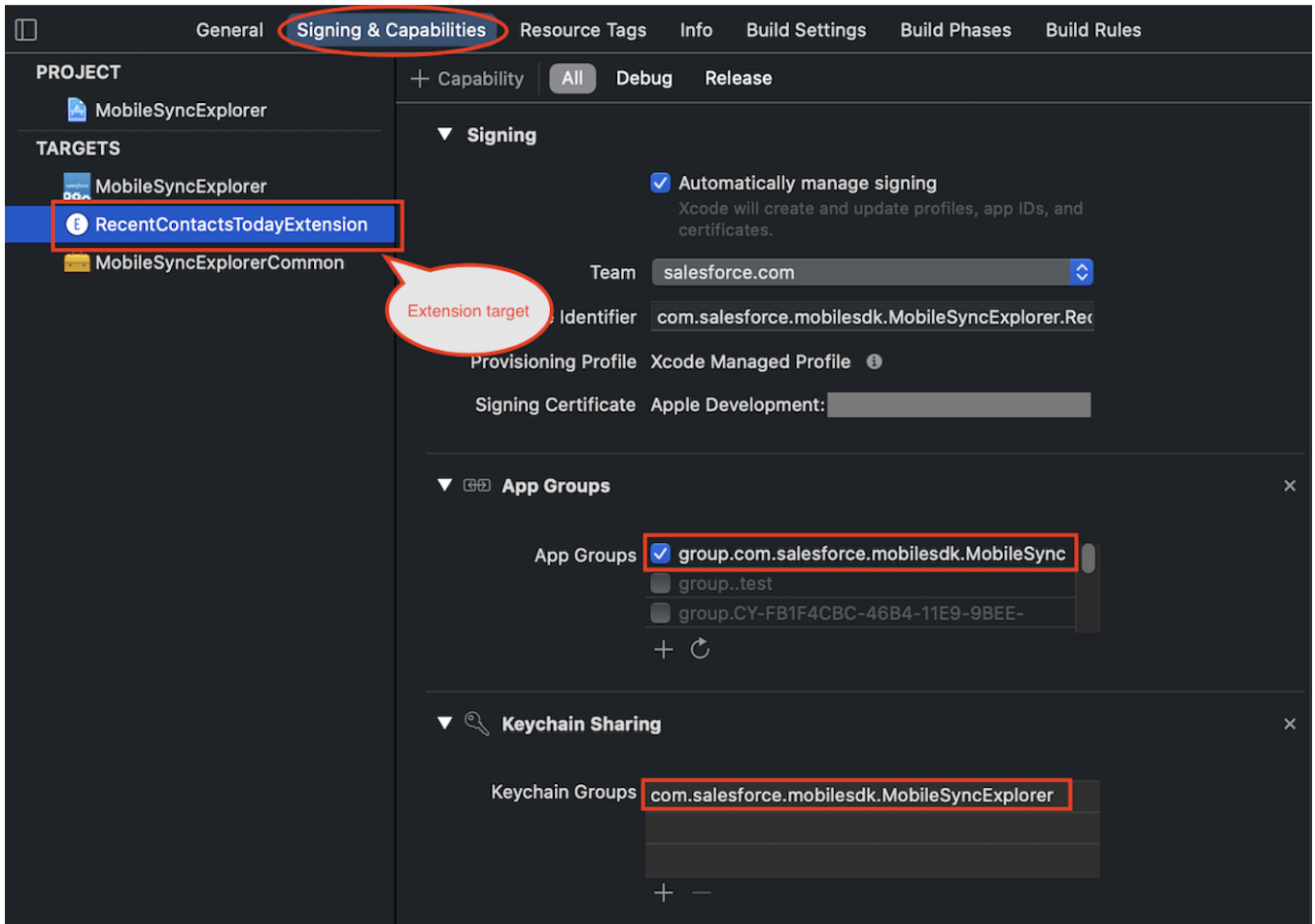


5. 拡張ターゲットにバンドル識別子の手順を繰り返します。この識別子も一意にする必要があります。
6. アプリケーション設定で、Mobile SDK アプリケーションターゲットを選択し、[Capabilities (機能)] をクリックします。
7. アプリケーションターゲットの [App Groups (アプリケーショングループ)] と [Keychain Sharing (キーチェーン共有)] をオンに切り替えます。
8. [App Groups (アプリケーショングループ)] で、アプリケーショングループを選択するか作成します。「group.myapp.shared」のように、アプリケーションを識別する、一意の表示ラベルを使用します。

9. [Keychain Sharing (キーチェーン共有)] で、キーチェーングループを選択するか作成します。
「com.myapp.MyApp」のように、アプリケーションを識別する、一意の表示ラベルを使用します。



10. 拡張ターゲットに、アプリケーショングループとキーチェーン共有の手順を繰り返します。拡張ターゲットの2つの値は、アプリケーションターゲットで対応する値と正確に一致する必要があります。



アプリケーショングループ設定

iOS アプリケーション拡張を forceios アプリケーションに組み込むときは、アプリケーショングループで作業していることを Mobile SDK に指示するコードを追加します。これらの行は、メインアプリケーションの AppDelegate クラスとアプリケーション拡張のビューコントローラーまたはシーンの代行クラスの2か所に追加します。

Mobile SDK には、必須プロパティを保存するユーティリティクラス SFSDKDatasharingHelper が用意されています。メインアプリケーションとアプリケーション拡張でこれらのプロパティに設定する値は同じである必要があります。

ユーザーが Salesforce にログインしようとした後で、拡張コードでアクセスできる場所に認証ステータスを保存します。そのためには、スイート名がアプリケーショングループ名に一致する UserDefaults 辞書を使用します。Mobile SDK には、このような辞書が提供される UserDefaults (SFAdditions) カテゴリが用意されています。このカテゴリにより、シングルトン SFSDKDatasharingHelper オブジェクトの

`appGroupEnabled` フラグが YES に設定されている場合にグループ辞書を返す 1 つのクラスメソッド `msdkUserDefaults` が定義されます。SDK カテゴリを次に示します。

```
@implementationNSUserDefaults (SFAdditions)

+ (NSUserDefaults *)msdkUserDefaults {
    NSUserDefaults *sharedDefaults = nil;
    if ([SFSDKDatasharingHelper sharedInstance].appGroupEnabled) {
        sharedDefaults = [[NSUserDefaults alloc]
            initWithSuiteName:[SFSDKDatasharingHelper sharedInstance].appGroupName];
    } else {
        sharedDefaults = [NSUserDefaults standardUserDefaults];
    }
    return sharedDefaults;
}

@end
```

AppDelegate のコード変更

次の手順は、メインアプリケーションの AppDelegate クラスの `init` メソッドに適用されます。

1. `init` メソッドの上部で、`SFSDKDatasharingHelper` 共有インスタンスの `appGroupName` と `appGroupEnabled` を設定します。

```
...
- (id)init
{
    self = [super init];
    if (self) {
        // Insert these two lines, using your app group name
        [SFSDKDatasharingHelper sharedInstance].appGroupName =
            @"<your app group name>";
        [SFSDKDatasharingHelper sharedInstance].appGroupEnabled = YES;
    }
    ...
}
```

2. ユーザーが正常にログインした後で、拡張コードでアクセスできるオブジェクトにログイン状況を反映した値を保存します。状況は、`SFUserAccountManager` オブジェクトの `currentUser` プロパティを参照すれば推測できます。ログイン状況を安全に一貫して更新するには、`init` メソッドの `registerBlockForCurrentUserChangeNotifications` コールのコールバックメソッドでこのプロパティを設定します。次の例では、Mobile SDK の `NSUserDefaults (SFAdditions)` カテゴリを使用して Boolean 値を `userLoggedIn` という名前の鍵の下に保存します。

```
[SFSDKAuthHelper registerBlockForCurrentUserChangeNotifications:^(
    __strong typeof (weakSelf) strongSelf = weakSelf;
    BOOL loggedIn = [SFUserAccountManager.sharedInstance currentUser] != nil;
    [[NSUserDefaults msdkUserDefaults] setBool:loggedIn forKey:@"userLoggedIn"];
    [strongSelf resetViewState:^(
        [strongSelf setupRootViewController];
    )];
)];
```

アプリケーション拡張のコード変更

実行時に、iOS アプリケーション拡張は 2 番目のアプリケーションとして動作するため、これも「ブートストラップ」する必要があります。メインアプリケーションの `AppDelegate` クラスで行ったのと同様の `appGroupName` と `appGroupEnabled` の変更を適用します。

アプリケーション拡張では、ユーザーログインのような認証タスクを実行できません。ただし、`MobileSyncSDKManager` などの Mobile SDK マネージャーをコールする前に、ユーザーがログインしたことを確認する必要があります。この確認を行うには、メインアプリケーションによって `NSUserDefaults` 辞書に保存された値を確認します。

1. アプリケーション拡張の初期化エントリポイントで、アプリケーショングループを設定して有効にします。

```
[SFSDKDatasharingHelper sharedInstance].appGroupName = @"<your app group name>";  
[SFSDKDatasharingHelper sharedInstance].appGroupEnabled = YES;
```

2. `userLoggedIn` 値が成功を示すことを確認します。`userLoggedIn` が YES である場合に限り、その他の Mobile SDK コールを続けます。

アプリケーションによってユーザーが正常にログインしたと判断された場合、アプリケーション拡張ではそのユーザーの共有ログイン情報を Salesforce API コールに適用できます。

❗ 重要:

- 開発者は、ユーザーのログイン状況を判断する必要があります。iOS アプリケーション拡張のコードでは、ユーザーが正常にログインする前に、`SalesforceSDKManager` オブジェクトを呼び出そうとしないでください。

 **例:** 次のコードは、[MobileSyncExplorer ネイティブサンプルアプリケーション](#)からの抜粋です。このサンプルではアプリケーション拡張を定義し、MRU 取引先責任者レコードのリストを表示しています。

アプリケーションおよび拡張ターゲット以外に、`MobileSyncExplorer` プロジェクトでは 3 番目のターゲット `MobileSyncExplorerCommon` が定義されます。このコンテナモジュールはメインターゲットと拡張ターゲットでアクセスできますが、アプリケーショングループには含まれていません。3つのプロパティを宣言して管理する `MobileSyncExplorerConfig` クラスが実装されます。

`appGroupName`

プロジェクト設定で定義される「アプリケーショングループ」名。この文字列は `MobileSyncExplorerConfig` クラスでハードコード化されています。

`appGroupsEnabled`

アプリケーショングループが有効化されているかどうかを示す Boolean。

`userLoginStatusKey`

グループ `NSUserDefaults` 辞書にユーザーログイン状況を保存するために使用された鍵の名前。

実行時に、メインアプリケーションと拡張の両方で値 `appGroupName` および `appGroupsEnabled` プロパティを使用して `SFSDKDataSharingHelper` クラスの独自のコピーで一致するプロパティを設定します。適切に設定されていれば、このクラスにより、両方のアプリケーションで現在のユーザーのログイン情報を使用して Salesforce API をコールできます。

ユーザーのログイン状況を保持するために、MobileSyncExplorer で `resetUserloginStatus` という名前の非公開メソッドを定義します。このメソッドでは、指定された値をユーザーのデフォルト辞書の `userLoggedIn` 鍵の下に保存します。

```
- (void)resetUserloginStatus {
    BOOL loggedIn = [SFUserAccountManager.sharedInstance currentUser] != nil;
    [[NSUserDefaults msdkUserDefaults] setBool:loggedIn forKey:@"userLoggedIn"];
    [SFSDKMobileSyncLogger log:[self class] level:SFLogLevelDebug format:@"%d
userLoggedIn", [[NSUserDefaults msdkUserDefaults] boolForKey:@"userLoggedIn"] ];
}
```

`init` メソッドでは `resetUserloginStatus` へのコールを使用して `registerBlockForCurrentUserChangeNotifications` ブロックを簡略化しています。

```
- (id)init
{
    self = [super init];
    if (self) {
        MobileSyncExplorerConfig *config = [MobileSyncExplorerConfig sharedInstance];

        [SFSDKDatasharingHelper sharedInstance].appGroupName = config.appGroupName;
        [SFSDKDatasharingHelper sharedInstance].appGroupEnabled =
config.appGroupsEnabled;

        [MobileSyncSDKManager initializeSDK];

        //App Setup for any changes to the current authenticated user
        __weak typeof (self) weakSelf = self;
        [SFSDKAuthHelper registerBlockForCurrentUserChangeNotifications:^(
            __strong typeof (weakSelf) strongSelf = weakSelf;
            [strongSelf resetUserloginStatus];
            [strongSelf resetViewState:^(
                [strongSelf setupRootViewController];
            )]];
    }
    return self;
}
```

このコードでは `MobileSyncExplorerConfig` のインスタンスを使用して `appGroupName` プロパティおよび `appGroupsEnabled` プロパティを保存しています。

アプリケーション拡張は、`RecentContactsTodayExtension/TodayViewController.m` ファイルで実装します。Today 拡張のカスタムコードへのエントリポイントは、`widgetPerformUpdateWithCompletionHandler:` です。アプリケーション拡張では、カスタムの `userIsLoggedIn` 「getter」メソッドで照会した `userLoginStatusKey` の値を読み取ります。この getter メソッドでは、メインアプリケーションによって保存された場所、つまり共有 `MobileSyncExplorerConfig` オブジェクトでアプリケーショングループ情報を取得します。

```
- (void)widgetPerformUpdateWithCompletionHandler:(void
(^) (NCUpdateResult))completionHandler {
    MobileSyncExplorerConfig *config = [MobileSyncExplorerConfig sharedInstance];
    [SFSDKDatasharingHelper sharedInstance].appGroupName = config.appGroupName;
```

```

[SFSDKDatasharingHelper sharedInstance].appGroupEnabled = config.appGroupsEnabled;

if ([self userIsLoggedIn] ) {
    [SFLogger log:[self class] level:SFLogLevelError format:@"User has logged in"];

    [MobileSyncSDKManager initializeSDK];
    SFUserAccount *currentUser = [SFUserAccountManager sharedInstance].currentUser;

    __weak typeof(self) weakSelf = self;
    void (^completionBlock)(void) = ^{
        [weakSelf refreshList];
    };
    if(currentUser) {
        if (!self.dataMgr) {
            self.dataMgr = [[SObjectDataManager alloc]
initWithDataSpec:[ContactSObjectData dataSpec]];
        }
        [self.dataMgr lastModifiedRecords:kNumberOfRecords
completion:completionBlock];
    }
    completionHandler(NCUpdateResultNewData);
}

- (BOOL)userIsLoggedIn {
    MobileSyncExplorerConfig *config = [MobileSyncExplorerConfig sharedInstance];
    return [[NSUserDefaults msdkUserDefaults] boolForKey:config.userLoginStatusKey];
}

```

Signposts を使用したプロファイリング

Signposts には、アプリケーションの実行時のリソース使用状況を記録する強力なオプションが用意されています。Signposts ログにより、Xcode の Instruments で表示するときに、アプリケーションの経時的なパフォーマンスをモデル化できます。これらのプロファイルは、コードのボトルネックやその他の異常を見つけるのに役立ちます。バージョン 7.1 以降、Mobile SDK では、コードの頻繁に使用される部分に Signposts が追加され、アプリケーションのそのパフォーマンスをプロファイリングできるようになっています。Mobile SDK の Signposts は、Swift および Objective-C アプリケーションで使用できます。

デバッグプロファイルの場合、Signposts を有効にするには、Xcode を 1 回切り替える必要があります。CocoaPods を使用するアプリケーションの本番プロファイルの場合、Podfile にいくつかの変更を加えて再ビルドします。

デバッグビルドでの Signposts の使用

Mobile SDK ライブラリでは、Signposts はデバッグビルド用に事前設定されています。ただし、Xcode プロファイルスキームのデフォルトはリリースビルドになります。これを変更しましょう。

デバッグビルドをプロファイリングする手順は、次のとおりです。

1. Xcode タスクバーで、スキーム名をクリックし、[Edit Scheme (スキームを編集)] を選択します。

2. [Profile (プロファイル)] を選択します。
3. [Build Configuration (ビルド設定)] を [Debug (デバッグ)] に設定します。

本番ビルドでの Signposts の使用

Mobile SDK の Signposts を使用して本番ビルドをプロファイリングする場合、デフォルトのプロファイルスキームを編集することはありません。ただし、すでにデフォルト設定を操作している場合、それらを次の手順で確認します。

1. Xcode タスクバーで、スキーム名をクリックし、[Edit Scheme (スキームを編集)] を選択します。
2. [Profile (プロファイル)] を選択します。
3. [Build Configuration (ビルド設定)] が [Release (リリース)] に設定されていることを確認します。

forceios や CocoaPods を使用せずに「手動」でプロジェクトを作成した場合、設定を実行できる状態になっています。「[Signposts プロファイルビルドの実行](#)」に進みます。

forceios を使用してアプリケーションを作成した場合、CocoaPods によって Mobile SDK pod の設定が制御されます。CocoaPods を使用する Mobile SDK アプリケーションの本番ビルドをプロファイリングする手順は、次のとおりです。

1. アプリケーションの Podfile の次のセクションを編集します。

```
# Comment the following if you do not want the SDK to emit signpost
# events for instrumentation. Signposts are enabled for non release version of the app.

post_install do |installer_representation|
  installer_representation.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      if config.name == 'Debug'
        config.build_settings['GCC_PREPROCESSOR_DEFINITIONS'] ||=
['$(inherited)', 'DEBUG=1', 'SIGNPOST_ENABLED=1']
        config.build_settings['OTHER_SWIFT_FLAGS'] = ['$(inherited)',
'-DDEBUG', '-DSIGNPOST_ENABLED']
      end
    end
  end
end
```

- if config.name == 'Debug' (上記) を if config.name == 'Release' に変更します。
- '\$(inherited)', 'DEBUG=1', '-DDEBUG' 設定を削除します。

意図する結果は次のようになります。

```
# Comment the following if you do not want the SDK to emit signpost
# events for instrumentation. Signposts are enabled for non release version of the app.

post_install do |installer_representation|
  installer_representation.pods_project.targets.each do |target|
    target.build_configurations.each do |config|
      if config.name == 'Release'
        config.build_settings['GCC_PREPROCESSOR_DEFINITIONS'] ||=
['SIGNPOST_ENABLED=1']
      end
    end
  end
end
```



```

        config.build_settings['OTHER_SWIFT_FLAGS'] = ['-DSIGNPOST_ENABLED']
    end
end
end
end

```

2. 重要: ワークスペースを閉じます。
3. ターミナルアプリケーションで、ワークスペースディレクトリに切り替えて `pod update` を実行します。
4. Xcode で再度ワークスペースを開きます。

Signposts プロファイルビルドの実行

Signposts ビルドを実行する手順は、次のとおりです。

1. Xcode メニューで、[Product (プロダクト)] > [Profile (プロファイル)] をクリックします。
2. 応答画面が表示されたら、プロファイリングテンプレート (「Blank (ブランク)」など) を選択し、[Choose (選択)] をクリックします。
3. [Instruments] パネルで、+ をクリックし、[os_signposts] を見つけます。
4. [os_signposts] をダブルクリックします。
5. [Record (記録)] をクリックします。処理が開始されます。

iOS サンプルアプリケーション

forceios で作成したネイティブアプリケーションは、範囲は限られますが、それ自体がサンプルアプリケーションです。ネイティブの iOS サンプルアプリケーションには、独自のアプリケーションで確認および操作できる多くの機能が含まれています。

- **MobileSyncExplorer** は、iOS 上でのネイティブ Mobile Sync ライブラリの機能を示します。これは、`native/SampleApps/MobileSyncExplorer` の Mobile SDK for iOS にあります。
- **RestAPIExplorer** では、ネイティブのすべての REST API ラッパーが実行されます。これは、`native/SampleApps/RestAPIExplorer` の Mobile SDK for iOS にあります。

Mobile SDK には、次のハイブリッドアプリケーション用の iOS ラッパーが用意されています。

- **AccountEditor**: `mobilesync.js` ライブラリを使用してオフラインデータを同期する方法について説明します。
- **MobileSyncExplorer**: Mobile Sync プラグインを使用してオフラインデータを同期する方法について説明します。
- **NoteSync**: 非 REST API を使用して Salesforce のメモを取得する方法を示します。

第7章

ネイティブ Android の開発

トピック:

- [Android ネイティブ クイックスタート](#)
- [ネイティブ Android の要件](#)
- [forcedroid を使用した Android プロジェクトの作成](#)
- [Android Studio での サンプルプロジェクトの設定](#)
- [ネイティブ Android アプリケーションの開発](#)
- [Android サンプルアプリケーション](#)

Salesforce Mobile SDK for Android には、ソースコード、開発スクリプト、ネイティブサンプルアプリケーションが揃っているため、すぐに開発に着手できます。また、Mobile SDK アプリケーションの2つの基本的な機能を実装するテンプレートアプリケーションが用意されています。

- OAuth2 認証フローの自動化。これにより、プロセスをアプリケーションに統合しやすくなります。
- Salesforce REST API へのアクセス。アクセスを簡略化するユーティリティクラスが使用されます。

サンプルネイティブアプリケーションは、ログイン、REST API コール、その他の Mobile SDK 機能を実装する基本的な方法を示します。

Android ネイティブクイックスタート

すばやく開始する手順は、次のとおりです。

1. [ネイティブ Android 要件](#)をすべて満たしていることを確認します。
2. [Mobile SDK for Android](#)をインストールします。必要に応じて、代わりに [Mobile SDK GitHub リポジトリ](#)から Mobile SDK をインストールすることもできます。
3. コマンドラインで、新しい [Android プロジェクト](#)を作成する forcedroid アプリケーションを実行し、Android Studio またはコマンドラインからそのアプリケーションを実行します。
4. 「[Android Studio でのサンプルプロジェクトの設定](#)」の手順に従います。

ネイティブ Android の要件

Salesforce の要件

- Salesforce Mobile SDK for Android 11.1 以降。
- [接続アプリケーション](#)が存在する Salesforce [Developer Edition 組織](#)。

SalesforceSDK プロジェクトは、Android Nougat (API 24) ライブラリを使用して作成されます。

Android の要件

Android の開発環境をインストールするには、[Mobile SDK Trailhead プロジェクト](#)に移動し、次の手順を実行します。

1. [共通コンポーネント](#)をインストールする
2. [Android 開発環境を設定する](#)

インストールする項目は次のとおりです。

- Java JDK 11.0.11+9 以降 — www.oracle.com/downloads。
- Android Studio の最新バージョン — developer.android.com/sdk。
- Android SDK — 次の範囲の API をすべて含めます。
 - 最小限の API: Android Nougat (API 24)
 - 対象の API: Android 13 (API 33)
- Android SDK ツール
- Android 仮想デバイス (AVD)



ヒント:

- 最良の結果を得るには、Android SDK マネージャーで推奨されるすべての Android SDK バージョンと Android SDK ツールの使用可能なすべてのバージョンをインストールします。
- Windows では、必ずシステム管理者として Android Studio を実行します。

関連トピック

developer.android.com の「Android Studio」

forcedroid を使用した Android プロジェクトの作成

アプリケーションを作成するには、ターミナルウィンドウまたは Windows コマンドプロンプトで forcedroid を使
用します。forcedroid ユーティリティでは、2つの方法でアプリケーションを作成できます。

- 目的のアプリケーションの種別と基本設定データを指定する。
または
- 既存の Mobile SDK アプリケーションをテンプレートとして使用する。この場合も基本設定データを指定しま
す。

forcedroid は、コマンドラインプロンプトの対話型モードまたはコマンドライン引数を指定したスクリプトモ
ードで使用できます。コマンドの使用方法に関する情報を表示するには、引数を指定せずに「forcedroid」と
入力します。

forcedroid のプロジェクトタイプ

`forcedroid create` コマンドには、次のいずれかのプロジェクトタイプを指定する必要があります。

アプリケーション種別	アーキテクチャ	言語
<code>native</code>	ネイティブ	Java
<code>native_kotlin</code> (または単に <code>RETURN</code> キーを押す)	ネイティブ	Kotlin

Java でネイティブ Android アプリケーションを開発するには、`native` を指定します。

対話形式での forcedroid create の使用

対話形式のコマンドプロンプトでアプリケーションオプションを入力するには、「forcedroid create」と
入力します。forcedroid ユーティリティにより、各設定オプションの入力を促すメッセージが表示されます。次
に例を示します。

```
$ forcedroid create
Enter your application type (native, native_kotlin): <press RETURN>
Enter your application name: testNative
Enter the package name for your app (com.mycompany.myapp): com.bestapps.android
Enter your organization name (Acme, Inc.): BestApps
Enter output directory for your app (leave empty for the current directory): testNative
```

このコマンドでは、現在のディレクトリの `testNative\` サブディレクトリに「testNative」という名前のネイ
ティブ Kotlin Android アプリケーションを作成します。

スクリプトモードでの `forcedroid create` の使用

スクリプトモードでは、対話形式の入力メッセージを使用せずに `forcedroid` を使用できます。たとえば、Java で記述されたネイティブアプリケーションを作成する場合、次のようになります。

```
$ forcedroid create --apptype="native" --appname="package-test"
--packagename="com.acme.mobile_apps"
--organization="Acme Widgets, Inc." --outputdir="PackageTest"
```

また、Kotlin で記述されたネイティブアプリケーションを作成する場合、次のようになります。

```
$ forcedroid create --apptype="native_kotlin" --appname="package-test"
--packagename="com.acme.mobile_apps"
--organization="Acme Widgets, Inc." --outputdir="PackageTest"
```

各コールは「package-test」というネイティブアプリケーションを作成し、現在のディレクトリの `PackageTest/` サブディレクトリ内に配置します。

テンプレートからのアプリケーションの作成

`forcedroid createWithTemplate` コマンドは、アプリケーション種別ではなく GitHub リポジトリの URI が求められる点を除いて `forcedroid create` と同じです。この URL を、テンプレートとして使用できる Mobile SDK アプリケーションを含む任意のリポジトリディレクトリを参照するように設定します。テンプレートアプリケーションには、サポートされている任意の Mobile SDK アプリケーション種別を使用できます。スクリプトは、他のパラメーターに指定された値に一致するようにテンプレートの識別子と設定を変更します。

`createWithTemplate` を使用する前に、使用可能なテンプレートを把握しておくに役立ちます。把握するには、「`forcedroid listtemplates`」と入力します。このコマンドにより、Mobile SDK によって提供されるテンプレートのリストが出力されます。各リストには、テンプレートおよびその GitHub URI の簡単な説明が含まれます。次に例を示します。

```
Available templates:

1) Basic Kotlin application
forcedroid createwithtemplate --templaterepoururi=AndroidNativeKotlinTemplate
2) Basic Java application
forcedroid createwithtemplate --templaterepoururi=AndroidNativeTemplate
3) Sample Kotlin Identity Provider application
forcedroid createwithtemplate --templaterepoururi=AndroidIDPTemplate
```

テンプレートの URI が見つかったら、`forcedroid` コマンドラインに挿入できます。`forcedroid createWithTemplate` のコマンドラインでの使用方法に関する情報は、次のようになります。

```
Usage:
forcedroid createWithTemplate
--templaterepoururi=<Template repo URI> (e.g.
https://github.com/forcedotcom/SalesforceMobileSDK-Templates/MobileSyncExplorerReactNative)]

--appname=<Application Name>
--packagename=<App Package Identifier> (e.g. com.mycompany.myapp)
--organization=<Organization Name> (Your company's/organization's name)
--outputdir=<Output directory> (Leave empty for current directory)]
```

SalesforceMobileSDK-Templates リポジトリ内のテンプレートでは、`templaterepouri` のパスを削除できます。テンプレート名だけで問題ありません。例を次に示します。

```
forcedroid createwithtemplate --templaterepouri=AndroidNativeKotlinTemplate
```

`forcedroid createWithTemplate` は、対話形式またはスクリプトモードで使用できます。たとえば、スクリプトモードのコールは次のようになります。

```
forcedroid createWithTemplate
--templaterepouri=SalesforceMobileSDK-Templates/AndroidIDPTemplate#v6.2.0
--appname=MyIDP-Android
--packagename=com.mycompany.react
--organization="Acme Software, Inc."
--outputdir=testWithTemplate
```

このコールでは、Android IdP サンプルアプリケーションと同じソースコードおよびリソースを使用して Android ID プロバイダーアプリケーションが作成されます。`forcedroid` によって、現在のディレクトリの `testWithTemplate/` サブディレクトリ内に新しいアプリケーションが配置されます。また、プロジェクト全体でアプリケーション名前「MyIDP-Android」に変更されます。

forcedroid バージョンの確認

インストールされている `forcedroid` のバージョンを確認するには、次のコマンドを実行します。

```
forcedroid version
```

Android Studio でのアプリケーションのインポートおよび作成

1. Android Studio でプロジェクトを開きます。

- ようこそ画面から [Import Project (Eclipse ADT, Gradle, etc.) (プロジェクトをインポート (Eclipse ADT、Gradle など))] をクリックします。

または

- [File (ファイル)] メニューから、[File (ファイル)] > [New (新規)] > [Import Project... (プロジェクトをインポート...)] をクリックします。

2. プロジェクトのディレクトリを参照し、[OK] をクリックします。

Android Studio によってワークスペースが自動的に作成されます。このプロセスに数分かかることがあります。ステータスバーに「Gradle build successful (Gradle のビルドに成功しました)」と表示されていれば、プロジェクトを実行できます。

3. [Run <project_name> (<project_name> を実行)] をクリックするか、SHIFT+F10 キーを押します。ネイティブプロジェクトの場合、プロジェクト名は指定したアプリケーション名になります。

Android Studio によってエミュレーターまたは接続されている Android デバイスにアプリケーションが起動します。

コマンドラインからのアプリケーションの作成および実行

コマンドラインからコマンドプロンプトに戻ると、プロジェクトを設定およびクリーンアップするための Android ユーティリティの実行手順が forcedroid スクリプトにより出力されます。コマンドラインからアプリケーションを作成および実行する場合は、これらの手順を実行します。

1. 新しいアプリケーションを作成します。

- **Windows:**

```
cd <your_project_directory>
gradlew assembleDebug
```

- **Mac:**

```
cd <your_project_directory>
./gradlew assembleDebug
```

正常に作成されると、プロジェクトの `build/outputs/apk` ディレクトリに署名付き APK デバッグファイルを検索できます。

2. 実行されていないエミュレーターを使用する場合は、Android AVD マネージャーを使用して起動します。物理デバイスを使用している場合は、接続します。
3. APK ファイルをエミュレーターまたはデバイスにインストールします。

- **Windows:**

```
adb install <path_to_your_app>\build\outputs\apk\<app_name>.apk
```

- **Mac:**

```
./adb install <path_to_your_app>/build/outputs/apk/<app_name>.apk
```

新しくインストールされたアプリケーションがない場合は、エミュレーターまたはデバイスを再起動します。詳細は、developer.android.com の「コマンドラインからアプリケーションをビルドする」を参照してください。

forcedroid スクリプトで新しいアプリケーションを生成する方法

- スクリプトは、実行時に GitHub リポジトリからテンプレートをダウンロードします。
- `forcedroid create` コマンドの場合は、[SalesforceMobileSDK-Templates](https://github.com/SalesforceMobileSDK-Templates) GitHub リポジトリのデフォルトの Kotlin テンプレートを使用します。
- 生成されたアプリケーションは Gradle を使用します。
- スクリプトは実行時に npm を使用して Mobile SDK ライブラリをダウンロードします。 `settings.gradle` ファイルの `node_modules` の下でこれらのライブラリが参照されます。

関連トピック:

[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)

カスタムテンプレートを使用したアプリケーションの作成

Mobile SDK テンプレートの概要

Mobile SDK は、iOS および Android でサポートするアーキテクチャ別のテンプレートを定義します。これらのテンプレートは github.com/forcedotcom/SalesforceMobileSDK-Templates リポジトリに保持されます。顧客が forcedroid または forceios の `create` コマンドを実行すると、スクリプトが、リポジトリから適切な組み込みテンプレートをコピーし、このコピーを新しいアプリケーションに変換します。この方法で作成されたアプリケーションは、機能のほとんどない基本的な Mobile SDK アプリケーションです。

実際には、追加の機能、リソース、ブランド設定などを備えた独自のテンプレートを作成したいのではないのでしょうか。この同じ Mobile SDK メカニズムを活用して、独自のアプリケーションをテンプレートに変換することができます。そして、forcedroid または forceios に、その独自のテンプレートではなく、この変換後のテンプレートを使用するように指示できます。

カスタムテンプレートの使用方法

Mobile SDK は、forcedroid および forceios の `create` のほか、`createWithTemplate` コマンドも定義します。forcedroid または forceios の `createWithTemplate` を実行するときは、アプリケーション種別ではなく、テンプレートアプリケーションリポジトリを指定し、続いて残りのアプリケーション作成パラメーターを指定します。テンプレートアプリケーションリポジトリには、スクリプトがテンプレートと認識する Mobile SDK アプリケーションが保持されます。このテンプレートから新しい Mobile SDK アプリケーションを作成するために、スクリプトはテンプレートアプリケーションを新しいフォルダーにコピーして、コピーされたコードにパラメーター値を適用します。

template.js ファイル

不明なアプリケーションをテンプレートとして受け入れる場合、forceios および forcedroid で `template.js` 設定ファイルを定義する必要があります。このファイルを、テンプレートアプリケーションリポジトリのルートに保存します。このファイルはスクリプトに、標準アプリケーションのリファクタリングタスク (ファイルの移動、テキストの置換、リソースの削除および名前変更) の実行方法を指示します。他方、より広範な変更を適用したいことがあります。そのような場合には、標準の範囲を超えるカスタマイズが実行されるように `template.js` を適応させることができます。たとえば、メインのエントリーポイントクラス以外のクラスにアプリケーション名を挿入する場合は、`template.js` を使用してこれらの変更を実行できます。

`template.js` ファイルは 2 つの部分で構成されます。テンプレートから新しいアプリケーションを準備する JavaScript の「準備」関数と、エクスポートの宣言です。

template.js の準備関数

`template.js` ファイルの大半は「準備」関数で構成されます。デフォルトで、準備関数は次の署名を使用します。

```
function prepare(config, replaceInFiles, moveFile, removeFile)
```

この関数の名前は変更できますが、更新された名前をエクスポートのリストに忘れずに指定する必要があります。Mobile SDK スクリプトは、次の引数を使用してエクスポートする関数をコールします。

- `config`: プラットフォーム (iOS または Android)、アプリケーション名、パッケージ名、組織、Mobile SDK バージョンを識別する辞書。
- `replaceInFiles`: ファイル内の文字列を置換するヘルパー関数。
- `moveFile`: ファイルおよびディレクトリを移動するヘルパー関数。
- `removeFile`: ファイルおよびディレクトリを削除するヘルパー関数。

Mobile SDK テンプレート内にあるデフォルトの準備関数は、標準テンプレートアプリケーションをパーソナライズするために必要な文字列の置換やファイルの移動および削除を行います。機能を追加する場合は、コードをこの準備関数内に配置します。ただし、準備関数に渡されるヘルパー関数は、標準テンプレートアプリケーションのタスクのみを実行できます。カスタムタスクについては、独自のメソッドを実装してコールする必要があります。

template.js で定義されたエクスポート

各 `template.js` ファイルは、次の2つのエクスポートを定義します。

appType

次のいずれかの値を割り当てます。

- `'native'`
- `'native_kotlin'` (forcedroid のみ)
- `'native_swift'` (forceios のみ)
- `'react_native'`
- `'hybrid_local'`
- `'hybrid_remote'`

prepare

準備関数のハンドル (引用符なしでリスト)。

`template.js` ファイルのエクスポートセクションの例を次に示します。このテンプレートは、`prepare` という名前の準備関数を定義するネイティブアプリケーション用です。

```
//  
// Exports  
//  
module.exports = {  
  appType: 'native',  
  prepare: prepare  
};
```

この場合、準備関数のハンドルは「`prepare`」です。

```
function prepare(config, replaceInFiles, moveFile, removeFile)
```

template.js のテンプレートアプリケーションの識別 (ネイティブおよび React Native アプリケーション)

ネイティブおよび React Native アプリケーションの場合は、テンプレートアプリケーションの準備関数が、アプリケーション名、パッケージ名、および組織名または会社名を定義します。これらの値は、テンプレートから

作成された新しいカスタムアプリケーションではなく、テンプレートアプリケーション自体を識別します。実行時に、Mobile SDK スクリプトがこれらの値を使用して、スクリプトの入力値に置換される文字列を見つけます。これらの `iOSNativeTemplate` テンプレートアプリケーションの設定の例を次に示します。

```
// Values in template
var templateAppName = 'iOSNativeTemplate';
var templatePackageName = 'com.salesforce.iosnativetemplate';
var templateOrganization = 'iOSNativeTemplateOrganizationName';
```

template.js ファイルの例

Mobile SDK は、github.com/forcedotcom/SalesforceMobileSDK-Templates リポジトリで独自のテンプレートを定義します。各テンプレートディレクトリには `template.js` ファイルが含まれます。テンプレートに含まれる具体的な内容は次のとおりです。

- `iOSNativeTemplate` (forceios のみ)
- `iOSNativeSwiftTemplate` (forceios のみ)
- `ReactNativeTemplate`
- `HybridLocalTemplate`
- `HybridRemoteTemplate`
- `AndroidNativeTemplate` (forcedroid のみ)
- `AndroidNativeKotlinTemplate` (forcedroid のみ)

これらのテンプレートは、`MobileSDKnpm` スクリプトがアプリケーションを作成するために使用する「ベアボーン」プロジェクトであるため、意図的に複雑性を低くしています。より高度なテンプレートをお探しの場合、次の項目を参照してください。

- `MobileSyncExplorerReactNative`
- `MobileSyncExplorerSwift`
- `AndroidIDPTemplate`
- `iOSIDPTemplate`

リポジトリパスを含めたこれらのテンプレートのリストは、`listtemplates` コマンドで取得できます。すべての Mobile SDK npm スクリプト — `forcedroid`、`forceios`、`forcehybrid`、`forcereact` — で、このコマンドがサポートされています。

 **メモ:** スクリプトのコマンドは常にテンプレートと一致させます。`forceios createWithTemplate` のみが設定された iOS 固有のテンプレートと、`forcedroid createWithTemplate` のみが設定された Android 固有のテンプレートを使用します。この制限はハイブリッドおよび React Native テンプレートには適用されません。

基本の `template.js` ファイルの定義

基本の `template.js` ファイルを最も迅速に作成する手順を次に示します。

1. github.com/forcedotcom/SalesforceMobileSDK-Templates リポジトリから `template.js` ファイルを、カスタムテンプレートアプリケーションリポジトリのルートにコピーします。必ずテンプレートで作成するアプリケーションの種別に一致するテンプレートを選択してください。

2. ネイティブまたは React Native アプリケーションのみについては、テンプレートアプリケーションを反映させるようにアプリケーション名、パッケージ名、および組織を更新します。
3. 必要に応じて、`module.exports` オブジェクトの `appType` および `prepare` 設定を前述のとおり更新します。この基本的な例ではこの手順は不要ですが、後々独自の `template.js` ファイルを作成する場合に必要になることがあります。

制限とガイドライン

カスタムテンプレートにはいくつかの制限が適用されます。

- テンプレートアプリケーションは、サポートされているすべてのプラットフォームおよびアーキテクチャを対象とする有効な Mobile SDK アプリケーションにすることができます。
- 主要な要件は、テンプレートリポジトリおよびローカルの Mobile SDK リポジトリを同じ Mobile SDK バージョンにする必要があることです。Git のバージョンタグを使用すると、両方のリポジトリを特定の旧バージョンに同期できますが、この処理はお勧めしません。
- スクリプトのコマンドは常にテンプレートと一致させます。`forceios createWithTemplate` のみが設定された iOS 固有のテンプレートと、`forcedroid createWithTemplate` のみが設定された Android 固有のテンプレートを使用します。この制限はハイブリッドおよび React Native テンプレートには適用されません。

Android Studio でのサンプルプロジェクトの設定

SalesforceMobileSDK-Android GitHub リポジトリには、作成して実行できるサンプルアプリケーションが含まれています。

1. SalesforceMobileSDK-Android GitHub リポジトリをまだコピーしていない場合は、コピーします。

- **Mac:**

```
git clone https://github.com/forcedotcom/SalesforceMobileSDK-Android.git
./install.sh
```

- **Windows:**

```
git clone https://github.com/forcedotcom/SalesforceMobileSDK-Android.git
cscript install.vbs
```

2. Android Studio でプロジェクトを開きます。

- ようこそ画面から [Import Project (Eclipse ADT, Gradle, etc.) (プロジェクトをインポート (Eclipse ADT、Gradle など))] をクリックします。

または

- [File (ファイル)] メニューから、[File (ファイル)] > [New (新規)] > [Import Project... (プロジェクトをインポート...)] をクリックします。

3. `<path_to_SalesforceMobileSDK-Android>/native/NativeSampleApps/` または `<path_to_SalesforceMobileSDK-Android>/hybrid/HybridSampleApps/` を参照します。

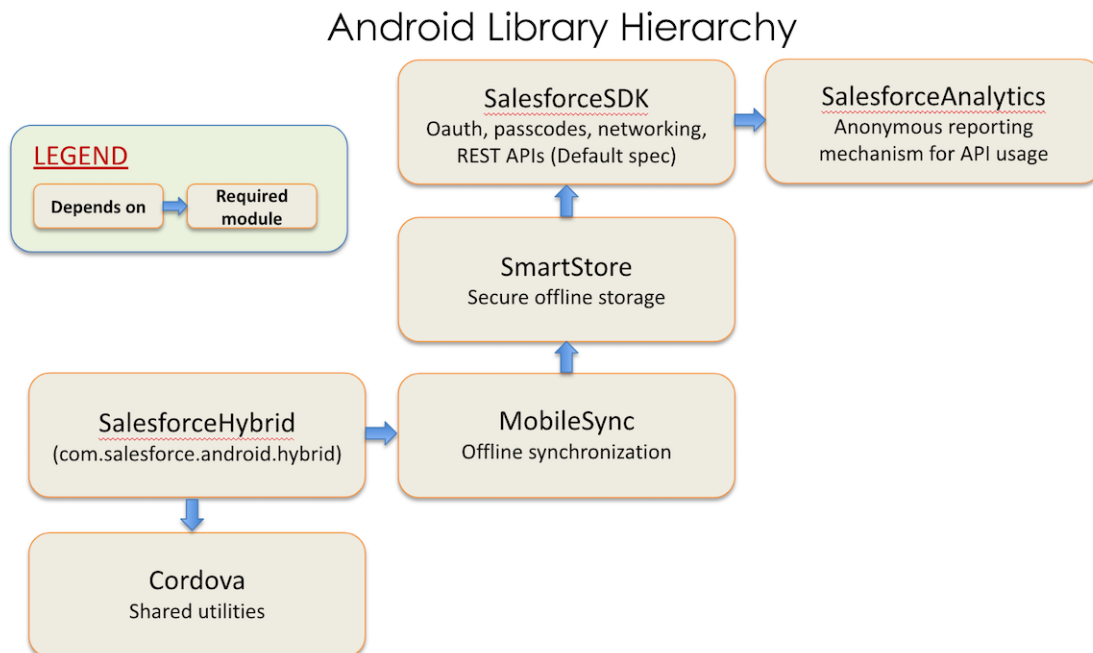
4. リストされているサンプルアプリケーションのいずれかを選択し、OK をクリックします。
5. プロジェクトの作成が終了したら、[Select Run/Debug Configurations (実行/デバッグ設定の選択)] ドロップダウンメニューでサンプルプロジェクトを選択します。
6. *SHIFT-F10* キーを押します。

Android プロジェクトファイル

Android Studio の [Project (プロジェクト)] ウィンドウでネイティブアプリケーションを参照すると、次のライブラリプロジェクトが表示されます。

- `libs/SalesforceAnalytics` — Analytics プロジェクト。Mobile SDK アプリケーションの使用状況の非機密データを Salesforce にレポートします。
- `libs/SalesforceSDK` — Salesforce Mobile SDK プロジェクト。OAuth2 および REST API コールのサポートを提供します。
- `libs/SmartStore` — SmartStore プロジェクト。オフラインストレージソリューションを提供します。
- `libs/MobileSync` — Mobile Sync プロジェクト。オフラインデータ同期ツールを実装します。

Mobile SDK の各ライブラリは、次の図で示すように連動関係階層で相互に参照し合います。



ネイティブ Android アプリケーションの開発

Salesforce Mobile SDK のネイティブ Android バージョンでは、ホストデバイスの Android オペレーティングシステムを直接使用するリッチモバイルアプリケーションを作成できます。このようなアプリケーションを作成するには、Android 開発を十分に理解し、Mobile SDK ネイティブクラスを使用するコードを Java または Kotlin で記述する必要があります。

Android アプリケーションの構造

通常、Mobile SDK を使用するネイティブ Android アプリケーションには、次のクラスとアクティビティが必要になります。

- `android.app.Application` を拡張するアプリケーションエントリポイントクラス。
- `android.app.Activity` を拡張する 1 つ以上のアクティビティ。

Mobile SDK を使用することで、Android テンプレートアプリケーションでは次のことが可能になります。

- `android.app.Application` を拡張するスタブアプリケーションクラスの作成。このクラスでは `onCreate()` メソッドです。
 - `SalesforceSDKManager.initNative()` を呼び出す。
 - (省略可能) IDP ログインサービスの有効化。
 - (省略可能) プッシュ通知の有効化。
- `SalesforceActivity`、`SalesforceListActivity`、または `SalesforceExpandableListActivity` を拡張するアクティビティクラスの作成。これらの Salesforce クラスは必須ではありませんが、`android.app.Activity` を拡張する場合には使用することをお勧めします。

認証とアプリケーションのライフサイクルのクラス

最上位レベルの `SalesforceSDKManager` クラスは、ログインのフェーズを設定したり、ログアウト後にクリーンアップしたり、システムレベルアカウントが削除されたときにアプリケーションに情報を通知する特殊なイベントウォッチャーを提供したりします。OAuth プロトコルは、内部クラスで自動的に処理されます。

`SalesforceActivity`、`SalesforceListActivity`、`SalesforceExpandableListActivity` クラスにより、標準の Android UI プロトコルの Mobile SDK 実装が提供されます。

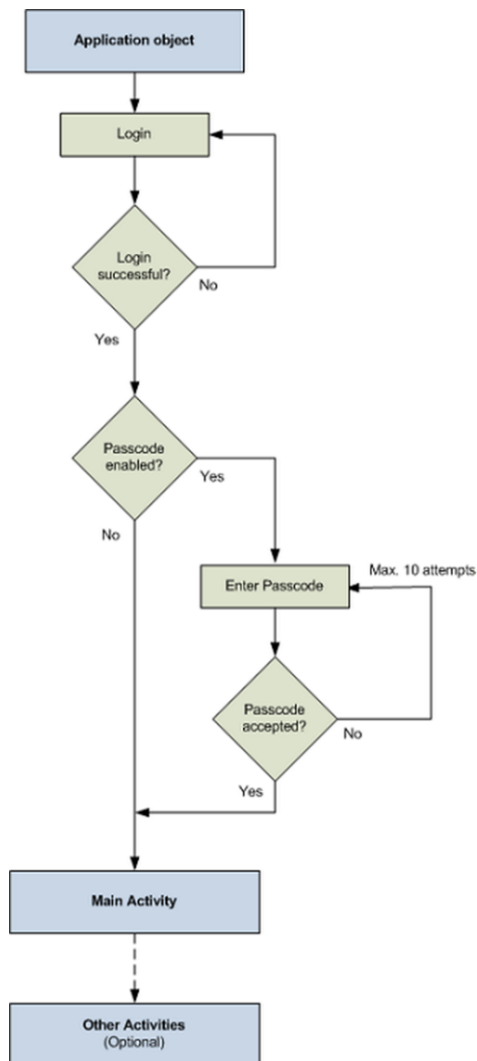
アプリケーションのメインアクティビティだけでなくすべてのアクティビティで、これらのいずれかのクラスを拡張することをお勧めします。アクティビティで別の基本クラスを使用する場合、`SalesforceActivity` にある `onResume()` プロトコルを自分で置き換える必要があります。

アクティビティ内でのリソースの使用

アクティビティ内で、Salesforce REST API をコールして Salesforce オブジェクトを操作します。Mobile SDK には、REST 要求および応答フローを簡略化する `com.salesforce.androidsdk.rest` パッケージが用意されています。

ユーザーインターフェースレイアウト、画像サイズ、文字列、および XML ファイルのその他のリソースを定義してカスタマイズします。SDK は、内部的に `R` クラスインスタンスを使用して、リソースを取得および操作し

ます。ただし、クライアントアプリケーションは Mobile SDK のリソースに直接アクセスできるため、これらの機能を管理するためにコードを記述する必要はありません。



ネイティブ API パッケージ

Salesforce Mobile SDK では、ネイティブ Android API が Java パッケージにグループ化されています。これらのパッケージの概要と要点については、「[Android パッケージおよびクラス](#)」を参照してください。

ネイティブクラスの概要

この Mobile SDK ネイティブクラスの概要では、各クラスの関連する詳細情報や必要なクラスがどこにあるのかについて説明します。

SalesforceSDKManager クラス

SalesforceSDKManager クラスは、Salesforce Mobile SDK を使用するすべてのネイティブ Android アプリケーションのエントリーポイントです。このクラスは次のメカニズムを提供します。

- ログインおよびログアウト
- パスコード
- ユーザーデータの暗号化および復号化
- 文字列変換
- ユーザーエージェントアクセス
- アプリケーションの終了
- アプリケーションのクリーンアップ

SalesforceSDKManager を直接コールする代わりに、forcedroid ネイティブテンプレートはサブクラス MobileSyncSDKManager を使用してアプリケーションを初期化します。

initNative() メソッド

起動時に、静的 `initNative()` メソッドをコールして、MobileSyncSDKManager クラスを初期化します。このメソッドが取る引数は、次のとおりです。

パラメーター名	説明
<code>applicationContext</code>	アプリケーションのコンテキストを説明する <code>Context</code> のインスタンス。Application 拡張クラスでは、コールを <code>getApplicationContext()</code> に渡してこのパラメーターを満たすことができます。
<code>mainActivity</code>	メインアクティビティを表示するクラスの記述子。メインアクティビティは、ログイン後に最初に表示されるアクティビティです。

forcedroid Java テンプレートアプリケーションの MainApplication クラスの例を次に示します。

```
import com.salesforce.androidsdk.mobilesync.app.MobileSyncSDKManager;
...

public class MainApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        MobileSyncSDKManager.initNative(getApplicationContext(),
            MainActivity.class);
        ...
    }
}
```

この例では、NativeKeyImpl は、KeyInterface のアプリケーションの実装です。MainActivity は、SalesforceActivity をサブクラス化し、ここではログイン後に最初にコールされるアクティビティとして指定されています。

Kotlin テンプレートアプリケーションでは、Kotlin 関係の定数を追加で定義し、Kotlin を使用されている機能として登録します。

```
class MainApplication : Application() {
    companion object {
        private const val FEATURE_APP_USES_KOTLIN = "KT"
    }

    override fun onCreate() {
        super.onCreate()
        MobileSyncSDKManager.initNative(applicationContext, MainActivity::class.java)
        MobileSyncSDKManager.getInstance().registerUsedAppFeature(FEATURE_APP_USES_KOTLIN)

        ...
    }
}
```

logout() メソッド

SalesforceSDKManager.logout() メソッドは、ユーザーデータをクリアします。たとえば、ユーザー固有の独自リソースを取り入れた場合、logout() を上書きすることで、そのリソースが次のユーザーセッションに引き継がれないようにできます。SmartStore は、Mobile SDK のオフラインデータベースであり、ログアウト時にユーザーデータおよびアカウント情報を自動的に破棄します。

クリーンアップ後(推奨)に、メソッド上書きのどこかでスーパークラス logout メソッドを常にコールします。Logout() を上書きする方法の例を次に示します。

Kotlin

```
override fun logout(frontActivity: Activity) {
    // Clean up all persistent and non-persistent app artifacts
    // ...
    // Call superclass after doing your own cleanup
    super.logout(frontActivity)
}
```

Java

```
@Override
public void logout(Activity frontActivity) {
    // Clean up all persistent and non-persistent app artifacts
    // ...
    // Call superclass after doing your own cleanup
    super.logout(frontActivity);
}
```

getLoginActivityClass() メソッド

このメソッドは、ログインアクティビティの記述子を返します。ログインアクティビティは、Salesforce サーバーがログインダイアログを提供するために使用する WebView を定義します。

getUserAgent() メソッド

Mobile SDK は、実行時にアプリケーションのバージョン設定情報を公開するためのユーザーエージェント文字列を作成します。たとえば、Mobile SDK 7.1 のユーザーエージェントでは、次の形式が使用されます。

```
SalesforceMobileSDK/<salesforceSDK version> android mobile/<android OS version>
(<device model>) <appName>/<appVersion> <appType with qualifier>
<device ID> <list of features>
```

機能のリストには、2文字で示された Mobile SDK の機能が1つ以上含まれます。次に、一般的な例を示します。

```
SalesforceMobileSDK/7.1.0 android mobile/9.0 (Pixel 3) RestExplorer/5.0 HybridRemote
uid_xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxxx ftr_MU.AI.BW
```

実行時にユーザーエージェントを取得するには、SalesforceSDKManager.getUserAgent() メソッドをコールします。

isHybrid() メソッド

Mobile SDK アプリケーションで、ネイティブクライアントとハイブリッドクライアントの両方を提供するように設計されたライブラリを作成するとします。コールするアプリケーションのタイプに応じてライブラリコードが内部的に切り替わりますが、実行時に何らかの方法でアプリケーションタイプを判断する必要があります。コードでコール元アプリケーションタイプを判断するには、boolean 型の

SalesforceSDKManager.isHybrid() メソッドをコールします。true はハイブリッド、false はネイティブを意味します。

PasscodeManager クラス

PasscodeManager クラスは 10.0 で削除されるため、Mobile SDK 9.2 で非推奨になっています。Mobile SDK 9.2 では内部クラスが含まれるパスコードフローも再実装されるため、このクラスは無効になります。

Encryptor クラス

Encryptor ヘルパークラスは、SDK で必要なハッシュを使用して文字列の暗号化および復号化を行う静的ヘルパーメソッドを提供します。Mobile SDK で使用されるすべての鍵は Base64 でエンコードされている必要があることを、ネイティブアプリケーションが認識していることが重要です。他の暗号化パターンは使用できません。ハッシュを作成して、正しいエンコードを使用していることを確認する場合は、Encryptor クラスを使用します。

大部分の Encryptor メソッドは内部で使用されますが、このユーティリティは必要に応じてアプリケーションで自由に使用できます。たとえば、アプリケーションで独自のデータベースを実装する場合、制限のない暗号化および復号化ツールとして Encryptor を使用できます。

SalesforceActivity クラス、SalesforceListActivity クラス、および SalesforceExpandableListActivity クラス

SalesforceActivity、SalesforceListActivity、SalesforceExpandableListActivity は、ネイティブ SDK アクティビティの骨組みとなる基本クラスです。これらは、それぞれ `android.app.Activity`、`android.app.ListActivity`、`android.app.ExpandableListActivity` を拡張します。

これらの各アクティビティクラスには、次の1つの抽象メソッドが含まれています。

```
public abstract void onResume(RestClient client);
```

このメソッドは、`Activity.onResume()` メソッドをオーバーロードします。このメソッドもクラスによって実装されるものです。Mobile SDK スーパークラスの代理である `SalesforceActivityDelegate` は、`RestClient` インスタンスを作成したときにオーバーロードをコールします。このメソッドを使用して、渡されるクライアントをキャッシュし、そのクライアントを使用して REST 要求を実行します。たとえば、Kotlin Mobile SDK テンプレートアプリケーションでは、`MainActivity` クラスで、次のコードを使用します。

```
override fun onResume(client: RestClient) {  
    // Keeping reference to rest client  
    this.client = client  
  
    // Show everything  
    findViewById<ViewGroup>(R.id.root).visibility = View.VISIBLE  
}
```

ClientManager クラス

`ClientManager` は、`Android AccountManager` クラスと連動して、ユーザーアカウントを管理します。アプリケーションにとって特に重要なことは、`ClientManager` クラスが次の2つのメソッドを介して `RestClient` インスタンスへのアクセスを提供するという点です。

- `getRestClient()`
- `peekRestClient()`

`getRestClient()` メソッドは、Salesforce データを照会するための `RestClient` インスタンスを非同期で作成します。この場合、非同期とは、このメソッドが UI スレッドでの使用を目的としていることを意味します。
`peekRestClient()` メソッドは、UI 以外のコンテキストで使用する `RestClient` インスタンスを同期して作成します。

`RestClient` インスタンスを取得したら、このインスタンスを使用して、REST API コールを Salesforce に送信できます。

RestClient クラス

名前が示すように、`RestClient` クラスは、Android アプリケーションと Salesforce REST API の連絡役になります。

`RestClient` クラスの新しいインスタンスは明示的に作成しません。代わりに、`ClientManager` ファクトリークラスを使用して、`RestClient` インスタンスを取得します。`RestClient` インスタンスを取得したら、このインスタンスを使用して、REST API コールを Salesforce に送信できます。コールするメソッドは、UI コンテキストからコールするかどうかによって異なります。「[ClientManager クラス](#)」を参照してください。

次の `RestClient` メソッドを使用して、REST 要求を送信します。

- `sendAsync()` — `ClientManager.getRestClient()` をコールして `RestClient` インスタンスを取得した場合、このメソッドをコールします。
- `sendSync()` — `ClientManager.peekRestClient()` をコールして `RestClient` インスタンスを取得した場合、このメソッドをコールします。

`sendSync()` メソッド

要求に対してどの程度の情報を提供できるかに応じて、`RestClient.sendSync()` の 3 つのオーバーロードから選択できます。

`sendAsync()` メソッド

`RestClient.sendAsync()` メソッドは、`OkHttpClient Call` クラスの新しいインスタンスの `RestRequest` オブジェクトをラップします。次に、`Call` オブジェクトを要求キューに追加して、そのオブジェクトを返します。

待機中に要求をキャンセルするには、`Call` オブジェクトで `cancel()` をコールします。基になる要求キューオブジェクトにアクセスするには、`OkHttpClient` クラスを使用します。[「要求キューの管理」](#)の例を参照してください。

`RestRequest` クラス

`RestRequest` クラスは、Salesforce REST API および SOAP API によって提供される標準の Salesforce データ操作をネイティブに処理します。アプリケーションから提供されるデータに基づいて REST API 要求を作成および書式設定します。このクラスは Mobile SDK によって実装され、自身の特殊なインスタンスのファクトリーとして機能します。

サポートされる操作は次のとおりです。

操作またはリソース	パラメーター	説明
ユーザー情報		
SOQL Query	クエリ文字列、API バージョン、バッチサイズ	要求されたレコード項目のうち、SOQL クエリに一致したものが返されます。デフォルトでは、一度に最大 2,000 レコードが返されます。バッチサイズを指定した場合、レコードは、そのバッチサイズ以下で返されます。バッチサイズを指定しても、返されるバッチがそのサイズになるという保証はありません。
Versions	なし	Salesforce バージョンのメタデータを返します。

操作またはリソース	パラメーター	説明
オブジェクトレイアウト		
Batch request	API バージョン、エラーの発生時に停止するかどうかをバッチプロセスに指示するフラグ、サブ要求のリスト	RestRequest オブジェクトのリストで指定された最大 25 個のサブ要求のバッチを含む RestRequest オブジェクトを返します。各サブ要求は、レート制限に含まれます。
Resources	API バージョン	リソース名および URI を含む、指定された API バージョンで使用可能なリソースを返します。
Metadata	API バージョン、オブジェクト種別	オブジェクトの完全なメタデータコレクションを返します。
DescribeGlobal	API バージョン	組織で利用できるすべてのオブジェクトとそのメタデータのリストを返します。
Describe	API バージョン、オブジェクト種別	1つのオブジェクト種別の説明を返します。
Create	API バージョン、オブジェクト種別、値オブジェクトに対する項目名の対応付け	指定したオブジェクトに新しいレコードを作成します。
CompositeRequest	API バージョン、エラー条件ですべての要求をトランザクションブロックとして扱うかどうかを示す「オールオアナッシング」フラグ、サブ要求(値)とその参照ID(キー)のハッシュマップ	Composite 要求の実行に使用する RestRequest オブジェクトを返します。サブ要求の数に関係なく、各 Composite 要求は 1つの API コールと見なされます。『 REST API 開発者ガイド 』の「Composite」を参照してください。
Retrieve	API バージョン、オブジェクト種別、オブジェクトID、項目のリスト	オブジェクト ID でレコードを取得します。
Search	API バージョン、SOQL クエリ文字列	指定された SOQL 検索を実行します。
SearchResultLayout	API バージョン、オブジェクトのリスト	指定されたオブジェクトの検索結果レイアウト情報を返します。
SearchScopeAndOrder	API バージョン	ログインユーザーのデフォルトのグローバル検索範囲内にあるオブジェクトの順序付きリストを返します。

操作またはリソース	パラメーター	説明
SObject Tree	API バージョン、オブジェクト種別、sObject ツリーノードのリスト	指定された sObject ツリーノードのリストに基づいて、sObject ツリーの <code>RestRequest</code> オブジェクトを返します。
Update	API バージョン、オブジェクト種別、オブジェクト ID、値オブジェクトに対する項目名の対応付け、 <code>If-Unmodified-Since</code> 日付 (省略可能)	指定された対応付けでオブジェクトを更新します。条件付き更新の場合、Mobile SDK では <code>If-Unmodified-Since</code> 要求がサポートされています。
Upsert	API バージョン、オブジェクト種別、外部 ID 項目、外部 ID、値オブジェクトに対する項目名の対応付け	外部 ID が外部 ID 項目に現在存在しているかどうかに基づいて、外部データからオブジェクトを更新または挿入します。外部 ID 項目の名前を「 <code>Id</code> 」に設定し、外部 ID が <code>null</code> の場合、新規レコードが作成されます。
Delete	API バージョン、オブジェクト種別、オブジェクト ID	指定された ID を持つ特定のタイプのオブジェクトを削除します。
Notification	API バージョン、通知 ID	指定された ID を持つ通知を取得します。
NotificationUpdate	API バージョン、通知 ID、既読状況、参照済み状況	指定された ID を持ち、「 <code>read</code> 」(<code>null</code> でない場合) と「 <code>seen</code> 」(<code>null</code> でない場合) 状況の通知を更新します。
Notifications	API バージョン、通知数、指定された日付以前、指定された日付以後	指定された日付よりも前または後の通知を取得します。「 <code>date before</code> 」および「 <code>date after</code> 」は相互に排他的なパラメーターです。
NotificationsStatus	API バージョン	現在のユーザーの通知の状況を取得します。
NotificationsUpdate	API バージョン、通知 ID、指定された日付以前、既読状況、参照済み状況	指定された ID を持ち、「 <code>read</code> 」(<code>null</code> でない場合) と「 <code>seen</code> 」(<code>null</code> でない場合) 状況の通知、または指定された日付よりも前の通知を更新します。ID と「 <code>date before</code> 」は相互に排他的なパラメーターです。

`RestRequest` のインスタンスを作成するには、次の `RestRequest` 静的ファクトリーメソッドを使用します。

- `getRequestForUserInfo()`

- `getRequestForVersions()`
- `getRequestForResources()`
- `getRequestForDescribeGlobal()`
- `getRequestForMetadata()`
- `getRequestForDescribe()`
- `getRequestForCreate()`
- `getRequestForRetrieve()`
- `getRequestForUpdate()`
- `getRequestForUpsert()`
- `getRequestForDelete()`
- `getRequestForQuery()`
- `getRequestForSearch()`
- `getRequestForSearchScopeAndOrder()`
- `getRequestForSearchResultLayout()`
- `getRequestForObjectLayout()`
- `getBatchRequest()`
- `getCompositeRequest()`
- `getRequestForSObjectTree()`
- `getRequestForNotification()`
- `getRequestForNotificationUpdate()`
- `getRequestForNotifications()`
- `getRequestForNotificationsStatus()`
- `getRequestForNotificationsUpdate()`

これらのメソッドはそれぞれ `RestRequest` オブジェクトを返します。要求を送信するには、`RestRequest` オブジェクトを `RestClient.sendAsync()` または `RestClient.sendSync()` に渡します。「[REST API の使用](#)」を参照してください。

 **例:** サンプルコールについては、github.com/forcedotcom/SalesforceMobileSDK-Android にある `/libs/test/SalesforceSDKTest/src/com/salesforce/androidsdk/rest/RestRequestTest.java` を参照してください。

関連トピック:

[サポート対象の Salesforce API](#)

FileRequests クラス

`FileRequests` クラスは、ファイル操作要求を構築するメソッドを提供します。各メソッドは、新しい `RestRequest` オブジェクトを返します。アプリケーションは、このオブジェクトを Salesforce サービスに送信して要求を処理します。たとえば、次のコードスニペットは `ownedFilesList()` メソッドをコールして `RestRequest` オブジェクトを取得します。次に、`RestClient.sendAsync()` を使用して `RestRequest` オブジェクトをサーバーに送信します。

Kotlin

```
val ownedFilesRequest = FileRequests.ownedFilesList(null, null)
val client = this.client
client?.sendAsync(ownedFilesRequest, object : AsyncRequestCallback {
    // Do something with the response
})
```

Java

```
RestRequest ownedFilesRequest = FileRequests.ownedFilesList(null, null);
RestClient client = this.client;
client.sendAsync(ownedFilesRequest, new AsyncRequestCallback() {
    // Do something with the response
});
```

 **メモ:** この例では、`null` を最初のパラメーター (`userId`) に渡します。この値で、コンテキスト (ログインしている) ユーザーの ID を使用するように `ownedFilesList()` メソッドに指示します。 `pageNum` パラメーターの 2 番目の `null` で、結果の最初のページを取得するようにメソッドに指示します。

`FileRequests` メソッドについての詳細は、「[ファイルとネットワーキング](#)」を参照してください。

メソッド

`FileRequests` メソッドについての詳細は、「[Package com.salesforce.androidsdk.rest.files](#)」を参照してください。REST 要求および応答の本文についての詳細は、<http://www.salesforce.com/us/developer/docs/chatterapi> の『**Connect REST API 開発者ガイド**』 > 「リソース」 > 「Files リソース」を参照してください。

OkHttp: 基盤となるネットワークライブラリ

Mobile SDK 4.2 以降では、Android REST 要求システムは、基盤となるアーキテクチャとして Square Open Source のオープンソース外部ライブラリの `OkHttp` (v3.2.0) を使用します。過去のリリースの `Google Volley` ライブラリは、このライブラリに置き換えられます。そのため、Mobile SDK では、`WrappedRestRequest` クラスが定義されなくなりました。

 **例:** 次の例に、`OkHttpClient` を使用したいいくつかの一般的なネットワーク操作の実行方法を示します。
一般的なインポート

```
import okhttp3.Headers;
import okhttp3.HttpUrl;
import okhttp3.OkHttpClient;
import okhttp3.Call;
import okhttp3.Dispatcher;
import okhttp3.Request;
import okhttp3.RequestBody;
import okhttp3.Response;
```

現在の `OkHttp` クライアントハンドルを取得する

現在の `RestClient` インスタンスが使用している `OkHttpClient` のハンドルを取得するには、次のように指定します。

Kotlin

```
var okClient = restClient.getOkHttpClient()
```

Java

```
OkHttpClient okClient = restClient.getOkHttpClient();
```

OkHttp ディスパッチャーを取得する**Kotlin**

```
var dispatcher = restClient.getOkHttpClient().dispatcher()
```

Java

```
Dispatcher dispatcher = restClient.getOkHttpClient().dispatcher();
```

待機中のすべてのコールをキャンセルする**Kotlin**

```
var dispatcher = restClient.getOkHttpClient().dispatcher()  
dispatcher.cancelAll()
```

Java

```
Dispatcher dispatcher = restClient.getOkHttpClient().dispatcher();  
dispatcher.cancelAll();
```

OkHttp ハンドルを REST 要求に格納する**Kotlin**

```
var call = restClient.sendAsync(restRequest, callback)
```

Java

```
Call call = restClient.sendAsync(restRequest, callback);
```

格納されたハンドルを使用して特定の REST 要求をキャンセルする**Kotlin**

```
var call = restClient.sendAsync(restRequest, callback)
```

Java

```
Call call = restClient.sendAsync(restRequest, callback);  
...  
call.cancel();
```

詳細は、square.github.io/okhttp/を参照してください。

UI クラス

`com.salesforce.androidsdk.ui` パッケージのアクティビティは、すべての Mobile SDK アプリケーションに共通する UI リソースを表しています。XML を介して、これらのリソースのスタイル、スキン、テーマを設定したり、カスタマイズしたりできます。これらのリソースを上書きした場合、Mobile SDK で互換性の破綻が生じたときには、アプリケーション側でそれらのリソースを維持する必要があります。SalesforceActivity、SalesforceListActivity、SalesforceExpandableListActivity を除き、実行時にリソースを置き換えることを目的として、これらのアクティビティクラスを上書きしないでください。

LoginActivity クラス

LoginActivity は、ログイン画面を定義します。ログインワークフローについては、説明する価値がありません。それは、ログインワークフローでアクティビティパッケージの他の2つのクラスが説明されているからです。ログインアクティビティで、[メニュー] ボタンを押すと、[Cookie をクリア]、[再読み込み]、[サーバーを変更] の3つのオプションが表示されます。[サーバーを変更] は、[本番] および [Sandbox] サーバーと、[接続を追加] オプションを表示する ServerPickerActivity クラスのインスタンスを起動します。[接続を追加] をクリックした場合、ServerPickerActivity によって CustomServerURLEditor クラスのインスタンスが起動されます。このクラスは、カスタムサーバーの名前を入力できるポップオーバーダイアログを表示します。

その他の UI クラス

SalesforceDroidGapActivity および SalesforceGapViewClient は、ハイブリッドアプリケーションでのみ使用します。これらのクラスは、ネイティブ API 開発作業には影響しません。

ユーティリティクラス

util パッケージの大部分のクラスは内部で使用されますが、いくつかのクラスはサードパーティ開発者にも利点があります。

クラス	説明
EventsObservable	Mobile SDK for Android が伝搬するすべてのイベントのリストについては、ソースコードを参照してください。
EventsObserver	イベントを傍受するにはこのインターフェースを実装します。この機能は、特定のタイプのイベント発生時に特殊な操作を実行する場合に便利です。
UriFragmentParser	この静的ヘルパークラスは直接コールできます。このクラスは、特定の URI を解析して、そのパラメーターを一連のキー/値ペアに分離し、対応付けで返します。

ForcePlugin クラス

`com.salesforce.androidsdk.phonegap` パッケージのすべてのクラスは、ハイブリッドアプリケーションのサポートを目的としています。これらのクラスの大部分には、ネイティブコードにアクセスする Javascript プ

ラグインが実装されています。これらの Mobile SDK プラグインの基本クラスは `ForcePlugin` です。Mobile SDK アプリケーションに独自の Javascript プラグインが必要な場合、`ForcePlugin` を拡張して、抽象 `execute()` 関数を実装します。

`ForcePlugin` は、Javascript フレームワークと連動する `CordovaPlugin` を拡張します。これにより、ネイティブ関数をコールできる Javascript モジュールを作成できるようになります。`PhoneGap` は両方のブリッジになります。`CordovaPlugin` でネイティブプラグインを作成し、それを複製する Javascript ファイルを作成します。`Cordova` は、いずれかのプラグインの Javascript 関数をスクリプトでコールすると、プラグインの `execute()` 関数をコールします。

パスコードの使用

パスコードとは、アプリケーションのログインセキュリティの追加のレイヤーとして使用できる、顧客定義のトークンのことです。必要に応じて、Salesforce システム管理者は、ログイン後にパスコードを必要とするように接続アプリケーションを設定することができます。この設定では、たとえばフォアグラウンドに戻るときにバックグラウンドアプリケーションでパスコードの入力を要求することが必須になります。接続アプリケーションでモバイルアプリケーションのパスコードが要求される場合、Mobile SDK 9.2 以降ではデバイスのシステムのパスコードを使用します。

パスコードを確認するために、Mobile SDK では顧客の設定済みの確認モード(たとえば、生体認証、パターン、PIN、またはパスワード)を使用するロック画面を表示します。デバイスのパスコードが設定されていない場合、Mobile SDK では顧客にデバイスでサポートされているセキュアな入力モードのいずれかを使用してパスコードを作成するよう要求します。接続アプリケーションでモバイルパスコードが要求されない場合、Mobile SDK ではパスコードの確認ステップがスキップされます。Mobile SDK では、すべてのログイン画面とパスコードロック画面、および認証ハンドシェイクの処理が行われます。これらの画面を表示するために、アプリケーションで何らかの操作をする必要はありません。

モバイルパスコードポリシー

各 Mobile SDK アプリケーションでは、特定の Salesforce 組織の接続アプリケーションのコンシューマー鍵と OAuth コールバック URL をハードコード化します。Mobile SDK では、その組織の指定された接続アプリケーションの設定可能なパスコード要件が優先されます。Mobile SDK バージョン 9.2 以降では、PIN 長などの組織設定が無視され、代わりにデバイス設定が利用されます。同様に、不正なパスコード入力はモバイルオペレーティングシステムの標準手順に従って処理されます。

 **メモ:** Mobile SDK バージョン 9.2 以降では、組織の接続アプリケーションの [次の時間が経過したらアプリケーションをロック] 設定が無視され、アイドル状態になった後にデバイスをロックするデバイス設定が利用されました。Mobile SDK バージョン 10.1.1 以降では、接続アプリケーションの [次の時間が経過したらアプリケーションをロック] 設定が再び利用されています。これが設定されている場合、モバイルアプリケーションがタイムアウト期間より長くバックグラウンドで実行されているとロックされます。モバイルアプリケーションを有効にするとロックがかかります。アプリケーションのロック解除は従来どおりです。

顧客がアプリケーションを使用して異なる組織にログインする場合、Mobile SDK では指定された接続アプリケーション設定を取得できません。そのため、その顧客にはパスコードプロンプトが一切表示されません。

ロック画面のマルチユーザー動作

複数のユーザーが同じデバイスの同じアプリケーションにログインしている場合、ロック画面の動作は次のようになります。

1. パスコードが必要なアプリケーションを再開するときに、顧客は最初にロック画面でモバイルオペレーティングシステムを通じて認証を行うよう要求されます。
2. 顧客が認証をキャンセルすると、[ログアウト] ボタンと [ロック解除を再試行] ボタンがロック画面に表示されます。
3. [ログアウト] ボタンは、ロック画面を必須にしている顧客の場合のみ機能します。
4. ロック画面を必須にした最後のユーザーがログアウトすると、Mobile SDK にロック画面が表示されなくなります。

生体認証の無効化

Mobile SDK 9.2 以降で作成されたアプリケーションでは、Salesforce 接続アプリケーションの生態認証設定が無視されます。代わりに、顧客はデバイス設定で認証モードを自分で設定できます。

関連トピック

- [生体認証](#) (ページ 472)

リソースの処理

Mobile SDK テンプレートアプリケーションでは、リソースが **SalesforceSDK > res** プロジェクトフォルダーの XML ファイルに保存されています。これらのリソースの多くは、それらのファイルを変更することでカスタマイズできます。

/res フォルダーのリソースは、次のようなカテゴリに分類されます。

- 描画可能 — 背景、影付き、画像リソース (PNG ファイルなど)
- レイアウト — 可視コンポーネント (パスコード画面など) の画面設定
- メニュー — ログイン画面のサブビューの画面 (オプションメニューなど)
- 値 — Mobile SDK で使用される文字列、色、サイズ
- XML — ビジュアル以外の設定 (ログインサーバーの設定や MDM のランタイムアプリケーションの制限など)

描画可能リソース、レイアウトリソース、および値リソースは、さまざまなフォーム要素に対応するフォルダーにさらに分類されます。これらのカテゴリは、各種デバイス種別および画面解像度に対応します。各カテゴリはそのフォルダー名で定義されるため、すべてのバージョンで同じリソースファイル名を使用できます。たとえば、開発者が `icon1.png` という名前のアイコンをさまざまなサイズで提供する場合、スマートフォン、ローエンドフォン、タブレットに対応する各フォルダーにアイコンを配置します。各フォルダーで、このファイル名は `icon1.png` になります。フォルダー名の語幹は同じですが、サフィックスが異なります。

次の表に、フォルダー名とサフィックスを示します。

フォルダー名	使用方法
drawable	描画可能リソースの汎用バージョン
drawable-hdpi	高解像度 (スマートフォン向け)
drawable-ldpi	低解像度 (ローエンド機能端末向け)
drawable-mdpi	中解像度 (ローエンド機能スマートフォン)
drawable-xhdpi	超高密度画面 (～ 320 dpi) のリソース
drawable-xlarge	タブレット画面 (横方向)
drawable-xlarge-port	タブレット画面 (縦方向)
drawable-xxhdpi-port	超超高密度画面 (～ 480 dpi) のリソース
layout	レイアウトの汎用バージョン
menus	[接続を追加] ダイアログおよび電話のログインメニュー
values	汎用スタイルおよび値
xml	一般的なアプリケーション設定

コンパイラーは、ターゲットデバイス設定に一致する名前のフォルダーでリソースを検索します。要求されたリソースが想定されるフォルダーにない場合 (ターゲットデバイスはタブレットであるが、要求されたアイコンが `drawables-xlarge` または `drawables-xlarge-port` フォルダーにない場合など)、コンパイラーは汎用 `drawable` フォルダーでアイコンファイルを検索します。

レイアウト

Mobile SDK のレイアウトは、すべてのアプリケーションが使用する画面リソースを指定します。たとえば、レイアウトは、ログインおよびパスワードを処理するダイアログボックスを設定します。

`layout` の XML ノードの名前は、レイアウトが指定するコントロールのタイプを示します。たとえば、`res/layout/sf__passcode.xml` にある次の `TextView` ノードは、テキスト編集コントロールを指定します。

```
<TextView
    android:id="@+id/sf__passcode_title"
    style="@style/SalesforceSDK.Passcode.Text.Title"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="17sp"
    android:textColor="@color/sf__passcode_text_color"
    android:textStyle="bold"/>
```

この場合、`TextView` コントロールは、参照のみです。style 属性は、リソースの他の場所で定義されたグローバルスタイルを参照します。所定の場所で style 属性を指定する代わりに、中央ファイルでスタイルを定義し、必要な場所でその属性を参照します。値 `@style/SalesforceSDK.Passcode.Text.Title` は、

res/values/sf__styles.xml で定義された SDK 所有のスタイルを参照します。次に、スタイルの定義を示します。

```
<style name="SalesforceSDK.Passcode.Text.Title">
    <item name="android:layout_marginTop">@dimen/sf__passcode_title_margin_top</item>
    <item name="android:layout_marginBottom">@dimen/sf__passcode_title_margin_bottom</item>
</style>
```

いずれかの独自のスタイルへの参照で style 属性を上書きできます。sf__styles.xml を変更する代わりに、xyzcorp__styles.xml などの別ファイルでスタイルを定義します。res/values フォルダー (汎用デバイススタイル) または res/values-xlarge フォルダー (タブレットデバイス) にファイルを配置します。

値

res/values および res/values-xlarge フォルダーには、サイズ、色、文字列のリソースやカスタムスタイルなどのスタイルコンポーネントの定義が含まれています。このフォルダーのファイル名は、リソースまたはスタイルコンポーネントのタイプを示します。

ファイル名	次の文字列を含む
sf__colors.xml	
sf__attr.xml	パスコード画面で使用される色と整数値
sf__colors.xml	Mobile SDK スタイルで参照される色
sf__dimens.xml	Mobile SDK スタイルで参照されるサイズ
sf__strings.xml	Mobile SDK スタイルで参照される文字列。エラーメッセージを上書きできます。
sf__styles.xml	Mobile SDK で使用されるビジュアルスタイル
strings.xml	アプリケーション定義の文字列

strings.xml の値を上書きできます。独自の値を提供するには、会社やプロジェクトを反映するファイル名プレフィックスを使用して、同じフォルダーに新しいファイルを作成します。たとえば、開発者のプレフィックスがXYZの場合、XYZ__styles.xml という名前の新しいファイルで sf__styles.xml を上書きできます。

その他のリソース

その他の 2 つのフォルダーには、Mobile SDK リソースが含まれています。

- res/menu は、内部的に使用されるメニューを定義します。アプリケーションで新しいメニューを定義する場合、新しいファイルのこの場所にリソースとして追加します。
- res/xml には、編集する必要があるファイル servers.xml が含まれています。このファイルでは、デフォルトの本番サーバーおよび Sandbox サーバーを組織のログインサーバーに変更する必要があります。このフォルダーにあるその他のファイルは内部で使用されます。authenticator.xml ファイルはアカウント

ト認証リソースを設定し、`config.xml` ファイルはハイブリッドアプリケーションの PhoneGap プラグインを定義します。

関連トピック:

[Android リソース](#)

REST API の使用

Mobile SDK アプリケーションは、Salesforce 組織のデータをクエリ、説明、作成、更新するために、Salesforce REST API をコールします。Salesforce REST API は、SOQL および SOSL 文字列を優先しますが、JSON または XML 形式のデータを受け入れて返すこともできます。Mobile SDK は、低レベルの HTTP 設定を処理するメソッドの標準の Salesforce REST 要求をラップします。他の Salesforce API については、Mobile SDK に、カスタム要求オブジェクトを手動で作成し、応答を受信するメソッドがあります。Mobile SDK REST API メソッドを使用して、認証されていない API コールや外部 API コールを実行することもできます。

Salesforce では、種類が増え続ける REST API をサポートしています。サポートされる REST API の概要については、Salesforce ヘルプの「[どの API を使用するか?](#)」を参照してください。標準の REST API についての詳細は、『[REST API 開発者ガイド](#)』を参照してください。

REST 操作のコーディング

Android ネイティブアプリケーションにより、最小限のコーディングで REST コールを介して Salesforce データにアクセスできます。`com.salesforce.androidsdk.rest` パッケージのクラスは、通信チャンネルを初期化して、低レベルの HTTP プライミングをカプセル化します。これらのクラスはすべて Mobile SDK によって実装されていて、次のものがあります。

- `ClientManager` — `RestClient` インスタンスのファクトリーとして機能します。また、アカウントログインや Salesforce サーバーとのハンドシェイクも処理します。
- `RestClient` — REST API 要求を Salesforce サーバーに送信するためのプロトコルを処理します。

`RestClient` のインスタンスを直接作成しないでください。代わりに、`ClientManager.getRestClient()` メソッドをコールします。

- `RestRequest` — 提供したデータに基づいて書式設定された REST API 要求を表します。また、自身のインスタンスのファクトリーとしても機能します。

 **重要:** `RestRequest` のインスタンスを直接作成しないでください。代わりに、適切な `RestRequest` の静的 getter 関数 (`RestRequest.getRequestForCreate()` など) をコールします。

- `RestResponse` — 要求された書式での応答コンテンツを含みます。`RestRequest` クラスは、`RestResponse` のインスタンスを作成し、`RestClient.AsyncRequestCallback` インターフェースの実装を介してアプリケーションに返します。

次に、UI スレッドで REST クラスを使用する場合の基本手順を示します。

1. `ClientManager` のインスタンスを作成します。
 - a. `SalesforceSDKManager.getInstance().getAccountType()` メソッドを使用して、`ClientManager` コンストラクターの第 2 引数として渡す値を取得します。

- b. ClientManager コンストラクターの LoginOptions パラメーターの場合、SalesforceSDKManager.getInstance().getLoginOptions() をコールします。
2. ClientManager.RestClientCallback インターフェースを実装します。
3. ClientManager.getRestClient() をコールして RestClient インスタンスを取得し、RestClientCallback 実装のインスタンスに渡します。次のコードは、RestClientCallback をインラインで実装およびインスタンス化しています。

Kotlin

```
val accountType = SalesforceSDKManager.getInstance().accountType

val loginOptions = SalesforceSDKManager.getInstance().loginOptions
// Get a rest client
ClientManager(this, accountType, loginOptions,
    SalesforceSDKManager.getInstance().shouldLogoutWhenTokenRevoked()).
    getRestClient(this, object : RestClientCallback() {
fun authenticatedRestClient(client: RestClient?) {
    if (client == null) {
        SalesforceSDKManager.getInstance().logout(this@MainActivity)
        return
    }
    // Cache the returned client
    this@MainActivity.client = client
}
}
)
```

Java

```
String accountType =
    SalesforceSDKManager.getInstance().getAccountType();

LoginOptions loginOptions =
    SalesforceSDKManager.getInstance().getLoginOptions();
// Get a rest client
new ClientManager(this, accountType, loginOptions,
    SalesforceSDKManager.getInstance().
    shouldLogoutWhenTokenRevoked()).
    getRestClient(this, new RestClientCallback() {
        @Override
        public void
        authenticatedRestClient(RestClient client) {
            if (client == null) {
                SalesforceSDKManager.getInstance().
                    logout(MyActivity.this);
                return;
            }
            // Cache the returned client
            MyActivity.this.client = client;
        }
    })
);
```

4. 静的 `RestRequest()` getter メソッドをコールして、ニーズに合った `RestRequest` オブジェクトを取得します。たとえば、Salesforce オブジェクトの説明を取得する場合は次のようになります。

```
final RestRequest request = RestRequest.getRequestForDescribe(apiVersion, objectType);
```

5. 前のステップで取得した `RestRequest` オブジェクトを `RestClient.sendAsync()` または `RestClient.sendSync()` に渡します。UI スレッドの場合、`sendAsync()` をコールします。
- `ClientManager.AsyncRequestCallback` インターフェースを実装します。
 - 実装のインスタンスを `sendAsync()` メソッドに渡します。
 - `AsyncRequestCallback.onSuccess()` メソッドを介して、書式設定された応答を取得します。応答を使用する前に、`RestResponse.isSuccess()` をコールして、応答が有効であることを再確認します。

次のコードは、`AsyncRequestCallback` をインラインで実装およびインスタンス化しています。

Kotlin

```
private fun sendFromUiThread(restRequest: RestRequest) {
    client.sendAsync(restRequest, object : AsyncRequestCallback {
        private val start = System.nanoTime()
        override fun onSuccess(request: RestRequest, result: RestResponse) {
            // Consume before going back to main thread
            // Not required if you don't do main (UI) thread tasks here
            result.consumeQuietly()
            runOnUiThread {
                // Network component doesn't report app layer status.
                // Use the Mobile SDK RestResponse.isSuccess() method to check
                // whether the REST request itself succeeded.
                if (result.isSuccess) {
                    try {
                        // Do something with the result
                    } catch (e: Exception) {
                        printException(e)
                    }
                }

                EventsObservable.get().notifyEvent(EventType.RenditionComplete)
            }
        }

        override fun onError(exception: Exception) {
            printException(exception)
            EventsObservable.get().notifyEvent(EventType.RenditionComplete)
        }
    })
}
```

Java

```
private void sendFromUiThread(RestRequest restRequest) {
    client.sendAsync(restRequest, new AsyncRequestCallback() {
        private long start = System.nanoTime();
        @Override
```

```

public void onSuccess(RestRequest request, final RestResponse result) {
    // Consume before going back to main thread
    // Not required if you don't do main (UI) thread tasks here
    result.consumeQuietly();
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            // Network component doesn't report app layer status.
            // Use the Mobile SDK RestResponse.isSuccess() method to check
            // whether the REST request itself succeeded.
            if (result.isSuccess()) {
                try {
                    // Do something with the result
                }
                catch (Exception e) {
                    printException(e);
                }
                EventsObservable.get().notifyEvent(EventType.RenditionComplete);
            }
        }
    });
}

@Override
public void onError(Exception exception)
{
    printException(exception);
    EventsObservable.get().notifyEvent(EventType.RenditionComplete);
}

});
}

```

サービスから `sendSync()` メソッドをコールする場合、次の点を変更して同じ手順を実行します。

1. `RestClient` インスタンスを取得するには、`ClientManager.getRestClient()` ではなく `ClientManager.peekRestClient()` をコールします。
2. `sendSync()` メソッドの戻り値から、書式設定された REST 応答を取得します。

REST 応答状況の確認

アプリケーションの `onSuccess()` コールバックメソッドに到達した REST レスポンスが示すのは、ネットワークコールが失敗しなかったことのみです。この上位レベルの状況はアプリケーションレベルの成功または失敗には関係しません。

Mobile SDK for Android では、`RestResponse` オブジェクトによって基になる `okHttp3.Response` がラップされます。`RestResponse` では、より防御性の高いコーディングを行うために、応答の詳細を検証するための次の有用なメソッドが提供されています。

public isSuccess()

public static boolean isSuccess(int statusCode)

HTTP 応答の状況コードまたは指定したコードが、200 から 299 の間の場合には、アプリケーションレベル成功を示し `true` を返します。

```
public int getStatusCode()
```

応答状況コードを返します。

```
public String getContentType()
```

値が検出された場合に、`content-type` ヘッダーを返します。

```
public Map<String, List<String>> getAllHeaders()
```

この応答に関連するすべてのヘッダーを返します。

```
public Response getRawResponse()
```

基になる `okHttp3.Response` オブジェクトを返します。

Batch 要求および Composite 要求

一括および複合 API は、複数の要求を単一のコールで処理するため、特別な処理が必要です。Mobile SDK のクラスにより、これらの複雑な要求を簡単に作成および設定できます。

Batch 要求クラスと Composite 要求クラス

- 要求:

```
BatchRequest
CompositeRequest
```

- ビルダー:

```
BatchRequestBuilder
CompositeRequestBuilder
```

- 応答:

```
BatchResponse
CompositeRequestBuilder
```

これらのクラスにより、Batch 要求と Composite 要求を容易に作成できます。これらのクラスを使用する手順は、次のとおりです。

1. ビルダーインスタンスを作成します。Batch 要求では、必要に応じて `haltOnError` プロパティを `true` に設定します。

Kotlin

```
val builder = BatchRequest.BatchRequestBuilder()
// Optional; defaults to false
builder.setHaltOnError(true)
```

Java

```
BatchRequest.BatchRequestBuilder builder =
    new BatchRequest.BatchRequestBuilder();
// Optional; defaults to false
builder.setHaltOnError(true);
```

Composite 要求では、必要に応じて `allOrNone` ロールバックプロパティを `true` に設定します。

Kotlin

```
val builder = CompositeRequest.CompositeRequestBuilder()
// Optional; defaults to false
builder.setAllOrNone(true)
```

Java

```
CompositeRequest.CompositeRequestBuilder builder =
    new CompositeRequest.CompositeRequestBuilder();
// Optional; defaults to false
builder.setAllOrNone(true);
```

2. REST 要求を作成する場合、REST 要求はビルダーオブジェクトに追加します。Batch 要求の場合:

Kotlin

```
builder.addRequest(request)
```

Java

```
builder.addRequest(request);
```

Composite 要求では、『REST API 開発者ガイド』の説明に従って、参照 ID も指定します。この ID は、RestRequest オブジェクトを追加するときに指定します。

Kotlin

```
builder.addRequest(referenceId, request)
```

Java

```
builder.addRequest(referenceId, request);
```

3. 準備が整ったら、ビルダーオブジェクトの build メソッドをコールします。

Kotlin

```
builder.build(ApiVersionStrings.VERSION_NUMBER)
```

Java

```
builder.build(ApiVersionStrings.VERSION_NUMBER);
```

各 build メソッドにより、共有の RestClient.sendAsync() メソッドを通じて送信できる特殊な RestRequest オブジェクト (BatchRestRequest または CompositeRestRequest インスタンス) が返されます。

 **ヒント:** デフォルトの引数として古いバージョンの API を使用するには、「v42.0」形式のリテラル文字列を渡します。

4. Batch 要求および Composite 要求に対する応答は、応答クラス (BatchResponse または CompositeResponse) のインスタンスとして返されます。

関連トピック

- 『REST API 開発者ガイド』の「Batch」

- 『REST API 開発者ガイド』の「Composite」

認証されていない REST 要求

特定のケースでは、ユーザーが認証される前に一部のアプリケーションで REST コールを実行する必要があります。その他のケースでは、アプリケーションで、Salesforce 認証を必要としない Salesforce 外のサービスにアクセスする必要があります。そのような要件を実装するには、認証トークンを必要としない特殊な `RestClient` インスタンスを使用します。

Android で認証されていない `RestClient` を取得するには、次のいずれかの `ClientManager` ファクトリメソッドを使用します。

```
/**
 * Method to create an unauthenticated RestClient asynchronously
 * @param activityContext
 * @param restClientCallback
 */
public void getUnauthenticatedRestClient(Activity activityContext, RestClientCallback restClientCallback);
```

```
/**
 * Method to create an unauthenticated RestClient.
 * @return
 */
public RestClient peekUnauthenticatedRestClient();
```

 **メモ:** これらの `RestClient` オブジェクトのいずれかを介して送信された REST 要求では、フルパスの URL が必要になります。Mobile SDK では、認証されていないエンドポイントの先頭にインスタンス URL は追加されません。

 **例:**

Kotlin

```
val unauthenticatedRestClient = clientManager.peekUnauthenticatedRestClient()
val request = RestRequest(RestMethod.GET,
    "https://api.spotify.com/v1/search?q=James%20Brown&type=artist", null)
val response = unauthenticatedRestClient.sendSync(request)
```

Java

```
RestClient unauthenticatedRestClient = clientManager.peekUnauthenticatedRestClient();
RestRequest request = new RestRequest(RestMethod.GET,
    "https://api.spotify.com/v1/search?q=James%20Brown&type=artist", null);

RestResponse response = unauthenticatedRestClient.sendSync(request);
```

ネイティブ Android アプリケーションのログインの延期

forcedroid で Mobile SDK アプリケーションを作成する場合、多数の無料の標準機能を提供するテンプレートアプリケーションがプロジェクトのベースになります。たとえば、認証を実装する必要はなく、ログインとパスワードの処理はランチャーアクティビティに組み込まれています。この設計は大部分のアプリケーションで適

切に機能し、この無償のコードにより、大幅に時間を節約できます。ただし、forcedroid アプリケーションを作成した後で、ランチャーアクティビティの実行後のある時点まで Salesforce 認証を延期する必要がある場合があります。

認証の延期は、テンプレートアプリケーションの組み込み機能を維持したまま簡単に実装できます。次に、ガイドラインと注意事項を示します。

- ランチャーアクティビティ (テンプレートアプリケーションの `MainActivity`) を、次のどの Mobile SDK アクティビティも拡張しないアクティビティに置き換えます。

- `SalesforceActivity`
- `SalesforceListActivity`
- `SalesforceExpandableListActivity`

このルールは、Salesforce で認証する前に実行される他のすべてのアクティビティにも同様に適用されます。

- `peekRestClient()` または `getRestClient()` `ClientManager` メソッドは、ランチャーアクティビティや他の認証前のアクティビティからコールしないでください。
- `TemplateApp` クラスの `initNative()` コールを変更しないでください。これは、認証後に起動するアクティビティクラス (テンプレートアプリケーションの `MainActivity`) を指し示している必要があります。
- Salesforce で認証する準備が整ったら、`MainActivity` クラスを起動します。

次の例は、Salesforce 認証の前に非 Salesforce アクティビティを置き換える方法を示します。もちろん、前述のガイドラインと注意事項に従って、その他の認証前のアクティビティでこの例を拡張することも可能です。この例は、[MobileSyncExplorer サンプルアプリケーション](#)に基づいています。

- アプリケーションの認証前のランディングページの XML レイアウトを作成します。たとえば、次のレイアウトファイル `launcher.xml` には、ログインフローをトリガーするボタンのみがあります。

-  **メモ:** 次の例では、次のように `res/strings.xml` ファイルで定義される文字列リソース `@string/login` を定義します。

```
<string name="login">Login</string>
```

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="@android:color/white"
    android:id="@+id/root">

    <Button android:id="@+id/login_button"
        android:layout_width="80dp"
        android:layout_height="60dp"
        android:text="@string/login"
        android:textColor="@android:color/black"
        android:textStyle="bold"
        android:gravity="center"
        android:layout_gravity="center"
        android:textSize="18sp"
```

```
        android:onClick="onLoginClicked" />
    </LinearLayout>
```

- ランディング画面アクティビティを作成します。たとえば、ここでは `LauncherActivity` という名前のランディング画面アクティビティを作成します。この画面は、`launcher.xml` で定義される XML レイアウトを拡張します。このクラスで Salesforce アクティビティを拡張したり、`peekRestClient()` または `getRestClient()` をコールしたりしないでください(これらのコールで認証フローがトリガーされるため)。ユーザーがログインボタンをタップすると、`onLoginClicked()` ボタンハンドラーによって `MainActivity` が起動され、ログインが行われます。

Kotlin

```
class LauncherActivity : Activity() {
    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.launcher)
    }

    /**
     * Callback received when the 'Delete' button is clicked.
     *
     * @param v View that was clicked.
     */
    fun onLoginClicked(v: View) {
        /*
         * TODO: Add logic here to determine if we are already
         * logged in, and skip this screen by calling
         * 'finish()', if that is the case.
         */
        val mainIntent = Intent(this, MainActivity::class.java)
        mainIntent.addCategory(Intent.CATEGORY_DEFAULT)
        startActivity(mainIntent)
        finish()
    }
}
```

Java

```
package com.salesforce.samples.mobilesyncexplorer.ui;

import com.salesforce.samples.mobilesyncexplorer.R;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class LauncherActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.launcher);
    }
}
```

```

/**
 * Callback received when the 'Delete' button is clicked.
 *
 * @param v View that was clicked.
 */
public void onLoginClicked(View v) {
    /**
     * TODO: Add logic here to determine if we are already
     * logged in, and skip this screen by calling
     * 'finish()', if that is the case.
     */
    final Intent mainIntent =
        new Intent(this, MainActivity.class);
    mainIntent.addCategory(Intent.CATEGORY_DEFAULT);
    startActivity(mainIntent);
    finish();
}
}

```

3. AndroidManifest.xml を変更し、アプリケーションの初回起動時に起動されるアクティビティとして `LauncherActivity` を指定します。

```

<!-- Launcher screen -->
<activity android:name=
"com.salesforce.samples.mobilesyncexplorer.ui.LauncherActivity"
    android:label="@string/app_name"
    android:theme="@style/SalesforceSDK.ActionBarTheme">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category
            android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<!-- Main screen -->
<activity android:name=
"com.salesforce.samples.mobilesyncexplorer.ui.MainActivity"
    android:label="@string/app_name"
    android:theme="@style/SalesforceSDK.ActionBarTheme">
    <intent-filter>
        <category android:name=
            "android.intent.category.DEFAULT" />
    </intent-filter>
</activity>

```

アプリケーションを起動すると、`LauncherActivity` 画面が表示されます。ログインボタンをクリックすると、Salesforce 認証フローが開始します。認証が完了すると、アプリケーションによって `MainActivity` が起動します。

Android テンプレートアプリケーション: 詳細

新しい Android アプリケーションを作成するには、`forcedroid create` コマンドで、2つの Mobile SDK テンプレートプロジェクトのうちの1つを再設定します。

- **AndroidNativeTemplate** — Java を使用する基本的な Mobile SDK ネイティブアプリケーションを実装します。
- **AndroidNativeTemplateKotlin** — Kotlin を使用する基本的な Mobile SDK ネイティブアプリケーションを実装します。

これらのプロジェクトは、両方とも github.com/forcedotcom/SalesforceMobileSDK-Templates GitHub リポジトリで入手できます。

このテンプレートアプリケーションを学習すれば、Mobile SDK for Android で作成されたネイティブアプリケーションを迅速に理解できます。

テンプレートプロジェクトは、`MainApplication` と `MainActivity` の2つのクラスを定義します。

- `MainApplication` クラスは、`Android Application` クラスを拡張し、その `onCreate()` の上書きで `MobileSyncSDKManager.initNative()` をコールします。
- `MainActivity` クラスは、`SalesforceActivity` クラスを拡張します。

これらの2つのクラスによって、ログイン画面とホーム画面を表示するモバイルアプリケーションを作成できます。

Mobile SDK のテンプレートアプリケーションには、200行程度のコードしか含まれていませんが、「HelloWorld」の例以上の機能を実現できます。このメインアクティビティでは、REST 要求を介して Salesforce データを取得し、その結果をモバイルページに表示します。Android オペレーティングシステム、デバイス、セキュリティ上の制約で許可されている範囲でアクティビティの追加や他のコンポーネントのコールなどを行ってこれらのアプリケーションを拡張できます。

MainApplication クラス

すべてのネイティブ Android アプリケーションで `android.app.Application` のインスタンスが必要です。`MainApplication` クラスは、次の基本的なタスクを実行します。

- `Android Application.onCreate()` メソッドを上書きします。
- その `onCreate()` 内で、次を上書きします。
 - スーパークラス `onCreate()` メソッドをコールする。
 - SDK マネージャーオブジェクト (`MobileSyncSDKManager`) で `initNative()` をコールして Salesforce Mobile SDK を初期化する。
 - 元に戻すことができるコメント付きコードを必要に応じて提供し、アプリケーションを Salesforce ID プロバイダーとして使用する。
 - 元に戻すことができるコメント付きコードを必要に応じて提供し、プッシュ通知をサポートする。

次に、クラス全体を示します。

Kotlin

```
package com.bestapps.android

import android.app.Application
```

```

import com.salesforce.androidsdk.mobilesync.app.MobileSyncSDKManager

/**
 * Application class for our application.
 */
class MainApplication : Application() {

    companion object {
        private const val FEATURE_APP_USES_KOTLIN = "KT"
    }

    override fun onCreate() {
        super.onCreate()
        MobileSyncSDKManager.initNative(applicationContext, MainActivity::class.java)
        MobileSyncSDKManager.getInstance().registerUsedAppFeature(FEATURE_APP_USES_KOTLIN)

        /*
         * Uncomment the following line to enable IDP login flow. This will allow the
user to
         * either authenticate using the current app or use a designated IDP app for
login.
         * Replace 'idpAppURIScheme' with the URI scheme of the IDP app meant to be
used.
         */
        // MobileSyncSDKManager.getInstance().idpAppURIScheme = idpAppURIScheme

        /*
         * Un-comment the line below to enable push notifications in this app.
         * Replace 'pnInterface' with your implementation of 'PushNotificationInterface'.
         * Add your Google package ID in 'bootonfig.xml', as the value
         * for the key 'androidPushNotificationClientId'.
         */
        // MobileSyncSDKManager.getInstance().pushNotificationReceiver = pnInterface
    }
}

```

Java

```

package com.salesforce.androidnativetemplate;
import android.app.Application;
import com.salesforce.androidsdk.mobilesync.app.MobileSyncSDKManager;
/**
 * Application class for our application.
 */
public class MainApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        MobileSyncSDKManager.initNative(getApplicationContext(), MainActivity.class);

        /*
         * Uncomment the following line to enable IDP login flow. This will allow the
user to
         * either authenticate using the current app or use a designated IDP app for
login.

```

```

        * Replace 'idpAppURIScheme' with the URI scheme of the IDP app meant to be
used.
        */
// MobileSyncSDKManager.getInstance().setIDPAppURIScheme(idpAppURIScheme);

/*
 * Un-comment the line below to enable push notifications in this app.
 * Replace 'pnInterface' with your implementation of 'PushNotificationInterface'.

 * Add your Google package ID in 'bootonfig.xml', as the value
 * for the key 'androidPushNotificationClientId'.
 */
// MobileSyncSDKManager.getInstance().setPushNotificationReceiver(pnInterface);
    }
}

```

大部分のネイティブ Android アプリケーションは同じようなコードを使用できます。このわずかな作業で、アプリケーションはパスワードおよびログイン/ログアウトメカニズムを自由に実装でき、その他のいくつかの利点も得られます。[SalesforceActivity クラス](#)、[SalesforceListActivity クラス](#)、および [SalesforceExpandableListActivity クラス](#) を参照してください。

MainActivity クラス

Mobile SDK アプリケーションでは、メインアクティビティはユーザーのログイン直後に始まります。メインアクティビティが実行されると、他のアクティビティを起動できるようになり、次にサブアクティビティを起動できるようになります。メインアクティビティを終了することで、アプリケーションが終了します。他のすべてのアクティビティは、メインアクティビティ内で連鎖的に終了します。

テンプレートアプリケーションの MainActivity クラスは、Mobile SDK の抽象アクティビティクラス `com.salesforce.androidsdk.ui.SalesforceActivity` を拡張します。このスーパークラスで、必須のパスワードおよびログインプロトコルを自由に実装できます。代わりに別の基本アクティビティクラスを使用する場合、これらのプロトコルを自分で実装する必要があります。MainActivity は、アプリケーションの UI を初期化してその UI ボタンを実装します。

MainActivity UI には、ユーザーの Salesforce 取引先責任者または取引先を表示できるリストビューも含まれます。ユーザーがこれらのいずれかのボタンをクリックすると、MainActivity オブジェクトが数個の基本クエリを実行して、ビューを作成します。たとえば、Salesforce からユーザーの取引先責任者を取得する場合、`onFetchContactsClick()` メッセージハンドラーが次のような単純な SOQL クエリを送信します。

Kotlin

```

@Throws(UnsupportedEncodingException::class)
@Suppress("UNUSED_PARAMETER")
fun onFetchContactsClick(v: View) {
    sendRequest("SELECT Name FROM Contact")
}

```

Java

```

public void onFetchContactsClick(View v) throws UnsupportedEncodingException {
    sendRequest("SELECT Name FROM Contact");
}

```


内部的には、`private sendRequest()` メソッドが `RestRequest` クラスと指定の SQL 文字列を使用して次のようにサーバー要求を生成します。

Kotlin

```
@Throws (UnsupportedEncodingException::class)
private fun sendRequest(soql: String) {
    val restRequest =
        RestRequest.getRequestForQuery(ApiVersionStrings.getVersionNumber(this), soql)

    client!!.sendAsync(restRequest, object : AsyncRequestCallback {
        override fun onSuccess(request: RestRequest, result: RestResponse) {
            result.consumeQuietly() // consume before going back to main thread
            runOnUiThread {
                try {
                    listAdapter!!.clear()
                    val records = result.asJSONObject().getJSONArray("records")
                    for (i in 0..records.length() - 1) {
                        listAdapter!!.add(records.getJSONObject(i).getString("Name"))
                    }
                } catch (e: Exception) {
                    onError(e)
                }
            }
        }

        override fun onError(exception: Exception) {
            runOnUiThread {
                Toast.makeText(this@MainActivity,
                    this@MainActivity.getString(R.string.sf__generic_error,
                        exception.toString()),
                    Toast.LENGTH_LONG).show()
            }
        }
    })
}
```

Java

```
private void sendRequest(String soql) throws UnsupportedEncodingException
{
    RestRequest restRequest =
        RestRequest.getRequestForQuery(
            getString(R.string.api_version), soql);
    client.sendAsync(restRequest, new AsyncRequestCallback()
    {
        @Override
        public void onSuccess(RestRequest request,
            RestResponse result) {
            // Consume before going back to main thread
            result.consumeQuietly();
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    // Network component doesn't report app layer status.
                }
            })
        }
    })
}
```

```

        // Use the Mobile SDK RestResponse.isSuccess() method to check
        // whether the REST request itself succeeded.
        if (result.isSuccess()) {
            try {
                listAdapter.clear();
                JSONArray records =
                    result.asJSONObject().getJSONArray("records");
                for (int i = 0; i < records.length(); i++) {
                    listAdapter.add(
                        records.getJSONObject(i).getString("Name"));
                }
            } catch (Exception e) {
                onError(e);
            }
        }
    }

    });
}

@Override
public void onError(final Exception exception) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(MainActivity.this,
                MainActivity.getString(R.string.sf__generic_error,
exception.toString()),
                Toast.LENGTH_LONG).show();
        }
    });
}

});
}
}

```

このメソッドは、`com.salesforce.androidsdk.rest.RestClient` クラスのインスタンス `client` を使用して SOQL クエリを処理します。`RestClient` クラスは、`RestRequest` と `RestResponse` の2つのヘルパークラスを使用して、クエリの送信とその結果の処理を行います。`sendRequest()` メソッドは、`RestClient.sendAsync()` をコールして SOQL クエリを非同期に処理します。

`sendRequest()` メソッドは、`sendAsync()` コールに対応するために API バージョンと SOQL クエリ文字列を渡して `com.salesforce.androidsdk.rest.RestRequest` のインスタンスを作成します。生成されたオブジェクトは、`sendAsync()` の第1引数になります。第2引数は、コールバックオブジェクトです。`sendAsync()` はクエリの実行の完了時に、このコールバックオブジェクトに結果を送信します。クエリに成功した場合、コールバックオブジェクトはクエリ結果を使用して UI リストのコントロールを設定します。クエリに失敗した場合、コールバックオブジェクトはトーストポップアップでエラーメッセージを表示します。

デフォルトの API バージョンの上書き

Android の場合、ハードコードされたデフォルトの API バージョンが Mobile SDK コアライブラリで定義されています。この値は、現在の Mobile SDK リリースの日付の Salesforce API バージョンと同じです。`strings.xml` リソースファイルで `api_version` を設定することで、アプリケーションのデフォルトバージョンを上書きできます。

デフォルトバージョンを上書きする場合、APIバージョンの引数が必要な

`RestRequest.getRequestForQuery()` などのメソッドをコールするときに注意してください。

`ApiVersionStrings.getVersionNumber()` を渡したいと思うかもしれませんが、このコールは予期しない値を返す場合があります。次のヒントを参考にしてください。

- 上書きされたバージョンを返すか、上書きが存在しない場合はデフォルト値を返すには、`context` 引数を `Context` の有効なサブクラスに設定します。概要レベルのサブクラスには、`Activity`、`Service`、および `Application` の拡張が含まれます。
- `Context` サブクラスではないクラスから `getVersionNumber()` をコールする場合、`context` 引数として `SalesforceSDKManager.getInstance().getAppContext()` を渡すことができます。
- `context` 引数を `null` に設定した場合、`getVersionNumber()` は常にハードコードされたデフォルト値を返します。

Java での匿名クラスの使用

`RestClient.sendAsync()` へのコールで、コードは新しい `AsyncRequestCallback` オブジェクトを第2引数としてインスタンス化します。ただし、`AsyncRequestCallbackConstructor` の後に、いくつかのメソッド (`onSuccess()` や `onError()`) より優先されるコードブロックが続きます。このコードに不自然な点がある場合、よく状況を確認してください。`AsyncRequestCallback` は、インターフェースとして定義されるため、実装はありません。これをインスタンス化するために、コードで2つの `AsyncRequestCallback` メソッドをインラインで実装して、匿名クラスオブジェクトを作成します。この手法により、別のオブジェクトからコールされない独自の `sendAsync()` 実装を `TemplateApp` に提供でき、一連の特殊なクラス名で API が見づらくなることもありません。

テンプレートマニフェスト

テンプレートプロジェクトの `AndroidManifest.xml` ファイルを見ると、Mobile SDK ネイティブ Android アプリケーションに必要なコンポーネントがわかります。必要なコンポーネントは、「`.MainActivity`」という名前のアクティビティのみです。このコンポーネントは、ログイン後に最初に呼び出されるアクティビティを表しています。この名前のクラスはプロジェクトで定義されます。

`forcedroid` によって作成されたアプリケーションは、Mobile SDK テンプレートプロジェクトに基づいているため、`MainActivity` コンポーネントはすでにそのマニフェストに含まれています。他の Android アプリケーションと同様に、Android Studio でマニフェストを編集して他のコンポーネント (カスタムアクティビティまたはサービスなど) を追加できます。

Android サンプルアプリケーション

Salesforce Mobile SDK には、次のネイティブ Android サンプルアプリケーションがあります。

- **MobileSyncExplorer**: Android 上でのネイティブ Mobile Sync ライブラリの機能を示します。これは、`native/NativeSampleApps/MobileSyncExplorer` の Mobile SDK for Android にあります。
- **RestExplorer**: Mobile SDK の OAuth および REST API 機能を示します。また、REST API アクションをタブレットから調べる場合にも便利です。

Mobile SDK の `hybrid/HybridSampleApps/` には、いくつかのハイブリッドアプリケーション用に Android ラッパーもあります。

- **AccountEditor:** `mobilesync.js` ライブラリを使用してオフラインデータを同期する方法について説明します。
- **MobileSyncExplorerHybrid:** Mobile Sync プラグインを使用してオフラインデータを同期する方法について説明します。
- **NoteSync:** 非 REST API を使用して Salesforce のメモを取得する方法を示します。

第 8 章

HTML5 およびハイブリッド開発

トピック:

- はじめに
- Visualforce を使用した HTML5 コンテンツの配信
- Salesforce データへのアクセス: コントローラーと API
- ハイブリッドアプリケーションのクイックスタート
- ハイブリッドアプリケーションの作成
- モバイルデバイスでのハイブリッドアプリケーションのデバッグ
- iOS 7 ハイブリッドアプリケーションでのステータスバーの制御
- ハイブリッドアプリケーションの JavaScript ファイル
- バージョン設定および JavaScript ライブラリの互換性
- ハイブリッドアプリケーションでのセッション管理
- ログインの延期

HTML5 により、ターゲットデバイスにソフトウェアをインストールせずに、軽量モバイルインターフェースを作成できます。どのモバイルデバイス、タッチデバイス、デスクトップデバイスからでも、このモバイルインターフェースにアクセスできます。HTML5 でカメラや GPS など高度なモバイル機能がサポートされるようになり、これらの一般的なデバイス機能を Salesforce モバイルアプリケーションで簡単に使用できるようになりました。

次の方法で、Salesforce Platform を使用する HTML5 アプリケーションを作成できます。

- Visualforce を使用して、HTML コンテンツを配信する。
- JavaScript Remoting を使用して、Salesforce からレコードを取得するための Apex コントローラーを呼び出す。

また、スタンドアロンの Mobile SDK ハイブリッドアプリケーションで HTML5 コードを再設定し、App Store から配布することもできます。ハイブリッドに変換するには、サードパーティの Cordova コマンドラインを使用して Mobile SDK コンテナプロジェクトを作成してから、HTML5、JavaScript、および CSS ファイルをこのプロジェクトにインポートします。

はじめに

Web 開発者には、Salesforce にアクセスする HTML5 アプリケーションを作成する準備が整えられています。HTML5 アプリケーションはブラウザで実行可能で、Salesforce Mobile SDK は必要ありません。開発者の作業は、Salesforce API をコールして戻り値を取得し、それをロジックと UI につなげるだけです。モバイルブラウザで他のアプリケーションを実行する場合と同じ利点と課題がありますが、Salesforce とそのパートナーがモバイル Web 設計およびコーディングの合理化に役立つツールを提供しています。

HTML5 アプリケーションをハイブリッドコンテナでスタンドアロンとして開発し、Apple App Store または Android マーケットプレイスで配布する場合は、Mobile SDK を使用してハイブリッドアプリケーションを作成する必要があります。

HTML5 および JavaScript の使用

HTML5 コードおよび JavaScript コードを記述するために、Xcode や Microsoft® Visual Studio® などの専門的な開発環境は必要ありません。最新のブラウザには、HTML や JavaScript デバッガーなどの高度な開発機能が含まれています。実際、アプリケーションをテキストエディターで作成し、ブラウザでテストすることができます。ただし、コーディング作業を最小限に抑える上で役立つ、業界の主要なライブラリに関しての十分な知識は必要です。

モバイル開発の近年の成長により、新しい Web 技術ツールキットが急増しました。これらの JavaScript ライブラリの多くはオープンソースで、ライセンスは必要ありません。HTML5 開発用に Salesforce で提供されるほとんどのツールは、このようなサードパーティ技術で構築されています。

HTML5 の開発要件

ブラウザベースの HTML5 Salesforce アプリケーションの作成を計画している場合は、Salesforce Mobile SDK は必要ありません。

- Salesforce 組織が必要です。
- Apex および Visualforce に関する多少の知識が必要です。

 **メモ:** このタイプの開発では、Visualforce を使用します。Database.com は使用できません。

マルチデバイス戦略

モバイルデバイスは世界的に普及しているため、HTML5 モバイルアプリケーションはさまざまなプラットフォームやフォーム要素、デバイス機能に対応する必要があります。Visualforce でデバイスに依存しないモバイルアプリケーションを作成する開発者は、次の重要な設計上の検討事項に直面します。

- アプリケーションでどのデバイスやフォーム要素をサポートするか?
- アプリケーションでデバイスの種類をどのように検出するか?
- 数種のデバイスを適切にサポートする Salesforce アプリケーションをどのように設計するか?

どのデバイスやフォーム要素をサポートするか?

この検討事項に対する答えは、特定の用途やエンドユーザーの要望によって異なります。ただし、具体的にどのデバイス、プラットフォーム、フォーム要素をサポートする必要があるかを時間をかけて検討することは重要です。「あらゆるプラットフォーム、デバイス、フォーム要素をサポート」にはじまり、たとえば「デスクトップと iPhone のみサポート」など、どの程度の範囲までサポートするのかによって、先述の残り 2 つの疑問への答えが大きく変化します。

お分かりのように、この決定を下すには、重要なトレードオフを考慮する必要があります。多くのフォーム要素をサポートすれば、当然アプリケーションの利用者層が増加します。しかし、アプリケーションの初期開発と長期的な管理の両面においては複雑性が増大するという犠牲を伴います。

真のクロスデバイスアプリケーションの開発とは、どのフォーム要素やデバイス(デスクトップ、スマートフォン、タブレット)でも Web ページの表示(およびパフォーマンス)を最適にするという単純な問題ではありません。特定のデバイスやフォーム要素ごとにユーザー操作性を再考してカスタマイズする必要に迫られます。スマートフォンやタブレット向けのアプリケーションでは、デスクトップ向けに最適化された既存の Web ページでサポートされる付加的な機能(ファイルのアップロードや、多数の異なるクリック操作を要する使用事例のサポートなど)が必要なことはほとんどありません。

反対に、スマートフォンやタブレット向けのアプリケーションでは、地理位置情報や写真撮影といった、デスクトップ環境では不可能な機能をサポートできます。LinkedIn や Flipboard のような優れた設計のアプリケーションでは、スマートフォン向けとタブレット向けでも機能が大きく異なります(たとえば、タブレット向けは水平ナビゲーションで、スマートフォン向けは片手操作の垂直スクロール)。アプリケーションでどのデバイスおよびフォーム要素をサポートするかを決定するときは、これらのすべての事項に加え、サポートに関連して必要な時間とコストも考慮します。

どのデバイスをサポートするかを決定したら、特定のユーザーがどのデバイスから Web アプリケーションにアクセスしているかを検出する必要があります。

クライアント側での検出

クライアント側の検出アプローチでは、クライアントブラウザ上で実行されている JavaScript (または CSS メディアクエリ) を使用してデバイスの種類を判別します。具体的には、次の 2 つの方法でデバイスの種類を検出できます。

- ユーザーエージェントヘッダーによるクライアント側のデバイス検出 — このアプローチでは、JavaScript を使用してユーザーエージェント HTTP ヘッダーを解析し、この情報に基づいてデバイスの種類を判別します。もちろん、この判別を行う独自の JavaScript を記述することも可能ですが、既存の JavaScript を再利用することをお勧めします。インターネットを大まかに検索しただけでも、ユーザーエージェントヘッダーに基づいてデバイスの種類を検出できる再利用可能な JavaScript スニペットが多数見つかります。ただし、こうした検索では、このアプローチを使用した場合の危険性も知ることになります。使用される可能性のある全ユーザーエージェントのリストは膨大なうえ現在も増え続けていることから、この方法は概して信頼性が低いものと考えられています。
- 画面サイズやデバイス機能によるクライアント側のデバイス検出 — JavaScript のユーザーエージェント文字列を探るよりも適切な代替法は、デバイスの画面サイズや機能(タッチ対応など)に基づいてデバイスの種類を判別することです。オープンソースの Contact Viewer HTML5 モバイルアプリケーションはこのアプローチの一例で、すべて Visualforce で構築されています。具体的には、MobileAppTemplate.page のページ上部に、デバイスの画面サイズに基づいてスマートフォンとタブレットを判別するシンプルな JavaScript スニペット

が記述されています。もう1つのオプションは、Device.js や Modernizr などのライブラリを使用してデバイスの種類を検出する方法です。これらのライブラリは、CSS メディアクエリと機能検出(タッチ対応など)の一定の組み合わせを使用するため、デバイスの種類を検出する信頼性の高い方法になります。Modernizr ライブラリを使用してデバイスの種類を検出する簡単な例については、

<http://www.html5rocks.com/static/demos/cross-device/feature/index.html> を参照してください。Device.js ライブラリを使用して Visualforce と統合する包括的な例については、GitHub リポジトリ (<https://github.com/sbhanot-sfdc/Visualforce-Device.js>) を参照してください。このリポジトリの DesktopVersion.page から抜粋したスニペットを次に示します。

```
<apex:page docType="html-5.0" sidebar="false" showHeader="false" standardStylesheets="false"
  cache="false" >

<head>
  <!-- Every version of your webapp should include a list of all
  versions. -->
  <link rel="alternate" href="/apex/DesktopVersion" id="desktop"
    media="only screen and (touch-enabled: 0)"/>
  <link rel="alternate" href="/apex/PhoneVersion" id="phone"
    media="only screen and (max-device-width: 640px)"/>
  <link rel="alternate" href="/apex/TabletVersion" id="tablet"
    media="only screen and (min-device-width: 641px)"/>

  <meta name="viewport" content="width=device-width, user-scalable=no"/>
  <script src="{!URLFOR($Resource.Device_js)}"/>
</head>

<body>
  <ul>
    <li><a href="?device=phone">Phone Version</a></li>
    <li><a href="?device=tablet">Tablet Version</a></li>
  </ul>
  <h1> This is the Desktop Version</h1>
</body>
</apex:page>
```

上記のスニペットは、アプリケーションでサポートするデバイスの種類ごとに <link> タグを簡単に追加できることを示しています。Device.js ライブラリが、検出されたデバイスの種類に基づいて自動的にユーザーを適切な Visualforce ページにリダイレクトします。また、上記の「?device=xxx」という形式を使用して、デフォルトの Device.js リダイレクトを上書きする方法もあります。

サーバー側でのデバイス検出

もう1つのオプションは、サーバー上 (Apex コントローラーや拡張クラスなど) でデバイスの種類を検出する方法です。サーバー側でのデバイス検出は、ユーザーエージェント HTTP ヘッダーの解析に基づきます。次に、Visualforce ページが iPhone クライアントから参照されているかどうかを検出できる方法を示す簡単なコードスニペットを示します。

```
<apex:page docType="html-5.0"
  sidebar="false"
  showHeader="false"
  cache="false"
```

```

standardStyleSheets="false"
controller="ServerSideDeviceDetection"
action="{!detectDevice}">
<h1> This is the Desktop Version</h1>
</apex:page>

```

```

public with sharing class ServerSideDeviceDetection {
    public boolean isIPhone {get;set;}
    public ServerSideDeviceDetection() {
        String userAgent =
            System.currentPageReference().
                getHeaders().get('User-Agent');
        isIPhone = userAgent.contains('iPhone');
    }
    public PageReference detectDevice(){
        if (isIPhone)
            return Page.PhoneVersion.setRedirect(true);
        else
            return null;
    }
}

```

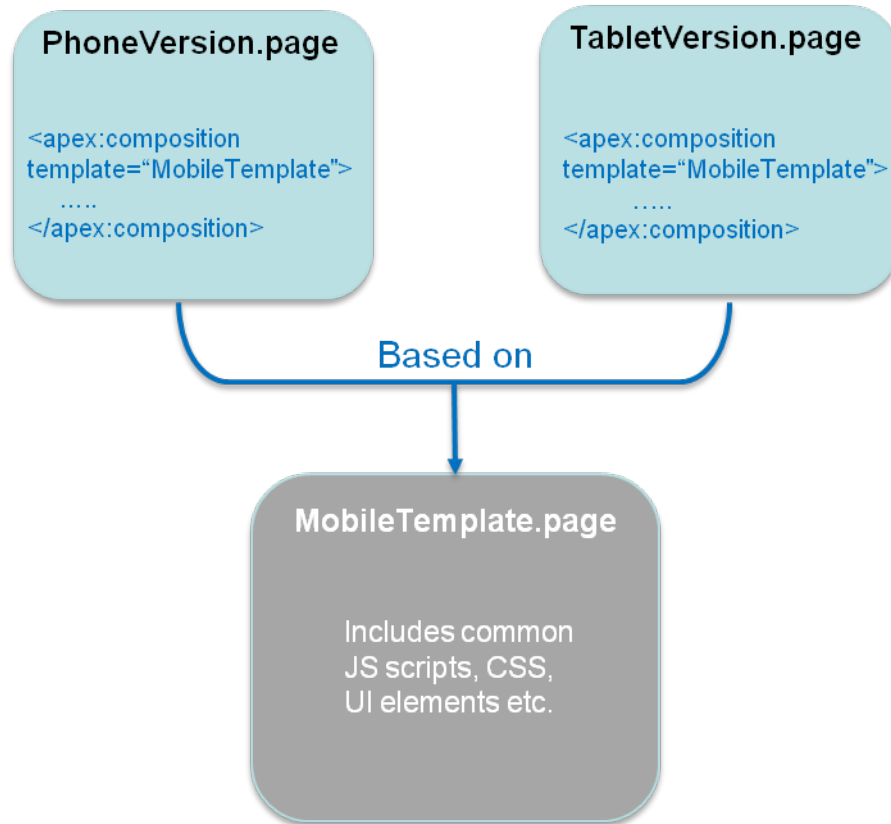
上記のコードスニペットのユーザーエージェント解析は、包括的ではないため、正規表現マッチングに基づいてサポートする必要があるすべてのデバイスを検出するより堅牢な機能を実装する必要があります。手始めに、detectmobilebrowsers.com のコードスニペットに含まれる正規表現を確認することをお勧めします。

数種のデバイスを適切にサポートする Salesforce アプリケーションをどのように設計するか？

サポートする必要のあるデバイスと、そのデバイスの判別方法が決まったら、デバイスやフォーム要素ごとにカスタマイズされたユーザー操作性を提供する最適なアプリケーション設計を考えます。ここでもいくつかのオプションを検討します。

同一の Visualforce ページを各種のフォーム要素で適切に表示させるだけでよいシンプルなアプリケーションの場合は、応答性の高い設計アプローチが効果的なオプションです。簡単に言うと、応答性の高い設計では、CCS3 メディアクエリを使用し、クライアントブラウザのフォーム要素に合わせてページの書式を動的に再設定します。Twitter Bootstrap のような応答性の高い設計フレームワークもこの用途に使用できます。

もう 1 つのオプションは、複数の Visualforce ページを設計し、特定のフォーム要素ごとに各ページを最適化してから、前のセクションで説明したいずれかの手法を使用してユーザーを適切なページにリダイレクトする方法です。個別の Visualforce ページを用意するからといって、コードや機能が重複するわけではなく、また重複すべきでもありません。適切に設計されたソリューションでは、クライアント側(コンポーネントやテンプレートなど Visualforce の戦略を使用する)とサーバー側(たとえば、共通のビジネスロジックを Apex クラスにカプセル化して複数のページコントローラーからコールできるようにする)の両方でコードを最大限に再利用できます。こうした設計の優れた例は、前述のオープンソースの Contact Viewer アプリケーションで確認できます。このアプリケーションはスマートフォン向け(ContactAppMobile.page)とタブレット向け(ContactApp.page)に異なるページを用意していますが、両方が共通のテンプレート(MobileAppTemplate.page)を共有しているため、コードやアイテムが最大限に再利用されています。下の図は、Contact Viewer アプリケーションの設計上の概念を示しています。



また、1つのVisualforceページを複数のフォーム要素に対応させることも可能です。この場合は、サーバー側でデバイスの検出を行い、ほとんどのVisualforceコンポーネント(具体的には、`<div>`タグのCSS「`display:none/block`」プロパティ)で使用可能な「`rendered`」属性を利用して、ページ要素を選択的に表示または非表示にします。ただし、このアプローチはコードが膨大になり、管理が困難になる可能性があるため、慎重に使用する必要があります。

Visualforce を使用した HTML5 コンテンツの配信

デスクトップ環境用にカスタム Web サイトを作成するには、従来より Visualforce を使用しています。ただし、HTML5 と組み合わせると、Visualforce がモバイル Web アプリケーションで実行できる配信メカニズムになります。これらのアプリケーションでは、Sencha のようなサードパーティの UI ウィジェットライブラリや、AngularJS および Backbone.js などのテンプレートフレームワークを活用して、Salesforce 内のデータにバインドできます。

HTML5 Apex ページを設定するには、`docType` 属性を「`html-5.0`」に変更し、次のような他の設定を使用します。

```
<apex:page docType="html-5.0" sidebar="false" showHeader="false" standardStylesheets="false"
  cache="true" >

</apex:page>
```

このコードは、HTML5 コンテンツを含めることができる Apex ページを設定しますが、作成されるのは空のページです。静的リソースとサードパーティのライブラリを使用することで、HTML と JavaScript のコードを追加して、完全に対話形式のモバイルアプリケーションを構築できます。

Salesforce データへのアクセス: コントローラーと API

HTML5 アプリケーションでは、2 通りの方法で Salesforce データにアクセスできます。

- JavaScript Remoting を使用して、Apex コントローラーを呼び出す。
- `force.js` を使用して Salesforce API にアクセスする。

JavaScript Remoting を使用した Apex コントローラーの呼び出し

Apex では、JavaScript から Apex コントローラーメソッドを呼び出す方法として次の 2 つがサポートされています。

- `apex:actionFunction`
- JavaScript Remoting

どちらの手法も AJAX 要求を使用して JavaScript から Apex コントローラーメソッドを直接呼び出します。JavaScript コードは Visualforce ページ上でホストする必要があります。

`apex:actionFunction` に比べて、JavaScript Remoting にはいくつかの利点があります。

- `apex:actionFunction` に比べて柔軟性に富み、パフォーマンスが高い。
- Apex コントローラーメソッドでパラメーターと戻り値の型がサポートされ、Apex と JavaScript 間でデータ型の対応付けが自動的に行われる。
- コールバックで非同期処理モデルを使用する。
- `apex:actionFunction` と異なり、AJAX 要求に Visualforce ページのビューステートが含まれない。この結果、往復速度が速くなります。

ただし、`apex:actionFunction` に比べて、JavaScript Remoting ではより多くのコードを作成する必要があります。

次の例では、Visualforce ページの `<script>` タグに JavaScript コードを挿入しています。このコードは、Visualforce Remoting マネージャーオブジェクトで `invokeAction()` メソッドをコールします。Apex コントローラーオブジェクト `objName` で `getItemId()` という名前の関数をコールするのに必要なメタデータを `invokeAction()` に渡します。`invokeAction()` は非同期で実行されるため、`getItemId()` から返される値を処理するコールバック関数もコードで定義します。Apex コントローラーでは、`@RemoteAction` アノテーションにより、外部の JavaScript コードに `getItemId()` 関数が公開されます。

```
//Visualforce page code
<script type="text/javascript">
    Visualforce.remoting.Manager.invokeAction(
        '{!$RemoteAction.MyController.getItemId}',
        objName,
        function(result, event){
            //process response here
        },
        {escape: true}
    );
</script>

//Apex Controller code
```

```
@RemoteAction
global static String getItemId(String objectName) { ... }
```

@RemoteAction アノテーションについての詳細は、
https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes_annotation_RemoteAction.htm を参照してください。

Force.js を使用した Salesforce API へのアクセス

次のサンプルコードでは、cordova.js および force.js ライブラリを使用して Apex から Salesforce レコードを照会します。これらのリソースを Apex ページに追加する手順は、次のとおりです。

1. cordova.js、force.js、およびプロジェクトに必要な他の静的リソースを含む、ZIP ファイルなどのアーカイブファイルを作成します。
2. Salesforce で、あなたの名前 > [アプリケーションの設定] > [開発] > [静的リソース] からアーカイブファイルをアップロードします。

サンプルコードでは、force.js ライブラリのインスタンスを使用して、Salesforce にログインします。その後、force.query() メソッドをコールして、SOQL クエリを処理します。クエリのコールバック関数では、クエリから返された Name 項目が、「contacts」という ID のオブジェクトに HTML として表示されます。Apex ページの最後では、HTML5 コンテンツにより contacts 要素が単純な タグとして定義されます。

```
<apex:page docType="html-5.0" sidebar="false" showHeader="false"
  contentType="text/html" applyHtmlTag="false" applyBodyTag="false"
  standardStylesheets="false" cache="true">
<html>
  <head>
    <meta charset="utf-8"></meta>
    <meta name="viewport"
      content="initial-scale=1, maximum-scale=1, user-scalable=no"></meta>

    <apex:includeScript value="{!URLFOR($Resource.Easy,
      'cordova/cordova.js')}" />
    <apex:includeScript value="{!URLFOR($Resource.Easy,
      'libs/force.js')}" />

    <script>
(function() {
  /* Do login */
  force.login(
    function() {
      console.log("Auth succeeded");
      showContactsList();
    },
    function(error) {
      console.log("Auth failed: " + error);
    }
  );
});

/* This method will render a list of contacts from current salesforce org */
var showContactsList = function() {
```

```

        fetchRecords(function(data) {
            var contacts = data.records;

            var listItemsHtml = '';
            for (var i=0; i < contacts.length; i++) {
                listItemsHtml += ('<li class="table-view-cell"><
                    div class="media-body">' + contacts[i].Name + '</div></li>');
            }

            document.querySelector('#contacts').innerHTML = listItemsHtml;
        })
    }

    /* This method will fetch a list of contact records from salesforce.
    Just change the soql query to fetch another subject. */
    var fetchRecords = function (successHandler) {
        var soql = 'SELECT Id, Name FROM Contact LIMIT 10';
        force.query(soql, successHandler, function(error) {
            alert('Failed to fetch contacts: ' + error);
        });
    };

    }) ();
</script>
</head>
<body>

    <header>
        <h1>Hello, Visualforce!</h1>
    </header>

    <!-- Placeholder to add Contacts list -->

    <ul id="contacts">
    </ul>

    <p>Welcome to Mobile SDK.</p>
</body>
</html>

</apex:page>

```

メモ:

- Visualforce ページからであっても REST API を使用すると、API コールが使用されます。
- Mobile SDK コンテナまたは Cordova Web ビューから行う Salesforce API コールには、プロキシサービスは必要ありません。Cordova Web ビューでは同じ発生元のポリシーが無効になるため、API コールを直接行うことができます。この免除は、すべての Mobile SDK ハイブリッドアプリケーションとネイティブアプリケーションに適用されます。

追加オプション

HTML5 アプリケーションでは、Mobile Sync を使用できます。必要な JavaScript ライブラリを静的リソースとして含めるだけで十分です。モデルおよび転送機能を利用してください。この場合、オフラインアクセスは無効になります。「[Mobile Sync を使用した Salesforce オブジェクトへのアクセス](#)」を参照してください。

Salesforce Developer Marketing には、開発者向けに HTML5 アプリケーションのクイックスタートに役立つ [モバイルパック](#) が用意されています。

オフラインの制限事項

Salesforce Platform で HTML5 をオフラインで使用する場合は、次の記事を参照してください。

- <https://developer.salesforce.com/blogs/developer-relations/2011/06/using-html5-offline-with-forcecom.html>
- <https://developer.salesforce.com/blogs/developer-relations/2013/03/using-javascript-with-force-com.html>

ハイブリッドアプリケーションのクイックスタート

ハイブリッドアプリケーションは、Salesforce Mobile SDK を活用しながら、容易に操作できる JavaScript および HTML5 を使用して開発できます。

❗ 重要: 2020 年に、WKWebView を支持する App Store は、すべてのアプリケーションの投稿から UIWebView アーキテクチャを削除しました。そのため、iOS 用の Mobile SDK ハイブリッドアプリケーションを実行できるのは Mobile SDK 8.1 以降のみです。「[iOS ハイブリッドアプリケーションからの UIWebView の削除](#)」を参照してください。

ハイブリッドアプリケーション開発の概念を理解している場合は、次の手順を使用してすばやく開発を開始できます。

1. Mobile SDK 11.1 用の Android ハイブリッドアプリケーション開発には、次の前提条件があります。

- Cordova 12.0.1。
- Cordova CLI 12.0.0 以降。
- Java JDK 11.0.11+9 以降 — www.oracle.com/downloads。
- Android Studio の最新バージョン — developer.android.com/sdk。
- Android SDK — 次の範囲の API をすべて含めます。
 - 最小限の API: Android Nougat (API 24)
 - 対象の API: Android 13 (API 33)
- Android SDK ツール
- Android 仮想デバイス (AVD)

2. Mobile SDK 11.1 用の iOS ハイブリッドアプリケーション開発には、次の前提条件があります。

- Cordova 7.0.1。
- Cordova CLI 12.0.0 以降。
- Xcode バージョン: 14 以降。(最新バージョンを推奨)。

- iOS SDK:
 - リリース対象: iOS 15
 - ベース SDK: iOS 16
 - CocoaPods (1.8 から最大値 (指定なし) までの任意のバージョン — cocoapods.org 参照)。
3. Mobile SDK をインストールします。
 - [Android の準備](#)
 - [iOS の準備](#)
 4. まだ接続アプリケーションがない場合は、「[接続アプリケーションの作成](#)」を参照してください。OAuth 範囲として `api`、`web`、および `refresh_token` を選択します。
 **メモ:** コールバック URL を指定する場合、実際のアドレスを使用する必要はありません。
`myapp:///mobilesdk/oauth/done` など、URL に似た任意の値を使用してください。
 5. ハイブリッドアプリケーションを作成します。
 - 「[ハイブリッドアプリケーションを作成する](#)」の手順を実行します。アプリケーション種別として `hybrid_local` を使用します。
 6. 新しいアプリケーションを実行します。
 - [Android でハイブリッドアプリケーションを作成および実行する](#)
 - [iOS でハイブリッドアプリケーションを作成および実行する](#)

ハイブリッドアプリケーションの作成

ハイブリッドアプリケーションは、HTML5 Web アプリケーションの開発の容易さと、ネイティブプラットフォームの機能とを統合したものです。これらは、Salesforce Mobile コンテナ (アプリケーションをデバイス固有のコードに変換するネイティブレイヤー) 内で実行され、その機能は HTML5 と JavaScript ファイルで定義します。これらのアプリケーションは、次の 2 つのカテゴリのいずれかになります。

- **ハイブリッドローカル** — `force.js` ライブラリを使用して開発されたハイブリッドアプリケーションが Mobile コンテナ内に Web アプリケーションをラップします。これらのアプリケーションは、その HTML、JavaScript、および CSS ファイルをデバイス上に保存します。
- **ハイブリッドリモート** — Visualforce テクノロジーを使用して開発されたハイブリッドアプリケーションが Mobile コンテナを介して Apex ページを配信します。これらのアプリケーションは、その HTML、JavaScript、および CSS ファイルの一部または全部を Salesforce サーバーまたはデバイス上 (の `http://localhost`) に保存します。

HTML および JavaScript コードを提供するだけでなく、対象プラットフォームで最小限のコンテナアプリケーションを保守することも必要です。これらのアプリケーションは、必要に応じて設定するネイティブテンプレートと似ています。

他の開発者が使用するためにライブラリまたはサンプルアプリケーションを作成している場合は、GitHub などのバージョン管理されるオンラインリポジトリに公開モジュールを投稿することをお勧めします

(<https://github.com>)。スニペットなどの小規模なサンプル用に、GitHub にはオーバーヘッドの低い共有フォーラムである *gist* が用意されています (<https://gist.github.com>)。

関連トピック:

[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)

ハイブリッド開発について

ブラウザでの JavaScript 開発は簡単に実行できます。コードを変更した後、ブラウザを更新するだけで変更内容を確認できます。ハイブリッドアプリケーションを Mobile SDK コンテナで開発する場合、変更後にリコンパイルとリビルドが必要になります。この理由により、ハイブリッドアプリケーションはブラウザで直接開発し、テストの最終段階でのみコンテナでコードを実行することをお勧めします。

開発者ツールが組み込まれている Google Chrome などのブラウザで開発することをお勧めします。これらのツールでは、実行時に Web アプリケーションのシンボルおよびコードにアクセスできます。

Mobile SDK JavaScript ライブラリでは、従来のコールバック関数を使用するコードを選択できます。

```
traditionalCallbackMethod(args, onSuccess, onFailure)
```

または `promise` を使用します。

```
promiseBasedMethod(args).then(onSuccess).catch(onFailure)
```

コールバック関数を使用して、次のコードを記述できます。

```
self.smartstoreClient.removeSoup(soupName,
    onSuccessRemoveSoup(soupName),
    onErrorRemoveSoup(soupName));
...

function onSuccessRemoveSoup(name) {...}
function onErrorRemoveSoup(name) {...}
```

`promise` を使用すると、非同期コードをインラインに維持できるため、従いやすくなります。`promise` 関数を使用する場合は、コールバックコードを次のように書き換えることができます。

```
self.smartstoreClient.removeSoup(soupName)
    .then(function(soupName) {
        ...
    })
    .catch(function(soupName) {
        ...
    })
```

Cordova を使用したハイブリッドアプリケーションの開発

Salesforce Mobile SDK 11.1 には、プラットフォームごとに特定のバージョンの Apache Cordova (iOS の場合は 7.0.1、Android の場合は 12.0.1) を使用するハイブリッドコンテナが用意されています。アーキテクチャ上、Mobile SDK ハイブリッドアプリケーションは、Cordova プラグインとして Salesforce Mobile SDK を使用する Cordova アプリケー

ションです。Cordova には、アプリケーションのプラグインを更新するための単純なコマンドラインツールが用意されています。Cordova の利点についての詳細は、<https://cordova.apache.org/>を参照してください。

Forcehybrid の使用

Mobile SDK では、ハイブリッドアプリケーションを作成するための `forcehybrid` npm ユーティリティが提供されます。このユーティリティを Cordova コマンドラインと共に使用してハイブリッド Mobile SDK プロジェクトを構築します。`forcehybrid` を使用すると、1つのパスで iOS、Android、またはその両方のハイブリッドプロジェクトを作成できます。

Forcedroid では、2つの方法でアプリケーションを作成できます。

- 目的のアプリケーションの種別と基本設定データを指定する。
または
- 既存の Mobile SDK アプリケーションをテンプレートとして使用する。この場合も基本設定データを指定します。

`forcehybrid` をコマンドラインで対話形式で使用したり、コマンドラインパラメーターと共にスクリプトモードで使用したりできます。使用方法に関する情報を表示するには、引数を指定せずに「`forcehybrid`」と入力します。

対話形式での `forcehybrid create` の使用

コマンドプロンプトで対話形式でアプリケーションオプションを入力するには、「`forcehybrid create`」と入力します。`forcehybrid` ユーティリティで各設定オプションの入力が促されます。次に例を示します。

```
$ forcehybrid create
Enter the target platform(s) separated by commas (ios, android): ios,android
Enter your application type (hybrid_local, hybrid_remote): <press RETURN>
Enter your application name: LocalHybridTest
Enter your package name: com.myhybrid.ios
Enter your organization name (Acme, Inc.): BestApps.com
Enter output directory for your app (leave empty for the current directory): LocalHybridTest
```

この例では、iOS および Android ターゲットを使用して、「LocalHybridTest」というハイブリッドローカルアプリケーションを `./LocalHybridTest/` ディレクトリに作成します。

- ☑ **メモ:** `forcehybrid create` を使用すると、指定したプラットフォームのハイブリッドプロジェクトが設定されますが、Cordova コマンドラインで設定を完了するまでアプリケーションを構築することはできません。「[ハイブリッドアプリケーションを作成する](#)」を参照してください。

スクリプトモードでの `forcehybrid create` の使用

スクリプトモードで、1つのコマンドライン手順内でパラメーターを入力します。

```
$ forcehybrid create --platform=ios,android --apptype=hybrid_local --appname=packagetest
--packagename=com.test.my_new_app --organization="Acme Widgets, Inc."
--outputdir=PackageTest
```

この例では、iOSおよびAndroidターゲットを使用して、「packagetest」というハイブリッドローカルアプリケーションを ./PackageTest/ ディレクトリに作成します。使用可能なオプションの説明は次のとおりです。

```
Usage:

# create ios/android hybrid_local or hybrid_remote mobile application
forcehybrid create
  --platform=Comma separated platforms (ios, android)
  --apptype=Application Type (hybrid_local, hybrid_remote)
  --appname=Application Name
  --packagename=App Package Identifier (e.g. com.mycompany.myapp)
  --organization=Organization Name (Your company's/organization's name)
  [--startpage=App Start Page (The start page of your remote app. Only required for
hybrid_remote)]
  [--outputdir=Output Directory (Leave empty for current directory)]
```

forcehybrid createWithTemplate の使用

forcehybrid createWithTemplate コマンドは、アプリケーション種別ではなく GitHub リポジトリの URI が求められる点を除いて *forcehybrid create* と同じです。このパスを、テンプレートとして使用できる Mobile SDK アプリケーションを含む任意のリポジトリディレクトリを参照するように設定します。テンプレートアプリケーションには、サポートされている任意の Mobile SDK アプリケーション種別を使用できます。force スクリプトは、他のパラメーターに指定された値に一致するようにテンプレートの識別子と設定を変更します。

createWithTemplate を使用する前に、使用可能なテンプレートを把握しておくことで役立ちます。それを調べるには、「*forcehybrid listtemplates*」と入力します。このコマンドにより、Mobile SDK によって提供されるテンプレートのリストが出力されます。各リストには、テンプレートおよびその GitHub URI の簡単な説明が含まれます。次に例を示します。

```
Available templates:

1) Basic hybrid local applicationforcehybrid createwithtemplate
  --templaterepouri=HybridLocalTemplate
2) Basic hybrid remote applicationforcehybrid createwithtemplate
  --templaterepouri=HybridRemoteTemplate
```

テンプレートの URI を見つけたら、*forcehybrid* コマンドラインにそれを挿入できます。*forcehybrid createWithTemplate* のコマンドラインでの使用方法に関する情報は、次のようになります。

```
# create ios/android hybrid_local or hybrid_remote mobile application from a template
forcehybrid createWithTemplate
  --platform=Comma separated platforms (ios, android)
  --templaterepouri=Template repo URI
  --appname=Application Name
  --packagename=App Package Identifier (e.g. com.mycompany.myapp)
  --organization=Organization Name (Your company's/organization's name)
  [--outputdir=Output Directory (Leave empty for current directory)]
```

SalesforceMobileSDK-Templates リポジトリ内のテンプレートでは、*templaterepouri* のパスを削除できます。テンプレート名だけで問題ありません。たとえば、次のコマンドラインコールを考えます。

```
forcehybrid createWithTemplate
--platform=android
```

```
--templaterepouri=HybridLocalTemplate
--appname=MyHybrid
--packagename=com.mycompany.hybridlocal
--organization="Acme Software, Inc."
```

このコールで、ハイブリッドローカルテンプレートアプリケーションを複製します。これは、テンプレートアプリケーションと同じソースコードおよびリソースを使用して、現在のディレクトリでアプリケーションを再作成します。forcehybridによってプロジェクト全体のアプリケーション名が「MyHybrid」に変更されます(このcreatewithtemplate コールは、forcehybridによる hybrid_local アプリケーションの作成と同じです)。

Forcehybrid スクリプトで新しいアプリケーションを生成する方法

forcehybrid スクリプト

- Cordova コマンドラインを使用してアプリケーションを生成します。
- GitHub からテンプレートアプリケーションと bootconfig.json ファイルをダウンロードします。
- GitHub から SalesforceMobileSDKCordova プラグインをダウンロードします。このプラグインは、Mobile SDK ライブラリを Android および iOS ライブラリプロジェクトとして提供します。

ハイブリッドアプリケーションを作成する

forcehybrid と Cordova コマンドラインをインストールしたら、機能的なハイブリッドアプリケーションをいつでも作成できます。

ツールの設定

必要なツールをまだ設定していない場合は、次の手順を実行します。または、必要に応じて、Trailhead プロジェクトの「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」を完了し、学習に応じて Trailhead ポイントを獲得してください。その後、ここに戻って「[ハイブリッド Mobile SDK アプリケーションの作成](#)」に進んで下さい。

 **メモ:** 次のいくつかのステップでキーワード `sudo` が使用されています。「参照・更新」権限がない場合は、Mac OS X でこのキーワードが必要です。これが不要でないことが明らかな場合、または Windows にインストールする場合は、`sudo` コマンドを省略してください。

1. 「[iOS と Android の開発の前提条件](#)」(ページ 25)に記載されている要件を満たしていることを確認します。ハイブリッド開発には少なくとも1つのネイティブ環境が必要です。このネイティブ環境はiOSまたはAndroidのどちらでもかまいません。
2. コマンドプロンプトまたはターミナルウィンドウを開きます。
3. `cordova -v` を実行します。
4. コマンドが見つからないことが `cordova -v` により報告された場合、Cordova コマンドライン (バージョン 12.0.0 以降) をインストールします。

```
sudo npm install -g cordova
```

5. forcehybrid npm パッケージをインストールします。

```
sudo npm install -g forcehybrid
```

forcehybrid の旧バージョンがすでにインストールされている場合、それを必ずアンインストールしてから forcehybrid を再インストールしてください。

ハイブリッド Mobile SDK アプリケーションの作成

1. コマンドプロンプトまたはターミナルウィンドウで、`forcehybrid create` を実行します。
 - プラットフォームには、「ios」、「android」、または「ios,android」と入力します。
 - アプリケーション種別は次のように指定します。
 - `hybrid_local` (モバイルデバイスにコードを保存している Cordova ハイブリッドアプリケーションの場合)。
 - `hybrid_remote` (サーバーの Visualforce アプリケーションを実行する Cordova ハイブリッドアプリケーションの場合)。
 - 独自のアプリケーション名、パッケージ名、および組織の文字列を指定します。
 - (ハイブリッドリモートアプリケーションのみ) 開始ページには、Apex ランディングページの相対 URL (apex/BasicVFPage など) を指定します。
2. HTML、JavaScript、CSS、`bootconfig.json` ファイルをインポートする場合、これらのファイルをプロジェクトの `<outputdir>/www/` ディレクトリに配置します。

 **重要:** `cordova.js`、`cordova.force.js`、Cordova プラグインを含めないでください。
3. プロジェクトディレクトリで、UTF-8 に準拠しているテキストエディターにより `www/bootconfig.json` ファイルを開き、次のプロパティの値を置き換えます。
 - `remoteAccessConsumerKey` — デフォルト値を接続アプリケーションからのコンシューマーキーに置き換えます
 - `oauthRedirectURI` — デフォルト値を接続アプリケーションからのコールバック URL に置き換えます
4. `cd` でアプリケーションのプロジェクトディレクトリに移動します。force スクリプトは、プロジェクトの作成が完了すると画面にディレクトリ名を出力します。たとえば、「Your application project is ready in `<project directory name>`」(アプリケーションプロジェクトが `<プロジェクトディレクトリ名>` に準備できました) と表示されます。
5. 追加の Cordova プラグインごとに、次のように入力します。

```
cordova plugin add <plug-in repo or plug-in name>
```

 **メモ:** <https://plugins.cordova.io> に移動して使用可能なプラグインを検索します。

6. (省略可能 — Mac のみ) 「事後」に、2 番目のプラットフォームを追加するには、次のように入力します。
 - iOS サポートを追加するには、次のように入力します。

```
cordova platform add ios@7.0.1
```

- Android サポートを追加するには、次のように入力します。

```
cordova platform add android@12.0.1
```


7. 次のように入力します。

```
cordova prepare
```

Webアセットを、`www/` ディレクトリの下にあるそれぞれのプラットフォーム固有ディレクトリにリリースします。

❗ 重要: リリース時、`www/` ディレクトリの内容を変更した後は必ず `cordova prepare` を実行して、変更をプラットフォーム固有のプロジェクトフォルダーにリリースします。

Cordova コマンドラインについての詳細は、[Cordova 3.5 のドキュメント](#)で「Command-Line Interface」を参照してください。

Android でハイブリッドアプリケーションを作成および実行する

作成する前に、Android SDK と少なくとも 1 つの Android エミュレーターを含む Android Studio がインストール済みであることを確認してください。Mobile SDK の Android 要件を参照して、正しいバージョンの Android コンポーネントをインストールしてください。

`cordova prepare` を実行したら、プロジェクトを作成して実行します。

アプリケーションを Android Studio で実行する手順は、次のとおりです。

1. Android Studio を起動します。
2. ようこそ画面から [Import Project (Eclipse ADT, Gradle, etc.)] (プロジェクトをインポート (Eclipse ADT, Gradle など)) を選択します。または、Android Studio がすでに実行されている場合は、[File (ファイル)] > [New (新規)] > [Import Project (プロジェクトをインポート)] を選択します。
3. `<your_project_dir>/platforms/android` を選択して、[OK] をクリックします。Gradle ラッパーを使用するように促されたら、その指示を受け入れます。
4. ビルドが終了したら、`android` ターゲットを選択して、メニューまたはツールバーの [Run 'android'] (「android」を実行)] をクリックします。
5. 接続されている Android デバイスまたはエミュレーターを選択します。

❗ 重要: Android Studio で Gradle ラッパーバージョンの更新を提案された場合は、提案を受け入れます。プロセスが完了したら、Android Studio によってプロジェクトが自動的に再インポートされます。

iOS でハイブリッドアプリケーションを作成および実行する

`cordova prepare` を iOS ハイブリッドアプリケーションで実行したら、Xcode でプロジェクトを開き iOS シミュレーターでアプリケーションを実行できます。

アプリケーションを Xcode で実行する手順は、次のとおりです。

1. Xcode で [File (ファイル)] > [Open (開く)] を選択します。
2. 新しいアプリケーションのディレクトリの `platforms/ios/` ディレクトリに移動します。
3. `<app_name>.xcodeworkspace` ファイルをダブルクリックします。
4. 左上隅にある [Run (実行)] ボタンをクリックするか、`COMMAND-R` キーを押します。

ハイブリッドリモートアプリケーションの開発

ハイブリッドローカルアプリケーションの FileExplorer SDK サンプル

(github.com/forcedotcom/SalesforceMobileSDK-Shared/tree/master/samples/fileexplorer) は、簡単にハイブリッドリモートアプリケーションに変換できます。アプリケーションを変換するには、メイン HTML ページをサーバーから配信される Visualforce ページとして再定義します。次に、CSS および JavaScript リソースをアプリケーションと一緒にバンドルして、デバイスに保存されるようにします。

Visualforce ページの作成から開始しましょう。

1. Salesforce Developer Edition 組織の場合、次の属性を持つ Visualforce ページを「FileExplorer」という名前で作成します。

```
<apex:page docType="html-5.0" showHeader="false" sidebar="false">
<!-- Paste content of FileExplorer.html here, but remove the "<!DOCTYPE html>" directive
-->
</apex:page>
```

2. `samples/fileexplorer/FileExplorer.html` ファイルの内容を FileExplorer の Visualforce ページにコピーします。
3. 挿入したコンテンツの上部にある `<!DOCTYPE html>` ディレクティブを削除します。
4. 作業を保存します。

次に、サンプルコードを含めるハイブリッドリモートアプリケーションを作成します。

1. `cd` を実行して、アプリケーションを開発するディレクトリに移動します。唯一の要件として、このディレクトリに「fileexplorer」という名前のサブディレクトリが含まれてない必要があります。
2. ターミナルウィンドウまたはコマンドプロンプトで、次の値を使用して `forcehybrid create` を実行します。

Platform:	<code>ios,android</code>
Application type:	<code>hybrid_remote</code>
Application name:	<code>fileexplorer</code>
Package name:	<code>com.salesforce.fileexplorer</code>
Organization name:	<code>Acme Apps, Inc.</code>
Start page:	<code>apex/FileExplorer</code>
Output directory:	<code><RETURN キーを押す></code>

3. テキストエディターで `fileexplorer/www/bootconfig.json` を開き、次のようにプロパティを変更します。

```
"isLocal": false,
"startPage": "apex/FileExplorer",
```

このように設定することで、アプリケーションがハイブリッドリモートアプリケーションになります。

4. ターミナルウィンドウまたはコマンドプロンプトに戻り、次のコマンドを入力します。

```
cordova prepare
```

完了しました。Android ターゲットを実行するには、

<my_app_directory>/fileexplorer/platforms/android フォルダを Android Studio にインポートして、アプリケーションを実行します。または、iOS ターゲットを実行するには、

<my_app_directory>/fileexplorer/platforms/ios/fileexplorer.xcworkspace ファイルを Xcode にインポートして、アプリケーションを実行します。このサンプルをテストするときは、必ず Visualforce ページを作成した組織にログインしてください。

iOS 用のハイブリッドリモートアプリケーションでの localhost の使用

Mobile SDK バージョン 5.0 以降では、Apple の指示に従って、UIWebView クラスが廃止され、WKWebView が採用されました。

App Store で実施予定のポリシー (UIWebView を使用したアプリケーションの拒否) に準拠するために、Mobile SDK 8.1 では、このクラスへのすべての参照が削除されています。この結果、Mobile SDK ハイブリッドリモートアプリケーションでは localhost を使用できなくなります。

ハイブリッドサンプルアプリケーション

Salesforce Mobile SDK には、JavaScript での Mobile SDK 機能の使用方法を示すハイブリッドサンプルが備えられています。ハイブリッドサンプルは、次の 2 つの方法で提供されます。

- ネイティブラッパーを使用するプラットフォーム固有のアプリケーションとして。これらのラッパーは、ハイブリッドサンプルの一部のサブセットに提供されます。iOS サンプルには、SalesforceMobileSDK-iOS GitHub リポジトリのルートディレクトリにある Mobile SDK ワークスペース (SalesforceMobileSDK.xcodeproj) からアクセスできます。また、Android サンプルには、SalesforceMobileSDK-Android リポジトリの hybrid/SampleApps ディレクトリからアクセスできます。
- HTML5、JavaScript、CSS ソースコードのみが含まれる、プラットフォームに依存しない Web アプリケーションとして。これらのアプリケーションには、すべてのハイブリッドサンプルが含まれていて、プラットフォーム固有のハイブリッドアプリケーションの基盤が提供されます。これらのサンプルアプリケーションは、SalesforceMobileSDK-Shared GitHub リポジトリからダウンロードし、Cordova コマンドラインを使用して作成できます。

Android ハイブリッドサンプルラッパー

- **AccountEditor**: mobilesync.js ライブラリを使用してオフラインデータを同期する方法について説明します。
- **MobileSyncExplorerHybrid**: Mobile Sync プラグインを使用してオフラインデータを同期する方法について説明します。
- **NoteSync**: 非 REST API を使用して Salesforce のメモを取得する方法を示します。

iOS ハイブリッドサンプルラッパー

- **AccountEditor:** `mobilesync.js` ライブラリを使用してオフラインデータを同期する方法について説明します。
- **MobileSyncExplorer:** Mobile Sync プラグインを使用してオフラインデータを同期する方法について説明します。
- **NoteSync:** 非 REST API を使用して Salesforce のメモを取得する方法を示します。

ハイブリッドサンプルアプリケーション (ソースのみ)

Salesforce Mobile SDK では、SalesforceMobileSDK-Shared GitHub リポジトリでプラットフォームに依存しない次のハイブリッドサンプルアプリケーションが提供されます。

- **accounteditor:** Mobile Sync を使用して Salesforce データにアクセスします。
- **contactexplorer:** Cordova を使用してローカルデバイスの取引先責任者を取得します。また、`force.js` ツールキットを使用して、Salesforce REST API で REST トランザクションを実装します。アプリケーションでは、Salesforce SDK の OAuth2 サポートを使用して OAuth ログイン情報を取得し、JavaScript イベントを送信することによって、そのログイン情報を `force.js` に伝搬します。
- **fileexplorer:** Files API を示します。
- **mobilesyncexplorer:** オフライン同期において、Mobile Sync プラグインではなく、`mobilesync.js` を使用してデモを実施します。
- **notesync:** 非 REST API を使用して Salesforce のメモを取得します。
- **simplesyncreact:** Mobile Sync プラグインを使用する React Native アプリケーションのデモを実施します。
- **smartstoreexplorer:** SmartStore API を探索できます。
- **userandgroupsearch:** グループのユーザーを検索できます。
- **userlist:** 組織のユーザーをリストします。これは、最も単純なハイブリッドサンプルアプリケーションです。
- **usersearch:** 組織のユーザーを検索できます。
- **vfconnector:** ネイティブコンテナ内の Visualforce ページをラップします。この例では、`BasicVFTest` という Visualforce ページが組織に存在することを想定しています。アプリケーションではまず、Salesforce SDK OAuth2 サポートを使用して OAuth のログイン情報を取得し、そのログイン情報を使用して、Visualforce ページにアクセスするための適切な Web ビュー Cookie を設定します。

ハイブリッドサンプルアプリケーションを作成する

[SalesforceMobileSDK-Shared](#) リポジトリの `samples` ディレクトリからハイブリッドアプリケーションを作成するには、`forcehybrid` と `Cordova` コマンドラインを使用します。`hybrid_local` または `hybrid_remote` アプリケーションを作成し、共有リポジトリから Web アセット (HTML、JavaScript、CSS ファイル) および `bootconfig.json` ファイルを追加します。

- ☑ **メモ:** ContactExplorer サンプルには `cordova-plugin-contacts` および `cordova-plugin-statusbar` プラグインが必要です。

他のハイブリッドサンプルアプリケーションには、特殊な Cordova プラグインは必要ありません。

サンプルアプリケーションを作成する手順は、次のとおりです。

1. コマンドプロンプトまたはターミナルウィンドウを開きます。
2. 共有リポジトリをコピーします。

```
git clone https://github.com/forcedotcom/SalesforceMobileSDK-Shared
```

3. forcehybrid を使用して、アプリケーションを作成します。
 - プラットフォームには、1つまたは両方のプラットフォーム名(「ios」、「android」、または「ios,android」)を入力します。
 - アプリケーション種別(または apptype パラメーター)には、「hybrid_local」と入力します。
4. 新しいアプリケーションのディレクトリに変更します。

```
cd <app_target_directory>
```

5. 追加の Cordova プラグインごとに、次のように入力します。

```
cordova plugin add <plug-in repo or plug-in name>
```



メモ: <https://plugins.cordova.io> に移動して使用可能なプラグインを検索します。

6. (省略可能—Macのみ)「事後」に、iOS サポートを Android プロジェクトに追加するには、次のように入力します。

```
cordova platform add ios@7.0.1
```

7. (省略可能—Macのみ)「事後」に、Android サポートを iOS プロジェクトに追加するには、次のように入力します。

```
cordova platform add android@12.0.1
```

8. サンプルソースファイルを新しいプロジェクトディレクトリの `www` フォルダーにコピーします。

Mac の場合:

```
cp -RL <local path to SalesforceMobileSDK-Shared>/SampleApps/<template>/* www/
```

Windows の場合:

```
copy <local path to SalesforceMobileSDK-Shared>\SampleApps\<template>\*.* www
```

既存のファイルを上書きするかどうかたずねられたら、上書きに同意します。

9. Cordova の最終準備を実行します。

```
cordova prepare
```



メモ:

- Android Studio では、プラットフォーム名(「android」)で forcehybrid プロジェクトを参照します。たとえば、プロジェクトを実行するには、スタートアッププロジェクトとして「android」を選択し、[Run (実行)]をクリックします。

- Windows の場合、Android Studio では、デフォルトプロジェクトエンコードが `windows-1252` に設定されます。この設定は、`forcehybrid` Gradle ビルドファイルの `UTF-8` エンコードと競合します。最良の結果を得るには、デフォルトプロジェクトエンコードを `UTF-8` に変更します。
- Windows では、必ずシステム管理者として Android Studio を実行します。

ContactExplorer ハイブリッドサンプルの実行

Mobile SDK に含まれている ContactExplorer サンプルアプリケーションを見てみましょう。この練習は、Mac OS または Windows で実行できますが、完全に iOS ターゲットを検証できるのは Mac のみです。

この練習を開始する前に、次のディレクトリがあることを確認してください。

- `SalesforceMobileSDK-Shared` のコピーしたリポジトリを格納するディレクトリ (ルートディレクトリまたは簡単にアクセスできる場所にあるその他のディレクトリ)。
- Mobile SDK ハイブリッドアプリケーションを作成および開発するためのディレクトリ。Cordova プロジェクトには iOS ターゲットと Android ターゲットの両方を含めることができるため、プラットフォームに依存しないディレクトリに配置することをお勧めします。

まず、共有リポジトリをコピーしてから、`forcehybrid` を使用してアプリケーションを作成します。

1. コマンドプロンプトまたはターミナルウィンドウで、`cd` を実行して、共有リポジトリをコピーする予定のディレクトリに移動します。
2. 次のコマンドを実行します。

```
git clone https://github.com/forcedotcom/SalesforceMobileSDK-Shared.git
```

3. `cd` を実行して、ハイブリッドプロジェクトを開発する予定のディレクトリに移動します。
4. 次の値を使用して `forcehybrid create` を実行します。

```
Enter the target platform(s) separated by commas (ios, android): ios,android
Enter your application type (hybrid_local or hybrid_remote, leave empty for hybrid_local):
<press RETURN>
Enter your application name: contactsApp
Enter the package name for your app (com.mycompany.myapp): com.acmeapps.contactexplorer
Enter your organization name (Acme, Inc.): AcmeApps.com
Enter output directory for your app (leave empty for the current directory): <press
RETURN>
```

汎用ハイブリッドプロジェクトができたため、`contactexplorer` というサンプルコードを追加できます。

5. 次のコマンドを実行して、`cp` コマンドのプレースホルダーをローカルパスに置換します。

```
cd contactsApp
cordova plugin add cordova-plugin-contacts
cordova plugin add cordova-plugin-statusbar
cp -RL <local path to SalesforceMobileSDK-Shared>/samples/contactexplorer/* www/
cordova prepare
```

 **メモ:** Windows ユーザー: Windows では、Unix の `cp` コマンドを `copy` コマンドに置き換えます。ただし、`/samples/contactexplorer/` のサブフォルダー `js` と `css` 内のファイルは、他のパスにあるソースファ

イルの別名であることに注意してください。必ず、別名ではなくソースファイル自体をコピーしてください。次に例を示します。

```
cd contactsApp
cordova plugin add cordova-plugin-contacts
cordova plugin add cordova-plugin-statusbar
rem Make a path variable
set SHAREDPATH=C:\SalesforceMobileSDK-Shared\
md www
cd www
md css
copy %SHAREDPATH%\samples\common\jquery.mobile-1.3.1.min.css css
md js
copy %SHAREDPATH%\test\MockCordova.js js
copy %SHAREDPATH%\libs\cordova.force.js js
copy %SHAREDPATH%\libs\force.js js
copy %SHAREDPATH%\dependencies\jquery\jquery.min.js js
copy %SHAREDPATH%\samples\common\jquery.mobile-1.3.1.min.js js
cordova prepare
```

forcedroid スクリプトとその後のコマンドでは iOS プロジェクトと Android プロジェクトを作成します。これらはどちらも ContactExplorer サンプルアプリケーションをラップします。これで、いずれかのプラットフォームでアプリケーションを実行する準備ができました。iOS デバイスを使用する場合、developer.apple.com/library にある『Xcode User Guide (Xcode ユーザーガイド)』の説明に従ってシミュレーターのプロファイルを設定する必要があります。同様に、Android デバイスは developer.android.com/tools の説明に従って設定する必要があります。

アプリケーションを iOS で実行する手順は、次のとおりです。

1. `cd` を実行して `platforms/ios/` に移動します。
2. `open contactsApp.xcworkspace` コマンドを実行します。
3. Xcode で、[実行] をクリックします。

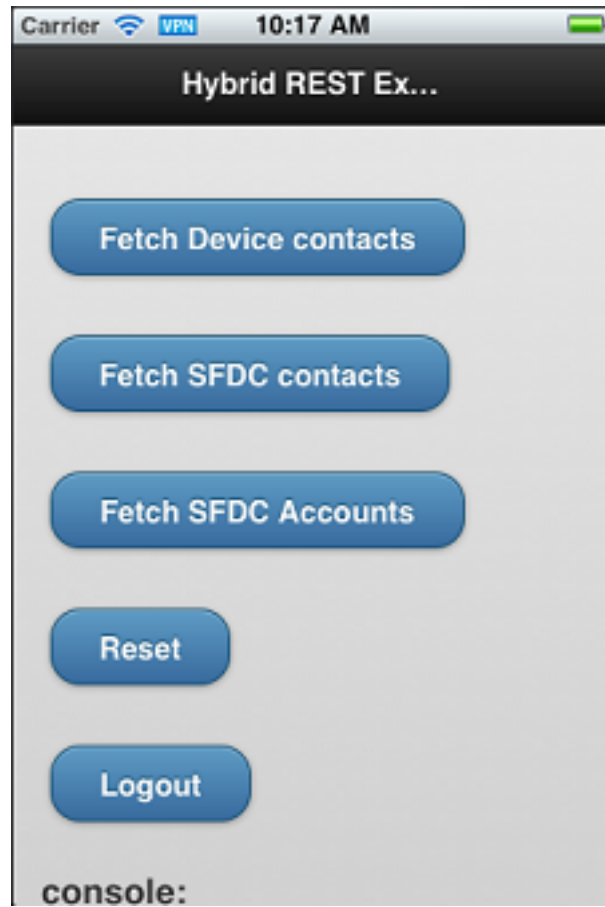
アプリケーションを Android で実行する手順は、次のとおりです。

1. Android Studio で、`<your-hybrid-projects-directory>/contactsApp/platforms/android` プロジェクトをインポートするか、開きます。
2. [実行] をクリックします。

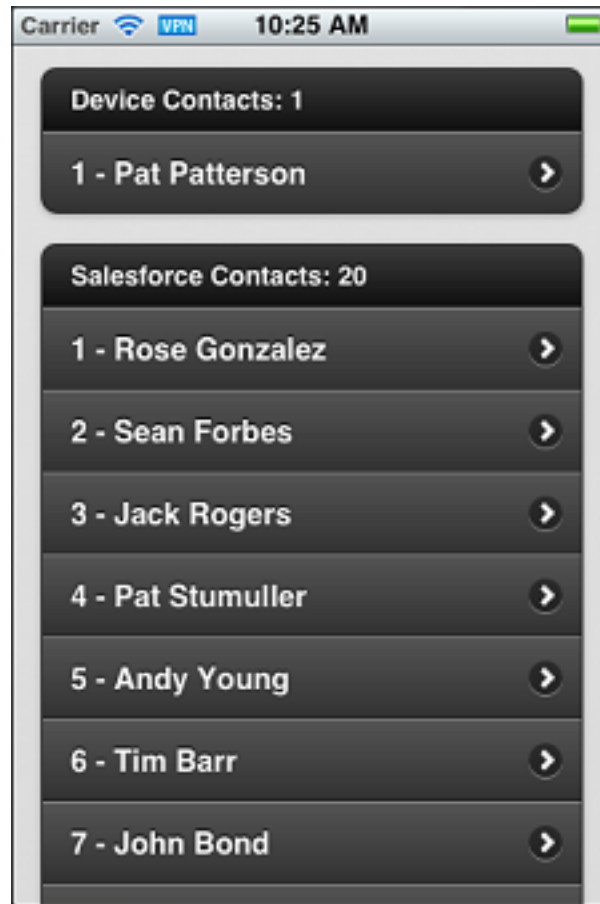
アプリケーションを実行すると、初期スプラッシュ画面が表示された後で Salesforce ログイン画面が表示されます。

The screenshot shows the Salesforce mobile app login interface on an iPhone 6s Plus running iOS 9.3. The status bar at the top displays 'Carrier', signal strength, time '10:23 AM', and battery level. The app's header is blue with 'Log In' text and a settings gear icon. Below the header is the Salesforce logo. The main form area contains two input fields: 'Username' and 'Password'. Below these is a blue 'Log In' button. Under the button is a checkbox labeled 'Remember me'. At the bottom of the form are two links: 'Forgot Your Password?' and 'Use Custom Domain'. The footer of the app shows the copyright notice '© 2016 salesforce.com, inc. All rights reserved.'

Developer Edition 組織のユーザー名とパスワードを使用してログインします。アプリケーションから Salesforce データにアクセスできるようにするには、[許可]をタップします。アプリケーションを実行している状態なので、取引先責任者と取引先のリストを取得できます。DE 組織から Salesforce 取引先責任者名を取得するには [SFDC の取引先責任者を取得] をタップし、取引先名を取得するには [SFDC の取引先を取得] をタップします。



タップするごとに、行が無数のリストに追加されます。完全なリストを表示するには、スクロールダウンします。



アプリケーションの動作を詳しく見てみましょう。

force.js を使用してユーザーセッションを開始するには、`force.login()` をコールします。コンテナ内で実行中のアプリケーションにユーザーがログインした後、アプリケーションがSalesforceリソースにアクセスしようとする、必要に応じてネットワークプラグインがトークンを更新します。ContactExplorerサンプルを基にした次のコードは、一般的な `force.login()` の実装を示します。

デバイスから準備が完了したことが通知されたら、`force.login()` メソッドをコールしてログイン画面をpostします。

```
/* Do login */
force.login(
  function() {
    console.log("Auth succeeded");
    // Call your app's entry point
    // ...
  },
  function(error) {
    console.log("Auth failed: " + error);
  }
);
```

ログインプロセスの完了後、サンプルアプリケーションに `index.html` (`www` フォルダー内)が表示されます。ページの読み込みが完了してモバイルフレームワークの準備ができたなら、`jQuery(document).ready()` 関数

によって `regLinkClickHandlers()` がコールされます。この関数(`inline.js` 内) が、サンプルアプリケーションのさまざまな関数のクリックハンドラーを設定します。たとえば、`#link_fetch_sfmc_contacts` ハンドラーは、`force` オブジェクトを使用してクエリを実行します。

```
$j('#link_fetch_sfmc_contacts').click(function() {
    logToConsole("link_fetch_sfmc_contacts clicked");
    force.query("SELECT Name FROM Contact LIMIT 25",
        onSuccessSfmcContacts, onErrorSfmc);
});
```

`force` オブジェクトは、OAuth2.0 との初期のやりとりの中で設定され、認証されたユーザーのコンテキストで REST API へのアクセス権を付与します。ここで DE 組織の全取引先責任者の名前を取得します。次に、`onSuccessSfmcContacts()` によって、`index.html` ページ上に取引先責任者がリストとして表示されます。

```
$j('#link_fetch_sfmc_accounts').click(function() {
    logToConsole("link_fetch_sfmc_accounts clicked");
    force.query("SELECT Name FROM Account LIMIT 25",
        onSuccessSfmcAccounts, onErrorSfmc);
});
```

`#link_fetch_sfmc_contacts` ハンドラーと同様に、`#link_fetch_sfmc_accounts` ハンドラーは REST API によって取引先レコードを取得します。`#link_reset` と `#link_logout` ハンドラーによって、表示されたリストのクリアとユーザーのログアウトがそれぞれ実行されます。

このアプリケーションでは、デバイスからも取引先責任者を取得できます。これは同等の Web アプリケーションではできない操作です。次のクリックハンドラーは、Cordova の取引責任者プラグインをコールすることにより、デバイスの取引責任者クエリを取得します。

```
$j('#link_fetch_device_contacts').click(function() {
    logToConsole("link_fetch_device_contacts clicked");
    var options = new ContactFindOptions();
    // empty search string returns all contacts
    options.filter = "";
    options.multiple = true;
    options.hasPhoneNumber = true;
    var fields =
        [navigator.contacts.fieldType.displayName,
         navigator.contacts.fieldType.name];
    navigator.contacts.find(fields, onSuccessDevice,
        onErrorDevice, options);
});
```

このハンドラーは、`cordova-plugin-contacts` の `ContactFindOptions` オブジェクトと `navigator.contacts` オブジェクトを使用して、検索を絞り込んで実行します。

`navigator.contacts.find()` をコールして、デバイスから取引先責任者とその電話番号を記載したリストを取得します。`onSuccessDevice()` 関数(ここには示されていません)によって、取引先責任者リストが `index.html` ページに表示されます。

関連トピック:

[iOS でハイブリッドアプリケーションを作成および実行する](#)

[Android でハイブリッドアプリケーションを作成および実行する](#)

モバイルデバイスでのハイブリッドアプリケーションのデバッグ

ハイブリッドアプリケーションは、モバイルデバイスで実行されている間にデバッグできます。デバッグの方法は、開発プラットフォームによって異なります。

実際のデバイスでアプリケーションが実行された場合にのみ発生するバグに遭遇した場合、デスクトップ開発者ツールを使用してそれらの問題をトラブルシューティングします。場合によっては、開発者が2つのランタイムに接続する方法がわからなかったり、その方法が詳細に記載されたドキュメントが見つからなかったりします。次に、マシンの Web アプリケーションデバッガーを、接続デバイスで実行されているアプリケーションに接続するためのプラットフォーム固有の一般的な手順を示します。

Android デバイスのハイブリッドアプリケーションのデバッグ

Android デバイスでハイブリッドアプリケーションをデバッグするには、Google Chrome を使用します。

次の手順は、<https://developer.chrome.com/devtools/docs/remote-debugging> に投稿されている完全な手順をまとめたものです。

1. <https://developer.chrome.com/devtools/docs/remote-debugging> の説明に従って、デバイスの USB デバッグを有効にします。
2. デスクトップ (開発) マシンで Chrome を開き、`chrome://inspect` に移動します。
3. [Discover USB Devices (USB デバイスを見つける)] を選択します。
4. デバイスを選択します。
5. デバイスを使用して、開発マシンで実行されている Web アプリケーションをデバッグする手順は、次のとおりです。
 - a. [Port forwarding... (ポート転送...)] をクリックします。
 - b. デバイスのポートおよび localhost ポートを設定します。
 - c. [Enable port forwarding (ポート転送を有効にする)] を選択します。詳細は、<https://developer.chrome.com/devtools/docs/remote-debugging#port-forwarding> を参照してください。

iOS デバイスで実行されているハイブリッドアプリケーションのデバッグ

iOS デバイスの実機またはシミュレーターでハイブリッドアプリケーションをデバッグするには、デスクトップおよびデバイスで Safari を使用します。

1. デスクトップで Safari を開きます。
2. [Safari] > [環境設定] を選択します。
3. [詳細] タブをクリックします。
4. [メニューバーに“開発”メニューを表示] をクリックします。
5. iOS シミュレーターを使用している場合、次の操作を実行します。

- システムのタスクバーで、CONTROL キーを押しながら、Xcode アイコンをクリックし、[Open Developer Tool (開発者ツールを開く)] > [Simulator (シミュレーター)] を選択します。
 - または、ターミナルウィンドウに「`open -a Simulator.app`」と入力します。
6. [iOS Simulator (iOS シミュレーター)] のメニューで、[File (ファイル)] > [Open Simulator (シミュレーターを開く)] を選択します。
 7. デバイスを選択します。
 8. デバイスまたは iOS シミュレーターのホーム画面から Safari を開きます。
 9. Web アプリケーションの場所に移動します。
 10. デスクトップの Safari で、[開発] > <あなたのデバイス> を選択し、デバイスまたはシミュレーターの Safari で開いた URL を選択します。
[Web Inspector] ウィンドウが開き、デバイスで実行されている Safari インスタンスに接続されます。

iOS 7 ハイブリッドアプリケーションでのステータスバーの制御

iOS 7 では、ステータスバーを表示するか非表示にするかと、ステータスバーを Web ビューにオーバーレイ表示するかどうかを選択できます。Cordova ステータスバープラグインを使用してこれらの設定を指定できます。Salesforce Mobile SDK 2.3 以降では、デフォルトでステータスバーが表示され、Web ビューにオーバーレイ表示されます。

ステータスバーを非表示にするには、次のキーをアプリケーションの plist に追加します。

```
<key>UIStatusBarHidden</key>
<true/>
<key>UIViewControllerBasedStatusBarAppearance</key>
<false/>
```

非表示のステータスバーの例については、AccountEditor サンプルアプリケーションを参照してください。

ステータスバーの外観(オーバーレイ、背景色、透過性など)を制御するには、次のようにorg.apache.cordova.statusbarをアプリケーションに追加します。

```
cordova plugin add org.apache.cordova.statusbar
```

外観の制御は、config.xml ファイルまたは JavaScript から行います。詳細は、

<https://github.com/apache/cordova-plugin-statusbar>を参照してください。Web ビューにオーバーレイ表示しないステータスバーの例については、ContactExplorer サンプルアプリケーションを参照してください。

関連トピック:

[ハイブリッドサンプルアプリケーション](#)

ハイブリッドアプリケーションの JavaScript ファイル

外部連動関係

Mobile SDK では、ハイブリッドアプリケーションのさまざまな機能に次の外部連動関係を使用します。

外部 JavaScript ファイル	説明
jquery.js	一般的な HTML ユーティリティライブラリ
underscore.js	Mobile Sync のサポート
backbone.js	Mobile Sync のサポート

含める JavaScript ファイル

Mobile SDK 2.3 以降では、Cordova ユーティリティがアプリケーションに必要な Cordova プラグインファイルをプロジェクトのプラットフォームディレクトリにコピーします。ユーザーがこれらのファイルを www/ フォルダにコピーする必要はありません。

HTML コードに (`<script>` タグを使用して) 含めるファイルはハイブリッドプロジェクトの種別によって異なります。次に示す各種別について、リストのファイルをすべて含めます。

基本的なハイブリッドアプリケーションの場合:

- cordova.js

基本的なハイブリッドアプリケーションから REST API コールを行う場合:

- cordova.js
- force.js

ハイブリッドアプリケーションで Mobile Sync を使用する場合:

- jquery.js
- underscore.js
- backbone.js
- cordova.js
- force.js
- mobilesinc.js

バージョン設定および JavaScript ライブラリの互換性

ハイブリッドアプリケーションでは、クライアント JavaScript コードは、Cordova (旧 PhoneGap) および SalesforceSDK プラグインを介してネイティブコードとやりとりします。JavaScript コードをモバイルアプリケーションとパッケージ化する場合は、テストによってコードがネイティブコードと動作することを確認します。ただし、アプリケーションが VisualForce で記述されている場合など、JavaScript コードがサーバーから取得される場合は、有

害な競合が発生する可能性があります。このような場合は、使用している Mobile SDK とバージョンが適合する Cordova の JavaScript ライブラリを使用するように十分注意する必要があります。

たとえば、PhoneGap 1.2 を使用する Mobile SDK 1.2 を含むアプリケーションを配信し、後から Mobile SDK 1.3 を使用する更新を配信するとします。バージョン 1.3 の Mobile SDK では、PhoneGap 1.2 ではなく Cordova 1.8.1 が使用されます。そのため、更新されたアプリケーションの JavaScript コードは、ネイティブコンポーネントへのアクセスに、Cordova 1.8.1 および Mobile SDK 1.3 バージョンの JavaScript ライブラリのみを使用する必要があります。バージョンが適合しない JavaScript ライブラリを使用すると、アプリケーションがクラッシュする可能性があります。

クライアントのアップグレードを顧客に強制することはできませんが、ではどのようにしてクラッシュを回避できるでしょうか。まず、クライアントのバージョンを識別します。次に、クライアントが古い場合にはアプリケーションへのアクセスを拒否する（「最新バージョンに更新してください」警告を表示するなど）か、互換性のある JavaScript ライブラリを提供することをお勧めします。

次の表に、Cordova および PhoneGap のバージョンと Mobile SDK バージョンの相関付けを示します。

Mobile SDK のバージョン	Cordova または PhoneGap のバージョン
1.2	PhoneGap 1.2
1.3	Cordova 1.8.1
1.4	Cordova 2.2
1.5	Cordova 2.3
2.0	Cordova 2.3
2.1	Cordova 2.3
2.2	Cordova 2.3
2.3	Cordova 3.5
3.0	Cordova 3.6
3.1	Cordova 3.6
3.2	Cordova 3.6
3.3	Cordova 3.6
4.0	Cordova 5.0.0 (Android)、3.9.2 (iOS)
4.1	Cordova 5.0.0 (Android)、3.9.2 (iOS)
4.2	Cordova 5.0.0 (Android)、3.9.2 (iOS)
4.3	Cordova 5.0.0 (Android)、4.2.0 (iOS)

ユーザーエージェントを使用した Mobile SDK のバージョンの検索

Mobile SDK のバージョンを検出するには、ユーザーエージェント文字列を活用できます。ユーザーエージェントは、SalesforceMobileSDK/<version> で開始します。ユーザーエージェントを取得したら、返された文字列を解析して Mobile SDK のバージョンを検出できます。

ユーザーエージェントは、サーバーで次の Apex コードを使用して取得できます。

```
userAgent = ApexPages.currentPage().getHeaders().get('User-Agent');
```

同様に、クライアントでは navigator オブジェクトを使用して JavaScript で取得できます。

```
userAgent = navigator.userAgent;
```

sdkinfo プラグインを使用した Mobile SDK のバージョンの検出

JavaScript では、sdkinfo プラグインを使用して、Mobile SDK のバージョンとその他の情報を取得することもできます。cordova.force.js ファイルで定義されるこのプラグインには、次のメソッドが備えられています。

```
getInfo(callback)
```

このメソッドは、次の情報を提供する連想配列を返します。

メンバー名	説明
sdkVersion	コンテナの作成に使用される Salesforce Mobile SDK のバージョン (「1.4」など)。
appName	ハイブリッドアプリケーションの名前。
appVersion	ハイブリッドアプリケーションのバージョン。
forcePluginsAvailable	コンテナにインストールされた Salesforce プラグインの名前を含む配列 (「com.salesforce.oauth」、 「com.salesforce.smartstore」など)。

次のコードは、sdkinfo プラグインに保存された情報を取得して、アラートボックスに表示します。

```
var sdkinfo = cordova.require("com.salesforce.plugin.sdkinfo");
sdkinfo.getInfo(new function(info) {
    alert("sdkVersion->" + info.sdkVersion);
    alert("appName->" + info.appName);
    alert("appVersion->" + info.appVersion);
    alert("forcePluginsAvailable->" +
        JSON.stringify(info.forcePluginsAvailable));
});
```

関連トピック:

[例: 適切な JavaScript ライブラリの提供](#)

例: 適切な JavaScript ライブラリの提供

正しいバージョンの JavaScript ライブラリを提供するには、使用する Salesforce Mobile SDK のバージョンごとに別個のバンドルを作成します。次に、必要なバージョンをダウンロードするサーバーに Apex コードを提供します。

1. アプリケーションでサポートする Salesforce Mobile SDK のバージョンごとに、次の操作を実行します。

- a. 目的の SDK バージョンの JavaScript ライブラリを含む ZIP ファイルを作成します。
- b. ZIP ファイルを静的リソースとして組織にアップロードします。

たとえば、Salesforce Mobile SDK バージョン 1.3 を使用するクライアントを配信する場合は、次のファイルを zip ファイルに追加します。

- cordova.force.js
- SalesforceOAuthPlugin.js
- bootconfig.js
- cordova-1.8.1.js (名前を cordova.js に変更する必要があります)

 **メモ:** バンドルでは、Cordova JavaScript ライブラリの名前を cordova.js (PhoneGap-x.x.js ライブラリを使用するバージョンをパッケージ化する場合は PhoneGap.js) に変更できます。

2. 使用するバンドルを判別する Apex コントローラーを作成します。コントローラーコードで、ユーザーエージェント文字列を解析して、クライアントで使用しているバージョンを見つけます。

- a. 組織の [設定] から、[開発] > [Apex クラス] をクリックします。
- b. 次の定義を使用して、SDKLibController という名前の新しい Apex コントローラーを作成します。

```
public class SDKLibController {
    public String getSDKLib() {
        String userAgent =
            ApexPages.currentPage().
                getHeaders().get('User-Agent');

        if (userAgent.contains('SalesforceMobileSDK/1.3')) {
            return 'sdklib13';
        }
        // Add if statements for other SalesforceSDK versions
        // for which you provide library bundles.
    }
}
```

3. バンドル内のライブラリごとに Visualforce ページを作成し、そのページを使用してクライアントを目的のライブラリにリダイレクトします。

たとえば、SalesforceOAuthPlugin ライブラリの場合は、次のようになります。

- a. 組織で、[設定] から、[クイック検索] ボックスに「Visualforce ページ」と入力し、[Visualforce ページ]を選択します。
- b. 次の定義を使用して、「SalesforceOAuthPlugin」という名前の新しいページを作成します。

```
<apex:page controller="SDKLibController"
    action="{!URLFor($Resource[SDKLib],
```

```
'SalesforceOAuthPlugin.js'}}">
</apex:page>
```

- c. HTML コードの `<script>` タグで VisualForce ページを参照します。必ずステップ 3b で作成したページを指し示すようにしてください。次に例を示します。

```
<script type="text/javascript"
src="/apex/SalesforceOAuthPlugin" />
```

 **メモ:** バンドル内のライブラリごとに別個の `<script>` タグを指定します。

ハイブリッドアプリケーションでのセッション管理

モバイルアプリケーションに影響を与えることが多い一般的な問題を解決するために、Mobile SDK ではハイブリッドアプリケーションをネイティブコンテナでラップします。これらのコンテナは、OAuth トークン変換を内部的に管理することによって、シームレスな認証とセッション管理を提供します。ただし、よく使用されるモバイルアプリケーションアーキテクチャの進化に伴い、「One Size Fits All (1つのサイズですべてに対応)」というアプローチは制限が厳しすぎることがわかっています。たとえば、モバイルアプリケーションで Visualforce の JavaScript Remoting を使用していると、セッションが期限切れになった場合に Salesforce Cookie が失われることがあります。この Cookie は、ユーザーが手動で再度ログインした場合にのみ取得されます。

Mobile SDK の最新バージョンでは、リアクティブなセッション管理を使用します。「リアクティブ」は、コンテナにすべての作業を任せるのではなくアプリケーションがセッション管理に参加できることを意味します。ただし、Mobile SDK 1.4 より前に作成されたアプリケーションでは、プロアクティブまたはコンテナによって制御されるセッション管理を使用していました。プロアクティブなシナリオでは、一部のアプリケーション種別がセッション期限切れになったときに再起動するため、ユーザーエクスペリエンスが満足いくものではなくなります。リアクティブなシナリオでは、セッショントークンを更新することでランタイムフローを中断することなくそのような再起動をアプリケーションで回避できます。

アプリケーションをバージョン 1.3 から以降のバージョンにアップグレードする場合、リアクティブ管理に切り替える必要があります。リアクティブ管理に切り替えるには、アプリケーションのアーキテクチャに応じてセッション管理の設定を調整します。次の表には、一般的なアーキテクチャの動作と推奨されるアプローチがまとめられています。

アプリケーションアーキテクチャ	Mobile SDK 5.0 以降のリアクティブな動作	コードのアップグレード手順
REST API	com.salesforce.plugin.network プラグインを使用した JavaScript からの更新	force.js は com.salesforce.plugin.network プラグインを介してネイティブにネットワークコールを処理するため、これを使用するアプリケーションではコーディングが不要になります。他のフレームワークを使用するアプリケーションでは、ネットワークコール用の

アプリケーションアーキテクチャ	Mobile SDK 5.0 以降のリアクティブな動作	コードのアップグレード手順
Visualforce の JavaScript Remoting	JavaScript からのセッションと CSRF トークンの更新	com.salesforce.plugin.network プラグインも導入する必要があります。 セッションタイムアウトイベントをキャッチし、ページの再読み込みまたは新しい iFrame の読み込みを行います。

 **メモ:** Mobile SDK 5.0 以降では、一部のハイブリッドアプリケーションでネットワーク用に使用する JQuery Mobile がネットワークオプションとしてサポートされなくなります。

次のセクションには、サポートされるアーキテクチャのコード例が記載されています。

REST API (Apex2REST を含む)

REST API を使用するハイブリッドアプリケーションでは、各 REST コールの前に期限切れのアクセストークンを更新する必要があります。この要件は、com.salesforce.plugin.network プラグインを介して暗黙的にセッションを更新する force.js を使用するだけで満たすことができます。force.js を使用すれば、アプリケーションで更新コードを追加する必要はありません。

force.js を使用してユーザーセッションを開始するには、force.login() をコールします。コンテナ内で実行中のアプリケーションにユーザーがログインした後、アプリケーションがSalesforceリソースにアクセスしようとする、必要に応じてネットワークプラグインがトークンを更新します。ContactExplorerサンプルを基にした次のコードは、一般的な force.login() の実装を示します。

- デバイスの準備が整ったことが通知されたら、force.login() メソッドをコールしてログイン画面を表示します。

```
/* Do login */
force.login(
  function() {
    console.log("Auth succeeded");
    // Call your app's entry point
    // ...
  },
  function(error) {
    console.log("Auth failed: " + error);
  }
);
```

完全な ContactExplorer サンプルアプリケーションは、
<https://github.com/forcedotcom/SalesforceMobileSDK-Shared/tree/master/samples/contactexplorer> から入手できます。

Visualforce の JavaScript Remoting

JavaScript Remoting を使用して Visualforce ページにアクセスするモバイルアプリケーションでは、セッションの更新コードをメソッドのパラメーターリストに組み込みます。JavaScript で、Visualforce リモートコールを使用して、セッション状態を調べ、適宜調整します。

```
<Controller>.<Method>(
  <params>,
  function(result, event) {
    if (hasSessionExpired(event)) {
      // Reload will try to redirect to login page
      // Container will intercept the redirect and
      // refresh the session before reloading the
      // origin page
      window.location.reload();
    } else {
      // Everything is OK.
      // Go ahead and use the result.
      // ...
    },
    {escape: true}
  );
```

この例では、hasSessionExpired() を次のように定義します。

```
function hasSessionExpired(event) {
  return (event.type == "exception" &&
    event.message.indexOf("Logged in?") != -1);
}
```

高度な使用事例 ページ全体を再読み込みしても、最適なユーザー操作性が提供されない場合があります。ページ全体の再読み込みを回避するには、次の操作が必要です。

1. アクセストークンを更新します。
2. Visualforce ドメインの Cookie を更新します。
3. 最後に、CSRF トークンを更新します。

次のようなページの完全な再読み込みは行いません。

```
window.location.reload();
```

代わりに、次のようにします。

```
// Refresh oauth token
cordova.require("com.salesforce.plugin.oauth").authenticate(
  function(creds) {
    // Reload hidden iframe that points to a blank page to
    // to refresh Visualforce domain cookies
    var iframe = document.getElementById("blankIframeId");
    iframe.src = src;

    // Refresh CSRF cookie
    // Get the provider array
    var providers = Visualforce.remoting.Manager.providers;
    // Get the last provider in the arrays (usually only one)
```

```
var provider = Visualforce.remoting.last;
provider.refresh(function() {
    //Retry call for a seamless user experience
});

},
function(error) {
    console.log("Refresh failed");
}
);
```

ログインの延期

MobileSDKハイブリッドアプリケーションでは、起動時に常にSalesforceログイン画面が表示されます。ただし、これらのアプリケーションは認証を後の時点まで延期することで利点を得られる場合があります。簡単な設定で、アプリケーションの任意の論理的場所までログインを延期できます。

force.js を使用した遅延ログインの実装は2ステップから成るプロセスです。

1. 起動時の認証をスキップするようにプロジェクトを設定します。
2. JavaScript コードで、認証を開始する時点で force.init() 関数(この後に force.login() 関数が続く)をコールします。

ステップ 1: 認証をスキップするようにプロジェクトを設定する

1. プラットフォーム固有のプロジェクトで、www/bootconfig.json ファイルを開きます。
2. shouldAuthenticate プロパティを「false」に設定します。

ステップ 2: JavaScript で認証を開始する

認証プロセスを開始するには、遅延ログインポイントで force.js login() 関数をコールします。通常、force.init() メソッドは、テストや他の本番以外のシナリオでのみ必要になります。

```
/* Do login */
force.login(
    function() {
        console.log("Auth succeeded");
        // Call your app's entry point
        // ...
    },
    function(error) {
        console.log("Auth failed: " + error);
    }
);
```

force.login() 関数は、成功コールバック関数と失敗コールバック関数の2つの引数を取ります。認証に失敗すると、失敗コールバックが呼び出されます。認証に成功すると、force オブジェクトは、その oauth.access_token 設定プロパティにアクセストークンをキャッシュし、成功コールバックを呼び出します。

Salesforce 組織では、開発者やシステム管理者が Lightning Web コンポーネントを使用して高度な Web コントロールを作成し、他の Salesforce コントロールと連携させています。Mobile SDK は、Visualforce ページを Web ビューで実行するアプリケーションを長年サポートしてきました。ただし、最近まで、それらのアプリケーションでは高い技術的なハードルにより LWC の実行が困難でした。Mobile SDK のバージョン 8.2 では、このハードルを解決するソリューションが導入されており、開発者は、カスタムハイブリッドリモートアプリケーションで Lightning Web コンポーネントをホストできるようになりました。

LWC は、新しいテンプレートアプリケーションと新しいアプリケーション種別を中心にサポートされています。LWC テンプレートには、コンポーネントを起動する Salesforce DX サーバーが含まれています。[Salesforce Mobile SDK Templates の GitHub リポジトリ](#)には、HybridLwcTemplate テンプレートアプリケーションがあります。

LWC 対応アプリケーションを作成するには、Mobile SDK forcehybrid ツールを使用して hybrid_lwc アプリケーション種別を指定します。

```
forcehybrid create
Enter the target platform(s) separated by commas (ios, android):
ios,android
Enter your application type (hybrid_local or hybrid_remote or
hybrid_lwc,
    leave empty for hybrid_local): hybrid_lwc
Enter your application name: helloLwc
Enter your package name: com.acme.apps
Enter output directory for your app (leave empty for the current
directory):
    helloLwc
```

生成されたプロジェクトには server フォルダがあり、Salesforce DX プロジェクトが含まれています。ハイブリッド LWC アプリケーションにログインする前に、このプロジェクトを対象組織にリリースします。たとえば、スクラッチ組織にリリースする場合は、次の手順を実行します。

1. プロジェクトの出力ディレクトリを指定している場合は、cd コマンドでそのディレクトリに移動します。

```
cd helloLwc
```

2. スクラッチ組織を作成します。

```
sf org create scratch --definition-file
server/config/project-scratch-def.json --alias MyOrg
```

3. cd コマンドで、プロジェクトの server ディレクトリに移動します。

```
cd server
```


4. プロジェクトのソースファイルをスクラッチ組織にリリースします。

```
sf project deploy start --target-org MyOrg
```

5. スクラッチ組織のパスワードを作成します。

```
sf org generate password --target-org MyOrg
```

6. 新しいスクラッチ組織のユーザー名とパスワードを使用して、仮想または物理デバイスでアプリケーションにログインします。

ハイブリッドLWCアプリケーションには、MobileSDKネイティブコンテナアプリケーションに加えて、次のものが含まれています。

- Visualforce ページのルート Lightning Web コンポーネント。
- Cordova 用の Javascript コードと Mobile SDK の Cordova プラグイン。
- iOS ライブラリまたは Android ライブラリを提供するかどうかを決定する Apex クラス。

ネイティブ Mobile SDK 機能の操作

ハイブリッドLWCアプリケーションは、Mobile SDK Cordova プラグインを介してネイティブコンテナと連携できます。Mobile SDK では、プロジェクト内の Lightning Web コンポーネントにプロパティとして渡すことができる `mobilesdk` オブジェクトが提供されています。たとえば、LWCJavascript ファイルで `mobilesdk` を使用するには、次のアノテーションを追加します。

```
@api mobilesdk;
```

`mobilesdk` オブジェクトにより、Mobile SDK のプラグイン機能をコールすることができます。たとえば、ネットワークコールを実行したり、SmartStoreにデータを保存したり、MobileSyncを使用したりできます。これを実現するには、テンプレートアプリケーションの `ContactsList` コンポーネントで `mobilesdk` を使用して SOQL クエリを実行します。次に、`ContactsList.js` の関連するコードを示します。

```
@api mobilesdk;

connectedCallback() {
    this.loadContacts();
}

loadContacts() {
    let soql = 'SELECT Id, Name, MobilePhone, Department FROM Contact
LIMIT 100';
    this.mobilesdk.force.query(soql,
        (result) => {
            this.contacts = result.records;
        },
        (error) => {
            this.error = error;
        }
    );
}
```

```
    );  
}
```

その他の考慮事項

- すべてではないにしても、ほとんどの開発は `server/force-app/main/default/lwc/` ディレクトリで行います。
- モバイルアプリケーション用の LWC の開発は、デスクトップブラウザ用の開発とほぼ同じです。違いがあるのは、`mobilesdk` オブジェクトを使用してネイティブコンテナにアクセスする場合のみです。
- ルートコンポーネントを変更するには、`server/force-app/main/default/pages/<app-name>.page` を適切なタイミングで更新します。
- ハイブリッド LWC アプリケーションでは、Visualforce ページで Lightning Out を使用して、Lightning の Web コンポーネントを実行します。この方法の既知の制限については、「[Lightning Out の考慮事項と制限](#)」を参照してください。

関連トピック

- [Visualforce ページでのコンポーネントの使用](#) (*Lightning Web Components Developer Guide* (*Lightning Web コンポーネント開発者ガイド*))
- 「[クイックスタート: Lightning Web コンポーネント](#)」 (Trailhead プロジェクト)

第 10 章

React Native 開発

トピック:

- forcereact を使用した React Native プロジェクトの作成
- React Native プロジェクトでの TypeScript の使用
- React Native アプリケーションでの Mobile SDK コンポーネントの使用
- React Native アプリケーション用の Mobile SDK ネイティブモジュール
- React Native を使用した Mobile SDK サンプルアプリケーション
- ログインの延期
- バイナリコンテンツのアップロード

React Native は、JavaScript、スタイルシート、およびマークアップを使用してネイティブ UI 要素に直接アクセスできるサードパーティのフレームワークです。このテクノロジーを特別な Mobile SDK ネイティブモジュールと組み合わせて、ネイティブリソースを使用した迅速な開発を行うことができます。

Mobile SDK は、当初から次の 2 種類のモバイルアプリケーションをサポートしています。

- ネイティブアプリケーション: 最適なユーザー環境とパフォーマンスを提供します。ただし、サポートするモバイルプラットフォームごとに異なる開発テクノロジーを使用する必要があります。
- ハイブリッドアプリケーション: プラットフォーム間で JavaScript とスタイルシートコードを共有できますが、基礎となる汎用 Web ビューでユーザー環境を構成できます。

Mobile SDK 4.0 以降では、3 つ目のオプションとして React Native があります。React Native は、JavaScript 開発のクロスプラットフォームの利点とネイティブアプリケーションのプラットフォーム固有の「デザイン」が 1 つになったアプリケーションです。さらに、開発者環境のスタイルと簡便性はハイブリッド開発に匹敵します。

- レイアウトとスタイル設定には、広く知られている柔軟な Web テクノロジー (JavaScript、スタイルシート、およびマークアップ) を使用します。
- リコンパイルしてコードの更新を確認する必要はありません。ブラウザを更新するだけで変更を確認できます。
- デバッグには、任意のブラウザの開発者ツールを使用できます。
- すべてのビューはネイティブで表示されるため、顧客にネイティブアプリケーションのユーザー環境を提供できます。

Mobile SDK 11.1 は React Native 0.70.14 を使用します。React Native 0.70.14 のソースコードおよびドキュメントは、github.com/facebook/react-native/releases/ の 0.70.14 タグの下にあります。

はじめに

React Native には、いくつかの共通 Mobile SDK コンポーネントと、少なくとも 1 つのネイティブ開発環境 (iOS または Android) が必要です。macOS マシンでは、iOS 版と Android 版の両方のアプリケーションを開発できます。Windows では、Android 版の開発に限定されます。

Mobile SDK 9.0 には、プレーン JavaScript (ES2015) または TypeScript を使用して React Native アプリケーションを開発するオプションがあります。TypeScript では、標準の JavaScript

環境でコンパイル時の静的型チェックやカスタム型を使用できます。詳細は、「[TypeScript Documentation \(TypeScript のドキュメント\)](#)」を参照してください。

使用開始に最適なのは、React Native [Trailhead モジュール](#)です。完了したら戻ってきて再開してください。

React Native 開発環境の設定

Mobile SDK では、forcereact というコマンドラインスクリプトが提供されており、React Native や Mobile SDK のライブラリをインストールしたり、テンプレートアプリケーションからプロジェクトを作成したりできます。このツールをサポートするには、次のパッケージをインストールします。

1. git をインストールします。
 - a. git がすでにインストールされているかどうかを確認するには、オペレーティングシステムのコマンドプロンプトで `git version` を入力し、Return キーを押します。
 - b. 「command not found (コマンドが見つかりません)」というエラーメッセージが表示された場合は、git-scm.com/downloads でオペレーティングシステムに対応する git パッケージをダウンロードしてインストールしてください。
2. Node.js と npm をインストールします。
 - a. これらのツールがすでにインストールされているかどうかを確認するには、コマンドプロンプトで `npm -v` と入力し、Return キーを押します。
 - b. 「command not found (コマンドが見つかりません)」というエラーメッセージが表示された場合は、<https://nodejs.org/en/> でオペレーティングシステムに対応する Node.js パッケージをダウンロードしてインストールしてください。

3. yarn をインストールします。

```
[sudo] npm install -g yarn
```

詳細は、<https://yarnpkg.com/> で、[Getting Started (はじめに)] > [Installation (インストール)] を選択して参照してください。

4. TypeScript コンパイラーをインストールします。

```
[sudo] npm install -g typescript
```

詳細は、<https://www.typescriptlang.org> の「Download (ダウンロード)」を参照してください。



警告: `npm install -g tsc` は使用しないでください。この方法では、互換性のないバージョンの TypeScript がインストールされるため、forcereact でエラーが発生する原因となります。

5. forcereact をインストールします。

```
[sudo] yarn global add forcereact
```

 **重要:** Google Play の要件を満たすため、Mobile SDK 7.2 は、その React Native コンポーネントを、64ビットのビルドをサポートするバージョンにアップグレードします。2019年8月1日以降、Google Play ストアで公開される新規および更新版の Mobile SDK React Native アプリケーションは、Mobile SDK 7.2 以降を使用してビルドする必要があります。詳細は、
<https://android-developers.googleblog.com/2019/01/get-your-apps-ready-for-64-bit.html?m=1> を参照してください。

forcereact を使用した React Native プロジェクトの作成

ネイティブの Mobile SDK 環境を正常にインストールしたら、React Native アプリケーションの作成を開始できます。

アプリケーションを作成するには、ターミナルウィンドウまたは Windows コマンドプロンプトで forcereact を使
用します。forcereact ユーティリティでは、2つの方法でアプリケーションを作成できます。

- 目的のアプリケーションの種別と基本設定データを指定する。
または
- 既存の Mobile SDK アプリケーションをテンプレートとして使用する。この場合も基本設定データを指定しま
す。

Mobile SDK 9.0 では、forcereact に TypeScript の `react_native_typescript` をサポートするアプリケーション種
別オプションが追加されました。この種別は、forcereact create コマンドのデフォルトになります。代わ
りに標準の JavaScript を使用するには、`react_native` アプリケーション種別を指定します。

コマンドラインプロンプトの対話型モードまたはパラメーター化されたコマンドラインバージョンのスクリプ
トモードで forcereact を使用できます。

対話形式での forcereact の使用

コマンドプロンプトから対話形式でアプリケーションオプションを入力するには、「*forcereact create*」
と入力します。forcereact ユーティリティにより、各設定オプションの入力が促されます。

コマンドラインオプションでの forcereact の使用

必要に応じて、コマンドラインから直接 forcereact パラメーターを指定できます。使用方法に関する情報を表
示するには、引数を指定せずに「*forcereact*」と入力します。次のように使用可能なオプションのリストが表
示されます。

```
$ forcereact

forcereact: Tool for building a React Native mobile application
              using Salesforce Mobile SDK

Usage:

# create a React Native mobile application
forcereact create
  --platform=comma-separated list of platforms (ios, android)
  [--apptype=application type (react_native_typescript or react_native,
    leave empty for react_native_typescript)]
  --appname=application name
  --packagename=app package identifier (e.g. com.mycompany.myapp)
  --organization=organization name (your company's/organization's name)
  [--outputdir=output directory (leave empty for current directory)]
```

この情報を使用して、「*forcereact create*」の後にオプションと値を続けて入力します。たとえば、TypeScript をサポートする React Native アプリケーションを作成する場合、次のようになります。

```
$ forcereact create
--platform=ios,android
--appname=CoolReact
--packagename=com.test.my_new_app
--organization="Acme Widgets, Inc."
--outputdir=CoolReact
```

テンプレートの指定

forcereact createWithTemplate は、テンプレートリポジトリの URI も求められる点を除いて *forcereact create* と同じです。このパスを、テンプレートとして使用できる Mobile SDK アプリケーションを含む任意のリポジトリディレクトリを参照するように設定します。テンプレートアプリケーションには、サポートされている任意の Mobile SDK アプリケーション種別、すなわち *react_native* または *react_native_typescript* を使用できます。*force* スクリプトは、他のパラメーターに指定された値に一致するようにテンプレートの識別子と設定を変更します。

createWithTemplate を使用する前に、使用可能なテンプレートを把握しておく役立ちます。それを調べるには、「*forcereact listtemplates*」と入力します。このコマンドにより、Mobile SDK によって提供されるテンプレートのリストが出力されます。各リストには、テンプレートおよびその GitHub URI の簡単な説明が含まれます。次に例を示します。

Available templates:

- 1) Basic React Native application:
`forcereact createwithtemplate --templaterepouri=ReactNativeTemplate`
- 2) Basic React Native application that uses deferred login
`forcereact createwithtemplate --templaterepouri=ReactNativeDeferredTemplate`
- 3) Basic React Native application written in TypeScript
`forcereact createwithtemplate --templaterepouri=ReactNativeTypeScriptTemplate`
- 4) Sample application using MobileSync data framework
`forcereact createwithtemplate --templaterepouri=MobileSyncExplorerReactNative`

テンプレートの URI を見つけたら、それを *forcereact* コマンドラインに入力できます。次に、TypeScript プロジェクトを *forcereact createWithTemplate* で作成する場合のコマンドラインの使用法に関する情報を示します。

```
forcereact createWithTemplate
--platform=Comma separated platforms (ios, android)
--templaterepouri=Template repo URI
--appname=Application Name
--packagename=App Package Identifier (e.g. com.mycompany.myapp)
--organization=Organization Name (Your company's/organization's name)
[--outputdir=Output Directory (Leave empty for current directory)]
```

SalesforceMobileSDK-Templates リポジトリ内のテンプレートでは、*templaterepouri* のパスを削除できます。テンプレート名だけで問題ありません。たとえば、次のコマンドラインコールを考えます。

```
forcereact createWithTemplate
--platform=ios,android
```



```
--templaterepour=MobileSyncExplorerReactNative
--appname=MyReact
--packagename=com.mycompany.react
--organization="Acme Software, Inc."
--outputdir=""
```

このコールでは、現在のディレクトリに iOS と Android の両方をサポートする React Native アプリケーションが作成されます。アプリケーションが使用するソースコードとリソースは、MobileSyncExplorerReactNative サンプルアプリケーションと同じです。Forcereact は、プロジェクト全体でアプリケーション名を「MyReact」に変更します。

Android Studio によるアプリケーションの作成と実行

Mobile SDK では、React Native について、パッケージ管理用に `npm` と `yarn` の両方がサポートされています。両方のオプションを知っている場合は、任意のオプションを選択できます。

1. ターミナルウィンドウまたは Windows コマンドプロンプトで、プロジェクトのルートディレクトリに移動します。

```
cd <my_project_root>
```

2. 次のコマンドを実行します。

- `npm start`
または
- `yarn start`

3. Android Studio でプロジェクトを開きます。

- ようこそ画面から [Import Project (Eclipse ADT, Gradle, etc.) (プロジェクトをインポート (Eclipse ADT、Gradle など))] をクリックします。
または
- [File (ファイル)] メニューから、[File (ファイル)] > [New (新規)] > [Import Project... (プロジェクトをインポート...)] をクリックします。

4. `<project_root>/android/` ディレクトリを参照し、[OK] をクリックします。

Android Studio によってワークスペースが自動的に作成されます。このプロセスに数分かかることがあります。ステータスバーに「Gradle build successful (Gradle のビルドに成功しました)」と表示されていれば、プロジェクトを実行できます。

5. [Run 'app' (<アプリケーション名> を実行)] をクリックするか、`Shift+F10` を押します。

Android Studio によってエミュレーターまたは接続されている Android デバイスにアプリケーションが起動します。

Xcode でのアプリケーションの作成と実行

1. ターミナルウィンドウで、プロジェクトのルートディレクトリに移動します。

```
cd <my_project_root>
```

2. 「`npm start`」と入力し、Enter キーを押します。
3. Xcode で、`<project_root>/ios/<project_name>.xcworkspace` を開きます。
4. [実行] をクリックします。

Xcode がアプリケーションをシミュレーターまたは接続された iOS デバイスで起動します。

forcereact スクリプトで新しいアプリケーションを生成する方法

- スクリプトは、実行時に GitHub リポジトリからテンプレートをダウンロードします。forcereact create コマンドの場合、スクリプトは [SalesforceMobileSDK-Templates](#) GitHub リポジトリのデフォルトのテンプレートを使用します。
- スクリプトは npm を使用して React Native の連動関係をダウンロードします。
- スクリプトは git (Android) または CocoaPods (iOS) を使用して Mobile SDK ライブラリをダウンロードします。

React Native プロジェクトでの TypeScript の使用

TypeScript は、React Native アプリケーションにとって、いくつかの利点があります。安全にコードを記述できるだけでなく、通常の JavaScript とシームレスに混在使用できます。TypeScript は状況に応じて使用でき、どの程度使用するかはまったくの任意です。

Mobile SDK では、TypeScript の使い方を示す、新しいテンプレートアプリケーションが用意されています。新しいテンプレートは、[SalesforceMobileSDK-Templates](#) GitHub リポジトリの [ReactNativeTypeScriptTemplate](#) ディレクトリで表示またはダウンロードできます。

TypeScript アプリケーションのソースコードは、`*.ts`、`*.tsx`、または `*.js` ファイルで保存できます。たとえば、新しいテンプレートアプリケーションで、`app.js` は `app.tsx` になります。古いファイルと新しいファイルを比較してみると、TypeScript に違いがほとんどないことがわかります。

- インポートは同じです。
- 新しい型についてのコード変更は最小限に抑えられています。

`app.tsx` では、元の JavaScript コードに次のカスタムデータ型が追加されています。

```
interface Response {
  records: Record[]
}

interface Record {
  Id: String,
  Name: String
}

interface Props {
```

```

}

interface State {
  data : Record[]
}

```

`Record` 型は、`Response` インターフェイス内で使用されています。新しいテンプレートでは、元のスクリプトのパラメーターに他の3つのデータ型が適用されています。たとえば、`constructor` には `Props` と `State` が適用されています。

```

class ContactListScreen extends React.Component<Props, State> {
  constructor(props:Props) {
    super(props);
    this.state = {data: []};
  }
}

```

`Response` 型は、SOQL クエリから受信した応答を保護します。

```

(response:Response) => that.setState({data: response.records}),
(error) => console.log('Failed to query:' + error)

```

Salesforce から `Response` 型の定義と競合するデータを `response` に渡すと、コンパイル時エラーが発生します。ただし、`Response` インターフェイスは、定義していないオブジェクトを不許可にしたり検査したりすることはありません。このため、`Response` には `Id` と `Name` の2つの項目しか指定されていませんが、`response.records` にアクセスする場合、マスターを通常の JavaScript として渡します。

React Native アプリケーションでの Mobile SDK コンポーネントの使用

React Native アプリケーションは、Mobile SDK ネイティブアプリケーションと同じ Mobile SDK ライブラリにアクセスします。React Native に対し、Mobile SDK は、ネイティブの Mobile SDK 命令として JavaScript コードを実行する JavaScript コンポーネント (ブリッジ) を提供します。

React Native では、次のネイティブブリッジを介して Mobile SDK 機能にアクセスします。

- `react.force.oauth.js`
- `react.force.net.js`
- `react.force.smartstore.js`
- `react.force.mobilesync.js`

これらのブリッジを使用するには、JavaScript コードに `import` ステートメントを追加します。次の例では、4つのブリッジすべてをインポートします。

```

import {oauth, net, smartstore, mobilesync} from 'react-native-force';

```

forcereact で作成された React Native アプリケーションでは、`package.json` ファイルで `react-native-force` ソースパスを指定します。

```

"react-native-force": "https://github.com/forcedotcom/SalesforceMobileSDK-ReactNative.git"

```

 **メモ:** React Native で `force.js` ライブラリを使用することはできません。

React Native アプリケーション用の Mobile SDK ネイティブモジュール

Mobile SDK には、ネイティブ Mobile SDK 機能への JavaScript ブリッジとして機能する React Native 用のネイティブモジュールが備えられています。

OAuth

OAuth ブリッジは、Cordova の OAuth プラグインと似ています。

使用方法

```
import {oauth} from 'react-native-force';
```

型とメソッド

SalesforceMobileSDK-ReactNative GitHub リポジトリの [react.force.oauth.ts](#) を参照してください。

Network

Network ブリッジは、ハイブリッドアプリケーションの force.js ライブラリと似ています。

使用方法

```
import {net} from 'react-native-force';
```

型とメソッド

SalesforceMobileSDK-ReactNative GitHub リポジトリの [react.force.net.ts](#) を参照してください。

SmartStore

SmartStore ブリッジは、Cordova の SmartStore プラグインと似ています。ただし、プラグインとは異なり、React Native では最初の引数は省略できません。

使用方法

```
import {smartstore} from 'react-native-force';
```

型とメソッド

SalesforceMobileSDK-ReactNative GitHub リポジトリの [react.force.smartstore.ts](#) を参照してください。

Mobile Sync

Mobile Sync ブリッジは、Cordova の Mobile Sync プラグインと似ています。ただし、プラグインとは異なり、React Native では最初の引数は省略できません。

使用方法

```
import {mobilesync} from 'react-native-force';
```

型とメソッド

SalesforceMobileSDK-ReactNative GitHub リポジトリの react.force.mobilesync.ts を参照してください。

 **メモ:** 「上位同期」操作のための項目リストの処理が Mobile SDK 5.1 で変更されました。JavaScript `syncUp()` メソッドについての説明は、「[Mobile Sync プラグインのメソッド](#)」を参照してください。

React Native を使用した Mobile SDK サンプルアプリケーション

Mobile SDK の React Native を把握する最適な方法は、サンプルコードで学習することです。

Mobile SDK には、MobileSyncExplorer アプリケーションの 4 つの実装があり、これには React Native バージョンも含まれています。MobileSyncExplorerReactNative を使用するには、[MobileSyncExplorerReactNative README.md](#) ファイルの手順に従ってください。

MobileSyncExplorerReactNative のファイルについての説明を次に示します。

表 1: 主要なフォルダーとファイル

パス	説明
README.md	開始時の説明。
installandroid.js	このスクリプトは、Android サンプルをインストールするために使用します。詳細は、README.md を参照してください。
installios.js	このスクリプトは、iOS サンプルをインストールするために使用します。詳細は、README.md を参照してください。
ios	iOS アプリケーション。
android	Android アプリケーション。
js	アプリケーションの JavaScript ソースファイル。
index.js	アプリケーションの開始ページ

表 2: React コンポーネント

ファイル	コンポーネント	説明
js/events.js		イベントモデル
js/App.js	MobileSyncExplorerReactNative	ルートコンポーネント (アプリケーション全体) (iOS と Android)。
js/SearchScreen.js	SearchScreen	検索画面 (iOS と Android)。
jsContactScreen.js	ContactScreen	1 つの取引先責任者を表示および編集するために使用します (iOS と Android)。

ファイル	コンポーネント	説明
js/ContactCell.js	ContactCell	検索画面内の結果リストの 1 行 (iOS と Android)。
js/ContactBadge.js	ContactBadge	検索結果画面で使用されるイニシャルが含まれた色付きの円 (iOS と Android)。
js/Field.js	Field	取引先責任者画面で使用される項目の名前と値 (iOS と Android)。
js/StoreMgr.js	StoreMgr	SmartStore とサーバーを操作します (Mobile Sync 経由)。
js/NavImgButton.js	NavImgButton	ナビゲーションバーボタン

表 3: プラットフォーム固有のネイティブプロジェクト

ファイル	説明
android/	Android ネイティブプロジェクト
ios/	iOS ネイティブプロジェクト

 **メモ:** ほとんどのコンポーネントは、iOS と Android で共有されます。ただし、いくつかのコンポーネントはプラットフォーム固有です。

ログインの延期

Mobile SDK の初期バージョンの React Native では、起動時に常に Salesforce ログイン画面が表示されます。ただし、これらのアプリケーションは認証を後の時点まで延期することで利点を得られる場合があります。Mobile SDK 4.2 の React Native 以降、アプリケーションの任意の論理的場所までログインを延期できます。

遅延ログインの実装は 2 ステップからなるプロセスです。

1. iOS または Android ネイティブコンテナアプリケーションでは、Mobile SDK ネイティブメソッドをコールして起動時の認証を無効にします。
2. React コードで、認証を開始する時点で Mobile SDK JavaScript 関数をコールします。

Mobile SDK では、遅延ログインを示すことに特化した React Native テンプレートアプリケーションの [ReactNativeDeferredTemplate](#) を提供しています。このテンプレートを元にアプリケーションを作成するには、次のように `forcereact createwithtemplate` コマンドを使用します。

```
$ forcereact createwithtemplate
Enter the target platform(s) separated by commas (ios, android): ios,android
Enter URI of repo containing template application or a Mobile SDK template name:
ReactNativeDeferredTemplate
```

```
Enter your application name: MyDeferred
...
```

実装の詳細は、この後で説明します。

ステップ 1: 起動時のログインを無効にする

iOS (Objective-C):

起動時に Salesforce ログイン画面が表示されないようにするには、次の手順を実行してください。

1. プロジェクトの `bootconfig.plist` ファイルで、`shouldAuthenticate` を `false` に設定します。

```
<dict>
  <key>remoteAccessConsumerKey</key>
  <string>3REW9Iu66....</string>
  <key>oauthRedirectURI</key>
  <string>testsfdc:///mobilesdk/detect/oauth/done</string>
  <key>oauthScopes</key>
  <array>
    <string>web</string>
    <string>api</string>
  </array>
  <key>shouldAuthenticate</key>
  <false/>
</dict>
```

Android (Java):

デフォルトでは、起動時に Salesforce ログイン画面が表示されます。この動作を無効にするには、`MainActivity` クラス(または `SalesforceReactActivity` をサブクラス化するクラス)の `shouldAuthenticate()` メソッドを次のように変更します。

```
@Override
public boolean shouldAuthenticate() {
    return false;
}
```

ステップ 2: React (JavaScript) で認証を開始する

認証プロセスを開始するには、次の `oauth` 関数をコールします。

```
function authenticate(success, fail)
```

この関数は、成功コールバック関数と失敗コールバック関数の2つの引数を取ります。認証に失敗すると、失敗コールバックが呼び出されます。認証に成功すると、次のキーを含む辞書と共に成功コールバックが呼び出されます。

- `accessToken`
- `refreshToken`
- `clientId`
- `userId`

- orgId
- loginUrl
- instanceUrl
- userAgent
- communityId
- communityUrl

バイナリコンテンツのアップロード

バイナリアップロード機能をサポートする任意の `force.com` エンドポイントにバイナリコンテンツをアップロードできます。

`react.force.net.js` の `sendRequest()` メソッドには、`fileParams` という新しい省略可能なパラメータがあります。

```
function sendRequest(endPoint, path, successCB, errorCallback, method, payload, headerParams, fileParams)
```

このパラメーターの形式は次のようになります。

```
{
  <fileParamNameInPost>: // value depends on the endpoint
  {
    fileMimeType:<someMimeType>,
    fileUrl:<fileUrl>, // url to file to upload
    fileName:<fileNameForPost>
  }
}
```

次に例を示します。

```
{
  fileUpload:
  {
    fileMimeType:'image/jpeg',
    fileUrl:localPhotoUrl,
    fileName:'pic.jpg'
  }
}
```



例: github.com/wmathurin/MyUserPicReactNative サンプルアプリケーションはバイナリアップロードを示しています。このサンプルでは、プロフィール写真を変更できます。新しい写真のバイナリアップロードは、`UserPic.js` ファイルの `uploadPhoto()` 関数で行われます。

このサンプルの `getUserInfo()` 関数での `sendRequest()` コールを次に示します。

```
getUserInfo(callback) {
  forceClient.sendRequest('/services/data',
    '/v36.0/connect/user-profiles/' + this.state.userId + '/photo',
    (response) => {
      callback(response);
    },
  ),
```

```
(error) => {
  console.log('Failed to upload user photo:' + error);
},
'POST',
{},
{ 'X-Connect-Bearer-Urls': 'true' },
{ fileUpload:
  {
    fileUrl: localPhotoUrl,
    fileMimeType: 'image/jpeg',
    fileName: 'pic.jpg'
  }
}
);
},
```

第 11 章 オフライン管理

トピック:

- SmartStore を使用したオフラインデータの安全な保存
- Mobile Sync を使用した Salesforce オブジェクトへのアクセス
- 設定ファイルの検証

Salesforce Mobile SDK には、オフラインで使用するためのデータを保存および同期するのに役立つ、次の 2 つのモジュールがあります。

- SmartStore は、暗号化された Salesforce データをデバイスに保存するオフラインストレージソリューションです。SqlLite を土台にしており、SqlCipher を使用して顧客データを暗号化します。SmartStore は、柔軟で拡大するデータモデルを処理できます。従来のデータベースとは異なり、さまざまな形式のレコードに対応しています。SmartStore を使用するアプリケーションは、「スープ」と呼ばれるインデックス付きのカスタムテーブルを定義して、そのデータを編成および管理します。スープを設定するには、Mobile SDK API を使用して Salesforce エンドポイントのデータを要求した後、SmartStore のメソッドをコールして、返されたレコードを更新/挿入します。SmartStore では、独自のクエリ言語である Smart SQL を使用して、ストアのデータを取得できます。SmartStore は、全文検索もサポートしており、ネイティブ、ハイブリッド、React Native の各プラットフォーム上で使用できます。
- Mobile Sync は、クラウド内の Salesforce レコードと SmartStore のローカルレコードを簡単に同期できるメカニズムを備えています。Mobile Sync API は、ユーザーに代わって Salesforce API をコールし、Salesforce データを取得または更新/挿入して、SmartStore スープを設定します。レコードを Salesforce に更新/挿入するときに、Mobile Sync では、同期プロセスをきめ細かく制御できます。Mobile Sync は、デフォルトのデータストアとして SmartStore を使用します。

たとえば、顧客を訪問するセールスエージェントが、インターネットに接続しにくいエリアを移動するとします。モバイルアプリケーションでは、次のことが可能です。

- Mobile SDK REST API または Mobile Sync `reSync()` メソッドを使用して、Salesforce クラウドの顧客データを SmartStore に事前に読み込みます。
- 移動中に、編集および更新したデータをローカルの SmartStore スープに保存します。これは、データの接続状況にかかわらず、行うことができます。
- デバイスがオンラインに復帰したとき、または事前に設定された間隔で Mobile Sync API を使用して、ローカルストアと Salesforce クラウド間でデータを同期します。

重要: SmartSync Data Framework は、「Salesforce Mobile Sync」に名前が変更されました。このため、SDK のコードで使用されているすべての「SmartSync」が「MobileSync」に変更され、モジュール名も変更されました。Mobile SDK 7.3 以前からアップグレードするには、これらの変更が反映されるように既存のプロジェクトを更新します。

「SmartStore」という名前は変わりません。

SmartStore を使用したオフラインデータの安全な保存

モバイルデバイスの接続はいつでも失われる可能性があり、病院や飛行機などの環境では接続が禁止されることがよくあります。このような状況に対処するには、モバイルアプリケーションがオフラインになっても引き続き動作するようにすることが重要です。

Mobile SDK は、モバイルデバイス向けの安全なマルチスレッドオフラインストレージソリューションである SmartStore を備えています。SmartStore により、ユーザーはデバイスの接続が失われても安全な環境で作業を続行できます。SmartStore を Mobile Sync と組み合わせると、接続が再開されたときにローカルの SmartStore データを Salesforce サーバーと簡単に同期できます。

このセクションの内容:

SmartStore について

SmartStore では、オフラインストレージのキャッシュと共に、非リレーショナルデスクトップデータベースの主要な機能(データセグメント化、インデックス付け、クエリ)を使用できます。

ハイブリッドおよびネイティブアプリケーションでの SmartStore の有効化

ハイブリッド Android アプリケーションで SmartStore を使用するには、いくつかの追加ステップを実行します。

既存の Android アプリケーションへの SmartStore の追加

Mobile SDK 4.0 以降で作成されたハイブリッドプロジェクトには、自動的に SmartStore が含まれます。Mobile SDK 4.0 以降を使用して SmartStore なしの Android ネイティブプロジェクトを作成した場合でも、後で簡単に SmartStore を追加できます。

ユーザーベースストアの作成およびアクセス

アプリケーションは SmartStore を初期化するときに、ストアのインスタンスを作成します。次に、ストアを使用してスープの登録と入力を行い、スープデータを操作します。ユーザーベースストアの場合、SmartStore がストアのライフサイクルを管理するため、ユーザーのセッションが終了した後でクリーンアップについて考慮する必要はありません。ただし、グローバルストアの場合、アプリケーションの終了時にアプリケーションでストアのデータを削除します。

グローバル SmartStore の使用

通常は SmartStore インスタンスを特定の顧客のログイン情報に関連付けますが、特別な要件でグローバルインスタンスにアクセスすることもできます。

設定ファイルを使用するスープの登録

Mobile SDK 6.0 より、SmartStore では、コードではなく設定ファイルによってスープの構造を定義できるようになりました。すべてのプラットフォームとアプリケーション種別は同じ設定ファイルを使用するため、1つのファイルにすべてのスープを記述できます。次に、そのファイルをコンパイルして、任意の Mobile SDK プロジェクトを作成できます。

インデックスパスでの配列の使用

インデックスパスに配列を含めることができますが、特定のルールが適用されます。

スープの入力

オフラインアクセス用に Salesforce レコードをスープに追加するには、REST API と SmartStore API を組み合わせて使用します。

スープからのデータの取得

SmartStore には、クエリ文字列を作成する一連のヘルパーメソッドが備えられています。

スマート SQL クエリ

クエリを完全に制御したり、既存の SQL クエリを再利用したりするには、カスタム SmartStore クエリを定義します。

全文検索クエリの使用

SmartStore で効率的かつ柔軟な検索を実行するには、全文クエリを使用します。全文クエリは「like」クエリと比べて、大きいデータセットを処理する場合にパフォーマンス上の大きな利点があります。

クエリ結果の処理

MobileSDK には、各プラットフォームで効率的、柔軟かつ動的にクエリ結果にアクセスできるメカニズムがあります。

データの挿入、更新、および更新/挿入

SmartStore では、エントリを追跡して外部サーバーとスープを同期できる標準項目が定義されます。

大きいスープ要素での外部ストレージの使用

スープ要素の削除

従来より、SmartStore のメソッドでは要素 ID の配列を指定してスープ要素を削除できます。通常これを行うには、候補 ID を取得する事前クエリを実行してから、削除を実行するメソッドをコールします。Mobile SDK 4.2 では SmartStore が強化され、要素削除メソッドにクエリオプションが追加されました。このオプションではクエリのみを指定し、SmartStore はそのクエリに適合するすべての要素を削除します。この方法では 1 回のコールでクエリと削除操作の両方が行われるため、パフォーマンスが向上します。

スープの管理

SmartStore には、スープメタデータを取得して、スूपレベルの他の操作を実行できるユーティリティ機能が用意されています。この機能は、ハイブリッド、React Native、Android ネイティブ、および iOS ネイティブアプリケーションで使用できます。

ストアの管理

グローバルストアを作成する場合、アプリケーションの終了時にクリーンアップを実行する必要があります。また、複数のユーザーストアを作成する場合、特定のストアを使用しなくなったときにクリーンアップを実行できます。SmartStore では、名前付きストアやグローバルストアを削除するメソッドが提供されます。ハイブリッドアプリケーションの場合、SmartStore では、名前付きストアのリストを取得する関数も提供されます。

SmartStore インспекターを使用したテスト

テスト中に SmartStore の操作を確認することは面倒で時間がかかります。SmartStore インспекターでこの問題を解消できます。

擬似 SmartStore の使用

コンテナ外で実行しながら SmartStore を使用するコードの開発とテストを容易にするため、エミュレートされた SmartStore を使用できます。

Mobile Sync のスープの準備

通常、Salesforce クラウドと情報を交換するスープは、Mobile Sync を使用して同期できます。Mobile Sync をサポートするために、大部分のアプリケーション種別では「上位同期」操作の特殊なスープ項目を作成および管理する必要があります。

Swift アプリケーションでの SmartStore の使用

SmartStore の基礎部分は、forceios ネイティブ Swift プロジェクトに簡単にインストールできます。

SmartStore について

SmartStore では、オフラインストレージのキャッシュと共に、非リレーショナルデスクトップデータベースの主要な機能 (データセグメント化、インデックス付け、クエリ) を使用できます。

⚠ 重要: 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

SmartStore の新機能

スマート SQL では、SELECT または WHERE 句で参照されるすべての項目のインデックスパスが不要になりました。この改良は、外部ストレージを使用するスープには適用されません。

データキャッシュについて

オフライン同期および競合の解決サービスを提供するため、SmartStore は StoreCache (Mobile SDK のキャッシュメカニズム) を使用します。StoreCache を使用して、Salesforce データの操作を管理することをお勧めします。

📌 メモ: 純粋な HTML5 アプリケーションは、オフライン情報をブラウザのキャッシュに保存します。ブラウザのキャッシュは Mobile SDK の一部ではないため、ここでは説明していません。SmartStore は、ストレージ機能をデバイスで使用します。この方法には、ネイティブ開発またはハイブリッド開発が必要です。

サンプルコードについて

この章の Objective-C コードスニペットでは、すべての Salesforce 組織で事前定義されている取引先オブジェクトと商談オブジェクトを使用しています。取引先と商談は主従関係でリンクされています。1つの取引先が複数の商談のマスターになる場合があります。

このセクションの内容:

SmartStore のスープ

SmartStore のスープでは、オフラインコンテンツをパーティションで区切ることができます。

SmartStore のストア

SmartStore では、暗号化されたスープデータがストアと呼ばれる基礎システムデータベースに置かれます。ストアでは、すべてのスープデータが保存、暗号化、関連付け、インデックス付けされます。デバイスの接続が失われた場合、ユーザーは Salesforce クラウドに再度アクセスできるようになるまで、ストア内のデータで作業を続行できます。

SmartStore のデータ型

他のデータベース同様、SmartStore ではスーブの作成に使用する一連のデータ型が定義されます。SmartStore のデータ型は、基礎となる SQLite データベースと同じです。

関連トピック:

[オフラインキャッシュのための StoreCache の使用](#)

[競合検出](#)

[スマート SQL クエリ](#)

SmartStore のスーブ

SmartStore のスーブでは、オフラインコンテンツをパーティションで区切ることができます。

SmartStore は、スーブと呼ばれる論理コレクションにオフラインデータを保存します。SmartStore スーブは、基礎となる SQLite データベース (ストア) の 1 つのテーブルを表し、通常は標準またはカスタム Salesforce オブジェクトに対応付けられます。スーブにはスーブ要素が含まれます。各要素は、1 つのデータベース行を反映する JSON オブジェクトです。データアクセスを合理化するには、各スーブのインデックスを定義します。SmartStore ヘルパーメソッドまたは SmartStore の Smart SQL クエリ言語でこれらのインデックスを使用してスーブを照会します。SmartStore インデックスはまた、全文検索クエリをサポートするため、管理業務が容易になります。

スーブをテーブル、ストアをデータベースとして考えるとわかりやすくなります。スーブはアプリケーションで必要なだけ定義できます。スーブは自己完結型のデータセットであるため、相互の定義済みリレーションはありませんが、Smart SQL 結合を使用してそれらを照会できます。また、ネイティブアプリケーションでは、1 つのトランザクション内で複数のスーブに書き込むことができます。



警告: SmartStore データは揮発性です。ほとんどの場合、この有効期限は、認証されたユーザーおよび OAuth トークンの状態に関係します。ユーザーがアプリケーションからログアウトすると、そのユーザーに関連付けられているスーブデータは SmartStore によりすべて削除されます。同様に、OAuth 更新トークンが取り消されるか期限切れになると、ユーザーのアプリケーション状態がリセットされ、SmartStore のすべてのデータが消去されます。アプリケーションを設計するときは、特に組織で更新トークンの有効期限を短期に設定する場合に、SmartStore データの揮発性に留意します。

SmartStore のストア

SmartStore では、暗号化されたスーブデータがストアと呼ばれる基礎システムデータベースに置かれます。ストアでは、すべてのスーブデータが保存、暗号化、関連付け、インデックス付けされます。デバイスの接続が失われた場合、ユーザーは Salesforce クラウドに再度アクセスできるようになるまで、ストア内のデータで作業を続行できます。

SmartStore を初期化するときは、開くストアの名前を指定します。カスタム名を割り当てるか、iOS ネイティブアプリケーションの `kDefaultSmartStoreName` などの標準名を使用して、ストアを定義します。指定ストアはスーブのようにユーザー固有であり、ユーザーのセッションが有効な間のみ存続します。

従来の SmartStore セッションでは、すべてのスーブで 1 つのストアのコンテンツが参照、整理、および操作されます。単一ストア設定は、多数のアプリケーションにとって最適です。ただし、アプリケーションが多数のオブジェクトの大量データを照会する場合、パフォーマンスが低下することがあります。応答時間が遅くなることを避けるため、複数の指定ストアを作成してストア間でデータを分割できます。たとえば、アプリケー

ションでSalesforceデータの明確な領域で作業するタスクが定義されている場合、各タスクに対してストアを作成できます。より小さいストアへのランタイムアクセスによって、ユーザー満足度が大幅に向上する可能性があります。

一部の使用事例では、ストアがユーザーのログイン情報に関連付けられず、ユーザーとアプリケーションのセッションで存続できる必要があります。SmartStoreはこの要件に対応するため、グローバルストアをサポートしています。グローバルストアも指定ストアですが、グローバルストアの作成と削除には異なるAPIセットを使用します。

関連トピック:

[グローバル SmartStore の使用](#)

[ユーザーベースストアの作成およびアクセス](#)

SmartStore のデータ型

他のデータベース同様、SmartStore ではスーパの作成に使用する一連のデータ型が定義されます。SmartStore のデータ型は、基礎となる SQLite データベースと同じです。

SmartStore では、「インデックス指定」を宣言するために次のデータ型がサポートされます。SmartStore のスーパ定義では、インデックス指定は SmartStore が特定の項目内で検索するデータ型を定義します。

型	説明
integer	4 バイト (SDK 2.1 以前) または 8 バイト (SDK 2.2 以降) で保存される符号付き整数
floating	8 バイトの IEEE 浮動小数点数として保存される浮動小数点値
string	データベースエンコード (UTF-8) で保存されるテキスト文字列
full_text	全文検索をサポートする文字列
JSON1	SQLite JSON1 拡張に基づくインデックスの型。integer、floating、string 型の代わりに使用できます。JSON1 では配列をトラバースするインデックスパスがサポートされないこと以外は、各型のインデックス指定と同様に動作します。

このセクションの内容:

[日付表現](#)

関連トピック:

[インデックスパスでの配列の使用](#)

日付表現

SmartStore では、日付/時間のデータ型は定義されていません。日付/時間項目のインデックス指定を作成する場合は、JSON 入力の形式に一致する SmartStore の型を選択します。たとえば、Salesforce は日付を文字列として送信するため、Salesforce の日付項目には常に文字列インデックス指定を使用します。Salesforce 以外の日付項目やカスタム日付項目のインデックスの型を選択する場合は、次の表を参照してください。

型	形式	説明
string	ISO 8601 文字列	「YYYY-MM-DD HH:MM:SS.SSS」
floating	ユリウス日数	グリニッジの紀元前 4714 年 11 月 24 日の正午からの予期的グレゴリオ暦に応じた日数。この値には、小数で表される部分的な日数を含めることができます。
integer	Unix 時間	1970-01-01 00:00:00 UTC からの秒数

ハイブリッドおよびネイティブアプリケーションでの SmartStore の有効化

ハイブリッド Android アプリケーションで SmartStore を使用するには、いくつかの追加ステップを実行します。

ハイブリッドアプリケーションでは、JavaScript を使用して SmartStore にアクセスします。ハイブリッドアプリケーションでオフラインアクセスを有効にするには、Visualforce または HTML ページに `cordova.js` ライブラリファイルを含める必要があります。

forceios および forcedroid ネイティブアプリケーションでは、SmartStore は常に含まれます。

関連トピック:

[forcedroid を使用した Android プロジェクトの作成](#)

[forceios を使用した iOS プロジェクトの作成](#)

既存の Android アプリケーションへの SmartStore の追加

Mobile SDK 4.0 以降で作成されたハイブリッドプロジェクトには、自動的に SmartStore が含まれます。Mobile SDK 4.0 以降を使用して SmartStore なしの Android ネイティブプロジェクトを作成した場合でも、後で簡単に SmartStore を追加できます。

既存のネイティブ Android プロジェクト (Mobile SDK 4.0 以降) に SmartStore を追加する手順は、次のとおりです。

1. SmartStore ライブラリプロジェクトをプロジェクトに追加します。Android Studio で、プロジェクトの `build.gradle` ファイルを開き、`dependencies` セクションで `:libs:SmartStore` の `compile` ディレクティブを追加します。`dependencies` セクションが存在しない場合は作成します。

```
dependencies {
  ...
}
```

```
compile project(':libs:SmartStore')
}
```

2. <projectname>App.java ファイルで、SalesforceSDKManager の代わりに SmartStoreSDKManager クラスをインポートします。次のステートメントを置換します。

```
import com.salesforce.androidsdk.
    app.SalesforceSDKManager
```

置換後のステートメントは、次のとおりです。

```
import com.salesforce.androidsdk.smartstore.app.SmartStoreSDKManager
```

3. <projectname>App.java ファイルで、SalesforceSDKManager ではなく SmartStoreSDKManager クラスを拡張するようにアプリケーションクラスを変更します。

メモ:

1. Mobile SDK 3.x 以前で作成したアプリケーションに SmartStore を追加するには、最初に最新バージョンの Mobile SDK にアップグレードします。
2. SmartStore プラグイン、com.salesforce.plugin.smartstore.client は内部で promise を使用します。

Mobile SDK の promise ベースの API には、以下が含まれます。

- force+promise.js
- smartstoreclient Cordova プラグイン (com.salesforce.plugin.smartstore.client)
- mobilesync.js

関連トピック:

[以前のリリースからの移行](#)

ユーザーベースストアの作成およびアクセス

アプリケーションは SmartStore を初期化するとき、ストアのインスタンスを作成します。次に、ストアを使用してスーパの登録と入力を行い、スーパデータを操作します。ユーザーベースストアの場合、SmartStore がストアのライフサイクルを管理するため、ユーザーのセッションが終了した後でクリーンアップについて考慮する必要はありません。ただし、グローバルストアの場合、アプリケーションの終了時にアプリケーションでストアのデータを削除します。

Android ネイティブアプリケーション

Android では、後でストアを作成するために使用する SmartStoreSDKManager のインスタンスを最初に取得する必要があります。

```
SmartStoreSDKManager sdkManager =
    SmartStoreSDKManager.getInstance();
```

```
SmartStore smartStore = sdkManager.getSmartStore(); // Creates a default store for the
current user
```

引数を指定しないで `SmartStoreSDKManager.getSmartStore()` をコールすると、必ずデフォルトの匿名ストアにアクセスします。名前付きユーザーベースストアを作成するには、次のメソッドをコールします。

```
public SmartStore getSmartStore(String dbNamePrefix, UserAccount account, String communityId)
```

`account` と `communityId` の両方を `null` にできます。これらのメソッドは、追加のストアを作成するために何度でもコールできます。

iOS ネイティブアプリケーション

ストアを作成するために、iOS では `sharedStoreWithName:` クラスメッセージが提供されます。

```
- (SFSmartStore *)store
{
    return [SFSmartStore sharedStoreWithName:kDefaultSmartStoreName]; // Creates a default
store for the current user
}
```

Swift:

```
var store = SmartStore.shared(withName: storeName)
```

`kDefaultSmartStoreName` 以外の文字列を渡して、カスタム名でストアを作成できます。このメソッドは、追加のストアを作成するために何度でもコールできます。

ハイブリッドアプリケーション

ハイブリッドアプリケーションでは、ユーザーベースストアとグローバルストアに同じ方法でアクセスします。ストアを明示的に作成する代わりに、`registerSoup()` をコールしたときにまだ存在していないストアを自動的に作成します。後で直接参照するためなどに名前付きストアを使用する場合は、`StoreConfig` オブジェクトを第1引数として指定してこの関数をコールします。この関数のオブジェクトは、ストア名と、ストアがグローバルかどうかを示す `Boolean` 値を取ります。

```
var StoreConfig = function (storeName, isGlobalStore) {
    this.storeName = storeName;
    this.isGlobalStore = isGlobalStore;
};
```

このオブジェクトは、省略可能な第1引数として大部分のスープ関数に渡すことができます。使用する場合、`StoreConfig` オブジェクトによって実行コンテキストが設定されます。`storeName` または `isGlobalStore` は省略可能で、1つまたは両方を指定できます。`SmartStore` は、`StoreConfig` オブジェクトを次のように評価します。

- `storeName` が指定されていない場合、`this.storeName` は `SmartStore` のデフォルトのストア名に設定されます。
- `isGlobalStore` が指定されていない場合、`this.isGlobalStore` は `false` に設定されます。
- ストア名は一意である必要はありません。1つのストア名を2回使用できます (ユーザーベースストアで1回、グローバルストアで1回)。

- `isGlobalStore` 設定で指定した領域に存在しないストア名を指定すると、SmartStore によって作成されます。

次の例では、`soupName` スープが含まれる「Store1」という名前のユーザーベースストアを作成します。

```
navigator.smartstore.registerSoup({storeName: "Store1", isGlobalStore:false}, soupName,
    indexSpecs, successCallback, errorCallback)
```

必要に応じていくつでも異なるスープ名で `registerSoup()` をコールできます。StoreConfig を渡さずに スープ関数をコールすると、SmartStore は必ずデフォルトのユーザーベース (非グローバル) ストアに対して操作を実行します。この動作は、名前付きストアを作成した場合でも適用されます。次の例では、提供されたインデックス指定を使用して `soupName` という名前のスープを現在のユーザーのデフォルトストアに作成します。

```
var sfSmartstore = function() {
    return cordova.require("com.salesforce.plugin.smartstore");
};
sfSmartstore().registerSoup(soupName, indexSpecs, successCallback, errorCallback);
```

関連トピック:

[SmartStore のストア](#)

グローバル SmartStore の使用

通常は SmartStore インスタンスを特定の顧客のログイン情報に関連付けますが、特別な要件でグローバルインスタンスにアクセスすることもできます。

特定の状況では、一部のアプリケーションで、Salesforce 認証に関連付けられていない SmartStore インスタンスへのアクセスが必要になります。このような状況は、アプリケーションの状態や、Salesforce ユーザー、組織、またはコミュニティに依存しない他のデータを保存するアプリケーションで発生する場合があります。この状況に対応するため、Mobile SDK ではアプリケーションのライフサイクルを過ぎても存続するグローバルストアがサポートされています。

グローバルストアに保存されるデータは、ユーザー認証に依存しないため、ログアウト時に削除されません。グローバルストアは、ログアウト後もそのまま保持されるため、アプリケーションの終了時にそのデータを削除する必要があります。Mobile SDK には、このための API が用意されています。

❗ 重要: ユーザー固有のデータをグローバル SmartStore に保存しないでください。このようにすると、ユーザーデータがユーザーのログアウト後も保持される可能性があるため、Mobile SDK セキュリティ要件に違反します。

Android API

Android の場合、SmartStoreSDKManager のインスタンスを介してグローバル SmartStore にアクセスします。

- ```
public SmartStore getGlobalSmartStore(String dbName)
```

指定したデータベース名のグローバル SmartStore インスタンスを返します。dbName には、「smartstore」以外の任意の文字列を設定できます。デフォルトのグローバル SmartStore データベースを使用するには、dbName を null に設定します。

- ```
public boolean hasGlobalSmartStore(String dbName)
```

指定したデータベース名のグローバル SmartStore インスタンスが存在しているかどうかをチェックします。デフォルトのグローバル SmartStore の存在を確認するには、dbName を null に設定します。

- ```
public void removeGlobalSmartStore(String dbName)
```

指定したグローバル SmartStore データベースを削除します。この名前には、「smartstore」以外の任意の文字列を設定できます。デフォルトのグローバル SmartStore を削除するには、dbName を null に設定します。

## iOS API

iOS の場合、SFSmartStore のインスタンスを介してグローバル SmartStore にアクセスします。

- Objective-C:**

```
+ (id)sharedGlobalStoreWithName:(NSString *)storeName
```

**Swift:**

```
var gstore = SmartStore.sharedGlobal(withName: storeName)
```

指定したデータベース名のグローバル SmartStore インスタンスを返します。storeName には、「defaultStore」以外の任意の文字列を設定できます。デフォルトのグローバル SmartStore を使用するには、storeName を kDefaultSmartStoreName に設定します。

- Objective-C:**

```
+ (void)removeSharedGlobalStoreWithName:(NSString *)storeName
```

**Swift:**

```
SmartStore.removeSharedGlobal(withName: storeName)
```

指定したグローバル SmartStore データベースを削除します。storeName には、「defaultStore」以外の任意の文字列を設定できます。デフォルトのグローバル SmartStore を使用するには、storeName を kDefaultSmartStoreName に設定します。

## ハイブリッド API

ほとんどの SmartStore JavaScript スープレットは、グローバル SmartStore を使用するかどうかを示す省略可能な最初の引数を取ります。この引数には Boolean 値または StoreConfig オブジェクトを指定できます。この引数が指定されていない場合、Mobile SDK はデフォルトのユーザーストアを使用します。

次に例を示します。

```
querySoup([isGlobalStore,]soupName, querySpec,
 successCB, errorCallback);
```

```
querySoup([storeConfig,],soupName, querySpec,
 successCB, errorCallback);
```

SmartStore には、ストアを削除する次の関数が定義されています。各関数は、成功コールバックとエラーコールバックを取得します。removeStore() 関数には、ストア名を指定する StoreConfig オブジェクト、または文字列のストア名のみのいずれかも必要です。

```
removeStore(storeConfig,successCB, errorCallback)
removeAllGlobalStores(successCB, errorCallback)
removeAllStores(successCB, errorCallback)
```

関連トピック:

[SmartStore のストア](#)

[ストアの管理](#)

[ユーザーベースストアの作成およびアクセス](#)

## 設定ファイルを使用するスープの登録

Mobile SDK 6.0 より、SmartStore では、コードではなく設定ファイルによってスープの構造を定義できるようになりました。すべてのプラットフォームとアプリケーション種別は同じ設定ファイルを使用するため、1つのファイルにすべてのスープを記述できます。次に、そのファイルをコンパイルして、任意の Mobile SDK プロジェクトを作成できます。

スープを登録するには、スープ名および1つ以上のインデックス指定のリストを指定します。

そのエントリで検出された1つ以上の項目にスープをインデックス付けします。SmartStore により、これらのインデックスが挿入、更新、削除操作を反映していることが確認されます。スープを登録する場合、少なくとも1つのインデックス項目を常に指定します。たとえば、シンプルなキー/値ストアとしてスープを使用している場合は、文字列型の1つのインデックス指定を使用します。

## 概要

SmartStore 設定ファイルは JSON オブジェクトを使用してスープ定義を表現します。設定ファイルの JSON スキーマは、すべてのアプリケーション種別およびプラットフォームで同じです。ハイブリッドアプリケーションは設定ファイルを自動的に読み込みますが、それ以外のアプリケーションは読み込みに1行のコードを使用します。メカニズムをシンプルにするために、Mobile SDK では次のファイル命名規則が適用されます。

- デフォルトのグローバルストアのスープを定義するには、globalstore.json というファイルを指定します。
- デフォルトのユーザーストアのスープを定義するには、userstore.json というファイルを指定します。

設定ファイルは、デフォルトのグローバルストアおよびデフォルトのユーザーストアのスープのみを定義できます。指定ストアについては、コードを使用してスープを登録します。設定ファイルを使用して、外部に保存されるスープを設定することはできません。



 **メモ:**

- 設定ファイルは、最初の設定のみを目的としています。JSON ファイルを改訂して実行時に再度読み込み、既存のスープを変更することはできません。代わりに、`alterSoup()` などの SmartStore メソッドを使用します。「[スープの管理](#)」を参照してください。
- 設定ファイルで既存のスープを定義すると、Mobile SDK ではその設定ファイルが無視されます。この場合、スープを設定して管理するにはコードを使用する必要があります。

## 設定ファイルの形式

JSON 形式は、次の例で示すように自明です。

```
{ "soups": [
 {
 "soupName": "soup1",
 "indexes": [
 { "path": "stringField1", "type": "string"},
 { "path": "integerField1", "type": "integer"},
 { "path": "floatingField1", "type": "floating"},
 { "path": "json1Field1", "type": "json1"},
 { "path": "ftsField1", "type": "full_text"}
]
 },
 {
 "soupName": "soup2",
 "indexes": [
 { "path": "stringField2", "type": "string"},
 { "path": "integerField2", "type": "integer"},
 { "path": "floatingField2", "type": "floating"},
 { "path": "json1Field2", "type": "json1"},
 { "path": "ftsField2", "type": "full_text"}
]
 }
]
```

Mobile Sync との互換性を保つために、設定ファイルにはいくつかのシステム項目のインデックスも必要です。「[Mobile Sync のスープの準備](#)」を参照してください。

## 設定ファイルの場所

設定ファイルがどこに配置されるかは、アプリケーション種別やプラットフォームによって異なります。Mobile SDK は、次の場所で設定ファイルを探します。

### iOS (ネイティブおよび React Native)

リソースバンドルの下/内

### Android (ネイティブおよび React Native)

`/res/raw` プロジェクトフォルダー内

### ハイブリッド

Cordova プロジェクトで、次の手順を実行します。

1. 設定ファイルを最上位の `www/` フォルダに配置します。
2. 最上位のプロジェクトディレクトリで、`cordova prepare` を実行します。

## ネイティブアプリケーションでの SmartStore 設定ファイルの読み込み

SmartStore とそのコンパニオン機能の Mobile Sync では、特別な SDK マネージャーオブジェクトが必要です。たとえば、iOS で SmartStore または Mobile Sync を使用するには、`SalesforceSDKManager.initializeSDK()` ではなく、`MobileSyncSDKManager.initializeSDK()` をコールして SDK を初期化します。

Mobile Sync を使用していない場合は、`SmartStoreSDKManager.initializeSDK()` をコールします。ただし、このようなケースはめったにありません。

ネイティブアプリケーションおよび React Native アプリケーションの場合は、読み込みメソッドをコールして JSON 設定ファイルを読み込みます。このコールは、顧客が正常にログインした後にアプリケーション初期化コード内で実行します。たとえば、iOS の場合、このコールは、`loginIfRequired` に渡すブロック内で実行します。このメソッドをコールするのは、SmartStore を設定するコードの代わりに `globalstore.json` ファイルまたは `userstore.json` ファイルを使用している場合に限られます。この読み込みメソッドは、複数回コールしないでください。これらを含むハイブリッドアプリケーションの場合、SmartStore 設定ファイルは、自動的に読み込まれます。

スーブ設定ファイルを読み込むには、対象にするストアのローダーメソッドをコールします。その他の SmartStore メソッドをコールする前に、このファイルを読み込みます。

### iOS (ネイティブおよび React Native)

`MobileSyncSDKManager` オブジェクトの適切なメソッドをコールすることによってスーブ設定ファイルを読み込みます。

### デフォルトのユーザーストアの読み込み

#### Swift

```
// In the AppDelegate class:
override init() {
 super.init()
 MobileSyncSDKManager.initializeSDK()
 ...

 // Load config files in the block you pass to loginIfRequired()
 func application(_ application: UIApplication, didFinishLaunchingWithOptions
 launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

 self.window = UIWindow(frame: UIScreen.main.bounds)
 self.initializeAppViewState()
 // ...

 AuthHelper.loginIfRequired {
 self.setupRootViewController()
 MobileSyncSDKManager.shared.setupUserStoreFromDefaultConfig()
 }
 ...
 }
```

## Objective-C

```
// In the AppDelegate class:
- (instancetype)init
{
 self = [super init];
 if (self) {
 [MobileSyncSDKManager initializeSDK];
 ...

// Load config files in the block you pass to loginIfRequired()
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
 self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen].bounds];
 [self initializeAppViewState];
 ...

 [SFSDKAuthHelper loginIfRequired:^(
 [self setupRootViewController];
 [MobileSyncSDKManager sharedManager] setupUserStoreFromDefaultConfig];
)];
 ...
}
```

## デフォルトのグローバルストアの読み込み

## Swift

```
// In the AppDelegate class:
override init() {
 super.init()
 MobileSyncSDKManager.initializeSDK()
 ...

// Load config files in the block you pass to loginIfRequired()
func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
 self.window = UIWindow(frame: UIScreen.main.bounds)
 self.initializeAppViewState()
 // ...

 AuthHelper.loginIfRequired {
 self.setupRootViewController()
 MobileSyncSDKManager.shared.setupGlobalStoreFromDefaultConfig()
 }
 ...
}
```

## Objective-C

```
// In the AppDelegate class:
- (instancetype)init
{
 self = [super init];
 if (self) {
 [MobileSyncSDKManager initializeSDK];
 ...
}
```

```
// Load config files in the block you pass to loginIfRequired()
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
 self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen].bounds];
 [self initializeAppViewState];
 ...

 [SFSDKAuthHelper loginIfRequired:^(
 [self setupRootViewController];
 [[MobileSyncSDKManager sharedManager] setupGlobalStoreFromDefaultConfig];
)];
 ...
}
```

### Android (ネイティブおよび React Native)

MobileSyncSDKManager オブジェクトの適切なメソッドをコールすることによってスーブ設定ファイルを読み込みます。

#### デフォルトのユーザーストアの読み込み

```
public class MainApplication extends Application {

 @Override
 public void onCreate() {
 super.onCreate();
 MobileSyncSDKManager.initNative(getApplicationContext(), MainActivity.class);
 MobileSyncSDKManager.getInstance().setupUserStoreFromDefaultConfig();
 ...
 }
}
```

#### デフォルトのグローバルストアの読み込み

```
public class MainApplication extends Application {

 @Override
 public void onCreate() {
 super.onCreate();
 MobileSyncSDKManager.initNative(getApplicationContext(), MainActivity.class);
 MobileSyncSDKManager.getInstance().setupGlobalStoreFromDefaultConfig();
 ...
 }
}
```

### ハイブリッド

SmartStore は、スーブ設定ファイルを見つけると、自動的にそのファイルを読み込みます。ファイルをプロジェクトに追加する手順は次のとおりです。

1. 設定ファイル (userstore.json または globalstore.json) をハイブリッドプロジェクトディレクトリの最上位の www/ ディレクトリにコピーします。
2. コマンドプロンプトまたはターミナルウィンドウで、ハイブリッドプロジェクトディレクトリに変更し、`cordova prepare` を実行します。

## サンプルコード

MobileSyncExplorer および MobileSyncExplorerHybrid サンプルアプリケーションは、設定ファイルを使用して SmartStore スープを設定します。

### メモ:

- SmartStore ローダーメソッドをコールするのは、スープの定義に `userstore.json` を使用している場合に限られます。設定ファイルではなく、コードでスープを設定する場合は、ローダーメソッドをコールしないでください。
- ローダーメソッドは、顧客がログインした後にコールしてください。
- ローダーメソッドは、複数回コールしないでください。

**例:** SmartStore は、ネイティブバージョンおよびハイブリッドバージョンの MobileSyncExplorer サンプルで、同じ設定ファイル `userstore.json` を使用します。Mobile Sync を使用している場合は、この設定の最後の5つのパスが必要です。

```
{ "soups": [
 {
 "soupName": "contacts",
 "indexes": [
 { "path": "Id", "type": "string"},
 { "path": "FirstName", "type": "string"},
 { "path": "LastName", "type": "string"},
 { "path": "__local__", "type": "string"},
 { "path": "__locally_created__", "type": "string"},
 { "path": "__locally_updated__", "type": "string"},
 { "path": "__locally_deleted__", "type": "string"},
 { "path": "__sync_id__", "type": "integer"}
]
 }
]
}
```

このセクションの内容:

### コードを使用したスープの登録

スープにアクセスするには、まずスープを登録する必要があります。

関連トピック:

[Mobile Sync のスープの準備](#)

## コードを使用したスープの登録

スープにアクセスするには、まずスープを登録する必要があります。

スープを登録するには、スープ名および1つ以上のインデックス指定のリストを指定します。何らかの理由で設定ファイルを使用してスープの構造を定義できない場合に、コードを使用して同じことを行う方法を次に示

します。次の各コード例では、名前、ID、および所有者(または親)ID項目で構成されるインデックス指定配列を作成します。

そのエントリで検出された1つ以上の項目にスープをインデックス付けします。SmartStoreにより、これらのインデックスが挿入、更新、削除操作を反映していることが確認されます。スープを登録する場合、少なくとも1つのインデックス項目を常に指定します。たとえば、シンプルなキー/値ストアとしてスープを使用している場合は、文字列型の1つのインデックス指定を使用します。

#### メモ:

- Mobile SDK 6.0 以降、ネイティブアプリケーションでは、JSON 設定ファイルを使用してスープを登録できるようになりました。可能な場合は、コーディングよりこちらの方法をお勧めします。[「設定ファイルを使用したスープの登録」](#)を参照してください。
- スープに非常に大きい要素 (1 MB 以上) が含まれている場合、外部ストレージを使用するように登録することを検討してください。[「大きいスープ要素での外部ストレージの使用」](#)を参照してください。

## ハイブリッドアプリケーション

スープを登録する JavaScript 関数には、正常およびエラー条件用のコールバック関数が必要です。

```
var sfSmartstore = function() {
 return cordova.require("com.salesforce.plugin.smartstore");
};
sfSmartstore().registerSoup(soupName, indexSpecs, successCallback, errorCallback);
```

スープが存在しない場合は、この関数で作成されます。スープが存在する場合は、登録することでスープにアクセスできます。

### indexSpecs

indexSpecs 配列を使用して、事前定義されたインデックス付けでスープを作成します。indexSpecs 配列のエントリは、スープのインデックス付け方法を指定します。各エントリは、path:type ペアから構成されます。path はインデックス項目の名前、type は「string」、「integer」、「floating」、「full\_text」、または「json1」のいずれかです。

```
var indexSpecs = [
 {path:"Name",type:"string"},
 {path:"Id",type:"string"}
];
```

#### メモ:

- インデックスパスでは大文字と小文字が区別され、Owner.Name などの複合パスを含めることができます。
- indexSpecs 配列で記述された項目が不足しているインデックスエントリは、そのインデックスで追跡されなくなります。
- インデックスのタイプは、インデックスにのみ適用されます。インデックス付き項目 (「select {soup:path} from {soup}」など) を照会すると、インデックス指定で指定したタイプのデータが返されます。
- インデックス列には null 項目を含めることができます。

- Mobile SDK 4.1 以降、「内部」(非リーフ) ノードへのインデックスパスを指定できます。これらのパスを `like` および `match` (全文) クエリで使用できます。内部ノードパスを定義する場合は、`string` 型を使用します。

たとえば、「spies」という名前のスープに次の要素があるとします。

```
{
 "first_name": "James",
 "last_name": "Bond",
 "address": {
 "street_number": 10,
 "street_name": "downing",
 "city": "london"
 }
}
```

この場合、「address」には子があるため、これは内部ノードです。「address」パスのインデックスを介して、`like` または `match` クエリを使用して「address」内の「city」で値「london」を検索できます。次に例を示します。

```
SELECT {spies:first_name, spies:last_name} FROM spies WHERE {spies:address} LIKE 'london'
```

- Mobile SDK 4.1 以降、インデックスパスに配列を含めることができますが、いくつかの制限があります。「[インデックスパスでの配列の使用](#)」を参照してください。

#### successCallback

指定した成功コールバック関数は、1つの引数(スープ名)を取ります。次に例を示します。

```
function(soupName) { alert("Soup " + soupName + " was successfully created"); }
```

スープが正常に作成されると、`registerSoup()` は、成功コールバック関数をコールして、スープの準備が整ったことを示します。トランザクションが完了するまで待機し、他の操作を開始する前にコールバックを受け取ります。渡された名前でスープを登録すると、成功コールバック関数からスープが返されます。

#### errorCallback

エラーコールバック関数は、1つの引数(エラーの説明文字列)を取ります。

```
function(err) { alert ("registerSoup failed with error: " + err); }
```

スープの作成時にエラーが発生する場合、次のようなさまざまな理由が考えられます。

- スープ名が無効または不正である
- インデックスがない(少なくとも1つのインデックス指定が必要)
- データベースエラーなど、他の予期しないエラーが発生した

スープが存在するかどうかを確認するには、次の関数を使用します。

```
navigator.smartstore.soupExists(soupName, successCallback, errorCallback);
```



## Android ネイティブアプリケーション

Android の場合、`com.salesforce.androidsdk.smartstore.store.IndexSpec` タイプの配列でインデックス指定を定義します。各インデックス指定はパス(インデックス項目名)と型で構成されます。インデックス指定の型は `SmartStore.Type Enum` で定義され、次の値が含まれます。

- `string`
- `integer`
- `floating`
- `full_text`
- `json1`

```
public class OfflineStorage extends SalesforceActivity {
 private SmartStore smartStore;
 final IndexSpec[] ACCOUNTS_INDEX_SPEC = {
 new IndexSpec("Name", SmartStore.Type.string),
 new IndexSpec("Id", SmartStore.Type.string),
 new IndexSpec("OwnerId", SmartStore.Type.string)
 };

 public OfflineStorage() {
 smartStore = SmartStoreSDKManager.getInstance().getSmartStore();
 smartStore.registerSoup("Account", ACCOUNTS_INDEX_SPEC);
 }

 // ...
}
```

## iOS ネイティブアプリケーション

iOS の場合、`SFSoupIndex` オブジェクトの配列でインデックス指定を定義します。各インデックス指定はパス(インデックス項目名)と型で構成されます。インデックス指定の型は、`SFSoupIndex` クラスで定数として定義され、次の値があります。

- `kSoupIndexTypeString`
- `kSoupIndexTypeInteger`
- `kSoupIndexTypeFloating`
- `kSoupIndexTypeFullText`
- `kSoupIndexTypeJSON1`

### Swift

Mobile SDK 8.0 以降では、SmartStore ネイティブ Swift 拡張により、次のスープ登録メソッドが提供されます。

```
func createAccountsSoup(name: String) {
 guard let user = UserAccountManager.shared.currentUserAccount,
 let store = SmartStore.shared(withName: SmartStore.defaultStoreName,
 forUserAccount: user),
 let index1 = SoupIndex(path: "Name", indexType: "String",
 columnName: "Name"),
 let index2 = SoupIndex(path: "Id", indexType: "String",
 columnName: "Id")
```

```

 else {
 return
 }

 do {
 try store.registerSoup(withName: name, withIndices:[index1,index2])
 } catch (let error) {
 SalesforceLogger.d(RootViewController.self,
 message:"Couldn't create soup \(name).
 Error: \(error.localizedDescription)")
 return
 }
}

```

## Objective-C

```

NSString* const kAccountSoupName = @"Account";

...
- (SFSmartStore *)store
{
 return [SFSmartStore sharedStoreWithName:kDefaultSmartStoreName];
}

...
- (void)createAccountsSoup {
 if (![self.store soupExists:kAccountSoupName]) {
 NSArray *keys = @[@"path", @"type"];
 NSArray *nameValues = @[@"Name", kSoupIndexTypeString];
 NSDictionary *nameDictionary = [NSDictionary
 dictionaryWithObjects:nameValues forKeys:keys];

 NSArray *idValues = @[@"Id", kSoupIndexTypeString];
 NSDictionary *idDictionary =
 [NSDictionary dictionaryWithObjects:idValues forKeys:keys];

 NSArray *ownerIdValues = @[@"OwnerId", kSoupIndexTypeString];
 NSDictionary *ownerIdDictionary =
 [NSDictionary dictionaryWithObjects:ownerIdValues
 forKeys:keys];

 NSArray *accountIndexSpecs =
 [SFSoupIndex asArraySoupIndexes:@[nameDictionary,
 idDictionary, ownerIdDictionary]];

 NSError* error = nil;
 [self.store registerSoup:kAccountSoupName
 withIndexSpecs:accountIndexSpecs
 error:&error];

 if (error) {
 NSLog(@"Cannot create SmartStore soup '%@\nError: '%@'",
 kAccountSoupName, error.localizedDescription);
 }
 }
}

```

```
 }
 }
}
```

関連トピック:

- SmartStore のデータ型
- 全文検索クエリの使用

## インデックスパスでの配列の使用

インデックスパスに配列を含めることができますが、特定のルールが適用されます。

Mobile SDK 4.1 より前は、インデックスパスでは対応付け (辞書または連想配列) のみがサポートされていました。たとえば、a.b.c などのパスの場合、SmartStore では b と c 両方の対応付けが必要でした。これらの対応付けがない場合にパスを評価すると、SmartStore は何も返しませんでした。

Mobile SDK 4.1 以降では、インデックスパスに配列と対応付けを含めることができます。a.b.c の例では、b の値が配列の場合、SmartStore では c を定義する対応付けが含まれる配列が求められます。その後、SmartStore は b 配列の対応付けで見つかった c キーの値が含まれる配列を返します。

 **メモ:** JSON1 インデックス指定で配列をトラバースするインデックスパスは使用できません。

 **例:** 次の表に、a.b.c パスのさまざまな例と SmartStore クエリによって返される値を示します。

| 説明                                                    | スープ要素の例                                                                                                                                                  | パス a.b.c の値                                  |
|-------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|
| 配列なし                                                  | <pre>{<br/>  "a": {<br/>    "b": { "c": 1 }<br/>  }<br/>}</pre>                                                                                          | 1                                            |
| c は配列 (内部ノード) を示します。                                  | <pre>{<br/>  "a": {<br/>    "b": {<br/>      "c": [1, 2, 3]<br/>    }<br/>  }<br/>}</pre>                                                                | <pre>[<br/>  1,<br/>  2,<br/>  3<br/>]</pre> |
| b は対応付けの配列を示します。一部の対応付けには c キーが含まれます。その他の対応付けは無視されます。 | <pre>{<br/>  "a": {<br/>    "b": [<br/>      {<br/>        "c": 1<br/>      },<br/>      {<br/>        "c": 2<br/>      },<br/>    ]<br/>  }<br/>}</pre> | <pre>[<br/>  1,<br/>  2<br/>]</pre>          |

| 説明                                                                                           | スーブ要素の例                                                                                                                                                                                                                             | パス a.b.c の値                                |
|----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
|                                                                                              | <pre> {   {     "no-c":3   } } </pre>                                                                                                                                                                                               |                                            |
| <p>a は対応付けの配列を示します。一部の対応付けには、b がキーを含む対応付けを示すものがあります。b が対応付けの配列を示すものもあります。c キーの値のみが返されます。</p> | <pre> {   "a": [     {       "b": {         "c": 0       }     },     {       "b": [         {           "c": 1         },         {           "c": 2         },         {           "no-c": 3         }       ]     }   ] } </pre> | <pre> [   0,   [     1,     2   ] ] </pre> |

## スーブの入力

オフラインアクセス用に Salesforce レコードをスーブに追加するには、REST API と SmartStore API を組み合わせて使用します。

スーブを登録する場合、データを待機しているメモリ内に空の名前付き構造を作成します。通常、Salesforce 組織のデータを使用してスーブを初期化します。Salesforce データを取得するには、Mobile SDK の標準 REST 要求メカニズムを使用します。REST の成功応答が返されたら、応答オブジェクトのデータを抽出して、スーブに更新/挿入します。

## ハイブリッドアプリケーション

ハイブリッドアプリケーションは、force.js ライブラリで定義された SmartStore 関数を使用します。この例では、[Fetch Contacts (取引先責任者の取得)] ボタンのクリックハンドラーが force.query() をコールし、単純な SOQL クエリ ("SELECT Name, Id FROM Contact") を Salesforce に送信します。このコールにより、REST 応答

を受信するコールバック関数として `onSuccessSfdcContacts(response)` が指定されます。

`onSuccessSfdcContacts(response)` 関数は、`response` で返されたレコードを反復処理し、UI コントロールに Salesforce 値を入力します。最後に、応答のすべてのレコードをサンプルスープに更新/挿入します。

```
force.query("SELECT Name,Id FROM Contact",
 onSuccessSfdcContacts, onErrorSfdc); var sfSmartstore = function() {
 return cordova.require("com.salesforce.plugin.smartstore");};
function onSuccessSfdcContacts(response) {
 logToConsole() ("onSuccessSfdcContacts: received " +
 response.totalSize + " contacts");
 var entries = [];

 response.records.forEach(function(contact, i) {
 entries.push(contact);
 });

 if (entries.length > 0) {
 sfSmartstore().upsertSoupEntries(CONTACTS_SOUP_NAME,
 entries,
 function(items) {
 var statusTxt = "upserted: " + items.length +
 " contacts";
 logToConsole() (statusTxt);
 },
 onErrorUpsert);
 }
}

function onErrorSfdc(param) {
 logToConsole() ("onErrorSfdc: " + param);
}function onErrorUpsert(param) {
 logToConsole() ("onErrorUpsert: " + param);
}
```

## iOS ネイティブアプリケーション

iOS ネイティブアプリケーションは、REST API のやりとりに SFRestAPI プロトコルを使用します。次のコードは、SQL クエリ `SELECT Name, Id, OwnerId FROM Account` の REST 要求を作成して送信します。要求に成功すると、Salesforce は REST 応答を `requestForQuery:send:delegate:` 代理メソッドに送信します。応答が解析され、返された各レコードが SmartStore スープに更新/挿入されます。

```
- (void)requestAccounts
{
 SFRestRequest *request = [[SFRestAPI sharedInstance]
 requestForQuery:@"SELECT Name, Id, OwnerId FROM Account"];
 [[SFRestAPI sharedInstance] send:request delegate:self];
}

//SFRestAPI protocol for successful response
- (void)request:(SFRestRequest *)request didLoadResponse:(id)dataResponse
{
 NSArray *records = dataResponse[@"records"];
 if (nil != records) {
```

```

 for (int i = 0; i < records.count; i++) {
 [self.store upsertEntries:@[records[i]]
 toSoup:kAccountSoupName];
 }
 }
}

```

## Android ネイティブアプリケーション

通常、Android ネイティブアプリケーションは、`RestClient.sendAsync()` メソッドと `AsyncRequestCallback` インターフェースの匿名インライン定義を組み合わせて使用し、REST API のやりとりを行います。次のコードは、SOQL クエリ `SELECT Name, Id, OwnerId FROM Account` の REST 要求を作成して送信します。要求に成功すると、Salesforce は REST 応答を指定の `AsyncRequestCallback.onSuccess()` コールバックメソッドに送信します。応答が解析され、返された各レコードが SmartStore スープに更新/挿入されます。

```

private void sendRequest(String soql, final String obj)
throws UnsupportedOperationException {
 final RestRequest restRequest =
 RestRequest.getRequestForQuery(
 getString(R.string.api_version),
 "SELECT Name, Id, OwnerId FROM Account", "Account");
 client.sendAsync(restRequest, new AsyncRequestCallback() {
 @Override
 public void onSuccess(RestRequest request,
 RestResponse result) {
 // Consume before going back to main thread
 // Not required if you don't do main (UI) thread tasks here
 result.consumeQuietly();
 runOnUiThread(new Runnable() {
 @Override
 public void run() {
 // Network component doesn't report app layer status.
 // Use the Mobile SDK RestResponse.isSuccess() method to check
 // whether the REST request itself succeeded.
 if (result.isSuccess()) {
 try {
 final JSONArray records =
 result.asJSONObject().getJSONArray("records");
 insertAccounts(records);
 } catch (Exception e) {
 onError(e);
 } finally {
 Toast.makeText(MainActivity.this,
 "Records ready for offline access.",
 Toast.LENGTH_SHORT).show();
 }
 }
 }
 });
 }

 @Override
 public void onError(Exception e) {

```

```

 // You might want to log the error
 // or show it to the user
 }
});
}

/**
 * Inserts accounts into the accounts soup.
 *
 * @param accounts Accounts.
 */
public void insertAccounts(JSONArray accounts) {
 try {
 if (accounts != null) {
 for (int i = 0; i < accounts.length(); i++) {
 if (accounts[i] != null) {
 try {
 smartStore.upsert(
 ACCOUNTS_SOUP, accounts[i]);
 } catch (JSONException exc) {
 Log.e(TAG,
 "Error occurred while attempting "
 + "to insert account. Please verify "
 + "validity of JSON data set.");
 }
 }
 }
 }
 } catch (JSONException e) {
 Log.e(TAG, "Error occurred while attempting to "
 + "insert accounts. Please verify validity "
 + "of JSON data set.");
 }
}
}

```

## スープからのデータの取得

SmartStore には、クエリ文字列を作成する一連のヘルパーメソッドが備えられています。

スープからデータを取得するため、SmartStore ではクエリ指定を作成するヘルパー関数を使用できます。クエリ指定はインデックス指定と似ていますが、クエリのタイプとそのパラメーターに関するより多くの情報が含まれています。クエリビルダーメソッドは、次のクエリを実行できる指定を生成します。

- すべて (「all」 クエリ)
- Smart SQL の使用
- キーの完全一致 (「exact」 クエリ)
- 特定のパスでの全文検索 (「match」 クエリ)
- 値の範囲 (「range」 クエリ)
- ワイルドカード一致 (「like」 クエリ)



レコードセットを照会するには、指定に適したクエリ指定ファクトリーメソッドをコールします。必要に応じて、次の表に示すインデックス項目、並び替え順、および絞り込みに使用する他のメタデータも定義できます。

| パラメーター                                                    | 説明                                                                                                                                                                                                                                                                                                                                                                                           |
|-----------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>selectPaths</code> または <code>withSelectPaths</code> | (JavaScript では省略可能) 指定した項目のリストのみにクエリ範囲を絞り込みます。「 <a href="#">項目のサブセットを返すクエリの絞り込み</a> 」を参照してください。                                                                                                                                                                                                                                                                                              |
| <code>indexPath</code> または <code>path</code>              | 名前、取引先番号、日付などの検索対象を説明します。                                                                                                                                                                                                                                                                                                                                                                    |
| <code>beginKey</code>                                     | (JavaScript では省略可能) クエリ範囲の開始を定義するために使用されます。                                                                                                                                                                                                                                                                                                                                                  |
| <code>endKey</code>                                       | (JavaScript では省略可能) クエリ範囲の終了を定義するために使用されます。                                                                                                                                                                                                                                                                                                                                                  |
| <code>matchKey</code>                                     | (JavaScript では省略可能) <code>exact</code> または <code>match</code> クエリの検索文字列を指定するために使用されます。                                                                                                                                                                                                                                                                                                       |
| <code>orderPath</code>                                    | (JavaScript では省略可能。デフォルトで <code>path</code> パラメーターの値が設定されます) <code>exact</code> 、 <code>range</code> 、および <code>like</code> クエリで、結果セットの並び替えに使用されるインデックス付きパス項目を指定します。並べ替えなしで照会するには、このパラメーターを <code>null</code> 値に設定します。<br><br> <b>メモ:</b> Mobile SDK バージョン 3.2 以前では、クエリで指定されたインデックス付きパス項目のすべてのクエリが並び替えられます。 |
| <code>order</code>                                        | (JavaScript では省略可能) <ul style="list-style-type: none"> <li>JavaScript: 「ascending」 (デフォルト) または 「descending」 のいずれかです。</li> <li>iOS: <code>kSFSoupQuerySortOrderAscending</code> または <code>kSFSoupQuerySortOrderDescending</code> のいずれかです。</li> <li>Android: <code>Order.ascending</code> または <code>Order.descending</code> のいずれかです。</li> </ul>                                                |
| <code>pageSize</code>                                     | (JavaScript では省略可能。指定されていない場合、ネイティブプラグインによって、結果の <code>Cursor.pageSize</code> の最適値が計算されます) 結果の各ページに返すレコード数。                                                                                                                                                                                                                                                                                  |

たとえば、次の `buildRangeQuerySpec()` JavaScript コールについて考えます。

```
navigator.smartstore.buildRangeQuerySpec(
 "name", "Aardvark", "Zoroastrian", "ascending", 10, "name");
```

このコールは、Aardvark と Zoroastrian の間の名前が含まれるエントリを検索し、`name` 項目を昇順で並び替える `range` クエリ指定を作成します。

```
{
 "querySpec": {
 "queryType": "range",
 "indexPath": "name",
 "beginKey": "Aardvark",
 "endKey": "Zoroastrian",
```

```

 "orderPath": "name",
 "order": "ascending",
 "pageSize": 10
 }
}

```

JavaScript `build*` 関数では、関数コールの最後でのみ省略可能なパラメーターを省略できます。1つ以上のパラメーターをスキップしてから、スキップする各オプションのダミー値または `null` 値を入力せずに次のパラメーターを指定することはできません。たとえば、次のコールを使用できます。

- `buildAllQuerySpec(indexPath)`
- `buildAllQuerySpec(indexPath, order)`
- `buildAllQuerySpec(indexPath, order, pageSize)`
- `buildAllQuerySpec(indexPath, order, pageSize, selectPaths)`

ただし、次のコールは `order` パラメーターが省略されているため、使用できません。

```
buildAllQuerySpec(indexPath, pageSize)
```

-  **メモ:** すべてのパラメーター化されたクエリは1つの述語の検索です。結合をサポートするのはスマート SQL クエリのみです。

## すべてのクエリ

スープ内のすべてをトラバースします。

ページサイズについての詳細は、「[クエリ結果の処理](#)」を参照してください。

-  **メモ:** 基本的には、画面に表示するエントリ数に合わせて `pageSize` を設定します。円滑なスクロール表示を行うには、表示するエントリ数の2倍または3倍に値を増やします。

### JavaScript:

`buildAllQuerySpec(indexPath, order, pageSize, selectPaths)` は、スープ内のすべてのエントリを順不同で返します。`order` および `pageSize` は省略可能で、デフォルトはそれぞれ「ascending」および10です。`selectPaths` 引数も省略可能です。

### iOS ネイティブ:

```

+ (SFQuerySpec*) newAllQuerySpec:(NSString*)soupName
 withPath:(NSString*)path
 withOrder:(SFSoupQuerySortOrder)order
 withPageSize:(NSUInteger)pageSize;

+ (SFQuerySpec*) newAllQuerySpec:(NSString*)soupName
 withSelectPaths:(NSArray*)selectPaths
 withOrderPath:(NSString*)orderPath
 withOrder:(SFSoupQuerySortOrder)order
 withPageSize:(NSUInteger)pageSize;

```

### Android ネイティブ:

```

public static QuerySpec buildAllQuerySpec(
 String soupName,
 String path,

```

```

 Order order,
 int pageSize)

public static QuerySpec buildAllQuerySpec(
 String soupName,
 String[] selectPaths,
 String orderPath,
 Order order,
 int pageSize);

```

## スマート SQL SELECT ステートメントを含むクエリ

特定の Smart SQL ステートメントで指定されたクエリを実行します。この関数では独自の SELECT ステートメントを指定するため、他のクエリファクトリー関数に比べて柔軟性が高まります。「[スマート SQL クエリ](#)」を参照してください。

次のサンプルコードは、SQL COUNT 関数をコールする Smart SQL クエリです。

### JavaScript:

```

var querySpec =
 navigator.smartstore.buildSmartQuerySpec(
 "select count(*) from {employees}", 1);

navigator.smartstore.runSmartQuery(querySpec, function(cursor) {
 // result should be [[n]] if there are n employees
});

```

JavaScript では、pageSize は省略可能で、デフォルトは 10 です。

### iOS ネイティブ:

Mobile SDK 8.0 以降では、ネイティブ Swift SmartStore 拡張により、2つの方法で Smart SQL クエリを実行できます。

### Swift

#### iOS 12.2 以降の場合

```
public func query(_ smartSql: String) -> Result<[Any], SmartStoreError>
```

#### iOS 13.0 以降の場合。Combine Publisher を使用

```
public func publisher(for smartSql: String) -> Future<[Any], SmartStoreError>
```

### Objective-C

```

SFQuerySpec* querySpec =
 [SFQuerySpec
 newSmartQuerySpec:@"select count(*) from {employees}"
 withPageSize:1];
NSArray* result = [_store queryWithQuerySpec:querySpec pageIndex:0 error:nil];
// result should be [[n]] if there are n employees

```

### Android ネイティブ:

```

try {
 JSONArray result =

```

```

 store.query(QuerySpec.buildSmartQuerySpec(
 "select count(*) from {Accounts}", 1), 0);
 // result should be [[n]] if there are n employees
 Log.println(Log.INFO, "REST Success!", "\nFound " +
 result.getString(0) + " accounts.");
 } catch (JSONException e) {
 Log.e(TAG, "Error occurred while counting the number of account records. "
 + "Please verify validity of JSON data set.");
 }
}

```

## Exact によるクエリ

indexPath 値の特定の matchKey と完全に一致するエントリを見つけます。特定の ID の子エンティティを検索する場合に、このメソッドを使用します。たとえば、Status 別の商談を検索できます。

### JavaScript:

JavaScript では、order パラメーターを「ascending」または「descending」に設定できます。order、pageSize、および orderPath は省略可能で、デフォルトはそれぞれ「ascending」、10、および path 引数です。selectPaths 引数も省略可能です。

```

navigator.smartstore.buildExactQuerySpec(
 path, matchKey, pageSize, order, orderPath, selectPaths)

```

次の JavaScript コードは、ID によって子を取得します。

```

var querySpec = navigator.smartstore.buildExactQuerySpec(
 "sfdcId",
 "some-sfdc-id");
navigator.smartstore.querySoup("Catalogs",
 querySpec, function(cursor) {
 // we expect the catalog to be in:
 // cursor.currentPageOrderedEntries[0]
});

```

次の JavaScript コードは、親 ID によって子を取得します。

```

var querySpec = navigator.smartstore.buildExactQuerySpec("parentSfdcId", "some-sfdc-id");
navigator.smartstore.querySoup("Catalogs", querySpec, function(cursor) {});

```

### iOS ネイティブ:

iOS では、order パラメーターを kSFSoupQuerySortOrderAscending または kSFSoupQuerySortOrderDescending のいずれかに設定できます。クエリ範囲を特定の項目に絞り込むには、2 つ目の形式を使用し、withSelectPaths パラメーターで項目名の配列を渡します。

```

+ (SFQuerySpec*) newExactQuerySpec:(NSString*) soupName
 withPath:(NSString*) path
 withMatchKey:(NSString*) matchKey
 withOrderPath:(NSString*) orderPath
 withOrder:(SFSoupQuerySortOrder) order
 withPageSize:(NSUInteger) pageSize;

+ (SFQuerySpec*) newExactQuerySpec:(NSString*) soupName
 withSelectPaths:(NSArray*) selectPaths

```

```

 withPath: (NSString*)path
 withMatchKey: (NSString*)matchKey
 withOrderPath: (NSString*)orderPath
 withOrder: (SFSoupQuerySortOrder)order
 withPageSize: (NSUInteger)pageSize;

```

#### Android ネイティブ:

Android では、order パラメーターを Order.ascending または Order.descending のいずれかに設定できます。クエリ範囲を特定の項目に絞り込むには、2つ目の形式を使用し、selectPaths パラメーターで項目名の配列を渡します。

```

public static QuerySpec buildExactQuerySpec(
 String soupName, String path, String exactMatchKey,
 String orderPath, Order order, int pageSize)

public static QuerySpec buildExactQuerySpec(
 String soupName, String[] selectPaths, String path,
 String exactMatchKey, String orderPath,
 Order order, int pageSize);

```

## Match によるクエリ

indexPath 値の matchKey で全文検索クエリと完全に一致するエントリを見つけます。[「全文検索クエリの使用」](#)を参照してください。

#### JavaScript:

JavaScript では、order パラメーターを「ascending」または「descending」に設定できます。order、pageSize、および orderPath は省略可能で、デフォルトはそれぞれ「ascending」、10、および path 引数です。selectPaths 引数も省略可能です。

```

navigator.smartstore.buildMatchQuerySpec(
 path, matchKey, order, pageSize, orderPath, selectPaths)

```

#### iOS ネイティブ:

iOS では、order パラメーターを kSFSoupQuerySortOrderAscending または kSFSoupQuerySortOrderDescending のいずれかに設定できます。クエリ範囲を特定の項目に絞り込むには、2つ目の形式を使用し、withSelectPaths パラメーターで項目名の配列を渡します。

```

+ (SFQuerySpec*) newMatchQuerySpec: (NSString*) soupName
 withPath: (NSString*) path
 withMatchKey: (NSString*) matchKey
 withOrderPath: (NSString*) orderPath
 withOrder: (SFSoupQuerySortOrder) order
 withPageSize: (NSUInteger) pageSize;

+ (SFQuerySpec*) newMatchQuerySpec: (NSString*) soupName
 withSelectPaths: (NSArray*) selectPaths
 withPath: (NSString*) path
 withMatchKey: (NSString*) matchKey
 withOrderPath: (NSString*) orderPath

```

```
withOrder: (SFSoupQuerySortOrder)order
withPageSize: (NSUInteger)pageSize;
```

#### Android ネイティブ:

Android では、`order` パラメーターを `Order.ascending` または `Order.descending` のいずれかに設定できます。クエリ範囲を特定の項目に絞り込むには、2つ目の形式を使用し、`selectPaths` パラメーターで項目名の配列を渡します。

```
public static QuerySpec buildMatchQuerySpec(
 String soupName, String path, String exactMatchKey,
 String orderPath, Order order, int pageSize)

public static QuerySpec buildMatchQuerySpec(
 String soupName, String[] selectPaths, String path,
 String matchKey, String orderPath, Order order,
 int pageSize)
```

## Range によるクエリ

`indexPath` の値が `beginKey` と `endKey` で定義された範囲内にあるエントリを見つけます。整数として保存された日付範囲など、数値範囲で検索する場合にこの関数を使用します。

`beginKey` と `endKey` に `null` 値を渡すと、範囲無期限の検索を実行できます。

- `indexPath` での項目が `beginKey` 以上のすべてのレコードを見つけるには、`null` 値を `endKey` に渡します。
- `indexPath` での項目が `endKey` 以下のすべてのレコードを見つけるには、`null` 値を `beginKey` に渡します。
- すべてを照会するには、`null` 値を `beginKey` と `endKey` の両方に渡します。

#### JavaScript:

JavaScript では、`order` パラメーターを「ascending」または「descending」に設定できます。`order`、`pageSize`、および `orderPath` は省略可能で、デフォルトはそれぞれ「ascending」、10、および `path` 引数です。`selectPaths` 引数も省略可能です。

```
navigator.smartstore.buildRangeQuerySpec(
 path, beginKey, endKey, order, pageSize, orderPath, selectPaths)
```

#### iOS ネイティブ:

iOS では、`order` パラメーターを `kSFSoupQuerySortOrderAscending` または `kSFSoupQuerySortOrderDescending` のいずれかに設定できます。クエリ範囲を特定の項目に絞り込むには、2つ目の形式を使用し、`withSelectPaths` パラメーターで項目名の配列を渡します。

```
+ (SFQuerySpec*) newRangeQuerySpec: (NSString*) soupName
 withPath: (NSString*) path
 withBeginKey: (NSString*) beginKey
 withEndKey: (NSString*) endKey
 withOrderPath: (NSString*) orderPath
 withOrder: (SFSoupQuerySortOrder) order
 withPageSize: (NSUInteger) pageSize;
```

```
+ (SFQuerySpec*) newRangeQuerySpec:(NSString*) soupName
 withSelectPaths:(NSArray*) selectPaths
 withPath:(NSString*) path
 withBeginKey:(NSString*) beginKey
 withEndKey:(NSString*) endKey
 withOrderPath:(NSString*) orderPath
 withOrder:(SFSoupQuerySortOrder) order
 withPageSize:(NSUInteger) pageSize;
```

#### Android ネイティブ:

Android では、order パラメーターを Order.ascending または Order.descending のいずれかに設定できます。クエリ範囲を特定の項目に絞るには、2つ目の形式を使用し、selectPaths パラメーターで項目名の配列を渡します。

```
public static QuerySpec buildRangeQuerySpec(
 String soupName, String path, String beginKey,
 String endKey, String orderPath, Order order, int pageSize)

public static QuerySpec buildRangeQuerySpec(
 String soupName, String[] selectPaths, String path,
 String beginKey, String endKey, String orderPath,
 Order order, int pageSize);
```

## Like によるクエリ

indexPath の値が指定の likeKey に似ているエントリを見つけます。次の構文例に示すように、部分一致を検索するには「%」ワイルドカードを使用します。

- キーワードで開始する用語を検索する場合は「foo%」
- キーワードで終了する用語を検索する場合は「%foo」
- indexPath 値のどこかにあるキーワードを検索する場合は「%foo%」

一般検索および名前の部分一致を行う場合に、この関数を使用します。全文クエリおよび大きいデータセットでの高速クエリには、「match」によるクエリメソッドを使用します。

 **メモ:** 「like」によるクエリは、クエリメソッドの中で最も低速です。

#### JavaScript:

JavaScript では、order パラメーターを「ascending」または「descending」に設定できます。order、pageSize、および orderPath は省略可能で、デフォルトはそれぞれ「ascending」、10、および path 引数です。selectPaths 引数も省略可能です。

```
navigator.smartstore.buildLikeQuerySpec(
 path, likeKey, order, pageSize, orderPath, selectPaths)
```

#### iOS ネイティブ:



iOS では、order パラメーターを `kSFSoupQuerySortOrderAscending` または `kSFSoupQuerySortOrderDescending` のいずれかに設定できます。クエリ範囲を特定の項目に絞り込むには、2つ目の形式を使用し、`withSelectPaths` パラメーターで項目名の配列を渡します。

```
+ (SFQuerySpec*) newLikeQuerySpec:(NSString*) soupName
 withPath:(NSString*) path
 withLikeKey:(NSString*) likeKey
 withOrderPath:(NSString*) orderPath
 withOrder:(SFSoupQuerySortOrder) order
 withPageSize:(NSUInteger) pageSize;

+ (SFQuerySpec*) newLikeQuerySpec:(NSString*) soupName
 withSelectPaths:(NSArray*) selectPaths
 withPath:(NSString*) path
 withLikeKey:(NSString*) likeKey
 withOrderPath:(NSString*) orderPath
 withOrder:(SFSoupQuerySortOrder) order
 withPageSize:(NSUInteger) pageSize;
```

#### Android ネイティブ:

Android では、order パラメーターを `Order.ascending` または `Order.descending` のいずれかに設定できます。クエリ範囲を特定の項目に絞り込むには、2つ目の形式を使用し、`selectPaths` パラメーターで項目名の配列を渡します。

```
public static QuerySpec buildLikeQuerySpec(
 String soupName, String path, String likeKey,
 String orderPath, Order order, int pageSize)

public static QuerySpec buildLikeQuerySpec(
 String soupName, String[] selectPaths,
 String path, String likeKey, String orderPath,
 Order order, int pageSize)
```

## クエリの実行

JavaScript では、クエリは非同期に実行され、成功コールバック関数にカーソルが返されるか、エラーコールバック関数にエラーが返されます。成功コールバックは、`function(cursor)` の形式を取ります。クエリ指定を `querySoup` メソッドに渡すには、`querySpec` パラメーターを使用します。

```
navigator.smartstore.querySoup(soupName, querySpec,
 successCallback, errorCallback);
```

## 項目のサブセットを返すクエリの絞り込み

SmartSQL クエリ指定では、SmartSQL ステートメントで項目を指定し、クエリが返す項目のリストを制限できます。他のタイプのクエリ指定では、`selectPaths` パラメーターを使用して同じ処理を行うことができます。この引数を使用する場合、メソッドはクエリを満たす各要素の配列を含む配列の配列を返します。各要素配列には、`selectPaths` で指定された項目のみが含まれます。このパラメーターは、「all」、「exact」、「match」、「range」、「like」クエリ指定で使用できます。

次に例を示します。次のような要素を含むスープがあるとします。

```
{ "_soupEntryId":1, "name":"abc", "status":"active", ... },
{ "_soupEntryId":2, "name":"abd", "status":"inactive", ... }, ...
```

LIKE キーとして「ab%」、パスとして name を使用する「like」クエリを実行しましょう。このクエリは、オブジェクトの配列を返します。各オブジェクトには要素全体が含まれます。

```
[{ "_soupEntryId":1, "name": "abc", "status":"active",... },
{ "_soupEntryId":2, "name":"abd", "status":"inactive",... },
...]
```

次に、選択されたパスとして \_soupEntryId と name を追加してクエリを絞り込みましょう。このクエリは、より効率的な \_soupEntryId および name 項目の値のみが含まれる配列の配列を返します。

```
[[1, "abc"], [2, "abd"], ...]
```

## 主キーによる個々のスープエントリの取得

すべてのスープエントリには、一意の内部 ID (スープ内のすべてのエントリを保持する内部テーブルにおける主キー) が自動的に与えられます。この ID 項目は、スープエントリで \_soupEntryId 項目として使用できます。

JavaScript で \_soupEntryId によってスープエントリを検索するには、retrieveSoupEntries 関数を使用します。この関数ではスープエントリを最も速く取得できますが、使用できるのは \_soupEntryId がわかっている場合のみです。

```
navigator.smartStore.retrieveSoupEntries(soupName, indexSpecs,
 successCallback, errorCallback)
```

返される順序は保証されません。また、削除されたエントリは結果の配列では返されません。

## スマート SQL クエリ

クエリを完全に制御したり、既存の SQL クエリを再利用したりするには、カスタム SmartStore クエリを定義します。

自由形式の SELECT ステートメント用に SmartStore でスマート SQL クエリ言語がサポートされます。スマート SQL クエリでは、スープおよびスープ項目の参照用に、標準 SQL SELECT の文法が他の記述子と組み合わせられます。これにより、結合の使用を含め、制御および柔軟性が最大限に高まります。スマート SQL では、標準 SQL SELECT 構造がすべてサポートされます。

Mobile SDK 9.1 の時点で、スマート SQL では、SELECT または WHERE 句で参照されるすべての項目のインデックスパスが不要になりました。ただし、例外として次の制限があります。

### スマート SQL の制限

- 廃止された外部ストレージ機能をスープで使用している場合、スマート SQL では、SELECT または WHERE 句で参照される項目のインデックスパスが依然として必要です。



**警告:** 外部ストレージは Mobile SDK 10.0 で廃止され、Mobile SDK 11.0 で削除されます。「[大きいスープ要素での外部ストレージの使用](#)」を参照してください。

- Smart SQL を使用して MATCH クエリを記述することはできません。たとえば、次のクエリは機能しません。  
`SELECT {soupName:_soup} FROM {soupName} WHERE {soupName:name} MATCH 'cat'`

## 構文

構文は標準 SQL SELECT の仕様と同じですが、次の点で違いがあります。

| 使用方法                     | 構文                                                    |
|--------------------------|-------------------------------------------------------|
| 列を指定する                   | <code>{&lt;soupName&gt;:&lt;path&gt;}</code>          |
| テーブルを指定する                | <code>{&lt;soupName&gt;}</code>                       |
| スープエントリの JSON 文字列全体を参照する | <code>{&lt;soupName&gt;:_soup}</code>                 |
| 内部スープエントリ ID を参照する       | <code>{&lt;soupName&gt;:_soupEntryId}</code>          |
| 最終更新日を参照する               | <code>{&lt;soupName&gt;:_soupLastModifiedDate}</code> |

## サンプルクエリ

Employees および Departments という名前の 2 つのスープについて考えます。Employees スープには、次の標準項目が含まれています。

- 名 (firstName)
- 姓 (lastName)
- 部署コード (deptCode)
- 従業員 ID (employeeId)
- マネージャー ID (managerId)

Departments スープには、次の項目が含まれています。

- 名 (name)
- 部署コード (deptCode)

次に、これらのスープを使用した基本的なスマート SQL クエリをいくつか示します。

```
select {employees:firstName}, {employees:lastName}
from {employees} order by {employees:lastName}

select {departments:name}
from {departments}
order by {departments:deptCode}
```

## 結合

スマート SQL では、ユーザーが結合を使用することもできます。次に例を示します。

```
select {departments:name}, {employees:firstName} || ' ' || {employees:lastName}
from {employees}, {departments}
where {departments:deptCode} = {employees:deptCode}
order by {departments:name}, {employees:lastName}
```

さらに、自己結合を行うこともできます。

```
select mgr.{employees:lastName}, e.{employees:lastName}
from {employees} as mgr, {employees} as e
where mgr.{employees:employeeId} = e.{employees:managerId}
```

 **メモ:** JSON1 インデックスで結合を行うには、若干拡張された構文が必要です。たとえば、

```
select {soup1:path1} from {soup1}, {soup2}
```

の代わりに、次を使用します。

```
select {soup1}.{soup1:path1} from {soup1}, {soup2}
```

## 集計関数

スマート SQL では、次のような集計関数の使用をサポートしています。

- COUNT
- SUM
- AVG

次に例を示します。

```
select {account:name},
 count({opportunity:name}),
 sum({opportunity:amount}),
 avg({opportunity:amount}),
 {account:id},
 {opportunity:accountid}
from {account},
 {opportunity}
where {account:id} = {opportunity:accountid}
group by {account:name}
```

## 全文検索クエリの使用

SmartStore で効率的かつ柔軟な検索を実行するには、全文クエリを使用します。全文クエリは「like」クエリと比べて、大きいデータセットを処理する場合にパフォーマンス上の大きな利点があります。

Mobile SDK 3.3 以降、SmartStore では全文検索がサポートされます。全文検索は、インターネット検索エンジンで Web 上にあるドキュメントを照合するテクノロジーです。

## 全文検索について

全文検索は次のように機能します。まず、顧客が1つ以上の用語を入力します。必要に応じて、顧客はバイナリ演算子を使用して用語を結合したり、グループ化して語句にしたりできます。全文検索エンジンが、指定された演算子とグルーピングを適用して該当用語を評価します。検索エンジンが評価後の検索パラメーターを使用して一致するドキュメントを検索するか、SmartStore の場合は一致するスープ要素を検索します。全文検索をサポートするため、SmartStore にはスープ項目を定義するための全文インデックス指定、およびスープ項目に対するクエリを実行するためのクエリ指定が備えられています。

全文クエリまたは「match」クエリは、「like」クエリよりも効率的です。「like」クエリはすべてのキーの完全なインデックススキャンが必要で、実行時間は検索行数に比例します。「match」クエリは、インデックス内で特定の用語を検索し、関連付けられたレコード ID を返します。全文検索の最適化は、1000 未満のレコードではあまり効果はありませんが、そのしきい値を超えると、レコード数が増加しても実行時間はほぼ一定になります。何万件ものレコードを検索する場合、「match」クエリは「like」クエリよりも 10 ～ 100 倍高速です。

全文項目と全文クエリを使用する場合は、次の点に留意してください。

- 全文インデックス項目を使用した挿入は、通常の挿入よりも時間がかかります。
- Smart SQL ステートメントでは MATCH クエリを実行できません。たとえば、次のクエリはサポートされていません。

```
SELECT {soupName:_soup} FROM {soupName} WHERE {soupName:name} MATCH 'cat'
```

代わりに、「match」クエリ指定を使用します。

## 全文検索による最新情報の把握

Mobile SDK 4.2 では、SmartStore の全文検索バージョンが FTS4 から FTS5 に更新されます。このアップグレードによって、Mobile SDK は全文インデックス指定を利用できます。

アプリケーションを Mobile SDK 4.1 から 4.2 にアップグレードした場合、既存の FTS4 仮想テーブルは変更されません。一方、アップグレード後に作成する新規スープでは FTS5 仮想テーブルが使用されます。これらのスープはすべてシームレスに機能しますが、従来のスープをアップグレードすることもできます。alterSoup をコールし、元のインデックス指定セットに渡すだけです。このコールは、FTS5 を使用して、全文インデックス指定をサポートする仮想テーブルを再作成します。

FTS4 と FTS5 の比較については、[www.sqlite.org/fts5.html](http://www.sqlite.org/fts5.html) の「Appendix A」を参照してください。

このセクションの内容:

### 全文検索インデックス指定

全文検索を使用するには、1つ以上の全文インデックス付きパスを使用してスープを登録します。SmartStore には、インデックス項目を設計するための full\_text インデックス指定が備えられています。

### 全文クエリ指定

全文クエリを実行するには、プラットフォームの照合クエリメソッドを使用して SmartStore の「match」クエリ指定を作成します。matchKey 引数の場合、全文検索クエリを指定します。

### 全文クエリ構文

Mobile SDK の全文クエリは、SQLite の拡張クエリ構文を使用します。この構文では、論理演算子を使用して検索を絞り込むことができます。

## 全文検索インデックス指定

全文検索を使用するには、1つ以上の全文インデックス付きパスを使用してスープを登録します。SmartStoreには、インデックス項目を設計するための `full_text` インデックス指定が備えられています。

全文インデックスを含むパスを定義した場合、そのパスを全文以外のクエリに使用することもできます。これらの他のタイプのクエリ(「all」、「exact」、「like」、「range」、「smart」クエリ)は、全文インデックス付き項目を文字列インデックス付き項目として解釈します。

次の例に、全文インデックス指定をインスタンス化する方法を示します。

 例: iOS:

```
[[SFSoupIndex alloc]
 initWithDictionary:@{kSoupIndexPath: @"some_path",
 kSoupIndexType: kSoupIndexTypeFullText}]
```

Android:

```
new IndexSpec("some_path", Type.full_text)
```

JavaScript:

```
new navigator.smartstore.SoupIndexSpec("some_path", "full_text")
```

## 全文クエリ指定

全文クエリを実行するには、プラットフォームの照合クエリメソッドを使用して SmartStore の「match」クエリ指定を作成します。matchKey 引数の場合、全文検索クエリを指定します。

次のメソッドを使用して、全文クエリ指定を作成します。

iOS:

```
+ (SFQuerySpec*) newMatchQuerySpec:(NSString*) soupName
 withPath:(NSString*) path
 withMatchKey:(NSString*) matchKey
 withOrderPath:(NSString*) orderPath
 withOrder:(SFSoupQuerySortOrder) order
 withPageSize:(NSUInteger) pageSize;

+ (SFQuerySpec*) newMatchQuerySpec:(NSString*) soupName
 withSelectPaths:(NSArray*) selectPaths
 withPath:(NSString*) path
 withMatchKey:(NSString*) matchKey
 withOrderPath:(NSString*) orderPath
 withOrder:(SFSoupQuerySortOrder) order
 withPageSize:(NSUInteger) pageSize;
```

Android:

```
public static QuerySpec buildMatchQuerySpec(
 String soupName, String path, String exactMatchKey,
 String orderPath, Order order, int pageSize)
```

```
public static QuerySpec buildMatchQuerySpec(
 String soupName, String[] selectPaths, String path,
 String matchKey, String orderPath, Order order,
 int pageSize)
```

JavaScript:

```
smartstore.buildMatchQuerySpec(
 path, matchKey, order, pageSize, orderPath, selectPaths)
```

全文クエリ構文

Mobile SDKの全文クエリは、SQLiteの拡張クエリ構文を使用します。この構文では、論理演算子を使用して検索を絞り込むことができます。

次の表に、Mobile SDKのクエリでサポートされる構文オプションを示します。表の後に、さまざまなクエリスタイルとサンプル出力の例をキー付きで示します。詳細は、[sqlite.org](https://www.sqlite.org)のセクション 3.1 「Full-text Index Queries」および 3.2 「Set Operations Using The Enhanced Query Syntax」を参照してください。

| クエリオプション                       | SmartStore の動作                                                                                                                                                                                          | 関連する例       |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| 1つ以上の全文インデックス付きパスを指定する         | 定義したパスにある値のみとの照合を実行します。                                                                                                                                                                                 | g、h、i、j、k   |
| パスを null 値に設定する                | すべての全文インデックス付きパスとの照合を実行します。<br> <b>メモ:</b> パスが null の場合でも、matchKey 引数で対象項目を指定できます。<br>{soupName:path}:term 形式を使用します。 | a、b、c、d、e、f |
| 演算子やグルーピングを使用せずに複数の用語を指定する     | 用語の間に「AND」があるとみなされます。                                                                                                                                                                                   | b、h         |
| 用語の末尾にアスタリスクを追加する              | クエリ用語で開始する単語が含まれる行が一致します。                                                                                                                                                                               | d、j         |
| 用語の間に「OR」を使用する                 | 一方または両方の用語を検索します。                                                                                                                                                                                       | c、i         |
| 用語の前に単項「NOT」演算子を使用する           | その用語が含まれる行を無視します。                                                                                                                                                                                       | e、f、k       |
| 複数の用語を二重引用符(" ")で囲んで語句検索を指定する。 | 1つ以上の全文インデックス付き項目に語句全体が含まれるスープ要素を返します。                                                                                                                                                                  |             |





例: この例では、「animals」という名前のスコープに次のレコードが含まれています。name 項目と color 項目は full\_text としてインデックス付けされています。

```
{ "id": 1, "name": "cat", "color": "black" }
{ "id": 2, "name": "cat", "color": "white" }
{ "id": 3, "name": "dog", "color": "black" }
{ "id": 4, "name": "dog", "color": "brown" }
{ "id": 5, "name": "dog", "color": "white" }
```

表 4: クエリ構文の例

| 例  | パス      | 一致キー                            | 選択レコード                                                | 返されるレコード |
|----|---------|---------------------------------|-------------------------------------------------------|----------|
| a. | null    | "black"                         | 全文インデックス付き項目に単語「black」が含まれるレコード                       | 1、3      |
| b. | null    | "black cat"                     | 全文インデックス付き項目に単語「black」および「cat」が含まれるレコード               | 1        |
| c. | null    | "black OR cat"                  | 全文インデックス付き項目に単語「black」または「cat」のいずれかが含まれるレコード          | 1、2、3    |
| d. | null    | "b*"                            | 全文インデックス付き項目に「b」で始まる単語が含まれるレコード                       | 1、3      |
| e. | null    | "black NOT cat"                 | 全文インデックス付き項目に単語「black」が含まれ、単語「cat」は含まれないレコード          | 3        |
| f. | null    | "{animals:color}:black NOT cat" | color 項目に単語「black」が含まれ、全文インデックス付き項目に単語「cat」が含まれないレコード | 3        |
| g. | "color" | "black"                         | color 項目に単語「black」が含まれるレコード                           | 1、3      |
| h. | "color" | "black cat"                     | color 項目に単語「black」および「cat」が含まれるレコード                   | レコードなし   |
| i. | "color" | "black OR cat"                  | color 項目に単語「black」または「cat」のいずれかが含まれるレコード              | 1、3      |

| 例  | パス      | 一致キー            | 選択レコード                                   | 返されるレコード |
|----|---------|-----------------|------------------------------------------|----------|
| j. | "color" | "b*"            | color 項目に「b」で始まる単語が含まれるレコード              | 1、3      |
| k. | "color" | "black NOT cat" | color 項目に単語「black」が含まれ、単語「cat」は含まれないレコード | 1、3      |

## クエリ結果の処理

Mobile SDK には、各プラットフォームで効率的、柔軟かつ動的にクエリ結果にアクセスできるメカニズムがあります。

多くの場合、クエリで返される結果セットはメモリにすべて一括で読み込むには大きすぎます。この場合、Mobile SDK は指定されたサイズに基づいて最初に結果の小さいサブセット (1 ページ) を返します。その後その他の結果ページを取得し、結果セットの前後に移動できます。

### JavaScript:

JavaScript でクエリを実行すると、SmartStore はクエリ結果をページ処理できるカーソルオブジェクトを返します。コードを使用して、カーソルが置かれるページを前後に移動できます。カーソルが置かれるページを移動するには、次の関数を使用します。

- `navigator.smartstore.moveCursorToPageIndex(cursor, newPageIndex, successCallback, errorCallback)` — カーソルを特定のページインデックスに移動します。0 は最初のページ、`totalPages - 1` は最後のページです。
- `navigator.smartstore.moveCursorToNextPage(cursor, successCallback, errorCallback)` — 次のエントリページに移動します (存在する場合)。
- `navigator.smartstore.moveCursorToPreviousPage(cursor, successCallback, errorCallback)` — 前のエントリページに移動します (存在する場合)。
- `navigator.smartstore.closeCursor(cursor, successCallback, errorCallback)` — 終了したら、カーソルを閉じます。

### メモ:

- `successCallback` 関数には、1 つの引数 (更新後のカーソル) を指定します。
- カーソルはデータの静的スナップショットではなく、動的です。カーソルで保持されるデータは、元のクエリと結果セットでの現在の位置のみです。カーソルを移動すると、クエリが再度実行されます。カーソルのページ移動中にスプーを変更すると、カーソルでその変更が表示されます。新しく作成されたスプーエントリにアクセスすることもできます (元のクエリの結果が返されている場合)。

### iOS ネイティブ:

内部的には、iOS ネイティブアプリケーションはサードパーティの `FMResultSet` クラスを使用してクエリ結果を取得します。SmartStore クエリ指定メソッドをコールするときに、`pageSize` パラメーターを使用して各コールから戻されるデータ量を制御します。結果のページをトラバースするには、同じクエリ指定オブジェク

トで `SFSmartStore` クラスの `queryWithQuerySpec:pageIndex:withDB:` または `queryWithQuerySpec:pageIndex:error:` メソッドを開始値0の `pageIndex` 引数を増減しながら繰り返しコールします。

#### Android ネイティブ:

内部的には、Android ネイティブアプリケーションはカーソルの操作に `android.database.Cursor` インターフェースを使用します。SmartStore クエリ指定メソッドをコールするときに、`pageSize` パラメーターを使用して各コールから戻されるデータ量を制御します。結果のページをトラバースするには、同じクエリ指定オブジェクトで `SmartStore.query()` メソッドを開始値0の `pageIndex` 引数を増減しながら繰り返しコールします。

## データの挿入、更新、および更新/挿入

SmartStore では、エントリを追跡して外部サーバーとスープを同期できる標準項目が定義されます。

### システム項目: `_soupEntryId` and `_soupLastModifiedDate`

スープエントリの挿入、更新、および削除アクションを追跡するために、SmartStore は各エントリに次の項目を追加します。

- `_soupEntryId` — この項目は、特定のスープのテーブルに含まれるスープエントリの主キーです。
- `_soupLastModifiedDate`、`_soupCreatedDate` — 1/1/1970 からのミリ秒数です。
  - 日付値を JavaScript 日付に変換するには、`new Date(entry._soupLastModifiedDate)` を使用します。
  - 日付を 1/1/1970 を起点とした対応するミリ秒数に変換するには、`date.getTime()` を使用します。

スープエントリを挿入または更新すると、これらの項目が SmartStore で自動的に設定されます。特定のエントリを削除または取得するときは、`_soupEntryId` でそのエントリを参照できます。

Mobile SDK 4.2 以降、SmartStore では `_soupLastModifiedDate` および `_soupCreatedDate` 項目のインデックスが作成されます。このインデックスにより、これらの項目を使用するクエリのパフォーマンスが向上します。古いスープでは、`_soupLastModifiedDate` および `_soupCreatedDate` 項目は存在しますが、インデックスは作成されません。従来のスープでこれらのインデックスを作成するには、`alterSoup` をコールし、元のインデックス指定セットを渡すだけです。

## 更新/挿入について

スープエントリを挿入または更新するには、`upsert` メソッドを使用します(挿入と更新のどちらのアクションが適切かは SmartStore が判断します)。

更新/挿入予定のエントリに `_soupEntryId` がすでに設定されている場合は、SmartStore がその ID と一致するスープエントリを更新します。更新/挿入エントリに `_soupEntryId` スロットが存在しないか、`_soupEntryId` が既存のスープエントリと一致しない場合は、SmartStore がエントリを挿入し、その `_soupEntryId` を上書きします。

## 外部 ID を使用した更新/挿入

スープエントリが外部システムのデータを反映している場合、通常は外部の主キー ID を使用してそのエントリを参照します。そのため、外部 ID を使用した更新/挿入が SmartStore でサポートされています。更新/挿入を実行する場合、任意のインデックス項目を外部 ID 項目として指定できます。SmartStore では、指定された項目と同じ値を持つ既存のスープエントリが検索され、次の結果が得られます。

- 同じ値を持つ項目が存在しない場合、SmartStore はスープエントリを作成します。
- 外部 ID 項目が存在する場合、SmartStore は一致する外部 ID 値を使用してエントリを更新します。
- 外部 ID と一致する項目が複数存在すると、SmartStore はエラーを返します。

エントリをローカルで作成するには、新しいエントリをサーバーにアップロードするときに照会できる値に、外部 ID 項目を設定します。

外部データを使用してスープを更新する場合は、常に外部 ID を使用します。これにより、同じリモートレコードに対してスープエントリが重複しないことが保証されます。

SmartStore では、オブジェクト間のリレーションを追跡することもできます。たとえば、サーバーにはまだ存在しないカタログに属する商品をオフラインで作成するとします。parentSoupEntryId 項目を使用して、商品のカタログエントリとのリレーションを取得できます。サーバーにカタログが存在するようになったら、ローカル商品レコードの parentExternalId 項目を更新して外部リレーションを取得できます。

## 更新/挿入メソッド

### JavaScript:

cordova.force.js ライブラリには、次の 2 つの JavaScript 更新/挿入関数があり、それぞれに 1 つのオーバーロードがあります。

```
navigator.smartStore.upsertSoupEntries(isGlobalStore, soupName,
 entries[], successCallback, errorCallback)
navigator.smartStore.upsertSoupEntries(storeConfig, soupName,
 entries[], successCallback, errorCallback)
```

```
navigator.smartStore.upsertSoupEntriesWithExternalId(isGlobalStore, soupName,
 entries[], externalPathId, successCallback, errorCallback)
navigator.smartStore.upsertSoupEntriesWithExternalId(storeConfig, soupName,
 entries[], externalPathId, successCallback, errorCallback)
```

ローカルデータのみを更新/挿入するには、最初の upsert() 関数を使用します。外部サーバーからデータを更新/挿入するには、externalPathId パラメーターがサポートされる 2 番目の関数を使用します。

### iOS ネイティブ:

iOS SFSSmartStore クラスには、2 つの更新/挿入用インスタンスメソッドがあります。最初のメソッドでは、使用可能なすべてのオプションを指定できます。

- スープ名
- インデックス指定が含まれる NSArray オブジェクト
- 外部 ID 項目名のパス
- エラーをアプリケーションに送信する出力 NSError オブジェクト

**Objective-C:**

```
- (NSArray *)upsertEntries:(NSArray *)entries
 toSoup:(NSString *)soupName
 withExternalIdPath:(NSString *)externalIdPath
 error:(NSError **)error;
```

**Swift:**

```
func upsert(entries: [Any], forSoupNamed: String,
 withExternalIdPath: String) throws -> [Any]
```

**例:**

```
var entries = store.upsert(entries: entries, forSoupNamed: soupName, withExternalIdPath:
 path)
```

2 番目のメソッドは、外部 ID パスに `_soupEntryId` 項目を使用します。

**Objective-C:**

```
- (NSArray *)upsertEntries:(NSArray *)entries
 toSoup:(NSString *)soupName;
```

**Swift:**

```
func upsert(entries: [[AnyHashable : Any]], forSoupNamed: String) -> [[AnyHashable : Any]]
```

**例:**

```
var entries = store.upsert(entries: entries, forSoupNamed: soupName)
```

**Android ネイティブ:**

Android には、`upsert()` メソッドの 3 つのオーバーロードがあります。最初のオーバーロードでは、使用可能なすべてのオプションを指定できます。

- スープ名
- 1 つ以上の更新/挿入のエントリが含まれる JSON オブジェクト
- 任意の外部 ID 項目名のパス
- 挿入と更新にトランザクションモデルを使用するかどうかを示すフラグ

```
public JSONObject upsert(
 String soupName, JSONObject soupElt, String externalIdPath,
 boolean handleTx)
 throws JSONException
```

2 番目のオーバーロードは、挿入と更新にトランザクションモデルを使用します。

```
public JSONObject upsert(
 String soupName, JSONObject soupElt, String externalIdPath)
 throws JSONException
```

3 番目のオーバーロードは、トランザクションモデルを適用して外部 ID パスに `_soupEntryId` 項目を使用します。

```
public JSONObject upsert(
 String soupName, JSONObject soupElt)
 throws JSONException
```

 **例:** 次の JavaScript コードには、サンプルシナリオが含まれています。まず、`upsertSoupEntries` をコールして、取引先のスープエントリを作成します。コールバックに成功したら、新しく割り当てられたスープエントリ ID を持つ新しいレコードを取得します。次に、取引先の説明を変更し、`forcetk.mobilesdk` メソッドをコールして、サーバーでアカウントを作成し、更新します。最後のコールは、外部 ID を使用した更新/挿入を示します。コードを見やすくするため、エラーコールバックは指定していません。また、`SmartStore` コールはすべて非同期であるため、実際のアプリケーションでは、前のステップの成功コールバックで各ステップを実行します。

このコードでは、レコードがサーバーにまだ存在しないため、`id` 項目の値に `new` を使用しています。アプリケーションがオンラインになったら、ローカルにのみ存在するレコードをクエリ (`id == "new"` であるレコードを検索) して、それをサーバーにアップロードできます。新しいレコードの ID がサーバーから返されると、アプリケーションはスープ内のその `id` 項目を更新できます。

```
var sfSmartstore =
 function() {return cordova.require("com.salesforce.plugin.smartstore");};
// ...
// Specify data for the account to be created
var acc = {id: "new", Name: "Cloud Inc",
 Description: "Getting started"};

// Create account in SmartStore
// This upsert does a "create" because
// the account has no _soupEntryId field
sfSmartstore().upsertSoupEntries("accounts", [acc],
 function(accounts) {
 acc = accounts[0];
 // acc should now have a _soupEntryId field
 // (and a _lastModifiedDate as well)
 });

// Update account's description in memory
acc["Description"] = "Just shipped our first app ";

// Update account in SmartStore
// This does an "update" because acc has a _soupEntryId field
sfSmartstore().upsertSoupEntries("accounts", [acc],
 function(accounts) {
 acc = accounts[0];
 });

// Create account on server
// (sync client -> server for entities created locally)
force.create("account", {
 "Name": acc["Name"],
 "Description": acc["Description"]},
 function(result) {
```

```
acc["id"] = result["id"];
// Update account in SmartStore
sfSmartstore().upsertSoupEntries("accounts", [acc]);
});

// Update account's description in memory
acc["Description"] = "Now shipping for iOS and Android";

// Update account's description on server
// Sync client -> server for entities existing on server
force.update("account", acc["id"],
 {"Description": acc["Description"]});

// Later, there is an account (with id: someSfdcId) that you want
// to get locally

// There might be an older version of that account in the
// SmartStore already

// Update account on client
// sync server -> client for entities that might or might not
// exist on client
force.retrieve(
 "account", someSfdcId, "id,Name,Description",
 function(result) {
 // Create or update account in SmartStore
 // (looking for an account with the same sfdcId)
 sfSmartstore().upsertSoupEntriesWithExternalId(
 "accounts", [result], "id");
 });
```

## 大きいスープ要素での外部ストレージの使用

数年前に Mobile SDK は外部ストレージを導入しました。これは、大きな JSON 文字列をデータベースに保存する際の制限に対応するためのもので、たとえば Android では 1MB のカーソルウィンドウの制限がありました。この制限はなくなりました。さらに、最近の性能分析では、外部ストレージの方が内部ストレージよりも遅くなっています。

Mobile SDK では、今後 SmartStore スープによって外付けストレージを使用することを推奨しません。外部ストレージ機能は Mobile SDK 11.0 で削除されました。

このセクションの内容:

[スープ指定](#)

[外部ストレージを使用するスープの登録](#)

[外部ストレージを使用するスープの変更](#)

## スープ指定

外部ストレージ機能とスープ指定は Mobile SDK 11.0 で削除されました。



## 外部ストレージを使用するスープの登録

外部ストレージ機能は Mobile SDK 11.0 で削除されました。

## 外部ストレージを使用するスープの変更

外部ストレージ機能と `alterSoup` メソッドは Mobile SDK 11.0 で削除されました。

## スープ要素の削除

従来より、SmartStore のメソッドでは要素 ID の配列を指定してスープ要素を削除できます。通常これを行うには、候補 ID を取得する事前クエリを実行してから、削除を実行するメソッドをコールします。Mobile SDK 4.2 では SmartStore が強化され、要素削除メソッドにクエリオプションが追加されました。このオプションではクエリのみを指定し、SmartStore はそのクエリに適合するすべての要素を削除します。この方法では1回のコールでクエリと削除操作の両方が行われるため、パフォーマンスが向上します。

## ハイブリッドアプリケーション

ハイブリッドアプリケーションでは、3 つ目のパラメーターを使用して ID 配列または SmartStore クエリ指定を渡します。

```
removeFromSoup([isGlobalStore,]soupName, entryIdsOrQuerySpec,
 successCB, errorCallback)
removeFromSoup([storeConfig,]soupName, entryIdsOrQuerySpec,
 successCB, errorCallback)
```

成功とエラーのコールバックに加えて、この関数は次の引数を取ります。

表 5: パラメーター

| パラメーター名                          | 引数の説明                                                                                             |
|----------------------------------|---------------------------------------------------------------------------------------------------|
| <code>isGlobalStore</code>       | (省略可能) この操作がグローバルまたはユーザーベースのどちらの SmartStore データベースで行われるかを示す Boolean。デフォルトは <code>false</code> です。 |
| <code>storeConfig</code>         | (省略可能) ストア名とストアがグローバルまたはユーザーベースのどちらかであるかを指定する <code>StoreConfig</code> オブジェクト。                    |
| <code>soupName</code>            | 文字列型。スープの名前を渡します。                                                                                 |
| <code>entryIdsOrQuerySpec</code> | 配列または <code>QuerySpec</code> オブジェクト。スープの名前を渡します。                                                  |

## Android ネイティブアプリケーション

エントリを削除する Android ネイティブメソッドでは、トランザクションを手動で処理するか、メソッドでトランザクションを透過的に処理させるかを選択できます。`handleTx` 引数を `false` に設定すると、コールの

前に手動でトランザクションを開始し、コール後に手動で終了する必要があります。handleTx が含まれないオーバーロードを使用する場合、または handleTx を false に設定した場合、Mobile SDK が自動的にトランザクションを処理します。

Android ネイティブアプリケーションで ID 配列によってエントリを削除するには、次のいずれかのメソッドをコールします。

```
public void delete(String soupName, Long... soupEntryIds)
public void delete(String soupName, Long[] soupEntryIds, boolean handleTx)
```

Android ネイティブアプリケーションでクエリによってエントリを削除するには、次のいずれかのメソッドをコールします。

```
public void deleteByQuery(String soupName, QuerySpec querySpec)
public void deleteByQuery(String soupName, QuerySpec querySpec, boolean handleTx)
```

## iOS ネイティブアプリケーション

iOS ネイティブアプリケーションで ID 配列によってエントリを削除するには、次のいずれかのメソッドをコールします。

Objective-C:

```
- (void)removeEntries:(NSArray*)entryIds fromSoup:(NSString*)soupName error:(NSError **)error;
```

Swift:

```
public func remove(entryIds: [Any], forSoupName: String) -> Void
```

例:

```
remove(entryIds: entries, forSoupNamed: soupName)
```

iOS ネイティブアプリケーションでクエリによってエントリを削除するには、次のいずれかのメソッドをコールします。

Objective-C:

```
- (void)removeEntriesByQuery:(SFQuerySpec*)querySpec
 fromSoup:(NSString*)soupName;
- (void)removeEntriesByQuery:(SFQuerySpec*)querySpec
 fromSoup:(NSString*)soupName
 error:(NSError **)error;
```

Swift:

```
func remove(usingQuerySpec: QuerySpec, entryIds: [Any], forSoupNamed: String) -> Void
```

例:

```
var removed = removeEntries(usingQuerySpec: querySpec, forSoupNamed: soupName)
```

## スープの管理

SmartStore には、スープメタデータを取得して、スープレベルの他の操作を実行できるユーティリティ機能が用意されています。この機能は、ハイブリッド、React Native、Android ネイティブ、および iOS ネイティブアプリケーションで使用できます。

### iOS ネイティブアプリケーション

iOS ネイティブアプリケーションでスープ管理 API を使用するには、SmartStore/SFSmartStore.h をインポートします。SmartStore オブジェクトの共有インスタンスでスープ管理メソッドをコールします。次のいずれかの SFSmartStore クラスメソッドを使用して、共有インスタンスを取得します。

現在のユーザーの SmartStore インスタンスを使用する場合

#### Swift

```
func shared(withName: String) -> SmartStore
```

例:

```
var store = SmartStore.shared(withName: storeName)
```

#### Objective-C

```
+ (id)sharedStoreWithName:(NSString*) storeName;
```

指定したユーザーの SmartStore インスタンスを使用する場合

#### Swift

```
func shared(withName: String, forUserAccount: SFUserAccount) -> SmartStore
```

例:

```
var store = SmartStore.shared(withName: storeName, forUserAccount: user)
```

#### Objective-C

```
+ (id)sharedStoreWithName:(NSString*) storeName
 user:(SFUserAccount *)user;
```

例:

```
self.store = [SFSmartStore sharedStoreWithName:kDefaultSmartStoreName];
if ([self.store soupExists:@"Accounts"]) {
 [self.store removeSoup:@"Accounts"];
}
```

### Android ネイティブアプリケーション

Android ネイティブアプリケーションでスープ管理 API を使用するには、SmartStore 共有インスタンスで次のメソッドをコールします。

```
SmartStore smartStore =
 SmartStoreSDKManager.getInstance().getSmartStore();
smartStore.clearSoup("user1Soup");
```

## ハイブリッドアプリケーション

JavaScript の各スープ管理関数が2つのコールバック関数を取ります。1つは要求されたデータを返す成功コールバックで、もう1つはエラーコールバックです。成功コールバックは、そのコールバックを使用するスープ管理関数によって異なります。エラーコールバックは、エラーの説明文字列を含む1つの引数を取ります。たとえば、エラーコールバック関数は次のように定義できます。

```
function(e) { alert("ERROR: " + e); }
```

JavaScript でスープ管理関数をコールするには、最初に Cordova プラグインを呼び出して SmartStore オブジェクトを初期化します。そして、SmartStore オブジェクトを使用してスープ管理関数をコールします。次の例では、名前付きのコールバック関数を個別に定義していますが、匿名のコールバック関数をインラインで定義することもできます。

```
var sfSmartstore = function() {
 return cordova.require("com.salesforce.plugin.smartstore");
};

function onSuccessRemoveSoup(param) {
 logToConsole() ("onSuccessRemoveSoup: " + param);
 $("#div_soup_status_line").html("Soup removed: "
 + SAMPLE_SOUP_NAME);
}

function onErrorRemoveSoup(param) {
 logToConsole() ("onErrorRemoveSoup: " + param);
 $("#div_soup_status_line").html("removeSoup ERROR");
}

sfSmartstore().removeSoup(SAMPLE_SOUP_NAME,
 onSuccessRemoveSoup,
 onErrorRemoveSoup);
```

このセクションの内容:

### データベースサイズの取得

データベースによるハードディスクの消費容量を照会するには、データベースサイズメソッドをコールします。

### スープのクリア

スープのすべてのエントリを削除するには、スープクリアメソッドをコールします。

### スープのインデックス指定の取得

スープのインデックス指定を検証または表示する場合は、適切なインデックス指定取得メソッドをコールします。

### スープの既存のインデックス指定の変更

既存のインデックス指定を変更するには、適切なスープ変更メソッドをコールします。

### スープへの再インデックス付け

以前にスープを変更しデータへの再インデックス付けを行わなかったが、その後スープの全要素に適切なインデックスを付けることにした場合は、再インデックス付けを行います。`alterSoup()` と `reindexSoup()` の両方は、他のインデックス指定のタイプよりも JSON1 インデックス指定の変換または作成で優れたパフォーマンスを発揮します。

### スープの削除

スープを削除すると消去されます。ユーザーがサインアウトすると、すべてのスープが自動的に削除されます。サインアウト以外でスープを削除する場合は、適切なスープ削除メソッドをコールします。

関連トピック:

[既存の Android アプリケーションへの SmartStore の追加](#)

## データベースサイズの取得

データベースによるハードディスクの消費容量を照会するには、データベースサイズメソッドをコールします。

### ハイブリッドアプリケーション

ハイブリッドアプリケーションでは、次の関数をコールします。

```
navigator.smartstore.getDatabaseSize(successCallback, errorCallback)
```

成功コールバックは、データベースサイズ(バイト単位)を含む1つのパラメーターをサポートします。次に例を示します。

```
function(dbSize) { alert("db file size is:" + dbSize + " bytes"); }
```

### Android ネイティブアプリケーション

```
public int getDatabaseSize ()
```

### iOS ネイティブアプリケーション

Objective-C:

```
- (long) getDatabaseSize
```

Swift では、Objective-C メソッドを使用します。

## スープのクリア

スープのすべてのエントリを削除するには、スープクリアメソッドをコールします。

## ハイブリッドアプリケーション

ハイブリッドアプリケーションでは、次の関数をコールします。

```
navigator.smartstore.clearSoup(soupName, successCallback, errorCallback)
```

成功コールバックは、スープ名を含む1つのパラメーターをサポートします。次に例を示します。

```
function(soupName) { alert("Soup " + soupName + " was successfully emptied."); }
```

## Android アプリケーション

Android アプリケーションでは、次のメソッドをコールします。

```
public void clearSoup (String soupName)
```

## iOS アプリケーション

Objective-C:

```
- (void)clearSoup:(NSString*) soupName;
```

Swift:

```
func clearSoup(soupName:) -> Void
```

例:

```
store.clearSoup(soupName: soupName)
```

## スープのインデックス指定の取得

スープのインデックス指定を検証または表示する場合は、適切なインデックス指定取得メソッドをコールします。

## ハイブリッドアプリケーション

ハイブリッドアプリケーションでは、次の関数をコールします。

```
getSoupIndexSpecs()
```

この関数は、成功およびエラーコールバック関数の他に、1つの引数 `soupName` を取ります。この引数はスープの名前です。次に例を示します。

```
navigator.smartstore.getSoupIndexSpecs(soupName, successCallback,
 errorCallback)
```

成功コールバックは、インデックス指定の配列を含む1つのパラメーターをサポートします。次に例を示します。

```
function(indexSpecs) { alert("Soup " + soupName +
 " has the following indexes:" + JSON.stringify(indexSpecs); }
```

## Android アプリケーション

```
public IndexSpec [] getSoupIndexSpecs (String soupName)
```

## iOS アプリケーション

Objective-C:

```
- (NSArray*)indicesForSoup:(NSString*) soupName
```

Swift:

```
func indices(forSoupNamed: String) -> [SoupIndex]
```

例:

```
var soupIndices = store.indices(forSoupNamed:name)
```

## スープの既存のインデックス指定の変更

既存のインデックス指定を変更するには、適切なスープ変更メソッドをコールします。

データに再インデックス付けする場合は、次のパフォーマンスに関するヒントに留意してください。

- 再インデックス付けはコストがかかる可能性があるため、`reIndexData` 引数は省略可能です。`reIndexData` を `false` に設定すると、スループットが大幅に加速するものと予想されます。
- すでにデータが含まれているスープを変更すると、アプリケーションのパフォーマンスが低下することがあります。`reIndexData` を `true` に設定すると、パフォーマンスヒットが悪化します。
- パフォーマンスの指針として、`reIndexData` が `true` に設定されている場合の `alterSoup()` 操作の所要時間は 1000 レコードあたり 1 秒です。個別のパフォーマンスは、デバイスの性能、要素のサイズ、およびインデックスの数によって異なります。
- `alterSoup()` および `reindexSoup()` は、他のインデックス指定のタイプよりも JSON1 インデックス指定の変換または作成で優れたパフォーマンスを発揮します。
- JSON1 インデックス指定では、挿入のパフォーマンスが向上します。
- JSON1 インデックス指定ではデータベースサイズが小さくなります。
- クエリのパフォーマンスは、一般的に JSON1 インデックス指定による影響はありません。
- スープの変更が完了するまで `SmartStore` の他の操作は実行できません。
- ユーザーがアプリケーションを終了するなどして操作が中断された場合は、アプリケーションで `SmartStore` データベースが再度開かれたときにその操作が自動的に再開されます。

## 外部ストレージを使用するインデックス指定の変更

外部ストレージ機能を使用するスープを登録した場合、「[外部ストレージを使用するスープの変更](#)」で説明されている `alterSoup` メソッドを使用します。



## ハイブリッドアプリケーション

ハイブリッドアプリケーションでは、次の関数をコールします。

```
navigator.smartstore.alterSoup(soupName, indexSpecs, reIndexData,
 successCallback, errorCallback)
```

成功とエラーのコールバックに加えて、この関数は次の引数を取ります。

表 6: パラメーター

| パラメーター名     | 引数の説明                                          |
|-------------|------------------------------------------------|
| soupName    | 文字列型。スープの名前を渡します。                              |
| indexSpecs  | 配列型。インデックス指定のインデックスエントリのセットを渡します。              |
| reIndexData | ブール型。インデックス指定を置換後に関数でスープに再インデックス付けするかどうかを示します。 |

成功コールバックは、スープ名を含む1つのパラメーターをサポートします。次に例を示します。

```
function(soupName) { alert("Soup " + soupName +
 " was successfully altered"); }
```

次の例では、スープの単純な変更を行います。最初に、開発者が name 項目および address 項目にインデックス付けされる項目を定義し、エージェントレコードを更新/挿入します。

```
navigator.smartstore.registerSoup("myAgents",
 [{path:'name', type:'string'},
 {path:'address', type:'string'}]);
navigator.smartstore.upsertSoupEntries("myAgents",
 [{name:'James Bond',
 address:'1 market st',
 agentNumber:"007"}]);
```

過去の経験から、ユーザーがエージェントの address ではなく「agentNumber」を使用したクエリを望んでいることが判明している場合は、address へのインデックスを削除して、agentNumber へのインデックスを追加します。

```
navigator.smartstore.alterSoup("myAgents", [{path:'name', type:'string'}, {path:'agentNumber',
 type:'string'}], true);
```

-  **メモ:** 開発者が reIndexData パラメーターを false に設定した場合は、agentNumber に対するクエリで、挿入済みのエントリ(「JamesBond」)が検索されません。ただし、name でレコードを照会することは可能です。agentNumber によるクエリをサポートするには、最初に navigator.smartstore.reIndexSoup("myAgents", ["agentNumber"]) をコールする必要があります。

## Android ネイティブアプリケーション

Android ネイティブアプリケーションでは、次のメソッドをコールします。

```
public void alterSoup(String soupName, IndexSpec [] indexSpecs, boolean reIndexData) throws
JSONException
```

## iOS ネイティブアプリケーション

Objective-C:

```
- (BOOL) alterSoup:(NSString*)soupName withIndexSpecs:(NSArray*)indexSpecs
reIndexData:(BOOL)reIndexData;
```

Swift では、Objective-C メソッドを使用します。

## スープへの再インデックス付け

以前にスープを変更しデータへの再インデックス付けを行わなかったが、その後スープの全要素に適切なインデックスを付けることにした場合は、再インデックス付けを行います。alterSoup() と reindexSoup() の両方は、他のインデックス指定のタイプよりも JSON1 インデックス指定の変換または作成で優れたパフォーマンスを発揮します。

## ハイブリッドアプリケーション

ハイブリッドアプリケーションでは、次の関数をコールします。

```
navigator.smartstore.reIndexSoup(soupName, listOfPaths, successCallback, errorCallback)
```

この関数は、成功およびエラーコールバック関数の他に、1つの引数 soupName を取ります。この引数はスープの名前です。たとえば、次の関数は追加の引数を取ります。

| パラメーター名     | 引数の説明                        |
|-------------|------------------------------|
| soupName    | 文字列型。スープの名前を渡します。            |
| listOfPaths | 配列型。再インデックス付けするインデックスパスのリスト。 |

成功コールバックは、スープ名を含む1つのパラメーターをサポートします。次に例を示します。

```
function(soupName) { alert("Soup " + soupName +
 " was successfully re-indexed."); }
```

## Android アプリケーション

Android アプリケーションでは、次のメソッドをコールします。

```
public void reIndexSoup(String soupName, String[] indexPaths, boolean handleTx)
```

## iOS アプリケーション

Objective-C:

```
- (BOOL) reIndexSoup:(NSString*) soupName
 withIndexPaths:(NSArray*) indexPaths
```

Swift では、Objective-C メソッドを使用します。

## スープの削除

スープを削除すると消去されます。ユーザーがサインアウトすると、すべてのスープが自動的に削除されます。サインアウト以外でスープを削除する場合は、適切なスープ削除メソッドをコールします。

## ハイブリッドアプリケーション

ハイブリッドアプリケーションでは、次の関数をコールします。

```
navigator.smartstore.removeSoup(soupName, successCallback, errorCallback);
```

## Android アプリケーション

Android アプリケーションでは、次のメソッドをコールします。

```
public void dropSoup (String soupName)
```

## iOS アプリケーション

Objective-C:

```
- (void) removeSoup:(NSString*) soupName
```

Swift:

```
func removeSoup(soupName: String) -> Void
```

例:

```
store.removeSoup(soupName: soupName)
```

## ストアの管理

グローバルストアを作成する場合、アプリケーションの終了時にクリーンアップを実行する必要があります。また、複数のユーザーストアを作成する場合、特定のストアを使用しなくなったときにクリーンアップを実行できます。SmartStore では、名前付きストアやグローバルストアを削除するメソッドが提供されます。ハイブリッドアプリケーションの場合、SmartStore では、名前付きストアのリストを取得する関数も提供されます。

## iOS ネイティブアプリケーション

Mobile SDK for iOS では、ストアを削除する次の `SFSmartStore` メソッドが定義されています。

**Swift**

```
func removeShared(withName: String) -> Void
func removeShared(withName: String, forUserAccount: UserAccount) -> Void
func removeSharedGlobal(withName: String) -> Void
func removeAllForCurrentUser() -> Void
func removeAll(forUserAccount: UserAccount) -> Void
func removeAllGlobal() -> Void
```

**Objective-C**

```
+ (void)removeSharedStoreWithName:(NSString *)storeName;
+ (void)removeSharedStoreWithName:(NSString *)storeName forUser:(SFUserAccount *)user;
+ (void)removeSharedGlobalStoreWithName:(NSString *)storeName;
+ (void)removeAllStores;
+ (void)removeAllStoresForUser:(SFUserAccount *)user;
+ (void)removeAllGlobalStores;
```

また、SmartStore では、ストア名を取得する次のメソッドも提供されます。このメソッドは、Swift アプリケーションと Objective-C アプリケーションの両方に使用します。

```
+ (NSArray *)allStoreNames;
+ (NSArray *)allGlobalStoreNames;
```

**Android ネイティブアプリケーション**

Mobile SDK for Android では、ストアを削除する次の SmartStoreSDKManager メソッドが定義されています。

```
public void removeGlobalSmartStore(String dbName)
public void removeSmartStore()
public void removeSmartStore(UserAccount account)
public void removeSmartStore(UserAccount account, String communityId)
public void removeSmartStore(String dbNamePrefix, UserAccount account, String communityId)
```

また、SmartStore では、ストア名を取得する次のメソッドも提供されます。

```
public List<String> getGlobalStoresPrefixList()
public List<String> getUserStoresPrefixList()
```

**ハイブリッドアプリケーション**

SmartStore には、ストアを削除する次の関数が定義されています。各関数は、成功コールバックとエラーコールバックを取得します。removeStore() 関数には、ストア名を指定する StoreConfig オブジェクト、または文字列のストア名のみのいずれかも必要です。

```
removeStore(storeConfig, successCB, errorCallback)
removeAllGlobalStores(successCB, errorCallback)
removeAllStores(successCB, errorCallback)
```

また、SmartStore のハイブリッドバージョンでは、定義されたストアの StoreConfig オブジェクトを取得する次の関数も提供されます。

```
getAllStores(successCB, errorCallback)
getAllGlobalStores(successCB, errorCallback)
getAllStores(successCB, errorCallback)
getAllGlobalStores(successCB, errorCallback)
```

### 例: 実行時に SmartStore のすべてのデータを削除する

アプリケーションは、現在のユーザーをログアウトさせずにストア内のすべてのデータを削除することが必要な場合があります。この場合、同期マネージャーオブジェクトでは、同期を追跡するテーブルが設定されます。このテーブルを削除すると、マネージャーは続行できなくなります。したがって、SmartStore をデータの無い状態にリセットする場合、推奨される方法は、次のようになります。

1. 現在のユーザーに関連付けられている同期マネージャーを削除します。

#### iOS

##### Swift

次の SyncManager メソッドをコールします。

```
func removeSharedInstance(user: UserAccount) -> Void
```

##### Objective-C

次の SFMobileSyncSyncManager メソッドをコールします。

```
+ (void)removeSharedInstance:(SFUserAccount*)user;
```

#### Android

次の MobileSyncSyncManager メソッドをコールします。

```
public static synchronized void reset(UserAccount account)
```

2. 現在のユーザーに関連付けられているストアを削除します。

#### iOS

##### Swift

次の SmartStore メソッドをコールします。

```
func removeAll(forUserAccount: UserAccount) -> Void
```

##### Objective-C

次の SFSmartStore メソッドをコールします。

```
+ (void)removeAllStoresForUser:(SFUserAccount *)user;
```

#### Android

次の SmartStoreSDKManager メソッドをコールします。

```
public void removeAllUserStores()
```

## SmartStore インспекターを使用したテスト

テスト中に SmartStore の操作を確認することは面倒で時間がかかります。SmartStore インспекターでこの問題を解消できます。

テスト中、SmartStore データがコードで意図したとおりに処理されているかどうかを確認できると便利です。SmartStore インспекターには、こうしたテストを行うためのモバイルUI クラスが用意されています。SmartStore インспекターでは、次のことができます。

- スープメタデータ (スープの名前やインデックス指定など) を検証する。
- スープのコンテンツをクリアする。
- Smart SQL クエリを実行する。

 **メモ:** SmartStore インспекターは、テストとデバッグのみのためにあります。SmartStore インспекターにコード参照を追加する場合は、アプリケーションの最終バージョンを作成する前に必ず削除してください。

Mobile SDK 6.0 以降、デバッグビルドで Dev Tools メニューから SmartStore インспекターにアクセスできるようになりました。この機能により、アプリにコードを追加する必要がなくなりました。「[開発者向け Mobile SDK ツール](#)」を参照してください。

## 擬似 SmartStore の使用

コンテナ外で実行しながら SmartStore を使用するコードの開発とテストを容易にするため、エミュレートされた SmartStore を使用できます。

MockSmartStore は、データをローカルストレージ (または必要に応じてメモリのみ) に保存する SmartStore の JavaScript 実装です。

external/shared/test ディレクトリには、次のファイルがあります。

- MockCordova.js — コンテナ外でプラグインのテストのみを目的とする、Cordova 関数の最小限の実装。Cordova プラグインコールを受信します。
- MockSmartStore.js — コンテナ外での開発およびテストのみを目的とする、SmartStore の JavaScript 実装。また、SmartStore Cordova プラグインコールを受信し、MockSmartStore を使用して処理します。

SmartStore を使用してアプリケーションを開発する場合、次の変更を加え、コンテナ外でアプリケーションをテストします。

- cordova.js ではなく MockCordova.js を含める。
- MockSmartStore.js を含める。

MockSmartStore の例を確認するには、[github.com/forcedotcom/SalesforceMobileSDK-Shared](https://github.com/forcedotcom/SalesforceMobileSDK-Shared) リポジトリの test/test.html を参照してください。

## 同じ発生元のポリシー

同じ発生元のポリシーでは、同じサイトを発生元とする各ページでスクリプトを実行すると、スクリプトは互いのメソッドおよびプロパティに制限なしでアクセスできます。また、異なるサイトのページ間では、ほとんどのメソッドおよびプロパティへのアクセスがブロックされます。コンテナでは同じ発生元のポリシーが Web ビューで無効になるため、同じ発生元のポリシーに関する制限は、コンテナ内でコードを実行する場合には問

題になりません。ただし、リモート API をコールする場合は、同じ発生元のポリシーに関する制限を考慮する必要があります。

幸い、ブラウザーには同じ発生元のポリシーを無効にする方法が備えられています。使用する特定のブラウザーでその方法について調べることができます。ローカルファイルシステムから読み込まれた JavaScript ファイルから Salesforce Platform に対して XHR コールを行う場合は、同じ発生元のポリシーを無効にしてブラウザーを起動する必要があります。次の記事は、一般的ないくつかのブラウザーで同じ発生元のポリシーを無効にする方法を説明しています: [「Getting Around Same-Origin Policy in Web Browsers \(Web ブラウザーでの同じ発生元のポリシーの回避\)」](#)。

## 認証

MockSmartStore を使用する認証では、アクセストークンと更新トークンを実際のセッションから取得し、そのコードを JavaScript アプリケーションで手動で作成する必要があります。また、force.js JavaScript Toolkit の初期化にもこれらのトークンが必要です。

### メモ:

- MockSmartStore は、データを暗号化せず、本番アプリケーションでの使用を目的としていません。
- 現在、MockSmartStore では、次の形式の Smart SQL クエリがサポートされています。
  - SELECT...WHERE...。次に例を示します。

```
SELECT {soupName:selectField} FROM {soupName} WHERE {soupName:whereField} IN (values)
```

- SELECT...WHERE...ORDER BY...。次に例を示します。

```
SELECT {soupName:_soup} FROM {soupName} WHERE {soupName:whereField} LIKE 'value' ORDER BY LOWER({soupName:orderByField})
```

- SELECT count(\*) FROM {soupName}

MockSmartStore では、build\*QuerySpec() 関数で処理される、より単純なタイプの Smart SQL ステートメントは直接サポートされていません。代わりに、目的に合ったクエリ指定の関数を使用されます。

関連トピック:

[スープからのデータの取得](#)

## Mobile Sync のスープの準備

通常、Salesforce クラウドと情報を交換するスープは、MobileSync を使用して同期できます。MobileSync をサポートするために、大部分のアプリケーション種別では「上位同期」操作の特殊なスープ項目を作成および管理する必要があります。

次のようなアプリケーション種別では、これらの特殊な項目をコーディングする必要があります。

- ローカルレコードの作成および管理に (mobilesync.js の) Force.SObject を使用しないハイブリッドアプリケーション
- ネイティブアプリケーション



- React Native アプリケーション

ハイブリッドアプリケーションでローカルレコードに `Force.SObject` を使用する場合、Mobile Sync によってこれらの項目が自動的に作成および管理されます。この説明の残りの部分は無視して問題ありません。

## 必須項目の追加

1. スーブ要素に次の項目を追加します。最初の3つは、操作種別項目です。

### 作業種別の項目

作成、更新、または削除操作ごとに適切な項目を `true` に設定します。

#### `__locally_created__`

- 型: `string`
- アプリケーションでローカルに作成する要素でこの項目を `true` に設定します。

#### `__locally_updated__`

- 型: `string`
- アプリケーションで要素をローカルに更新したらこの項目を `true` に設定します。

#### `__locally_deleted__`

- 型: `string`
- アプリケーションで要素をローカルに削除しているときにこの項目を `true` に設定します。

### 制御項目

#### `__local__`

- 型: `string`
- この項目は、何らかのローカルな変更が行われたことを示します。次の作業が必要です。
  - いずれかの操作種別項目が `true` の場合にこの項目を `true` に設定する。
  - この項目の文字列インデックス指定を追加する。

#### `__sync_id__`

- 型: `integer`
- この項目によって、`cleanResyncGhosts()` メソッドが目的のスーブ要素のみを削除するように指定できます。Mobile Sync はこの項目の内容を管理します。

2. 操作項目と制御項目のそれぞれにスーブインデックスを追加します。「[設定ファイルを使用したスーブの登録](#)」を参照してください。

## Mobile Sync の動作

Mobile Sync は、上位同期操作中に `__local__` が `true` に設定されたスーブ要素を検索します。一致するたびに操作種別項目を評価し、次の優先順位階層で示される操作を実行します。

| 優先順位   | 項目                  | true に設定されている場合                                                                                                                                                                                         |
|--------|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 (最高) | __locally_deleted__ | <ul style="list-style-type: none"> <li>__locally_created__ および __locally_updated__ フラグは無視されます。</li> <li>Mobile Sync によってローカルレコードとサーバーレコード(存在する場合)が削除されます。サーバーレコードが存在しない場合、リモートアクションは実行されません。</li> </ul> |
| 2      | __locally_created__ | <ul style="list-style-type: none"> <li>__locally_updated__ フラグは無視されます。</li> <li>__locally_deleted__ が true でない場合、Mobile Sync は、サーバー上でレコードを作成します。</li> </ul>                                             |
| 3      | __locally_updated__ | <ul style="list-style-type: none"> <li>__locally_deleted__ または __locally_created__ のいずれかが true の場合は無視されます。</li> <li>それ以外の場合、Mobile Sync は、更新されたレコードをサーバーに書き込みます。</li> </ul>                             |

最後に、Mobile Sync によって 4 つのすべての項目が false にリセットされます。

 **例:** 次の例は、MobileSyncExplorer サンプルアプリケーションのさまざまな言語バージョンから取得されています。

#### iOS ネイティブ

この Objective-C の例では、updateSoupForFieldName:fieldValue: メッセージを SObjectData オブジェクトに送信してシステム項目が設定されます。項目名に SFMobileSyncSyncManager 定数を使用すると、新しい要素を更新/挿入する前に \_\_local\_\_ および \_\_locally\_created\_\_ 項目が設定されます。iOS サンプルアプリケーションには SObjectData 定義があります。

```
- (void)createLocalData:(SObjectData *)newData {
 [newData updateSoupForFieldName:kSyncManagerLocal fieldValue:@YES];
 [newData updateSoupForFieldName:kSyncManagerLocallyCreated fieldValue:@YES];
 [self.store upsertEntries:@[newData.soupDict] toSoup:[newData class]
dataSpec].soupName];
}
```

#### Android ネイティブ

次の Java の例では、作成および更新された要素が処理されますが、削除は処理されません。項目名に SyncManager 定数を使用して、システム項目を作成および初期化する JSONObject.put() メソッドがコールされます。項目が適切に割り当てられたら、isCreate 制御フラグに基づいて要素の作成または更新/挿入が行われます。

```
contact.put(SyncTarget.LOCAL, true);
contact.put(SyncTarget.LOCALLY_UPDATED, !isCreate);
contact.put(SyncTarget.LOCALLY_CREATED, isCreate);
contact.put(SyncTarget.LOCALLY_DELETED, false);
```

```

if (isCreate) {
 smartStore.create(ContactListLoader.CONTACT_SOUP, contact);
} else {
 smartStore.upsert(ContactListLoader.CONTACT_SOUP, contact);
}

```

### Mobile Sync プラグインと React Native を使用するハイブリッド

次の React Native コードは、Mobile Sync プラグインを使用するハイブリッドアプリケーションに簡単に適応できます。次の例では、取引先責任者を更新、削除、または復元する方法を示します。onSaveContact() 関数は、レコードを更新済みとしてマークして \_\_local\_\_ を true に設定し、変更を保存します。onDeleteUndeleteContact() 関数は、\_\_locally\_deleted\_\_ 項目を反転します。次に、操作種別の値に合わせて \_\_local\_\_ 項目を設定し、変更を保存します。

サンプルプロジェクトでは、storeMgr オブジェクトは SmartStore と MobileSync プラグインを囲むラッパーとして定義されます。その saveContact() 関数は、取引先責任者オブジェクトとコールバックを受け入れ、取引先責任者をスープに更新/挿入します。ここに示すコールバックは、React Native に固有の navigator.pop() をコールします。ハイブリッドアプリケーションは、saveContact() 関数を、SmartStore upsert() 関数をコールする任意のコードに置き換えることができます。

```

onSaveContact: {
 const contact = this.state.contact;
 contact.__locally_updated__ = contact.__local__ = true;
 storeMgr.saveContact(contact, () => {navigator.pop();});
},

onDeleteUndeleteContact: {
 const contact = this.state.contact;
 contact.__locally_deleted__ = !contact.__locally_deleted__;
 contact.__local__ = contact.__locally_deleted__ || contact.__locally_updated__ ||
 contact.__locally_created__;
 storeMgr.saveContact(contact, () => {navigator.pop();});
},

```

## Swift アプリケーションでの SmartStore の使用

SmartStore の基礎部分は、forceios ネイティブ Swift プロジェクトに簡単にインストールできます。

この例では、SmartStore スープを作成し、照会された取引先責任者名のリストをそのスープに更新/挿入します。次に、Swift テンプレートアプリケーションのフローを変更して、REST 応答から直接ではなく、スープからテーブルビューに入力します。Xcode プロジェクトの構造に馴染みがない場合は、Xcode のヘルプを参照してください。

1. Forceios を使用して、次の例のようなネイティブ Swift プロジェクトを作成します。

```

$ forceios create
Enter your application type (native_swift or native, leave empty for native_swift):
<Press RETURN>
Enter your application name: <Enter any name you like>
Enter your package name: com.myapps.ios
Enter your organization name (Acme, Inc.): MyAppS.com

```

```
Enter output directory for your app (leave empty for the current directory): <Press
RETURN or enter a directory name>
```

2. プロジェクトのルートディレクトリで、次の内容の `userstore.json` ファイルを作成します。

```
{ "soups": [
 {
 "soupName": "Contact",
 "indexes": [
 { "path": "Name", "type": "string"},
 { "path": "Id", "type": "string"}
]
 }
]}
```

3. Xcode で、アプリケーションの `.xcworkspace` ファイルを開きます。
4. 設定ファイルをプロジェクトに追加します。
  - a. Xcode プロジェクトナビゲーターで、プロジェクトノードを選択します。
  - b. エディターウィンドウで、[Build Phases (フェーズの作成)] を選択します。
  - c. [Copy Bundle Resources (バンドルリリースのコピー)] を展開します。
  - d. [+] ([Add items (項目を追加)]) をクリックします。
  - e. スープ設定ファイルを選択します。Xcode プロジェクトフォルダーにまだファイルが準備できていない場合は、次の手順を実行します。
    - i. ファインダーでファイルを選択するには、[Add Other... (その他を追加...)] をクリックします。
    - ii. [Open (開く)] をクリックして、[Finish (完了)] をクリックします。
5. プロジェクトのソースコードフォルダーで、`Classes/AppDelegate.swift` を選択します。
6. `application(_:didFinishLaunchingWithOptions:)` コールバックメソッドで、コールの `userstore.json` 定義を `AuthHelper.loginIfRequired` に読み込みます。

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool
{
 self.window = UIWindow(frame: UIScreen.main.bounds)
 self.initializeAppViewState();
 ...

 AuthHelper.loginIfRequired { [weak self] in
 MobileSyncSDKManager.shared.setupUserStoreFromDefaultConfig()
 self?.setupRootViewController()
 }
 return true
}
```

これで、アプリケーションが起動時に SmartStore 設定を読み込むように設定されました。この操作によって、空のテーブルと指定したスープが作成されます。RootViewController クラスを、SmartStore を使用するように設定しましょう。

7. RootViewController.swift で、SmartStore をインポートします。

```
import SmartStore
```

8. RootViewController の最上部で、SmartStore インスタンスの変数を宣言します。

```
class RootViewController : UITableViewController
{
 var dataRows = [NSDictionary]()
 var store = SmartStore.shared(withName: SmartStore.defaultStoreName)
```

9. 次の行で、OSLog コンポーネントを定義する定数を宣言します。

```
class RootViewController : UITableViewController
{
 var dataRows = [NSDictionary]()
 var store = SmartStore.shared(withName: SmartStore.defaultStoreName)
 let mylog = OSLog(subsystem: "com.testapp.swift", category: "tutorial")
```

10. loadView() メソッドで、.query へのコールを見つけて、SQL ステートメントに Id 項目を追加します。

```
let request = RestClient.shared.request(forQuery: "SELECT Name, Id FROM Contact LIMIT 10")
```

11. guard ブロックの直後の handleSuccess(\_:\_) メソッドで、次のコードを追加します。

```
func handleSuccess(response: RestResponse, request: RestRequest) {
 guard let jsonResponse = try? response.asJson() as? [String:Any],
 let records = jsonResponse["records"] as? [[String:Any]] else {
 SalesforceLogger.d(RootViewController.self, message:"Empty Response for : \
(request)")
 return
 }

 if let smartstore = self.store,
 smartstore.soupExists(forName: "Contact") {
 smartstore.clearSoup("Contact")
 smartstore.upsert(entries: records, forSoupNamed: "Contact")
 os_log("\nSmartStore loaded records.", log: self.mylog, type: .debug)
 }

 SalesforceLogger.d(type(of:self), message:"Invoked: \ (request)")
 DispatchQueue.main.async {
 self.dataRows = records
 self.tableView.reloadData()
 }
}
// end of handleSuccess method
```

このコードは、Contact スープが存在するかどうかをチェックします。スープが存在する場合は、コードがスープのデータをすべてクリアし、取得したレコードを更新/挿入します。

12. アプリケーションを起動し、Dev Tools メニューを使用して作業結果を確認します。

- a. メニューを表示するには、*control + command + z* キーを押すか (iOS エミュレーターを使用している場合)、iOS デバイスをシェイクします。
- b. [Inspect SmartStore (SmartStore の検査)] をクリックします。
- c. Contact スープとレコード数をリストするには、[Soups (スープ)] をクリックします。

 **メモ:** 「Query: No soups found (クエリ: スープが見つかりません)」メッセージが表示される場合は、`userstore.json` ファイルにエラーがある可能性があります。

これで、SmartStore スープの作成と入力完了しました。ただし、この時点では、スープは何の目的も果たしません。REST 応答からではなく、SmartStore レコードからリストビューを入力することで、より役に立つものにしましょう。

1. `handleSuccess(_:_)` メソッドの後に、`loadFromStore()` というメソッドを追加します。

```
func loadFromStore() {

}
```

2. `loadFromStore()` で、最初の条件として SmartSQL クエリ指定を構築する `if` ブロックを定義します。Contact スープから最初の 10 件の Name 値を抽出するようにクエリを設定します。

```
func loadFromStore() {
 if let querySpec = QuerySpec.buildSmartQuerySpec(
 smartSql: "select {Contact:Name} from {Contact}", pageSize: 10),
 {

 }
```

3. SmartStore ハンドルを検証する 2 番目の条件と、SmartStore クエリを実行する 3 番目の条件を追加します。query メソッドで例外が発生するため、`do...try...catch` ブロックからコールします。

```
func loadFromStore() {
 if let querySpec = QuerySpec.buildSmartQuerySpec(
 smartSql: "select {Contact:Name} from {Contact}", pageSize: 10),
 let smartStore = self.store,
 let records = try? smartStore.query(using: querySpec,
 startingFromPageIndex: 0) as? [[String]] {

 }

}
```

4. SmartStore クエリで返された名前をビューの `dataRows` メンバーに転送します。

```
func loadFromStore() {
 if let querySpec = QuerySpec.buildSmartQuerySpec(
 smartSql: "select {Contact:Name} from {Contact}", pageSize: 10),
 let smartStore = self.store,
 let records = try? smartStore.query(using: querySpec,
 startingFromPageIndex: 0) as? [[String]] {
 self.dataRows = records.map({ row in
 return ["Name": row[0]]
 })
 }
```

```

 }
}

```

5. `DispatchQueue` システムオブジェクトを使用して、メインスレッドに切り替え、ビューに表示されたデータを更新します。

```

func loadFromStore() {
 if let querySpec = QuerySpec.buildSmartQuerySpec(
 smartSql: "select {Contact:Name} from {Contact}", pageSize: 10),
 let smartStore = self.store,
 let records = try? smartStore.query(using: querySpec,
 startingFromPageIndex: 0) as? [[String]] {
 self.dataRows = records.map({ row in
 return ["Name": row[0]]
 })
 DispatchQueue.main.async {
 self.tableView.reloadData()
 }
 }
}

```

6. `handleSuccess(_:_:_)` メソッドまでスクロールして戻り、ビューのデータを再読み込みする既存のコードを削除します。

```

func handleSuccess(response: RestResponse, request: RestRequest) {
 guard let jsonResponse = try? response.asJson() as? [String:Any],
 let records = jsonResponse["records"] as? [[String:Any]] else {
 SalesforceLogger.d(RootViewController.self, message:"Empty Response for :
\(request)")
 return
 }

 if ((self.store.soupExists(forName: "Contact"))){
 self.store.clearSoup("Contact")
 self.store.upsert(entries: records, forSoupNamed: "Contact")
 os_log("\nSmartStore loaded records.", log: self.mylog, type: .debug)
 }

 // Remove the following lines
 SalesforceLogger.d(type(of:self), message:"Invoked: \(request)")
 DispatchQueue.main.async {
 self.dataRows = records
 self.tableView.reloadData()
 }
}
} // end of handleSuccess method

```

7. `self` を使用して、`upsert(entries:forSoupNamed:)` コールの直後で新しい `loadFromStore()` メソッドをコールします。

```

func handleSuccess(response: RestResponse, request: RestRequest) {
 guard let jsonResponse = try? response.asJson() as? [String:Any],
 let records = jsonResponse["records"] as? [[String:Any]] else {
 SalesforceLogger.d(RootViewController.self, message:"Empty Response for :

```



```
\(request)")
 return
}

if ((self.store.soupExists(forName: "Contact"))) {
 self.store.clearSoup("Contact")
 self.store.upsert(entries: records, forSoupNamed: "Contact")
 self.loadFromStore()
 os_log("\nSmartStore loaded records.", log: self.mylog, type: .debug)
}
} // end of loadView
```

アプリケーションを再テストすると、テーブルビューが以前と同じように入力されていますが、ライブの REST 応答ではなく SmartStore から入力されていることを確認できます。実際には、Contact リストの編集インターフェースを作成し、顧客の編集を SmartStore に更新/挿入します。その後、顧客はモバイルデバイスの接続が失われても、Contact リストの作業を続行できます。接続が回復したら、Mobile Sync を使用して、顧客の作業をサーバーにマージでき、さらに SmartStore を再同期できます。

## Mobile Sync を使用した Salesforce オブジェクトへのアクセス

Mobile Sync によって、モバイルをご利用のお客様はデバイスの接続が失われた場合でも Salesforce データの作業を続けることができるようになります。接続が回復すると、Mobile Sync は SmartStore のローカルレコードへのすべての変更を Salesforce サーバー上の対応するレコードに同期します。

Mobile Sync では、オフラインシナリオおよびオンラインシナリオでキャッシュデータとサーバーデータ間のやり取りを調整するためのキャッシュポリシーが事前定義されています。Mobile SDK の便利なメソッドを使用すると、sObject メタデータの取得、最近使用されたオブジェクトのリストの取得、SOQL および SOSL クエリの作成など、よく行うネットワーク作業を自動化できます。

## ネイティブアプリケーションでの Mobile Sync の使用

ネイティブ Mobile Sync ライブラリでは、オフライン対応アプリケーションを簡単に開発できるネイティブ iOS および Android API が提供されています。このネイティブ機能のサブセットは、Cordova プラグインを介してハイブリッドアプリケーションでも使用できます。

Mobile Sync ライブラリには、iOS と Android で同様のアーキテクチャおよび機能が用意され、各プラットフォームのネイティブ言語で示されます。両方に共通の分かりやすい機能的な概念が採用されています。

- Salesforce REST API をコールして、Salesforce オブジェクトメタデータを照会する。
- 取得したオブジェクトデータをオフラインでできるようにローカルに安全に保存する。
- デバイスがオフラインからオンライン状態になったときにデータの変更を同期する。

Mobile Sync ネイティブライブラリでは、次の操作を実行できます。

- サーバーエンドポイントとやりとりして、データを取得および投稿する。Mobile Sync ヘルパー API は、使用頻度が最も高いエンドポイントをエンコードします。これらの API は、sObject メタデータの取得、MRU (最近使用した) オブジェクトのリストの取得、SOQL および SOSL クエリの作成に役立ちます。カスタムクラスで指定した任意のエンドポイントを使用することもできます。

- Salesforce レコードおよびメタデータを取得し、事前定義されたキャッシュポリシーの1つを使用してデバイスにキャッシュする。
- レコードをオフラインで編集し、SmartStore にオフラインで保存する。
- ローカルで変更されたデータを Salesforce クラウドに転送して、レコードのバッチを同期する。

## Mobile Sync のコンポーネント

Mobile Sync アーキテクチャの基盤となるコンポーネントは次のとおりです。

### 同期マネージャーのクラス

- **iOS クラス:**

| Swift       | Objective-C             |
|-------------|-------------------------|
| SyncManager | SFMobileSyncSyncManager |

- **Android クラス:** `com.salesforce.androidsdk.mobilesync.manager.SyncManager`

sObject のサイズの大きなバッチをサーバーと SmartStore 間で同期する API を提供します。このクラスは、メタデータマネージャーとは独立して機能し、最も簡単で最も一般的な同期操作の実行を目的とします。同期マネージャーは、sObject のセットをサーバーから SmartStore にダウンロードする「下位同期」、およびローカル sObject をサーバーにアップロードする「上位同期」を行うことができます。

同期マネージャーは、次のユーティリティクラスと連動します。

### 同期状態クラス

同期操作の状態を追跡します。この操作の状態は次のとおりです。

- **新規** — 同期操作が開始されましたが、サーバーとのトランザクションはまだ始まっていません。
- **実行中** — 同期操作がサーバーと同期トランザクションのネゴシエーションを行っています。
- **完了** — 同期操作が正常に完了しました。
- **失敗** — 同期操作に失敗しました。
- **iOS:**

| Swift     | Objective-C |
|-----------|-------------|
| SyncState | SFSyncState |

- **Android:** `com.salesforce.androidsdk.mobilesync.util.SyncState`

### 同期ターゲットクラス

「下位同期」操作中にダウンロードする sObject を指定する親クラス。

- **iOS:**

| Swift      | Objective-C  |
|------------|--------------|
| SyncTarget | SFSyncTarget |

- **Android:** `com.salesforce.androidsdk.mobilesync.util.SyncTarget`

#### 同期オプションクラス

「上位同期」操作の設定オプションを指定します。オプションには、同期する項目名のリストが示されます。

- **iOS:**

| Swift                    | Objective-C                |
|--------------------------|----------------------------|
| <code>SyncOptions</code> | <code>SFSyncOptions</code> |

- **Android:** `com.salesforce.androidsdk.mobilesync.util.SyncOptions`

#### SOQL ビルダー

クエリの句を個々に指定して、SOQL クエリステートメントを簡単に作成できるようにするユーティリティクラスです。

- **iOS クラス:**

| Swift                         | Objective-C                   |
|-------------------------------|-------------------------------|
| <code>SFSDKSoqlBuilder</code> | <code>SFSDKSoqlBuilder</code> |

- **Android クラス:** `com.salesforce.androidsdk.mobilesync.util.SOQLBuilder`

#### SOSL ビルダー

クエリの句を個々に指定して、SOSL クエリステートメントを簡単に作成できるようにするユーティリティクラスです。

- **iOS クラス:**

| Swift                         | Objective-C                   |
|-------------------------------|-------------------------------|
| <code>SFSDKSoslBuilder</code> | <code>SFSDKSoslBuilder</code> |

- **Android クラス:** `com.salesforce.androidsdk.mobilesync.util.SOSLBuilder`

#### MobileSyncSDKManager

Mobile SDK 6.0 以降、すべての forcedroid および forceios テンプレートアプリケーションでは、ベース SDK エントリーポイントとして `MobileSyncSDKManager` を使用します。クラス名の `MobileSyncSDKManager` は、iOS (Objective-C および Swift) と Android で同じです。Android では、App クラスにより、`SalesforceSDKManager` ではなく、`MobileSyncSDKManager` が拡張されます。iOS では、AppDelegate クラスの `init` メソッドで、`SalesforceSDKManager` ではなく `MobileSyncSDKManager` の共有インスタンスが使用されます。この変更は、ネイティブアプリケーションとハイブリッドアプリケーションの両方に適用されます。

-  **メモ:** マルチユーザーの切り替えをサポートするために、Mobile Sync ではユーザーアカウントごとにそのコンポーネントの固有のインスタンスを作成します。

関連トピック:

[SDK マネージャークラス](#)

[SalesforceSDKManager クラス](#)

## Mobile Sync を使用したネイティブアプリケーションの作成

forceios および forcedroid バージョン 5.0 以降では Mobile Sync ネイティブアプリケーションを生成するのに特別な努力は必要ありません。作成するネイティブアプリケーションには自動的に SmartStore および Mobile Sync ライブラリが含まれます。

## 既存の Android アプリケーションへの Mobile Sync の追加

Mobile SDK 4.0 以降で作成された既存の Android プロジェクト (ハイブリッドまたはネイティブ) に Mobile Sync を追加する手順を次に示します。

1. 現在アプリケーションが Mobile SDK 3.3 以前に構築されている場合は、「[以前のリリースからの移行](#)」の説明に従って、プロジェクトを SDK の最新バージョンにアップグレードします。
2. MobileSync ライブラリプロジェクトをプロジェクトに追加します。Mobile Sync は SmartStore を使用するため、プロジェクトが SmartStore を使用して構築されていない場合は、このライブラリも追加する必要があります。
  - a. Android Studio で、`libs/MobileSync` プロジェクトをモジュール連動関係に追加します。
3. プロジェクト全体で、`SalesforceSDKManager` オブジェクトを使用するすべてのコードを、代わりに `MobileSyncSDKManager` を使用するように変更します。

-  **メモ:** プロジェクト全体の検索および置換を実行する場合は、`KeyInterface` インポートは変更せず、そのままにします。

```
import com.salesforce.androidsdk.app.SalesforceSDKManager.KeyInterface;
```

## 既存の iOS アプリケーションへの Mobile Sync の追加

Mobile Sync をサポートするための既存の iOS プロジェクトのアップグレードは簡単に行うことができます。forceios を使用して新しいプロジェクトを作成し、アセットに追加するだけです。ただし、従来の Mobile Sync アプリケーションを Mobile SDK 6.0 以降にアップグレードする手順を知りたい場合は、ここの記載されている情報が役立ちます。

Mobile SDK 4.0 では、Mobile Sync が Mobile SDK コアからそのライブラリに移されています。Mobile Sync は SmartStore に依存しているため、forceios によって自動的に SmartStore がプロジェクトに追加されます。また、Mobile SDK 6.0 以降のネイティブ iOS プロジェクトでは、アプリケーションを初期化するのに `MobileSyncSDKManager` オブジェクトが必要になります。

1つずつ変更する代わりに、新しいネイティブプロジェクトを作成し、アセットをそのプロジェクトにコピーすることをお勧めします。新しいシェルプロジェクトは、テンプレートソースファイルを使用して手動で作成するか、forceios を使用して作成できます。手動でプロジェクトを作成するには、「[iOS Swift プロジェクトの手動作成](#)」を参照してください。

ネイティブテンプレートアプリケーションは、デフォルトで MobileSyncSDKManager クラスを使用して、適切なライブラリやその他の多くの更新をインポートします。

<https://github.com/forcedotcom/SalesforceMobileSDK-Templates/blob/master/iOSNativeTemplate/iOSNativeTemplate/AppDelegate.m> を参照してください。

## 同期ターゲットについて

同期ターゲットは、モバイルデバイスでの Salesforce クラウドとローカルデータベース間のデータ転送を設定します。Mobile SDK 5.1 では同期ターゲットの機能が強化され、開発者は双方向のデータ同期を以前よりも詳細に管理できます。

Mobile Sync はデータの同期専用です。基本的に、次の同期を行います。

- サーバーからローカルデータベースへのデータの **下位同期**
- ローカルデータベースからサーバーへのデータの **上位同期**。

多くの場合、転送するデータはルールを破ることがなく、デフォルトの同期ターゲットが正常に動作します。ただし特殊なケースでは、データ転送が確実に期待どおり行われるように、カスタムの同期ターゲットを指定できます。たとえば、必須項目が含まれるオブジェクトがあり、アプリケーションがその項目を更新できない場合などがあります。上位同期操作が1つの項目リストを使用して新規レコードと更新レコードの両方をアップロードし、ロックされた項目を更新しようとする、操作は失敗します。Mobile SDK 5.1 以降、多くの場合カスタムの同期ターゲットの実装が不要になる他のオプションがあります。

## 同期マネージャーのタスクの分散 (カスタムの同期ターゲットの機能強化)。

最初の Mobile Sync リリースでは、同期マネージャークラスがすべてのサーバーとローカルデータベースのやりとりを内部的に処理していました。また、同期マネージャーは開発者がカスタマイズできない「最後の」クラスでした。開発者は、独自のニュアンスを加えたり機能を拡張したりすることができませんでした。

その後アーキテクチャのリファクタリングにより、すべてのサーバーとのやりとりは同期マネージャークラスから下位同期クラスと上位同期クラスに委任されました。ここから、単一の同期マネージャーから柔軟な同期ターゲットへの機能の移行が始まりました。同期マネージャーとは異なり、第二世代の同期ターゲットクラスでは開発者が独自の目的で同期ターゲットをサブクラス化できます。サーバーとのやりとりを制御することで、カスタムの同期ターゲットは任意のサーバーエンドポイントと通信したり、データを保存する前に変換したりできます。

Mobile SDK 5.1 では、ローカルデータベースとのやりとりを同期ターゲットに移行することで、Mobile Sync がさらに強化されています。この機能強化にはいくつかの利点があります。

- Mobile Sync が SmartStore から分離されるため、管理者は自由に他のストアを使用できる。
- 管理者は独自のデータレイアウトを使用して、任意の方法でローカルデータの変更を捕捉できる。
- 複数のレコードタイプを同時に処理できる同期ターゲットなど、複雑なオブジェクトを使用できる。

つまり、Mobile Sync で開発者がデータ同期の往復処理全体を詳細に制御できるようになりました。



-  **メモ:** Mobile SDK 5.1 以降、追加の「上位同期」オプションにより、カスタムの同期ターゲットが不要になる場合があります。「[カスタム上位同期ターゲットの定義](#)」を参照してください。

## 同期名と同期設定ファイルの定義

Mobile SDK 6.0 以降、同期設定ファイルを定義して、名前を同期設定に割り当てることができます。同期名を使用して、保存した同期操作を実行、編集、または削除できます。すべてのプラットフォームとアプリケーション種別は同じ設定ファイルを使用するため、1つのファイルにすべての同期を記述できます。次に、そのファイルをコンパイルして、任意の Mobile SDK プロジェクトを作成できます。

Mobile Sync 設定ファイルは、JSON オブジェクトを使用して同期の定義を表現します。このファイルを用意すると、下位同期および上位同期の設定をコーディングする必要がなくなります。設定ファイルの JSON スキーマは、すべてのアプリケーション種別およびプラットフォームで同じです。ハイブリッドアプリケーションは設定ファイルを自動的に読み込みますが、それ以外のアプリケーションは読み込みに 1 行のコードを使用します。メカニズムをシンプルにするために、Mobile SDK では次のファイル命名規則が適用されます。

- デフォルトのグローバルストアの同期操作を定義するには、`globalsyncs.json` というファイルを指定します。
- デフォルトのユーザーストアの同期操作を定義するには、`usersyncs.json` というファイルを指定します。

設定ファイルは、デフォルトのグローバルストアおよびデフォルトのユーザーストアの同期のみを定義できます。名前指定ストアの場合は、コードを通じて同期を定義します。

ネイティブアプリケーションおよび ReactNative アプリケーションの場合は、同期読み込みメソッドをコールして JSON 設定ファイルを読み込みます。このメソッドは、顧客が正常にログインした後にアプリケーション初期化コード内でコールします。たとえば、iOS の場合、このメソッドは、`loginIfRequired` に渡すブロック内でコールします。このメソッドをコールするのは、同期を設定するコードの代わりに `globalsyncs.json` ファイルまたは `usersyncs.json` ファイルを使用している場合に限られます。同期読み込みメソッドは、複数回コールしないでください。

これらを含むハイブリッドアプリケーションの場合、同期設定ファイルは、自動的に読み込まれます。ローダーの例については、`MobileSyncExplorer` および `MobileSyncExplorerHybrid` サンプルアプリケーションを参照してください。これらのアプリケーションでは、設定ファイルを使用して同期操作を設定します。

 **メモ:**

- 設定ファイルは、最初の設定のみを目的としています。JSON ファイルを改訂して実行時に再度読み込み、既存の同期を変更することはできません。その代わりに、`syncUp` または `syncDown` メソッドを使用して、同期をインラインで定義できます。
- 設定ファイルで同期操作に割り当てられる名前が存在する場合、その設定ファイルは Mobile SDK で無視されます。この場合、同期の設定および管理は、コードでのみ行うことができます。

## 設定ファイルの形式

次の例は、設定ファイルの形式を示しています。

```
{
 "syncs": [
```

```

 {
 "syncName": "sync1",
 "syncType": "syncDown",
 "soupName": "accounts",
 "target": { "type": "soql",
 "query": "SELECT Id, Name, LastModifiedDate
 FROM Account",
 "maxBatchSize": 200 },
 "options": { "mergeMode": "OVERWRITE" }
 },
 {
 "syncName": "sync2",
 "syncType": "syncUp",
 "soupName": "accounts",
 "target": { "createFieldlist": ["Name"] },
 "options": { "fieldlist": ["Id", "Name", "LastModifiedDate"],
 "mergeMode": "LEAVE_IF_CHANGED" }
 }
]
}

```

下位同期の場合、"target" プロパティの "type" プロパティは、次のいずれかの値を受け入れます。

- **"soql"**  
 下位同期用の SOQL クエリを使用します。  
 プロパティ:
  - "type": "soql"
  - "query": <string>
  - "idFieldName": <string>
  - "modificationDateFieldName": <string>
  - "maxBatchSize": <integer>。<integer> には、200 から 2,000 までの任意の値を指定します (デフォルト値は 2,000)。
 必須項目: "type"、"query"
- **"sosl"**  
 下位同期用の SOSL クエリを使用します。  
 プロパティ:
  - "type": "sosl"
  - "query": <string>
  - "idFieldName": <string>
  - "modificationDateFieldName": <string>
  - "maxBatchSize": <integer>。<integer> には、200 から 2,000 までの任意の値を指定します (デフォルト値は 2,000)。
 必須項目: "type"、"query"
- **"briefcase"**  
 下位同期用のブリーフケースを使用します。



**プロパティ:**

- "type": "briefcase"
- "infos": <BriefcaseObjectInfo> **項目の配列**

**必須項目:** "type"、"infos"

「[ブリーフケース下位同期ターゲットの使用](#)」を参照してください

- **"mru"**

特定のオブジェクトの最後に使用されたレコードを同期します。

**プロパティ:**

- "type": "mru"
- "subjectType": <string>
- "fieldlist": <string> **項目の配列**
- "idFieldName": <string>
- "modificationDateFieldName": <string>

**必須項目:** "type"、"subjectType"、"fieldlist"

- **"refresh"**

特定のスープ内にある特定のオブジェクトと項目の同期を更新します。

**プロパティ:**

- "type": "refresh"
- "subjectType": <string>
- "fieldlist": <string> **項目の配列**
- "soupName": <string>
- "idFieldName": <string>
- "modificationDateFieldName": <string>

**必須項目:** "type"、"subjectType"、"fieldlist"、"soupName"

- **"layout"**

特定のオブジェクトのレイアウトを同期します。

**プロパティ:**

- "type": "layout"
- "subjectType": <string>
- "formFactor": <"Large" | "Medium" | "Small"> **のいずれかを選択します。**
- "layoutType": <"Compact" | "Full"> **のいずれかを選択します。**
- "mode": <"Create" | "Edit" | "View"> **のいずれかを選択します。**
- "recordTypeeld": <string>
- "idFieldName": <string>
- "modificationDateFieldName": <string>

**必須項目:** "type"、"subjectType"、"layoutType"

- "metadata"**  
 特定のオブジェクトのメタデータを同期します。  
 プロパティ:
  - "type": "metadata"
  - "subjectType": <string>
  - "idFieldName": <string>
  - "modificationDateFieldName": <string>
 必須項目: "type"、"subjectType"
- "parent\_children"**  
 特定の親オブジェクトの関連レコードを同期します。  
 プロパティ: 「[関連レコードの同期](#)」 (ページ 372) を参照してください。
- "custom"**  
 カスタム下位同期ターゲットを使用して同期します。iOS および Android のネイティブターゲット下位同期クラスの名前を "iOSImpl" および "androidImpl" プロパティに割り当てます。  
 プロパティ:
  - "type": "custom"
  - "iOSImpl": <string>
  - "androidImpl": <string>
  - "idFieldName": <string>
  - "modificationDateFieldName": <string>
 必須項目: "type"、"iOSImpl"、"androidImpl"

デフォルトでは、同期設定ファイルで定義された上位同期ターゲットは、一括 API を使用します。また、すべての上位同期ターゲットは、次のようになります。

- "createFieldList" プロパティを定義できます。
- "updateFieldList" プロパティを定義できます。
- "type" プロパティを定義しません。

各種の上位同期ターゲットの設定を次に示します。

- sObject コレクション**  
 Mobile SDK 10.1 以降の標準の上位同期ターゲット種別。  
 プロパティ:
  - "createFieldlist": <string> 項目の配列
  - "updateFieldlist": <string> 項目の配列
  - "externalIdFieldName": <string>
 必須項目: なし

- バッチ

標準ターゲットに似ていますが、より小さな一括操作を使用します。Mobile SDK 7.1 ～ 10.0 の標準ターゲット。

プロパティ:

- "createFieldlist": <string> 項目の配列
- "updateFieldlist": <string> 項目の配列
- "externalIdFieldName": <string>

必須項目: なし

- バッチ以外

標準ターゲットに似ていますが、一括操作またはコレクション操作は使用しません。

プロパティ:

- "iOSImpl": "SFSyncUpTarget"
- "androidImpl": "com.salesforce.androidsdk.mobilesync.target.SyncUpTarget"
- "createFieldlist": <string> 項目の配列
- "updateFieldlist": <string> 項目の配列
- "externalIdFieldName": <string>

必須項目: "iOSImpl"、"androidImpl"

- 親子

特定の親オブジェクトの関連レコードを同期します。

プロパティ: [「関連レコードの同期」](#) (ページ 372) を参照してください。

- カスタム

カスタム上位同期ターゲットを使用して同期します。iOS および Android のネイティブ上位同期ターゲットクラスの名前を "iOSImpl" および "androidImpl" プロパティに割り当てます。

プロパティ:

- "iOSImpl": <string>
- "androidImpl": <string>
- "createFieldlist": <string> 項目の配列
- "updateFieldlist": <string> 項目の配列

必須項目: "iOSImpl"、"androidImpl"

ターゲットの JSON 定義は、[Mobile Sync JSON スキーマ](#)で指定されます。

## 設定ファイルの場所

設定ファイルがどこに配置されるかは、アプリケーション種別やプラットフォームによって異なります。Mobile SDK は、次の場所で設定ファイルを探します。

### iOS (ネイティブおよび React Native)

リソースバンドルの下/内

## Android (ネイティブおよび React Native)

/res/raw プロジェクトフォルダー内

### ハイブリッド

Cordova プロジェクトで、次の手順を実行します。

1. 設定ファイルを最上位の `www/` フォルダーに配置します。
2. 最上位のプロジェクトディレクトリで、`cordova prepare` を実行します。

## 設定ファイルからの同期定義の読み込み (iOS ネイティブアプリケーション)

読み込み方法は、`MobileSyncSDKManager` クラスで定義されます。

### ユーザーストア

#### Swift

```
MobileSyncSDKManager.sharedManager.setupUserSyncsFromDefaultConfig()
```

#### Objective-C

```
[[MobileSyncSDKManager sharedManager] setupUserSyncsFromDefaultConfig];
```

### グローバルストア

#### Swift

```
MobileSyncSDKManager.sharedManager.setupGlobalSyncsFromDefaultConfig()
```

#### Objective-C

```
[[MobileSyncSDKManager sharedManager] setupGlobalSyncsFromDefaultConfig];
```

## 設定ファイルからの同期定義の読み込み (Android ネイティブアプリケーション)

読み込み方法は、`MobileSyncSDKManager` で定義されます。これらのローダーは、アプリケーションの任意の場所からコールできます。同期または再同期機能をコールする前にこのコールが実行されるようにします。

### ユーザーストア

```
MobileSyncSDKManager.getInstance().setupUserSyncsFromDefaultConfig();
```

### グローバルストア

```
MobileSyncSDKManager.getInstance().setupGlobalSyncsFromDefaultConfig();
```

## 同期名の使用

Mobile SDK には、名前指定同期操作を使用および管理するための一連の API が用意されています。実行時にプログラムで名前指定同期を作成および削除したり、名前指定同期を名前で実行または再実行したり、メモリ内の名前指定同期を管理したりできます。

### 名前ベースの API (iOS)

これらのメソッドの多くは新規追加されました。更新されたメソッドでは、`target`、`options`、`updateBlock` で既存の類似メソッドと同じパラメーターが使用されます。

## 名前による同期状況の取得

## Swift

```
var syncStatus = syncManager.getSyncStatus(syncName: syncState.syncName)
```

## Objective-C

```
- (nullable
 SFSyncState*)getSyncStatusByName:(NSString*) syncName;
```

## 名前による既存の同期の確認

## Swift

```
var syncExists = syncManager.hasSync(syncName: syncState.syncName)
```

## Objective-C

```
- (BOOL)hasSyncWithName:(NSString*) syncName;
```

## 名前による同期設定の削除

## Swift

```
syncManager.deleteSync(syncName: syncState.syncName)
```

## Objective-C

```
- (void)deleteSyncByName:(NSString*) syncName;
```

## 名前付き同期設定の作成、実行または再実行

「[下位同期](#)」および「[上位同期](#)」を参照してください。

## 名前付き同期設定を使用した cleanResyncGhosts のコール

「[同期名による cleanResyncGhosts メソッドのコール](#)」を参照してください。

## 名前ベースの API (Android)

これらのメソッドの多くは新規追加されました。上書きされたメソッドでは、target、options、callback で既存の類似メソッドと同じパラメーターが使用されます。

## 名前による同期状況の取得

```
public SyncState getSyncStatus(String name);
```

## 名前による既存の同期の確認

```
public boolean hasSyncWithName(String name);
```

## 名前による同期設定の削除

```
public void deleteSync(String name);
```

## 名前付き同期設定の作成、実行または再実行

「[下位同期](#)」および「[上位同期](#)」を参照してください。

## 名前付き同期設定を使用した cleanResyncGhosts のコール

「[同期名による cleanResyncGhosts メソッドのコール](#)」を参照してください。

## 名前ベースの API (ハイブリッド)

これらのメソッドの多くは既存の従来の API です。同期 ID が受け入れられるかどうかに関係なく、代わりに同期名を渡すことができます。

### 名前による同期状況の取得

この関数を使用して、同期設定が存在するかどうかを判断できます。同期設定が存在しない場合、null が返されます。

```
getSyncStatus(storeConfig, syncIdOrName, successCB, errorCB)
```

### 名前による同期設定の削除

```
deleteSync(storeConfig, syncIdOrName, successCB, errorCB)
```

### 名前指定同期設定の作成および実行

従来の `syncDown()` 関数に `syncName` パラメーターが含まれるようになりました。名前を指定すると、Mobile SDK によって指定された名前で作成されます。要求された同期名がすでに存在する場合、この関数は失敗します。

```
syncDown(storeConfig, target, soupName, options, syncName, successCB, errorCB)
```

```
syncUp(storeConfig, target, soupName, options, syncName, successCB, errorCB)
```

### 名前による同期の実行 (再実行)

この既存のメソッドに、同期 ID または同期名のいずれかを受け入れるオーバーロードが追加されました。

```
reSync(storeConfig, syncIdOrName, successCB, errorCB)
```

## 名前ベースの API (React Native)

これらのメソッドの多くは既存の従来の API です。同期 ID が受け入れられるかどうかに関係なく、代わりに同期名を渡すことができます。

### 名前による同期状況の取得

この既存のメソッドに、同期 ID または同期名のいずれかを受け入れるオーバーロードが追加されました。

```
getSyncStatus(storeConfig, syncIdOrName, successCB, errorCB)
```

### 名前による削除

Mobile SDK 6.0 で新規追加されたこのメソッドは、同期 ID または同期名を受け入れます。

```
deleteSync(storeConfig, syncIdOrName, successCB, errorCB)
```

### 名前による同期の作成および実行 - 新しい省略可能なパラメーター `syncName`

この既存のメソッドに、同期名を受け入れる省略可能なパラメーターが追加されました。

```
syncDown(storeConfig, target, soupName, options, syncName, successCB, errorCB)
syncUp(storeConfig, target, soupName, options, syncName, successCB, errorCB)
```

### 名前による同期の実行 (再実行) - ID または名前を受け入れるためにオーバーロード

この既存のメソッドに、同期 ID または同期名のいずれかを受け入れるオーバーロードが追加されました。

```
reSync(storeConfig, syncIdOrName, successCB, errorCB)
```

### 例: iOS ネイティブアプリケーションでの再同期メソッドの呼び出し

MobileSyncExplorerSwift テンプレートアプリケーションの SObject.swift からの抜粋。この例の updateRemoteData は、上位同期設定(kSyncUpName)を使用して reSync をコールします。この操作に成功すると、下位同期設定(kSyncDownName)を使用して refreshRemoteData をコールします。このフォローアップステップにより、サーバーのすべての最新の変更がスープに反映されます。

```
func updateRemoteData(_ onSuccess: @escaping ([SObjectData]) -> Void, onFailure:@escaping
(NSError?, SyncState?) -> Void) -> Void {
 do {
 try self.syncMgr.reSync(named: kSyncUpName) { [weak self] (syncState) in
 guard let strongSelf = self else {
 return
 }
 switch (syncState.status) {
 case .done:
 do {
 let objects = try strongSelf.queryLocalData()
 strongSelf.populateDataRows(objects)
 try strongSelf.refreshRemoteData({ (sobjs) in
 onSuccess(sobjs)
 }, onFailure: { (error,syncState) in
 onFailure(error,syncState)
 })
 } catch let error as NSError {
 MobileSyncLogger.e(SObjectDataManager.self, message: "Error with
Resync \(error)")
 onFailure(error,syncState)
 }
 break
 case .failed:
 MobileSyncLogger.e(SObjectDataManager.self, message: "Resync
\(syncState.syncName) failed")
 onFailure(nil,syncState)
 break
 default:
 break
 }
 } catch {
 onFailure(error as NSError, nil)
 }
 }
}
```

```
func refreshRemoteData(_ completion: @escaping ([SObjectData]) -> Void,
 onFailure: @escaping (NSError?, SyncState) -> Void)
 throws -> Void {
 try self.syncMgr.reSync(named: kSyncDownName) { [weak self] (syncState) in
 switch (syncState.status) {
 case .done:
 do {
 let objects = try self?.queryLocalData()
 self?.populateDataRows(objects)
 } catch {
 onFailure(error as NSError, nil)
 }
 }
 }
}
```



```

 completion(self?.fullDataRowList ?? [])
 } catch {
 MobileSyncLogger.e(SObjectDataManager.self, message: "Resync
\ (syncState.syncName) failed \ (error)")
 }
 break
case .failed:
 MobileSyncLogger.e(SObjectDataManager.self, message: "Resync
\ (syncState.syncName) failed")
 onFailure(nil, syncState)
default:
 break
}
}
}

```



### 例: Android ネイティブアプリケーションでの再同期メソッドの呼び出し

MobileSyncExplorerKotlinTemplate の SObjectSyncableRepoBase.kt からの抜粋。

```

private suspend fun doSyncUp(): SyncState {
 try {
 return syncManager.suspendReSync(syncUpName)
 } catch (es: SyncManager.ReSyncException.FailedToStart) {
 throw SyncUpException.FailedToStart(cause = es)
 } catch (ef: SyncManager.ReSyncException.FailedToFinish) {
 throw SyncUpException.FailedToFinish(cause = ef)
 }
}

```

Android MobileSyncExplorer ネイティブサンプルアプリケーションの ContactListLoader.java からの抜粋:

```

public synchronized void syncUp() {
 try {
 syncMgr.reSync(SYNC_UP_NAME /* see usersyncs.json */, new SyncUpdateCallback()
 {
 @Override
 public void onUpdate(SyncState sync) {
 if (Status.DONE.equals(sync.getStatus())) {
 syncDown();
 }
 }
 });
 } catch (JSONException e) {
 Log.e(TAG, "JSONException occurred while parsing", e);
 } catch (MobileSyncException e) {
 Log.e(TAG, "MobileSyncException occurred while attempting to sync up", e);
 }
}

```

```

public synchronized void syncDown() {
 try {
 syncMgr.reSync(SYNC_DOWN_NAME /* see usersyncs.json */, new SyncUpdateCallback()
 {
 @Override

```

```

 public void onUpdate(SyncState sync) {
 if (Status.DONE.equals(sync.getStatus())) {
 fireLoadCompleteIntent();
 }
 }
 });
} catch (JSONException e) {
 Log.e(TAG, "JSONException occurred while parsing", e);
} catch (MobileSyncException e) {
 Log.e(TAG, "MobileSyncException occurred while attempting to sync down", e);
}
}

```



### 例: ハイブリッドアプリケーションでの再同期メソッドの呼び出し

MobileSyncExplorerHybrid サンプルアプリケーションの MobileSyncExplorer.html からの抜粋:

```

syncDown: function() {
 cordova.require("com.salesforce.plugin.mobilesync").reSync("syncDownContacts" /*
see usersyncs.json */,
 this.handleSyncUpdate.bind(this));
},
syncUp: function() {
 cordova.require("com.salesforce.plugin.mobilesync").reSync("syncUpContacts" /* see
usersyncs.json */,
 this.handleSyncUpdate.bind(this));},

```

## データの同期

Mobile Sync ネイティブアプリケーションでは、同期マネージャーを使用してデバイスと Salesforce サーバー間のデータを簡単に同期できます。同期マネージャーは、「上方」同期(デバイスからサーバー)または「下方」同期(サーバーからデバイス)のメソッドを提供します。

Mobile Sync アプリケーションのデータ要求はすべて非同期です。非同期であるため、コールする同期メソッドは、サーバーの応答をコールバックメソッドまたは定義された更新ブロックで返します。

上位同期または下位同期メソッドは、それぞれ同期状態オブジェクトを返します。このオブジェクトには次の情報が含まれます。

- 同期操作 ID。この ID を同期マネージャーの `getSyncStatus` メソッドに渡すことで、いつでも操作の進行状況を確認できます。
- 同期パラメーター (スコープ名、下位同期操作のターゲット、上位同期操作のオプション)。
- 操作の種類 (上方または下方)。
- 進行率 (整数、0 ~ 100)。
- トランザクション内の総レコード数。

## 同期マネージャーの使用

同期マネージャーオブジェクトは、単純な上位同期操作と下位同期操作を処理します。下位同期の場合、ユーザーに代わって認証済み要求をサーバーに送信し、応答データを SmartStore にローカルに保存します。上位同

期の場合、指定したレコードを SmartStore から収集し、ユーザーの指示に従って対応するサーバーレコードとマージします。同期マネージャーは、Salesforce ユーザーおよびコミュニティユーザーの認証を処理する方法を把握しています。同期マネージャーは、すべてのユーザーまたはグローバル SmartStore インスタンス (デフォルトのインスタンスまたは名前付きインスタンス) にレコードを保存できます。

同期マネージャークラスには、カスタマイズされた同期マネージャーインスタンスを返すファクトリーメソッドがあります。同期マネージャーを使用するには、同期操作の要件に一致するインスタンスを作成します。たとえば、Mobile SDK は、レイアウト専用の同期マネージャークラスとメタデータ専用の同期マネージャークラスを提供します。

**❗ 重要:** 各同期アクティビティの正しいタイプの同期マネージャーを作成することが最も重要です。そのようにしないと、顧客にランタイム認証エラーが発生する可能性があります。

インスタンスを作成したら、次のような一般的な同期マネージャー機能をコールするために使用できます。

- 下位同期
- 上位同期
- 再同期
- 停止
- 再開

同期マネージャーは、SmartStore スープエントリおよび Salesforce レコードで次の 3 つの種別のアクションを実行できます。

- 作成
- 更新
- 削除

カスタムターゲットを指定すると、同期マネージャーはカスタムターゲットを使用して、任意の REST エンドポイントでデータを同期できます。

## 同期マネージャーの状態

同期マネージャーオブジェクトは、実行時に次の 3 つの状態を進行します。

- `accepting_syncs` — 同期マネージャーは同期操作を開始できる。
- `stopping` — 同期マネージャーの `stop` メソッドがコールされた。
- `stopped` — すべての同期操作が停止した。

同期マネージャーは、`accepting_syncs` 状態の場合にのみ同期操作を開始できます。同期マネージャーの `restart` メソッドをコールすると、`stopping` または `stopped` の状態が `accepting_syncs` にリセットされます。`restartStoppedSyncs` を `true` に設定して `restart` メソッドをコールすると、同期マネージャーは停止された同期操作ごとに `reSync` をコールします。

## 同期マネージャーのインスタンス化 (iOS)

| Swift クラス                | Objective-C クラス                      |
|--------------------------|--------------------------------------|
| <code>SyncManager</code> | <code>SFMobileSyncSyncManager</code> |

iOS では、アクセスおよび削除メソッドのペアを使用します。SFMobileSyncSyncManager クラスで sharedInstance クラスメソッドをコールし、各シナリオの事前設定済み共有インスタンスにアクセスします。特定の使用事例の共有インスタンスの使用を終了する場合、対応する removeSharedInstance メソッドで削除します。

指定のユーザーの場合:

#### Swift

```
SFMobileSyncSyncManager.sharedInstance(user: userAccount)
SFMobileSyncSyncManager.removeSharedInstance(user: userAccount)
```

#### Objective-C

```
+ (instancetype) sharedInstance:(SFUserAccount *)user;
+ (void) removeSharedInstance:(SFUserAccount *)user;
```

特定の SmartStore データベースを使用する指定のユーザーの場合:

#### Swift

```
SFMobileSyncSyncManager.sharedInstance(forUser: userAccount, storeName: "StoreName")
SFMobileSyncSyncManager.removeSharedInstance(forUser: userAccount, storeName:
"StoreName")
```

#### Objective-C

```
+ (instancetype) sharedInstanceForUser:(SFUserAccount *)user
 storeName:(NSString *)storeName;

+ (void) removeSharedInstanceForUser:(SFUserAccount *)user
 storeName:(NSString *)storeName;
```

現在のユーザーと特定の SmartStore データベースの場合:

#### Swift

```
SFMobileSyncSyncManager.sharedInstance(for: store)
SFMobileSyncSyncManager.removeSharedInstance(for: store)
```

#### Objective-C

```
+ (instancetype) sharedInstanceForStore:(SFSmartStore *)store;
+ (void) removeSharedInstanceForStore:(SFSmartStore *)store;
```

## 同期マネージャーのインスタンス化 (Android)

Android では、次の各シナリオで異なるファクトリーメソッドを使用します。

現在のユーザーの場合:

```
public static synchronized SyncManager getInstance();
```

指定のユーザーの場合:

```
public static synchronized SyncManager
getInstance(UserAccount account);
```

指定のコミュニティの指定のユーザーの場合:

```
public static synchronized SyncManager
getInstance(UserAccount account, String communityId);
```

指定の SmartStore データベースを使用する指定のコミュニティの指定のユーザーの場合:

```
public static synchronized SyncManager
getInstance(UserAccount account, String communityId, SmartStore smartStore);
```



**例:** 共有ストアを使用して同期マネージャーを初期化する Swift の例を次に示します。

```
store = SFSmartStore.sharedStore(withName: kDefaultSmartStoreName) as? SFSmartStore
syncManager = SFMobileSyncSyncManager.sharedInstance(for:store!)
```

## グローバル SmartStore での同期マネージャーの使用

グローバル SmartStore インスタンスで Mobile Sync を使用するには、同期マネージャーオブジェクトで静的ファクトリーメソッドをコールして、互換性のある同期マネージャーインスタンスを取得します。

### iOS

#### Swift

```
SyncManager.sharedInstance(store: store!)
```

#### Objective-C

```
+ (instancetype)
sharedInstanceForStore:
(SFSmartStore *)store;
```

現在のユーザーとしてサーバーと通信し、特定の SmartStore インスタンスに対する読み書きを行う同期マネージャーインスタンスを返します。このファクトリーメソッドを使用して、グローバル SmartStore インスタンスとデータを同期します。

### Android

```
SyncManager getInstance(UserAccount account, String communityId, SmartStore smartStore);
```

指定したコミュニティユーザーとしてサーバーと通信し、特定の SmartStore インスタンスに対する読み書きを行う同期マネージャーインスタンスを返します。このファクトリーメソッドを使用して、グローバル SmartStore インスタンスとデータを同期します。

### ハイブリッド

次の各メソッドの省略可能な最初の引数では、グローバルストアを使用するかどうかを Mobile Sync プラグインに通知します。この引数は、Boolean 値または StoreConfig オブジェクトを受け入れます。StoreConfig オブジェクトを使用する場合、コンテキストに応じて storeName、isGlobalStore、またはその両方を指定できます。「[ユーザーベースストアの作成およびアクセス](#)」を参照してください。

- `syncDown(isGlobalStore, target, soupName, options, successCB, errorCallback);`  
`syncDown(storeConfig, target, soupName, options, successCB, errorCallback);`
- `reSync(isGlobalStore, syncId, successCB, errorCallback);`  
`reSync(storeConfig, syncId, successCB, errorCallback);`
- `syncUp(isGlobalStore, target, soupName, options, successCB, errorCallback);`  
`syncUp(storeConfig, target, soupName, options, successCB, errorCallback);`
- `getSyncStatus(isGlobalStore, syncId, successCB, errorCallback);`  
`getSyncStatus(storeConfig, syncId, successCB, errorCallback);`

関連トピック:

[ユーザーベースストアの作成およびアクセス](#)

[グローバル SmartStore の使用](#)

## 同期状況について

「処理中」の同期操作を完全に制御できるように、Mobile SDK 7.1 では、操作が停止されたことを示す新しい状況が追加されています。

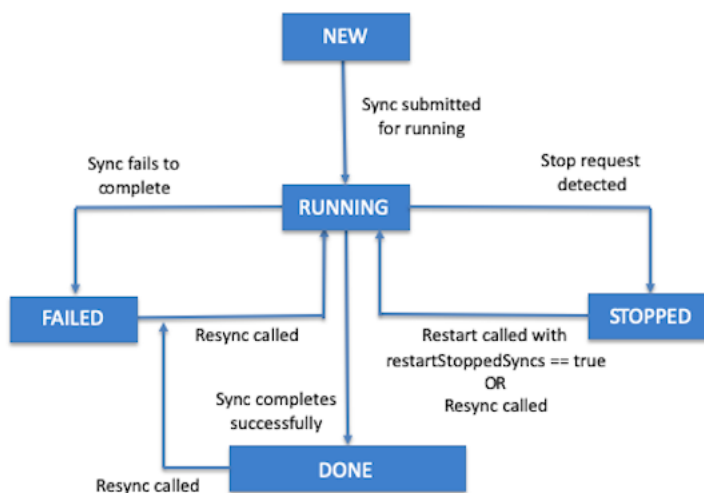
Mobile SDK 7.1 より前では、各同期操作は `new`、`running`、`failed/done` の状況サイクルを進行していました。Mobile SDK 7.1 では、`stopped` 状況が最終ステップに追加され、サイクルが次のように変更されます。

1. `new`
2. `running`
3. `stopped` | `failed` | `done`

重要なイベントが発生すると、同期状況が次のように更新されます。

| イベント                                                                          | 以前の状況                | 報告される状況              |
|-------------------------------------------------------------------------------|----------------------|----------------------|
| アプリケーションで同期オブジェクトが作成される                                                       | —                    | <code>new</code>     |
| 下位同期メソッドまたは上位同期メソッドへのコール、または <code>reSync</code> への最初のコールで実行される同期オブジェクトが送信される | <code>new</code>     | <code>running</code> |
| 同期操作が正常に完了する                                                                  | <code>running</code> | <code>done</code>    |
| 同期操作の完了に失敗する                                                                  | <code>running</code> | <code>failed</code>  |
| 同期マネージャーの停止要求が検出される                                                           | <code>running</code> | <code>stopped</code> |

| イベント                                                                           | 以前の状況               | 報告される状況 |
|--------------------------------------------------------------------------------|---------------------|---------|
| <code>restartStoppedSyncs</code> パラメーターが <code>true</code> に設定されて同期マネージャーが再開する | stopped             | running |
| <code>reSync</code> がコールされる                                                    | stopped/failed/done | running |



 メモ:

- 同期操作が現在実行中の場合、`reSync` メソッドへのコールは失敗します。
- 同期マネージャーの状況が `accepting_syncs` でない場合、`reSync`、`syncDown`、`syncUp`、`cleanResyncGhosts` メソッドへのコールは失敗します。
- たとえば、同期マネージャーの最初の初期化時に同期操作が実行されていて、アプリケーションが同期の開始直後にクラッシュしている場合、同期マネージャーはそれらの状況を `stopped` に設定します。



## 状況による同期オブジェクトの取得

Mobile SDK 7.1 以降、同期マネージャーオブジェクトでは、状況が指定の値に一致するすべての同期状態オブジェクトを返す `getSyncsWithStatus` メソッドがサポートされています。たとえば、このメソッドを使用して、停止しているすべての同期を反復処理し、手動で再開できます。

 **メモ:** 停止された同期の再開を手動で制御することを想定するには、`restartStoppedSyncs` を `false` に設定して同期マネージャーの `restart` メソッドをコールします。

iOS

Swift

```
open class func getSyncsWithStatus(_ store: SmartStore,
 status: SyncStatus) -> [SyncState]
```

Objective-C

```
+ (NSArray<SFSyncState>) getSyncsWithStatus:(SFSmartStore*) store
 status:(SFSyncStateStatus) status;
```

Android (Java)

```
public static List<SyncState> getSyncsWithStatus(SmartStore store, Status status)
```

## reSync を使用した増分同期

上位同期ターゲットおよび SOQL ベースの下位同期ターゲットの場合、事前定義された同期操作を増分更新できます。増分 `reSync` メソッドは、新規または更新されたレコードのみ、同期元との間でダウンロードまたはアップロードします。`reSync` をコールするときは、同期 ID または同期名を指定します。実行されたことがない同期設定に対して `reSync` がコールされた場合、`reSync` は、完全同期を実行するものと認識します。

`syncUp` と `syncDown`、または `reSync` のいずれを使用するかを決定するには、次の一般的なルールが役立ちます。

- 完全同期を実行するときはいつでも、`syncUp` または `syncDown` を使用できます。以前定義した同期を渡すことも、コールで同期を定義することもできます。
- `reSync` を使用できるのは、すでに定義されている同期を実行する場合のみです。最初の実行時、`reSync` は、完全同期を実行します。以降の同期では、`reSync` は、最後の実行と現在の状態の差分を返します。

Mobile SDK は、下位同期中にダウンロード済みのレコードから同期ターゲットで指定された変更日付項目をチェックし、最新のタイムスタンプを記憶します。その下位同期の再同期を要求する場合、Mobile SDK は最新のタイムスタンプ(利用可能な場合)を下位同期ターゲットに渡します。その後、下位同期ターゲットは、指定したタイムスタンプ以降に作成または更新されたレコードのみを取得します。デフォルトの変更日付項目は `lastModifiedDate` です。

組み込みの 3 つの下位同期ターゲット (MRU、SOQL ベース、および SOQL ベース) のうち、SOQL ベースの下位同期ターゲットでのみ `reSync` がサポートされています。カスタム下位同期ターゲットで `reSync` をサポートするには、取得操作中に渡される `maxTimeStamp` パラメーターを使用します。

**制限:** ストアにゴーストがあります。

増分下位同期後、次の未使用のレコードはローカルスープに保持されます。

- サーバー上で削除されたレコード
- 下位同期ターゲットを満たさなくなったレコード

これらの孤立したレコードを削除する場合、次の操作を実行できます。

- 完全下位同期操作を実行する
- `cleanResyncGhosts` メソッドのいずれかをコールする

## reSync メソッドの呼び出し

### iOS:

同期マネージャーインスタンスで、次をコールします。

#### Swift

```
var syncState =
 try syncManager.reSync(id: syncState.syncId, onUpdate: updateFunc)
```

#### Objective-C

```
- (nullable SFSyncState*) reSync:(NSNumber*) syncId
updateBlock:(SFSyncSyncManagerUpdateBlock) updateBlock;
```

### Android:

SyncManager インスタンスで、次をコールします。

#### Kotlin

```
suspend fun suspendReSync(syncId: Long): SyncState
```

#### Java

```
SyncState reSync(long syncId, SyncUpdateCallback callback);
```

### ハイブリッド:

次をコールします。

```
cordova.require("com.salesforce.plugin.MobileSync").reSync(syncId, successCB);
```

## サンプルアプリケーション

### iOS

MobileSyncExplorer サンプルアプリケーションは、SObjectDataManager クラスで `reSync()` を使用します。

### Android

MobileSyncExplorer サンプルアプリケーションは、ContactListLoader クラスで `reSync()` を使用します。

### ハイブリッド

SimpleSync サンプルアプリケーションは、SimpleSync.html の `app.views.SearchPage` クラスで `reSync()` を使用します。

## 下位同期

サーバーからローカルの Mobile Sync スープに sObject をダウンロードするには、適切な「下位同期」メソッドを使用します。

下位同期メソッドの場合、ダウンロードする sObject のリストを指定する同期ターゲットを定義します。明示的なリストを指定するには、Android では `JSONObject`、iOS では `NSDictionary` を使用します。ただし、クエリ文字列で同期ターゲットを定義することもできます。同期ターゲットクラスでは、SOSL、SOQL、または MRU クエリから同期ターゲットオブジェクトを作成するためのファクトリーメソッドが提供されます。

ダウンロードされたデータを受信する SmartStore スープの名前も指定します。このスープには、`__local__` という名前のインデックス付けされた文字列項目が必要です。Mobile SDK は、コールバックメソッドか、指定された更新ブロックを使用して同期操作の進行状況をレポートします。

## マージモード

下位同期メソッドは、受信データとローカルで変更されたレコードをマージする方法を制御するオプションをサポートしています。次のいずれかの動作を選択します。

1. 変更されたローカルレコードを上書きし、ローカルの変更をすべて消去する。options パラメーターを次の値に設定します。

- iOS:

Swift

```
SyncOptions.newSyncOptions(forSyncDown: SyncMergeMode.overwrite)
```

Objective-C

```
[SFSyncOptions newSyncOptionsForSyncDown:SFSyncStateMergeModeOverwrite]
```

- Android: `SyncOptions.optionsForSyncDown (MergeMode.OVERWRITE)`

2. すべてのローカルの変更とローカルで変更されたレコードを保持する。options パラメーターを次の値に設定します。

- iOS:

Swift

```
SyncOptions.newSyncOptions(forSyncDown: SyncMergeMode.leaveIfChanged)
```

Objective-C

```
[SFSyncOptions newSyncOptionsForSyncDown:SFSyncStateMergeModeLeaveIfChanged]
```

- Android: `SyncOptions.optionsForSyncDown (MergeMode.LEAVE_IF_CHANGED)`

### ❗ 重要:

- options パラメーターを取らないバージョンの `syncDown` を使用する場合、キャッシュ内にある既存の sObject が上書きされる可能性があります。ローカルの変更を保持するには、下位同期を実行する前に必ず上位同期を実行してください。
- ペイロードの下位同期を実行した場合、サーバー上で削除されたレコードは反映されません。このため、更新操作では、同期ターゲットスープ内の対応するレコードは自動的に削除されません。サー

バー上で削除された後にスープ内に残るこれらのレコードは、ゴーストと呼ばれます。これを削除するには、下位同期の後に `cleanResyncGhosts` メソッドのいずれかをコールします。

## iOS SyncManager メソッド

### Swift クラス名

```
SyncManager
```

### Objective-C クラス名

```
SFMobileSyncSyncManager
```

## 下位同期操作を作成して実行しない場合

### Swift

```
var syncState = syncManager.createSyncDown(target: target, options: options,
 soupName: CONTACTS_SOUP, syncName: syncState.syncName)
```

### Objective-C

```
- (SFSyncState *)createSyncDown:(SFSyncDownTarget *)target
 options:(SFSyncOptions *)options
 soupName:(NSString *)soupName
 syncName:(NSString *)syncName;
```

## ローカルの変更を上書きする下位同期操作を作成して実行する場合

### Swift

```
func syncDown(target: SFSyncDownTarget, soupName: String) -> SyncState
```

### Objective-C

```
- (SFSyncState*) syncDownWithTarget:(SFSyncDownTarget*)target
 soupName:(NSString*) soupName
 updateBlock:(SFSyncSyncManagerUpdateBlock)updateBlock;
```

## 名前のない下位同期操作を作成して実行する場合

### Swift

```
var syncState = syncManager.syncDown(target: target, soupName: CONTACTS_SOUP,
 onUpdate:updateFunc)
```

### Objective-C

```
- (SFSyncState*) syncDownWithTarget:(SFSyncDownTarget*)target
 options:(SFSyncOptions*)options
 soupName:(NSString*) soupName
 updateBlock:(SFSyncSyncManagerUpdateBlock)updateBlock;
```

## 名前付き下位同期操作を作成して実行する場合

### Swift

```
var syncState = try syncManager.syncDown(target: target, options: options,
 soupName: CONTACTS_SOUP, syncName: syncState.syncName, onUpdate:updateFunc)
```

**Objective-C**

```
- (SFSyncState*) syncDownWithTarget:(SFSyncDownTarget*)target
 options:(SFSyncOptions*)options
 soupName:(NSString*)soupName
 syncName:(NSString* __nullable)syncName
 updateBlock:(SFSyncSyncManagerUpdateBlock)updateBlock
 error:(NSError**)error;
```

**名前付き下位同期操作を実行する場合****Swift**

```
var syncState =
 try syncManager.reSync(named: syncState.syncName, onUpdate: updateFunc)
```

**Objective-C**

```
- (nullable SFSyncState*) reSyncByName:(NSString*)syncName
 updateBlock:(SFSyncSyncManagerUpdateBlock)updateBlock
 error:(NSError**)error;
```

**同期 ID を使用して以前の同期操作を再実行する場合****Swift**

```
var syncState =
 try syncManager.reSync(id: syncState.syncId, onUpdate: updateFunc)
```

**Objective-C**

```
- (nullable SFSyncState*) reSync:(NSNumber*)syncId
 updateBlock:(SFSyncSyncManagerUpdateBlock)updateBlock
 error:(NSError**)error;
```

**Android SyncManager メソッド****下位同期操作を作成して実行しない場合**

```
public SyncState createSyncDown(SyncDownTarget target,
 SyncOptions options, String soupName, String syncName)
 throws JSONException;
```

**ローカルの変更を上書きする下位同期操作を作成して実行する場合**

```
public SyncState syncDown(SyncDownTarget target, String soupName,
 SyncUpdateCallback callback) throws JSONException;
```

**名前のない下位同期操作を作成して実行する場合**

```
public SyncState syncDown(SyncDownTarget target, SyncOptions options,
 String soupName, SyncUpdateCallback callback)
 throws JSONException;
```

## 名前付き下位同期操作を作成して実行する場合

```
public SyncState syncDown(SyncDownTarget target, SyncOptions options,
 String soupName, String syncName, SyncUpdateCallback callback)
 throws JSONException;
```

## 名前付き同期設定を実行または再実行する場合

```
public SyncState reSync(String syncName, SyncUpdateCallback callback)
 throws JSONException;
```

## 同期 ID を使用して以前の同期操作を再実行する場合

```
public SyncState reSync(long syncId, SyncUpdateCallback callback)
 throws JSONException;
```

 例: iOS:

[MobileSyncExplorerSwift](#) サンプルアプリケーションは、Salesforce 取引先責任者オブジェクトで名前付き同期および同期設定ファイルを使用する方法を示します。iOS の場合、このサンプルでは Contact レコードを Swift オブジェクトとして表す `ContactSObjectData` クラスを定義します。サンプルでは、次の複数のサポートクラスも定義します。

- `ContactSObjectDataSpec`
- `SObjectData`
- `SObjectDataSpec`
- `SObjectDataFieldSpec`
- `SObjectDataManager`

Contact データを SmartStore スープと同期するために、このアプリケーションは `Resources/usersyncs.json` ファイルかの次の名前付き同期操作を定義します。

```
{
 "syncs": [
 {
 "syncName": "syncDownContacts",
 "syncType": "syncDown",
 "soupName": "contacts",
 "target": {"type": "soql", "query": "SELECT FirstName, LastName, Title,
 MobilePhone, Email, Department, HomePhone FROM Contact LIMIT 10000"},
 "options": {"mergeMode": "OVERWRITE"}
 },
 {
 "syncName": "syncUpContacts",
 "syncType": "syncUp",
 "soupName": "contacts",
 "target": {"createFieldlist": ["FirstName", "LastName", "Title",
 "MobilePhone", "Email", "Department", "HomePhone"]},
 "options": {"fieldlist": ["Id", "FirstName", "LastName", "Title",
 "MobilePhone", "Email", "Department", "HomePhone"],
 "mergeMode": "LEAVE_IF_CHANGED"}
 }
]
}
```

RootViewController クラスで、syncUpDown() メソッドは SObjectDataManager の updateRemoteData(\_:onFailure:) メソッドをコールしてフローを開始します。

```
func syncUpDown(){
 let alert = self.showAlert("Syncing", message: "Syncing with Salesforce")
 sObjectsDataManager.updateRemoteData({ [weak self] (sObjectsData) in
 DispatchQueue.main.async {
 alert.message = "Sync Complete!"
 alert.dismiss(animated: true, completion: nil)
 self?.refreshList()
 }
 }, onFailure: { [weak self] (error, syncState) in
 alert.message = "Sync Failed!"
 alert.dismiss(animated: true, completion: nil)
 self?.refreshList()
 })
}
```

syncUpDown は、成功を表す updateRemoteData の第1引数で、RootViewController の refreshList() メソッドをコールするブロックを渡します。このメソッドは、顧客の入力に従ってローカル取引先責任者を絞り込み、ビューを更新します。

updateRemoteData による上位同期の実行により、許可されたスूपの変更が Salesforce 組織にマージされます。上位同期が成功した場合、つまり、SyncState 状況が「done」の場合、updateRemoteData は次の処理を実行します。

1. 新規に更新されたスूपからすべての未加工データを取得します。
2. スूपのデータを ContactSObjectData オブジェクトに変換し、それらのオブジェクトを内部配列に保存します。
3. 制御を refreshRemoteData(\_:onFailure:) に渡します。refreshRemoteData に渡される onSuccess 引数は、syncUpDown から渡されるブロックです。

```
func updateRemoteData(_ onSuccess: @escaping ([SObjectData]) -> Void,
 onFailure:@escaping (NSError?, SyncState?) -> Void) -> Void
{
 ...
 try strongSelf.refreshRemoteData({ (sobjs) in
 onSuccess(sobjs)
 }, onFailure: { (error, syncState) in
 onFailure(error, syncState)
 })
 ...
}
```

refreshRemoteData で、アプリケーションは syncDownContacts モデル (別名 kSyncDownName) を使用して reSync を再度コールしてスूपを更新します。下位同期が成功した場合、refreshRemoteData は、updateRemoteData 経由で syncUpDown から渡されたブロックを実行して「サイクルを終了」します。

```
func refreshRemoteData(_ completion: @escaping ([SObjectData]) -> Void, onFailure:
@escaping (NSError?, SyncState) -> Void) throws -> Void {

 try self.syncMgr.reSync(named: kSyncDownName) { [weak self] (syncState) in
```



```

switch (syncState.status) {
case .done:
 do {
 let objects = try self?.queryLocalData()
 self?.populateDataRows(objects)
 completion(self?.fullDataRowList ?? [])
 } catch {
 MobileSyncLogger.e(SObjectDataManager.self,
 message: "Resync \(syncState.syncName) failed \(error)")
 }
 break
case .failed:
 MobileSyncLogger.e(SObjectDataManager.self,
 message: "Resync \(syncState.syncName) failed")
 onFailure(nil, syncState)
default:
 break
}
}
}

```

syncUpDown コールスタックで発生するすべての内容をまとめると次のようになります。

1. **上位同期:** SObjectsDataManager で updateRemoteData をコールして、スープの変更をサーバーに上位同期します。このステップにより、許可されるローカルおよびオフラインのすべての変更が Salesforce にマージされます。
2. **下位同期:** スープレコードがサーバーデータとマージされたら、refreshRemoteData へのコールでサーバーデータをスープに下位同期します。このステップにより、サーバーで発生した変更と上位同期でマージされた変更がスープに反映されます。注意: 上位同期のマージモードによって、サーバーで許可されるスープの編集が決まります。
3. 最後に、スープの更新された Contact レコードで UI を更新します。

**❗ 重要:** レコードを同期するときは必ず、MobileSyncExplorerSwift サンプルアプリケーションで示されているように、上位同期の後に下位同期をコールします。

#### 例: Android:

MobileSyncExplorer ネイティブサンプルアプリケーションは、Contact レコードに Mobile Sync 名前付き同期および同期設定ファイルを使用する方法を示します。Android の場合、このサンプルでは Salesforce Contact レコードを Java オブジェクトとして表す ContactObject クラスを定義します。Contact データを SmartStore スープと下位同期するために、syncDown() メソッドは SOQL クエリを定義する名前付き下位同期設定を再同期します。

次のスニペットでは、reSync() メソッドは res/raw/usersyncs.json ファイルから次の名前付き同期操作を読み込みます。

```

{
 "syncs": [
 {
 "syncName": "syncDownContacts",
 "syncType": "syncDown",
 "soupName": "contacts",

```

```

 "target": {"type": "soql", "query": "SELECT FirstName, LastName, Title,
 MobilePhone, Email, Department, HomePhone FROM Contact LIMIT 10000",
 "maxBatchSize": 500},
 "options": {"mergeMode": "OVERWRITE"}
 },
 {
 "syncName": "syncUpContacts",
 "syncType": "syncUp",
 "soupName": "contacts",
 "target": {"createFieldlist": ["FirstName", "LastName", "Title", "MobilePhone",
 "Email", "Department", "HomePhone"]},
 "options": {"fieldlist": ["Id", "FirstName", "LastName", "Title", "MobilePhone",
 "Email", "Department", "HomePhone"], "mergeMode": "LEAVE_IF_CHANGED"}
 }
]
}

```

下位同期操作が成功した場合、つまり `sync.getStatus()` が `Status.DONE` の場合は、受信したデータが指定されたスープに取り込まれます。その後、コールバックメソッドは、Contact リストのデータを再読み込みするインテントを起動します。

```

public synchronized void syncDown() {

 try {
 syncMgr.reSync(SYNC_DOWN_NAME /* see usersyncs.json */, new SyncUpdateCallback() {
 @Override
 public void onUpdate(SyncState sync) {
 if (Status.DONE.equals(sync.getStatus())) {
 fireLoadCompleteIntent();
 }
 }
 });
 } catch (JSONException e) {
 Log.e(TAG, "JSONException occurred while parsing", e);
 } catch (MobileSyncException e) {
 Log.e(TAG, "MobileSyncException occurred while attempting to sync down", e);
 }
}

```

## 上位同期

ローカルの変更をサーバーに適用するには、次のいずれかの「上位同期」メソッドを使用します。これらのメソッドはサーバーを特定の SmartStore スープのデータで更新します。スープ内にある作成、更新、または削除されたレコードが検索され、それらの変更がサーバー上で複製されます。`options` 引数には更新する項目のリストを指定します。Mobile SDK 5.1 以降では、作成操作と更新操作に個別の項目リストを使用して同期マネージャーオブジェクトを初期化し、この項目リストを上書きできます。[「作成操作と更新操作での項目リストの処理」](#)を参照してください。

ローカルに作成されたオブジェクトには、sObjectの型を指定する「type」項目を含む「attributes」項目を含める必要があります。たとえば、Acmeという名前の取引先の場合、`{Id:"local_x", Name: Acme, attributes: {type:"Account"}}`を使用します。

## iOS: SFMobileSyncSyncManager メソッド

- 名前付き同期を実行せずに作成できます。

### Swift

```
var syncState = syncManager.createSyncUp(target: target, options: options,
 soupName: CONTACTS_SOUP, syncName: syncState.syncName)
```

### Objective-C

```
- (SFSyncState *)createSyncUp:(SFSyncUpTarget *)target
 options:(SFSyncOptions *)options
 soupName:(NSString *)soupName
 syncName:(NSString *)syncName;
```

- デフォルトの同期ターゲットを使用するオプションのみを使用して同期を作成して実行できます。

### Swift

```
var syncState = syncManager.syncUp(options: options, soupName: CONTACTS_SOUP,
 onUpdate: updateFunc)
```

### Objective-C

```
- (SFSyncState*) syncUpWithOptions:(SFSyncOptions*)options
 soupName:(NSString*) soupName
 updateBlock:(SFSyncSyncManagerUpdateBlock)updateBlock;
```

- コード内で設定する同期ターゲットに基づいて同期を作成して実行できます。

### Swift

```
var syncState = syncManager.syncUp(options: options, soupName: CONTACTS_SOUP,
 onUpdate: updateFunc)
```

### Objective-C

```
- (SFSyncState*) syncUpWithTarget:(SFSyncUpTarget*)target
 options:(SFSyncOptions*)options
 soupName:(NSString*) soupName
 updateBlock:(SFSyncSyncManagerUpdateBlock)updateBlock;
```

- または、名前付き同期を作成して実行できます。同じ名前の同期を設定ファイルから読み込んだ場合、この同期がそれを上書きします。

### Swift

```
var syncState =
 try syncManager.syncUp(target: target, options: options,
 soupName: CONTACTS_SOUP, syncName: syncState.syncName,
 onUpdate: updateFunc)
```

## Objective-C

```
- (SFSyncState*) syncUpWithTarget:(SFSyncUpTarget*) target
 options:(SFSyncOptions*) options
 soupName:(NSString*) soupName
 syncName:(NSString*) syncName
 updateBlock:(SFSyncSyncManagerUpdateBlock) updateBlock
 error:(NSError**) error;
```

- 既存の名前付き同期設定を実行または再実行する場合

## Swift

```
open func reSync(named syncName: String,
 onUpdate updateBlock: @escaping SyncUpdateBlock) throws -> SyncState
```

## Objective-C

```
- (nullable SFSyncState*) reSyncByName:(NSString*) syncName
 updateBlock:(SFSyncSyncManagerUpdateBlock) updateBlock
 error:(NSError**) error;
```

- または、同期 ID を使用して以前の同期操作を再実行する場合

## Swift

```
open func reSync(id syncId: NSNumber,
 onUpdate updateBlock: @escaping SyncUpdateBlock) throws -> SyncState
```

## Objective-C

```
- (nullable SFSyncState*) reSync:(NSNumber*) syncId
 updateBlock:(SFSyncSyncManagerUpdateBlock) updateBlock
 error:(NSError**) error;
```

- 進行状況の最新情報を取得せずに同期を再実行するには、Mobile Sync Swift 拡張の次の関数を使用します。

## Swift

```
public func reSyncWithoutUpdates(named syncName: String,
 _ completionBlock: @escaping (Result<SyncState, MobileSyncError>) -> Void)
```

## Objective-C

(利用できません。)

- 外部 ID で上位同期を行うには、「[外部 ID による上位同期](#)」を参照してください。

## Android: SyncManager メソッド

- 名前付き上位同期設定を実行せずに作成できます。

```
public SyncState createSyncUp(SyncUpTarget target,
 SyncOptions options,
 String soupName,
 String syncName)
 throws JSONException;
```

- 名前のない上位同期設定を作成して実行できます。

```
public SyncState syncUp(SyncUpTarget target,
 SyncOptions options,
 String soupName,
 SyncUpdateCallback callback)
 throws JSONException;
```

- 名前付き上位同期設定を作成して実行できます。

```
public SyncState syncUp(SyncUpTarget target,
 SyncOptions options,
 String soupName,
 String syncName,
 SyncUpdateCallback callback)
 throws JSONException;
```

- 既存の名前付き同期設定を実行または再実行する場合

```
public SyncState reSync(String syncName, SyncUpdateCallback callback)
 throws JSONException;
```

- または、同期 ID を使用して以前の同期操作を再実行する場合

```
public SyncState reSync(long syncId, SyncUpdateCallback callback)
 throws JSONException;
```

- 外部 ID で上位同期を行うには、「[外部 ID による上位同期](#)」を参照してください。

## マージモードの指定

上位同期操作の場合、mergeMode オプションを指定できます。次のいずれかの動作を選択できます。

1. サーバーレコードが該当のクライアントに下位同期してから変更された場合でも、サーバーレコードを上書きします。syncUp メソッドをコールすると、次のようになります。

- **iOS:** options パラメーターを次のように設定します。

### Swift

```
SyncOptions.newSyncOptions(forSyncUp: ["Name"], mergeMode: SyncMergeMode.overwrite)
```

### Objective-C

```
[SFSyncOptions newSyncOptionsForSyncUp: ["Name"],
 mergeMode:SFSyncStateMergeModeOverwrite]
```

- **Android:** options パラメーターが SSyncOptions.optionsForSyncUp(fieldlist, SyncState.MergeMode.OVERWRITE) に設定される
- **ハイブリッド:** syncOptions パラメーターが {mergeMode:"OVERWRITE"} に設定される

2. サーバーレコードが該当のクライアントに下位同期してから変更された場合、現在の状態のまま保持されます。対応するクライアントレコードも現在の状態のまま保持されます。syncUp() メソッドをコールすると、次のようになります。

- **iOS:** options パラメーターを次のように設定します。

**Swift**

```
SyncOptions.newSyncOptions(forSyncUp: ["Name"], mergeMode:
SyncMergeMode.leaveIfChanged)
```

**Objective-C**

```
[SFSyncOptions newSyncOptionsForSyncUp:fieldlist
mergeMode:SFSyncStateMergeModeLeaveIfChanged]
```

- **Android:** options パラメーターが SyncOptions.optionsForSyncUp(fieldlist, SyncState.MergeMode.LEAVE\_IF\_CHANGED) に設定される
- **ハイブリッド:** syncOptions パラメーターが {mergeMode:"LEAVE\_IF\_CHANGED"} に設定される

ローカルレコードに同期ターゲットの変更日付項目が含まれている場合、Mobile SDK は、その項目とサーバーレコードの対応項目を比較して変更を検出します。デフォルトの変更日付項目は lastModifiedDate です。ローカルレコードに変更日付項目が含まれていない場合、LEAVE\_IF\_CHANGED 上位同期操作が上書きモードの上位同期に戻ります。

 **重要:** LEAVE\_IF\_CHANGED マージでは、サーバーへの追加の往復処理が必要になります。さらに重要なのは、状況チェック操作とレコード保存操作が連続した2つのコールで発生することです。まれに、これらのコールの間に更新されるレコードがサーバーで時期尚早に変更される場合があります。

 **例: iOS:**

[MobileSyncExplorerSwift](#) サンプルアプリケーションは、Salesforce 取引先責任者オブジェクトで名前付き同期および同期設定ファイルを使用する方法を示します。iOS の場合、このサンプルでは Contact レコードを Swift オブジェクトとして表す ContactSObjectData クラスを定義します。サンプルでは、次の複数のサポークラスも定義します。

- ContactSObjectDataSpec
- SObjectData
- SObjectDataSpec
- SObjectDataFieldSpec
- SObjectDataManager

Contact データを SmartStore スープと同期するために、このアプリケーションは Resources/usersyncs.json ファイルかの次の名前付き同期操作を定義します。

```
{
 "syncs": [
 {
 "syncName": "syncDownContacts",
 "syncType": "syncDown",
 "soupName": "contacts",
 "target": {"type":"soql", "query":"SELECT FirstName, LastName, Title,
 MobilePhone, Email, Department, HomePhone FROM Contact LIMIT 10000"},
 "options": {"mergeMode":"OVERWRITE"}
 },
 {
 "syncName": "syncUpContacts",
 "syncType": "syncUp",
 "soupName": "contacts",
```

```

 "target": {"createFieldlist":["FirstName", "LastName", "Title",
 "MobilePhone", "Email", "Department", "HomePhone"]},
 "options": {"fieldlist":["Id", "FirstName", "LastName", "Title",
 "MobilePhone", "Email", "Department", "HomePhone"],
 "mergeMode":"LEAVE_IF_CHANGED"}
 }
}
}

```

RootViewController クラスで、syncUpDown() メソッドは SObjectDataManager の updateRemoteData(\_:onFailure:) メソッドをコールしてフローを開始します。

```

func syncUpDown() {
 let alert = self.showAlert("Syncing", message: "Syncing with Salesforce")
 sObjectsDataManager.updateRemoteData({ [weak self] (sObjectsData) in
 DispatchQueue.main.async {
 alert.message = "Sync Complete!"
 alert.dismiss(animated: true, completion: nil)
 self?.refreshList()
 }
 }, onFailure: { [weak self] (error, syncState) in
 alert.message = "Sync Failed!"
 alert.dismiss(animated: true, completion: nil)
 self?.refreshList()
 })
}

```

syncUpDown は、成功を表す updateRemoteData の第1引数で、RootViewController の refreshList() メソッドをコールするブロックを渡します。このメソッドは、顧客の入力に従ってローカル取引先責任者を絞り込み、ビューを更新します。

updateRemoteData は、syncUpContacts モデル(ここでは別名 kSyncUpName)を使用して reSync をコールします。上位同期により、許可されたスプーの変更が Salesforce 組織にマージされます。

```

func updateRemoteData(_ onSuccess:@escaping ([SObjectData]) -> Void,
 onFailure:@escaping (NSError?, SyncState) -> Void) -> Void {
 do {
 try self.syncMgr.reSync(named: kSyncUpName) { [weak self] (syncState) in
 guard let strongSelf = self else {
 return
 }
 switch (syncState.status) {
 case .done:
 do {
 let objects = try strongSelf.queryLocalData()
 strongSelf.populateDataRows(objects)
 try strongSelf.refreshRemoteData({ (sobjs) in
 onSuccess(sobjs)
 }, onFailure: { (error, syncState) in
 onFailure(error, syncState)
 })
 } catch let error as NSError {
 MobileSyncLogger.e(SObjectDataManager.self,
 message: "Error with Resync \(error)")
 }
 }
 }
 }
}

```



```

 onFailure(error, syncState)
 }
 break
case .failed:
 MobileSyncLogger.e(SObjectDataManager.self,
 message: "Resync \(syncState.syncName) failed")
 onFailure(nil, syncState)
 break
default:
 break
}
}
} catch {
 onFailure(error as NSError, nil)
}
}

```

上位同期が成功した場合、つまり、SyncState 状況が「done」の場合、いくつかの処理が実行されます。

1. queryLocalData は、新規に更新されたスープからすべての未加工データを取得します。

```
let objects = try strongSelf.queryLocalData()
```

2. populateDataRows は、スープのデータを ContactSObjectData オブジェクトに変換し、それらのオブジェクトを内部配列に保存します。

```
strongSelf.populateDataRows(objects)
```

3. 制御が refreshRemoteData(\_:onFailure:) に渡されます。refreshRemoteData メソッドは updateRemoteData と似ていますが、次の 2 つの例外があります。

- 上位同期ではなく下位同期を実行する。
- 下位同期が成功した場合、updateRemoteData 経由で syncUpDown から渡されたブロックを実行して「サイクルを終了」する。

syncUpDown コールスタックで発生するすべての内容をまとめると次のようになります。

1. **上位同期:** SObjectsDataManager で updateRemoteData をコールして、スープの変更をサーバーに上位同期します。このステップにより、許可されるローカルおよびオフラインのすべての変更が Salesforce にマージされます。
2. **下位同期:** スープレコードがサーバーデータとマージされたら、refreshRemoteData へのコールでサーバーデータをスープに下位同期します。このステップにより、サーバーで発生した変更と上位同期でマージされた変更がスープに反映されます。注意: 上位同期のマージモードによって、サーバーで許可されるスープの編集が決まります。
3. 最後に、スープの更新された Contact レコードで UI を更新します。

**❗ 重要:** レコードを同期する場合、常に MobileSyncExplorerSwift サンプルアプリケーションで示される順序で上位同期/下位同期のペアが適用されます。

 例: Android:

サーバーに上位同期する場合は、`syncDown()` と同じ引数 (項目のリスト、同期元 SmartStore スープの名前、更新コールバック) を指定して `syncUp()` をコールします。コーディング上の違いは、影響を受ける項目のリストを `SyncTarget` ではなく `SyncOptions` のインスタンスとして形式設定できるという点のみです。次に、MobileSyncExplorer サンプルでのその処理方法を示します。

```
public synchronized void syncUp() {

 final SyncUpTarget target = new SyncUpTarget();

 final SyncOptions options =
 SyncOptions.optionsForSyncUp(Arrays.asList(ContactObject.CONTACT_FIELDS_SYNC_UP),

 MergeMode.LEAVE_IF_CHANGED);

 try {

 syncMgr.syncUp(target, options, ContactListLoader.CONTACT_SOUP,
 new SyncUpdateCallback() {

 @Override

 public void onUpdate(SyncState sync) {

 if (Status.DONE.equals(sync.getStatus())) {

 syncDown();

 }

 }

 });

 } catch (JSONException e) {

 Log.e(TAG, "JSONException occurred while parsing", e);

 } catch (MobileSyncException e) {

 Log.e(TAG, "MobileSyncException occurred while attempting to sync up", e);

 }

}
```

この例では、内部の `SyncUpdateCallback` 実装で、同期が完了したときに `syncDown()` をコールするという追加のステップを実行します。このステップにより、SmartStore スープにサーバーでの Contact への最近の変更がすべて反映され、最新の状態であることが保証されます。

## 外部 ID による上位同期

Mobile SDK は、外部 ID で上位同期を行う機能を追加して、上位同期機能を強化しています。この機能を使用するには、上位同期ターゲット定義で外部 ID 項目名を指定します。スープレコードがローカルで作成、更新、または削除されたものとしてマークされており、スープレコードに外部 ID 値がある場合、Mobile Sync は、`create` ではなく `upsert` を使用して上位同期を行います。ただし、レコードに有効な `Id` 値もある場合、Mobile Sync は、示された Salesforce レコードを更新します。

外部 ID 項目名は、Mobile Sync 設定ファイルまたはコードで設定できます。外部 ID の場合、標準の上位同期ターゲットと親子の上位同期ターゲットがサポートされます。

## 更新/挿入を使用した場合の利点

更新/挿入はレコードの重複を回避するために役立ちます。同期ターゲットで外部 ID 項目名がサポートされており、ローカルで作成されたレコードにその項目の値がある場合、Mobile Sync ではサーバーレコードが作成されるのではなく更新/挿入されます。

真の更新/挿入動作により、アプリケーションではネットワークを適切に切断できます。上位同期中にネットワークが切断された場合、アプリケーションでは通常、最後の要求がサーバーに到達したかどうかを認識できません。この場合、接続が回復したときのアプリケーションの論理的な応答は再同期することです。

ただし、サーバーで実際に要求が受信され、レコードが作成された場合にどうなるかを考慮してください。この場合、`upsert` を使用せずに再同期すると重複レコードが作成されます。上位同期のターゲットで外部 ID 項目名を設定している場合は、再同期を安全に実行できます。`upsert` によって最近作成されたレコードが検出され、単純にレコードの項目が既存の値で書き換えられます。

## 親子の上位同期に関する推奨事項

親子の関係があるシナリオでは、親、子、または両方に外部 ID 項目名を指定できます。適切なオフライン環境を構築するには、親および子の同期設定で外部 ID 項目名を定義するのが最善の方法です。

## 外部 ID 項目名の宣言的な設定

同期設定ファイルで同期ターゲットの外部 ID 項目名を宣言的に指定する例を次に示します。

### 標準の上位同期ターゲット

```
{
 "syncName": "myExampleSyncUp",
 "syncType": "syncUp",
 "soupName": "MySoup",
 "target": {
 "idFieldName": "Id",
 "modificationDateFieldName": "LastModifiedDate",
 "externalIdFieldName": "TheExternalId"
 },
 "options": {
 "fieldlist": ["Name", "Description"],
 "mergeMode": "OVERWRITE"
 }
}
```

## 親子の上位同期ターゲット

```
{
 "syncName": "parentChildrenSyncUp",
 "syncType": "syncUp",
 "soupName": "accounts",
 "target": {
 "iOSImpl": "SFParentChildrenSyncUpTarget",
 "androidImpl": "com.salesforce.androidsdk.mobilesync.target.ParentChildrenSyncUpTarget",

 "parent": {
 "idFieldName": "IdX",
 "externalIdFieldName": "ExternalIdX",
 "subjectType": "Account",
 "modificationDateFieldName": "LastModifiedDateX",
 "soupName": "accounts"
 },
 "createFieldlist": [
 "IdX",
 "Name",
 "Description"
],
 "updateFieldlist": [
 "Name",
 "Description"
],
 "children": {
 "parentIdFieldName": "AccountId",
 "idFieldName": "IdY",
 "externalIdFieldName": "ExternalIdY",
 "subjectType": "Contact",
 "modificationDateFieldName": "LastModifiedDateY",
 "soupName": "contacts",
 "subjectTypePlural": "Contacts"
 },
 "childrenCreateFieldlist": [
 "LastName",
 "AccountId"
],
 "childrenUpdateFieldlist": [
 "FirstName",
 "AccountId"
],
 "relationshipType": "MASTER_DETAIL"
 },
 "options": {
 "fieldlist": [
],
 "mergeMode": "LEAVE_IF_CHANGED"
 }
}
```

## 拡張例

さまざまな種類の同期設定を「1つにまとめた」例については、[github.com/forcedotcom/SalesforceMobileSDK-Shared](https://github.com/forcedotcom/SalesforceMobileSDK-Shared) GitHub リポジトリの [shared/example.usersyncs.json](#) を参照してください。

## 外部 ID 項目名のプログラムによる設定

### iOS

標準の上位同期ターゲット:

#### Swift

`externalIdFieldName` プロパティを `SyncUpTarget` または `BatchSyncUpTarget` クラスに設定します。

#### Objective-C

`externalIdFieldName` プロパティを `SFSyncUpTarget` または `SFBatchSyncUpTarget` クラスに設定します。

親子の上位同期ターゲット:

#### Swift

#### Objective-C

次のファクトリーメソッドの `externalIdFieldName` パラメーターに非 `nil` 値を渡します (Mobile SDK 9.0 の新機能)。

```
// From SFParentInfo.h
+ (SFParentInfo *)newWithSObjectType:(NSString *)sObjectType
 soupName:(NSString *)soupName
 idFieldName:(NSString *)idFieldName
 modificationDateFieldName:(NSString *)modificationDateFieldName
 externalIdFieldName:(NSString * __nullable) externalIdFieldName;

// From SFChildrenInfo.h
+ (SFChildrenInfo *)newWithSObjectType:(NSString *)sObjectType
 sObjectTypePlural:(NSString *)sObjectTypePlural
 soupName:(NSString *)soupName
 parentIdFieldName:(NSString *)parentIdFieldName
 idFieldName:(NSString *)idFieldName
 modificationDateFieldName:(NSString *)modificationDateFieldName
 externalIdFieldName:(NSString * __nullable) externalIdFieldName;
```

### Android

標準の上位同期ターゲット:

次のいずれかのコンストラクターを使用して同期ターゲットを作成します。

```
public SyncUpTarget(List<String> createFieldlist, List<String> updateFieldlist,
 String idFieldName, String modificationDateFieldName,
 String externalIdFieldName)
```

```
public BatchSyncUpTarget(List<String> createFieldlist, List<String> updateFieldlist,

 String idFieldName, String modificationDateFieldName,
 String externalIdFieldName)
```

親子の上位同期ターゲット:

次のファクトリーメソッドの `externalIdFieldName` パラメーターに非 `nil` 値を渡します (Mobile SDK 9.0 の新機能)。

```
// From ParentInfo.java
public ParentInfo(String objectType, String soupName, String idFieldName,
 String modificationDateFieldName, String externalIdFieldName)
```

```
// From ChildrenInfo.java
public ChildrenInfo(String objectType, String objectTypePlural, String soupName,
 String parentIdFieldName, String idFieldName,
 String modificationDateFieldName, String externalIdFieldName)
```

## 同期操作の停止と再開

Mobile SDK 7.1 以降、同期マネージャークラスでは、アプリケーションで同期を停止および再開できるメソッドが提供されます。同期が停止されても、一時停止されただけで後で再開できる場合があります。その他の場合、停止要求によって同期操作が強制的に失敗し、再開できなくなります。これらの新しい API では、Mobile SDK 7.1 の別の機能強化 (複数のアプリケーション間、またはアプリケーションとその拡張機能間でデータを安全に共有する機能) もサポートされています。

停止/再開 API では、複数のアプリケーション間、またはアプリケーションとその拡張機能間でデータを共有するための重要な保護が提供されます。データ共有アプリケーションは、複数のプロセスで同じストアに書き込まないように、バックグラウンドに移動するときに同期操作を一時停止する必要があります。

 **重要:** Mobile SDK は、アプリケーションがバックグラウンドやフォアグラウンドに移動するときに自動的に `stop` または `restart` をコールしません。

### stop/restart メソッドの説明

同期マネージャークラスの `stop/restart` メソッドは、次の同期操作を制御します。

- 上位同期
- 下位同期
- 再同期
- ゴーストのクリーンアップ/再同期

次のメソッドの一般的な説明は、サポートされているすべてのプラットフォームに適用されます。

#### stop

現在実行されているまたは実行のためにキューに追加されているすべての同期操作を一時停止するように同期マネージャークラスに要求します。下位同期、上位同期、ゴーストのクリーンアップ/再同期のタスクで停止要求が定期的にチェックされるようになりました。下位同期または上位同期のタスクで停止要求が検出されると、実行がすぐに停止されて、それらの状況が `stopped` に変わります。ゴーストのクリーンアップ/再同期のタスクで停止要求が検出されると、実行が停止されてエラーが報告されます。停止要求の後に送信されたタスクでは、すぐにエラーが返されます。

#### restart

- 同期マネージャークラスを再開します。`restart` がコールされると、停止された同期マネージャークラスは同期タスクを再度受け入れることができます。`restart` メソッドは、Boolean 引数 (`restartStoppedSyncs`) とコールバックブロックを取り、次のように動作します。

- `restartStoppedSyncs` が `true` の場合、`restart` は、停止しているすべての下位同期および上位同期のタスクで `reSync` をコールして更新をコールバックブロックに送信します。
- `restartStoppedSyncs` が `false` の場合、タスクは停止したままになります。その後、必要に応じて `reSync` をコールしてアプリケーション自体で上位同期および下位同期操作を再開できます。

**isStopping**

停止が要求されたが、一部のタスクは引き続き実行されている場合、`true` を返します。

**isStopped**

停止が要求されて、すべてのタスクの実行が停止している場合、`true` を返します。

**iOS**

停止/再開機能は、ネイティブiOSプラットフォームでのみ使用できます。次に、プラットフォーム固有の署名を示します。

| Swift クラス                | Objective-C クラス                      |
|--------------------------|--------------------------------------|
| <code>SyncManager</code> | <code>SFMobileSyncSyncManager</code> |

**Swift**

```
open func stop()
open func restart(restartStoppedSyncs: Bool,
 onUpdate updateBlock: @escaping SyncUpdateBlock) throws
open func isStopping() -> Bool
open func isStopped() -> Bool
```

**Objective-C**

```
- (void)stop;
- (BOOL)restart:(BOOL)restartStoppedSyncs
 updateBlock: (SFSyncSyncManagerUpdateBlock) updateBlock
 error: (NSError**)error;
- (BOOL)isStopping;
- (BOOL)isStopped;
```

**Android**

```
public synchronized void stop();
public synchronized void restart(boolean restartStoppedSyncs,
 SyncUpdateCallback callback);
public boolean isStopping();
public boolean isStopped();
```

**ハイブリッドおよび React Native について**

現在、`stop` および `restart` メソッドは、Cordova または React Native ブリッジで公開されません。これらの新しい API は、アプリケーションライフサイクルイベント (バックグラウンドやフォアグラウンドへの移動など)



に対応する場合に最適です。通常、アプリケーションはこの種の低レベル動作をネイティブコードで処理します。

### 一時停止された下位同期操作の効率的な再開

Mobile Sync は、最大タイムスタンプを記録します。この設定には、ダウンロードされたレコードの「最終更新」日が含まれる項目の最大値が反映されます。通常、Mobile SDK は同期操作の完了時にこの値を取得します。Mobile SDK 7.1 より前では、Mobile Sync はこの値を使用して、繰り返される `reSync` コールで取得されるレコード数を削減していました。

Mobile SDK 7.1 では、下位同期ターゲットの基本クラス `SyncDownTarget` (Android) および `SFSyncDownTarget` (iOS) で、サブクラスで実装できる新しいメソッド `isSyncDownSortedByLatestModification()` がサポートされています。このメソッドは Boolean 値を返します。true 場合、Mobile Sync にレコードのバッチを並び替えるように指示します。下位同期操作でレコードが並び替えられる場合、取得されるレコードの各バッチで同期状態オブジェクトの最大タイムスタンプが更新されます。最大タイムスタンプにより、下位同期が停止されて後で再開されても、以前のバッチが再取得されなくなります。

SOQL ベースの下位同期操作 `SoqlSyncDownTarget` (Android) および `SFSOqlSyncDownTarget` (iOS) の場合、並び替え動作は SOQL クエリによって異なります。クエリに `ORDER BY` 句が含まれていない場合、Mobile SDK は `ORDER BY last-modified-field` 句を追加し、`isSyncDownSortedByLatestModification()` は true を返します。

完了した下位同期操作の再同期と停止された下位同期操作の再同期では動作が若干異なります。

- 完了した下位同期で `reSync` が呼び出される場合、Mobile SDK はタイムスタンプが取得した最大タイムスタンプを超えるレコードのみを取得します。
- 停止された下位同期で `reSync` が呼び出される場合、Mobile SDK はタイムスタンプが取得した最大タイムスタンプ以上のレコードのみを取得します。同じタイムスタンプのレコードが複数のバッチにまたがることがあるため、このバリエーションが必要になります。

### 同期タスクエラーについて

Mobile SDK 7.1 以降の同期タスクメソッドは、プラットフォーム間で一貫性のあるプラットフォームエラー情報を返します。

同期タスク (下位同期、上位同期、再同期、ゴーストのクリーンアップ/再同期) は、次の理由で失敗することがあります。

- 無効な ID または名前
- 要求された同期操作がすでに実行されている
- 同期マネージャーが停止しているか、停止中である

アプリケーションにエラーの性質を通知するために、Mobile SDK 7.1 以降の同期メソッドは、次のようにエラー情報を返します。

- iOS:
  - Swift メソッドでは、`SyncState` オブジェクトが発生します。
  - Objective-C メソッドには、`NSError**` パラメーターがあります。

これらのメソッドの以前のバージョンは廃止されました。

- Android の場合、これらのメソッドでは `MobileSyncException` が発生します。

## Mobile SDK 7.1 の新しいおよび更新された同期/再同期メソッド

### 新しい iOS メソッド

#### Objective-C

次のメソッドで `NSError` パラメーターがサポートされるようになりました。

```
- (nullable SFSyncState*)
 syncDownWithTarget:(SFSyncDownTarget*)target
 options:(SFSyncOptions*)options
 soupName:(NSString*)soupName
 syncName:(nullable NSString*)syncName
 updateBlock:(SFSyncSyncManagerUpdateBlock)updateBlock
 error:(NSError**)error
;

- (nullable* SFSyncState*)
 syncUpWithTarget:(SFSyncUpTarget*)target
 options:(SFSyncOptions*)options
 soupName:(NSString*)soupName
 syncName:(nullable NSString*)syncName
 updateBlock:(SFSyncSyncManagerUpdateBlock)updateBlock
 error:(NSError**)error;

- (nullable* SFSyncState*)
 reSync:(NSNumber*)syncId
 updateBlock:(SFSyncSyncManagerUpdateBlock)updateBlock
 error:(NSError**) error;

- (nullable* SFSyncState*)
 reSyncByName:(NSString*)syncName
 updateBlock:(SFSyncSyncManagerUpdateBlock)updateBlock
 error:(NSError**)error NS_SWIFT_NAME(reSync(named:update:));
```

### 更新された iOS メソッド

#### Swift

次のメソッドで `SyncState` オブジェクトが発生するようになりました。

```
open func syncDown(target: SyncDownTarget,
 options: SyncOptions,
 soupName: String,
 syncName: String?,
 onUpdate updateBlock: @escaping SyncUpdateBlock) throws -> SyncState

open func syncUp(target: SyncUpTarget,
 options: SyncOptions,
 soupName: String,
 syncName: String?,
 onUpdate updateBlock: @escaping SyncUpdateBlock) throws -> SyncState

open func reSync(id syncId: NSNumber,
 onUpdate updateBlock: @escaping SyncUpdateBlock) throws -> SyncState
```

```
open func reSync(
 named syncName: String,
 onUpdate updateBlock: @escaping SyncUpdateBlock) throws -> SyncState
```

## Mobile SDK 7.1 の廃止された iOS メソッド

### Objective-C

NSError 出力パラメーターがサポートされていない既存の同期メソッドは、今後のメジャーリリースで削除される予定です。

```
- (nullable SFSyncState*)
 syncDownWithTarget:(SFSyncDownTarget*)target
 options:(SFSyncOptions*)options
 soupName:(NSString*) soupName
 syncName:(*nullable* NSString*) syncName
 updateBlock:(SFSyncSyncManagerUpdateBlock) updateBlock;

- (nullable SFSyncState*)
 syncUpWithTarget:(SFSyncUpTarget*) target
 options:(SFSyncOptions*) options
 soupName:(NSString*) soupName
 syncName:(*nullable* NSString*) syncName
 updateBlock:(SFSyncSyncManagerUpdateBlock) updateBlock;

- (nullable SFSyncState*)
 reSync:(NSNumber*) syncId
 updateBlock:(SFSyncSyncManagerUpdateBlock) updateBlock;

- (nullable SFSyncState*)
 reSyncByName:(NSString*) syncName
 updateBlock:(SFSyncSyncManagerUpdateBlock) updateBlock;
```

### 関連トピック

cleanResyncGhosts メソッドの変更については、「[下位同期操作後の「ゴースト」レコードの処理](#)」を参照してください。

## 下位同期操作後の「ゴースト」レコードの処理

下位同期操作で SmartStore スープに不要なレコードが残ることがある場合、cleanResyncGhosts API を使用して削除できます。

規定された特定のケースでは、最新の下位同期要求のコンテンツが SmartStore スープに正確に反映されません。たとえば、Salesforce サーバーでレコードが削除されても、次の下位同期操作で SmartStore からそのレコードが削除されません。また、同期条件を満たしていないレコードが下位同期結果から除外されても、自動的にスープから削除されません。SmartStore スープに予期せず残るこれらのレコードは、ゴーストと呼ばれます。

これらのゴーストを排除するために、MobileSync には、ゴーストを識別して削除する一連の cleanResyncGhosts メソッドが用意されています。同期オブジェクトの ID または名前を私、コールバックブロックを定義します。これらのメソッドは、Android ネイティブ、iOS ネイティブ、ハイブリッド、React Native プラットフォームで使用できます。

 **警告:** 練習では、`cleanResyncGhosts` メソッドの使用に制限があります。これらのメソッドへのコールは、ランタイムパフォーマンスとペイロードサイズの両方でコストかかる可能性があります。これらのメソッドは、各下位同期操作の一部としてではなく、頻度の低いクリーンアップ操作で使用します。特定の一連のゴーストに問題があり、ただちにクリーンアップが必要かどうかは、自分自身の判断で決定してください。

## カスタム下位同期ターゲットと `cleanResyncGhosts` の併用

アプリケーションでカスタム下位同期ターゲットを使用する場合、`cleanResyncGhosts` で `getListOfRemoteIds` メソッドを実装するカスタムの同期ターゲットが必要になります。このメソッドは、下位同期ターゲットの条件を満たす Salesforce ID のリストを返します。`getListOfRemoteIds` のコード例については、次の Mobile Sync ライブラリフォルダーで SOQL、SOSL、または MRU の下位同期ターゲットを確認してください。

### iOS

<https://github.com/forcedotcom/SalesforceMobileSDK-iOS/tree/master/libs/MobileSync/MobileSync/Classes/Util>

### Android

<https://github.com/forcedotcom/SalesforceMobileSDK-Android/tree/master/libs/MobileSync/src/com/salesforce/androidsdk/mobilesync/util>

## `cleanResyncGhosts` のスプの準備

対象のスプについて、次の項目にインデックスを追加します。

### `__sync_id__`

この項目によって、`cleanResyncGhosts()` メソッドが目的のスプ要素のみを削除するように指定できます。Mobile Sync はこの項目の内容を管理します。

## 同期 ID による `cleanResyncGhosts` メソッドのコール

### iOS ネイティブ

| Swift                    | Objective-C                          |
|--------------------------|--------------------------------------|
| <code>SyncManager</code> | <code>SFMobileSyncSyncManager</code> |

### Swift

Mobile SDK 7.1 では、例外が発生するように次のメソッドが更新されています。

```
open func cleanResyncGhosts(
 forId syncId: NSNumber,
 onComplete completionStatusBlock: @escaping SyncCompletionBlock) throws
```

### Objective-C

Mobile SDK 7.1 では、`NSError` 出力パラメーターをサポートするように次のメソッドが更新されています。

```
- (BOOL)
 cleanResyncGhosts:(NSNumber*) syncId
 completionStatusBlock:
```

```
(SFSyncSyncManagerCompletionStatusBlock) completionStatusBlock
error: (NSError**) error;
```

## Android ネイティブ

### Kotlin

```
suspend fun suspendCleanResyncGhosts(syncId: Long): Int
```

### Java

```
public void cleanResyncGhosts(final long syncId)
 throws JSONException, IOException

public void cleanResyncGhosts(long syncId,
 final CleanResyncGhostsCallback callback)
 throws JSONException
```

## ハイブリッド

```
cleanResyncGhosts(isGlobalStore, syncId, successCB, errorCB)

cleanResyncGhosts(storeConfig, syncId, successCB, errorCB)
```

## React Native

```
mobilesync.cleanResyncGhosts(isGlobalStore, syncId, successCB, errorCB)

mobilesync.cleanResyncGhosts(storeConfig, syncId, successCB, errorCB)
```

## 同期名による cleanResyncGhosts メソッドのコール

同期名で `cleanResyncGhosts` をコールすることもできます。

### iOS (Swift)

```
//Mobile Sync native Swift extension function
public func cleanGhosts(named syncName: String,
 _ completionBlock: @escaping (Result<UInt, MobileSyncError>) -> Void

//Objective-C function renamed for Swift
open func cleanResyncGhosts(
 forName syncName: String,
 onComplete completionStatusBlock: @escaping SyncCompletionBlock) throws

//iOS 13 or above only:
//Mobile Sync native Swift extension function, using Combine Publisher
public func cleanGhostsPublisher(for syncName: String) ->
 Future<UInt, MobileSyncError>
```

### iOS (Objective-C)

```
- (*BOOL*)
 cleanResyncGhostsByName: (NSString*) syncName
 completionStatusBlock:
 (SFSyncSyncManagerCompletionStatusBlock) completionStatusBlock
 error: (NSError**) error;
```

```
- (*BOOL*)
 cleanResyncGhosts:(NSNumber*) syncId
 completionStatusBlock:
 (SFSSyncSyncManagerCompletionStatusBlock) completionStatusBlock
 error:(NSError**) error;
```

**Android (Kotlin)**

```
suspend fun suspendCleanResyncGhosts(syncId: Long): Int
```

**Android (Java)**

```
public void cleanResyncGhosts(final String syncName, final CleanResyncGhostsCallback
callback)
```

**Mobile SDK 7.1 での廃止****廃止された iOS メソッド (Objective-C)**

NSError 出力パラメーターがサポートされていない次のメソッドは、今後のメジャーリリースで削除される予定です。

```
- (*BOOL*)
 cleanResyncGhosts:(NSNumber*) syncId
 completionStatusBlock:
 (SFSSyncSyncManagerCompletionStatusBlock) completionStatusBlock;
```

**標準ターゲットの使用**

Mobile Sync は、いくつかの標準 Salesforce 要求種別ですぐに使用できるターゲットクラスを用意しています。これらのターゲットは、設定ファイルを通じて暗黙的に使用することも、コードを通じて直接使用することもできます。

サポートされる標準種別を次に示します。

- Layout
- Metadata
- MRU
- SOQL query
- SOSL query
- Refresh
- Batch
- ブリーフケース
- sObject Collections

**MRU 下位同期ターゲットの使用**

組織内の特定の Salesforce オブジェクトのレコードのうち最後に表示されたレコードを取得するには、MRU 下位同期ターゲットを使用します。

MRU 下位同期ターゲットは、Salesforce API で決定された特定のオブジェクトのレコードのうち最後に表示されたレコードのみを返します。

## 設定ファイルの使用法

"target" プロパティには、次の値を指定します。

**target** プロパティ

"type": "mru"

"fieldList": <string> の配列  
同期する項目のリスト。

"objectType": <string>  
Salesforce オブジェクトの名前。

## iOS API

### Swift

クラス: MruSyncDownTarget

```
MruSyncDownTarget.newSyncTarget(objectType: String, fieldlist: [Any])
```

### Objective-C

クラス: SFMruSyncDownTarget

```
+ (SFMruSyncDownTarget*) newSyncTarget:(NSString*)objectType
fieldlist:(NSArray*)fieldlist;
```

## Android API

### Kotlin

クラス: MruSyncDownTarget

```
public MruSyncDownTarget(fieldlist: List<String>, objectType: String)
```

### Java

クラス: MruSyncDownTarget

```
public MruSyncDownTarget(List<String> fieldlist, String objectType)
```



例:

```
{
 "syncs": [
 {
 "syncName": "syncDownMruContacts",
 "syncType": "syncDown",
 "soupName": "contacts",
 "target": { "type": "mru",
 "fieldlist": ["FirstName", "LastName", "Title",
 "MobilePhone", "Email",
 "Department", "HomePhone"],
 "object": "Contact"}
 }
]
}
```



```

 "options": { "mergeMode": "OVERWRITE" }
 },
 {
 "syncName": "syncUpContacts",
 "syncType": "syncUp",
 "soupName": "contacts",
 "target": { "createFieldlist": ["FirstName", "LastName",
 "Title", "MobilePhone",
 "Email", "Department",
 "HomePhone"] },
 "options": { "fieldlist": ["Id", "FirstName", "LastName",
 "Title", "MobilePhone",
 "Email", "Department",
 "HomePhone"],
 "mergeMode": "LEAVE_IF_CHANGED" }
 }
]
}

```

## SOQL 下位同期ターゲットの使用

ビジネスニーズに対して必要なすべてのものを選択する SOQL クエリを定義できる場合、SOQL のターゲットは、最も簡単な下位同期オプションとなります。このターゲットは、SOQL クエリと、サポートされる省略可能な引数を取得します。

Mobile Sync は、REST 要求として指定された SOQL クエリをラップして Salesforce に送信します。

## 設定ファイルの使用方法

"target" プロパティには、次の値を指定します。

**target** プロパティ

**"type": "soql"**

**"query": <string>**

SOQL クエリ。

**"idFieldName": <string>**

(省略可能) カスタム ID 項目の名前。"idFieldName" を指定した場合、Mobile Sync は、指定された名前の項目を使用して、レコードの ID を取得します。たとえば、"idFieldName": "AcmeId" を指定した場合、Mobile Sync は、デフォルトの Id 項目ではなく、AcmeId 項目からレコードの ID を取得します。

**"modificationDateFieldName": <string>**

(省略可能) レコードの最終変更日付を含む項目の名前。modificationDateFieldName を指定した場合、Mobile Sync は、この名前の項目を使用して、startFetch がレコードの再同期のために使用する maxTimestamp 値を計算します。デフォルトの項目名は lastModifiedDate です。

**"maxBatchSize": <integer>**

(省略可能) それぞれの取得操作で取得する提案のレコード数。maxBatchSize 値が指定されている場合、Mobile Sync は、この値を使用して、それぞれの取得操作で返されるレコードの最大数を提案します。実際に取得されるレコード数は、指定した値より多くなることも少なくなることもあります。実行時の

バッチの実際のサイズは、パフォーマンスの問題、一致レコード数、またはクエリで指定された制限によって変化します。

## iOS API

### Swift

クラス: `SqlSyncDownTarget`

```
SqlSyncDownTarget.newSyncTarget(_ query:String) → Self
```

```
SqlSyncDownTarget.newSyncTarget(_ query:String, batchSize size:Int) → Self
```

### Objective-C

クラス: `SFSqlSyncDownTarget`

```
+ (SFSqlSyncDownTarget*) newSyncTarget:(NSString*)query;
+ (SFSqlSyncDownTarget*) newSyncTarget:(NSString*)query
 batchSize:(NSInteger) batchSize;
```

## Android API

### Kotlin

クラス: `SqlSyncDownTarget`

```
public fun SqlSyncDownTarget(query: String)
public fun SqlSyncDownTarget(idFieldName: String, modificationDateFieldName: String,
 query: String)
public fun SqlSyncDownTarget(idFieldName: String, modificationDateFieldName: String,
 query: String, batchSize: int)
```

### Java

クラス: `SqlSyncDownTarget`

```
public SqlSyncDownTarget(String idFieldName, String modificationDateFieldName,
 String query)
public SqlSyncDownTarget(String idFieldName, String modificationDateFieldName,
 String query, int batchSize)
```



例:

```
{
 "syncs": [
 {
 "syncName": "syncDownContacts",
 "syncType": "syncDown",
 "soupName": "contacts",
 "target": { "type": "sql", "query": "SELECT FirstName, LastName, Title,
 MobilePhone, Email, Department, HomePhone FROM Contact LIMIT 10000",
 "maxBatchSize": 500 },
 "options": { "mergeMode": "OVERWRITE" }
 },
 {
```

```

 "syncName": "syncUpContacts",
 "syncType": "syncUp",
 "soupName": "contacts",
 "target": { "createFieldlist": ["FirstName", "LastName", "Title", "MobilePhone",
 "Email", "Department", "HomePhone"] },
 "options": { "fieldlist": ["Id", "FirstName", "LastName", "Title", "MobilePhone",
 "Email", "Department", "HomePhone"], "mergeMode": "LEAVE_IF_CHANGED" }
 }
]
}

```

## SOSL 下位同期ターゲットの使用

Mobile Sync は、REST 要求として指定された SOSL クエリをラップして Salesforce に送信します。

### 設定ファイルの使用法

"target" プロパティには、次の値を指定します。

**target** プロパティ

"type": "sosl"

"query": <string>

SOSL クエリ。

"idFieldName": <string>

(省略可能) カスタム ID 項目の名前。"idFieldName" を指定した場合、Mobile Sync は、指定された名前の項目を使用して、レコードの ID を取得します。たとえば、"idFieldName": "AcmeId" を指定した場合、Mobile Sync は、デフォルトの Id 項目ではなく、AcmeId 項目からレコードの ID を取得します。

"modificationDateFieldName": <string>

(省略可能) レコードの最終変更日付を含む項目の名前。modificationDateFieldName を指定した場合、Mobile Sync は、この名前の項目を使用して、startFetch がレコードの再同期のために使用する maxTimestamp 値を計算します。デフォルトの項目名は lastModifiedDate です。

### iOS API

これらのファクトリーメソッドは、"query" プロパティを定義する SOSL 下位同期ターゲットを作成します。省略可能な "idFieldName" および "modificationDateFieldName" プロパティを指定するには、返されたターゲット上でそのスーパークラスメンバーを設定します。

#### Swift

クラス: SoslSyncDownTarget

```
SoslSyncDownTarget.newSyncTarget(_ query:String) -> Self
```

#### Objective-C

クラス: SFSoslSyncDownTarget

```
+ (SFSoslSyncDownTarget*) newSyncTarget:(NSString*)query;
```

## Android API

これらのファクトリーメソッドは、"query" プロパティを含む SOSL 下位同期ターゲットを作成します。省略可能な "idFieldName" および "modificationDateFieldName" プロパティを指定するには、返されたターゲット上でそのスーパークラスメンバーを設定します。

### Kotlin

クラス: `SoslSyncDownTarget`

```
public fun SoslSyncDownTarget(query: String)
```

### Java

クラス: `SoslSyncDownTarget`

```
public SoslSyncDownTarget(String query)
```

### 例:

```
{
 "syncs": [
 {
 "syncName": "syncDownAcme",
 "syncType": "syncDown",
 "soupName": "contacts",
 "target": {"type": "soql", "query": "FIND 'Acme' IN ALL FIELDS RETURNING
Account (Name),
 Contact (FirstName, LastName) "},
 "options": {"mergeMode": "OVERWRITE"}
 }
]
}
```

## ブリーフケース下位同期ターゲットの使用

組織でモバイルユーザー向けのブリーフケースを使用している場合、Mobile SDK 10.1 でブリーフケース下位同期ターゲットが導入されています。この同期ターゲットを使用すると、一度に多くのレコードを効率的に読み込むことができます。同期は、現在のユーザーに割り当てられたブリーフケースのうち、モバイルクライアントの接続アプリケーションでアクセス可能なすべてのブリーフケースに含まれるレコードに制限されます。これらのブリーフケースに含まれる特定のオブジェクトと項目に同期ターゲットを制限できます。このターゲットは、`sObject` 型、項目、およびそれらの追加先のスープが含まれる `BriefcaseObjectInfo` オブジェクトの配列を取得します。

 **ヒント:** SOQL 下位同期ターゲットでは一度に1つのレコードタイプのみを取得できます。また、親子の下位同期ターゲットでは、子を持つ親レコードの取得のみがサポートされます。これに対して、ブリーフケース下位同期ターゲットにはより高い柔軟性があります。ブリーフケースは、3レベルの深さまで、関連オブジェクトを含めさまざまな種類のオブジェクトを含めるように定義できます。同期ターゲット定義には、これらのオブジェクトの一部または全部を含めることができます。

ブリーフケースは、関連レコードのまとまりのあるコレクションを同時に選択する一連のクエリであり、現在のユーザーに合わせて最適化されています。ブリーフケースはブリーフケースビルダーを使用して事前に定義されますが、ブリーフケースクエリの実行は、ブリーフケースがクライアントアプリケーションによってアク

セスされるまで遅延します。ブリーフケースは、オフラインになる前のデータプライミングや他の一般的なデータ読み込みの目的でレコードを選択するために使用できます。

ブリーフケースは特定のユーザーに割り当てられ、クエリではその条件に現在のユーザーを含めることができます。ブリーフケースには、モバイルアプリケーションでモバイルクライアントの接続アプリケーションを介してアクセスできます。この同期ターゲットを介して使用できるオブジェクト、レコード、項目はこれらの設定の詳細によって自動的に制限されます。

また、ブリーフケースは組織のシステム管理者によって定義および管理されるため、システム管理者はコードを変更することなくクライアントアプリケーションの同期動作を変更することができます。

ブリーフケース同期ターゲットの Mobile Sync では、1つのセッションで大量のレコードを読み込むために最適化された Salesforce API が使用されます。特にオフラインになるための準備で数百または数千のレコードを一度に読み込む必要がある場合、このターゲットが他のオプションよりも効率的である場合があります。

## 設定ファイルの使用法

"target" プロパティには、次の値を指定します。

target プロパティ

"type": "briefcase"

"infos": <BriefcaseObjectInfo> 項目の配列

取得する特定の sObject と項目およびそれらを配置するスープを記述するオブジェクトの配列。

BriefcaseObjectInfo プロパティ

"soupName": <string>

同期中にこのオブジェクト種別のレコードを保存するスープの名前。

"objectType": <string>

同期する Salesforce オブジェクトの名前。

"fieldlist": <string> の配列

このオブジェクトの同期する項目のリスト。

"idFieldName": <string>

(省略可能) カスタム ID 項目の名前。"idFieldName" を指定した場合、Mobile Sync は、指定された名前項目を使用して、レコードの ID を取得します。たとえば、"idFieldName": "AcmeId" を指定した場合、Mobile Sync は、デフォルトの Id 項目ではなく、AcmeId 項目からレコードの ID を取得します。

"modificationDateFieldName": <string>

(省略可能) レコードの最終変更日付を含む項目の名前。modificationDateFieldName を指定した場合、Mobile Sync は、この名前項目を使用して、startFetch がレコードの再同期のために使用する maxTimestamp 値を計算します。デフォルトの項目名は lastModifiedDate です。

必須項目: "soupName"、"objectType"、"fieldlist"

## iOS API

必要に応じて BriefcaseObjectInfo オブジェクトを作成し、それを使用して下位同期ターゲットオブジェクトを作成します。

**Swift**

クラス: BriefcaseSyncDownTarget

```
let briefcaseAccountInfo = BriefcaseObjectInfo(
 soupName: "soup_for_accounts",
 objectType: "Account",
 fieldlist: ["Name", "Description"])
let briefcaseContactInfo = BriefcaseObjectInfo(
 soupName: "soup_for_contacts",
 objectType: "Contact",
 fieldlist: ["LastName"])
let target = BriefcaseSyncDownTarget(
 infos: [briefcaseAccountInfo, briefcaseContactInfo])
```

**Objective-C**

クラス: SFBriefcaseSyncDownTarget

```
SFBriefcaseObjectInfo *briefcaseAccountInfo = [[SFBriefcaseObjectInfo alloc]
 initWithSoupName:@"soup_for_accounts"
 objectType:@"Account"
 fieldlist:@[@"Name", @"Description"]];
SFBriefcaseObjectInfo *briefcaseContactInfo = [[SFBriefcaseObjectInfo alloc]
 initWithSoupName:@"soup_for_contacts"
 objectType:@"Contact"
 fieldlist:@[@"LastName"]];
SFBriefcaseSyncDownTarget *target = [[SFBriefcaseSyncDownTarget alloc]
 initWithInfos:[briefcaseAccountInfo, briefcaseContactInfo]];
```

**Android API**

必要に応じて BriefcaseObjectInfo オブジェクトを作成し、それを使用して下位同期ターゲットオブジェクトを作成します。

**Java**

クラス: BriefcaseSyncDownTarget

```
BriefcaseSyncDownTarget target = new BriefcaseSyncDownTarget(
 Arrays.asList(
 new BriefcaseObjectInfo(
 "soupForAccounts",
 "Account",
 Arrays.asList("Name", "Description")),
 new BriefcaseObjectInfo(
 "soupForContacts",
 "Contact",
 Arrays.asList("LastName"))
)
);
```

 **例:**

```
{
 "syncs": [
 {
```

```

"syncName": "myBriefcaseSyncDown",
"syncType": "syncDown",
"soupName": "does-not-matter",
"target": {
 "type": "briefcase",
 "infos": [
 {
 "subjectType": "Account",
 "fieldlist": [
 "Name",
 "Description"
],
 "soupName": "accounts"
 },
 {
 "subjectType": "Contact",
 "fieldlist": [
 "LastName"
],
 "soupName": "contacts"
 }
]
},
"options": {
 "mergeMode": "OVERWRITE"
}
}
]
}

```

## 関連トピック

- [カスタムターゲットでの下位同期メソッドの呼び出し](#)
- [「ブリーフケースビルダー」オンラインヘルプの「ブリーフケースの設定」](#)

## メタデータおよびレイアウトの同期

Mobile SDK 6.2 では、オブジェクトの検出と表現を簡略化する新しい API 機能が導入されています。これらの機能は、Mobile Sync の機能を利用して、Salesforce オブジェクトのメタデータとレイアウトにアクセスします。Mobile SDK は自動的にオフラインで使えるように定義済みスープにデータを保存し、簡単に照会できるように構造化されたデータモデルを保存します。

Mobile SDK のメタデータ API とレイアウト API を使用するとき、Mobile Sync は自動的に REST 要求を作成して送信し、アプリケーションにデータを返します。ただし、未加工の JSON 表現を返す代わりに、API はカスタムデータモデルオブジェクトで応答の形式を設定します。オフラインで使えるようにするため、これらの API は SmartStore スープにローカルに応答データを保存します。この機能自体によって必要なスープが作成され、内部下位同期ターゲットを使用してスープに入力されます。

この機能を使用するには、1つのメソッド(メタデータ用に1つ、レイアウト用に1つ)をコールして、応答のコールバックハンドラーを実装します。



## メタデータでできること

オブジェクトメタデータにより、要求された sObject またはカスタムオブジェクト種別の構造がわかります。メタデータを使用して、実行時に有効なクエリを作成する場合などに、アプリケーションでオブジェクトの項目を調査できます。または、応答から項目の設定プロパティ、リレーション、グラフ、URL、その他のオブジェクトデータを直接取得できます。内部的には、メタデータクラスは Salesforce の「[記述 \(describe\)](#)」API をコールします。

## レイアウトでできること

レイアウトでは、オブジェクトの表示ラベルと項目や検索結果の標準設定を定義する JSON 構造が提供されます。オブジェクトのデータを表示する画面の書式設定済み設計仕様として、この設定を使用できます。内部的には、Mobile SDK のレイアウト API は Salesforce の「[レコードレイアウト](#)」API をコールします。

## API の実装

Mobile SDK では、メタデータおよびレイアウトを同期するために iOS と Android のネイティブクラスのレベルをいくつか定義します。

- 同期マネージャークラスは、メタデータおよびレイアウトを取得するためにコールするメソッドを定義します。
  - iOS:

| Swift               | Objective-C           |
|---------------------|-----------------------|
| MetadataSyncManager | SFMetadataSyncManager |
| LayoutSyncManager   | SFLayoutSyncManager   |

- Android: MetadataSyncManager、LayoutSyncManager

同期マネージャークラスは、次の SmartStore スープを作成し、下位同期中にスープに入力します。

- sfdcMetadata
- sfdcLayouts

これらのスープは、効率的にオフラインで使用するためにインデックス付けされます。

- データモデルクラスは、コールバックブロックに返されるメタデータおよびレイアウトの構造化されたコンテンツです。
  - iOS:

| Swift    | Objective-C |
|----------|-------------|
| Metadata | SFMetadata  |
| Layout   | SFLayout    |

- Android: Metadata、Layout

- カスタム下位同期ターゲットクラスは、バックグラウンドで Salesforce クエリおよび SmartStore スープの同期を処理します。
  - iOS:

| Swift                  | Objective-C              |
|------------------------|--------------------------|
| MetadataSyncDownTarget | SFMetadataSyncDownTarget |
| LayoutSyncDownTarget   | SFLayoutSyncDownTarget   |

- Android: MetadataSyncDownTarget、LayoutSyncDownTarget

## iOS でのメタデータおよびレイアウトの同期

iOS でのメタデータおよびレイアウトの同期は簡単に使用できます。使用を開始するには、API の初期化と設定を行う方法を習得してください。

### メタデータおよびレイアウト同期マネージャーの初期化

iOS では、メタデータマネージャーとレイアウトマネージャーは両方とも共有オブジェクトです。それらにアクセスするには、次のように `sharedInstance` クラスをコールします。

#### Swift

```
MetadataSyncManager.sharedInstance()
LayoutSyncManager.sharedInstance()
```

#### Objective-C

```
[SFMetadataSyncManager sharedInstance]
[SFLayoutSyncManager sharedInstance]
```

この形式では、`sharedInstance` は現在のユーザーのログイン情報とデフォルトストアを使用してマネージャーを初期化します。

マルチユーザー環境の場合、現在のユーザーではないログインユーザーを使用してマネージャーを初期化することもできます。

#### Swift

```
MetadataSyncManager.sharedInstance(user)
LayoutSyncManager.sharedInstance(user)
```

#### Objective-C

```
[SFMetadataSyncManager sharedInstance:user]
[SFLayoutSyncManager sharedInstance:user]
```

ここで、`user` は `UserAccount` (Swift) または `SFUserAccount` (Objective-C) のインスタンスです。

ユーザーのデフォルト以外のストアを指定するには、次の形式を使用します。

**Swift**

```
MetadataSyncManager.sharedInstance(user, smartStore: store)
LayoutSyncManager.sharedInstance(user, smartStore: store)
```

**Objective-C**

```
[SFMetadataSyncManager sharedInstance:user smartStore:store]
[SFLayoutSyncManager sharedInstance:user smartStore:store]
```

マネージャーにデフォルトで現在のユーザーを設定するように指示するには、`user` を `null` に設定します。  
`store` 引数は `SmartStore` (Swift) または `SFSmartStore` (Objective-C) のインスタンスであり、特定のユーザーに関連付けられている必要があります。有効なストアが提供された場合、Mobile Sync はその特定のストアを使用して、そのメタデータおよびレイアウトスープを作成します。

**メタデータの取得 (iOS)**

メタデータマネージャーのクラスを使用して、Salesforce 組織または `SmartStore` インスタンスからメタデータを取得します。取得するには、メタデータ同期マネージャーの共有インスタンスで次の非同期メソッドをコールします。

**Swift**

```
MetadataSyncManager.sharedInstance().fetchMetadata(forObject: String,
 mode: FetchMode,
 completionBlock: MetadataSyncCompletionBlock)
```

**Objective-C**

```
- (void)fetchMetadataForObject:(nonnull NSString *)objectType
 mode: (SFSDKFetchMode)mode
 completionBlock:(nonnull SFMetadataSyncCompletionBlock) completionBlock;
```

**objectType**

取得するメタデータが含まれる Salesforce オブジェクト。たとえば、「Account」や「Contact」などです。

**mode**

このパラメーターは、データのソースの場所を決定するために役立ちます。次のようなデータ取得モードがあります。

- - iOS (Swift): `FetchMode.cacheOnly`
- iOS (Objective-C): `SFSDKFetchModeCacheOnly`
- Android: `CACHE_ONLY`

キャッシュからデータを取得します。キャッシュデータがない場合、`null` が返されます。

- - iOS (Swift): `FetchMode.cacheFirst`
- iOS (Objective-C): `SFSDKFetchModeCacheFirst`
- Android: `CACHE_FIRST`

キャッシュからデータを取得します。キャッシュデータがない場合、サーバーからデータを取得します。

- - iOS (Swift): `FetchMode.serverFirst`
- iOS (Objective-C): `SFSDKFetchModeServerFirst`

- Android: SERVER\_FIRST

サーバーからデータを取得します。サーバーデータがない場合、キャッシュからデータを取得します。サーバーから取得されたデータは、自動的にキャッシュされます。

#### completionBlock

操作の完了時に非同期に実行されるコールバックブロック。ブロックの実装または処理をこのパラメーターに渡します。このブロックは、次のメソッドプロトタイプを実装します。

```
typedef void (^SFMetadataSyncCompletionBlock) (SFMetadata * _Nullable metadata);
```

Mobile SDK は、メタデータオブジェクトをこのコールバックメソッドに渡します。このオブジェクトには、要求された Salesforce オブジェクトの適切なデータモデルが含まれます。このメタデータを使用して、特定の項目を照会できます。このクラスは、オブジェクトのマニフェストの項目名と名前が一致するプロパティを定義します。クラスのプロパティは、すべてのカスタム項目とカスタマイズ可能な標準項目を表します。

#### レイアウトの取得 (iOS)

レイアウトを同期するには、レイアウト同期マネージャーの共有インスタンスで次の非同期メソッドをコールします。

#### Swift

#### Objective-C

```
- (void)fetchLayoutForObjectAPIName:(nonnull NSString *)objectAPIName
 formFactor:(nullable NSString *)formFactor
 layoutType:(nullable NSString *)layoutType
 mode:(nullable NSString *)mode
 recordTypeId:(nullable NSString *)recordTypeId
 syncMode:(SFSDKFetchMode)syncMode
 completionBlock:(nonnull SFLayoutSyncCompletionBlock)completionBlock;
```

#### objectAPIName

(必須) 取得するレイアウトが含まれる Salesforce オブジェクト。たとえば、「Account」や「Contact」などです。

#### formFactor

(省略可能) 取得するレイアウトのフォーム要素。サポートされている値は、「Large」、「Medium」、および「Small」です。指定されていない場合、デフォルトの「Large」に設定されます。

#### layoutType

(省略可能) 取得するレイアウトタイプ。サポートされる値は、「Compact」および「Full」です。指定されていない場合、デフォルトの「Full」に設定されます。

#### mode

(省略可能) 取得するレイアウトのレコードモード。サポートされている値は、「Create」、「Edit」、および「View」です。指定されていない場合、デフォルトの「View」に設定されます。

#### recordTypeId

(省略可能) 取得するレコードタイプ。指定されていない場合は、デフォルトのレコードタイプが使用されます。

#### syncMode

データの取得中に使用する取得モード。サポートされている値は、次のとおりです。

- `SFSDKFetchModeCacheOnly` — キャッシュからデータを取得します。キャッシュデータがない場合、`null` が返されます。
- `SFSDKFetchModeCacheFirst` — キャッシュからデータを取得します。キャッシュデータがない場合、サーバーからデータを取得します。
- `SFSDKFetchModeServerFirst` — サーバーからデータを取得します。サーバーデータがない場合、キャッシュからデータを取得します。サーバーから取得されたデータは、自動的にキャッシュされます。

#### completionBlock

操作の完了時にトリガーされる非同期ブロック。ブロックの実装または処理をこのパラメーターに渡します。このブロックは、次のメソッドプロトタイプを実装します。

```
typedef void (^SFLayoutSyncCompletionBlock) (NSString * _Nonnull objectAPIName,
 NSString * _Nullable formFactor,
 NSString * _Nullable layoutType,
 NSString * _Nullable mode,
 NSString * _Nullable recordTypeId,
 SFLayout * _Nullable layout);
```

Mobile SDK は、`SFLayout` オブジェクトをこのコールバックメソッドに渡します。このオブジェクトには、要求された Salesforce オブジェクトのレイアウトの実際のデータモデルが含まれます。このオブジェクトのプロパティを使用して、特定の項目を照会できます。

バックグラウンドでは、`SFLayoutSyncManager` が Mobile Sync の `SFLayoutSyncDownTarget` オブジェクトを使用して、返されたデータを含んだ SmartStore スープを自動的に作成します。このスープには、デバイスのオフライン時にレイアウトデータを取得するためのインデックス指定が含まれています。

## Android でのメタデータおよびレイアウトの同期

Android でのメタデータおよびレイアウトの同期は簡単に使用できます。使用を開始するには、API の初期化と設定を行う方法を習得してください。

### MetadataSyncManager および LayoutSyncManager の初期化

Android では、メタデータマネージャーまたはレイアウトマネージャーにアクセスするには、次のように `getInstance()` 静的メソッドをコールします。

```
MetadataSyncManager.getInstance();
LayoutSyncManager.getInstance();
```

この形式では、`getInstance()` は現在のユーザーのログイン情報とデフォルトストアを使用してマネージャーを初期化します。

マルチユーザー環境の場合、次のようにコールして、現在のユーザーではないログインユーザーを使用してメタデータマネージャーを初期化することもできます。

```
MetadataSyncManager.getInstance(account);
LayoutSyncManager.getInstance(account);
```

この形式では、`account` は `UserAccount` のインスタンスです。この `getInstance()` メソッドは、特定のユーザーのログイン情報とデフォルトストアを使用してマネージャーを初期化します。

特定のユーザーが属するコミュニティを指定するには、次のようにコールして、その ID を提供します。

```
MetadataSyncManager.getInstance(account, id);
LayoutSyncManager.getInstance(account, id);
```

`account` が `null` の場合、マネージャーはデフォルトで現在のユーザーに設定されます。

特定のユーザーに関連付けられた名前付き `SmartStore` インスタンスを指定するには、次の形式を使用します。

```
MetadataSyncManager.getInstance(account, id, store);
LayoutSyncManager.getInstance(account, id, store);
```

マネージャーにコミュニティ設定を無視するように指示するには、`id` を `null` に設定します。`store` 引数は `SmartStore` のインスタンスであり、特定のユーザーに関連付けられている必要があります。有効なストアが提供された場合、Mobile Sync はその特定のストアを使用して、そのメタデータおよびレイアウトスナップを作成します。

## メタデータの取得

メタデータマネージャーのクラスを使用して、Salesforce 組織または `SmartStore` インスタンスからメタデータを取得します。取得するには、`MetadataSyncManager` インスタンスで次の非同期メソッドをコールします。

```
public void fetchMetadata(String objectType, Constants.Mode mode, MetadataSyncCallback
syncCallback);
```

### `objectType`

取得するメタデータが含まれる Salesforce オブジェクト。たとえば、「Account」や「Contact」などです。

### `mode`

このパラメーターは、データのソースの場所を決定するために役立ちます。次のようなデータ取得モードがあります。

- `CACHE_ONLY` (Android) または `SFSDKFetchModeCacheOnly` (iOS) - キャッシュからデータを取得します。キャッシュデータがない場合、`null` が返されます。
- `CACHE_FIRST` (Android) または `SFSDKFetchModeCacheFirst` (iOS) - キャッシュからデータを取得します。キャッシュデータがない場合、サーバーからデータを取得します。
- `SERVER_FIRST` (Android) または `SFSDKFetchModeServerFirst` (iOS) - サーバーからデータを取得します。サーバーデータがない場合、キャッシュからデータを取得します。サーバーから取得されたデータは、自動的にキャッシュされます。

### `syncCallback`

操作の完了時に実行される非同期ブロック。ブロックの実装または処理をこのパラメーターに渡します。このブロックでは、次の 1 つのメソッドを定義する `MetadataSyncCallback` インターフェースの形式を使用します。

```
void onSyncComplete(Metadata metadata);
```

Mobile SDK は、メタデータオブジェクトをこのコールバックメソッドに渡します。このオブジェクトには、要求された Salesforce オブジェクトの適切なデータモデルが含まれます。このメタデータを使用して、特定の項目を照会できます。このクラスは、オブジェクトのマニフェストの項目名と名前が一致するプロパティを定義します。クラスのプロパティは、すべてのカスタム項目とカスタマイズ可能な標準項目を表します。

## レイアウトの取得 (Android)

レイアウトを同期するには、`LayoutSyncManager` インスタンスで次の非同期メソッドをコールします。

```
public void fetchLayout(String objectAPIName, String formFactor,
 String layoutType, String mode, String recordTypeId,
 Constants.Mode syncMode, LayoutSyncCallback syncCallback);
```

### **objectAPIName**

(必須) 取得するレイアウトが含まれる Salesforce オブジェクト。たとえば、「Account」や「Contact」などです。

### **formFactor**

(省略可能) 取得するレイアウトのフォーム要素。サポートされている値は、「Large」、「Medium」、および「Small」です。指定されていない場合、デフォルトの「Large」に設定されます。

### **layoutType**

(省略可能) 取得するレイアウトタイプ。サポートされる値は、「Compact」および「Full」です。指定されていない場合、デフォルトの「Full」に設定されます。

### **mode**

(省略可能) 取得するレイアウトのレコードモード。サポートされている値は、「Create」、「Edit」、および「View」です。指定されていない場合、デフォルトの「View」に設定されます。

### **recordTypeId**

(省略可能) 取得するレコードタイプ。指定されていない場合は、デフォルトのレコードタイプが使用されます。

### **syncMode**

データの取得中に使用する取得モード。サポートされている値は、次のとおりです。

- `CACHE_ONLY` — キャッシュからデータを取得します。キャッシュデータがない場合、null が返されます。
- `CACHE_FIRST` — キャッシュからデータを取得します。キャッシュデータがない場合、サーバーからデータを取得します。
- `SERVER_FIRST` — サーバーからデータを取得します。サーバーデータがない場合、キャッシュからデータを取得します。サーバーから取得されたデータは、自動的にキャッシュされます。

- **syncCallback**

操作の完了時に非同期に実行されるコールバックメソッド。メソッドの実装または処理をこのパラメーターに渡します。このブロックは、次のメソッドプロトタイプを実装します。

```
void onSyncComplete(String objectAPIName, String formFactor, String layoutType,
 String mode, String recordTypeId, Layout layout);
```

Mobile SDK は、`SFLayout` オブジェクトをこのコールバックメソッドに渡します。このオブジェクトには、要求された Salesforce オブジェクトのレイアウトの実際のデータモデルが含まれます。このオブジェクトのプロパティを使用して、特定の項目を照会できます。

バックグラウンドでは、`LayoutSyncManager` が Mobile Sync の `LayoutSyncDownTarget` オブジェクトを使用して、返されたデータを含んだ `SmartStore` スープを自動的に作成します。このスープには、デバイスのオフライン時にレイアウトデータを取得するためのインデックス指定が含まれています。



## 更新下位同期ターゲットの使用

多くのアプリケーションでは、レコードをダウンロードしてそれらをすべてキャッシュし、接続が切断されるとユーザーは SmartStore キャッシュからレコードを編集できます。ローカルの「オフライン」作業はすばやく効率的で、操作性に優れていますが、接続が再開されたときには、キャッシュされたレコードをサーバーの更新情報によって更新することが重要です。

パフォーマンスと効率を最大化するために、MobileSDK では更新下位同期ターゲットが提供されます。更新ターゲットでは、準備のコーディングが不要な単一のコールがサポートされています。スープ名、オブジェクト種別、項目のリストを指定してターゲットのインスタンスを作成し、ターゲットのインスタンスを下位同期メソッドに渡すと、更新ターゲットは、関連するスープレコードの ID を収集し、サーバーに最新の項目値を照会してから、スープを更新します。

## 更新ターゲット API

更新下位同期ターゲットは、iOS および Android のネイティブアプリケーション、ReactNative アプリケーション、ハイブリッドアプリケーションで使用できます。

### iOS

クラス:

| Swift                              | Objective-C                          |
|------------------------------------|--------------------------------------|
| <code>RefreshSyncDownTarget</code> | <code>SFRefreshSyncDownTarget</code> |

ファクトリーメソッド:

### Swift

```
RefreshSyncDownTarget.newSyncTarget(soupName:objectType:fieldList:)
```

次に例を示します。

```
let refreshTarget = RefreshSyncDownTarget.newSyncTarget("MySoup", objectType: "Contact",
 fieldList: ["Id","Name"])
```

### Objective-C

```
+ (SFRefreshSyncDownTarget*) newSyncTarget:(NSString*) soupName
 objectType:(NSString*)objectType fieldlist:(NSArray*) fieldlist
```

### Android

クラス:

```
com.salesforce.androidsdk.mobilesync.util.RefreshSyncDownTarget
```

コンストラクター:

```
public RefreshSyncDownTarget(List<String> fieldlist,
 String objectType, String soupName)
```

### JavaScript (ハイブリッド、React Native)

## 関数

```
var target = {soupName:xxx, type:"refresh",
 objectType:yyy, fieldlist:["Id", ...]};
```

## 一括上位同期ターゲットの使用

大規模な上位同期操作のパフォーマンスを向上させるために、Mobile SDK 7.1 では、一括上位同期ターゲットが導入されています。

| iOS ネイティブ                                                                          | Android ネイティブ     |
|------------------------------------------------------------------------------------|-------------------|
| <b>Swift</b><br>BatchSyncUpTarget<br><br><b>Objective-C</b><br>SFBatchSyncUpTarget | BatchSyncUpTarget |

このターゲットは、Salesforce 複合 API をコールして、標準の上位同期ターゲットの動作を改善します。複合 API は、最大 25 件のローカルレコードをサーバーに一括送信します。

## iOS ネイティブ

### Swift

```
var target = BatchSyncUpTarget.init(createFieldList, updateFieldList)
let syncMgr = SyncManager.sharedInstance(store: self.store!)
syncMgr!.syncUp(target: target, options: options, soupName: soupName) {...}
```

### Objective-C

```
SFBatchSyncUpTarget* target = [[SFBatchSyncUpTarget alloc] init];
// or
SFBatchSyncUpTarget* target =
 [[SFBatchSyncUpTarget alloc] initWithCreateFieldlist:createList
 updateFieldlist:updateList];

[syncManager syncUpWithTarget:target
 options:options
 soupName:soupName
 updateBlock:updateBlock];
```

## Android ネイティブ

```
BatchSyncUpTarget target = new BatchSyncUpTarget();
// or
BatchSyncUpTarget target = new BatchSyncUpTarget(createFieldlist, updateFieldlist);
syncManager.syncUp(target, options, soupName, callback);
```

## ハイブリッドおよび React Native

ハイブリッドアプリケーションと React Native アプリケーションの既存の上位同期ターゲットは、自動的に一括上位同期ターゲットを使用します。別の上位同期ターゲットの実装 (従来の `SyncUpTarget` クラスなど) を使用するには、「`androidImpl`」または「`iOSImpl`」を指定します。

## 同期設定ファイルの使用方法

デフォルトでは、同期設定ファイルで定義された上位同期ターゲットは、一括 API を使用します。たとえば、次の同期設定では、一括上位同期ターゲットが作成されます。

```
{ "syncs": [
 {
 "syncName": "syncUpContacts",
 "syncType": "syncUp",
 "soupName": "contacts",
 "target": {
 "createFieldlist": [
 "FirstName",
 "LastName",
 "Title",
 "MobilePhone",
 "Email",
 "Department",
 "HomePhone"
],
 "options": {
 "fieldlist": [
 "Id",
 "FirstName",
 "LastName",
 "Title",
 "MobilePhone",
 "Email",
 "Department",
 "HomePhone"
],
 "mergeMode": "LEAVE_IF_CHANGED"
 }
 }
 }
]}
```

## 関連トピック

- [カスタムターゲットでの上位同期メソッドの呼び出し](#)
- [『REST API 開発者ガイド』の「Composite」](#)

## sObject コレクション上位同期ターゲットの使用

大規模な上位同期操作で最高のパフォーマンスが発揮されるように、Mobile SDK 10.1 で sObject コレクション上位同期ターゲットが導入されました。

| iOS ネイティブ                                                                                    | Android ネイティブ          |
|----------------------------------------------------------------------------------------------|------------------------|
| <b>Swift</b><br>CollectionSyncUpTarget<br><br><b>Objective-C</b><br>SFCollectionSyncUpTarget | CollectionSyncUpTarget |

このターゲットは、Salesforce の sObject コレクション API を使用して、標準の上位同期ターゲットの動作を改善します。この API は、最大 200 件のローカルレコードをサーバーに一括送信します。このターゲットは一括上位同期ターゲットよりも最大 5 倍速く、標準の単一レコード上位同期ターゲットよりも最大 10 倍速くなる可能性があります。

 **メモ:** 実際のパフォーマンスは、特定のレコード、sObject、ネットワーク条件に応じて異なる可能性があります。

## iOS ネイティブ

### Objective-C

```

SFCollectionSyncUpTarget* target = [[SFCollectionSyncUpTarget alloc] init];
// or
SFCollectionSyncUpTarget* target =
 [[SFCollectionSyncUpTarget alloc] initWithCreateFieldlist:createList
 updateFieldlist:updateList];

[syncManager syncUpWithTarget:target
 options:options
 soupName:soupName
 updateBlock:updateBlock];

```

## Android ネイティブ

```

CollectionSyncUpTarget target = new CollectionSyncUpTarget();
// or
CollectionSyncUpTarget target = new CollectionSyncUpTarget(createFieldlist, updateFieldlist);

syncManager.syncUp(target, options, soupName, callback);

```

## ハイブリッドおよび React Native

ハイブリッドアプリケーションと React Native アプリケーションの既存の上位同期ターゲットは、自動的に sObject コレクション上位同期ターゲットを使用します。

 **メモ:** この動作は一括上位同期ターゲットの自動使用を置き換えます。新しい sObject コレクションターゲットには後方互換性があるため、コードを変更しなくてもパフォーマンスが改善されます。

従来の SyncUpTarget クラスなど、別の上位同期ターゲットの実装を使用するには、そのクラスを「androidImpl」または「iOSImpl」設定で指定します。

## 同期設定ファイルの使用方法

デフォルトでは、同期設定ファイルで定義された上位同期ターゲットは、sObject コレクション API を使用します。たとえば、次の同期設定では、sObject コレクション上位同期ターゲットが作成されます。

```
{
 "syncs": [
 {
 "syncName": "syncUpContacts",
 "syncType": "syncUp",
 "soupName": "contacts",
 "target": {
 "createFieldlist": [
 "FirstName",
 "LastName",
 "Title",
 "MobilePhone",
 "Email",
 "Department",
 "HomePhone"
]
 },
 "options": {
 "fieldlist": [
 "Id",
 "FirstName",
 "LastName",
 "Title",
 "MobilePhone",
 "Email",
 "Department",
 "HomePhone"
],
 "mergeMode": "LEAVE_IF_CHANGED"
 }
 }
]
}
```

## 関連トピック

- [カスタムターゲットでの上位同期メソッドの呼び出し](#)
- [『REST API 開発者ガイド』の「sObject コレクション」](#)

## カスタム下位同期ターゲットの使用

下位同期操作中、下位同期ターゲットを使用して、ダウンロードするレコードセットや要求エンドポイントを制御します。事前書式設定済みの MRU ターゲット、SOQL ベースターゲット、および SOSL ベースターゲットを使用できます。または、カスタムターゲットを作成することもできます。カスタムターゲットでは、Salesforce 内外の任意の REST エンドポイントにアクセスできます。

## カスタム下位同期ターゲットの定義

下位同期ターゲットのプラットフォームの抽象基本クラスをサブクラス化して、下位同期操作のカスタムターゲットを定義します。ハイブリッドアプリケーションでカスタムターゲットを使用するには、サポートされている各プラットフォームのカスタムネイティブターゲットクラスを実装します。下位同期ターゲットの基本クラスは次のとおりです。

- iOS:

| Swift          | Objective-C      |
|----------------|------------------|
| SyncDownTarget | SFSyncDownTarget |

- Android: SyncDownTarget

 **メモ:** これらのクラスは、要求されたレコードを同期しますが、その関連レコードは同期しません。関連レコードを含めるには、「[関連レコードの同期](#)」に記載の同期ターゲットクラスを使用します。

### Required メソッド

各カスタムターゲットクラスでは、次の必須メソッドを実装する必要があります。

#### Start Fetch メソッド

下位同期操作を開始するために同期マネージャーによってコールされます。maxTimeStamp が 0 より大きい場合、この操作は「再同期」になります。その後、指定した時間以降に作成または更新されたレコードのみが返されます。

iOS:

Swift

```
func startFetch(syncManager: SyncManager,
 maxTimeStamp: Int64,
 errorCallback: SyncDownErrorBlock,
 complete: SyncDownCompleteBlock)
```

Objective-C

```
- (void) startFetch:(SFMobileSyncSyncManager*) syncManager
 maxTimeStamp:(long long) maxTimeStamp
 errorCallback:(SFSyncDownTargetFetchErrorBlock)
 errorCallback
 completeBlock:(SFSyncDownTargetFetchCompleteBlock)
 completeBlock;
```

Android:

```
JSONArray startFetch(SyncManager syncManager, long maxTimeStamp);
```

#### Continue Fetching メソッド

メソッドで null が返されるまで同期マネージャーによって繰り返しコールされます。このプロセスでは、同期が必要なすべてのレコードが取得されます。

**iOS:****Swift**

```
func continueFetch(syncManager: SyncManager,
 onFail: SyncDownErrorBlock,
 onComplete: SyncDownCompletionBlock?)
```

**Objective-C**

```
- (void)
 continueFetch:(SFMobileSyncSyncManager*) syncManager
 errorBlock:(SFSyncDownTargetFetchErrorBlock)
 errorBlock
 completeBlock:(SFSyncDownTargetFetchCompleteBlock)
 completeBlock;
```

**Android:**

```
JSONArray continueFetch(SyncManager syncManager);
```

**modificationDateFieldName プロパティ (省略可能)**

必要に応じて、カスタムクラスで `modificationDateFieldName` プロパティを上書きできます。

`modificationDateFieldName` を指定した場合、Mobile Sync は、この名前の項目を使用して、`startFetch` がレコードの再同期のために使用する `maxTimestamp` 値を計算します。デフォルトの項目名は `lastModifiedDate` です。

**iOS (Swift および Objective-C):**

`modificationDateFieldName` プロパティ

**Android:**

```
String getModificationDateFieldName();
```

**idFieldName プロパティ (省略可能)**

"`idFieldName`" を指定した場合、Mobile Sync は、指定された名前の項目を使用して、レコードの ID を取得します。たとえば、"`idFieldName`": "`AcmeId`" を指定した場合、Mobile Sync は、デフォルトの `Id` 項目ではなく、`AcmeId` 項目からレコードの ID を取得します。

**iOS (Swift および Objective-C):**

`idFieldName` プロパティ

**Android:**

```
String getIdFieldName();
```

**カスタムターゲットでの下位同期メソッドの呼び出し****iOS:**

カスタム `SFSyncDownTarget` クラスのインスタンスを `SFMobileSyncSyncManager` 下位同期メソッドに渡します。

**Swift**

```
func syncDown(target: SyncDownTarget,
 soupName: String,
 onUpdate: () -> SyncState
```



**Objective-C**

```
- (SFSyncState*)
 syncDownWithTarget: (SFSyncDownTarget*) target
 soupName: (NSString*) soupName
 updateBlock:
 (SFSyncSyncManagerUpdateBlock) updateBlock;
```

**Android:**

カスタム SyncDownTarget クラスのインスタンスを SyncManager 下位同期メソッドに渡します。

```
SyncState syncDown(SyncDownTarget target, SyncOptions options, String soupName,
 SyncUpdateCallback callback);
```

**ハイブリッド:**

1. 次のプロパティ設定を使用してターゲットオブジェクトを作成します。

- type を「custom」に設定する。
- 少なくとも次のいずれかのプロパティを設定する。

**iOS** (サポートされている場合):

iOS カスタムクラスの名前に iOSImpl を設定します。

**Android** (サポートされている場合):

Android カスタムクラスのパッケージ修飾名に androidImpl を設定します。

次の例は、iOS と Android の両方に対応しています。

```
var target =
{type:"custom",
 androidImpl:
 "com.salesforce.samples.notesync.ContentSoqlSyncDownTarget",
 iOSImpl:"SFContentSoqlSyncDownTarget",
 ...
};
```

2. このターゲットをハイブリッド下位同期メソッドに渡します。

```
cordova.require("com.salesforce.plugin.MobileSync").syncDown(target, ...);
```

**サンプルアプリケーション****iOS**

NoteSync ネイティブ iOS サンプルアプリケーションは、SFContentSoqlSyncDownTarget 下位同期ターゲットを定義および使用します。

**Android**

NoteSync ネイティブ Android サンプルアプリケーションは、

com.salesforce.samples.notesync.ContentSoqlSyncDownTarget 下位同期ターゲットを定義および使用します。

## カスタム上位同期ターゲットの使用

上位同期操作中、上位同期ターゲットを使用して、アップロードするレコードセットや、サーバーのレコードを更新する REST エンドポイントを制御します。カスタム上位同期ターゲットを作成して、Salesforce 内外の任意の REST エンドポイントにアクセスできます。

### カスタムの上位同期ターゲットの定義

上位同期ターゲットのプラットフォームの抽象基本クラスをサブクラス化して、上位同期操作のカスタムの同期ターゲットを定義します。ハイブリッドアプリケーションでカスタムの同期ターゲットを使用するには、サポートされている各プラットフォームに固有のカスタムの同期ターゲットクラスを実装する必要があります。上位同期ターゲットの基本クラスは次のとおりです。

- iOS:

| Swift        | Objective-C    |
|--------------|----------------|
| SyncUpTarget | SFSyncUpTarget |

- Android: SyncUpTarget

 **メモ:** これらのクラスは、要求されたレコードを同期しますが、その関連レコードは同期しません。関連レコードを含めるには、「[関連レコードの同期](#)」に記載の同期ターゲットクラスを使用します。

### 作成操作および更新操作での項目リストの処理

同期ターゲットの Create On Server メソッドおよび Update On Server メソッドは、引数リストで指定した項目のリストに対してのみ動作します。ただし、Salesforce オブジェクトは、アプリケーションで更新できない特定の項目を作成するように同期ターゲットに要求することがあります。このようなオブジェクトの場合、単一の項目リストを作成と更新の両方の操作に使用する同期ターゲットでロック項目を更新しようとすると、失敗する可能性があります。

作成操作と更新操作に異なる項目リストを指定するには、createFieldlist および updateFieldlist パラメーターをサポートするイニシャライザーメソッドを使用できます。これらの項目リストを提供するためだけにカスタムの同期ターゲットを定義する場合、この方法によってその手間を省くことができます。

#### iOS

次の SFSyncUpTarget init メソッドをコールします。

##### Swift

```
SyncUpTarget.init(createFieldlist: [Any]?, updateFieldlist: [Any]?)
```

次に例を示します。

```
SyncUpTarget.init(createFieldlist: nil, updateFieldlist: ["Name"])
```

##### Objective-C

```
- (instancetype)initWithCreateFieldlist:(NSArray *)createFieldlist
 updateFieldlist:(NSArray *)updateFieldlist
```

`createFieldlist` および `updateFieldlist` 引数を指定すると、該当する場合に同期ターゲットで使用されます。その場合、同期ターゲットは同期オプションオブジェクトで定義された項目リストを無視します。

#### Android

次の `SyncUpTarget` コンストラクターを使用します。

```
public SyncUpTarget(List<String> createFieldlist, List<String> updateFieldlist)
```

`createFieldlist` および `updateFieldlist` 引数を指定すると、該当する場合に同期ターゲットで使用されます。そのような場合、同期ターゲットでは `SyncOptions` オブジェクトで定義された項目リストが無視されます。

#### Required メソッド

各カスタムターゲットクラスでは、次の必須メソッドを実装する必要があります。

#### Create On Server メソッド

ローカルで作成されたレコードを上位同期します。ハイブリッドアプリケーションや React Native アプリケーションは、省略可能な `createFieldList` パラメーターと共に `syncUp` をコールして `fields` パラメーターを上書きできます。

iOS:

##### Swift

```
func createOnServer(syncManager: SyncManager,
 record: [AnyHashable : Any],
 fieldlist: [Any],
 onComplete: SyncUpcompletionBlock([AnyHashable : Any]?)
 -> Void,
 onFail: SyncUpErrorBlock(Error) -> Void)
```

##### Objective-C

```
- (void) createOnServer:(NSString*)objectType
 fields:(NSDictionary*)fields
 completionBlock:(SFSyncUpTargetCompleteBlock)
 completionBlock
 failBlock:(SFSyncUpTargetErrorBlock)failBlock;
```

Android:

```
String createOnServer(SyncManager syncManager,
 String objectType, Map<String, Object> fields);
```

#### Update On Server メソッド

ローカルで更新されたレコードを上位同期します。 `objectId` パラメーターの場合、MobileSync は、カスタムの同期ターゲットの `getIdFieldName()` メソッド (Android) または `idFieldName` プロパティ (iOS) で指定された項目を使用します。ハイブリッドアプリケーションや React Native アプリケーションは、省略可能な `updateFieldList` パラメーターと共に `syncUp` をコールして `fields` パラメーターを上書きできます。

iOS:

Swift

```
func updateOnServer(syncManager: SyncManager,
 record: [AnyHashable : Any],
 fieldlist: [Any],
 onComplete: SyncUpcompletionBlock([AnyHashable : Any]?)
 -> Void,
 onFail: SyncUpErrorBlock(Error) -> Void)
```

Objective-C

```
- (void) updateOnServer:(NSString*)objectType
 objectId:(NSString*)objectId
 fields:(NSDictionary*)fields
 completionBlock:(SFSyncUpTargetCompleteBlock)
 completionBlock
 failBlock:(SFSyncUpTargetErrorBlock) failBlock;
```

Android:

```
updateOnServer(SyncManager syncManager, String objectType, String objectId,
 Map<String, Object> fields);
```

**Delete On Server メソッド**

ローカルで削除されたレコードを上位同期します。objectId パラメーターの場合、MobileSync は、カスタムの同期ターゲットの getIdFieldName() メソッド (Android) または idFieldName プロパティ (iOS) で指定された項目を使用します。

iOS:

Swift

```
func deleteOnServer(syncManager: SyncManager,
 record: [AnyHashable : Any],
 fieldlist: [Any],
 onComplete: SyncUpcompletionBlock([AnyHashable : Any]?)
 -> Void,
 onFail: SyncUpErrorBlock(Error) -> Void)
```

Objective-C

```
- (void) deleteOnServer:(NSString*)objectType
 objectId:(NSString*)objectId
 completionBlock:(SFSyncUpTargetCompleteBlock)
 completionBlock
 failBlock:(SFSyncUpTargetErrorBlock) failBlock;
```

Android:

```
deleteOnServer(SyncManager syncManager, String objectType,
 String objectId);
```

**省略可能な設定の変更**

必要に応じて、カスタムクラスで次の値を上書きできます。

**getIdsOfRecordsToSyncUp**

上位同期で返されるレコード ID のリスト。デフォルトでは、これらのメソッドで `__local__` が `true` のすべてのレコードが返されます。

iOS:

Swift

```
func getIdsOfRecords(toSyncUp: SyncManager, soupName: String)
```

Objective-C

```
- (NSArray*)
getIdsOfRecordsToSyncUp: (SFMobileSyncSyncManager*) syncManager
 soupName: (NSString*) soupName;
```

Android:

```
Set<String> getIdsOfRecordsToSyncUp(SyncManager syncManager,
 String soupName);
```

**変更日付項目名**

レコードがリモートで変更されたかどうかを判断するために `LEAVE_IF_CHANGED` 上位同期操作中使用される項目。デフォルト値は `lastModifiedDate` です。

iOS (Swift および Objective-C):

`modificationDateFieldName` プロパティ

Android:

```
String getModificationDateFieldName();
```

**isNewerThanServer**

スープ要素が対応するサーバーレコードよりも新しいかどうかを判断します。

iOS:

Swift

```
func isNewerThanServer(syncManager: SyncManager,
 record: [AnyHashable : Any],
 resultBlock: RecordNewerThanServerBlock(Bool) -> Void)
```

Objective-C

```
- (void)isNewerThanServer: (SFMobileSyncSyncManager *) syncManager
 record: (NSDictionary*) record
 resultBlock: (SFSyncUpRecordNewerThanServerBlock) resultBlock;
```

Android:

```
public boolean isNewerThanServer(SyncManager syncManager,
 JSONObject record) throws JSONException, IOException
```

**ID 項目名**

レコードの ID を取得するために使用される項目。たとえば、Mobile SDK は、上位同期中に `updateOnServer()` メソッドに渡す ID をローカルレコードの `idFieldName` と名前が一致する項目から取得します。

iOS (Swift および Objective-C):

idFieldName プロパティ

Android:

String getIdFieldName();

## カスタムターゲットでの上位同期メソッドの呼び出し

iOS:

同期マネージャーインスタンスで、次をコールします。

Swift

```
func syncUp(target: SyncUpTarget,
 options: SyncOptions,
 soupName: String,
 onUpdate: () -> SyncState
```

次に例を示します。

```
let syncState = syncManager.syncUp(target: target, options: options,
 soupName: CONTACTS_SOUP, onUpdate: updateFunc)
```

Objective-C

```
- (SFSSyncState*)
 syncUpWithTarget:(SFSSyncUpTarget*)target
 syncOptions:(SFSSyncOptions*)options
 soupName:(NSString*)soupName
 updateBlock:
 (SFSSyncSyncManagerUpdateBlock)updateBlock;
```

Android:

SyncManager インスタンスで、次をコールします。

```
SyncState syncUp(SyncUpTarget target,
 SyncOptions options, String soupName,
 SyncUpdateCallback callback);
```

ハイブリッド:

```
cordova.require("com.salesforce.plugin.mobilesync").
 syncUp(isGlobalStore, target, soupName,
 options, successCB, errorCB);
cordova.require("com.salesforce.plugin.mobilesync").
 syncUp(storeConfig, target, soupName,
 options, successCB, errorCB);
```

## 関連レコードの同期

オフラインデータの同期は一般的な問題です。明示的な変更は簡単に同期できますが、影響を受ける関連レコードを更新するにはどうすればよいでしょうか? 十分な知識、決意、そして洞察力があれば手動で実行できますが、これは旧式な方法です。Mobile SDK 5.2 から、1 回のコールで親レコードと関連レコードを同期するツールが Mobile Sync に搭載されました。

 **重要:** 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

## サポートされるリレーション種別

関連レコードの同期では、参照と主従の2種類の一对多リレーションがサポートされます。どちらの種別も子から親、多から一の方向に定義されます。各子は親を認識しますが、親は子を認識しません。

### 参照関係

参照関係は、1つまたは複数の項目に基づくオブジェクト間の「緩い」つながりです。参照関係の場合:

- 子レコードに親項目は不要です。
- 1つのオブジェクトを変更しても、リンクされたオブジェクトのセキュリティ、アクセス、削除には影響しません。

Salesforce は、オブジェクトごとに最大25個の参照項目をサポートします。参照関係が何層ものレイヤーになっていることがあります。

### 主従関係

主従関係は、親オブジェクトがその子に対して一定の支配力を有する親子のリンクです。主従関係のリンクの場合:

- 子レコードに親項目が必要です。
- 親のアクセスレベルによって、その子のアクセスレベルが決まります。
- 親レコードが削除されると、その子も削除されます。

Salesforce は、オブジェクトごとに最大2つの主従項目と最大3つのレベルの主従関係をサポートします。

Salesforce ヘルプの「[オブジェクトリレーションの概要](#)」を参照してください。

 **メモ:** Mobile Sync は多対多のリレーションをサポートしていません。

## 関連レコードの同期で使用されるオブジェクト

関連レコードの同期では、2つの特別な種別の同期ターゲットを使用します。

### 親子の上位同期ターゲット

- **iOS:** `SFParentChildrenSyncUpTarget`
- **Android:** `ParentChildrenSyncUpTarget`

ローカルで作成または更新されたレコードおよび削除されたレコードを処理します。

### 親子の下位同期ターゲット

- **iOS:** `SFParentChildrenSyncDownTarget`
- **Android:** `ParentChildrenSyncDownTarget`

`resync` および `cleanResyncGhosts` メソッドをサポートします。

これらの同期ターゲットは、*leave-if-changed* および *overwrite* マージモードをサポートします。各同期ターゲットは、指定した必須情報を使用してインスタンスを初期化するファクトリーメソッド (iOS) またはコンストラクター (Android) を提供します。同期を実行するには、新しい同期ターゲットクラスインスタンスを設定し、同期ターゲットオブジェクトを受け入れる標準の Mobile Sync 同期メソッドに渡します。



同期ターゲットを初期化するために、必要な関連レコード情報を配信する、次の2つのヘルパーオブジェクトも提供します。

#### 親情報オブジェクト

- **iOS:**  
**Swift**  
`ParentInfo`  
**Objective-C**  
`SFParentInfo`
- **Android:** `ParentInfo`

次の値を含む:

- オブジェクト種別
- スープ名
- ID項目名(いずれの場合も省略可能。デフォルト設定は「Id」。この値は、レコードを識別する別の項目を指定する場合にのみ設定します。)
- 最終更新日項目名(いずれの場合も省略可能。デフォルト設定は「LastModifiedDate」。この値は、タイムスタンプに別の項目を指定する場合にのみ設定します。)
- 外部ID項目名(いずれの場合も省略可能。指定した場合、なおかつローカルに作成された親または子レコードでその値が指定されている場合、Mobile Sync は、`create` ではなく `upsert` を実行します)。

#### 子情報オブジェクト

- **iOS:** `ChildrenInfo`
- **Android:** `ChildrenInfo`

次の値を含む:

- オブジェクト種別
- スープ名
- 親ID項目名
- ID項目名(いずれの場合も省略可能。デフォルト設定は「Id」。この値は、レコードを識別する別の項目を指定する場合にのみ設定します。)
- 最終更新日項目名(いずれの場合も省略可能。デフォルト設定は「LastModifiedDate」。この値は、タイムスタンプに別の項目を指定する場合にのみ設定します。)
- 外部ID項目名(いずれの場合も省略可能。指定した場合、なおかつローカルに作成された親または子レコードでその値が指定されている場合、Mobile Sync は、`create` ではなく `upsert` を実行します)。

### SmartStore データモデルの準備

関連オブジェクトをオフラインで処理するための準備としてまず、予想される親および子のオブジェクト種別ごとに SmartStore スープを設定します。スープごとに、サーバー側のリレーションをモデル化するインデックス付き ID 項目を追加します。以下は、必須のインデックス付き項目のリストです。

親オブジェクトのSoup:

レコードのサーバー ID の項目

子オブジェクトのスープ:

レコードのサーバー ID の項目

親レコードのサーバー ID の項目

上位および下位の同期操作はすべて、親スープから開始して子スープへと進みます。このフローは次のように機能します。

- 関連レコードを下位同期すると、親レコードを同期ターゲットにし、この同期によってその親レコードとすべての子がダウンロードされます。たとえば、取引先を取引先責任者と同期すると、ダウンロードした取引先にリンクされた取引先責任者を取得します。対象の取引先にリンクされていない取引先責任者は取得しません。
- 関連レコードを上位同期すると、MobileSync が親レコードのスープを反復して、関連する子レコードを選択します。子レコードが変更され親レコードと関連していない場合、その子レコードは上位同期操作時に無視されます。

## 上位同期

親子の上位同期ターゲットを初期化するには、親情報オブジェクトと子情報オブジェクトを指定します。これらのオブジェクトは、サーバー上とローカルストア内の両方の親子レコード間の関係を完全に説明します。また、同期する項目のリストも指定します。以下は、必須情報の完全リストです。

- 親情報オブジェクト
- 子情報オブジェクト
- リレーション種別 (主従、参照など)
- 親および子スープの上位同期する項目

上位同期操作では、親レコードを含むスープを反復し、指定した情報を使用して子のスープから関連レコードを取り込みます。\_\_local\_\_ 項目が true に設定されている場合は、レコードがダーティとみなされます。1つの親とその子からなるレコードツリー内のいずれかのレコードがダーティの場合に、上位同期の候補になります。上位同期が実際に行われるかどうかは、Mobile Sync がマージモードをどのように処理するかによって異なります。

### 「Leave-if-changed」マージモードの処理

Mobile Sync は、同期ターゲットの親および子サーバーレコードの最終更新日項目を取得します。次に該当する場合は、上位同期操作でこのレコードがスキップされます。

- 取得したサーバーレコードの最終更新日が、対応するローカルレコードよりも新しい場合  
または
- 取得したサーバーレコードが削除されている場合。

### 上位同期後に適用される更新

ローカルの変更がサーバーと同期されたら、Mobile Sync が関連レコードをクリーンアップします。

- 上位同期によってサーバーに親または子レコードが作成されると、MobileSync が対応するローカルレコードのサーバー ID 項目を更新します。作成されたレコードが親の場合は、ローカルストアにあるその子の親 ID 項目も更新されます。

- 前回の下位同期後にいずれかのサーバーレコードが削除されている場合には、MobileSync が対応するローカルレコードを削除します。
- 上位同期によって親レコードが削除され、リレーション種別が主従の場合は、Mobile Sync がサーバー上とローカルストア内のレコードの子を削除します。

## 下位同期

親子の下位同期ターゲットを初期化するために、親情報オブジェクトと子情報オブジェクトを指定します。これらのオブジェクトは、サーバー上とローカルストア内の両方の親子レコード間の関係を完全に説明します。また、同期する項目のリストと一定の SOQL 検索条件も指定します。

新しい下位同期ターゲットは実際、SOQL 下位同期ターゲットのサブクラスです。ただし、この同期ターゲットは、SOQL クエリを指定されるのではなく、親および子情報オブジェクト、項目のリスト、SOQL 検索条件から SOQL クエリを生成します。

下位同期ターゲットに渡される情報

- 親情報オブジェクト
- 子情報オブジェクト
- リレーション種別 (主従、参照など)
- 親および子スープの下位同期する項目
- 下位同期時にクエリでルートレコードに適用される SOQL 検索条件 (WHERE 句の条件など)

サーバーコール

Mobile Sync は SOQL を使用して、それぞれ 1 つの親とその子からなるレコードツリーを取得します。そして、親と子を分離して、それぞれのスープに保存します。

「Leave-if-changed」マージモードの処理

ダーティレコード (ローカルで作成、変更、削除されたレコード) を含むローカルのレコードツリーが変更されることはありません。たとえば、親レコードに 1 つのダーティな子がある場合、Mobile Sync は親も子も更新しません。親がサーバー上で変更され、ローカルではクリーンの場合も、このルールは適用されます。

再同期の処理

再同期時に、Mobile Sync は SOQL クエリを調整し、前回の同期後に親が変更されたレコードツリーのみをダウンロードします。

## 関連レコードの同期の実装

関係する原則や要件を理解すれば、実装は簡単です。関連レコードの同期はいくつかのステップでコードに追加できます。次のコードスニペットは、取引先 (親) および取引先責任者 (子) オブジェクトを使用した手法を示しています。

 **メモ:** 次のサーバー側の制限は、関連オブジェクトおよび関連しないオブジェクトからのレコードの同期方法に影響を与えます。

- 複合 API を使用して、スープを更新できます。ただし、この API には、1 度に 25 レコードという制限があります。たとえば、iOS の `SFParentChildrenSyncUpTarget` または Android の `ParentChildrenSyncUpTarget` の実装を参照してください。

- SOQL の制限は、Mobile Sync が単一クエリで子レコードを選択する方法に影響を与えます。このため、reSync は、変更された親の変更された子は同期できますが、変更されない親の変更された子は同期できません。この制限に対処するには、子オブジェクトを直接照会する別個のカスタムの同期ターゲットを使用します。
- SOQL には、UNION 演算子はありません。そのため、サーバーを1回コールして関連しないエンティティを取得することはできません。代わりに、別個のクエリを使用してください。たとえば、クエリごとに異なる SOQLSyncDownTarget オブジェクトを使用します。
- 関連しないレコードを上位同期するには、カスタム上位同期ターゲットを実装することを検討してください。

## iOS および Android

1. 必須のインデックス付き項目を使用して SmartStore スープを定義する userstore.json ファイルを作成します。

```
{
 "soups": [
 {
 "soupName": "ContactSoup",
 "indexes": [
 { "path": "Id", "type": "string"},
 { "path": "LastName", "type": "string"},
 { "path": "AccountId", "type": "string"},
 { "path": "__local__", "type": "string"},
 { "path": "__locally_created__", "type": "string"},
 { "path": "__locally_updated__", "type": "string"},
 { "path": "__locally_deleted__", "type": "string"}
]
 },
 {
 "soupName": "AccountSoup",
 "indexes": [
 { "path": "Id", "type": "string"},
 { "path": "Name", "type": "string"},
 { "path": "Description", "type": "string"},
 { "path": "__local__", "type": "string"},
 { "path": "__locally_created__", "type": "string"},
 { "path": "__locally_updated__", "type": "string"},
 { "path": "__locally_deleted__", "type": "string"}
]
 }
]
}
```

2. 2つの名前指定同期を定義する usersyncs.json ファイルを作成します。この例の場合、「DownSync」および「UpSync」設定で、取引先オブジェクトと取引先責任者オブジェクト間の親子関係をモデル化します。

```
{
 "syncs": [{
 "syncName": "DownSync",
 "syncType": "syncDown",
 "soupName": "AccountSoup",
```

```

"target": {
 "iOSImpl": "SFParentChildrenSyncDownTarget",
 "AndroidImpl": "ParentChildrenSyncDownTarget",
 "parent": {
 "idFieldName": "Id",
 "subjectType": "Account",
 "modificationDateFieldName": "LastModifiedDate",
 "soupName": "AccountSoup"
 },
 "parentFieldlist": [
 "Id",
 "Name",
 "Description"
],
 "children": {
 "parentIdFieldName": "AccountId",
 "idFieldName": "Id",
 "subjectType": "Contact",
 "modificationDateFieldName": "LastModifiedDate",
 "soupName": "ContactSoup",
 "subjectTypePlural": "Contacts"
 },
 "childrenFieldlist": [
 "LastName",
 "AccountId"
],
 "relationshipType": "MASTER_DETAIL",
 "parentSqlFilter": "Name LIKE 'A%' ",
 "type": "parent_children",
 "idFieldName": "Id"
},
"options": {"mergeMode": "OVERWRITE"}
},

{
 "syncName": "UpSync",
 "syncType": "syncUp",
 "soupName": "AccountSoup",
 "target": {
 "iOSImpl": "SFParentChildrenSyncUpTarget",
 "AndroidImpl": "ParentChildrenSyncUpTarget",
 "childrenCreateFieldlist": [
 "LastName",
 "AccountId"
],
 "parentCreateFieldlist": [
 "Id",
 "Name",
 "Description"
],
 "childrenUpdateFieldlist": [
 "LastName",
 "AccountId"
],
 },

```

```

 "parentUpdateFieldlist": [
 "Name",
 "Description"
],
 "parent": {
 "idFieldName": "Id",
 "subjectType": "Account",
 "modificationDateFieldName": "LastModifiedDate",
 "soupName": "AccountSoup"
 },
 "relationshipType": "MASTER_DETAIL",
 "type": "rest",
 "modificationDateFieldName": "LastModifiedDate",
 "children": {
 "parentIdFieldName": "AccountId",
 "idFieldName": "Id",
 "subjectType": "Contact",
 "modificationDateFieldName": "LastModifiedDate",
 "soupName": "ContactSoup",
 "subjectTypePlural": "Contacts"
 },
 "parentUpdateFieldlist": [
 "Name",
 "Description"
],
 "idFieldName": "Id"
 },
 "options": {"mergeMode": "LEAVE_IF_CHANGED"}
}
]
}

```

次のステップは、これらの設定を使用して関連する取引先(親)レコードと取引先責任者(子)レコードを同期する方法を示しています。

#### iOS

1. Xcode プロジェクトの **[Build Phases (フェーズを作成)] > [Copy Bundle Resources (バンドルリソースをコピー)]** で、userstore.json および usersyncs.json ファイルを Xcode 同期ターゲットを追加します。
2. ストアと同期設定を読み込みます。Mobile SDK が初期化されてから SmartStore または Mobile Sync メソッドをコールするまでにこれらのローダーをコールします。たとえば、AppDelegate クラスで、loginIfRequired に渡すブロック内でこれらのメソッドをコールします。これらのいずれのメソッドも複数回コールしないでください。

#### Swift

```

func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
 self.window = UIWindow(frame: UIScreen.main.bounds)
 self.initializeAppViewState()
 AuthHelper.loginIfRequired {
 MobileSyncSDKManager.shared.setupUserStoreFromDefaultConfig()
 MobileSyncSDKManager.shared.setupUserSyncsFromDefaultConfig()
 self.setupRootViewController()
 }
}

```

```

 }
 ...
 return true
}

```

### Objective-C

```

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
 self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen].bounds];
 [self initializeAppViewState];
 [SFSDKAuthHelper loginIfRequired:^(
 [self setupRootViewController];
 [[MobileSyncSDKManager sharedManager] setupUserStoreFromDefaultConfig];
 [[MobileSyncSDKManager sharedManager] setupUserSyncsFromDefaultConfig];
)];
 ...
 return YES;
}

```

3. 同期名を取得する Mobile Sync `reSync` メソッドをコールします。設定ファイルからの同期名を渡すことにより、すべての上位同期および下位同期操作で `reSync(named:update:)` メソッドを使用できます。

### Swift

```

self.syncMgr.reSync(named: kSyncName) { [weak self] (syncState) in
 // Handle updates
}

```

### Objective-C

```

[self.syncMgr reSyncByName:kSyncUpName updateBlock:^(SFSyncState* sync) {
 // Handle updates
}

```

### Android

1. `userstore.json` および `usersyncs.json` ファイルをプロジェクトの `res/raw` フォルダーに配置します。
2. 同期名を取得する Mobile Sync 同期メソッドをコールします。設定ファイルからの同期名を渡すことにより、すべての上位同期および下位同期操作で `reSync(named:update:)` メソッドを使用できます。

上位同期および下位同期の場合:

```

//Resync a predefined sync by its name

syncManager.reSync(syncName,
 new SyncUpdateCallback() {
 @Override
 public void onUpdate(SyncState sync) {
 // Handle updates
 }
 });

```



## ハイブリッドおよび React Native アプリケーションでの Mobile Sync の使用

JavaScript 用の Mobile Sync は、Salesforce オブジェクトを JavaScript オブジェクトとして表す Mobile SDK ライブラリです。JavaScript で Mobile Sync を使用すると、Salesforce オブジェクトのモデルを作成でき、そのモデルのデータを変更するだけで基盤となるレコードを操作できます。SOQL または SOSL クエリを実行すると、JSON 文字列としてではなくモデルコレクションで結果レコードが取得されます。

ハイブリッドアプリケーションの Mobile SDK では、Mobile Sync を使用するための2つのオプションがあります。

- `com.salesforce.plugin.mobilesync`: Mobile Sync プラグインは基本的な「上位同期」と「下位同期」機能を提供します。このプラグインは、ネイティブ Mobile Sync ライブラリの一部を公開します。単純な同期タスクの場合、プラグインを使用して、Web ビューではなくネイティブスレッドでレコードを迅速に同期できます。
- `mobilesync.js`: Mobile Sync JavaScript ライブラリは、より複雑な同期操作のための `Force.SObject` データフレームワークを提供します。このライブラリは、拡張性の高いデータモデリングメカニズムを定義するオープンソースの JavaScript フレームワークである `backbone.js` をベースにしています。この技術を理解するには、[backbonejs.org](http://backbonejs.org) にある例とドキュメントを参照してください。

Mobile Sync の使用方法を示す一連のサンプルハイブリッドアプリケーションが次の場所に用意されています。

`hybrid/SampleApps/AccountEditor/assets/www` フォルダのサンプルアプリケーションは、`mobilesync.js` での `Force.SObject` ライブラリの使用方法を示します。

- Account Editor (`AccountEditor.html`)
- User Search (`UserSearch.html`)
- User and Group Search (`UserAndGroupSearch.html`)

`hybrid/SampleApps/SimpleSync` フォルダのサンプルアプリケーションは、Mobile Sync プラグインの使用方法を示します。

## どのハイブリッドバージョンの Mobile Sync を使用すべきですか？

Mobile Sync の JavaScript バージョンである `mobilesync.js` ファイルと、Cordova プラグインを通じてハイブリッドアプリケーションで使用可能なネイティブ Mobile Sync は、名前を共有していますが、得られる利点は異なります。

`mobilesync.js` は、`backbone.js` 上に構築され、1つまたは一連のレコードを表す使いやすいモデルオブジェクトを提供します。また、便利な取得、保存、および削除メソッドも提供します。ただし、完全な上位同期または下位同期機能はありません。`SObjectCollection` でのレコードの取得はプラグインの `syncDown` メソッドに似ていますが、取得されたすべてのオブジェクトはメモリ内に格納されます。そのため、大量のデータセットを移動する場合には適していません。さらに、上位同期機能を自分で実装する必要があります。AccountEditor サンプルアプリケーションは、一般的な JavaScript `syncUp()` の実装を示します。

ネイティブ Mobile Sync は、モデルオブジェクトを返しません、サーバーとの間で大量のデータセットを移動するための堅牢な `syncUp` および `syncDown` メソッドを提供します。

また、2つのライブラリを一緒に使用することもできます。たとえば、`mobilesync.js` で `Force.StoreCache` を設定して、Mobile Sync プラグインを使用してデータを同期し、`mobilesync.js` を使用して取得または保存

をコールできます。その後、Mobile Sync プラグインを使用して同じキャッシュから上位同期できます。これにより、すべてが機能するようになります。

両方のライブラリで独自のカスタムエンドポイントを定義できます。どちらを選択しますか? この決定をする場合、次のガイドラインが役立ちます。

- JavaScript でデータを保存または取得するためにサーバーと直接通信する場合、`mobilesync.js` のカスタムエンドポイントを使用します。
- SmartStore とのみ通信し、Mobile Sync プラグインを使用して SmartStore にデータを取り込む場合、`mobilesync.js` のカスタムエンドポイントは必要ありません。ただし、ネイティブカスタムターゲットを定義する必要があります。

 **メモ:** `mobilesync.js` は内部で promise を使用します。

Mobile SDK の promise ベースの API には、以下が含まれます。

- `force+promise.js`
- `smartstoreclient` Cordova プラグイン (`com.salesforce.plugin.smartstore.client`)
- `mobilesync.js`

## Backbone テクノロジーについて

Mobile Sync ライブラリ `mobilesync.js` は、オープンソースの Backbone JavaScript ライブラリの拡張機能を提供します。Backbone ライブラリは、Web アプリケーションを構築する主要なビルディングブロックを定義します。

- 情報をモデル化する Models (キーと値のバインドイベントおよびカスタムイベントを含む)
- データセットを格納する Collections (豊富な enumerable 関数の API を含む)
- モデルの情報を表示する Views (宣言型イベント処理を含む)
- ビュー間の移動を制御するルーター

Salesforce Mobile Sync は、Model および Collection コア Backbone オブジェクトを拡張し、Salesforce REST API に接続します。Mobile Sync は、SmartStore (Mobile SDK の安全なストレージコンポーネント) を介してオフラインサポート (省略可能) も提供します。

Backbone についての詳細は、<http://backbonejs.org/> および <http://backbonetutorials.com/> を参照してください。また、オンラインで「backbonejavascript」を検索すれば、多数のチュートリアルやビデオを見つけることもできます。

## モデルとモデルコレクション

Mobile Sync データフレームワークは、次の 2 つの種別のオブジェクトで構成されています。

- モデル
- モデルコレクション

これらのオブジェクトの定義は、`backbone.js` (一般的なサードパーティの JavaScript フレームワーク) で定義されているクラスを拡張します。背景情報は、「[モデルとモデルコレクション](#)」を参照してください。

## モデル

クライアントのモデルはサーバーレコードを表します。MobileSyncでは、モデルオブジェクトは `Force.SObject` (`Backbone.Model` クラスのサブクラス) のインスタンスです。 `SObject` は、Salesforce API および SmartStore (必要に応じて) と連動するように `Model` を拡張します。

`SObject` モデルオブジェクトで次の CRUD 操作を実行できます。

- Create
- Destroy
- Fetch
- Save
- Get/set attributes

また、モデルオブジェクトは監視可能です。ビューとコントローラーはオブジェクトの変更時に通知を受信できます。

## プロパティ

`Force.SObject` は、次のプロパティを `Backbone.Model` に追加します。

### **subjectType**

必須。このモデルが表す Salesforce オブジェクトの名前。この値は、標準オブジェクトまたはカスタムオブジェクトのいずれかを参照できます。

### **fieldlist**

必須。取得、保存、破棄する項目の名前。

### **cacheMode**

オフラインの動作。

### **mergeMode**

競合処理の動作。

### **cache**

更新可能なレコードのオフラインストレージの場合に使用します。MobileSyncには、`Force.StoreCache` (SmartStore によってサポートされるキャッシュ実装) が組み込まれています。

### **cacheForOriginals**

競合検出に対応できるように、サーバーから取得したレコードの元のコピーを格納します。

## 例

モデルプロパティの値は、次のような複数の方法で割り当てることができます。

- `Force.SObject` インスタンスのプロパティとして割り当てる。
- `Force.SObject` サブクラスのメソッドとして割り当てる。これらのメソッドでは、目的の CRUD アクション (「create」、「read」、「update」、「delete」) を指定するパラメーターを使用します。
- `fetch()`、`save()`、`destroy()` 関数コールのパラメーターで割り当てる。

たとえば、次のコードスニペットはどれも同じ内容になります。

```
// As properties on a Force.SObject instance
acc = new Force.SObject({Id:"<some_id>"});
```

```
acc.subjectType = "account";
acc.fieldlist = ["Id", "Name"];
acc.fetch();
```

```
// As methods on a Force.SObject sub-class
Account = Force.SObject.extend({
 subjectType: "account",
 fieldlist: function(method) { return ["Id", "Name"]; }
});
Acc = new Account({Id:"<some_id>"});
acc.fetch();
```

```
// In the options parameter of fetch()
acc = new Force.SObject({Id:"<some_id>"});
acc.subjectType = "account";
acc.fetch({fieldlist:["Id", "Name"]});
```

## モデルコレクション

Mobile Sync のモデルコレクションは、クエリ結果のコンテナです。モデルコレクションに保存されるクエリ結果は、SOQL、SOSL、MRU クエリを介してサーバーから取得できます。また、必要に応じて、SmartSQL (キャッシュが SmartStore の場合) や別のクエリメカニズム (代替キャッシュを使用している場合) を介してキャッシュから取得することもできます。

モデルコレクションオブジェクトは、`Force.SObjectCollection` (`Backbone.Collection` クラスのサブクラス) のインスタンスです。`SObjectCollection` は、Salesforce API および SmartStore (必要に応じて) と連動するように `Collection` を拡張します。

## プロパティ

`Force.SObjectCollection` は、次のプロパティを `Backbone.Collection` に追加します。

### **config**

必須。コレクションで保持できるレコードを定義します (SOQL、SOSL、MRU、SmartSQL を使用)。

### **cache**

更新可能なレコードのオフラインストレージの場合に使用します。Mobile Sync には、`Force.StoreCache` (SmartStore によってサポートされるキャッシュ実装) が組み込まれています。

### **cacheForOriginals**

競合検出に対応できるように、サーバーから取得したレコードの元のコピーを格納します。

## 例

モデルコレクションプロパティの値は、次のような複数の方法で割り当てることができます。

- `Force.SObject` インスタンスのプロパティとして割り当てる
- `Force.SObject` サブクラスのメソッドとして割り当てる
- `fetch()`、`save()`、`destroy()` 関数コールのオプションパラメーターで割り当てる

たとえば、次のコードスニペットはどれも同じ内容になります。

```
// As properties on a Force.SObject instance
list = new Force.SObjectCollection({config:<valid_config>});
list.fetch();
```

```
// As methods on a Force.SObject sub-class
MyCollection = Force.SObjectCollection.extend({
 config: function() { return <valid_config>; }
});
list = new MyCollection();
list.fetch();
```

```
// In the options parameter of fetch()
list = new Force.SObjectCollection();
list.fetch({config:valid_config});
```

## Mobile Sync プラグインの使用

Mobile SDK 3.0 以降の Mobile Sync プラグインは、Mobile Sync ネイティブライブラリの「上位同期」および「下位同期」機能への JavaScript アクセスを提供します。そのため、パフォーマンス集約的な操作(ネットワークネゴシエーション、解析、SmartStore 管理など)はネイティブスレッド上で実行され、Web 表示操作に影響を与えません。

Mobile Sync プラグインは、Mobile SDK の npm スクリプトによってハイブリッドプロジェクトに追加されます。

- forceios バージョン 3.0 以降の場合、プラグインは自動的に含まれます。
- forcedroid バージョン 3.0 以降の場合、SmartStore を使用するときは「yes」と回答します。

Mobile Sync プラグインを既存のハイブリッドアプリケーションに追加する場合、forcedroid または forceios の最新バージョンを使用してアプリケーションを再作成することをお勧めします。新しいアプリケーションの準備ができたなら、カスタム HTML、CSS、および JavaScript ファイルを古いプロジェクトから新しいプロジェクトにコピーします。

## Mobile Sync プラグインのメソッド

Mobile Sync プラグインは、`syncDown()` と `syncUp()` の2つのメソッドを公開します。これらのメソッドを使用する場合、いくつかの重要なガイドラインに従うと作業が簡素化されます。

- プラグインを使用して同期するレコードをローカルで作成、更新、または削除するには、`mobilesync.js` の `Force.SObject` を使用します。Mobile Sync は、`mobilesync.js` によって作成されるスープレコードにいくつかの特殊な項目があるものと想定します。
- 同様に、同期操作で使用するスープを作成するには、`mobilesync.js` の `Force.StoreCache` を使用します。
- スープのオブジェクトを変更している場合は、`syncDown()` をコールする前に必ず `syncUp()` をコールします。

## syncDown() メソッド

`target` で指定した `sObject` を、`soupName` で指定した SmartStore スープにダウンロードします。スープの `sObject` の ID が同期ターゲットで指定したオブジェクトの ID と同じ場合は、Mobile Sync がスープの重複オブジェクトを上書きします。

また、Mobile Sync は更新下位同期ターゲットをサポートしているため、キャッシュされたレコードの更新プロセスが簡単になります。「[更新下位同期ターゲットの使用](#)」を参照してください。

### 構文

```
cordova.require("com.salesforce.plugin.mobilesync").syncDown(
 [isGlobalStore,]target, soupName, options, callback);
cordova.require("com.salesforce.plugin.mobilesync").syncDown(
 [storeConfig,]target, soupName, options, callback);
```

### パラメーター

#### isGlobalStore

(省略可能) この操作がグローバルまたはユーザーベースのどちらの SmartStore データベースで行われるかを示す Boolean。デフォルトは `false` です。

#### storeConfig

(省略可能) ストア名とストアがグローバルまたはユーザーベースのどちらかであることを指定する StoreConfig オブジェクト。

#### target

スープにダウンロードする `sObject` を示します。次のいずれかの文字列にすることができます。

- `{type:"soql", query:"<soql query>"}`

指定した SOQL クエリで返された `sObject` をダウンロードします。

- `{type:"sosl", query:"<sosl query>"}`

指定した SOSL クエリで返された `sObject` をダウンロードします。

- `{type:"mru", objectType:"<object type>", fieldlist:"<fields to fetch>"}`

指定した `sObject` 型の最近使用した `sObject` の指定した項目をダウンロードします。

- `{type:"custom", androidImpl:"<name of native Android target class (if supported)>", iosImpl:"<name of native iOS target class (if supported)>"}`

カスタムの同期ターゲットで指定されたレコードをダウンロードします。カスタムの同期ターゲットを使用する場合、`androidImpl` または `iosImpl`、あるいはその両方 (推奨) を指定します。「[カスタム下位同期ターゲットの使用](#)」を参照してください。

#### soupName

ダウンロードされた `sObject` を受信するスープの名前。

#### options

次のいずれかの値を使用します。

- 変更されたローカルレコードを上書きする場合は、`{mergeMode:Force.MERGE_MODE_DOWNLOAD.OVERWRITE}` を渡します。



- 変更されたローカルレコードを維持する場合は、  
{mergeMode:Force.MERGE\_MODE\_DOWNLOAD.LEAVE\_IF\_CHANGED} を渡します。この値を使用すると、ローカルで変更されたレコードが上書きされません。

**callback**

同期の開始後にコールされる関数。この関数は、同期操作中に複数回コールされます。

1. 同期操作の開始時
2. 内部 REST 要求の完了時
3. 結果の各ページをダウンロード後 (結果を 100% 受信するまで)

同期操作の状況更新は、ブラウザーイベントによって通知されます。これらの更新をリスンするには、次のコードを使用します。

```
document.addEventListener("sync",
 function(event) {
 // event.detail contains the status of the sync operation
 }
);
```

event.detail メンバーには、次の項目との対応付けが含まれます。

- syncId: この同期操作の ID
- type: 「syncDown」
- target: 指定した同期ターゲット
- soupName: 指定したスープ名
- options: 「{}」
- status: 同期状況 (「新規」、「実行中」、「完了」、「失敗」のいずれか)
- progress: 現時点でダウンロード済みの合計レコードの割合 (整数、0 ~ 100%)
- totalSize: 現時点でダウンロード済みのレコード数

**syncUp() メソッド**

soupName で指定した SmartStore スープで作成、削除、または更新されたレコードをアップロードして、Salesforce サーバーの対応するレコードを更新、作成、または削除します。更新はブラウザーイベントによって通知されます。

**構文**

```
cordova.require("com.salesforce.plugin.mobilesync").syncUp(isGlobalStore, target, soupName,
 options, callback);
cordova.require("com.salesforce.plugin.mobilesync").syncUp(storeConfig, target, soupName,
 options, callback);
```

**パラメーター****isGlobalStore**

(省略可能) この操作がグローバルまたはユーザーベースのどちらの SmartStore データベースで行われるかを示す Boolean。デフォルトは false です。



**storeConfig**

(省略可能)ストア名とストアがグローバルまたはユーザーベースのどちらかであることを指定する StoreConfig オブジェクト。

**target**

少なくとも1つのカスタムの同期ターゲットクラスの名前を含む JSON オブジェクト (カスタムの同期ターゲットを定義している場合)。

Salesforce オブジェクトは、アプリケーションでは更新できない特定の項目が必要な場合があります。このようなオブジェクトの場合、単一の項目リストを作成と更新の両方の操作に使用する同期ターゲットでロック項目を更新しようとする、失敗する可能性があります。過去のバージョンの Mobile Sync では、開発者が独自の固有同期ターゲットを作成して作成項目リストと更新項目リストを区別する必要がありました。

Mobile SDK 5.1 以降、このような場合に独自の固有同期ターゲットを定義する必要はなくなりました。代わりに、作成操作と更新操作に個別の項目リストを指定するには、次の JSON オブジェクトを target オブジェクトに追加します。

```
{createFieldlist: [<array_of_fields_to_create>], updateFieldlist:
[<another_array_of_fields_to_update>]}
```

createFieldlist および updateFieldlist 引数を指定すると、独自の固有同期ターゲットは該当する場合にそれらを使用します。その場合、同期ターゲットは「同期オプション」設定で定義された項目リストを無視します。

target メタデータについての詳細は、syncDown() メソッドの説明を参照してください。

**soupName**

アップロードする sObject があるスープの名前。

**options**

次のキーとの対応付け:

- fieldlist: サーバーに送信される項目のリスト。
- mergeMode:
  - 変更されたリモートレコードを上書きする場合は、「OVERWRITE」を渡します。
  - 変更されたリモートレコードを維持する場合は、「LEAVE\_IF\_CHANGED」を渡します。この値を使用すると、サーバーで変更されたレコードが上書きされません。
  - 指定されていない場合、デフォルトの「OVERWRITE」に設定されます。

**callback**

同期の開始後に複数回コールされる関数。同期操作中、この関数は次のようなイベントでコールされます。

1. 同期操作の開始時
2. 内部 REST 要求の完了時
3. 結果の各ページをアップロード後 (結果を 100% 受信するまで)

同期操作の状況更新は、ブラウザーイベントによって通知されます。これらの更新をリスンするには、次のコードを使用します。

```
document.addEventListener("sync",
 function(event) {
 // event.detail contains the status of the sync operation
```

```
 }
);
```

event.detail メンバーには、次の項目との対応付けが含まれます。

- syncId: この同期操作の ID
- type: 「syncUp」
- target: 「{}」、または実装した iOS および Android カスタムの同期ターゲットクラスのクラス名を含む対応付けやディクショナリ
- soupName: 指定したスープ名
- options:
  - fieldlist: サーバーに送信される項目のリスト
  - mergeMode: 「OVERWRITE」または「LEAVE\_IF\_CHANGED」
- status: 同期状況 (「新規」、「実行中」、「完了」、「失敗」のいずれか)
- progress: 現時点でダウンロード済みの合計レコードの割合 (整数、0 ～ 100%)
- totalSize: 現時点でダウンロード済みのレコード数

関連トピック:

[ユーザーベースストアの作成およびアクセス](#)

## JavaScript での Mobile Sync の使用

ハイブリッドアプリケーションで MobileSync を使用するには、<script> タグを使用して次のファイルをインポートします。

- jquery-x.x.x.min.js ([SalesforceMobileSDK-Shared](#) リポジトリの dependencies/jquery/ ディレクトリ内のバージョンを使用)
- underscore-x.x.x.min.js ([SalesforceMobileSDK-Shared](#) リポジトリの dependencies/underscore/ ディレクトリ内のバージョンを使用)
- backbone-x.x.x.min.js ([SalesforceMobileSDK-Shared](#) リポジトリの dependencies/backbone/ ディレクトリ内のバージョンを使用)
- cordova.js
- force.js
- mobilesync.js

## モデルオブジェクトの実装

Mobile Sync オブジェクトの使用を開始するには、操作する各 SObject を表すモデルオブジェクトを定義します。SObjects は、標準 Salesforce オブジェクトまたはカスタムオブジェクトになります。たとえば、次のコードは、2つの必須プロパティ (subjectType および fieldlist) を設定する Account オブジェクトのモデルを作成し、cacheMode() 関数を定義します。

```
app.models.Account = Force.SObject.extend({
 subjectType: "Account",
```

```

 fieldlist: ["Id", "Name", "Industry", "Phone"],

 cacheMode: function(method) {
 if (app.offlineTracker.get("offlineStatus") == "offline") {
 return "cache-only";
 }
 else {
 return (method == "read" ?
 "cache-first" : "server-first");
 }
 }
 }
});

```

app.models.Account モデルオブジェクトは、mobilesync.js で定義されている Force.SObject を拡張します。また、cacheMode() 関数は、ローカル offlineTracker オブジェクトを照会して、デバイスのオフライン状況を取得します。Cordova ライブラリを使用して、特定時点のオフライン状況を判断できます。

Mobile Sync は、モデルに対して取得操作または保存操作を実行できます。アプリケーションの cacheMode の値を使用して、操作の実行対象(サーバーまたはキャッシュ)を決定します。cacheMode メンバーは、単純な string 型のプロパティか、文字列を返す関数のいずれかになります。

## モデルコレクションの実装

このサンプルアプリケーションのモデルコレクションは、Force.SObjectCollection を拡張します。

```

// The AccountCollection Model
app.models.AccountCollection = Force.SObjectCollection.extend({
 model: app.models.Account,
 fieldlist: ["Id", "Name", "Industry", "Phone"],
 setCriteria: function(key) {
 this.key = key;
 },
 config: function() {
 // Offline: do a cache query
 if (app.offlineTracker.get("offlineStatus") == "offline") {
 return {type:"cache", cacheQuery:{queryType:"like",
 indexPath:"Name", likeKey: this.key+"%",
 order:"ascending"}};
 }
 // Online
 else {
 // First time: do a MRU query
 if (this.key == null) {
 return {type:"mru", objectType:"Account",
 fieldlist: this.fieldlist};
 }
 // Other times: do a SOQL query
 else {
 var soql = "SELECT " + this.fieldlist.join(",")
 + " FROM Account"
 + " WHERE Name like '" + this.key + "%'";
 return {type:"soql", query:soql};
 }
 }
 }
});

```

```
 }
 });
};
```

このモデルコレクションは、コレクションから取得する取引先の名前であるキー(省略可能)を使用します。また、取得される情報を決定する `config()` 関数も定義します。デバイスがオフラインの場合、`config()` 関数はキャッシュクエリステートメントを作成します。デバイスがオンラインで、キーが指定されていない場合、最後に使用したレコード("mru")を照会します。デバイスがオンラインで、キーが指定されている場合、キーに一致する名前のレコードを取得する標準 SOQL クエリを作成します。Force.SObjectCollection プロトタイプの取得操作は、返された設定を透過的に使用して、モデルコレクションにクエリレコードを自動的に入力します。

キャッシュクエリの書式設定についての詳細は、「[オフラインキャッシュのための StoreCache の使用](#)」を参照してください。

 **メモ:** これらのコード例は、Account Editor サンプルアプリケーションの一部です。サンプルの説明は、「[Account Editor のサンプル](#)」を参照してください。

## オフラインキャッシュ

オフラインサポートを提供するには、アプリケーションでそのモデルおよびコレクションをキャッシュできる必要があります。Mobile Sync は、キャッシュの操作を完全に制御できるメカニズム(設定可能)を提供します。

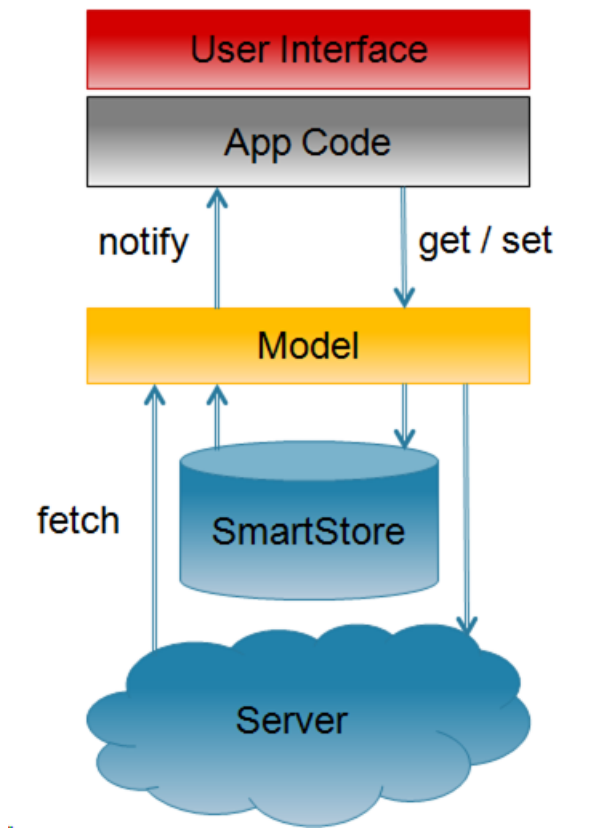
### デフォルトキャッシュおよびカスタムキャッシュの実装

デフォルトキャッシュの場合、Mobile Sync ライブラリは StoreCache (SmartStore を使用するキャッシュ実装)を定義します。StoreCache と SmartStore はどちらも Mobile Sync アプリケーションのコンポーネント(省略可能)です。アプリケーションが Mobile SDK コンテナではなくブラウザーで実行されている場合や、SmartStore を使用しない場合、代替キャッシュ実装を提供する必要があります。MobileSync では、次の操作をサポートするためにキャッシュオブジェクトが必要になります。

- retrieve
- save
- save all
- remove
- find

### Mobile Sync のキャッシュワークフロー

Mobile Sync モデルは、アプリケーションの代わりにキャッシュおよび Salesforce サーバーとのすべてのやり取りを実行します。アプリケーションは、モデルオブジェクトの属性を取得および設定します。モデルは、保存操作時にこれらの属性設定を使用して、変更をキャッシュまたはサーバーに書き込むかどうかを判断したり、新しいデータと既存のデータのマージ方法を決定したりします。基本データまたはモデル自体に何らかの変更があると、モデルはイベント通知を送信します。同様に、取得を要求すると、モデルはモデルコレクションでデータを取得してアプリケーションに提供します。



Mobile Sync は、CRUD 操作時にキャッシュのデータを透過的に更新します。フラグ(省略可能)を介して透過レベルを制御できます。キャッシュされたオブジェクトは、ローカルで作成、更新、削除されたかどうかを示す「dirty」属性を保持します。

キャッシュモデル

キャッシュを使用する場合、各CRUD操作のモデルを指定できます。サポートされるモードは次のとおりです。

| モード           | 定数                           | 説明                                         |
|---------------|------------------------------|--------------------------------------------|
| "cache-only"  | Force.CACHE_MODE.CACHE_ONLY  | キャッシュから読み取るか、キャッシュに書き込みます。サーバーの操作は実行しません。  |
| "server-only" | Force.CACHE_MODE.SERVER_ONLY | サーバーから読み取るか、サーバーに書き込みます。キャッシュの操作は実行しません。   |
| "cache-first" | Force.CACHE_MODE.CACHE_FIRST | 取得操作の場合のみ。キャッシュからレコードを取得します。キャッシュにレコードが含まれ |

| モード                    | 定数                            | 説明                            |
|------------------------|-------------------------------|-------------------------------|
|                        |                               | ていない場合、サーバーから取得してキャッシュを更新します。 |
| "server-first" (デフォルト) | Force.CACHE_MODE.SERVER_FIRST | サーバーの操作を実行し、キャッシュを更新します。      |

直接キャッシュを照会するには、キャッシュクエリを使用します。SmartStore は、クエリ API、独自のクエリ言語、Smart SQL を提供します。「[スープからのデータの取得](#)」を参照してください。

## オフラインキャッシュの実装

Mobile Sync でオフラインキャッシュをサポートするには、次のようないくつかのタスクを実装する必要があります。

- [app.models.Account](#) の例で示すようにオフライン状況を追跡して、CRUD 操作の適切なキャッシュコントロールフラグを指定する。
- ローカルで編集されたレコードを収集し、デバイスがオンラインに戻ったときにその変更をサーバーに保存する。次の例では、SmartStore キャッシュクエリを使用して、ローカルで変更されたレコードを取得し、SyncPage 関数をコールして、HTML でその結果を表示しています。

```
sync: function() {
 var that = this;
 var localAccounts = new app.models.AccountCollection();
 localAccounts.fetch({
 config: {type:"cache", cacheQuery: {queryType:"exact",
 indexPath:"__local__", matchKey:true}},
 success: function(data) {
 that.slidePage(new app.views.SyncPage({model: data}).render());
 }
 });
}

app.views.SyncPage = Backbone.View.extend({

 template: _.template($("#sync-page").html()),

 render: function(eventName) {
 $(this.el).html(this.template(_.extend(
 {countLocallyModified: this.model.length},
 this.model.toJSON())));
 this.listView = new app.views.AccountListView(
 {el: $("ul", this.el), model: this.model});
 this.listView.render();
 return this;
 },
});
```

```
...
});
```

## オフラインキャッシュのための StoreCache の使用

mobilesync.js ライブラリは、データを SmartStore に保存する StoreCache という名前のキャッシュを実装します。Mobile Sync は StoreCache をデフォルトキャッシュとして使用しますが、StoreCache はスタンドアロンコンポーネントです。Mobile Sync を使用しない場合でも、SmartStore の操作に StoreCache を活用できます。

 **メモ:** StoreCache は Mobile Sync と併用することを想定していますが、「[オフラインキャッシュ](#)」で説明されているように、要件を満たす任意のキャッシュメカニズムを Mobile Sync と共に使用することもできます。

## 作成および初期化

StoreCache オブジェクトは、内部的に SmartStore スープと連動します。スープ soupName によってサポートされる StoreCache オブジェクトを作成するには、次のコンストラクターを使用します。

```
new Force.StoreCache(soupName [, additionalIndexSpecs, keyField])
```

### soupName

必須。基盤となる SmartStore スープの名前。

### additionalIndexSpecs

デフォルトのインデックス項目に加えてキャッシュインデックスに含める項目。書式設定の手順は、「[スープの登録](#)」を参照してください。

### keyField

レコード ID を格納する項目の名前。指定しない場合、StoreCache は "Id" という名前の項目で ID を検索します。

StoreCache オブジェクトのスープ項目には、オフライン編集を追跡する次の 4 つの追加の Boolean 項目が含まれます。

- \_\_locally\_created\_\_
- \_\_locally\_updated\_\_
- \_\_locally\_deleted\_\_
- \_\_local\_\_ (前の 3 つのいずれかが true の場合、true に設定)

これらの項目は内部で使用することを目的としていますが、アプリケーションでも使用できます。アプリケーションで、Mobile Sync プラグインを使用してサーバーに上位同期する場合は、通常、同期元スープでこれらの項目を作成する必要があります。手順は、「[Mobile Sync のスープの準備](#)」を参照してください。

StoreCache は、\_\_local\_\_ 項目およびその ID 項目の各スープにインデックスを付けます。

additionalIndexSpecs パラメーターを使用して、インデックスに含める追加項目を指定できます。

基盤となるスープを登録するには、StoreCache オブジェクトで init() をコールします。この関数は、スープの登録が完了したときに解決する jQuery promise を返します。



## StoreCache メソッド

### init()

基盤となる SmartStore スープを登録します。スープの登録が完了したときに解決する jQuery promise を返します。

### retrieve(key [, fieldlist])

SmartStore によって返される keyField のキーを持つレコードに解決する jQuery promise を返します。レコードが検出されない場合や、検出されたレコードに fieldlist パラメーターのすべての項目が含まれていない場合、promise は null に解決されます。

#### key

取得されるレコードのキー値。

#### fieldlist

(省略可能) 必須項目の JavaScript 配列。次に例を示します。

```
["field1", "field2", "field3"]
```

### save(record [, noMerge])

SmartStore の更新/挿入が完了したときに、保存されたレコードに解決する jQuery promise を返します。noMerge が指定されていない場合や、false の場合、渡されたレコードは、同じキーのサーバーレコード (存在する場合) とマージされます。

#### record

保存されるレコード。次のような形式になります。

```
{<field_name1>:"<field_value1>"[,<field_name2>:"<field_value2>",...]}
```

次に例を示します。

```
{Id:"007", Name:"JamesBond", Mission:"TopSecret"}
```

#### noMerge

(省略可能) 渡されたレコードが、一致するサーバーレコードとマージされるかどうかを示す boolean 値。デフォルトは false です。

### saveAll(records [, noMerge])

records が、保存されるレコードの配列である点を除き、save () と同じです。保存されるレコードに解決する jQuery promise を返します。

#### records

レコードの配列。配列の各項目は、save () 関数に示されるように書式設定されます。

#### noMerge

(省略可能) 渡されたレコードが、一致するサーバーレコードとマージされるかどうかを示す boolean 値。デフォルトは false です。

### remove(key)

指定されたキーを持つレコードが SmartStore から削除されたときに解決する jQuery promise を返します。

#### key

削除されるレコードのキー値。

**find(querySpec)**

SmartStore に対してクエリが実行されたときに解決する jQuery promise を返します。解決される値は、次の項目を持つオブジェクトです。

| 項目          | 説明                          |
|-------------|-----------------------------|
| records     | 取得されたすべてのレコード               |
| hasMore     | さらにレコードを取得できるかどうかをチェックする関数  |
| getMore     | さらにレコードを取得する関数              |
| closeCursor | 開いているカーソルを閉じて、今後の取得を無効にする関数 |

**querySpec**

SmartStore クエリ関数コールに基づく仕様。次のような形式になります。

```
{queryType: "like" | "exact" | "range" | "smart"[, query_type_params]}
```

ここで、query\_type\_params は、関連する SmartStore クエリ関数コールの形式に一致します。「[スプからのデータの取得](#)」を参照してください。

次に、いくつか例を示します。

```
{queryType:"exact", indexPath:"<indexed_field_to_match_on>", matchKey:<value_to_match>,
 order:"ascending"|"descending", pageSize:<entries_per_page>}

{queryType:"range", indexPath:"<indexed_field_to_match_on>", beginKey:<start_of_range>,
 endKey:<end_of_range>, order:"ascending"|"descending", pageSize:<entries_per_page>}

{queryType:"like", indexPath:"<indexed_field_to_match_on>", likeKey:"<value_to_match>",
 order:"ascending"|"descending", pageSize:<entries_per_page>}

{queryType:"smart", smartSql:"<smart_sql_query>", order:"ascending"|"descending",
 pageSize:<entries_per_page>}
```

**例**

次の例では、StoreCache オブジェクトの作成、初期化、使用方法を示します。

```
var cache = new Force.StoreCache("agents", [{path:"Mission", type:"string"}]);
// initialization of the cache / underlying soup
cache.init()
.then(function() {
 // saving a record to the cache
 return cache.save({Id:"007", Name:"JamesBond", Mission:"TopSecret"});
})
.then(function(savedRecord) {
 // retrieving a record from the cache
```

```

 return cache.retrieve("007");
 })
 .then(function(retrievedRecord) {
 // searching for records in the cache
 return cache.find({queryType:"like", indexPath:"Mission", likeKey:"Top%",
order:"ascending", pageSize:1});
 })
 .then(function(result) {
 // removing a record from the cache
 return cache.remove("007");
 });

```

次の例では、`saveAll()` 関数の使用方法と `find()` 関数の結果を示します。

```

// initialization
var cache = new Force.StoreCache("agents", [{path:"Name", type:"string"}, {path:"Mission",
type:"string"}]);
cache.init()
.then(function() {
 // saving some records
 return cache.saveAll([{Id:"007", Name:"JamesBond"},{Id:"008", Name:"Agent008"},
{Id:"009", Name:"JamesOther"}]);
})
.then(function() {
 // doing an exact query
 return cache.find({queryType:"exact", indexPath:"Name", matchKey:"Agent008",
order:"ascending", pageSize:1});
})
.then(function(result) {
 alert("Agent mission is:" + result.records[0]["Mission"]);
});

```

## 競合検出

モデルオブジェクトは、オブジェクトをサーバーに保存するときに不要なデータ損失を回避する競合検出 (省略可能) をサポートしています。競合検出は、デバイスがオフライン状態から復帰しているかどうかに関係なく、任意の保存操作と共に使用できます。

競合検出をサポートするには、サーバーから取得した元の値を格納するセカンダリキャッシュを指定します。Mobile Sync は、後で参照できるようにこのキャッシュを保持します。保存または削除する場合、マージモードを指定します。次の表に、サポートされるモードの概要を示します。モードの説明を理解するには、「theirs」を現在のサーバーレコード、「yours」を現在のローカルレコード、「base」をサーバーから最初に取得したレコードとして考えます。

| モードの定数                                  | 説明                                              |
|-----------------------------------------|-------------------------------------------------|
| <code>Force.MERGE_MODE.OVERWRITE</code> | 「theirs」または「base」と比較することなく「yours」をサーバーに書き込みます。こ |

## モードの定数

## 説明

Force.MERGE\_MODE.MERGE\_ACCEPT\_YOURS

これは、競合検出を使用しない場合と同じです。

Force.MERGE\_MODE.MERGE\_FAIL\_IF\_CONFLICT

「theirs」と「yours」をマージします。同じ項目がローカルとリモートの両方で変更された場合、ローカルの値が保持されます。

Force.MERGE\_MODE.MERGE\_FAIL\_IF\_CHANGED

「theirs」と「yours」をマージします。同じ項目がローカルとリモートの両方で変更された場合、この操作は失敗します。

「theirs」と「yours」をマージします。何らかの項目がリモートで変更された場合、この操作は失敗します。

保存または削除操作が失敗した場合、次の項目を含むレポートオブジェクトを取得します。

## 項目名

## 次の文字列を含む

base

最初に取得した属性

theirs

最新のサーバー属性

yours

ローカルで変更された属性

remoteChanges

base と theirs 間で変更された項目のリスト

localChanges

base と yours 間で変更された項目のリスト

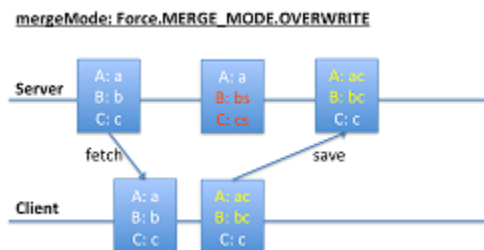
conflictingChanges

異なる値を使用して theirs と yours の両方で変更された項目のリスト

ダイアグラムは、マージモードの仕組みを明確にするのに役立ちます。

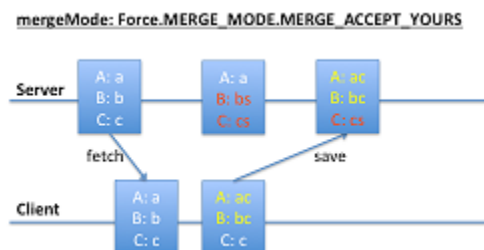
## MERGE\_MODE.OVERWRITE

MERGE\_MODE.OVERWRITE ダイアグラムでは、クライアントは A と B を変更し、サーバーは B と C を変更しています。B への変更は競合していますが、A と C への変更は競合していません。ただし、保存操作では、すべてのクライアントの値が無差別にサーバーに書き込まれるため、サーバーの変更が上書きされます。



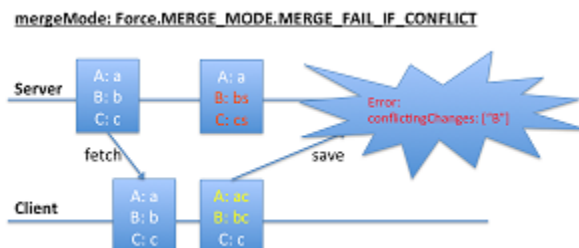
## MERGE\_ACCEPT\_YOURS

MERGE\_MODE.MERGE\_ACCEPT\_YOURS ダイアグラムでは、クライアントは A と B を変更し、サーバーは B と C を変更しています。競合があるかどうかに関係なく、クライアントの変更 (A と B) によってサーバーの対応する項目が上書きされます。ただし、クライアントが変更しなかった項目 (C) では、対応するサーバーの値は上書きされません。



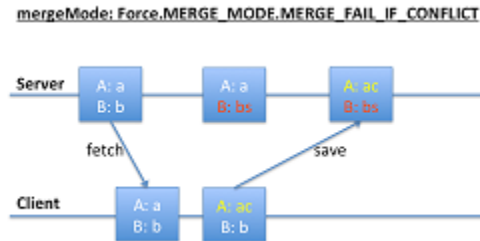
## MERGE\_FAIL\_IF\_CONFLICT (失敗)

最初の MERGE\_MODE.MERGE\_FAIL\_IF\_CONFLICT ダイアグラムでは、クライアントとサーバーの両方が B を変更しています。これらの競合する変更が原因で保存操作が失敗します。



## MERGE\_FAIL\_IF\_CONFLICT (成功)

2 番目の MERGE\_MODE.MERGE\_FAIL\_IF\_CONFLICT ダイアグラムでは、クライアントは A を変更し、サーバーは B を変更しています。これらの変更は競合しないため、保存操作は成功します。



## ミニチュートリアル: 競合検出

次のミニチュートリアルでは、さまざまな状況下でマージモードが保存操作にどのような影響を与えるかについて説明します。HTML コンテキスト内のさまざまな例を挙げて説明します。

1. 必要なキャッシュを設定します。

```
var cache = new Force.StoreCache(soupName);
var cacheForOriginals =
 new Force.StoreCache(soupNameForOriginals);
var Account = Force.SObject.extend({
 objectType: "Account",
 fieldlist: ["Id", "Name", "Industry"],
 cache: cache,
 cacheForOriginals: cacheForOriginals});
```

2. 既存の取引先を取得します。

```
var account = new Account({Id: <some actual account id>});
account.fetch();
```

3. 取引先の名前が「Acme」で業種が「ソフトウェア」だとします。名前を「Acme2」に変更します。

```
Account.set("Name", "Acme2");
```

4. デフォルトの "overwrite" マージモードが使用されるように、マージモードを指定せずにサーバーに保存します。

```
account.save(null);
```

取引先の名前が「Acme2」でその業種が「ソフトウェア」になりました。サーバー上の業種が「電子機器」に変更されるとします。

5. 取引先名を再度変更します。

```
Account.set("Name", "Acme3");
```

現在、キャッシュの変更 (名前) とサーバーの変更 (業種) があります。

6. "merge-fail-if-changed" マージモードを使用して再度保存します。

```
account.save(null,
 {mergeMode: "merge-fail-if-changed", error: function(err) {
 // err will be a map of the form:
 // {base:..., theirs:..., yours:...,
 // remoteChanges:["Industry"], localChanges:["Name"],
 // conflictingChanges:[]}}
 });
```

サーバーレコードが変更されているため、エラーコールバックがコールされます。

7. "merge-fail-if-conflict" マージモードを使用して再度保存します。サーバーの変更とクライアントの変更間で競合が存在しないため、このマージは成功します。

```
account.save(null, {mergeMode: "merge-fail-if-conflict"});
```

取引先の名前は「Acme3」(yours)で、その業種は「電子機器」(theirs)になりました。一方、サーバー上の名前が「NewAcme」、業種が「サービス」に変更されるとします。

8. 取引先名を再度変更します。

```
Account.set("Name", "Acme4");
```

9. "merge-fail-if-changed" マージモードを使用して再度保存します。サーバーレコードが変更されているため、エラーコールバックがコールされます。

```
account.save(null, {mergeMode: "merge-fail-if-changed", error: function(err) {
 // err will be a map of the form:
 // {base:..., theirs:..., yours:...,
 // remoteChanges:["Name", "Industry"],
 // localChanges:["Name"], conflictingChanges:["Name"]}}
});
```

10. "merge-fail-if-conflict" マージモードを使用して再度保存します。

```
account.save(null, {mergeMode: "merge-fail-if-changed", error: function(err) {
 // err will be a map of the form:
 // {base:..., theirs:..., yours:...,
 // remoteChanges:["Name", "Industry"],
 // localChanges:["Name"], conflictingChanges:["Name"]}}
});
```

サーバーとキャッシュの両方で名前項目が変更されて競合が発生しているため、エラーコールバックがコールされます。

11. "merge-accept-yours" マージモードを使用して再度保存します。マージモードでは、どちらの名前の値を受け入れるかが `save()` 関数に通知されているため、このマージは成功します。また、業種を変更していないため、この項目は競合しません。

```
account.save(null, {mergeMode: "merge-accept-yours"});
```

名前は「Acme4」(yours)で、業種は「サービス」(theirs)です。どちらもキャッシュとサーバーにあります。



## カスタム API エンドポイントへのアクセス

Mobile SDK 2.1 では、Mobile Sync の範囲が拡張されており、REST API と併用できるようになっています。以前は、Salesforce Platform API を使用して sObject に基本的な操作しか実行できませんでした。Apex REST オブジェクト、Chatter Files、および他の Salesforce REST API と Mobile Sync を併用できるようになりました。また、非 Salesforce REST API をコールすることもできます。

### Force.RemoteObject クラス

任意の REST コールをサポートするために、Mobile Sync には Force.RemoteObject 抽象クラスが導入されています。Force.RemoteObject は、Force.SObject と Backbone.Model 間の抽象化レイヤーとして機能します。Force.SObject は、Backbone.Model を直接サブクラス化するのではなく、Backbone.Model をサブクラス化する Force.RemoteObject をサブクラス化するようになりました。Force.RemoteObject は、サーバーとの通信を除き、Force.SObject が以前に行っていたすべてのことを実行します。

### syncRemoteObjectWithServer() を使用したカスタムエンドポイントのコール

RemoteObject.syncRemoteObjectWithServer() プロトタイプメソッドは、サーバーとのやりとりを処理します。Force.SObject は、syncRemoteObjectWithServer() を実装して Salesforce Platform REST API を使用します。他のサーバーエンドポイントを使用する場合、Force.RemoteObject のサブクラスを作成し、syncRemoteObjectWithServer() を実装します。このメソッドは、サブクラスのオブジェクトに対して fetch() をコールしたときにコールされます(サーバーから取得するようにオブジェクトが現在設定されている場合)。

#### 例: 例

FileExplorer サンプルアプリケーションは、Force.RemoteObject の使用方法を示す Mobile Sync アプリケーションです。HybridFileExplorer は、Connect REST API をコールしてファイル进行操作します。これは、Force.RemoteObject を拡張する app.models.File オブジェクトを定義します。app.models.File は、syncRemoteObjectWithServer() の実装で /chatter/files/fileId REST API をラップする Force.forceJsClient.fileDetails() をコールします。

```
app.models.File = Force.RemoteObject.extend({
 syncRemoteObjectWithServer: function(method, id) {
 if (method !== "read")
 throw "Method not supported " + method;
 return Force.forceJsClient.fileDetails(id, null);
 }
});
```

### Force.RemoteObjectCollection クラス

取得されたオブジェクトのコレクションをサポートするために、Mobile Sync には Force.RemoteObjectCollection 抽象クラスが導入されています。このクラスは、Force.SObjectCollection と Backbone.Collection 間の抽象化レイヤーとして機能します。Force.SObjectCollection は、Backbone.Collection を直接サブクラス化するのではなく、Backbone.Collection をサブクラス化する Force.RemoteObjectCollection をサブクラス化するようになりました。Force.RemoteObjectCollection は、サーバーとの通信を除き、Force.SObjectCollection が以前に行っていたすべてのことを実行します。

## fetchRemoteObjectFromServer() を使用してカスタムエンドポイントを実装する

`RemoteObject.fetchRemoteObjectFromServer()` プロトタイプメソッドは、サーバーとのやりとりを処理します。このメソッドは REST API を使用して SOQL/SOSL および MRU クエリを実行します。任意のサーバーエンドポイントを使用する場合、`Force.RemoteObjectCollection` のサブクラスを作成し、`fetchRemoteObjectFromServer()` を実装します。このメソッドは、サブクラスのオブジェクトに対して `fetch()` をコールしたときにコールされます(サーバーから取得するようにオブジェクトが現在設定されている場合)。

`app.models.FileCollection.fetchRemoteObjectsFromServer()` 関数が返されると、応答のメタデータを使用する有効な情報や、便利な関数を格納するオブジェクトが promise 化されます。このオブジェクトには、次が含まれます。

- `totalSize`: 返されたコレクションのファイル数
- `records`: 返されたファイルのコレクション
- `hasMore`: 結果の別のページを取得できるかどうかを示す boolean 値を返す関数
- `getMore`: 結果の次のページを取得する関数 (`hasMore()` で true が返された場合)
- `closeCursor`: コレクションの反復処理が終了したことを示す関数

これらの関数は、サーバー応答に含まれている情報 (`Files.length` や `nextPageUrl` など) を利用します。

### 例: 例

HybridFileExplorer サンプルアプリケーションでは、`Force.RemoteObjectCollection` の使用方法も示します。この例では、`ConnectRESTAPI` をコールして、ファイルのリストを反復処理します。`ownedFilesList`、`filesInUsersGroups`、および `filesSharedWithUser` の 3 つの REST 操作がサポートされています。

この例の `hasMore()` や `getMore()` などの関数を記述して、結果のページを移動できます。ただし、アプリケーションは直接 `fetchRemoteObjectsFromServer()` をコールしないため、コレクションオブジェクトで `fetch()` をコールすると返される promise オブジェクトを取得します。

```
app.models.FileCollection = Force.RemoteObjectCollection.extend({
 model: app.models.File,

 setCriteria: function(key) {
 this.config = {type:key};
 },

 fetchRemoteObjectsFromServer: function(config) {
 var fetchPromise;
 switch(config.type) {
 case "ownedFilesList": fetchPromise =
 Force.forceJsClient.ownedFilesList("me", 0);
 break;
 case "filesInUsersGroups": fetchPromise =
 Force.forceJsClient.
 filesInUsersGroups("me", 0);
 break;
 case "filesSharedWithUser": fetchPromise =
 Force.forceJsClient.
 filesSharedWithUser("me", 0);
 break;
 }
 }
});
```

```

 };

 return fetchPromise
 .then(function(resp) {
 var nextPageUrl = resp.nextPageUrl;
 return {
 totalSize: resp.files.length,
 records: resp.files,
 hasMore: function() {
 return nextPageUrl != null; },
 getMore: function() {
 var that = this;
 if (!nextPageUrl)
 return null;
 return
 forceJsClient.queryMore(nextPageUrl)
 .then(function(resp) {
 nextPageUrl = resp.nextPageUrl;
 that.records.
 pushObjects(resp.files);
 return resp.files;
 });
 },
 closeCursor: function() {
 return $.when(function() {
 nextPageUrl = null;
 });
 }
 };
 });
 }
});

```

## Apex REST リソースの使用

Apex REST リソースをサポートするために、Mobile SDK には、`Force.ApexRestObject` と `Force.ApexRestObjectCollection` の2つのクラスが用意されています。これらのクラスは、それぞれ `Force.RemoteObject` と `Force.RemoteObjectCollection` をサブクラス化し、Apex REST を使用して作成した REST API と通信できます。

### **Force.ApexRestObject**

`Force.ApexRestObject` は、`Force.SObject` と似ています。 `sobjectType` の代わりに、`Force.ApexRestObject` には `services/apexrest` を基準とする Apex REST リソースの相対パスが必要です。たとえば、リソースのフルパスが `services/apexrest/simpleAccount/*` の場合、`/simpleAccount/*` のみを指定します。また、`Force.ApexRestObject` では、ID 項目名が「Id」以外の場合、その名前を指定する必要があります。

 例: 例

accountId と accountName の2つの項目を含む取引先である「simple account」（簡単な取引先）という名前の Apex REST リソースを作成したとします。

```
@RestResource(urlMapping='/simpleAccount/*')
global with sharing class SimpleAccountResource {
 static String getIdFromURI() {
 RestRequest req = RestContext.request;
 return req.requestURI.substring(req.requestURI.lastIndexOf('/')+1);
 }

 @HttpGet global static Map<String, String> doGet() {
 String id = getIdFromURI();
 Account acc = [select Id, Name from Account
 where Id = :id];
 return new Map<String, String>({
 'accountId'=>acc.Id, 'accountName'=>acc.Name});
 }

 @HttpPost global static Map<String, String>
 doPost(String accountName) {
 Account acc = new Account(Name=accountName);
 insert acc;
 return new Map<String, String>({
 'accountId'=>acc.Id, 'accountName'=>acc.Name});
 }

 @HttpPatch global static Map<String, String>
 doPatch(String accountName) {
 String id = getIdFromURI();
 Account acc = [select Id from Account
 where Id = :id];
 acc.Name = accountName;
 update acc;
 return new Map<String, String>({
 'accountId'=>acc.Id, 'accountName'=>acc.Name});
 }

 @HttpDelete global static void doDelete() {
 String id = getIdFromURI();
 Account acc = [select Id from Account where Id = :id];
 delete acc;
 RestContext.response.statusCode = 204;
 }
}
```

Mobile Sync で、次を実行して「simple account」を作成します。

```
var SimpleAccount = Force.ApexRestObject.extend(
 {apexRestPath:"/simpleAccount",
 idAttribute:"accountId",
 fieldlist:["accountId", "accountName"]});
var acc = new SimpleAccount({accountName:"MyFirstAccount"});
acc.save();
```

その「simple account」を更新できます。

```
acc.set("accountName", "MyFirstAccountUpdated");
acc.save(null, {fieldlist:["accountName"]});
// our apex patch endpoint only expects accountName
```

別の「simple account」を取得できます。

```
var acc2 = new SimpleAccount({accountId:"<valid id>"})
acc.fetch();
```

「simple account」を削除できます。

```
acc.destroy();
```

 **メモ:** 通常、Mobile Sync のコール (fetch()、save()、および destroy() など) では、成功コールバック関数およびエラーコールバック関数を定義するオプションパラメーターを渡します。次に例を示します。

```
acc.destroy({success:function(){alert("delete succeeded");}});
```

### Force.ApexRestObjectCollection

Force.ApexRestObjectCollection は、Force.SObjectCollection と似ています。取得用に指定する設定では、SOQL、SOSL、または MRU がサポートされていません。代わりに、services/apexrest を基準とする Apex REST リソースの相対パスが必要です。たとえば、リソースのフルパスが services/apexrest/simpleAccount/\* の場合、/simpleAccount/\* のみを指定します。

エンドポイントでクエリ文字列のパラメーターがサポートされていれば、これらを渡すこともできます。Apex REST エンドポイントは、次の形式で応答を返す必要があります。

```
{
 totalSize: <number of records returned>
 records: <all fetched records>
 nextRecordsUrl: <url to get next records or null>
}
```

### 例: 例

「simple account」という名前の Apex REST リソースを作成したとします。これは、指定された名前と一致する「simple account」を返します。

```
@RestResource(urlMapping='/simpleAccounts/*')
global with sharing class SimpleAccountsResource {
 @HttpGet global static SimpleAccountsList doGet() {
 String namePattern =
 RestContext.request.params.get('namePattern');
 List<SimpleAccount> records = new List<SimpleAccount>();
 for (SObject sobj : Database.query(
 'select Id, Name from Account
 where Name like \'' + namePattern + '\'')) {
 Account acc = (Account) sobj;
 records.add(new
 SimpleAccount(acc.Id, acc.Name));
 }
 }
}
```

```
 return new SimpleAccountsList(records.size(), records);
 }

 global class SimpleAccountsList {
 global Integer totalSize;
 global List<SimpleAccount> records;

 global SimpleAccountsList(Integer totalSize,
 List<SimpleAccount> records) {
 this.totalSize = totalSize;
 this.records = records;
 }
 }

 global class SimpleAccount {
 global String accountId;
 global String accountName;

 global SimpleAccount(String accountId, String accountName)
 {
 this.accountId = accountId;
 this.accountName = accountName;
 }
 }
}
```

Mobile Sync で、次を実行して「simple account」レコードのリストを取得します。

```
var getSimple = function() {
 console.log("## Trying fetch with apex rest end point");
 var config = {
 apexRestPath: "/simpleAccounts",
 params: {namePattern: accountNamePrefix + "%"}
 }
 return Force.fetchApexRestObjectsFromServer(config);
}
```

## チュートリアル: ハイブリッド Mobile Sync アプリケーションの作成

このチュートリアルでは、Mobile Sync を使用するローカルハイブリッドアプリケーションの作成方法について説明します。Mobile SDK に付属する UserSearch のサンプルアプリケーションを再作成します。UserSearch では、Salesforce 組織のユーザーレコードを検索し、ユーザーレコードに関する基本情報の詳細を確認できます。

このサンプルでは、次の Web テクノロジーを使用しています。

- Backbone.js
- Ratchet
- HTML5
- JavaScript

## テンプレートプロジェクトの作成

まず、NPM インストーラーを使用して Salesforce Mobile SDK をインストールしていることを確認します。iOS の手順は、「[iOS の準備](#)」を参照してください。Android の手順は、「[Android の準備](#)」を参照してください。

また、<http://goratchet.com/> から `ratchet.css` ファイルをダウンロードします。

Mobile SDK をインストールしたら、プラットフォームのローカルハイブリッドプロジェクトを作成します。

1. ターミナルウィンドウまたは Windows コマンドプロンプトで、次の値を使用して `forcehybrid create` を実行します。

| プロンプト (パラメーター)         | 値                                                                     |
|------------------------|-----------------------------------------------------------------------|
| プラットフォーム (--platform)  | <code>ios</code> 、 <code>android</code> 、または <code>ios,android</code> |
| アプリケーション種別 (--apptype) | <code>hybrid_local</code>                                             |
| アプリケーション名 (--appname)  | <code>UserSearch</code>                                               |
| パッケージ名 (--packagename) | <code>com.acme.usersearch</code>                                      |
| 組織 (--organization)    | <code>"Acme Widgets, Inc."</code>                                     |
| 出力ディレクトリ (--outputdir) | 現在のディレクトリの場合は空白にします。プロジェクトの新しいサブディレクトリを作成する場合は名前を入力します。               |

次に、コマンドラインの例を示します。

```
forcehybrid create --platform=ios,android --apptype=hybrid_local
--appname=UserSearch --packagename=com.acme.usersearch
--organization="Acme Widgets, Inc." --outputdir=""
```

2. 次のように、すべてのファイル (実際のファイルとシンボリックファイル) を <https://github.com/forcedotcom/SalesforceMobileSDK-Shared/> リポジトリの `samples/usersearch` ディレクトリから `www/` フォルダーにコピーします。

- Mac OS X ターミナルウィンドウで、プロジェクトのルートディレクトリ (`./UserSearch/`) に変更し、次のコマンドを入力します。

```
cp -RL <insert local path to SalesforceMobileSDK-Shared>/samples/UserSearch/* www/
```

- Windows で、`<shared_repo>\samples\usersearch` フォルダーで参照されるすべてのファイルが `<project_name>\www` フォルダーにも表示されていることを確認します。次のスクリプトに示すように、シンボリックリンクを明示的に解決します。

```
cd <your project's root directory>
set SHARED_REPO=<insert local path to SalesforceMobileSDK-Shared>
copy %SHARED_REPO%\samples\usersearch\UserSearch.html www
copy %SHARED_REPO%\samples\usersearch\bootconfig.json www
copy %SHARED_REPO%\dependencies\ratchet\ratchet.css www
copy %SHARED_REPO%\samples\common\styles.css www
```



```
copy %SHARED_REPO%\test\MockCordova.js www
copy %SHARED_REPO%\samples\common\auth.js www
copy %SHARED_REPO%\dependencies\backbone\backbone-min.js www
copy %SHARED_REPO%\libs\cordova.force.js www
copy %SHARED_REPO%\dependencies\fastclick\fastclick.js www
copy %SHARED_REPO%\libs\force.js www
copy %SHARED_REPO%\libs\force+promise.js www
copy %SHARED_REPO%\dependencies\jquery\jquery.min.js www
copy %SHARED_REPO%\libs\mobilesync.js www
copy %SHARED_REPO%\samples\common\stackrouter.js www
copy %SHARED_REPO%\dependencies\underscore\underscore-min.js www
```

### 3. 次のコマンドを実行します。

```
cordova prepare
```

4. forcehybrid で出力される画面の指示に従って、Android Studio (Android の場合) または Xcode (iOS の場合) で platforms/android/ プロジェクトフォルダーを開きます。
5. www フォルダーから、コードエディターを使用して UserSearch.html を開き、そのすべてのコンテンツを削除します。

## アプリケーションの HTML ファイルの編集

アプリケーションの基本構造を作成するには、参照、リンク、コードインフラストラクチャを格納する空の HTML ページを定義します。

1. www フォルダーから、コードエディターを使用して UserSearch.html を開き、そのすべてのコンテンツを削除します。
2. コンテンツを削除して次の基本構造を追加します。

```
<!DOCTYPE html>
<html>
 <head>
 </head>
 <body>
 </body>
</html>
```

### 3. <head> 要素で、次の操作を行います。

- a. ページタイトルが「Users」であることを指定します。

```
<title>Users</title>
```

- b. Web ページではなくアプリケーションのようなページにするために拡大縮小を無効にします。

```
<meta name="viewport" content="width=device-width, initial-scale=1.0,
 maximum-scale=1.0, user-scalable=no;" />
```

- c. モバイル用の「外観」になるように `styles.css` および `ratchet.css` ファイルへのリンクを追加します。

```
<link rel="stylesheet" href="css/styles.css"/>
<link rel="stylesheet" href="css/ratchet.css"/>
```

4. コンテンツを本文に追加します。`<body>` ブロックで、ID が「content」に設定された空の `div` タグを追加して、アプリケーションの生成された UI を格納します。

```
<body>
<div id="content"></div>
```

5. 必要な JavaScript ファイルを含めます。

```
<script src="js/jquery.min.js"></script>
<script src="js/underscore-min.js"></script>
<script src="js/backbone-min.js"></script>
<script src="cordova.js"></script>
<script src="js/force.js"></script>
<script src="js/force+promise.js"></script>
<script src="js/mobilesync.js"></script>
<script src="js/fastclick.js"></script>
<script src="js/stackrouter.js"></script>
<script src="js/auth.js"></script>
```



例: 次に、この時点の完全なアプリケーションを示します。

```
<!DOCTYPE html>
<html>
 <head>
 <title>Users</title>
 <meta name="viewport" content="width=device-width,
 initial-scale=1.0, maximum-scale=1.0;
 user-scalable=no" />
 <link rel="stylesheet" href="css/styles.css"/>
 <link rel="stylesheet" href="css/ratchet.css"/>
 </head>
 <body>
 <div id="content"></div>
 <script src="js/jquery.min.js"></script>
 <script src="js/underscore-min.js"></script>
 <script src="js/backbone-min.js"></script>
 <script src="cordova.js"></script>
 <script src="js/force.js"></script>
 <script src="js/force+promise.js"></script>
 <script src="js/mobilesync.js"></script>
 <script src="js/fastclick.js"></script>
 <script src="js/stackrouter.js"></script>
 <script src="js/auth.js"></script>
 </body>
</html>
```

## Mobile Sync モデルおよびコレクションの作成

HTML インフラストラクチャを設定したので、次の2つの主オブジェクトを拡張して Mobile Sync を使用します。

- Force.SObject
- Force.SObjectCollection

これらのオブジェクトは、Backbone.Model.extend() 関数をサポートできるように Backbone.Model を拡張します。この関数を使用してオブジェクトを拡張するには、カスタムプロパティおよびカスタム機能を含む JavaScript オブジェクトに渡します。

1. <body> タグで、<script> オブジェクトを作成します。
2. <script> タグで、SalesforceusersObject のモデルオブジェクトを作成します。Force.SObject を拡張して、対象とする sObject タイプおよび項目を指定します。

```
app.models.User = Force.SObject.extend({
 sobjectType: "User",
 fieldlist: ["Id", "FirstName", "LastName",
 "SmallPhotoUrl", "Title", "Email",
 "MobilePhone", "City"]
});
```

3. User オブジェクトの設定直後に、UserSearch 結果を保持する UserCollection オブジェクトを作成します。Force.SObjectCollection を拡張して、コレクションの項目のモデルとして新しいモデル (app.models.User) を指定します。

```
app.models.UserCollection = Force.SObjectCollection.extend({
 model: app.models.User,
 fieldlist: ["Id", "FirstName", "LastName",
 "SmallPhotoUrl", "Title"],
});
```

4. このコレクションでは、検索キーを取得し、それを使用して SOQL クエリを作成する setCriteria という名前関数を実装します。また、後でキーを返す getter も必要です。

```
<script>
 // The Models
 // =====
 // The User Model
 app.models.User = Force.SObject.extend({
 sobjectType: "User",
 fieldlist: ["Id", "FirstName",
 "LastName", "SmallPhotoUrl",
 "Title", "Email",
 "MobilePhone", "City"]
 });

 // The UserCollection Model
 app.models.UserCollection = Force.SObjectCollection.extend({
 model: app.models.User
 fieldlist: ["Id", "FirstName", "LastName",
 "SmallPhotoUrl", "Title"],
 });
```

```

getCriteria: function() {
 return this.key;
},

setCriteria: function(key) {
 this.key = key;
 this.config = {type:"soql", query:"SELECT "
 + this.fieldlist.join(",")
 + " FROM User"
 + " WHERE Name like '" + key + "%'"
 + " ORDER BY Name "
 + " LIMIT 25 "
 };
}

});
</script>

```

 **例:** 次に、完全なモデルコードを示します。

```

<script>
 // The Models

 // The User Model
 app.models.User = Force.SObject.extend({
 objectType: "User",
 fieldlist: ["Id", "FirstName", "LastName",
 "SmallPhotoUrl", "Title", "Email",
 "MobilePhone","City"]
 });

 // The UserCollection Model
 app.models.UserCollection = Force.SObjectCollection.extend({
 model: app.models.User
 fieldlist: ["Id", "FirstName", "LastName",
 "SmallPhotoUrl", "Title"],

 getCriteria: function() {
 return this.key;
 },

 setCriteria: function(key) {
 this.key = key;
 this.config = {
 type:"soql",
 query:"SELECT " + this.fieldlist.join(",")
 + " FROM User"
 + " WHERE Name like '" + key + "%'"
 + " ORDER BY Name "
 + " LIMIT 25 "
 };
 }
 });
</script>

```

## ビューテンプレートの作成

テンプレートを使用すると、コンテナの HTML ページ内に HTML レイアウトを設定できます。HTML ページにインラインテンプレートを定義するには、タイプが「text/template」の `<script>` タグを使用します。JavaScript コードでは、実行時に新しい HTML ページをインスタンス化する場合、テンプレートをページの設計に適用できます。

`search-page` は、単純なテンプレートです。このテンプレートには、ヘッダー、検索項目、および検索結果を保持するリストが含まれます。検索ページでは、実行時に `user-list-item` テンプレートがインスタンス化されて、結果リストが表示されます。顧客がリスト項目をクリックすると、リストで `user-page` テンプレートがインスタンス化されてユーザーの詳細が表示されます。

1. ID が「`search-page`」に設定されたテンプレートスクリプトブロックを追加します。`<body>` ブロック内の「`content`」 `<div>` タグの後にこのブロックを配置します。

```
<script id="search-page" type="text/template">
</script>
```

2. 新しい `<script>` ブロックで、Ratchet スタイルを使用して検索ページの HTML テンプレートを定義します。

```
<script id="search-page" type="text/template">
 <header class="bar-title">
 <h1 class="title">Users</h1>
 </header>

 <div class="bar-standard bar-header-secondary">
 <input type="search" class="search-key"
 placeholder="Search"/>
 </div>

 <div class="content">
 <ul class="list">
 </div>
</script>
```

3. ユーザーリストテンプレートの 2 番目のスクリプトブロックを追加します。

```
<script id="user-list-item" type="text/template">
</script>
```

4. ユーザーリストテンプレートを定義します。このテンプレートには、Salesforce ユーザーレコードの `SmallPhotoUrl`、`FirstName`、`LastName`、`Title` 項目への参照が含まれています。`<%= varname %>` の形式を使用する参照は、Ratchet アプリケーションでは「自由変数」と呼ばれます。

```
<script id="user-list-item" type="text/template">
 <a href="#users/<%= Id %>" class="pad-right">

 <div class="details-short">
 <%= FirstName %> <%= LastName %>

 Title<%= Title %>
 </div>

</script>
```

5. ユーザー詳細テンプレートの3番目のスクリプトブロックを追加します。

```
<script id="user-page" type="text/template">
</script>
```

6. テンプレート本文を追加します。このテンプレートには、Salesforce ユーザーレコードの `SmallPhotoUrl`、`FirstName`、`LastName`、`Title` 項目への参照が含まれています。Ratchet アプリケーションで `<%= varname %>` の形式を使用する参照は、「自由変数」と呼ばれます。

```
<script id="user-page" type="text/template">
 <header class="bar-title">
 Back
 <h1 class="title">User</h1>
 </header>

 <footer class="bar-footer">

 </footer>

 <div class="content">
 <div class="content-padded">

 <div class="details">
 <%= FirstName %> <%= LastName %>

 <%= Id %>

 <% if (Title) { %><%= Title %>
<% } %>
 <% if (City) { %><%= City %>
<% } %>
 <% if (MobilePhone) { %> <a
 href="tel:<%= MobilePhone %>">
 <%= MobilePhone %>
<% } %>
 <% if (Email) { %><a
 href="mailto:<%= Email %>">
 <%= Email %>
<% } %>
 </div>
 </div>
 </div>
</script>
```

## 検索ビューの追加

画面のビューを作成するには、`Backbone.View` を拡張します。まず、検索ビューの定義から始めましょう。この拡張で、テンプレートの読み込み、サブビューとイベントハンドラーの定義、およびビューを表示して SOQL 検索クエリを実行する機能の実装を行います。

1. User および UserCollection モデルを定義した `<script>` ブロックで、`SearchPage` という名前の `Backbone.View` 拡張を `app.views` 配列に作成します。

```
app.views.SearchPage = Backbone.View.extend({
});
```

この手順の残りの部分で、すべてのコードを `extend({})` ブロックに追加します。各ステップで別の項目を実装リストに追加するため、最後の項目まで末尾はカンマになります。

2. `_.template()` 関数をコールして `search-page` テンプレートを読み込みます。これに `search-page` スクリプトタグの未加工の HTML コンテンツを渡します。

```
template: _.template($("#search-page").html()),
```

3. `keyup` イベントを追加します。 `search` ハンドラー関数はもう少し後で定義します。

```
events: {
 "keyup .search-key": "search"
},
```

4. 検索結果のリストを格納する `UserListView` という名前のサブビューをインスタンス化します。後で `app.views.UserListView` を定義します。

```
initialize: function() {
 this.listView = new app.views.UserListView({model: this.model});
},
```

5. 検索ページビューの `render()` 関数を作成します。アプリケーションの HTML コンテンツとしてテンプレートを読み込んでビューを表示できます。検索項目に以前に入力した条件を復元し、`<ul>` 要素内のサブビューを表示します。

```
render: function(eventName) {
 $(this.el).html(this.template());
 $(".search-key", this.el).val(this.model.getCriteria());
 this.listView.setElement($(".ul", this.el)).render();
 return this;
},
```

6. `search` 関数を実装します。この関数は、顧客が検索項目に文字を入力したときに検索を実行する `keyup` イベントハンドラーです。

```
search: function(event) {
 this.model.setCriteria($(".search-key", this.el).val());
 this.model.fetch();
}
```

 例: 次に、完全な拡張を示します。

```
app.views.SearchPage = Backbone.View.extend({
 template: _.template($("#search-page").html()),
 events: {
 "keyup .search-key": "search"
 },
 initialize: function() {
 this.listView = new app.views.UserListView({model: this.model});
 },
 render: function(eventName) {
 $(this.el).html(this.template());
 $(".search-key", this.el).val(this.model.getCriteria());
 }
});
```



```

 this.listView.setElement($("#ul", this.el)).render();
 return this;
 },
 search: function(event) {
 this.model.setCriteria($(".search-key", this.el).val());
 this.model.fetch();
 }
});

```

## 検索結果リストビューの追加

検索結果リストのビューでは、テンプレートは必要ありません。これは、単純にリスト項目ビューのコンテナです。listItemViews メンバーのこれらのビューを追跡します。基盤となるコレクションが変更されると、それ自体を再表示します。

1. SearchPage ビューを含む `<script>` ブロックで、Backbone.View を拡張して検索ビューの結果のリストを表示します。リスト項目ビューの配列と initialize() 関数を追加します。

```

app.views.UserListView = Backbone.View.extend({
 listItemViews: [],
 initialize: function() {
 this.model.bind("reset", this.render, this);
 },

```

この手順の残りの部分で、すべてのコードを extend({}) ブロックに追加します。

2. render() 関数を作成します。この関数は、既存の各リスト項目ビューで close() をコールしてこれらをクリーンアップします。

```

render: function(eventName) {
 _.each(this.listItemViews,
 function(itemView) { itemView.close(); });

```

3. render() 関数で、基盤となるコレクションのレコードに対応する一連のリスト項目ビューを作成します。これらの各ビューは、リストのエントリにすぎません。後で app.views.UserListItemView を定義します。

```

this.listItemViews = _.map(this.model.models, function(model) { return new
 app.views.UserListItemView({model: model}); });

```

4. render() 関数で、各リスト項目ビューをルート DOM 要素に追加し、表示される UserListView オブジェクトを返します。

```

$(this.el).append(_.map(this.listItemViews, function(itemView) {
 return itemView.render().el; }));
return this;
}

```

 例: 次に、完全な拡張を示します。

```
app.views.UserListView = Backbone.View.extend({
 listItemViews: [],

 initialize: function() {
 this.model.bind("reset", this.render, this);
 },
 render: function(eventName) {
 _.each(this.listItemViews, function(itemView) {
 itemView.close(); });
 this.listItemViews = _.map(this.model.models,
 function(model) {
 return new app.views.UserListItemView(
 {model: model}); });
 $(this.el).append(_.map(this.listItemViews,
 function(itemView) {
 return itemView.render().el;
 }));
 return this;
 }
});
```

## 検索結果リスト項目ビューの追加

検索結果リスト項目ビューを定義するには、リストの1行のビューを設計して実装します。各リスト項目には、次のユーザー項目が表示されます。

- SmallPhotoUrl
- FirstName
- LastName
- Title

1. UserListView ビュー定義の直後に、検索結果リスト項目のビューを作成します。再度、Backbone.View を拡張し、tagName メンバーを定義して、このビューがリスト項目であることを指定します。この手順の残りの部分で、すべてのコードを extend({}) ブロックに追加します。

```
app.views.UserListItemView = Backbone.View.extend({
});
```

2. <li> タグを追加します。

```
app.views.UserListItemView = Backbone.View.extend({
 tagName: "li",
});
```

3. user-list-item スクリプトの未加工のコンテンツを使用して \_.template() をコールし、テンプレートを読み込みます。

```
template: _.template($("#user-list-item").html()),
```

4. `render()` 関数を追加します。underscore.js の `template()` 関数は、JSON データを取得して、関連付けられているテンプレートから作成された HTML を返します。この例では、関数は JSON から顧客のデータを抽出し、`user-list-item` テンプレートに適合する HTML を返します。HTML への変換時に、`template()` 関数は、テンプレートの自由変数を JSON データの対応するプロパティに置き換えます。

```
render: function(eventName) {
 $(this.el).html(this.template(this.model.toJSON()));
 return this;
},
```

5. リストビューからコールされる `close()` メソッドを追加し、必要なクリーンアップを行ってメモリリークを防止します。

```
close: function() {
 this.remove();
 this.off();
}
```



**例:** 次に、完全な拡張を示します。

```
app.views.UserListItemView = Backbone.View.extend({
 tagName: "li",
 template: _.template($("#user-list-item").html()),
 render: function(eventName) {
 $(this.el).html(this.template(this.model.toJSON()));
 return this;
 },
 close: function() {
 this.remove();
 this.off();
 }
});
```

## ユーザービューの追加

最後に、選択した顧客の詳細を表示する単純なページビューを追加します。このビューは、このアプリケーションの2番目のページです。顧客は、ユーザーリストビューの項目をタップしてこのビューに移動します。`user-page` テンプレートでは、検索リストに戻る [戻る] ボタンが定義されています。

1. `UserListItemView` ビュー定義の直後に、顧客の詳細のビューを作成します。`Backbone.View` を再度拡張します。この手順の残りの部分で、すべてのコードを `extend({})` ブロックに追加します。

```
app.views.UserPage = Backbone.View.extend({
});
```

2. インスタンス化するテンプレートを指定します。

```
app.views.UserPage = Backbone.View.extend({
 template: _.template($("#user-page").html()),
});
```

3. `render()` 関数を実装します。この関数は、モデルを再度読み取り、JSONに変換してからHTMLに変換します。

```
app.views.UserPage = Backbone.View.extend({
 template: _.template($("#user-page").html()),

 render: function(eventName) {
 $(this.el).html(this.template(this.model.toJSON()));
 return this;
 }
});
```

 **例:** 次に、完全な拡張を示します。

```
app.views.UserPage = Backbone.View.extend({
 template: _.template($("#user-page").html()),
 render: function(eventName) {
 $(this.el).html(this.template(this.model.toJSON()));
 return this;
 }
});
```

## ルーターの定義

Backbone ルーターは、ビュー間の移動パスを定義します。ルーターについての詳細は、「[ルーターとは?](#)」を参照してください。

1. 最後の `<script>` ブロックで、`Backbone.StackRouter` を拡張してアプリケーションルーターを定義します。

```
app.Router = Backbone.StackRouter.extend({

});
```

この手順の残りの部分で、すべてのコードを `extend({})` ブロックに追加します。

2. アプリケーションでは検索リストページおよびユーザーページがサポートされているため、各ページのルートは `routes` オブジェクト内に追加します。また、メインコンテンツページ("")のルートも追加します。

```
routes: {
 "": "list",
 "list": "list",
 "users/:id": "viewUser"
},
```

3. 検索結果のコレクション、検索ページビュー、およびユーザーページビューを作成する `initialize()` 関数を定義します。

```
initialize: function() {
 Backbone.Router.prototype.initialize.call(this);

 // Collection behind search screen
 app.searchResults = new app.models.UserCollection();
```

```

 app.searchPage = new app.views.SearchPage(
 {model: app.searchResults});
 app.userPage = new app.views.UserPage();
},

```

4. このルートの子項目のみを処理する `list()` 関数を定義します。検索結果ページをすぐに表示するために `slidePage()` をコールします。データを受信すると、リストが再作成されます。

```

list: function() {
 app.searchResults.fetch();
 this.slidePage(app.searchPage);
},

```

5. 特定のユーザーの詳細を取得および表示する `viewUser()` 関数を定義します。

```

viewUser: function(id) {
 var that = this;
 var user = new app.models.User({Id: id});
 user.fetch({
 success: function() {
 app.userPage.model = user;
 that.slidePage(app.userPage);
 }
 });
}

```

6. ファイルを保存した後に `cordova prepare` コマンドを実行します。  
7. アプリケーションを実行します。



**例:** これで完了です。次に、アプリケーション全体を示します。

```

<!DOCTYPE html>
<html>
<head>
<title>Users</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0,
 user-scalable=no;" />
<link rel="stylesheet" href="css/styles.css"/>
<link rel="stylesheet" href="css/ratchet.css"/>
</head>

<body>

<div id="content"></div>
<script src="js/jquery.min.js"></script>
<script src="js/underscore-min.js"></script>
<script src="js/backbone-min.js"></script>

<!-- Local Testing -->
<script src="js/MockCordova.js"></script>
<script src="js/cordova.force.js"></script>
<script src="js/MockSmartStore.js"></script>
<!-- End Local Testing -->

```

```

<!-- Container -->
<script src="cordova.js"></script>
<!-- End Container -->

<script src="js/force.js"></script>
<script src="js/force+promise.js"></script>
<script src="js/mobilesync.js"></script>
<script src="js/fastclick.js"></script>
<script src="js/stackrouter.js"></script>
<script src="js/auth.js"></script>

<!-- ----Search page template ---- -->
<script id="search-page" type="text/template">
 <header class="bar-title">
 <h1 class="title">Users</h1>
 </header>

 <div class="bar-standard bar-header-secondary">
 <input type="search"
 class="search-key"
 placeholder="Search"/>
 </div>

 <div class="content">
 <ul class="list">
 </div>
</script>

<!-- ---- User list item template ---- -->
<script id="user-list-item" type="text/template">

 <a href="#users/<%= Id %>" class="pad-right">

 <div class="details-short">
 <%= FirstName %> <%= LastName %>

 Title<%= Title %>
 </div>

</script>

<!-- ---- User page template ---- -->
<script id="user-page" type="text/template">
 <header class="bar-title">
 Back
 <h1 class="title">User</h1>
 </header>

 <footer class="bar-footer">

 </footer>

 <div class="content">

```

```

<div class="content-padded">

 <div class="details">
 <%= FirstName %> <%= LastName %>

 <%= Id %>

 <% if (Title) { %><%= Title %>
<% } %>
 <% if (City) { %><%= City %>
<% } %>
 <% if (MobilePhone) { %>
 <a href="tel:<%= MobilePhone %>">
 <%= MobilePhone %>
<% } %>
 <% if (Email) { %>
 <a href="mailto:<%= Email %>">
 <%= Email %><% } %>
 </div>
 </div>
</div>
</script>

<script>
// ---- The Models ---- //
// The User Model
app.models.User = Force.SObject.extend({
 objectType: "User",
 fieldlist: ["Id", "FirstName", "LastName", "SmallPhotoUrl",
 "Title", "Email", "MobilePhone", "City"]
});

// The UserCollection Model
app.models.UserCollection = Force.SObjectCollection.extend({
 model: app.models.User,
 fieldlist: ["Id", "FirstName", "LastName", "SmallPhotoUrl",
 "Title"],

 getCriteria: function() {
 return this.key;
 },

 setCriteria: function(key) {
 this.key = key;
 this.config = {type:"soql",
 query:"SELECT "
 + this.fieldlist.join(",")
 + " FROM User"
 + " WHERE Name like '" + key + "%'"
 + " ORDER BY Name "
 + " LIMIT 25 "
 };
 }
});

// ----- The Views
----- //

```



```
app.views.SearchPage = Backbone.View.extend({

 template: _.template($("#search-page").html()),

 events: {
 "keyup .search-key": "search"
 },

 initialize: function() {
 this.listView =
 new app.views.UserListView(
 {model: this.model});
 },

 render: function(eventName) {
 $(this.el).html(this.template());
 $(".search-key", this.el).val(this.model.getCriteria());
 this.listView.setElement($("#ul", this.el)).render();
 return this;
 },

 search: function(event) {
 this.model.setCriteria($(".search-key", this.el).val());
 this.model.fetch();
 }
});

app.views.UserListView = Backbone.View.extend({

 listItemViews: [],

 initialize: function() {
 this.model.bind("reset", this.render, this);
 },

 render: function(eventName) {
 _.each(this.listItemViews,
 function(itemView) {itemView.close(); });
 this.listItemViews =
 _.map(this.model.models, function(model) {
 return new app.views.UserListItemView(
 {model: model}); });
 $(this.el).append(_.map(this.listItemViews,
 function(itemView) {
 return itemView.render().el; }));
 return this;
 }
});

app.views.UserListItemView = Backbone.View.extend({

 tagName: "li",
 template: _.template($("#user-list-item").html()),
```

```
render: function(eventName) {
 $(this.el).html(this.template(this.model.toJSON()));
 return this;
},

close: function() {
 this.remove();
 this.off();
}
});

app.views.UserPage = Backbone.View.extend({

 template: _.template($("#user-page").html()),

 render: function(eventName) {
 $(this.el).html(this.template(this.model.toJSON()));
 return this;
 }
});

// ----- The Application Router
// -----

app.Router = Backbone.StackRouter.extend({

 routes: {
 "": "list",
 "list": "list",
 "users/:id": "viewUser"
 },

 initialize: function() {
 Backbone.Router.prototype.initialize.call(this);

 // Collection behind search screen
 app.searchResults = new app.models.UserCollection();

 // We keep a single instance of SearchPage and UserPage
 app.searchPage = new app.views.SearchPage(
 {model: app.searchResults});
 app.userPage = new app.views.UserPage();
 },

 list: function() {
 app.searchResults.fetch();
 // Show page right away
 // List will redraw when data comes in
 this.slidePage(app.searchPage);
 },
});
```

```
viewUser: function(id) {
 var that = this;
 var user = new app.models.User({Id: id});
 user.fetch({
 success: function() {
 app.userPage.model = user;
 that.slidePage(app.userPage);
 }
 });
}
});
</script>
</body>
</html>
```

## Mobile Sync サンプルアプリケーション

Salesforce Mobile SDK には、ハイブリッドアプリケーションでの Mobile Sync の使用方法を示すサンプルアプリケーションが備えられています。Account Editor には、これらのサンプルの全機能が装備されています。いずれかの単純なサンプルに切り替えるには、`bootconfig.json` ファイルの `startPage` プロパティを変更します。

### iOS でのサンプルの実行

Salesforce Mobile SDK for iOS インストールディレクトリで、`SalesforceMobileSDK.xcworkspace` をダブルクリックして Xcode で開きます。Xcode プロジェクトナビゲーターで、Hybrid SDK/AccountEditor プロジェクトを選択し、[Run (実行)] をクリックします。

### Android でのサンプルの実行

Android Studio でサンプルを実行するには、最初に SalesforceMobileSDK-Android リポジトリのコピーから基本ライブラリへの参照を追加します。次の連動関係をサンプルモジュールに追加し、それぞれの [Scope (範囲)] を「Compile」に設定します。

- `libs/SalesforceSDK`
- `libs/SmartStore`
- `hybrid/SampleApps/AccountEditor`

Android Studio での作成が完了したら、ツールバーまたはメニューで [Run '<sample\_name>'] (「<sample\_name>」を実行)] をクリックします。

### Account Editor のサンプル

Account Editor は、Mobile SDK 2.0 の最も複雑な Mobile Sync ベースのサンプルアプリケーションです。取引先をオンラインまたはオフラインで作成/編集/削除したり、競合の検出を示したりすることができます。

サンプルを実行する手順は、次のとおりです。

1. `external/shared/sampleApps/mobilesync/bootconfig.json` に変更を加えた場合は、元のコンテンツに戻します。

## 2. Account Editor を起動します。

このアプリケーションには、3つの画面が含まれています。

- 取引先の検索
- 取引先の詳細
- 同期

アプリケーションを初めて起動すると、最後に使用した取引先をリストする取引先検索画面が表示されます。この画面では、次の操作を実行できます。

- 検索文字列を入力して、指定した文字列が名前に含まれている取引先を検索する。
- 取引先をタップして、取引先詳細画面を起動する。
- [作成]をタップして、空の取引先詳細画面を起動する。
- [オンライン]をタップして、オフラインにする。すでにオフラインになっている場合は、[オフライン]ボタンをタップして、オンラインに戻すことができます(デバイスを機内モードにして、オフラインにすることもできます)。

取引先詳細画面を起動するには、取引先検索画面で取引先レコードをタップします。選択した取引先の項目が、詳細画面に表示されます。この画面では、次の操作を実行できます。

- 項目をタップして、値を変更する。
- [保存]をタップして、取引先を更新または作成する。入力規則エラーが発生すると、問題のある項目が強調表示されます。

保存時にオンラインになっており、最後の取得以降にサーバーのレコードが変更された場合は、リモートで変更された項目について警告が表示されます。

[マージ]および[上書き]という他の2つのボタンを使用して、アプリケーションで変更をどのように保存するかを制御できます。[上書き]をタップすると、画面に現在表示されているすべての値がサーバーに保存されます。[マージ]をタップすると、ユーザーが変更した項目のみがサーバーに保存され、ユーザーが変更しない項目についてはサーバーで変更が継続されます。

- [削除]をタップして、取引先を削除する。
- [オンライン]をタップしてオフラインにするか、[オフライン]をタップしてオンラインにする。

同期画面を表示するには、[オンライン]をタップしてオフラインにしてから、取引先を作成、更新、または削除します。[オフライン]を再度タップしてオンラインに戻すと、デバイスで変更したすべての取引先が同期画面に表示されます。

ローカルの変更をサーバーに保存するには、[n 件のレコードを処理]をタップします。取引先の保存に失敗すると、同期に失敗したことが表記されて取引先がリストに残ります。リスト内の任意の取引先をタップしてさらに編集したり、ローカルで削除した取引先を復元したりできます。

## 内部で実行される処理の確認

このサンプルのソースコードを表示するには、HTML またはテキストエディターで `AccountEditor.html` を開きます。

ファイルの主要なセクションは、次のとおりです。

- スクリプトの内容

- テンプレート
- モデル
- ビュー
- ルーター

## スクリプトの内容

このサンプルには、Mobile Sync アプリケーション用のライブラリの標準リストが含まれています。

- jQuery — <http://jquery.com/> を参照してください。
- Underscore — Backbone に必要な JavaScript 用のユーティリティ関連のライブラリです。 <http://underscorejs.org/> を参照してください。
- Backbone — Web アプリケーションに構造を提供します。 Mobile Sync で使用されます。 <http://backbonejs.org/> を参照してください。
- cordova.js — Salesforce Mobile SDK を使用するハイブリッドアプリケーションに必要です。
- force.js — REST API コールを行うための Salesforce Platform JavaScript ライブラリです。 Mobile Sync で必要です。
- mobilesync.js — Mobile Sync。
- fastclick.js — 物理タップしてからクリックイベントが発生するまでの 300 ミリ秒の遅延を回避するために使用されるライブラリです。 <https://github.com/ftlabs/fastclick> を参照してください。
- stackrouter.js および auth.js — 3 つのすべてのサンプルアプリケーションで使用されるヘルパー JavaScript ライブラリです。

## テンプレート

このアプリケーションのテンプレートは、次のとおりです。

- search-page
- sync-page
- account-list-item
- edit-account-page (取引先詳細ページ用)

## モデル

このアプリケーションでは、AccountCollection、Account および OfflineTracker という 3 つのモデルが定義されます。

AccountCollection は Mobile Sync の Force.SObjectCollection クラスのサブクラスであり、Force.SObjectCollection は Backbone フレームワークの Collection クラスのサブクラスです。

AccountCollection.config() メソッドは、コレクションに対する適切なクエリを返します。クエリモードには、次のいずれかを指定できます。

- オンラインでありクエリ条件を指定していない場合は、最後に使用 (MRU)
- オンラインでありクエリ条件を指定した場合は SOQL
- オフラインの場合は SmartSQL

コレクションでアプリケーションから `fetch()` をコールすると、`config()` から返されたクエリが `fetch()` 関数で実行されます。このクエリ結果を使用して、オフラインキャッシュまたはサーバーのいずれかの `Account` オブジェクトで `AccountCollection` が入力されます。

`AccountCollection` では、`AccountEditor` アプリケーションで設定された 2 つのグローバルキャッシュを使用します。`app.cache` はオフラインストレージ用で、`app.cacheForOriginals` は競合検出用です。コードでは、`AccountCollection` モデルが次のように指定されていることがわかります。

- `app.models.Account` モデルのオブジェクトを含む (`model` 項目)
- クエリ対象の項目のリストを指定する (`fieldlist` 項目)
- サンプルアプリケーションのグローバルオフラインキャッシュを使用する (`cache` 項目)
- サンプルアプリケーションのグローバル競合検出キャッシュを使用する (`cacheForOriginals` 項目)
- オンラインクエリおよびオフラインクエリを処理する `config()` 関数を定義する

次にコードを示します (読みやすくするため簡略化しています)。

```
app.models.AccountCollection = Force.SObjectCollection.extend({
 model: app.models.Account,
 fieldlist: ["Id", "Name", "Industry", "Phone", "Owner.Name",
 "LastModifiedBy.Name", "LastModifiedDate"],
 cache: function() { return app.cache; },
 cacheForOriginals: function() {
 return app.cacheForOriginals; },

 config: function() {
 // Offline: do a cache query
 if (!app.offlineTracker.get("isOnline")) {
 // ...
 }
 // Online
 else {
 // ...
 }
 }
});
```

`Account` は Mobile Sync の `Force.SObject` クラスのサブクラスであり、`Force.SObjectCollection` は Backbone フレームワークの `Model` クラスのサブクラスです。`Account` モデルのコードは、次のように指定していることを示します。

- `subjectType` 項目を使用して、モデルが表す `sObject` の種別を示す (この場合は `Account`)。
- サーバーから取得する項目がサーバーに送信する項目と同じではないため、`fieldlist` を項目としてではなくメソッドとして定義する。
- サンプルアプリケーションのグローバルオフラインキャッシュを使用する (`cache` 項目)。
- サンプルアプリケーションのグローバル競合検出キャッシュを使用する (`cacheForOriginals` 項目)。
- 現在のオフライン状況に基づいたキャッシュ方法を示す値を返す `cacheMode()` メソッドをサポートする。

次にコードを示します。

```
app.models.Account = Force.SObject.extend({
 subjectType: "Account",
```

```

fieldlist: function(method) {
 return method == "read"
 ? ["Id", "Name", "Industry", "Phone", "Owner.Name",
 "LastModifiedBy.Name", "LastModifiedDate"]
 : ["Id", "Name", "Industry", "Phone"];
},
cache: function() { return app.cache;},
cacheForOriginals: function() { return app.cacheForOriginals;},
cacheMode: function(method) {
 if (!app.offlineTracker.get("isOnline")) {
 return Force.CACHE_MODE.CACHE_ONLY;
 }
 // Online
 else {
 return (method == "read" ?
 Force.CACHE_MODE.CACHE_FIRST :
 Force.CACHE_MODE.SERVER_FIRST);
 }
}
});

```

OfflineTracker は、Backbone の Model クラスのサブクラスです。このクラスは、ブラウザーのオフライン状況を監視することによって、アプリケーションのオフライン状況を追跡します。ブラウザーがオフラインであることが検出された場合は、アプリケーションが自動的にオフラインに切り替わります。ただし、オンラインになるのはユーザーが要求した場合のみです。

次にコードを示します。

```

app.models.OfflineTracker = Backbone.Model.extend({
 initialize: function() {
 var that = this;
 this.set("isOnline", navigator.onLine);
 document.addEventListener("offline", function() {
 console.log("Received OFFLINE event");
 that.set("isOnline", false);
 }, false);
 document.addEventListener("online", function() {
 console.log("Received ONLINE event");
 // User decides when to go back online
 }, false);
 }
});

```

## ビュー

このサンプルでは、5つのビューが定義されます。

- SearchPage
- AccountListView
- AccountListItemView
- EditAccountView
- SyncPage



ビューには通常、デザインテンプレート、`initialize()` 関数、および `render()` 関数を指定するテンプレート項目が含まれています。

各ビューでは、`events` 項目も定義できます。この項目には配列が含まれ、キー/値エントリがイベント種別およびイベントハンドラー関数名をそれぞれ指定します。エントリには次の形式が使用されます。

```
"<event-type>[<control>]": "<event-handler-function-name>"
```

次に例を示します。

```
events: {
 "click .button-prev": "goBack",
 "change": "change",
 "click .save": "save",
 "click .merge": "saveMerge",
 "click .overwrite": "saveOverwrite",
 "click .toggleDelete": "toggleDelete"
},
```

### SearchPage

検索画面全体のビュー。モデルとして `AccountCollection` が指定されます。検索入力項目に変更があるかどうか (`keyup` イベント) を監視し、`search()` 関数でモデルを適宜更新します。

```
events: {
 "keyup .search-key": "search"
},
search: function(event) {
 this.model.setCriteria($(".search-key", this.el).val());
 this.model.fetch();
}
```

### AccountListView

検索画面のリスト部分のビュー。モデルとして `AccountCollection` が指定され、`AccountCollection` オブジェクトの取引先ごとに `AccountListItemView` オブジェクトが作成されます。

### AccountListItemView

リスト内の項目のビュー。

### EditAccountPage

取引先詳細ページのビュー。このビューでは、いくつかのイベントを監視します。

| イベント種別 | 対象コントロール                | ハンドラー関数名      |
|--------|-------------------------|---------------|
| クリック   | button-prev             | goBack        |
| 変更     | set 以外 (任意の編集コントロールが有効) | change        |
| クリック   | save                    | save          |
| クリック   | merge                   | saveMerge     |
| クリック   | overwrite               | saveOverwrite |
| クリック   | toggleDelete            | toggleDelete  |

いくつかのイベントハンドラー関数には特に注意が必要です。change() 関数は、ユーザー編集をモデルに戻すためにビューでイベント対象がどのように使用されるかを表示します。

```
change: function(evt) {
 // apply change to model
 var target = event.target;
 this.model.set(target.name, target.value);
 $("#account" + target.name + "Error", this.el).hide();
}
```

toggleDelete() 関数は、ユーザーが取引先を削除または復元できる切り替えを処理します。ユーザーが復元をクリックした場合は、内部の \_\_locally\_deleted\_\_ flag が false に設定され、レコードがキャッシュ内に復元されていることを示します。それ以外の場合は、ローカルモデルを破棄することによって、サーバーでのレコードの削除が試みられます。

```
toggleDelete: function() {
 if (this.model.get("__locally_deleted__")) {
 this.model.set("__locally_deleted__", false);
 this.model.save(null, this.getSaveOptions(
 null, Force.CACHE_MODE.CACHE_ONLY));
 }
 else {
 this.model.destroy({
 success: function(data) {
 app.router.navigate("#", {trigger:true});
 },
 error: function(data, err, options) {
 var error = new Force.Error(err);
 alert("Failed to delete account:
 " + (error.type === "RestError" ?
 error.details[0].message :
 "Remote change detected - delete aborted"));
 }
 });
 }
}
```

### SyncPage

同期ページのビュー。このビューでは、いくつかのイベントを監視します。

| イベント種別 | コントロール      | ハンドラー関数名 |
|--------|-------------|----------|
| クリック   | button-prev | goBack   |
| クリック   | sync        | sync     |

画面がどのように表示されるかを確認するには、render メソッドを調べます。

```
render: function(eventName) {
 $(this.el).html(this.template(_.extend(
 {countLocallyModified: this.model.length},
 this.model.toJSON())));
}
```

```

 this.listView.setElement($("ul", this.el)).render();

 return this;
 },

```

ユーザーが[実行](同期コントロール)をタップしたときの動作を見てみましょう。

`sync()` 関数は、まずローカルで変更された取引先をビューのコレクションで確認し、それをサーバーに保存しようとします。保存に成功し、ローカルで変更された他のレコードがない場合は、アプリケーションが検索画面に戻ります。それ以外の場合は、取引先がローカルの保存に失敗したとマークされ、`sync()` を再度コールします。

```

sync: function(event) {
 var that = this;
 if (this.model.length == 0 ||
 this.model.at(0).get("__sync_failed__")) {
 // We push sync failures back to the end of the list.
 // If we encounter one, it means we are done.
 return;
 }
 else {
 var record = this.model.shift();

 var options = {
 mergeMode: Force.MERGE_MODE.MERGE_FAIL_IF_CHANGED,
 success: function() {
 if (that.model.length == 0) {
 app.router.navigate("#", {trigger:true});
 }
 else {
 that.sync();
 }
 },
 error: function() {
 record = record.set("__sync_failed__", true);
 that.model.push(record);
 that.sync();
 }
 };
 return record.get("__locally_deleted__")
 ? record.destroy(options) :
 record.save(null, options);
 }
});

```

## ルーター

ルーターが初期化されると、サンプル全体で使用される2つグローバルキャッシュが設定されます。

```

setupCaches: function() {
 // Cache for offline support
 app.cache = new Force.StoreCache("accounts",

```

```

 [{path:"Name", type:"string"}]});

 // Cache for conflict detection
 app.cacheForOriginals = new Force.StoreCache("original-accounts");

 return $.when(app.cache.init(), app.cacheForOriginals.init());
 },

```

グローバルキャッシュが設定されると、2つの AccountCollection オブジェクトも設定されます。1つは検索画面用、もう1つは同期画面用です。

```

// Collection behind search screen
app.searchResults = new app.models.AccountCollection();

// Collection behind sync screen
app.localAccounts = new app.models.AccountCollection();
app.localAccounts.config = {
 type:"cache",
 cacheQuery: {
 queryType:"exact",
 indexPath:"__local__",
 matchKey:true,
 order:"ascending",
 pageSize:25}};

```

最後に、Search、Sync、および EditAccount の各画面用にビューオブジェクトが作成されます。

```

// We keep a single instance of SearchPage / SyncPage and EditAccountPage
app.searchPage = new app.views.SearchPage({model: app.searchResults});
app.syncPage = new app.views.SyncPage({model: app.localAccounts});
app.editPage = new app.views.EditAccountPage();

```

ルーターには、アクションをルータークラスのメソッドに対応付ける routes 項目があります。

```

routes: {
 "": "list",
 "list": "list",
 "add": "addAccount",
 "edit/accounts/:id": "editAccount",
 "sync":"sync"
},

```

list アクションは、fetch() をコールして検索結果のコレクションを取得し、検索ページをビューに表示します。

```

list: function() {
 app.searchResults.fetch();
 // Show page right away - list will redraw when data comes in
 this.slidePage(app.searchPage);
},

```

addAccount アクションは、空の取引先オブジェクトを作成し、その取引先の編集ページをビューに表示します。

```

addAccount: function() {
 app.editPage.model = new app.models.Account({Id: null});
}

```

```

 this.slidePage(app.editPage);
 },

```

`editAccount` アクションは、指定された `Account` オブジェクトを取得し、取引先詳細ページをビューに表示します。

```

editAccount: function(id) {
 var that = this;
 var account = new app.models.Account({Id: id});
 account.fetch({
 success: function(data) {
 app.editPage.model = account;
 that.slidePage(app.editPage);
 },
 error: function() {
 alert("Failed to get record for edit");
 }
 });
}

```

`sync` アクションは、`fetch` をコールして `localAccounts` コレクションを計算し、同期ページをビューに表示します。

```

sync: function() {
 app.localAccounts.fetch();
 // Show page right away - list will redraw when data comes in
 this.slidePage(app.syncPage);
}

```

## 設定ファイルの検証

形式的に構造化されたテキストファイルを作成するときは、どの段階でもスキーマ検証が役立ちます。SmartStore および MobileSync の設定ファイルの場合、スキーマ検証は、関連レコードを処理する設定が複雑であるときに特に役立ちます。

Mobile SDK 8.0 以降、設定ファイルの検証に次の Mobile SDK ユーティリティが使用できるようになりました。

- forceios
- forcedroid
- forcereact
- Salesforce CLI Mobile SDK プラグイン

設定を検証するには、`checkconfig` アクションを使用します。

## Mobile SDK ユーティリティによる検証

Mobile SDK npm ユーティリティの `checkconfig` に対するコールを次に示します。

```
$ forceios checkconfig
```

これによりユーザーは、次の情報を入力するように促されます。

- `configpath` — 設定ファイルへのパス
- `configtype` — 設定ファイルの種類。「store」または「syncs」のいずれかを指定する必要があります。

## Salesforce CLI コマンドラインによる検証

Salesforce CLI の `checkconfig` に対するコールを次に示します。

```
$ sf mobilesdk:ios:checkconfig
```

`ios` は、`android`、`hybrid`、または `reactnative` に置き換えることができますが、いずれの場合も、結果は同じです。次の必須オプションを指定します。

- `-c`, `--configpath=<設定ファイルへのパス>`
- `-y`, `--configtype=<「store」または「syncs」のいずれか>`

## 他の検証の使用

必要な場合は、Mobile SDK スキーマとサードパーティの JSON 検証を併用できます。公開されているスキーマ定義は、次で確認できます。

- 保存設定: `store.schema.json`
- 同期設定: `syncs.schema.json`

## スキーマの変更履歴

### Mobile SDK 9.1 の更新

種別 `soql` のターゲットで、省略可能な `maxBatchSize` プロパティを使用できるようになりました。このプロパティには、200 ~ 2,000 の間の任意の整数を指定できます。デフォルト値は 2,000 です。

### Mobile SDK 8.0 の更新

iOS と Android の同期設定の構文の違いを解決するため、iOS スキーマは次のように修正され、Android の構文と同じになりました。

- 親子の下位同期設定の種別が `parent_children` になりました。以前の iOS では、`parentChildren` が使用されていました。
- 親子の上位同期設定の親の項目の作成リストに `createFieldlist` のラベルが付けられます。以前の iOS では、`parentCreateFieldlist` が使用されていました。
- 親子の上位同期設定の親の項目の更新リストに `updateFieldlist` のラベルが付けられます。以前の iOS では、`parentUpdateFieldlist` が使用されていました。



例: サンプルの設定ファイルを次に示します。

- `userstore.json`
- `usersyncs.json`

## 第 12 章      ファイルとネットワーキング

トピック:

- [アーキテクチャ](#)
- [ファイルのダウンロードと共有の管理](#)
- [ファイルのアップロード](#)
- [暗号化とキャッシュ](#)
- [Android アプリケーションでのファイルの使用](#)
- [iOS ネイティブアプリケーションでのファイルの使用](#)
- [ハイブリッドアプリケーションでのファイルの使用](#)

Mobile SDK には、2 つのレベルのテクノロジーを実装するファイル管理の API が用意されています。Mobile SDK では、Connect REST API を介してファイル要求を処理する便利なファイル管理用のメソッドが提供されています。ネットワーキングクラスにより、アプリケーションを使用して REST API レベルで待機中の REST 要求を制御できます。このような表裏一体のテクノロジーを合わせることで、堅牢なコンテンツ管理機能と高度なネットワーキングパフォーマンスを SDK で実現できます。



## アーキテクチャ

Mobile SDK 4.2 以降では、Android REST 要求システムは、基盤となるアーキテクチャとして Square Open Source のオープンソース外部ライブラリの OkHttp (v3.2.0) を使用します。過去のリリースの Google Volley ライブラリは、このライブラリに置き換えられます。そのため、Mobile SDK では、`WrappedRestRequest` クラスが定義されなくなりました。詳細は、[square.github.io/okhttp/](https://square.github.io/okhttp/) を参照してください。

iOS では、`SalesforceNetwork` ライブラリを使用して、ファイル管理およびネットワーキングを行います。ファイルや他の REST 要求の REST API コールは、すべてこのライブラリを経由します。

 **メモ:** 古いバージョンのアプリケーションでサードパーティネットワーキングライブラリに直接アクセスした場合、`SalesforceNetwork` ライブラリを使用するようにそのコードを更新する必要があります。

ハイブリッド JavaScript 関数では、デバイスのオペレーティングシステム (Android、iOS、または Windows) に対応する Mobile SDK のアーキテクチャを使用して、ファイル操作を実装します。これらの関数は、`force.js` で定義されています。

## ファイルのダウンロードと共有の管理

Salesforce Mobile SDK には、ファイルのダウンロード操作や共有操作のための特殊な REST 要求を構築する便利なメソッドがあります。これらの要求を使用して、次のことを実行できます。

- ファイルのバイトストリームにアクセスする。
- ファイルのページをダウンロードする。
- ファイルのページをプレビューする。
- ファイルレコードの詳細を取得する。
- ファイル共有情報にアクセスする。
- ファイル共有を追加および削除する。

## 要求のページ

REST 要求の「ページ」という用語は、コンテキストに応じて結果セットの特定項目または項目グループを表します。たとえば、特定のファイルのページをプレビューする場合、要求は表示されたページの中から指定されたページを取得します。他の大部分の要求では、ページとは結果のリストのセクションを表します。1 ページのレコードまたはトピックの最大数は、デフォルトの 25 に設定されます。

応答には、`NextPageUrl` 項目が含まれます。この値が定義されている場合、結果には別のページがあります。アプリケーションで結果のページをスクロールする場合、この項目を使用して不要な要求が送信されないようにできます。また、応答状況を確認するだけで、リストの最後であるかどうかを検出することもできます。何も返されない場合やエラーが返される場合は、これ以上表示するものがないため、別の要求を発行する必要はありません。

## ファイルのアップロード

ネイティブモバイルプラットフォームでは、ファイルをアップロードするためのメソッドがサポートされています。アップロードするローカルファイルへのパス、ファイルの名前またはタイトル、および説明を入力します。MIMEタイプがわかっている場合、そのタイプも指定できます。アップロードメソッドは、ファイルをサーバーにアップロードできるプラットフォーム固有の要求オブジェクトを返します。この要求をサーバーに送信すると、バージョンが1に設定されたファイルがサーバーで作成されます。

次に、アプリケーション種別ごとに使用するメソッドを示します。

| アプリケーション 種別               | Upload メソッド                                                   | 署名                                                                                                                                                         |
|---------------------------|---------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Android ネイティブ             | <code>FileRequests.uploadFile()</code>                        | <pre>public static RestRequest uploadFile( File theFile, String name, String description, String mimeType) throws UnsupportedOperationException</pre>      |
| iOS ネイティブ                 | <pre>- requestForUploadFile: name:description:mimeType:</pre> | <pre>- (SFRestRequest *) requestForUploadFile:(NSData *)data name:(NSString *)name description:(NSString *)description mimeType:(NSString *)mimeType</pre> |
| ハイブリッド<br>(Android と iOS) | なし                                                            | なし                                                                                                                                                         |

## 暗号化とキャッシュ

Mobile SDK は、ファイルの暗号化されていないバイトストリームへのアクセスを提供しますが、ファイルのキャッシュまたは保存は実装しません。アプリケーションでファイルをデバイスに保存する必要がある場合、独自の解決策を自由に考案できます。

## Android アプリケーションでのファイルの使用

`FileRequests` クラスは、ファイル操作を実行する `RestRequest` オブジェクトを作成するための静的メソッドを提供します。各メソッドは、新しい `RestRequest` オブジェクトを返します。次に、アプリケーションは `ownedFilesList()` メソッドをコールして `RestRequest` オブジェクトを取得します。このオブジェクトは、`RestRequest` オブジェクトを使用してサーバーに要求を送信する関数にパラメーターとして渡されます。

```
performRequest(FileRequests.ownedFilesList(null, null));
```

この例では、`null` を最初のパラメーター (`userId`) に渡します。この値で、コンテキスト (ログインしている) ユーザーの ID を使用するように `ownedFilesList()` メソッドに指示しています。`pageNum` パラメーターの 2 番目の `null` で、結果の最初のページを取得するようにメソッドに指示します。

ネイティブ Android アプリケーションの場合、ファイル管理クラスおよびメソッドは `com.salesforce.androidsdk.rest.files` パッケージにあります。

関連トピック:

[Files API リファレンス](#)

## 要求キューの管理

`RestClient` クラスは、内部的に `OkHttpClient` クラスのインスタンスを使用して REST API 要求を管理します。基盤となる `OkHttp` オブジェクトに直接アクセスして、待機中の要求をキャンセルできます。特定の要求を管理するには、`RestClient.sendAsync()` Mobile SDK メソッドによって返される `OkHttp Call` オブジェクトを使用します。

 **例:** 次の例に、`OkHttpClient` を使用したいいくつかの一般的なネットワーク操作の実行方法を示します。

一般的なインポート

```
import okhttp3.Headers;
import okhttp3.HttpUrl;
import okhttp3.OkHttpClient;
import okhttp3.Call;
import okhttp3.Dispatcher;
import okhttp3.Request;
import okhttp3.RequestBody;
import okhttp3.Response;
```

現在の `OkHttp` クライアントハンドルを取得する

現在の `RestClient` インスタンスが使用している `OkHttpClient` のハンドルを取得するには、次のように指定します。

Kotlin

```
var okClient = restClient.getOkHttpClient()
```

Java

```
OkHttpClient okClient = restClient.getOkHttpClient();
```

`OkHttp` ディスパッチャーを取得する

Kotlin

```
var dispatcher = restClient.getOkHttpClient().dispatcher()
```

Java

```
Dispatcher dispatcher = restClient.getOkHttpClient().dispatcher();
```

待機中のすべてのコールをキャンセルする

**Kotlin**

```
var dispatcher = restClient.getOkHttpClient().dispatcher()
dispatcher.cancelAll()
```

**Java**

```
Dispatcher dispatcher = restClient.getOkHttpClient().dispatcher();
dispatcher.cancelAll();
```

OkHttp ハンドルを REST 要求に格納する

**Kotlin**

```
var call = restClient.sendAsync(restRequest, callback)
```

**Java**

```
Call call = restClient.sendAsync(restRequest, callback);
```

格納されたハンドルを使用して特定の REST 要求をキャンセルする

**Kotlin**

```
var call = restClient.sendAsync(restRequest, callback)
```

**Java**

```
Call call = restClient.sendAsync(restRequest, callback);
...
call.cancel();
```

詳細は、[square.github.io/okhttp/](https://square.github.io/okhttp/)を参照してください。

関連トピック:

OkHttp: 基盤となるネットワークライブラリ

## iOS ネイティブアプリケーションでのファイルの使用

ネイティブ iOS アプリケーションでファイルを処理するには、SFRestAPI (Files) カテゴリで定義されている便利なメソッドを使用します。これらのメソッドは、Android ネイティブアプリケーションとハイブリッドアプリケーションの Files API と似ています。これらは、要求を REST API の同じリストに送信しますが、異なる基盤を使用します。

たとえば、次のコードスニペットは `requestForOwnedFilesList:page:` メソッドをコールして `SFRestRequest` オブジェクトを取得します。次に、そのオブジェクトが所有するオブジェクトを、応答を受信する代理オブジェクトとして指定して、要求オブジェクトをサーバーに送信します。

```
SFRestRequest *request = [[SFRestAPI sharedInstance] requestForOwnedFilesList:nil page:0];
[[SFRestAPI sharedInstance] send:request delegate:self];
...
```

この例では、最初のパラメーター(`userId`)に `nil` を渡します。この値で、コンテキスト(ログインしている)ユーザーのIDを使用するように `requestForOwnedFilesList:page:` メソッドに指示します。`pageNum` パラメーターに `0` を渡すと、最初のページを取得するようメソッドに指示します。

 **メモ:** Swift バージョンの `SFRestAPI (Files)` メソッドは、Mobile SDK で明示的に定義されていません。Swift でこれらのメソッドをコーディングするには、Xcode コンパイラで提供されるオートコンプリート候補表示機能を使用します。これらのメソッドおよびパラメーター名の候補は、Swift コンパイラによって決まり、Objective-C とは異なる可能性があります。

## REST 応答とマルチスレッド

`SalesforceNetwork` ライブラリは常に、現在 `SFRestDelegate` が実行されているスレッドに REST 応答を送ります。この設計では、代理メソッドでどのようにサーバー応答を処理しても、目的の用途に対応できます。応答を受信したら、返されたデータを使用して任意の操作を実行できます。たとえば、キャッシュしたり、データベースに保存したり、すぐに UI コントロールに一括送信したりできます。ただし、応答を直接 UI に送信する場合代理メソッドでそのメッセージをメインスレッドに送る必要があります。

関連トピック:

[Files API リファレンス](#)

## 要求の管理

iOS 用の `SalesforceNetwork` ライブラリでは、`SFNetworkEngine` と `SFNetworkOperation` の2つの主オブジェクトを定義します。`SFRestRequest` は、内部的に `SFNetworkOperation` オブジェクトを使用して、各サーバーコールを実行します。

要求のために `SFNetworkOperation` オブジェクトにアクセスする方法は2つあります。

- 次のメソッドは `SFNetworkOperation*` を返します。
  - `[SFRestRequest send:]`
  - `[SFRestAPI send:delegate:]`
- `SFRestRequest` オブジェクトには、種別が `SFNetworkOperation*` の `networkOperation` オブジェクトが含まれます。

待機中の REST 要求をキャンセルする方法も2つあります。

- `SFRestRequest` は、要求をキャンセルする新しいメソッドを提供します。

```
- (void) cancel;
```

- SFRestAPI には、現在実行されているすべての要求をキャンセルするメソッドがあります。

```
- (void)cancelAllRequests;
```



#### 例: 要求をキャンセルする方法の例

すべての要求をキャンセルする

```
[[SFRestAPI sharedInstance] cancelAllRequests];
```

1つの要求をキャンセルする

```
SFRestRequest *request = [[SFRestAPI sharedInstance] requestForOwnedFilesList:nil
page:0];
[[SFRestAPI sharedInstance] send:request delegate:self];
...
// User taps Cancel Request button while waiting for the response
-(void) cancelRequest:(SFRestRequest *) request {
 [request cancel];
}
```

## ハイブリッドアプリケーションでのファイルの使用

ハイブリッドファイル要求ラッパーは、[force+files.js](#) JavaScript ライブラリにあります。ハイブリッド関数を使用する場合、サーバー応答を受信して処理するコールバック関数を渡します。また、エラーを処理する関数も渡します。

コードを簡素化するために、`mobilesync.js` および `force.js` ライブラリを使用して、HTML アプリケーションを構築できます。[github.com/forcedotcom/SalesforceMobileSDK-Shared](https://github.com/forcedotcom/SalesforceMobileSDK-Shared) リポジトリの [FileExplorer](#) サンプルアプリケーションはこの設定を示しています。



**メモ:** Mobile SDK では、ハイブリッドアプリケーションでのファイルのアップロードをサポートしていません。

## 第 13 章      プッシュ通知と Mobile SDK

トピック:

- [プッシュ通知について](#)
- [ハイブリッドアプリケーションでのプッシュ通知の使用](#)
- [Android でのプッシュ通知の使用](#)
- [iOS でのプッシュ通知の使用](#)

Salesforce から送信された通知は、モバイルユーザーが組織内の重要な動向を常に把握しておくのに役立ちます。Salesforce 通知サービスでは、コードを実装する前にモバイルプッシュ通知を設定およびテストできます。本番環境でモバイル通知を受信するには、Mobile SDK アプリケーションをモバイル OS プロバイダーに登録し、次に Salesforce 通知に登録します。Mobile SDK では、大部分の登録タスクが内部的に実装されているため、コーディング作業を最小限に抑えることができます。

Mobile SDK では、クライアントアプリケーションにコストがかからない、Salesforce 通知ビルダーのカスタム通知がサポートされています。Mobile SDK アプリケーションには、通知ビルダーのプッシュ通知を受信して表示するための余分なコーディングが必要ありません。ただし、アプリケーションにメッセージトレイを実装することを選択した場合、Mobile SDK には、必要な Salesforce API を呼び出すための便利なメソッドが用意されています。

通知ビルダーの通知は暗号化されて着信します。Mobile SDK は内部的にメッセージを復号化するため、iOS アプリケーションでは定型クラスの拡張を追加するだけで済みます。さらに、Android では追加のコーディングは不要です。Apex のプッシュ通知は暗号化されませんが、完全にサポートされています。



## プッシュ通知について

Salesforce 通知サービスを使用すると、Mobile SDK カスタムアプリケーションでプッシュ通知を開発してテストできます。Mobile SDK には、プッシュ通知サービスへのデバイス登録を実装できる API が用意されています。ただし、通知の受信と処理は、引き続き開発者が行う必要があります。

プッシュ通知を受信するための設定は、次のようにいくつかのレベルで行います。

- デバイステクノロジープロバイダーからの転送サービスの設定 (Apple for iOS、Google for Android)
- プッシュ通知を有効にするための Salesforce 接続アプリケーション定義の設定
- アプリケーションにプッシュ通知を送信するための次の 1 つ以上の方法による Salesforce 組織の設定
  - Apex トリガーの実装
  - 通知ビルダーのカスタム通知種別への登録と、プロセスビルダー、Flow Builder、または Connect REST API を介したそれらの通知のプッシュ
  - Connect REST API のプッシュ通知リソースのコール
- Mobile SDK アプリケーションでの登録をサポートするための軽微なコード変更
- iOS アプリケーションで復号化をサポートするクラス拡張の実装
- 実行時のモバイルデバイスの登録

Apple または Google のサービス設定、接続アプリケーションの設定、Apex または Connect REST API のコーディング、および Mobile SDK アプリケーションへの軽微な変更は、お客様自身で行います。Salesforce Mobile SDK は、ランタイム登録を透過的に処理します。

組織のモバイルプッシュ通知の設定と、独自のカスタム通知トレイの作成については、『[Mobile Notifications Implementation Guide \(モバイル通知実装ガイド\)](#)』を参照してください。

## ハイブリッドアプリケーションでのプッシュ通知の使用

ハイブリッドアプリケーションでプッシュ通知を使用するには、まず次のことを行います。

- OS プロバイダーのプッシュ通知に登録する。
- ターゲットデバイスプラットフォームのプッシュ通知をサポートするように接続アプリケーションを設定する。

Salesforce Mobile SDK では、ハイブリッドアプリケーション自体で通知を受信するように登録できます。登録したら、受信通知を処理する動作を定義します。

関連トピック:

[Android でのプッシュ通知の使用](#)

[iOS でのプッシュ通知の使用](#)

## コードの変更 (ハイブリッド)

1. (Android のみ) 対象プラットフォームが Android の場合は、次の操作を実行します。

- a. `androidPushNotificationClientId` のエントリを追加します。 `assets/www/bootconfig.json` の場合:

```
"androidPushNotificationClientId": "333333344444"
```

この値は、Android デバイスに対してプッシュ通知の送信が承認された Google プロジェクトのプロジェクト番号です。

2. `cordova.require("com.salesforce.plugin.oauth").getAuthCredentials()` のコールバックに、次のコードを追加します。

```
cordova.require("com.salesforce.util.push").registerPushNotificationHandler(
 function(message) {
 // add code to handle notifications
 },
 function(error) {
 // add code to handle errors
 }
);
```



例: このコードは、メッセージの処理方法の一例を示しています。サーバーは、`message["payload"]` のペイロードを配信します。

```
function(message) {
 var payload = message["payload"];
 if (message["foreground"]) {
 // Notification is received while the app is in
 // the foreground
 // Do something appropriate with payload
 }
 if (!message["foreground"]) {
 // Notification was received while the app was in
 // the background, and the notification was clicked,
 // bringing the app to the foreground
 // Do something appropriate with payload
 }
}
```

## Android でのプッシュ通知の使用

Salesforce では、Firebase Cloud Messaging (FCM) フレームワークを介してプッシュ通知が Android アプリケーションに送信されます。このフレームワークの概要については、「[Firebase Cloud Messaging](#)」を参照してください。

プッシュ通知をサポートする Android アプリケーションを開発する場合、次の重要な点に留意してください。

- 必要条件
  - Google アカウントにサインインする。
  - Firebase へのアクセス権を持っている。
- FCM プッシュサービスをテストするには、Android Market アプリケーションまたは Google Play サービスのいずれかがインストールされた Android 物理デバイスを使用することをお勧めします。プッシュ通知はエミュ

レーターでは信頼性が低く、「Android with Google Play Services」のエミュレータータイプでしか動作しません。

- 接続アプリケーションの詳細ビューにある [テスト通知の送信] リンクを使用して、デバイスを ping することなく「模擬実行」テストを行うこともできます。この機能は、通知ビルダーのプッシュ通知で使用することもできます。

まず、アプリケーションに Google API プロジェクトを作成します。プロジェクトでは、FCM for Android 機能が有効になっている必要があります。プロジェクトの設定手順については、「[Firebase Cloud Messaging](#)」を参照してください。

Google API プロジェクトの設定プロセスでは、アプリケーションのキーが作成されます。プロジェクトの設定が完了したら、FCM キーを接続アプリケーションの設定に追加します。

- 📌 **メモ:** プッシュ通知の登録は、OAuth ログインフローの最後に行われます。そのため、ユーザーが Salesforce 組織にログインするまで、プッシュ通知はアプリケーションで受信されません。

## FCM (Android) 用の接続アプリケーションの設定

プッシュ通知をサポートするように Salesforce 接続アプリケーションを設定する手順は、次のとおりです。

1. 既存の Firebase プロジェクトをアプリケーションに関連付ける場合は、既存のプロジェクトを開きます。そうでない場合は、[Google Firebase コンソール](#)でアプリケーションのプロジェクトを作成します。
2. Firebase プロジェクトで、[プロジェクトの概要] の横にある歯車アイコンをクリックし、[プロジェクトの設定] をクリックします。
3. Firebase プロジェクト ID を収集します。
  - a. [全般] タブをクリックします。
  - b. [プロジェクト ID] 項目の値を記録します。このプロジェクト ID は後のステップで必要になります。
4. Firebase サービスアカウントの管理 SDK 非公開鍵を生成します。
  - a. [サービスアカウント] タブを選択し、[新しい秘密鍵を生成] をクリックします。
  - b. 非公開鍵が含まれる JSON ファイルをダウンロードします。ダウンロードしたファイルは後で必要になるため、ダウンロードしたファイルの場所をメモしておきます。  
この非公開鍵のファイル名の形式は、  
`Pc-api-1234567890123456789-123-firebase-adminsdk-albcd-a1234bc5678.json` のようになります。
5. アプリケーションマネージャーで [モバイル接続アプリケーション](#) を作成します。
6. モバイル接続アプリケーションの [モバイルアプリケーション設定] セクションで、次の項目に入力します。
  - a. [アプリケーションプラットフォーム] 項目で、[Android] を選択します。
  - b. [プッシュメッセージング] を選択します。
  - c. [プラットフォーム] 項目で、[Android] を選択します。
  - d. [Firebase 管理 SDK 非公開鍵] 項目で、Firebase サービスアカウント用に生成した非公開鍵が含まれる JSON ファイルをアップロードします。
  - e. [プロジェクト ID] 項目に、Firebase プロジェクトから収集したプロジェクト ID を入力します。

 **メモ:** Spring '24 以降、[モバイルアプリケーション設定] セクションには、従来の Firebase Cloud Messaging API サーバーキーの項目が含まれます。ただし、従来の Cloud Messaging API ユーザーは、2024 年 6 月までに [新しい Firebase Cloud Messaging API \(HTTP v1\) に移行](#)する必要があります。新しい Firebase Cloud Messaging API に必要な Firebase 管理 SDK 非公開鍵とプロジェクト ID を送信することをお勧めします。

## 7. 変更内容を保存します。

# コードの変更 (Android)

プッシュ通知をサポートするように Mobile SDK アプリケーションを設定する手順は、次のとおりです。

### 1. androidPushNotificationClientId のエントリを追加します。

- res/values/bootconfig.xml は次のようになります (ネイティブアプリケーションの場合)。

```
<string name="androidPushNotificationClientId">33333344444</string>
```

- assets/www/bootconfig.json の場合 (ハイブリッドアプリケーション)。

```
"androidPushNotificationClientId": "33333344444"
```

この値は、Android デバイスに対してプッシュ通知の送信が承認された Google プロジェクトのプロジェクト番号です。

Mobile SDK はこの値を自動的に読み取り、それを使用して、Salesforce 接続アプリケーションに対してデバイスを登録します。この検証により、Salesforce から接続アプリケーションに通知を送信できます。また、ログアウト時にはプッシュ通知用のデバイスが Mobile SDK で自動的に登録解除されます。

### 2. PushNotificationInterface を実装するアプリケーションでクラスを作成します。

PushNotificationInterface は、プッシュ通知を処理するための Mobile SDK Android インターフェースです。PushNotificationInterface には、onPushMessageReceived(Bundle message) という 1 つのメソッドがあります。

```
public interface PushNotificationInterface {
 public void onPushMessageReceived(Bundle message);
}
```

このメソッドでは、プッシュ通知を表示または破棄するカスタム機能を実装します。

### 3. Application サブクラスの onCreate() メソッドで、PushNotificationInterface の実装を渡して SalesforceSDKManager.setPushNotificationReceiver() メソッドをコールします。 SalesforceSDKManager.initNative() コールの直後に、このメソッドをコールします。次に例を示します。

```
@Override
public void onCreate() {
 super.onCreate();
 SalesforceSDKManager.initNative(getApplicationContext(),
 new KeyImpl(), MainActivity.class);
 SalesforceSDKManager.getInstance().
 setPushNotificationReceiver(myPushNotificationInterface);
}
```

Android アプリケーションでは、Notification Builder のプッシュ通知の復号化が自動的に実行されます。Apex のプッシュ通知は暗号化されません。

## iOS でのプッシュ通知の使用

プッシュ通知をサポートする iOS アプリケーションを開発する場合、次の重要な点に留意してください。

- iOS Developer Program のメンバーである必要があります。
- Apple 転送サービスは、iOS 物理デバイスでのみテストできます。プッシュ通知は、iOS エミュレーターでは動作しません。
- 通知が Apple で受け入れられても、すべてのプッシュ通知が対象デバイスに到達するという保証はありません。
- Apple Push Notification Services 設定では、Mac OS X で提供されている OpenSSL コマンドラインユーティリティを使用する必要があります。

Salesforce 側で登録を完了する前に、Apple Push Notification Services に登録する必要があります。以下の手順では、必要な処理の概要を示します。完全な手順は、<http://www.raywenderlich.com/32960/> を参照してください。

## Apple Push Notification Services の設定

Apple Push Notification Services (APNS) に登録するには、次のものがが必要です。

### 証明書署名要求 (CSR) ファイル

Mac OS X の Keychain Access 機能を使用して、この要求を生成します。また、後で使用するために、OpenSSL を使用して、CSR 秘密鍵をファイルにエクスポートします。

### iOS Developer Program のアプリケーション ID

iOS Developer Member Center で、アプリケーションの ID を作成し、CSR ファイルを使用して証明書を生成します。次に、OpenSSL を使用して、この証明書と秘密鍵のファイルを組み合わせて .p12 ファイルを作成します。このファイルは、接続アプリケーションを後で設定するときに必要になります。

### iOS Provisioning Profile

iOS Developer Member Center から、iOS アプリケーション ID と開発者証明書を使用して新しいプロビジョニングプロファイルを作成します。次に、プロファイルに含めるデバイスを選択して、プロビジョニングプロファイルをダウンロードして作成します。このようにすると、プロファイルを Xcode に追加できます。Xcode の Organizer を使用して、テストデバイスにプロファイルをインストールします。

設定が完了したら、Xcode でアプリケーションに署名し、アプリケーションを作成します。アプリケーションで正しいプロビジョニングプロファイルが使用されていることを、作成ログで確認します。プロビジョニングプロファイルの内容を表示するには、ターミナルウィンドウでコマンド `security cms -D -i <your profile>.mobileprovision` を実行します。

## APNS (iOS) 用に接続アプリケーションを設定する

モバイルアプリケーションの iOS バージョンでは、トークン、転送証明書、またはその両方を使用できます。トークンが指定された場合、Salesforce では常にトークンが使用されます。

1. **モバイル接続アプリケーション**を作成します。

2. [アプリケーションプラットフォーム] で [iOS] を選択します。
3. [プッシュメッセージングの有効化] を選択します。
4. [プラットフォーム] で [Apple] を選択します。
5. 必要に応じて、プッシュ通知設定での問題を回避するために、アプリケーションバンドル ID を指定します。
6. プッシュ通知を有効にする環境を選択します。
7. Apple 開発者アカウントから [署名鍵] をアップロードします。
8. Apple 開発者アカウントから [鍵識別子] を入力します。
9. Apple 開発者アカウントから [チーム識別子] を入力します。

## コードの変更 (iOS)

iOS アプリケーションで通知を処理するには、提供されているテンプレートコードを使用して AppDelegate クラスに登録します。Swift アプリケーションでは、受信した通知ビルダーの通知を復号化するための拡張を追加する必要があります。

## 通知の受信の登録

Mobile SDK for iOS には、転送登録を処理するための `SFPushNotificationManager` クラスが備えられています。Objective-C でこれを使用するには、`<SalesforceSDKCore/SFPushNotificationManager>` をインポートします。Swift では、特別なインポートは不要です。

`SFPushNotificationManager` クラスは、ランタイム共有インスタンスとして使用できます。

### Swift

```
SFPushNotificationManager.sharedInstance()
```

### Objective-C

```
[SFPushNotificationManager sharedInstance]
```

このクラスは、次の登録メソッドを実装します。

### Swift

```
func registerForRemoteNotifications()

func application(_ application: UIApplication,
 didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data)

func registerSalesforceNotifications(completionBlock: (() → Void)?,
 fail: (() → Void)?)

func unregisterSalesforceNotifications(withCompletionBlock: UserAccount,
 completionBlock: (() → Void)?) // for internal use
```



## Objective-C

```

- (void)registerForRemoteNotifications;

- (void)didRegisterForRemoteNotificationsWithDeviceToken:
 (NSData*) deviceTokenData;

- (BOOL)registerSalesforceNotificationsWithCompletionBlock:(nullable
 void (^)(void))completionBlock failBlock:(nullable void (^)(void))failBlock;

- (BOOL)unregisterSalesforceNotificationsWithCompletionBlock:(SFUserAccount*)user
 completionBlock:(nullable void (^)(void))completionBlock; // for internal use

```

Mobile SDK は、ログアウト時に `unregisterSalesforceNotifications` をコールします。

 メモ:

- Salesforce では通知ビルダーの通知が暗号化されます。
- 完全なコンテンツのプッシュ通知をサポートするには、iOS アプリケーションに復号化クラス拡張機能を実装する必要があります。
- 復号化を実装していない Mobile SDK アプリケーションが暗号化された Salesforce 通知を受信した場合、通知のタイトルのみが表示されます。
- Salesforce では Apex のプッシュ通知が暗号化されません。

## 復号化拡張の実装 (Swift)

Mobile SDK 8.2 以降では、forceios [iOSNativeSwiftTemplate](#) アプリケーションは、iOS `UNUserNotificationCenter` オブジェクトを介して通知の承認を要求します。`iOSNativeSwiftTemplate` の特殊バージョンである [iOSNativeSwiftEncryptedNotificationTemplate](#) は、通知の復号化を処理するために `UNNotificationServiceExtension` iOS クラスを拡張します。この拡張クラスである [NotificationService](#) は、Mobile SDK アプリケーションを変更なしに使用できる定型の復号化コードを提供します。暗号化された通知をサポートするには、Mobile SDK 8.2 以降を使用している必要があり、アプリケーションにこの拡張が含まれている必要があります。

通知ビルダーの暗号化通知をサポートする Swift プロジェクトを作成するには、forceios で `iOSNativeSwiftEncryptedNotificationTemplate` テンプレートを使用します。既存の Mobile SDK プロジェクトを更新する場合でも、新しい forceios テンプレートプロジェクトで最初から開始するのが最も簡単です。Swift プロジェクトを手動で更新する場合は、「例:手動でのプッシュ登録の追加(Swift)」まで進んでください。

1. 次のコマンドで、node.js から最新の forceios バージョンをインストールします。

```
[sudo] npm install -g forceios
```

2. `forceios createWithTemplate` コマンドをコールします。

```
forceios createWithTemplate
```



3. 次のように、最初のプロンプトで、「`iOSNativeSwiftEncryptedNotificationTemplate`」と入力します。

```
forceios createWithTemplate
Enter URI of repo containing template application or a Mobile SDK template name:
 iOSNativeSwiftEncryptedNotificationTemplate
```

4. 後続のプロンプトで、会社とプロジェクトの情報を入力します。入力した情報が許可されると、forceios によって暗号化通知に対応するプロジェクトが作成されます。
5. 古い Mobile SDK プロジェクトを更新する場合は、古いプロジェクトから新しいプロジェクトにアプリケーション固有のリソースをコピーします。

### 例: 手動でのプッシュ登録の追加 (Swift)

この例では、Mobile SDK 8.2 以上で構築され、`iOSSwiftNativeTemplate` プロジェクトを基にした既存のアプリケーションが必要です。

AppDelegate でのプッシュ通知登録の設定

1. `application(_:didFinishLaunchingWithOptions:)` メソッドで、`registerForRemotePushNotifications` へのコールのコメント化を解除します。

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
 launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

 self.window = UIWindow(frame: UIScreen.main.bounds)
 self.initializeAppViewState()
 // If you wish to register for push notifications, uncomment the line below.
 // Note that if you want to receive push notifications from Salesforce,
 // you will also need to implement the
 // application(application, didRegisterForRemoteNotificationsWithDeviceToken)
 // method (below).
 //
 self.registerForRemotePushNotifications()
 ...
}
```

`registerForRemotePushNotifications` メソッドは、リモート通知を受信するために Apple へのアプリケーションの登録を試行します。登録に成功すると、Apple から AppDelegate クラスの `application(_:didRegisterForRemoteNotificationsWithDeviceToken:)` メソッドにデバイストークンが渡されます。

2. `application(_:didRegisterForRemoteNotificationsWithDeviceToken:)` メソッドで、`didRegisterForRemoteNotifications(withDeviceToken:)` へのコールのコメント化を解除します。

```
func application(_ application: UIApplication,
 didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
 // Uncomment the code below to register your device token with the
 // push notification manager
 //
 didRegisterForRemoteNotifications(deviceToken)
 ...
}
```

```

 }
 ...

```

3. Apple への登録が失敗した場合にデバッガーエラーをログ記録するには、`application(_:didFailToRegisterForRemoteNotificationsWithError:)` メソッドに次のコードを追加します。

```

func application(_ application: UIApplication,
 didFailToRegisterForRemoteNotificationsWithError error: Error) {

 // Respond to any push notification registration errors here.
 SalesforceLogger.d(type(of:AppDelegate.self),
 message:"Failed to get token, error: \(error)")
 }
}

```

### 復号化拡張の追加

1. プロジェクトナビゲーターで、AppDelegate を変更した対象を選択します。
2. [Project (プロジェクト)] 設定で、[Signing & Capabilities (署名と機能)] を選択します。
3. [+Capability(+機能)] をクリックして、「Push Notifications」 (プッシュ通知) を検索します。ダブルクリックして機能を追加します。
4. [developer.apple.com/documentation/Modifying\\_Content\\_in\\_Newly\\_Delivered\\_Notifications\\_\(新たに配信された通知でコンテンツを変更する\)](https://developer.apple.com/documentation/Modifying_Content_in_Newly_Delivered_Notifications_(新たに配信された通知でコンテンツを変更する)) の「Add a Service App Extension to Your Project (プロジェクトへのサービスアプリケーション拡張の追加)」の手順に従ってください。
5. 拡張を独自の設定でテストする予定がある場合は、スキームを有効にする指示を受け入れてください。必要ない場合は、[キャンセル] をクリックします。
6. [github.com/forcedotcom/SalesforceMobileSDK-Templates](https://github.com/forcedotcom/SalesforceMobileSDK-Templates) リポジトリをコピーまたはダウンロードします。
7. `iOSNativeSwiftEncryptedNotificationTemplate/NotificationServiceExtension` フォルダーで、新しい拡張の Swift ファイルのコードを `NotificationServiceExtension.swift` のコードに置き換えます。

## トピック:

- OAuth の用語
- OAuth2.0 認証フロー
- 接続アプリケーション
- OAuth 2.0 サイトおよび Salesforce サイトを使用したポータル認証
- Salesforce Mobile SDK アプリケーションでの MDM の使用
- 高度な認証の使用
- Google ログイン要件に対応するための Android シングルサインオンアプリケーションのアップグレード
- 外部サービスにアクセスするための OpenID トークンの使用
- iOS での安全な鍵の保管
- Android での安全な鍵の保管

モバイルデバイスで実行されているエンタープライズアプリケーションにとって、安全な認証は不可欠です。OAuth 2.0 は、ユーザーのデータへのアクセスに対する安全な認証を可能にする業界標準のプロトコルで、ユーザー名およびパスワードを渡す必要がありません。OAuth は、ソフトウェアアクセスのバレットキーと呼ばれることがよくあります。バレットキーを使用すると、自動車の特定の部分へのアクセスを制限できます。たとえば、駐車場係にバレットキーを預けても、トランクやダッシュボードの小物入れなどが開けられることはありません。

モバイルアプリケーションの開発者は、Salesforce OAuth2.0 の実装をすばやく簡単に組み込むことができます。実装では HTML ビューを使用してユーザー名とパスワードが収集され、それがサーバーに送信されます。サーバーからセッショントークンと永続更新トークンが返され、今後のやり取りのためにデバイスに保存されます。

Salesforce 接続アプリケーションは、モバイルアプリケーションを Salesforce に接続するための主要な手段です。接続アプリケーションにより、開発者とシステム管理者は、アプリケーションの接続方法およびアクセス権を付与するユーザーを制御できます。たとえば、接続アプリケーションでアクセスを一連のユーザーに制限したり、IP 範囲の設定または緩和などを行うことができます。

## OAuth の用語

---

### アクセストークン

ユーザーの Salesforce ログイン情報を使用する代わりに、保護されたリソースへのユーザーのアクセスを許可するためにコンシューマーにより使用される値。アクセストークンはセッション ID であり、直接使用できます。

### 認証コード

エンドユーザーによって付与されるアクセスを表示する、有効期間の短いトークンです。認証コードは、アクセストークンと更新トークンを取得するために使用されます。

### 接続アプリケーション

OAuth プロトコルを使用して Salesforce ユーザーと外部アプリケーションの両方を検証する、Salesforce の外部アプリケーション。

### コンシューマー鍵

コンシューマー (この場合は、Mobile SDK アプリケーション) が Salesforce に自身の身分を証明するために使用する値です。client\_id と呼ばれます。

### コンシューマーの秘密

コンシューマー鍵の所有権を確認するためにコンシューマーにより使用される秘密。セキュリティを強化するため、Mobile SDK アプリケーションは、コンシューマーの秘密を使用しません。

### 更新トークン

新しいアクセストークンを取得するためにコンシューマーが使用するトークン。エンドユーザーがアクセスを再度承認する必要がありません。

### リモートアクセスアプリケーション (廃止)

リモートアクセスアプリケーションとは、OAuth プロトコルを使用して Salesforce ユーザーと外部アプリケーションの両方を検証する、Salesforce の外部アプリケーションです。リモートアクセスアプリケーションは、接続アプリケーションとして実装されています。リモートアクセスアプリケーションは廃止され、接続アプリケーションになりました。

## OAuth2.0 認証フロー

---

認証フローは、デバイスにおける認証状態によって異なります。次の手順は、アプリケーションの起動時に Salesforce 認証が行われることを前提としています。

### 初回の認証フロー

1. 顧客が Mobile SDK アプリケーションを開きます。
2. 認証を促すメッセージが表示されます。
3. 顧客がユーザー名とパスワードを入力します。
4. アプリケーションが顧客のログイン情報を Salesforce に送信し、それと引き換えに認証成功の確認としてセッション ID を受信します。
5. 顧客が、アプリケーションの要求を承認してアプリケーションにアクセス権を付与します。

6. アプリケーションが起動します。

## 継続的な認証

1. 顧客がモバイルアプリケーションを開きます。
2. セッション ID が有効な場合は、アプリケーションが直ちに起動します。セッション ID が期限切れの場合は、初期認証から取得された更新トークンをアプリケーションで使用して、更新されたセッション ID を取得します。
3. アプリケーションが起動します。

## PIN の認証 (省略可能)

PIN の保護は、モバイルポリシーの機能であり、Salesforce 接続アプリケーション定義で有効になっている場合にのみ使用されます。「[PIN セキュリティについて](#)」を参照してください。

## OAuth 2.0 Web サーバーフロー

Mobile SDK 11.0 以降、OAuth 2.0 Web サーバーフローがデフォルトの認証フローになります。Mobile SDK 11.0 にアップグレードした時点でクライアントアプリケーションを変更する必要はありません。ただし、接続アプリケーションで [Web サーバーフローの秘密が必要] チェックボックスがオフになっていることを確認してください。

Mobile SDK 11.0 より前のバージョンでは、ログイン試行ごとに中間認証画面でユーザーに認証要求を承認または拒否するように促すメッセージが表示されます。11.0 の Web サーバーフローにアップグレードすると、この画面は、ユーザーが接続アプリケーションを初めて承認するときのみ表示されます (ユーザーが自己承認できるようにアプリケーションが設定されている場合)。この動作の変更が日常的な使用に影響する可能性はありませんが、自動化されたテストなどの領域では、たとえば、ログインフロー内でこの中間画面を想定するようにテストが記述されている場合は影響することがあります。

Web サーバーフローの使用方法についての詳細は、「[Web アプリケーションインテグレーションの OAuth 2.0 Web サーバーフロー](#)」を参照してください。

## ユーザーエージェントフローのオプトアウト

Mobile SDK 11.0 以降、Web サーバーフローをオプトアウトするには、SalesforceSDKManager で [ユーザーエージェントフロー](#) (ページ 456) に戻すことができます。

### Android

```
SalesforceSDKManager.getInstance().setUseWebServerAuthentication(false)
```

### iOS

#### Swift

```
SalesforceManager.shared.useWebServerAuthentication = false
```

#### Objective-C

```
[SalesforceSDKManager sharedManager].useWebServerAuthentication = NO;
```

## OAuth 2.0 ユーザーエージェントフロー

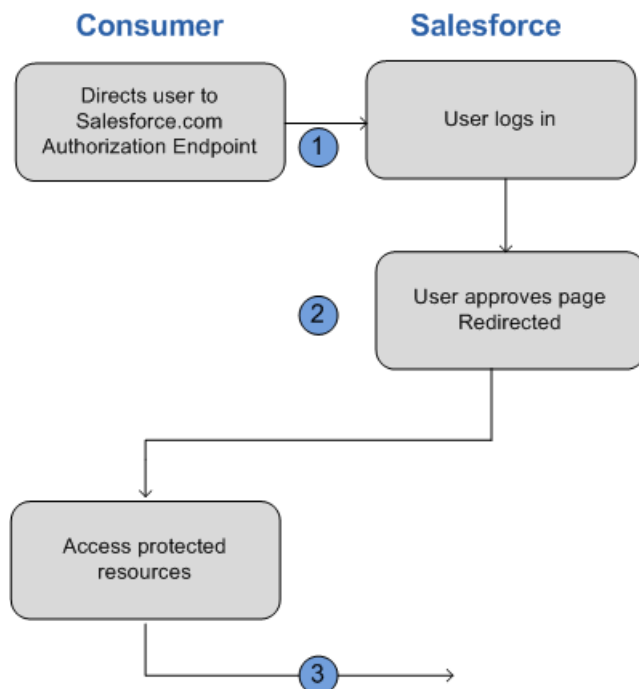
ユーザーは、OAuth 2.0 ユーザーエージェントフローを使用して、デスクトップまたはモバイルアプリケーションが外部ブラウザや組み込みブラウザを使用してデータにアクセスすることを承認します。JavaScriptなどのスクリプト言語を使用してブラウザ内で実行されるクライアントアプリケーションでも、このフローを使用できます。このフローは、[OAuth 2.0 暗黙的許可種別](#)を使用します。

ユーザーエージェントフローでは、クライアントアプリケーションを Salesforce API に統合する接続アプリケーションは、アクセストークンを HTTP リダイレクトとして受信します。接続アプリケーションは、ユーザーエージェントを Web サーバーまたはアクセス可能なローカルリソースにリダイレクトするよう認証サーバーに要求します。Web サーバーは応答からアクセストークンを抽出して接続アプリケーションに渡すことができます。セキュリティ上の理由により、トークン応答は、URL でハッシュタグ (#) フラグメントとして指定されます。この形式により、サーバーや、参照ヘッダー内の他のサーバーにトークンが渡されないようにします。

 **警告:** アクセストークンはエンコードされてリダイレクト URL に挿入されているため、ユーザーや、デバイス上の他のアプリケーションに公開される場合があります。

JavaScript を使用して認証する場合、`window.location.replace()` をコールし、ブラウザの履歴からコールバックを削除してください。

 **メモ:** これらのクライアント種別の接続アプリケーションはユーザーごとの秘密を保護できます。ただし、クライアントの実行可能ファイルはユーザーのデバイス上にあるため、クライアントの秘密にアクセスしたり悪用したりすることができます。このため、ユーザーエージェントフローでは、クライアントの秘密を使用しません。認証は、ユーザーエージェントの発生元が同じポリシーに基づいています。また、ユーザーエージェントフローでは、帯域外 POST はサポートしていません。



たとえば、Salesforce Mobile SDK を使用して、Salesforce 組織にある顧客の連絡先情報を検索するモバイルアプリケーションを構築するとします。Mobile SDK は、接続アプリケーションに OAuth 2.0 ユーザーエージェントフロー

を実装し、モバイルアプリケーションを Salesforce API に統合して、定義したデータに対する承認されたアクセスを許可します。フローの手順は、次のとおりです。

1. ユーザーがモバイルアプリケーションを開きます。
2. このモバイルアプリケーションを認証して承認するために、接続アプリケーションがユーザーを Salesforce にリダイレクトします。
3. ユーザーはこの認証フローのアクセスを承認します。
4. 接続アプリケーションが Salesforce からリダイレクト URL へのコールバックを受信し、アクセストークンと更新トークンを抽出します。
5. 接続アプリケーションがこのアクセストークンを使用して、ユーザーの代わりにデータにアクセスします。

## Mobile SDK 11.0 以降のユーザーエージェントフロー

 **メモ:** Mobile SDK 10.2 までは、iOS と Android の両方の Web ビュー認証と Android の高度な認証で OAuth 2.0 ユーザーエージェントフローが使用されます。Mobile SDK 11.0 以降、この両方のプラットフォームのデフォルトの認証で、コード交換の証明鍵 (PKCE) を使用した [OAuth 2.0 Web サーバーフロー](#) (ページ 455) が使用され、セキュリティが強化されます。

Mobile SDK 11.0 以降、アプリケーションでユーザーエージェントフローを引き続き使用する場合は、そのオプションを `SalesforceSDKManager` で有効にすることができます。ログインを開始する前に、次のいずれかのメソッドをアプリケーションクラスの `init` メソッドで使用します。

### Android

```
SalesforceSDKManager.getInstance().setUseWebServerAuthentication(false)
```

### iOS

#### Swift

```
SalesforceManager.shared.useWebServerAuthentication = false
```

#### Objective-C

```
[SalesforceSDKManager sharedManager].useWebServerAuthentication = NO;
```

## OAuth 2.0 更新トークンフロー

クライアントは、接続アプリケーション経由でアクセストークンを取得すると、現在のセッションの期限が切れたときに更新トークンを使用して新しいセッションを取得できます。接続アプリケーションのセッションタイムアウト値によって、アクセストークンを無効にするタイミングおよび更新トークンを使用して新しいトークンを適用するタイミングが決まります。

更新トークンフローの手順は、次のとおりです。

 **メモ:** Mobile SDK アプリケーションでは、SmartStore 機能を使用してデータをオフラインで使用するためにローカルに保存できます。SmartStore データは、本質的に揮発性です。有効期限は、認証済みユーザーおよび OAuth トークンの状態に関係しています。ユーザーがアプリケーションからログアウトすると、そのユーザーに関連付けられているスプードデータは SmartStore によりすべて削除されます。同様に、OAuth 更新トークンが取り消されるか期限切れになると、ユーザーのアプリケーション状態がリセットされ、



SmartStore のすべてのデータが消去されます。アプリケーションの設計時には、SmartStore データの揮発性を十分に考慮してください。この警告は、組織で更新トークンに短期の有効期限を設定している場合に特に重要です。

## Mobile SDK のログインおよび認証フローの詳細

Mobile SDK では複雑なログインおよび認証フローは内部で処理されるため、開発者自身が調整する必要はありません。アプリケーションでログイン失敗が報告された場合は、この詳細な説明を問題の特定に役立ててください。



## 範囲パラメーターの値

OAuth では、サーバーおよびクライアントの両方に範囲設定が必要です。サーバー側とクライアント側間の同意によって、範囲に関する契約が定義されます。

- **サーバー側** — Salesforce サーバーの接続アプリケーションで、範囲権限を定義します。この設定により、MobileSDK アプリケーションなど、クライアントアプリケーションが要求できるアクセスレベルが決まります。少なくとも、コードで指定した内容に合わせて接続アプリケーションの OAuth 設定を指定します。ほとんどのアプリケーションでは、`refresh_token`、`web`、`api` で十分です。

- クライアント側 — Mobile SDK アプリケーションで範囲要求を指定します。クライアントの範囲要求は、接続アプリケーションの範囲権限のサブセットである必要があります。

## サーバー側の設定

次の範囲パラメーター値を設定できます。

| 値                                                                     | 説明                                                                                                                                                                        |
|-----------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Customer Data Platform データに対する ANSI SQL クエリを実行</b> (cdp_query_api) | ユーザーの代わりに Data Cloud データの ANSI SQL クエリを許可します。                                                                                                                             |
| <b>Pardot サービスを管理</b> (pardot_api)                                    | ユーザーの代理として Marketing Cloud Account Engagement API サービスへのアクセスを許可します。Account Engagement でアクセス可能なすべてのサービスを管理します。(Pardot は Marketing Cloud Account Engagement という名前に変更されました。) |
| <b>Customer Data Platform プロファイルデータを管理</b> (cdp_profile_api)          | Data Cloud REST API データへのアクセスを許可します。この範囲を使用して、プロファイルレコードを管理します。                                                                                                           |
| <b>Connect REST API リソースにアクセス</b> (chatter_api)                       | ユーザーの代理として Connect REST API リソースへのアクセスを許可します。                                                                                                                             |
| <b>Customer Data Platform 取り込み API データを管理</b> (cdp_ingest_api)        | Data Cloud 取り込み API データへのアクセスを許可します。この範囲を使用して、Data Cloud で外部データセットをアップロードおよび管理します。この範囲は JSON Web トークン (JWT) でパッケージ化されます。                                                  |
| <b>Analytics REST API Charts Geodata リソースにアクセス</b> (eclair_api)       | Analytics REST API Charts Geodata リソースへのアクセスを許可します。                                                                                                                       |
| <b>Analytics REST API リソースにアクセス</b> (wave_api)                        | Analytics REST API リソースへのアクセスを許可します。                                                                                                                                      |
| <b>API を使用してユーザーデータを管理</b> (api)                                      | REST API や Bulk API 2.0 などの API を使用して、現在のログインユーザーのアカウントへのアクセスを許可します。この範囲には、Connect REST API リソースへのアクセスを許可する chatter_api も含まれます。                                           |
| <b>カスタム権限にアクセス</b> (custom_permissions)                               | 接続アプリケーションに関連付けられた組織内のカスタム権限へのアクセスを許可します。この範囲は、現在のユーザーの各権限が有効になっているかどうかを示します。                                                                                             |
| <b>ID URL サービスにアクセス</b> (id, profile, email, address, phone)          | ID URL サービスへのアクセスを許可します。profile、email、address、phone を要求すれば、id を使用した場合と同じ結果を個別に得られます。これは、これらが同義であるためです。                                                                    |

| 値                                         | 説明                                                                                                                                                                                                                           |
|-------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Lightning アプリケーションにアクセス (lightning)       | ハイブリッドアプリケーションで OAuth 2.0 ハイブリッドアプリケーショントークンフローおよびハイブリッドアプリケーション更新トークンフローを介して Lightning の子セッションを直接取得できるようにします。                                                                                                               |
| コンテンツリソースにアクセス (content)                  | ハイブリッドアプリケーションで OAuth 2.0 ハイブリッドアプリケーショントークンフローおよびハイブリッドアプリケーション更新トークンフローを介してコンテンツの子セッションを直接取得できるようにします。                                                                                                                     |
| 一意のユーザー識別子にアクセス (openid)                  | <p>OpenID Connect アプリケーションの現在のログインユーザーの一意の識別子へのアクセスを許可します。</p> <p>OAuth 2.0 ユーザーエージェントフローと OAuth 2.0 Web サーバーフローでは、openid 範囲を使用します。この範囲を使用すると、アクセストークンの他にも、<a href="#">「OpenID Connect の仕様」</a>に適合する署名付き ID トークンを受信できます。</p> |
| フルアクセス (full)                             | <p>ログインユーザーがアクセスできるすべてのデータへのアクセスを許可し、その他すべての範囲が対象となります。</p> <p>full は更新トークンを返しません。更新トークンを取得するには、refresh_token の範囲を明示的に要求する必要があります。</p>                                                                                       |
| いつでも要求を実行 (refresh_token, offline_access) | 更新トークンを受け取る資格が要求元クライアントにある場合に、更新トークンを返すことを許可します。更新トークンにより、ユーザーがオフラインのときにアプリケーションはユーザーのデータを操作できます。このトークンは、offline_access を要求した場合と同じ意味になります。                                                                                   |
| Visualforce アプリケーションにアクセス (visualforce)   | <p>顧客が作成した Visualforce ページへのアクセスのみを許可します。この範囲では標準の Salesforce UI へのアクセスは許可されません。</p> <p>ハイブリッドアプリケーションで Visualforce の子セッションを直接取得できるようにするには、この範囲を OAuth 2.0 ハイブリッドアプリケーショントークンフローまたはハイブリッドアプリケーション更新トークンフローに含めます。</p>         |

| 値                                           | 説明                                                                                                                      |
|---------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| Webブラウザを使用してユーザーデータを管理 (web)                | Webで <code>access_token</code> を使用することを許可します。この範囲には <code>visualforce</code> も含まれ、顧客が作成した Visualforce ページへのアクセスが許可されます。 |
| チャットボットサービスにアクセス (chatbot_api)              | Einstein ボット API サービスへのアクセスを許可します。                                                                                      |
| ヘッドレス登録 API にアクセス (user_registration_api)   | ヘッドレス登録フローの API へのアクセスを許可します。認証を要求するようにフローを設定する場合、この範囲が含まれるアクセストークンを渡す必要があります。                                          |
| パスワードを忘れた場合のヘッドレスAPIにアクセス (forgot_password) | ヘッドレスのパスワードを忘れた場合のフローの API へのアクセスを許可します。認証を要求するようにフローを設定する場合、この範囲が含まれるアクセストークンを渡す必要があります。                               |
| すべてのDataCloudAPIリソースにアクセス (cdp_api)         | すべての Data Cloud API リソースへのアクセスを許可します。                                                                                   |
| SalesforceAPIプラットフォームにアクセス (sfap_api)       | 将来の使用のために予約されています。                                                                                                      |
| インタラクション API リソースにアクセス (interaction_api)    | 将来の使用のために予約されています。                                                                                                      |

 **メモ:** Mobile SDK アプリケーションでは、サーバー側の[接続アプリケーション]設定で常に `refresh_token` を選択する必要があります。範囲に `full` を選択しても、`refresh_token` を明示的に選択する必要があります。

## クライアント側の設定

Mobile SDK アプリケーションの範囲の設定は、次のルールで制御されます。

| 範囲                         | Mobile SDK アプリケーションの設定                                        |
|----------------------------|---------------------------------------------------------------|
| <code>refresh_token</code> | アプリケーションの Mobile SDK から暗黙的に要求されます。アプリケーションの範囲リストに含める必要はありません。 |
| <code>api</code>           | Salesforce REST API コールを行う場合に含めます (ほとんどのアプリケーションに適用されます)。     |

| 範囲          | Mobile SDK アプリケーションの設定                                                                    |
|-------------|-------------------------------------------------------------------------------------------|
| web         | Salesforce 組織で定義されたページにアプリケーションからアクセスする場合に含めます (Salesforce ベースの Web ページを読み込むアプリケーションの場合)。 |
| full        | すべての権限を要求する場合に含めます。(Mobile SDK では、refresh_token が暗黙的に要求されます)。                             |
| chatter_api | アプリケーションから Connect REST API をコールする場合に含めます。                                                |
| id          | (不要)                                                                                      |
| visualforce | 代わりに、web を使用します。                                                                          |

## ID URL の使用

ID URL は id 範囲パラメーターで返されます。たとえば、  
<https://login.salesforce.com/id/00Dx0000000BV7z/005x00000012Q9P> のようになります。

ID URL は、ユーザーに関する追加の情報(ユーザー名、メールアドレス、組織 ID など)を照会するための RESTful API でもあります。また、クライアントが通信できるエンドポイント (プロフィールの写真、アクセス可能な API エンドポイントなど) も返します。

URL の形式は、<https://login.salesforce.com/id/orgID/userID> です。ここで、orgID はユーザーが属する Salesforce 組織の ID、userID は Salesforce のユーザー ID です。

## ID URL 要求パラメーター

次のパラメーターは、アクセストークンや ID URL と併用できます。アクセストークンは、認証要求ヘッダー、または oauth\_token パラメーターのある要求で使用できます。

| パラメーター       | 説明                                                                                                                                                  |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| access_token | 接続アプリケーションがクライアントアプリケーションの代わりに保護されたリソースへのアクセスを要求するために使用する OAuth トークン。範囲の形式の追加権限にはアクセストークンが付随することがあります。                                              |
| format       | 省略可能。返される出力の形式を指定します。値は次のとおりです。 <ul style="list-style-type: none"> <li>json</li> <li>xml</li> </ul> クライアントは次のいずれかの形式を使用して、受け入れ要求ヘッダーの返される形式も指定できます。 |

| パラメーター      | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|             | <ul style="list-style-type: none"> <li>Accept: application/json</li> <li>Accept: application/xml</li> <li>Accept: application/x-www-form-urlencoded</li> </ul> <p>要求ヘッダーでは次の情報もサポートされています。</p> <ul style="list-style-type: none"> <li>*/* のワイルドカードが受け入れられ、JSON を返します。</li> <li>値のリスト (左から右に確認される)。たとえば、application/xml,application/json,application/html,*/* は XML を返します。</li> </ul> <p>format パラメーターは、アクセス要求ヘッダーより優先されます。</p>                                                                                                                                                 |
| version     | <p>省略可能。SOAP API バージョン番号またはリテラル文字列 latest を指定します。この値が指定されていない場合、返される API URL には、バージョン番号の代わりに、リテラル値 {version} が含まれます。値が latest として指定されると、最新の API バージョンが使用されます。</p>                                                                                                                                                                                                                                                                                                                                                                                                              |
| PrettyPrint | <p>省略可能。ヘッダー内でのみ使用できます。URL パラメーターとしては使用できません。サイズではなく読みやすさを目的として、返される XML または JSON 出力を最適化するためにこのパラメーターを指定します。たとえば、ヘッダーで X-PrettyPrint:1 を使用します。</p>                                                                                                                                                                                                                                                                                                                                                                                                                             |
| callback    | <p>省略可能。有効な JavaScript 関数名を指定します。指定した形式が JSONP の場合、このパラメーターを使用できます。出力はこの関数名 (JSONP) でラップされます。たとえば、https://server/id/orgid/userid/ に対する要求では {"function":"name"} を返し、https://server/id/orgid/userid/?callback=baz に対する要求では baz({"function":"name"}) を返します。</p> <p> <b>メモ:</b> JSONP は、厳格な MIME タイプのチェックにより、ID サービス要求では返されなくなりました。要求でコールバックパラメーターに 'format=jsonp' を追加して ID サービスが JavaScript を返すようにする必要があります。ID サービスで JSONP 形式が検出されると、必要な JavaScript タイプ ('application/javascript') が返されます。</p> |

## ID URL レスポンス

要求に成功すると、ID URL 応答はクエリ対象ユーザーおよび組織に関する情報を返します。

次の ID URL 応答は XML 形式です。

```
<?xml version="1.0" encoding="UTF-8"?>
<user xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<id>https://MyDomainName.my.salesforce.com/id/00Dx000.../005x000...</id>
<asserted_user>true</asserted_user>
<user_id>005x000000...</user_id>
```

```

<organization_id>00Dx000...</organization_id>
<nick_name>admin1.2777578168398293E12...</nick_name>
<display_name>Alan Van</display_name>
<email>admin@mycompany.com</email>
<status>
 <created_date xsi:nil="true"/>
 <body xsi:nil="true"/>
</status>
<photos>
 <picture>https://MyDomainName--03925205UAF.file.force-user-content.com</picture>
 <thumbnail>https://MyDomainName--03925205UAF.file.force-user-content.com</thumbnail>
</photos>
<urls>

<enterprise>https://MyDomainName.my.salesforce.com/services/Soap/c/{version}/00Dx000...</enterprise>

<metadata>https://MyDomainName.my.salesforce.com/services/Soap/m/{version}/00Dx000...</metadata>

<partner>https://MyDomainName.my.salesforce.com/services/Soap/u/{version}/00Dx000...</partner>

 <rest>https://MyDomainName.my.salesforce.com/services/data/v{version}/
</rest>
 <subjects>https://MyDomainName.my.salesforce.com/services/data/v{version}/subjects/
</subjects>
 <search>https://MyDomainName.my.salesforce.com/services/data/v{version}/search/
</search>
 <query>https://MyDomainName.my.salesforce.com/services/data/v{version}/query/
</query>
 <profile>https://MyDomainName.my.salesforce.com/005x000...</profile>
</urls>
<active>true</active>
<user_type>STANDARD</user_type>
<language>en_US</language>
<locale>en_US</locale>
<utcOffset>-28800000</utcOffset>
<last_modified_date>2021-04-28T20:54:09.000Z</last_modified_date>
</user>

```

また、この応答は JSON 形式です。

```

{"id":"https://MyDomainName.my.salesforce.com/id/00Dx000.../005x000...",
"asserted_user":true,
"user_id":"005x000...",
"organization_id":"00Dx000...",
"nick_name":"admin1.2777578168398293E12...",
"display_name":"Alan Van",
"email":"admin@mycompany.com",
"status":{"created_date":null,"body":null},
"photos":{"picture":"https://MyDomainName--03925205UAF.file.force-user-content.com",
 "thumbnail":"https://MyDomainName--03925205UAF.file.force-user-content.com"},
"urls":

```



```
{
 "enterprise": "https://MyDomainName.my.salesforce.com/services/Soap/c/{version}/00Dx000...",
 "metadata": "https://MyDomainName.my.salesforce.com/services/Soap/m/{version}/00Dx000...",
 "partner": "https://MyDomainName.my.salesforce.com/services/Soap/u/{version}/00Dx000...",
 "rest": "https://MyDomainName.my.salesforce.com/services/data/v{version}/",
 "subjects": "https://MyDomainName.my.salesforce.com/services/data/v{version}/subjects/",
 "search": "https://MyDomainName.my.salesforce.com/services/data/v{version}/search/",
 "query": "https://MyDomainName.my.salesforce.com/services/data/v{version}/query/",
 "profile": "https://MyDomainName.my.salesforce.com/005x000...",
 "active": true,
 "user_type": "STANDARD",
 "language": "en_US",
 "locale": "en_US",
 "utcOffset": -28800000,
 "last_modified_date": "2021-04-28T20:54:09.000+0000"
}
```

次の表に、返されるパラメーターを示します。

| パラメーター          | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| id              | ID URL。照会された URL と同じ URL です。                                                                                                                                                                                                                                                                                                                                                                                                                             |
| asserted_user   | 指定されたアクセストークンがこの ID に発行されたかどうかを示す Boolean 値。                                                                                                                                                                                                                                                                                                                                                                                                             |
| user_id         | クエリ対象ユーザーのユーザー ID。                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| username        | クエリ対象ユーザーのユーザー名。                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| organization_id | クエリ対象ユーザーの Salesforce 組織の ID。                                                                                                                                                                                                                                                                                                                                                                                                                            |
| nick_name       | クエリ対象ユーザーの Experience Cloud のニックネーム。                                                                                                                                                                                                                                                                                                                                                                                                                     |
| display_name    | クエリ対象ユーザーの表示名 (氏名)。                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| email           | クエリ対象ユーザーのメールアドレス。                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| email_verified  | <p>「Salesforce によるこそ」メールのリンクをクリックして、クエリ対象ユーザーのメールが確認されたかどうかを示します。</p> <p>次の操作の後、ユーザーが受信したメール内のリンクをクリックした場合、email_verified 値は true に設定されます。たとえば、Salesforce システム管理者が「RobertaSmith」というユーザーを作成したとします。Roberta は、アカウントを確認するためのリンクが含まれる「Salesforce へようこそ」のメールメッセージを受信します。Roberta がリンクをクリックすると、email_verified 値が true に設定されます。</p> <ul style="list-style-type: none"> <li>ユーザーがメールアドレスを変更した</li> <li>ユーザーがパスワードを変更したか、Salesforce システム管理者がパスワードをリセットした</li> </ul> |

| パラメーター                | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                       | <ul style="list-style-type: none"> <li>• ユーザーが新しいブラウザーまたはデバイスからのログイン時にIDを検証した</li> <li>• Salesforce システム管理者がユーザーを新しいユーザーとして作成した</li> </ul>                                                                                                                                                                                                                                                                                                                      |
| first_name            | クエリ対象ユーザーの名。                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| last_name             | クエリ対象ユーザーの姓。                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| timezone              | クエリ対象ユーザーの設定で指定されているタイムゾーン。                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| photos                | <p>picture または thumbnail として指定される、クエリ対象ユーザーのプロファイル写真への URL の対応付け。</p> <p> <b>メモ:</b> これらの URL にアクセスするには、アクセストークンを渡す必要があります。「<a href="#">アクセストークン</a>」を参照してください。</p>                                                                                                                                                                                                            |
| addr_street           | クエリ対象ユーザーの設定の住所で指定されている町名・番地。                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| addr_city             | クエリ対象ユーザーの設定の住所で指定されている市区郡。                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| addr_state            | クエリ対象ユーザーの設定の住所で指定されている都道府県。                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| addr_country          | クエリ対象ユーザーの設定の住所で指定されている国。                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| addr_zip              | クエリ対象ユーザーの設定の住所で指定されている郵便番号。                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| mobile_phone          | クエリ対象ユーザーの設定で指定されている携帯電話番号。                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| mobile_phone_verified | 有効な番号であることをクエリ対象ユーザーが確認した携帯電話番号。                                                                                                                                                                                                                                                                                                                                                                                                                                |
| status                | <p>クエリ対象ユーザーの現在の Chatter 状況。</p> <ul style="list-style-type: none"> <li>• created_date — ユーザーの前の投稿の作成日付の xsd datetime 値 (2010-05-08T05:17:51.000Z など)。</li> <li>• body — 投稿の本文。</li> </ul>                                                                                                                                                                                                                                                                      |
| urls                  | <p>クエリ対象ユーザーで利用できる、さまざまなAPIエンドポイントを含む対応付け。</p> <p> <b>メモ:</b> REST エンドポイントへのアクセスでは、アクセストークンを渡す必要があります。「<a href="#">アクセストークン</a>」を参照してください。</p> <ul style="list-style-type: none"> <li>• enterprise (SOAP)</li> <li>• metadata (SOAP)</li> <li>• partner (SOAP)</li> <li>• rest (REST)</li> <li>• subjects (REST)</li> <li>• search (REST)</li> <li>• query (REST)</li> </ul> |

| パラメーター             | 説明                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                    | <ul style="list-style-type: none"> <li>recent (REST)</li> <li>profile</li> <li>feeds (Chatter)</li> <li>feed-items (Chatter)</li> <li>groups (Chatter)</li> <li>users (Chatter)</li> <li>custom_domain</li> </ul> <p> <b>メモ:</b> 設定または伝搬されたカスタムドメインが組織にない場合、この値は省略されます。</p> |
| active             | クエリ対象ユーザーが有効であるかどうかを指定する Boolean 値。                                                                                                                                                                                                                                                                                                                          |
| user_type          | クエリ対象ユーザーの種別。                                                                                                                                                                                                                                                                                                                                                |
| language           | クエリ対象ユーザーの言語。                                                                                                                                                                                                                                                                                                                                                |
| locale             | クエリ対象ユーザーのロケール。                                                                                                                                                                                                                                                                                                                                              |
| utcOffset          | クエリ対象ユーザーのタイムゾーンのUTCからのオフセット(ミリ秒単位)。                                                                                                                                                                                                                                                                                                                         |
| last_modified_date | ユーザーの前回の変更の xsd datetime 形式のタイムスタンプ (2010-06-28T20:54:09.000Z など)。                                                                                                                                                                                                                                                                                           |
| is_app_installed   | 接続アプリケーションがユーザーの組織にインストールされていて、ユーザーのアクセストークンがOAuthフローを使用して作成されている場合、値は true になります。接続アプリケーションがインストールされていない場合、応答にこの値は含まれません。応答を解析する場合、このプロパティの有無と値を確認してください。                                                                                                                                                                                                   |
| mobile_policy      | <p>モバイル接続アプリケーションを管理するための特定の値。これらの値は、接続アプリケーションが現在のユーザーの組織にインストールされていて、そのアプリケーションでセッションタイムアウト値が定義され、モバイル PIN の長さの値が定義されている場合にのみ使用できます。</p> <ul style="list-style-type: none"> <li>screen_lock — 操作しなくなってから画面をロックするまでの待機時間。</li> <li>pin_length — モバイルアプリケーションにアクセスするために必要な識別番号の長さ。</li> </ul>                                                                |
| push_service_type  | <p>接続アプリケーションが Apple Push Notification Service (APNS) for iOS プッシュ通知に登録されている場合は apple に設定されます。Google Cloud Messaging (GCM) for Android プッシュ通知に登録されている場合は androidGcm に設定されます。</p> <p>応答値は配列型です。</p>                                                                                                                                                           |

| パラメーター             | 説明                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| custom_permissions | <p>custom_permissions 範囲パラメーターが要求にある場合、接続アプリケーションに関連付けられた組織のカスタム権限を含む対応付けが応答にあります。接続アプリケーションが組織にインストールされていない場合や、カスタム権限に関連付けられていない場合、custom_permissions の対応付けは応答にありません。</p> <p>要求の例を次に示します。</p> <pre>http://MyDomainName.my.salesforce.com/services/oauth2/authorize?response_type=token&amp;client_id=3MGBLK6RNNMBKVB3UjF.sSDw6_2sS7BcHEJC_&amp;redirect_uri=http://www.example.org/cp/security/auth/useragent_flow_callback.jsp&amp;scope=api%20id%20custom_permissions</pre> <p>ID URL レスポンスの JSON ブロックを次に示します。</p> <pre>"custom_permissions": {   "Email.View":true,   "Email.Create":false,   "Email.Delete":false }</pre> |

## Android アプリケーションでのカスタムログインサーバーの設定

Trialforceを使用する Salesforce パートナーなど、特殊な場合は、ユーザーのログイン要求をカスタムログイン URI にリダイレクトできます。

Android では、ログインホストはサーバー接続と呼ばれます。SalesforceSDK プロジェクトの `res/xml/servers.xml` ファイルでサーバー接続の標準リストを参照できます。Mobile SDK では、このファイルを使用して本番サーバーと Sandbox サーバーを定義します。

Android の場合、次のいずれかの方法を使用してデフォルトのログインホストを設定できる可能性があります。

### 1. MDM の適用

- 起動時に、アプリケーションの MDM プロバイダーによってログイン URI が設定されます。
- MDM ポリシーでは、ユーザーがログインホストを変更できないようにナビゲーションバーと設定アイコンを非表示にすることもできます。

### 2. servers.xml ファイルを使用したアプリケーションによる設定

ネイティブ Android プロジェクトで独自の `res/xml/servers.xml` ファイルを作成することで、カスタムサーバーをランタイムリストに追加できます。servers.xml ファイルの最初にリストされているサーバーがデフォルトのログインサーバーとしてアプリケーションの起動時に使用されます。servers.xml のルー

ト XML 要素は <servers> です。このルートには、<server> エントリをいくつでも含めることができます。各 <server> エントリには、name (わかりやすい任意の表示ラベル) と url (「https://」プレフィックスが含まれるログインサーバーの Web アドレス) という 2 つの属性が必要です。

servers.xml ファイルの例を次に示します。

```
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <server name="XYZ.com Login" url="https://myloginserver.cloudforce.com"/>
</servers>
```

 **メモ:** Android エミュレーターで XML の変更をテストするには、次の手順を実行することをお勧めします。

- a. エミュレーターでアプリケーションが実行中の場合は強制終了します。
- b. エミュレーターでアプリケーションをアンインストールします。
- c. フルクリーンアップと再構築を行います。
- d. アプリケーションを実行します。

### 3. [接続を追加] ボタンを使用したユーザーによる設定

ユーザーは次の手順でカスタムログインサーバーを設定できます。

- a. ログインしないでアプリケーションを開始します。
- b. ログイン画面で、左上隅にある設定アイコンをタップします。
- c. [サーバーを変更] をタップします。
- d. [接続を追加] をタップします。
- e. 今後アクセスしたときにこの設定を識別できるように、名前を入力します。
- f. カスタムログインホストの URL を入力します。https:// プレフィックスを必ず含めてください。次に、Experience Cloud サイトの一般的な URL の入力方法の例を示します。

```
https://MyDomainName.my.site.com/fineapps
```

Mobile SDK ではデフォルトでこの機能が有効になっています。Mobile SDK for Android では、[サーバーを変更] または [接続を追加] オプションをプログラムで無効にすることはできません。

### **重要:**

- Android では、ログイン URL を指定するときに「https://」が常に含まれます。
- 起動時に、コンパイル時の設定が MDM ランタイム設定で上書きされます。

## iOS アプリケーションでのカスタムログインサーバーの設定

Trialforce を使用する Salesforce パートナーなど、特殊な場合は、ユーザーのログイン要求をカスタムログイン URI にリダイレクトできます。

iOS アプリケーションでは、ログインサーバーは多くの場合「ログインホスト」と呼ばれます。Mobile SDK によって、SalesforceSDKCore 製品で本番サーバーと Sandbox サーバーの標準ログイン URI が定義されます。これらの 2 つのログインホストは、[接続を選択] ログイン画面に表示されます。

iOS の場合、次のいずれかの方法を使用してデフォルトのログインホストを設定できる可能性があります。

### 1. MDM の適用

- 起動時に、アプリケーションの MDM プロバイダーによってログイン URI が設定されます。
- MDM ポリシーでは、ユーザーがログインホストを変更できないようにナビゲーションバーと設定アイコンを非表示にすることもできます。

### 2. info.plist ファイルを使用したアプリケーションによる設定

- アプリケーションは、プロジェクトの info.plist プロパティファイルでデフォルトのログイン URI を設定できます。ログインホストプロパティ名は SFDCOAuthLoginHost です。
- 起動時に、ユーザー定義のログインホストが SFDCOAuthLoginHost 設定で上書きされます。
- デフォルトでは、SFDCOAuthLoginHost プロパティは「login.salesforce.com」に設定されています。
- ログイン URI を指定するときに、「https://」などのプロトコルプレフィックスは使用しないでください。

### 3. [接続を追加] 画面を使用したユーザーによる設定

ユーザーは次の手順でカスタムログインサーバーを設定できます。

- a. ログインしないでアプリケーションを開始します。
- b. ログイン画面で [設定] をタップするか、上部のナビゲーションバーの「歯車」アイコン  をタップします。
- c. [接続を選択] 画面で、プラスアイコン  をタップします。
- d. (省略可能ですが推奨) 今後アクセスしたときにこの設定を簡単に識別できるように、表示ラベルを入力します。
- e. カスタムログインホストの URI を入力します。必ず https:// プレフィックスを削除してください。次に、Experience Cloud サイトの一般的な URI の入力方法の例を示します。

```
MyDomainName.my.site.com/fineapps
```

Mobile SDK ではデフォルトでこの機能が有効になっています。[接続を追加] オプションを無効にする場合は、SFLoginHostViewController プロパティを設定します。

### ❗ 重要:

- 起動時に、コンパイル時の設定が MDM ランタイム設定で上書きされます。
- バージョン 4.1 より前の Mobile SDK for iOS アプリケーションでは、アプリケーションの設定バンドルでカスタムログイン URI を定義していました。Mobile SDK 4.1 以降では、iOS アプリケーションの設定バンドルがなくなりました。代わりに、アプリケーションの info.plist ファイルの SFDCOAuthLoginHost プロパティを使用して、カスタムログイン URI を組み込むことができます。

### 関連トピック:

[プログラムによる iOS ログイン画面のカスタマイズ](#)



## OAuth トークンの取り消し

ユーザーがアプリケーションからログアウトした、アプリケーションがタイムアウトした、または他の理由により無効になった場合、ログインしたユーザーのログイン情報はモバイルアプリケーションからクリアされます。これにより、サーバー接続が効果的に終了します。

また、Mobile SDK では、ログアウトの一環としてサーバーから更新トークンも取り消されます。

「[不透明 OAuth トークンの取り消し](#)」を参照してください

## Android ネイティブアプリケーションでの更新トークンの取り消し

更新トークンが管理者によって取り消されると、デフォルトの動作では現在のユーザーが自動的にログアウトされます。その結果、次のようになります。

- アプリケーションで行う後続の REST API コールがすべて失敗する。
- ユーザーのアカウント情報と、キャッシュされたオフラインデータが破棄される。
- ユーザーがページから強制的に移動される。
- アプリケーションを引き続き使用するには、ユーザーが Salesforce に再度ログインする必要がある。

これらの副次的影響として、管理者のアクションに対して安全な応答が行われます。

## トークンの取り消しイベント

トークンの取り消しイベントが発生すると、ClientManager オブジェクトから Android スタイルの通知が送信されます。この通知が意図するアクションは、ClientManager.ACCESS\_TOKEN\_REVOKE\_INTENT 定数で宣言されます。

ACCESS\_TOKEN\_REVOKE\_INTENT イベントリスナーは、SalesforceActivity.java、SalesforceListActivity.java、SalesforceExpandableListActivity.java、および SalesforceDroidGapActivity.java で実装されます。更新トークンが取り消されると、これらのリスナーによりユーザーがログインページに自動的にログアウトされます。このことは、トーストメッセージによりユーザーに通知されます。

## 接続アプリケーション

接続アプリケーションは、API を使用して Salesforce と統合します。接続アプリケーションは、標準の SAML および OAuth プロトコルを使用して認証し、シングルサインオンと Salesforce API で使用するトークンを提供します。接続アプリケーションでは、標準の OAuth 機能に加え、Salesforce システム管理者がさまざまなセキュリティポリシーを設定したり、対応するアプリケーションを使用できるユーザーを明示的に制御したりすることができます。

接続アプリケーションの作成時に指定する情報の一般的なリストを次に示します。

- 名前、説明、ロゴ、連絡先情報
- Salesforce がアプリケーションを見つけて認証または識別できるようにするための URL
- 認証プロトコル: OAuth、SAML、またはその両方
- IP 範囲 (省略可能)。この範囲のユーザーが接続アプリケーションにログインできます



- 接続アプリケーションが適用できる、モバイルポリシーに関する情報(省略可能)

最も基本的なレベルでは、Salesforce Mobile SDK アプリケーションは、接続アプリケーションを使用して Salesforce OAuth サービスにアクセスし、Salesforce REST API へのアクセスを有効にします。

## PIN セキュリティについて

Mobile SDK バージョン 9.2 以降では、アプリケーション固有の PIN の代わりにモバイルデバイスの PIN を使用してパスコードフローを簡略化しています。

- iOS の場合は、「[ログインとパスコードについて](#)」(ページ 76)を参照してください。
- Android の場合は、「[パスコードの使用](#)」(ページ 154)を参照してください。

## 生体認証

Mobile SDK 11.0 以降では、デバイスシステムの生体認証を使用して「ログイン」するようにアプリケーションを設定できます。たとえば、アプリケーションがバックグラウンドでタイムアウト期間を超過した場合、ユーザーがアプリケーションに戻ったときにログイン画面が表示されます。ユーザーのログインセッションが期限切れになっていなくても、この動作によりユーザーがログアウトしたように見えます。その後、ユーザーはユーザー名とパスワードまたはデバイスの生体認証(このオプションを有効にしている場合)を使用して「ログイン」できます。

 **メモ:** 代わりに、ログイン後にパスコードを要求するようにアプリケーションを設定して、同様の効果を実現することもできます。この機能はロック画面として表示されますが、Mobile SDK 11.0 で導入された生体認証ではログイン画面が表示されます。関連情報:

- [ログインとパスコードについて](#) (ページ 76)
- [パスコードの使用](#) (ページ 154)

## 接続アプリケーション設定

生体認証を設定するには、組織の接続アプリケーションに移動します。

1. キー `ENABLE_BIOMETRIC_AUTHENTICATION` を使用してカスタム属性を接続アプリケーションに追加します。
2. タイムアウト期間を変更するには、`BIOMETRIC_AUTHENTICATION_TIMEOUT` キーを追加します。この値を調整して、ロックされるまでアプリケーションをバックグラウンドで実行する時間(分)を設定できます。このキーを追加しない場合、デフォルト値の 15 分が設定されます。

 **メモ:** アプリケーションがロックされている間、ユーザーの認証トークンは更新されません。システム管理者が接続アプリケーションでセッションタイムアウトを有効にし、[次の時間が経過したらアプリケーションをロック] 属性を同じ値に設定することをお勧めします。

## API とカスタマイズ

アプリケーション内から生体認証オプトインを管理するには、`SalesforceSDKManager` を使用して `BiometricAuthenticationManager` インスタンスを取得します。このインスタンスを使用して、次のことができます。

- ユーザーが生体認証にオプトインしたかどうかを確認する。
- 生体認証のオプトインまたはオプトアウトを行うようにユーザーを促して、その応答で SDK を更新する。
- アプリケーションがロックされているかどうかを確認する。
- アプリケーションをすぐにロックする。
- ネイティブの生体認証ロック解除ボタンを完全に無効にする。

ユーザーを生体認証にオプトインするには、Mobile SDK で提供されるプロンプトを使用するか、独自のプロンプトを作成します。

Mobile SDK で提供されるプロンプトを使用するには、次のいずれかの行を実装します。

### Android (Kotlin)

```
SalesforceSDKManager.getInstance().biometricAuthenticationManager.presentOptInDialog(fragmentManager)
```

### iOS (Swift)

```
SalesforceManager.shared.biometricAuthenticationManager().presentOptInDialog(viewController: viewController)
```

カスタムプロンプトを作成する場合は、次のいずれかの行を実装して、ユーザーの応答を Mobile SDK に渡します。

### Android (Kotlin)

```
SalesforceSDKManager.getInstance().biometricAuthenticationManager.biometricOptIn(userResponse)
```

### iOS (Swift)

```
SalesforceManager.shared.biometricAuthenticationManager().biometricOptIn(optIn: userResponse)
```

ユーザーが生体認証を有効にすると、ネイティブのボタンがログイン画面に追加され、OS プロンプトをユーザーがトリガーできるようになります。

アプリケーションの Web ビュー内にカスタムボタンを作成するには、`BiometricAuthenticationManager` にある API を使用してネイティブのボタンを無効にすることができます。次に、`mobilesdk://biometric/authentication/prompt` にリダイレクトするようにボタンを設定します。Mobile SDK では、このリダイレクトが自動的に無視され、ネイティブの OS プロンプトが表示されます。

## OAuth 2.0 サイトおよび Salesforce サイトを使用したポータル認証

Salesforce Spring '13 リリースでは、ポータル認証の柔軟性が強化されました。アプリケーションが Salesforce ポータルで実行されている場合、Salesforce サイトで OAuth 2.0 を使用して、ポータルユーザーの代わりに API アクセストークンを取得できます。この設定では、次の操作を実行できます。

- SOAP API `login()` コールではなく、認証プロバイダーおよび SAML を使用してポータルユーザーを認証する。
- アプリケーション内でのユーザーのログイン情報の処理を回避する。
- Salesforce サイトで提供されるログイン画面をカスタマイズする。

開始方法は次のとおりです。

1. Salesforce サイトをポータルに関連付けます。サイトで、ポータルに一意の URL が生成されます。「[ポータルと Salesforce サイトの関連付け](#)」を参照してください。
2. Salesforce サイトで、カスタムログインページを作成します。「[Salesforce サイトのログインおよび登録設定の管理](#)」を参照してください。
3. サイトで生成された一意の URL を、ユーザーのログイン要求のリダイレクトドメインとして使用します。

OAuth2.0 サービスでカスタムホスト名が認識され、ユーザーがまだ認証されていない場合はサイトのログインページにリダイレクトされます。

 **例:** たとえば、次のように `https://login.salesforce.com` にはリダイレクトしません。

```
https://login.salesforce.com/services/oauth2/authorize?
response_type=code&client_id=<your_client_id>&
redirect_uri=<your_redirect_uri>
```

次のように、`https://MyDomainName.my.salesforce-sites.com` などの一意の Salesforce サイト URL にリダイレクトします。

```
https://MyDomainName.my.salesforce-sites.com/services/oauth2/authorize?
response_type=code&client_id=<your_client_id>&
redirect_uri=<your_redirect_uri>
```

詳細およびデモビデオについては、Salesforce Platform 開発者リレーションブログページの「[OAuth for Portal Users \(ポータルユーザーの OAuth\)](#)」を参照してください。

## Salesforce Mobile SDK アプリケーションでの MDM の使用

モバイルデバイス管理 (MDM) では、アプリケーションの設定、更新、および認証を容易に行うことができます。Salesforce および Mobile SDK では、接続アプリケーションに MDM を使用できます。

MDM を使用するには、Salesforce システム管理者および MDM プロバイダーと協力します。Salesforce システム管理者は、使用事例に合わせて接続アプリケーションを設定します。MDM プロバイダーは、モバイルアプリケーション設定を顧客のデバイスに配布する、信頼できるサードパーティです。たとえば、MDM を使用して、アプリケーションのカスタムログイン URL を設定できます。また、MDM を使用して、証明書ベースの認証を行うこともできます。この場合、証明書を MDM プロバイダーにアップロードします。

MDM を有効にするために、Mobile SDK アプリケーションコードを変更する必要はありません。

次に、基本的な MDM ランタイムフローの概要を示します。

## 認証および設定のランタイムフロー

- 顧客が MDM 対応 Mobile SDK アプリケーションをダウンロードするには、まず MDM プロバイダーのアプリケーションをインストールします。
- MDM プロバイダーは、そのアプリケーションを使用して、次の項目をデバイスに転送します。
  - Mobile SDK アプリケーション
  - 指定した設定の詳細 (カスタムログイン URL や高度なセキュリティ設定など)
  - ユーザー証明書 (認証にも MDM を使用している場合)
- 顧客がアプリケーションを起動したときの動作は、モバイルオペレーティングシステムによって異なります。
  - Android:** 証明書ベースの認証をサポートしている場合、ログインサーバーが証明書を要求します。Android は、Web ビューを起動して、顧客が選択できる 1 つ以上の証明書のリストを提供します。
  - iOS:** Mobile SDK アプリケーションは、Salesforce 接続アプリケーション定義で証明書ベースの認証が有効になっているかどうかをチェックします。有効になっている場合、アプリケーションは Safari ウィンドウに移動します。Safari は、保存されている MDM 証明書を取得し、透過的にデバイスを認証します。
- 証明書を受け入れたら、ログインサーバーがアクセストークンおよび更新トークンをアプリケーションに送信します。
- Salesforce は、顧客のデータへのアクセスを要求する標準画面を表示します。

次のセクションでは、Mobile SDK でサポートされる MDM 設定オプションについて説明します。

## 証明書ベースの認証

証明書を使用した認証により、ユーザー名とパスワードを排除して、モバイルユーザーのプロビジョニングと日常のモバイル管理業務を簡略化します。Salesforce では、ユーザー認証をさらに効率的に行うために X.509 証明書を使用するか、ログインプロセスの 2 つ目の要素として使用します。

### 証明書ベースの認証の MDM 設定

モバイルユーザーの証明書ベースの認証を有効にするには、MDM スイッチを介してキー - 値ペアの割り当てを設定する必要があります。サポートされているキーは、次のとおりです。

| キー              | 型       | プラットフォーム    | 説明                                                                                                                                                |
|-----------------|---------|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| RequireCertAuth | Boolean | Android、iOS | <p>true の場合、証明書ベースの認証フローが開始されます。</p> <p><b>Android:</b> Web ビュー内の認証でデバイスのユーザー証明書を 사용합니다。</p> <p><b>iOS:</b> すべての認証要求でユーザーを Safari にリダイレクトします。</p> |

| キー                  | 型      | プラットフォーム | 説明                                                             |
|---------------------|--------|----------|----------------------------------------------------------------|
| ManagedAppCertAlias | String | Android  | ユーザー認証のアプリケーションで選択されたデバイスにリリースされる証明書の別名。Android の場合にのみ必要になります。 |

 **メモ:** 証明書ベースの認証を使用するためのデバイスの OS の最小バージョン要件があります。Android の場合、サポートされる最小バージョンは 5.0 です。iOS の場合、サポートされる最小バージョンは 7.0 です。

キー - 値ペアの割り当てを保存すると、更新された証明書ベースの認証フローを使用するモバイルアプリケーションを MDM スイートを介してユーザーに転送できます。

## 自動カスタムホストプロビジョニング

カスタムログインホストの設定をモバイルユーザーに転送できるようになりました。これにより、モバイルユーザーは、ストレスやミスの原因となりやすいログインホストの長い URL を手動で入力する必要がなくなります。モバイルユーザーに複数のカスタムログインホストを定義するように、MDM を介してキー - 値ペアの割り当てを設定できます。

### 自動カスタムホストプロビジョニングの MDM 設定

カスタムログインホスト設定をモバイルユーザーに転送するには、MDM スイートを介してキー - 値ペアの割り当てを設定する必要があります。サポートされているキーは、次のとおりです。

| キー                   | データ型                | プラットフォーム    | 説明                                                                                                                              |
|----------------------|---------------------|-------------|---------------------------------------------------------------------------------------------------------------------------------|
| AppServiceHosts      | String、String Array | Android、iOS | ログインホスト。配列の最初の値がデフォルトのホストです。<br><br><b>Android:</b> ホストの URL に https:// が必要です。<br><br><b>iOS:</b> ホストの URL に https:// は必要ありません。 |
| AppServiceHostLabels | String、String Array | Android、iOS | ホストの表示ラベル。<br><br>AppServiceHostLabels エントリの数は、AppServiceHosts エントリの数と一致している必要があります。                                            |

| キー                      | データ型    | プラットフォーム    | 説明                                                            |
|-------------------------|---------|-------------|---------------------------------------------------------------|
| OnlyShowAuthorizedHosts | Boolean | Android、iOS | true の場合、ユーザーは Salesforce モバイルアプリケーションから接続できるホストのリストを変更できません。 |

## その他のセキュリティの機能強化

モバイルアプリケーションがバックグラウンドで実行されているときは常にクリップボードのコンテンツをクリアすることで、iOS ユーザーのセキュリティ層をさらに追加できます。ユーザーは、日々の業務の一環として機密データをコピーして貼り付けることがありますが、この機能強化により、バックグラウンドでアプリケーションが実行されているときは常にクリップボードにコピーするデータが確実にクリアされます。

### その他のセキュリティの機能強化の MDM 設定

モバイルアプリケーションがバックグラウンドで実行されているときに iOS ユーザーのクリップボードをクリアするには、MDM スイートを介してキー - 値ペアの割り当てを設定する必要があります。サポートされているキーは、次のとおりです。

| キー                         | データ型    | プラットフォーム | 説明                                                                                                                          |
|----------------------------|---------|----------|-----------------------------------------------------------------------------------------------------------------------------|
| ClearClipboardOnBackground | Boolean | iOS      | true の場合、モバイルアプリケーションがバックグラウンドで実行されているときに iOS クリップボードのコンテンツがクリアされます。これにより、ユーザーが誤ってアプリケーションの外部に機密データをコピーして貼り付けてしまうことを回避できます。 |

 **メモ:** モバイルアプリケーションの動作が予期せず停止すると、コピーされたデータがクリップボードに残る可能性があります。ユーザーがモバイルアプリケーションを開始して、そのモバイルアプリケーションがバックグラウンドで実行されると、クリップボードのコンテンツがクリアされます。

このセキュリティ機能は、OS 5.0 以降が実行されていて、Android for Work が設定されている Android デバイスの Android で使用できます。MDM プロバイダーに連絡して、Android ユーザーにこの機能を設定してください。

## サンプルプロパティリスト設定

 **メモ:** plist を使用したキー - 値ペアの割り当ての設定は、iOS でのみ行うことができます。



キー-値ペアの割り当てを設定する1つの方法は、XMLプロパティリスト (plist) を使用することです。plist には、MDM プロバイダーがセキュリティ設定を適用するためにモバイルアプリケーションに送信するキー-値ペアの割り当てが含まれます。

plist のサンプルを次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
 <key>AppServiceHosts</key>
 <array>
 <string>host1</string>
 <string>host2</string>
 </array>
 <key>AppServiceHostLabels</key>
 <array>
 <string>Production</string>
 <string>Sandbox</string>
 </array>
 <key>RequireCertAuth</key>
 <true/>
 <key>ClearClipboardOnBackground</key>
 <false/>
 <key>OnlyShowAuthorizedHosts</key>
 <false/>
</dict>
</plist>
```

## 高度な認証の使用

デフォルトでは、Mobile SDK は自動的に標準認証を使用します。ただし、サーバー側では、Salesforce 組織が [私のドメイン] や MDM を設定して、高度な認証を使用することを選択できます。高度な認証を使用するには、大部分の Mobile SDK アプリケーションで若干の設定が必要です。

## Mobile SDK で使用される認証の種別

実行時、Mobile SDK の認証種別は、ログイン組織の設定に基づいて決定されます。

- 組織の [私のドメイン] にブラウザーベースの認証が設定されている場合、Mobile SDK は [私のドメイン] 設定に従います。
- 組織の [私のドメイン] にブラウザーベースの認証が設定されていない場合、Mobile SDK は、組織で MDM 証明書ベースの認証が使用されている場合に限り高度な認証を使用します。
- 組織で [私のドメイン] のブラウザーベースの認証も MDM 証明書ベースの認証も使用されていない場合、Mobile SDK は標準認証を使用します。



## iOS での高度な認証のユーザーフロー

ブラウザーベースの認証の場合は、顧客が使い慣れた Salesforce Web ビューを通じてログインした後に認証画面が表示されます。

証明書ベースの認証の場合は、ログイン画面を表示する代わりに、オペレーティングシステムは顧客に認証用の証明書を選択するよう促します。顧客がログイン情報を入力することはありません。顧客が証明書を選択すると、認証画面が表示されます。

## 開発要件

高度な認証を実装するための Mobile SDK の要件は、最小限のものです。ほとんどの場合、アプリケーションで必要な設定は、ごくわずかです。Android アプリケーションで MDM 証明書ベースの認証が使用されている場合、クライアント側の設定は不要です。

## 接続アプリケーションの設定

- Salesforce 接続アプリケーションの API (OAuth 設定の有効化) で、次の操作を行います。
  - Mobile SDK アプリケーションの一般的な OAuth 設定を適用する。「[API \(OAuth 設定の有効化\)](#)」を参照してください。
  - [Web サーバーフローの秘密が必要] が選択されていないことを確認する。

## [私のドメイン] 設定の構成

組織のシステム管理者は、[私のドメイン] 設定を通じて高度な認証を要求できます。高度な認証を利用する手順は、次のとおりです。

1. [設定] から、[クイック検索] ボックスに「私のドメイン」と入力し、[私のドメイン] を選択します。
2. [私のドメイン] 設定の [認証設定] で、システム管理者が次のオプションの一方または両方を選択します。
  - Android でのユーザー認証にネイティブブラウザーを使用
  - iOS でのユーザー認証にネイティブブラウザーを使用

Salesforce ヘルプの「[モバイル認証方法を使用した \[私のドメイン\] ログインページのカスタマイズ](#)」を参照してください。

## MDM 設定の構成

ログインが MDM で管理されている場合、Mobile SDK が高度な認証を使用するのは、組織の MDM 設定で証明書ベースの認証が指定されている場合に限られます。組織の MDM スイートでは、次の操作を行う必要があります。

- RequireCertAuth プロパティを true に設定する。
- Android のみ: ManagedAppCertAlias プロパティを別名に設定する。

## 高度な認証を使用したログインセッション管理

高度な認証を使用する場合、アプリケーションからログアウトすると、意外な動作が発生することがあります。この動作がアプリケーションに及ぼす影響は、アプリケーションで使用するログインの種類によって異なります。

### 証明書ベース (MDM) のログイン

証明書を使用すると、その証明書が取り消されるまで顧客は認証済みのままです。証明書は、発行者が取り消すまで有効です。証明書が有効なうちに顧客がアプリケーションからログアウトすると、Salesforce のログイン画面が一時的に表示されます。ただし、証明書が自動的に顧客のログイン情報を提供するため、フローは直接、認証 ([アクセスを許可]) 画面に移動します。[許可] を選択すると、顧客はアクセストークンと更新トークンを取得し、アプリケーションの使用を継続できます。つまり、MDM 発行者が証明書を取り消すまで顧客はログアウトできません。

### Web サーバー OAuth ログイン

OAuth 2.0 認証時に、Salesforce はログインと Salesforce 認証 ([アクセスを許可]) 画面の間のギャップを埋めるために一時的な短期セッションを作成します。この一時的なセッションは Cookie を使用し、OAuth の更新トークンやアクセストークンに関連付けられていないため、ログアウト時に無効になりません。セッションは有効期限が切れるまで有効なままです。最後に認証された顧客は、一時的なセッションの有効期限が切れるまでログインしたままになります。これらのセッションは意図的に有効期間が短くなっており、その後にはユーザーは通常どおりログインできます。

一時的なセッションの間、次に示す予期しない動作が発生する場合があります。顧客が Cookie の有効期限が切れる前にログアウトしてもう一度ログインしようとする、フローがログイン画面をスキップして、直接 Salesforce 認証に進みます。[許可] を選択すると、顧客はアクセストークンと更新トークンを自動的に取得し、アプリケーションの使用を継続できます。

この動作は標準 Web ビュー認証では発生しません。これは、Web ビューは以前の認証の Cookie を保持しないためです。また、一時的なセッションの有効期限が切れた後に顧客がログアウトした場合にも発生しません。Mobile SDK アプリケーション (Salesforce アプリケーションを含む) は、Salesforce サービスからの Cookie を制御できません。

**❗ 重要:** 高度な認証はスウィズルを行いませんが、ログインページはフルスクリーンのままです。こうした表示により、顧客は一時的にアプリケーションから別のところに移動したと思うかもしれません。

## 関連トピック

- [Salesforce Mobile SDK アプリケーションでの MDM の使用](#)。
- サーバー側の [私のドメイン] 設定についての詳細は、Salesforce ヘルプの「[ブランドを使用した \[私のドメイン\] ログインページのカスタマイズ](#)」を参照してください。
- 接続アプリケーションについての詳細は、Salesforce ヘルプの「[接続アプリケーションの作成](#)」を参照してください。
- MDM 設定についての詳細は、『Salesforce モバイルアプリケーションセキュリティガイド』の「[モバイルデバイス管理 \(MDM\)](#)」を参照してください。
- iOS URL スキームの設定についての詳細は、『[iOS アプリケーションプログラミングガイド](#)』(developer.apple.com/documentation/) の「[アプリケーション間通信](#)」または「[カスタム URL スキーム](#)」を参照してください。

このセクションの内容:

### [iOS アプリケーションでの高度な認証の設定](#)

高度な認証をサポートするには、すべての iOS アプリケーションでいくつかのカスタム設定が必要です。

### [Android アプリケーションでの高度な認証の設定](#)

Salesforce 組織が高度な認証のために [私のドメイン] を使用している場合、Mobile SDK では、クライアントアプリケーションの設定を若干行う必要があります。Android アプリケーションで証明書ベースの認証が使用されている場合、Mobile SDK アプリケーション内の設定は不要です。

## iOS アプリケーションでの高度な認証の設定

高度な認証をサポートするには、すべての iOS アプリケーションでいくつかのカスタム設定が必要です。

iOS の高度な認証では、現在の Mobile SDK リリースでサポートされる最新の iOS テクノロジーが使用されます。

## 標準認証と高度な認証の比較

iOS の標準認証と高度な認証の違いの一部を次に挙げます。これらの違いは、ブラウザーベースの認証[私のドメイン]に固有のものです。

- **標準認証フロー:** このフローは `WKWebView` を使用します。このクラスは、iOS ビューツールバーをはじめとする強力な機能にアクセスでき、ユーザーエクスペリエンスが優れています。
- **高度な認証フロー:** 高度な認証では、現在の Mobile SDK リリースでサポートされる最新の iOS テクノロジーが使用されます。高度な認証は安全性の高いオプションです。アプリケーションが顧客の同意なしに Cookie を設定したり、コンテンツをビューに挿入したりすることができません。高度なモードでは、認証フローがスウィズルを行いません。

## アプリケーション設定

iOS アプリケーションでのステップは、MDM 証明書ベースおよびブラウザーベースの両方の方法の場合と同じです。すべての組織との互換性を保証するために、次の手順を実行します。

- OAuth リダイレクト URI のカスタム URL スキームを、プロジェクトの `Info.plist` ファイルに追加します。
  1. アプリケーションの `Info.plist` ファイルで、`CFBundleURLTypes` という名前のキーを作成します。
  2. このキーを、次のキーがある辞書を含む配列に割り当てます。

| 名前                              | 型          | 値                                                                            |
|---------------------------------|------------|------------------------------------------------------------------------------|
| <code>CFBundleURLName</code>    | String     | URL スキームの一意の抽象名 (逆引き DNS スタイルの識別子を推奨)。次に例を示します。 <div>com.acme.myscheme</div> |
| <code>CFBundleURLSchemes</code> | String の配列 | URL スキーム名 (http、mailto など)。                                                  |

 **例:** OAuth コールバック URI が `com.mydomain.myapp://oauth/success` の場合は、`Info.plist` ファイルに次のキーを追加します。

```
<key>CFBundleURLTypes</key>
<array>
 <dict>
 <key>CFBundleURLName</key>
 <string>com.mydomain.myapp</string>
 <key>CFBundleURLSchemes</key>
 <array>
 <string>com.mydomain.myapp</string>
 </array>
 </dict>
</array>
```

この例では URL スキームと URL 名が同じですが、必ずしも一致している必要はありません。アプリケーションで必要なだけのスキームを追加できます。

## 関連トピック

- サーバー側の [私のドメイン] 設定についての詳細は、Salesforce ヘルプの「[ブランドを使用した \[私のドメイン\] ログインページのカスタマイズ](#)」を参照してください。
- MDM 設定についての詳細は、『Salesforce モバイルアプリケーションセキュリティガイド』の「[モバイルデバイス管理 \(MDM\)](#)」を参照してください。
- iOS URL スキームの設定についての詳細は、『[iOS アプリケーションプログラミングガイド](#)』([developer.apple.com/documentation/](https://developer.apple.com/documentation/))の「[アプリケーション間通信](#)」または「[カスタム URL スキーム](#)」を参照してください。

## Android アプリケーションでの高度な認証の設定

Salesforce 組織が高度な認証のために [私のドメイン] を使用している場合、Mobile SDK では、クライアントアプリケーションの設定を若干行う必要があります。Android アプリケーションで証明書ベースの認証が使用されている場合、Mobile SDK アプリケーション内の設定は不要です。

## Android 実装

Android アプリケーションで高度な認証をサポートするため、Mobile SDK は Chrome カスタムタブを使用します。実行時に Chrome を使用できない場合、Mobile SDK はデフォルトのシステムブラウザを使用します。ブラウザベース認証では、次のことが求められます。

- ブラウザーがデバイスにインストールされている。
- MDM を使用している場合は、ブラウザが作業パーティションにインストールされている。

必要に応じて、次のメソッドを使用してアプリケーションで選択するブラウザを設定できます。

```
SalesforceSDKManager.getInstance().setCustomTabBrowser(browserPackage);
```

現在選択されているカスタムタブブラウザを表示するには、次のメソッドを使用します。

```
browserPackage = SalesforceSDKManager.getInstance().getCustomTabBrowser();
```

## 証明書ベースのアプリケーション設定

証明書ベースの認証では、MDMベンダーが利用されます。このベンダーは、Salesforceとクライアントモバイルデバイス間で識別サービスを仲介します。証明書ベースの認証では、Mobile SDK Android プロジェクトの設定は不要です。

## ブラウザベースのアプリケーション設定

1. Android Studio で、アプリケーションの `AndroidManifest.xml` ファイルを開きます。
2. `LoginActivity` 宣言で、次の行のコメントを解除します。

```
<activity android:name="com.salesforce.androidsdk.ui.LoginActivity"
 android:theme="@style/SalesforceSDK.ActionBarTheme"
 android:launchMode="singleInstance">

 <!--
 <intent-filter>
 <data android:scheme="testsfdc"
 android:host="*"
 android:path="/mobilesdk/detect/oauth/done" />
 <action android:name="android.intent.action.VIEW" />
 <category android:name="android.intent.category.BROWSABLE" />
 <category android:name="android.intent.category.DEFAULT" />
 </intent-filter>
 -->
</activity>
```

3. `android:scheme`、`android:host`、および `android:path` の値を、接続アプリケーションの対応する値に置き換えます。次は、その例です。

接続アプリケーションのコールバック URL が `testsfdc:///mobilesdk/detect/oauth/done` の場合:

- `android:scheme` は `testsfdc` です。
- `android:host` は `*`、つまり存在しません。
- `android:path` は `/mobilesdk/detect/oauth/done` です。

接続アプリケーションのコールバック URL が `sfdc://login.salesforce.com/oauth/done` の場合:

- `android:scheme` は `sfdc` です。
- `android:host` は `login.salesforce.com` です。
- `android:path` は `/oauth/done` です。

`testsfdc:///mobilesdk/detect/oauth/done` スキームを使用した `AndroidManifest.xml` の更新された部分を次に示します。

```
<!-- Login activity -->
<!--
```

```
Launch mode of "singleInstance" ensures that the activity isn't restarted
by a callback from Chrome custom tab when auth flow is complete. This is
required for the Chrome custom tab auth flow to work correctly.
-->

<!--
To enable browser bath authentication, uncomment the lines below and replace
'scheme', 'host' and 'path' with their corresponding values from your connected app.

For example, if the callback URL of your connected app is
"testsfdc:///mobilesdk/detect/oauth/done",
'scheme' would be "testsfdc", 'host' would be "*" since it doesn't exist, and
'path' would be "/mobilesdk/detect/oauth/done".

If the callback URL is "sfdc://login.salesforce.com/oauth/done",
'scheme' would be "sfdc", 'host' would be "login.salesforce.com",
and 'path' would be "/oauth/done".
-->
<activity android:name="com.salesforce.androidsdk.ui.LoginActivity"
 android:theme="@style/SalesforceSDK.ActionBarTheme"
 android:launchMode="singleInstance">
 <intent-filter>
 <data android:scheme="testsfdc"
 android:host="*"
 android:path="/mobilesdk/detect/oauth/done" />
 <action android:name="android.intent.action.VIEW" />
 <category android:name="android.intent.category.BROWSABLE" />
 <category android:name="android.intent.category.DEFAULT" />
 </intent-filter>
</activity>
```

これですべて完了です。

## 関連トピック

- [Salesforce ヘルプの「ブランドを使用した \[私のドメイン\] ログインページのカスタマイズ」](#)。
- 『Salesforce モバイルアプリケーションセキュリティガイド』の「モバイルデバイス管理 (MDM)」。
- iOS URL スキームの設定についての詳細は、『iOS アプリケーションプログラミングガイド』([developer.apple.com/documentation/](https://developer.apple.com/documentation/))の「アプリケーション間通信」または「カスタム URL スキーム」を参照してください。

## Google ログイン要件に対応するための Android シングルサイン オンアプリケーションのアップグレード

2018 年、Google は、ブラウザーベースのログインフローを採用するため、埋め込み Web ビューを介したログインのサポートを正式に打ち切りました。このため、SSOID プロバイダーとして Google を使用している古い Mobile SDK Android アプリケーションでは、コードの変更が必要になる場合があります。



ブラウザーベースの認証は、「高度な認証」とも呼ばれます。古いアプリケーションをアップグレードするには、「[iOS アプリケーションでの高度な認証の設定](#)」の説明に従って、高度な認証を実装してください。

Mobile SDK では、アプリケーション内の Chrome カスタムタブによる高度な認証をサポートしています。実行時に Chrome を使用できない場合、Mobile SDK はデフォルトのシステムブラウザーを使用します。ブラウザーベース認証では、次のことが求められます。

- ブラウザーがデバイスにインストールされている。
- MDM を使用している場合は、ブラウザーが作業パーティションにインストールされている。

## 関連トピック

- [Chrome Custom Tabs \(Chrome カスタムタブ\)](#)
- [Google の開発者ブログ](#)

## 外部サービスにアクセスするための OpenID トークンの使用

---

Salesforce の外部にある承認済みサービスが Mobile SDK アプリケーションで必要な場合は、OpenID トークンを使用して、必要な認証ハンドシェイクを実行できます。

ほとんどの Mobile SDK アプリケーションは、Salesforce に対して認証を行い、Mobile SDK REST API ラッパーを使用して Salesforce リソースにアクセスします。このシナリオの場合、Mobile SDK は、アプリケーションの明示的な関与なしに、バックグラウンドで認証トークンの交換を処理します。

ただし、アプリケーションによっては、Salesforce トークンを受け入れない認証済みサービスからのデータが必要なものもあります。これらのサービスは、Quip、Heroku、承認済みサードパーティなどの Salesforce が所有する商品から提供できます。これらの外部 API コールを Salesforce 環境から行うには、OpenID トークンを使用します。

## OpenID トークン

OpenID トークンにより、トークンを生成するアプリケーションは、情報を外部 Web サービスと共有できます。Mobile SDK のために、OpenID トークンは、ユーザーとアプリケーションの ID を共有します。このトークンを受信する外部サービスは、ユーザーの外部ログイン情報のフルセットをアプリケーションに提供できます。通常、Salesforce で提供される OpenID トークンは、有効期限が短いため、セキュリティ違反が発生する可能性は限定的です。

OpenID サポートでは、Salesforce サーバーおよび Mobile SDK アプリケーションでの設定が必要です。Salesforce 側で、組織のシステム管理者は、OpenID トークンをサポートするように接続アプリケーションを設定します。Mobile SDK アプリケーションで、開発者は、アプリケーションの OAuth 範囲を設定し、OpenID トークンを提供する Mobile SDK メソッドをコールします。アプリケーションは、このトークンを外部サービスからのログイン情報のフルセットと交換できます。アプリケーションは、使用する外部ログイン情報を管理する責任があります。

OpenID についての詳細は、「[openid.net/what-is-openid](https://openid.net/what-is-openid) ([openid.net/OpenID](https://openid.net/OpenID) とは)」を参照してください。



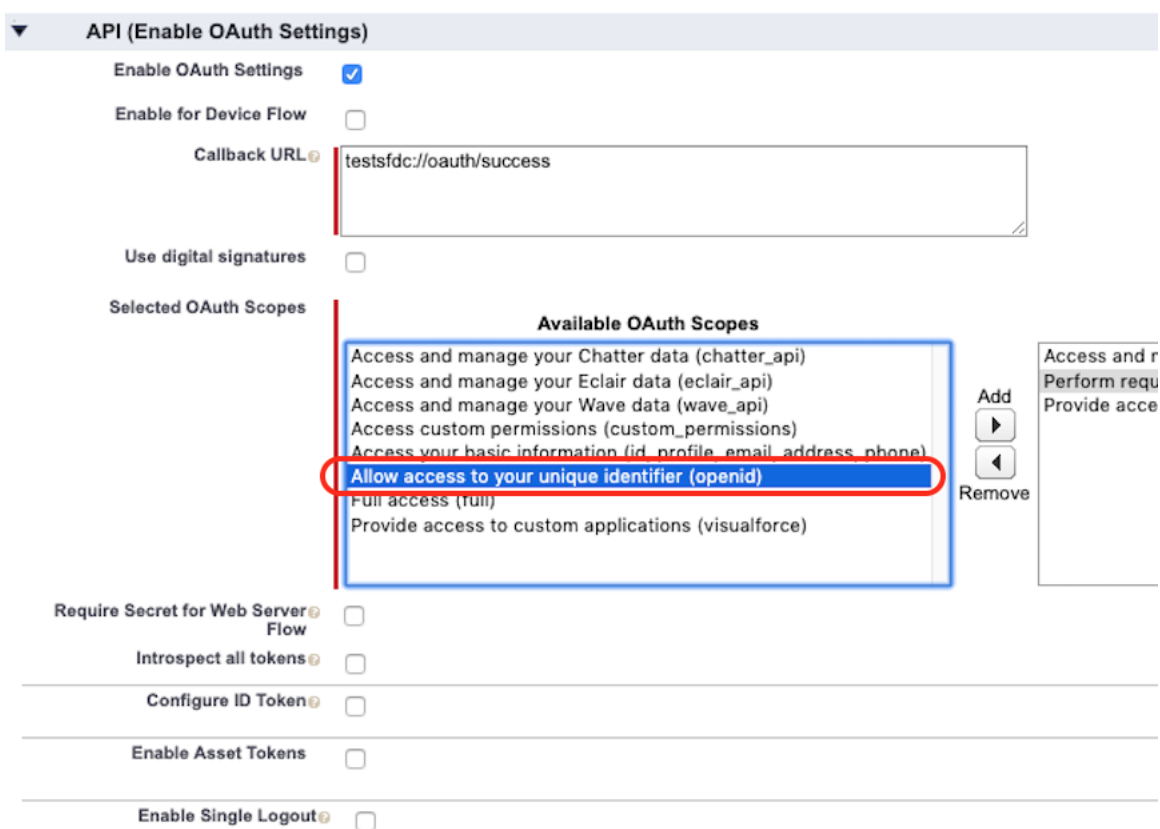
## サーバー側設定の構成

新規または既存の接続アプリケーションを編集するときの API (OAuth 設定の有効化) の下の接続アプリケーションの設定。

1. [OAuth 設定の有効化] を選択します。



2. [選択した OAuth 範囲] の下で [一意の識別子へのアクセスを許可 (openid)] を選択し、[追加] をクリックします。



3. Salesforce ヘルプの「[接続アプリケーションの作成](#)」の説明に従って、[ID トークンを設定] を選択し、そのサブ設定を構成します。

▼ API (Enable OAuth Settings)

Enable OAuth Settings☒

Enable for Device Flow☐

Callback URL

Use digital signatures☐

Selected OAuth Scopes

Available OAuth Scopes

Access and manage your Chatter data (chatter\_api)

Access and manage your Eclair data (eclair\_api)

Access and manage your Wave data (wave\_api)

Access custom permissions (custom\_permissions)

Access your basic information (id, profile, email, address, phone)

Full access (full)

Provide access to custom applications (visualforce)

Add

Remove

Require Secret for Web Server Flow☐

Introspect all tokens☐

Configure ID Token☒

Token Valid for Minutes

ID Token Audiences

☒ Include Standard Claims

☒ Include Custom Attributes

☒ Include Custom Permissions

Enable Asset Tokens☐

アプリケーション設定

1. bootconfig.xml ファイル (Android) または bootconfig.plist ファイル (iOS) で、openid を oauthScopes リストに追加します。

iOS:

| Key                     | Type       | Value          |
|-------------------------|------------|----------------|
| ▼ Root                  | Dictionary | (4 items)      |
| remoteAccessConsumerKey | String     | 3MVG9lu66F     |
| oauthRedirectURI        | String     | testsfdc:///mc |
| ▼ oauthScopes           | Array      | (3 items)      |
| Item 0                  | String     | web            |
| Item 1                  | String     | api            |
| Item 2                  | String     | openid         |
| shouldAuthenticate      | Boolean    | YES            |

Android:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
 <string name="remoteAccessConsumerKey">3MVG9Iu66FKeHhINkB1
 <string name="oauthRedirectURI">testsfdc:///mobilesdk/dete
 <string-array name="oauthScopes">
 <item>api</item>
 <item>web</item>
 <item>openid</item>
 </string-array>
 <string name="androidPushNotificationClientId"></string>
</resources>
```

2. OpenID トークン文字列を取得するには、プラットフォーム固有の API をコールします。

#### iOS (Objective-C)

SFSDKOAuth2 クラスの次のメソッドをコールします。

```
- (void)openIDTokenForRefresh:(SFSDKOAuthTokenEndpointRequest *)endpointReq
 completion:(void (^)(NSString *))completionBlock;
```

#### Android

OAuth2 クラスの次のメソッドをコールします。

```
public static String getOpenIDToken(String loginServer, String clientId,
 String refreshToken);
```

## 関連トピック

- [Salesforce ヘルプの「接続アプリケーションの作成」](#)。

## iOS での安全な鍵の保管

顧客情報を保護するために、Mobile SDK ではユーザー ID トークンや SmartStore データベースなどの機密データを暗号化します。通常のワークフローでは、Mobile SDK が使用するデータ保護パターンは安全性が高いと考えられていますが、「ゼロデー」の脆弱性は、嚴重に保護されたスキームでも発生する可能性があります。ハッカーや悪意のある攻撃に後れを取らないように、Mobile SDK では、その鍵の暗号化が iOS でサポートされる最も高い標準に継続的にアップグレードされます。

**重要:** 可能な場合は、Equality の会社の値に一致するように、含めない用語を変更しました。顧客の実装に対する影響を回避するために、一部の用語は変更されていません。

iOS では、各種顧客データで独自の一意の鍵が使用されます。これらの鍵は、iOS キーチェーンに保存され、主鍵で暗号化されます。

iOS キーチェーン内では、主鍵はディスク上で暗号化され、アプリケーションのみがアクセスできます。主鍵は、業界標準の 256 ビットの Advanced Encryption Standard (AES) 鍵で保護されます。それでも、悪意のあるアプリケーションに漏洩するリスクはわずかに存在します。保護を強化するために、Mobile SDK には次の機能強化があります。

- Apple のハードウェアベースの Secure Enclave に主鍵を保存する。Mobile SDK アプリケーションは、Secure Enclave に保存されている非公開鍵にアクセスしません。鍵を作成したり、鍵を安全に保存したり、保護されている他の操作を実行したりする場合、Mobile SDK 自体が Secure Enclave API をコールします。アプリケーションは、暗号化されていない機密データを処理することなく、要求された出力 (暗号化された SmartStore データなど) のみを受信します。
- 256 ビットの Elliptic Curve Cryptography (ECC) 非公開鍵を使用して主鍵を保護する。256 ビットの ECC 非公開鍵は、3072 ビットの RSA 非公開鍵に相当します。ビットコインやイーサリアムなどの暗号通貨は、ECC 形式に基づいています。

アップグレードしたアプリケーションを初めてデバイスで実行すると、Mobile SDK によって自動的に主鍵が現在の暗号化レベルに変換されて、Secure Enclave に移動されます。

 **メモ:** Secure Enclave は、Apple A7 以降の A シリーズプロセッサを搭載したデバイスで使用できます。Secure Enclave がサポートされていないデバイスの場合、主鍵はキーチェーンに残ります。

## アプリケーションでの暗号化のアップグレード

Mobile SDK 9.2 では、そのデフォルトの暗号化が AES-CBC から AES-GCM に更新します。ほとんどのアプリケーションでは、Mobile SDK によってアプリケーションの変更を要求することなくこのアップグレードが処理されます。ただし、いくつかのケースでは、アプリケーション自体がマイナーアップグレードステップを実行します。

次の場合、ユーザーがアクションを実行する必要があります。

- アプリケーションで `KeyValueEncryptedFileStore` が共有ストアクラスメソッドを通じてではなく直接初期化される場合。

この場合、キーストアを初期化する前に、`KeyValueEncryptedFileStore.updateEncryption(_:_:):` をコールします。legacyKey 引数として、ストアを作成するために元々使用された鍵を渡します。アップグレードすると、鍵は Mobile SDK によって管理されるようになります。次に例を示します。

```
// Pre 9.2
let key = SFKeyStoreManager.sharedInstance().retrieveKey(withLabel: "kv_key",
 autoCreate: true)
let store = KeyValueEncryptedFileStore(parentDirectory: "directory/path",
 name: "storeName", encryptionKey: key)

// 9.2 upgrade
let key = SFKeyStoreManager.sharedInstance().retrieveKey(withLabel: "kv_key",
 autoCreate: true)
KeyValueEncryptedFileStore.updateEncryption(parentDirectory: "directory/path",
 name: "storeName", legacyKey: key)
let store = KeyValueEncryptedFileStore(parentDirectory: "directory/path",
 name: "storeName")
```

- アプリで `SFSmartStore setEncryptionKeyBlock:` Objective-C メソッドを使用している場合。

ストアの暗号化をアップグレードするには、元の鍵で引き続き `setEncryptionKeyBlock:` を使用します。Mobile SDK 9.2 以降では、暗号化のアップグレードが実行時に実行され、デフォルトの SmartStore キーが生成された新しい鍵に置き換えられます。

この自動鍵置換は推奨パスですが、自分でデフォルトの鍵を置き換えることもできます。これを行うには、指定した新しい鍵を使用して `setEncryptionKeyGenerator:` をコールします。

今後のリリースでは、`setEncryptionKeyBlock:` へのコールを削除できます。

## 関連トピック

- [Storing Keys in the Secure Enclave \(Secure Enclave への鍵の保存\)](https://developer.apple.com/documentation/security) (developer.apple.com/documentation/security)
- [Elliptic Curve Cryptography \(楕円曲線暗号\)](https://wikipedia.com) (wikipedia.com)

## Android での安全な鍵の保管

---

Mobile SDK は、ユーザー ID トークンや SmartStore データベースなどのデータを暗号化します。通常のワークフローでは、Mobile SDK が使用するデータ保護パターンは安全性が高いと考えられていますが、「ゼロデー」の脆弱性は、厳重に保護されたスキームでも発生する可能性があります。ハッカーや悪意のある攻撃に後れを取らないように、Mobile SDK では、Android でサポートされる最も高い標準でその暗号化が強化されます。

Mobile SDK では、以前から暗号化タスクのために対称鍵のペアが使用されてきました。Mobile SDK 7.1 以降では、このスキームの上に、Android Keystore を利用する別の暗号化レベルが追加されています。現在の Mobile SDK Android の要件を満たしていれば、どのデバイスでも Android Keystore をサポートできます。Keystore の実装は、ハードウェアに依存し、Android API バージョン、デバイスメーカー、およびその他の要素によって異なります。

Mobile SDK は、セキュリティを強化するために非対称の公開-非公開鍵のペアを生成してその対称鍵のペアを暗号化します。RSA-2048 暗号化を使用するこの非対称鍵のペアは、Android Keystore に保存されます。Mobile SDK は、実行時に非対称の公開鍵で対称鍵を暗号化し、暗号化した鍵を SharedPreferences ファイルに保存します。顧客データを復号化する場合、アプリケーションは Mobile SDK に対称鍵を要求します。Mobile SDK は、その鍵にアクセスするために Keystore から非対称の非公開鍵を取得し、その鍵を使用して SharedPreferences ファイルの内容を復号化します。次に、Mobile SDK は、暗号化されていない対称鍵をアプリケーションに配信します。

## アプリケーションのアップグレード

Mobile SDK は、自動的にその鍵を新しい暗号化スキームにアップグレードします。`getEncryptionKey()` の動作や使用方法は変更されません。

## 関連トピック

- [Android Keystore システム](https://developer.android.com) (developer.android.com)

## 第 15 章

# ログイン画面のカスタマイズ

トピック:

- プログラムによる iOS ログイン画面のカスタマイズ

Mobile SDK では Salesforce ログインページを制御できませんが、特定のケースでは、ログインページのカスタマイズとブランド設定を行うことができます。

## Mobile SDK を使用したログイン画面の外観のカスタマイズ

iOS では、ナビゲーションバーや設定アイコンなど、ログインページの一部のプロパティも設定できます。「プログラムによる iOS ログイン画面のカスタマイズ」を参照してください。

## Salesforce サーバーを使用したログイン画面のカスタマイズ

Salesforce Mobile SDK ではそのクライアントアプリケーションの OAuth 実装が提供されますが、ログインページの定義や制御は行われません。代わりに、Salesforce サーバーからページを要求します。Salesforce 自体が、顧客のログイン情報を収集する Web ビューを表示します。ログイン Web ビューは、Mobile SDK アプリケーションには含まれません。

サーバー側でログイン Web ビューを変更するには、ユーザーが [私のドメイン] と Experience Cloud サイトのいずれかを使用します。

この両方の機能には、ログインページのブランド設定とカスタマイズを行うための便利なユーティリティがあります。ブランド情報入りページを使用するには、Mobile SDK アプリケーションのデフォルトログイン URL を Experience Cloud サイトのカスタムドメインまたは私のドメインのサブドメインアドレスに設定します。これで、アプリケーションにカスタマイズされたログインページが表示されます。

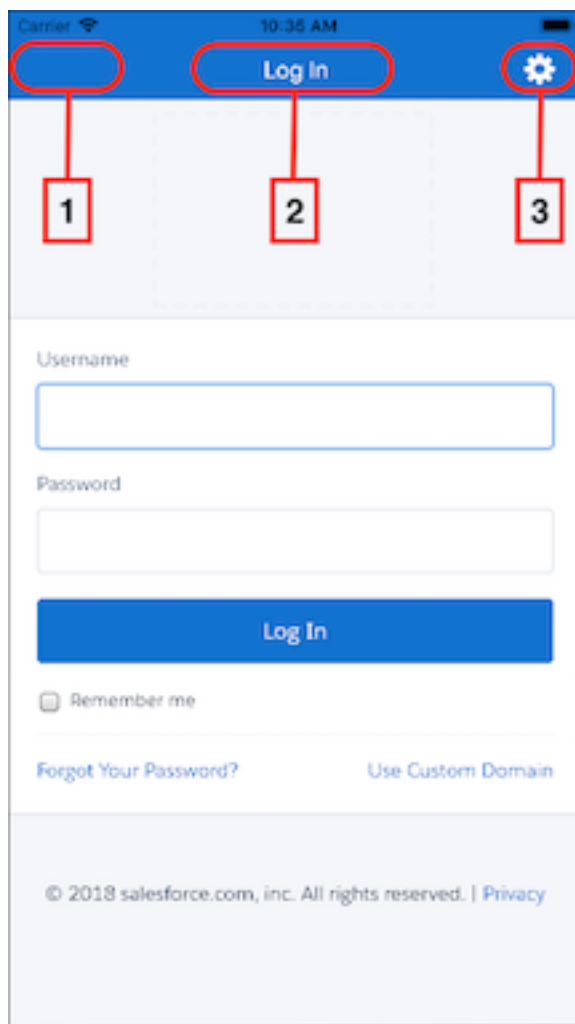
これらの機能についての詳細は、次のリンクを参照してください。

- [ブランド名が入ったログインページの使用](#)
- [コミュニティでのログイン、セルフ登録、およびパスワード管理のカスタマイズ](#) — [https://help.salesforce.com/articleView?id=networks\\_customize\\_login.htm](https://help.salesforce.com/articleView?id=networks_customize_login.htm)
- [\[私のドメイン\] とは?](#) — [https://help.salesforce.com/articleView?id=domain\\_name\\_overview.htm](https://help.salesforce.com/articleView?id=domain_name_overview.htm)
- [\[私のドメイン\] ログインページのカスタマイズ手順](#) — [https://help.salesforce.com/articleView?id=domain\\_name\\_login\\_branding.htm](https://help.salesforce.com/articleView?id=domain_name_login_branding.htm)
- [カスタマイズされた \[私のドメイン\] ログインページのサンプル](#) — <https://github.com/salesforceidentity/MyDomain-Sample>

## プログラムによる iOS ログイン画面のカスタマイズ

Mobile SDK for iOS には、ログイン画面のナビゲーションバーのスタイルと動作をカスタマイズする幅広いオプションがあります。宣言的に変更を行ってウィジェットの外観や表示を制御することも、ログインビューコントローラークラスを拡張することによってナビゲーションバーを刷新することもできます。

標準的な Salesforce ログイン画面は次のようになります。



カスタマイズできるナビゲーションバーウィジェットには、[戻る] ボタン (1) (通常は非表示)、タイトル (2)、設定アイコン (3) があります。ナビゲーションバー全体を非表示にすることもできます。デフォルトでは、ログイン画面には上部ナビゲーションバーとその埋め込み設定アイコンの両方が表示されます。

設定アイコン  は、その歯車のような形状から多くの場合「歯車」アイコンと呼ばれます。ユーザーは設定アイコンを使用して、組み込みリストからログインサーバーを選択したり、カスタムログイン URI を指定したりできます。ただし、会社によってはユーザーにログインサーバーの選択を許可しない場合があります。ログインサーバーの選択を無効にするには、設定アイコン自体を非表示にするか、ナビゲーションバーを非表示にします。ナビゲーションバーの色やそのテキストの色およびフォントをカスタマイズすることもできます。次



の `SFSDKLoginViewControllerConfig` プロパティを使用して、これらの UI 要素の表示と外観を制御します。

## showSettingsIcon

設定アイコンのみの表示を制御します。ナビゲーションバーの表示には影響しません。

### 動作

| 値           | 意味                                                           |
|-------------|--------------------------------------------------------------|
| YES (デフォルト) | デフォルト値。設定アイコンは表示され、アクセス可能です。                                 |
| NO          | 設定アイコンは表示されません。ユーザーはログインホストリストにアクセスできず、カスタムホストを追加することもできません。 |

## showNavbar

ナビゲーションバーの表示を制御します。これにより、設定アイコンの表示も制御されます。

### 動作

| 値           | 意味                                                                              |
|-------------|---------------------------------------------------------------------------------|
| YES (デフォルト) | デフォルト値。ナビゲーションバーは表示されます。設定アイコンは、 <code>showSettingsIcon</code> プロパティに応じて表示されます。 |
| NO          | ナビゲーションバーと設定アイコンは表示されません。ユーザーはログインホストリストにアクセスできず、カスタムホストを追加することもできません。          |

 **メモ:** ハイブリッドアプリケーションで歯車アイコンを非表示にするには、ネイティブラッパーアプリケーションに上記の手順を適用します。

## ナビゲーションバーの色とフォント

次のスタイルプロパティを設定できます。

- `navBarColor`
- `navBarTextColor`
- `navBarFont`

次の例は、設定アイコンの表示とナビゲーションバーのスタイルを設定する方法を示しています。ヘッダーファイルをインポートし、AppDelegate クラスの `application:didFinishLaunchingWithOptions:` メソッドに後続の行を追加します。

```
let loginViewConfig = SalesforceLoginViewControllerConfig()
loginViewConfig.showsSettingsIcon = false
loginViewConfig.showsNavigationBar = true
loginViewConfig.navigationBarColor = UIColor(red: 0.051, green: 0.765, blue: 0.733, alpha: 1.0)
loginViewConfig.navigationBarTextColor = UIColor.white
loginViewConfig.navigationBarFont = UIFont(name: "Helvetica", size: 16.0)
UserAccountManager.shared.loginViewControllerConfig = loginViewConfig
```

## SFLoginViewController の拡張によるナビゲーションバーウィジェットの上書き

ナビゲーションバーを詳細にカスタマイズするには、`SFLoginViewController` クラスを拡張し、必要なメソッドを実装します。そうすることで、次のことを実現できます。

- [戻る] ボタンの有効化 — 一部の開発者は、[戻る] ボタンを有効にして、ログインが失敗した場合のカスタマーエクスペリエンスを強化します。[戻る] ボタンを有効にした場合は、カスタムアクションを指定する必要があります。アクションをカスタマイズしない場合、MobileSDKはログイン失敗のデフォルトの動作を使用しますが、これは [戻る] ボタンを無視するものです。
- デフォルトのタイトルウィジェットの上書き — 独自のタイトルテキストを指定し、カスタムアクションを定義できます。
- デフォルトの設定アイコンの上書き — 歯車画像の代わりにカスタムアイコンを使用できます。

次に、部分的なコード例を示します。

1. `SFLoginViewController` を拡張します。

```
@interface SFLoginExtendedViewController : SFLoginViewController

@end

@implementation SFLoginExtendedViewController

- (UIBarButtonItem *)createBackButton {
 // Setup left bar button:
 // UIImage *image = [[SFSDKResourceUtils
 // imageNamed:@"globalheader-back-arrow"]
 // initWithRenderingMode:UIImageRenderingModeAlwaysTemplate];
 //
 // Return a custom UIBarButtonItem:
 // return [[UIBarButtonItem alloc]
 // initWithImage:image
 // style:UIBarButtonItemStylePlain
 // target:self action:nil];
}

- (void)handleBackButtonAction {
```

```

 [super handleBackButtonAction];
 // Add your custom code here
}

- (BOOL)shouldShowBackButton {
 // Add your custom code here.
 // Return YES to show the back button
}

- (UIBarButtonItem *)createSettingsButton {
 // Set up left bar button:
 // UIImage *image = [[SFSDKResourceUtils
 // imageNamed:@"login-window-gear"]
 // initWithRenderingMode:UIImageRenderingModeAlwaysTemplate];
 //
 // Return a custom UIBarButtonItem:
 // return [[UIBarButtonItem alloc]
 // initWithImage:image
 // style:UIBarButtonItemStylePlain
 // target:self action:nil];
}

- (UIView *)createTitleItem {
 // Setup top item.
 // Create a UIView for title
 // UIView *item = [[UIView alloc] ...];
 NSString *title = [SFSDKResourceUtils localizedString:@"MobileSyncLogin"];
 ...
 return item;
}

- (void)viewDidLoad {
 [super viewDidLoad];
 // Do any additional setup after loading the view.
}

@end

```

2. AppDelegate クラスの `application:didFinishLoadingWithOptions:` メソッドで、次のブロックを設定します。

```

...
[SFUserAccountManager sharedInstance].
 loginViewControllerConfig.loginViewControllerCreationBlock =
 ^SFLoginViewController * _Nonnull{
 SFLoginExtendedViewController *controller =
 [[SFLoginExtendedViewController alloc] init];
 return controller;
 };
[SFUserAccountManager sharedInstance].loginViewControllerConfig =

```

```
loginViewConfig;
....
```

関連トピック:

[iOS アプリケーションでのカスタムログインサーバーの設定](#)

## 第 16 章

## ID プロバイダーアプリケーション

### トピック:

- ID プロバイダー: アーキテクチャ、フロー、および接続アプリケーションの要件
- Android アーキテクチャおよびフロー
- ID プロバイダーとしての Android アプリケーションの設定
- ID プロバイダークライアントとしての Android アプリケーションの設定
- ID プロバイダーとしての iOS アプリケーションの設定
- ID プロバイダークライアントとしての iOS アプリケーションの設定
- Mobile SDK を使用しないモバイル ID プロバイダーアプリケーションの実装

ID プロバイダーは、既知のユーザーが Mobile SDK アプリケーションにログインするたびに Salesforce ログイン情報を再入力しないで済むようにするために役立ちます。さらに、以前の Mobile SDK リリースの厳格なセキュリティレベルも保持します。

ID プロバイダー設定は、2 つの主コンポーネントで構成されます。

- ID プロバイダー (IDP) は、単一モバイルデバイスで 1 人以上のユーザーの Salesforce ログインを管理するように設定された通常の Mobile SDK アプリケーションです。このアプリケーションは、デバイスの Mobile SDK アプリケーションと Salesforce 認証サービス間のブローカーとして機能します。最近ログインしたデバイスユーザーを追跡し、ユーザーがアプリケーションに戻ったときに認証プロセスを開始します。
- ID プロバイダークライアントは、ログインに ID プロバイダーを使用するように設定された通常の Mobile SDK アプリケーションです。これらのアプリケーションはユーザーがアクセスを試みているサービスを提供するため、「サービスプロバイダー」または「SP」とも呼ばれます。従来のサービスプロバイダーは、ユーザーに Salesforce ログイン画面の 1 つの選択肢のみを提供します。一方 ID プロバイダークライアントは、ユーザーに Salesforce ログイン画面または特定の ID プロバイダーを使用したログインの選択肢を提供します。ID プロバイダーオプションを使用すると、IDP アプリケーションの現在の有効ユーザーの認証要求が Salesforce 認証エンドポイントに送信されます。

通常、単一デバイスが 1 つの ID プロバイダーアプリケーションをホストしますが、この制限は適用されません。単一デバイスが任意の数の ID プロバイダークライアントアプリケーションをホストできます。

 **メモ:** Mobile SDK では、バージョン 6.0 で ID プロバイダーサービスのサポートが導入されました。Mobile SDK 11.0 では、サポートが改良されてユーザーエクスペリエンスが向上し、新機能が追加されました。Mobile SDK バージョン 6.0 ~ 10.2 で作成されたアプリケーションでは、引き続き Mobile SDK 6.0 で導入された IDP / SP フローを使用できます。ただし、Mobile SDK 11.0 以降で作成されたアプリケーションでは、Mobile SDK 11.0 で改良された各 IDP / SP フローを使用して機能を実装する必要があります。

ID プロバイダーアプリケーションとそのクライアントには、いくつかのルールが適用されます。Mobile SDK 11.0 でサポートされる最新の IDP および SP フローには次の制限が適用されます。

- Mobile SDK 11.0 以降を使用して作成されている必要があります。
- 個別のアプリケーションである必要があります。単一アプリケーションに ID プロバイダーと ID プロバイダークライアントの実装を組み合わせることはできません。

## ID プロバイダーアプリケーション

- クライアントとその委任先の ID プロバイダーは、異なる OAuth コールバック URI が設定された異なる Salesforce 接続アプリケーションを使用する必要があります。

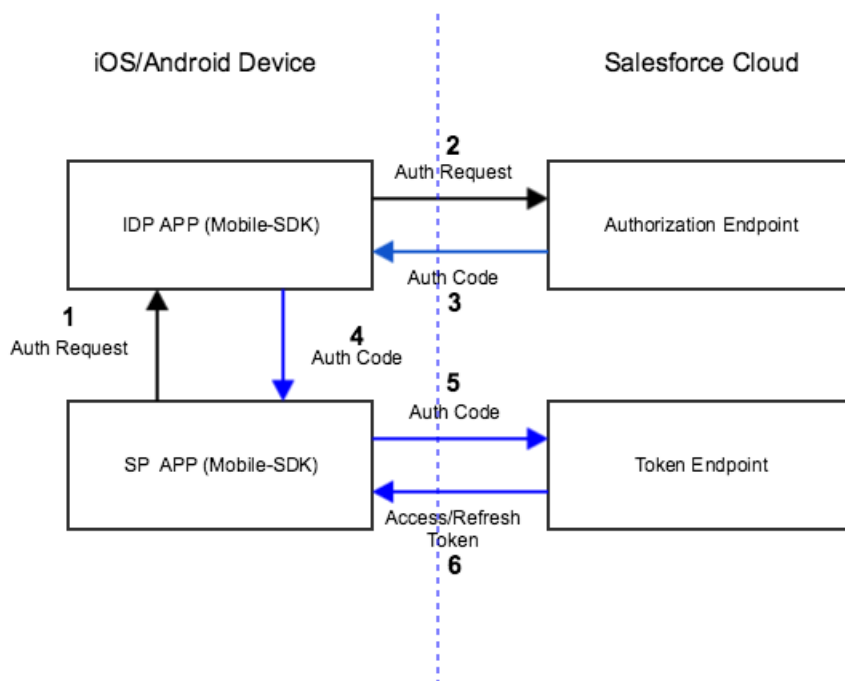
## ID プロバイダー: アーキテクチャ、フロー、および接続アプリケーションの要件

すべての Mobile SDK ID プロバイダーの設定は、プラットフォームに関係なく同じフローとアーキテクチャ概要に従います。ID プロバイダーコンポーネントの Salesforce 接続アプリケーションには、いくつかの特別な要件も適用されます。

### 概要レベルのフロー

ユーザーは、ID プロバイダーアプリケーションまたはそのクライアントのいずれかから ID プロバイダーフローを開始できます。ユーザーが ID プロバイダークライアントアプリケーションを起動してフローを開始する場合、最初に標準の Salesforce ログインビューまたは ID プロバイダーを使用して、ログイン設定を選択します。ID プロバイダーアプリケーションを直接起動して、ID プロバイダーログインを選択することもできます。この場合、ID プロバイダーが要求されたクライアントアプリケーションを起動して、認証を開始します。

ID プロバイダークライアントでフローが開始されるときの動作の概要ダイアグラムを次に示します。クライアントアプリケーションは、ユーザーからログイン情報を収集しません。代わりに、ID プロバイダーアプリケーションにログイン処理を委任します。



1. ユーザーの要求に応じて、ID プロバイダークライアントアプリケーションが ID プロバイダーに認証を要求します。ID プロバイダーは認証要求を Salesforce 認証エンドポイントに転送することで応答します。
2. Salesforce が認証コードを ID プロバイダーに送信します。
3. ID プロバイダーが認証コードを ID プロバイダークライアントに転送します。
4. ID プロバイダークライアントが認証コードを Salesforce OAuth トークンエンドポイントに転送します。



5. Salesforce が認証コードをアクセストークンおよび更新トークンと交換し、トークンを ID プロバイダークライアントアプリケーションに返します。ユーザーの OAuth トークンを処理するのはクライアントアプリケーションのみです。

ユーザーは、ID プロバイダーのボタンをクリックしてから、アカウントを選択するだけです。ユーザーは Salesforce データにアクセスするための標準要求を受け入れたら、クライアントアプリケーションで作業を開始できます。

## 詳細情報

認証フローのより詳しい情報を次に示します。ID プロバイダークライアントでは次のように動作します。

1. ユーザーが ID プロバイダークライアントアプリケーションで有効な更新トークンを持っている場合、自動的にログインし、通常の更新トークンの交換が発生するだけです。
2. ユーザーが有効な更新トークンを持っていない場合、ID プロバイダークライアントはユーザーに Salesforce ログインまたは ID プロバイダーログインの選択を促します。

次の条件が該当する場合、Mobile SDK はユーザーの ID プロバイダーログインの選択を無視し、代わりに標準の Salesforce ログイン画面を表示します。

- ID プロバイダークライアントアプリケーションで設定されている ID プロバイダーアプリケーションがない。
- 設定された ID プロバイダーアプリケーションが見つからない。

認証に成功すると、クライアントは Salesforce にユーザーのトークンを更新するように要求します。Salesforce が更新トークンを返すと、フローはデフォルトで標準 Mobile SDK ログイン後フローに設定されます。Mobile SDK は、Salesforce から返された更新トークンをキャッシュします。

## 接続アプリケーションの要件

- 2 つの接続アプリケーション (ID プロバイダーアプリケーションに 1 つ、ID プロバイダークライアントアプリケーションに 1 つ) を作成します。
- ID プロバイダークライアントアプリケーションの接続アプリケーションの [コンシューマーの秘密が必要] をオフにします。
- OAuth コールバック URI を独自のカスタムスキームに設定します。ID プロバイダークライアントアプリケーションのカスタム URI スキームは、その ID プロバイダーアプリケーションのスキームと同じにすることはできません。

### ❗ 重要:

- ID プロバイダーアプリケーションは、接続アプリケーションで web 範囲を有効にし、それをモバイルアプリケーションで要求する必要があります。そうしないと、Mobile SDK はユーザーをエラーメッセージのあるログインページにリダイレクトします。この時点で、フローは続行できなくなります。
- ID プロバイダーとそのクライアントは、独自の consumerKey と callbackUrl の値が設定された異なる接続アプリケーションを使用する必要があります。そうしないと、ID プロバイダーが認証コードを要求したときに、Salesforce サービスは「ログイン情報が無効です」というエラーメッセージを返します。現在、すべての Mobile SDK サンプルアプリケーションは同じ consumerKey と callbackUrl を使用しています。これらのアプリケーションを ID プロバイダーとクライアントのペアとして採用する場合は、各アプリケーションが一意の値を使用していることを確認してください。

## Android アーキテクチャおよびフロー

---

ユーザーは、ID プロバイダーまたは ID プロバイダークライアントアプリケーションを開いて、ID プロバイダーフローを開始できます。Android では、どちらから開始されるかによって実装が異なります。

### ID プロバイダークライアント

次の手順は、ID プロバイダークライアントから起動された場合の ID プロバイダーフローを示しています。

1. ユーザーが ID プロバイダークライアントを起動し、[Log in with IDP (IDP でログイン)] を選択します。
2. ID プロバイダークライアントが認証のために ID プロバイダーにスウィズルします。
3. ID プロバイダーが認証を完了し、クライアントアプリケーションにスウィズルします。

### ID プロバイダーによる開始

次の手順は、ID プロバイダーから起動された場合の ID プロバイダーフローを示しています。

1. ユーザーが ID プロバイダーアプリケーションを起動します。
2. ユーザーがクライアントアプリケーションを選択します。
3. IDP アプリケーションの現在の有効ユーザーの認証要求が Salesforce 認証エンドポイントに送信されます。
4. ID プロバイダーアプリケーションがバックグラウンドで、選択されたクライアントアプリケーションと通信し、ユーザー名 (`user_hint`) を渡します。クライアントアプリケーションにユーザーが存在する場合、準備ができていることがユーザーに通知され、フローはステップ 6 に進みます。
5. クライアントアプリケーションにユーザーが存在しない場合、次の手順が実行されます。
  - クライアントアプリケーションは、認証コードを要求する応答を ID プロバイダーアプリケーションに返します。
  - ID プロバイダーアプリケーションは認証コードを取得してクライアントアプリケーションに送信します。
  - クライアントアプリケーションは認証コードの更新トークンの交換を実行し、準備ができたことを IDP アプリケーションに伝えます。
6. ID プロバイダーアプリケーションは、現在ログインされているクライアントアプリケーションにスウィズルします。

### ID プロバイダーとしての Android アプリケーションの設定

---

Mobile SDK 11.0 以降で作成されたアプリケーションは、ID プロバイダーとして設定できます。接続先の ID プロバイダークライアントアプリケーションを定義するメソッドをコールした後、アプリケーションの UI で ID プロバイダークライアントを選択します。

ID プロバイダーアプリケーションを作成する最も簡単な方法は、Mobile SDK AndroidIDPTemplate を使用することです。このテンプレートは GitHub の [github.com/forcedotcom/SalesforceMobileSDK-Templates](https://github.com/forcedotcom/SalesforceMobileSDK-Templates) リポジトリで使用できま

す。次のコマンドライン例のように、テンプレートリポジトリの URI で *forcedroid createwithtemplate* コマンドを使用します。

```
$ forcedroid createwithtemplate
Enter URI of repo containing template application: AndroidIDPTemplate
Enter your application name: MyIDP-Android
Enter your package name: com.acme.android
Enter your organization name (Acme, Inc.): Acme Systems
Enter output directory for your app (leave empty for the current directory): MyIDP-Android
```

## 既存の Mobile SDK Android アプリケーションの ID プロバイダーへの変換

ID プロバイダーアプリケーションで、サービスを提供するクライアントアプリケーションを認識するには、1 つ(または複数)のクライアントアプリケーション設定をそのアプリケーションサブクラスの `onCreate` メソッドで定義します。

```
class MyApplication : Application() {

 override fun onCreate() {
 super.onCreate()
 SalesforceSDKManager.initNative(applicationContext, MainActivity::class.java)
 SalesforceSDKManager.getInstance()
 .setAllowedSPApps(listOf(
 SPConfig(
 "com....restexplorer", /* Package name of SP app */
 "com....restexplorer.ExplorerActivity", /* Main activity of SP app */
 "<-- the oauth consumer key of the sp app -->",
 "<-- the oauth callback url of the sp app -->",
 arrayOf("api", "web") /* Oauth scopes of the SP app */
),
),
)
 /* Code here not shown */
 }
}
```

IDP-init のログインフローを開始するために、次のサンプルコードでは、選択されたクライアントアプリケーションパッケージ名で `kickOffIDPInitiatedLoginFlow` をコールし、状況更新を処理します。この例では、更新情報はトースト通知に表示されますが、ユーザーに進行状況を表示する方法は最終的にアプリケーションで決定されます。このコードの対応するバージョンは、IDP アプリケーションの UI で確認できます。選択 UI の例は、GitHub リポジトリ (<https://github.com/forcedotcom/SalesforceMobileSDK-Templates>) を参照してください。Android の場合は、`AndroidIDPTemplate` を参照してください。

```
SalesforceSDKManager.getInstance().idpManager?.let { idpManager ->
 idpManager.kickOffIDPInitiatedLoginFlow(this, spAppPackageName,
 object:IDPManager.StatusUpdateCallback {
 override fun onStatusUpdate(status: IDPManager.Status) {
 CoroutineScope(Dispatchers.Main).launch {
 Toast.makeText(
 applicationContext,
 getString(status.resIdForDescription),
```

```

 Toast.LENGTH_SHORT
).show()
 }
}
}
)
} ?: run {
 Log.e(TAG, "Cannot proceed with launch of ${appName} - not configured as IDP")
}

```

onStatusUpdate コールバックから、次のいずれかの状況更新が返される場合があります。

- LOGIN\_REQUEST\_SENT\_TO\_SP
- GETTING\_AUTH\_CODE\_FROM\_SERVER
- ERROR\_RECEIVED\_FROM\_SERVER
- AUTH\_CODE\_SENT\_TO\_SP
- SP\_LOGIN\_COMPLETE

## ID プロバイダークライアントとしての Android アプリケーションの設定

MobileSDK11.0以降で作成されたアプリケーションは、IDプロバイダーとして使用されているアプリケーションと同じでなければ、ID プロバイダークライアントとして設定できます。アプリケーション自体を ID プロバイダークライアントとして識別するように設定し、IDP アプリケーションを識別するメソッドをコールすれば、残りは Mobile SDK が処理します。

クライアントアプリケーションで IDP アプリケーションを認識できるようにするには、IDP アプリケーションパッケージ名をそのアプリケーションサブクラスの onCreate メソッドで定義します。

```

class MyApplication : Application() {

 override fun onCreate() {
 super.onCreate()
 SalesforceSDKManager.initNative(applicationContext, MainActivity::class.java)
 SalesforceSDKManager.getInstance()
 .setIDPAppPackageName(
 "com.salesforce.samples.salesforceandroididptemplateapp" /* Package name of the IDP
app */
)
 }
 /* Code here not shown */
}

```

クライアントアプリケーション側で必要なことはこれのみです。IDP アプリケーションと IDP クライアントアプリケーションの両方からログインできるようにするには、IDP アプリケーションでクライアントアプリケーションを認識するように設定していることも確認します。両方のアプリケーションを設定すると、次のようになります。

- **[Login with IDP app (IDP アプリケーションを使用してログイン)]** ボタンがログイン画面に表示されます。選択した場合、クライアントアプリケーションによって開始されるログインフローが開始されます。このボ

タンの表示ラベルはリソース文字列 `sf__launch_idp` によって管理されますが、これをアプリケーション内で上書きできます。

- これで、アプリケーションは IDP-init のログイン要求に応答します。

## ID プロバイダーとしての iOS アプリケーションの設定

MobileSDK11.0以降で作成されたアプリケーションは、IDプロバイダーとして設定できます。アプリケーション自体をIDプロバイダークライアントとして識別するように設定すれば、残りは Mobile SDK が処理します。

IDプロバイダーアプリケーションを作成する最も簡単な方法は、MobileSDKiOSIDPTemplateを使用することです。このテンプレートはGitHubの[github.com/forcedotcom/SalesforceMobileSDK-Templates](https://github.com/forcedotcom/SalesforceMobileSDK-Templates) リポジトリで使用できます。次のコマンドライン例のように、テンプレートリポジトリのURIで `forceios createwithtemplate` コマンドを使用します。

```
$ forceios createwithtemplate
Enter URI of repo containing template application: IOSIDPTemplate
Enter your application name: MyIDP-iOS
Enter your package name: com.acme.android
Enter your organization name (Acme, Inc.): Acme Systems
Enter output directory for your app (leave empty for the current directory): MyIDP-iOS
```

## 既存の Mobile SDK iOS アプリケーションの ID プロバイダーへの変換

既存の Mobile SDK 11.x (以降) iOS アプリケーションを ID プロバイダーに変換する手順は、次のとおりです。

1. SalesforceSDKManager で、`isIdentityProvider` を `true` に設定します。
2. AppDelegate クラスの実装で次のメソッドを探し、コメント化されているコードを次のように元に戻します。

### Swift

```
func application(_ app: UIApplication,
 open url: URL,
 options: [UIApplication.OpenURLOptionsKey : Any] = [:]) -> Bool {
 return UserAccountManager.shared.handleIdentityProviderCommand(
 from: url, with: options)
}
```

### Objective-C

```
- (BOOL)application:(UIApplication *)app
 openURL:(NSURL *)url
 options:(NSDictionary<UIApplicationOpenURLOptionsKey,id>*)options {

 return [[SFUserAccountManager sharedInstance]
 handleIDPAuthenticationResponse:url options:options];
}
```

3. `info.plist` 設定にカスタム URI スキームを追加します。たとえば、次の XML はカスタム URI スキームとして「sampleidpapp」を定義します。

```
<key>CFBundleURLTypes</key>
<array>
 <dict>
 <key>CFBundleURLSchemes</key>
 <array>
 <string>sampleidpapp</string>
 </array>
 </dict>
</array>
```

既存の Mobile SDK 11.x (以降) iOS アプリケーションを ID プロバイダーに変換するには、`SalesforceSDKManager` に移動します。次に、以下の操作を行います。

1. `isIdentityProvider` を `true` に設定します。
2. `SFUserAccountManager` で、次の方法を使用してフローを開始します。

```
- (void)kickOffIDPInitiatedLoginFlowForSP:(SFSDKSPConfig *)config
 statusUpdate:(void (^)(SFSPLoginStatus))statusBlock
 failure:(void (^)(SFSPLoginError))failureBlock;
```

## (省略可能) IDP フローのキーチェーンの設定

IDP が開始したログインの場合、共有キーチェーングループを使用して IDP と IDP クライアントアプリケーション間で通信できるため、ユーザーがアプリケーション間を切り替える回数が減ります。

1. Xcode プロジェクト設定の [Keychain Sharing (キーチェーン共有)] セクションにキーチェーングループを追加します。

すでにアプリケーショングループを設定している場合、アプリケーショングループから自動的に生成されたキーチェーングループを使用できます。



**メモ:** IDP に使用するキーチェーンと他の Mobile SDK 操作に使用するキーチェーンは、個別に設定できます。

2. キーチェーングループでアプリケーションを設定し、残りのキーチェーン項目を共有せずに IDP トークンのみを共有する場合、`KeychainHelper.accessGroup` をアプリケーションの非公開キーチェーンアクセスグループに設定します。それ以外の場合、アプリケーションはデフォルトでリストの最初のキーチェーングループを使用します。

## (省略可能) キーチェーンを使用する IDP アプリケーションの設定

1. IDP アプリケーションで、`SalesforceSDKManager` に移動し、`isIdentityProvider` を `true` に設定します。



2. 次のメソッドを使用して、SFUserAccountManager でフローを開始します。

```
- (void)kickOffIDPInitiatedLoginFlowForSP:(SFSDKSPConfig *)config
statusUpdate:(void (^)(SFSPLoginStatus))statusBlock
failure:(void (^)(SFSPLoginError))failureBlock;
```

## (省略可能) ID プロバイダー UI のカスタマイズ

クライアントアプリケーションがログイン要求を転送すると、通常は ID プロバイダーが選択ダイアログボックスを提示します。既知のユーザーがリストされるこのダイアログボックスは、次の条件の1つ以上が true の場合にのみ表示されます。

- ユーザーがこの要求の前に他の ID プロバイダークライアントアプリケーションからログインしている。
- ユーザーがこの要求の前に ID プロバイダーアプリケーションに直接ログインしている。
- 現在複数のユーザーがログインしている。

 **メモ:** 注意: この要求の前にログインしているユーザーがいらない場合、ユーザーがログインフローを正常に完了した後に Mobile SDK はログイン画面を表示して認証に進みます。

ユーザー選択ビューをカスタマイズするには、ID プロバイダーアプリケーションで UIViewController クラスを拡張し、SFSDKUserSelectionView プロトコルも実装する必要があります。

```
@protocol SFSDKUserSelectionViewDelegate
- (void)createNewUser:(NSDictionary *)spAppOptions;
- (void)selectedUser:(SFUserAccount *)user
 spAppContext:(NSDictionary *)spAppOptions;
- (void)cancel();

@protocol SFSDKUserSelectionView<NSObject>
@property (nonatomic,weak) id<SFSDKUserSelectionViewDelegate> userSelectionDelegate;
@property (nonatomic,strong) NSDictionary *spAppOptions;
@end
```

ID プロバイダークライアントアプリケーションでは、Mobile SDK は SFSDKUserSelectionView プロトコルに定義された userSelectionDelegate および spAppOptions プロパティのインスタンスを設定します。ID プロバイダーのビューコントローラーでこれらのオブジェクトを使用して、Mobile SDK にユーザーのユーザーアカウント選択を通知します。たとえば、UserSelectionViewController という UIViewController クラスの SFSDKUserSelectionView プロトコルを実装しているとします。その場合、次のように SalesforceSDKManager 共有インスタンスで idpUserSelectionBlock を設定することで、そのビューコントローラーをユーザー選択ダイアログボックスとして使用できます。

```
//optional : Customize the User Selection Screen
[SalesforceSDKManager sharedManager].idpUserSelectionBlock =
 ^UIViewController<SFSDKUserSelectionView> *{
 UserSelectionViewController *controller =
 [[UserSelectionViewController alloc] init];
 return controller;
 }
```



# ID プロバイダークライアントとしてのiOSアプリケーションの設定

MobileSDK11.0以降で作成されたアプリケーションは、IDプロバイダークライアントとして設定できます。アプリケーション自体をIDプロバイダークライアントとして識別するように設定し、そのIDプロバイダーを指定します。残りは Mobile SDK が処理します。

1. AppDelegate クラスの `init()` メソッドで、使用している ID プロバイダーの URI スキームを指定します。

## Swift

```
SalesforceManager.shared.identityProviderURLScheme = "sampleidpapp"
```

## Objective-C

```
[SalesforceSDKManager sharedManager].idpAppURIScheme = @"sampleidpapp";
```

2. アプリケーションの `info.plist` ファイルで、IDプロバイダークライアントの接続アプリケーションで定義されている URI スキームを追加します。

```
<key>CFBundleURLTypes</key>
<array>
 <dict>
 <key>CFBundleURLSchemes</key>
 <array>
 <string>sampleidpclientapp</string>
 </array>
 </dict>
</array>
```

3. AppDelegate クラスの実装で次のメソッドを探し、コメント化されているコードを次のように元に戻します。

## Swift

```
func application(_ app: UIApplication,
 open url: URL,
 options: [UIApplication.OpenURLOptionsKey : Any] = [:]) -> Bool {
 return UserAccountManager.shared.handleIdentityProviderCommand(
 from: url, with: options)
}
```

## Objective-C

```
- (BOOL)application:(UIApplication *)app
 openURL:(NSURL *)url
 options:(NSDictionary<UIApplicationOpenURLOptionsKey,id>*)options {

 return [[SFUserAccountManager sharedInstance]
 handleIDPAuthenticationResponse:url options:options];
}
```

これで、アプリケーションを ID プロバイダークライアントとして使用できます。

## (省略可能) キーチェーンを使用する IDP クライアントアプリケーションの設定

認証を開始するように IDP アプリケーションを設定したら、クライアントを更新してフローを完了します。次のメソッドを使用して、受信 IDP URL を処理できます。

### Swift

```
public func handleIdentityProviderCommand(from url: URL,
with options: [AnyHashable: Any],
completion: @escaping (Result<(UserAccount, AuthInfo),
UserAccountManagerError>) -> Void) -> Bool {
 return __handleIDPAuthenticationCommand(url, options: options, completion: {
 (authInfo, userAccount) in
 completion(Result.success((userAccount, authInfo)))
 }) { (authInfo, error) in
 completion(Result.failure(.loginFailed(underlyingError: error, authInfo:
authInfo)))
 }
 })
}
```

### Objective-C

```
- (BOOL)handleIDPAuthenticationCommand:(NSURL *)url
options:(nonnull NSDictionary *)options
completion:(nullable SFUserAccountManagerSuccessCallbackBlock)completionBlock
failure:(nullable SFUserAccountManagerFailureCallbackBlock)failureBlock
NS_REFINED_FOR_SWIFT;
```

## (省略可能) クライアントアプリケーションでのログインフロー選択ビューのカスタマイズ

Mobile SDK には、ID プロバイダーとそのクライアントアプリケーション両方に対するテンプレートアプリケーションが用意されています。クライアントテンプレートには、ユーザーが ID プロバイダーまたは Salesforce ログイン画面を使用してログインを選択できるビューが定義されています。次の条件が該当する場合、ユーザーがクライアントテンプレートから作成されたアプリケーションを開くと、アプリケーションにこのビューが表示されます。

- ユーザーがまだログインしていない。または
- 現在のユーザーが設定されていない。

ログインスタイル選択ビューをカスタマイズするには、クライアントアプリケーションで UIViewController クラスを拡張し、SFSDKLoginFlowSelectionView プロトコルを実装する必要があります。

```
@protocol SFSDKLoginFlowSelectionViewDelegate<NSObject>
/**
 * Used to notify the SDK of user selection on the login flow selection view
 * @param controller instance invoking this delegate
 * @param appOptions addl. name value pairs sent from the sdk for
 * the SFSDKLoginFlowSelectionView
 */
```

```

-(void)loginFlowSelectionIDPSelected:(UIViewController *)controller
 options:(NSDictionary *)appOptions;

/**
 * Used to notify the SDK of user selection on the login flow selection view
 * @param controller instance invoking this delegate
 * @param appOptions addl. name value pairs sent from the sdk for
 * the SFSDKLoginFlowSelectionView
 */
-(void)loginFlowSelectionLocalLoginSelected:(UIViewController *)controller
 options:(NSDictionary *)appOptions;

@end

@protocol SFSDKLoginFlowSelectionView<NSObject>
@property (weak, nonatomic) id <SFSDKLoginFlowSelectionViewDelegate>selectionFlowDelegate;

@property (nonatomic, strong) NSDictionary *appOptions;
@end

```

クライアントアプリケーションの ID プロバイダーフロー中、Mobile SDK はこのプロトコルに定義された `selectionFlowDelegate` および `appOptions` プロパティのインスタンスを設定します。ビューコントローラーでこれらのアイテムを使用して、Mobile SDK にユーザーのログイン方法の選択を通知します。たとえば、`IDPLoginNavController` という `UIViewController` クラスの `SFSDKUserSelectionView` プロトコルを実装しているとします。その場合、次のように `SalesforceSDKManager` 共有インスタンスで `idpLoginFlowSelectionAction` を設定することで、そのビューコントローラーをユーザー選択ダイアログボックスとして使用できます。

```

//optional : Customize the Login Flow Selection screen
[SalesforceSDKManager sharedManager].idpLoginFlowSelectionAction =
^UIViewController<SFSDKLoginFlowSelectionView> *{
 IDPLoginNavController *controller =[[IDPLoginNavController alloc] init];
 return controller;
}

```

## Mobile SDK を使用しないモバイル ID プロバイダーアプリケーションの実装

Salesforce に接続するアプリケーションをホストする Web サイトを所有している場合、それらのアプリケーションの ID サービスを提供するように Salesforce を設定できます。こうすることで、ホストされるアプリケーションのユーザーは、ホストの Web サイトでシングルサインオンの利便性を活用できます。しかし、ネイティブモバイルアプリケーションで同じサービスを採用できるのでしょうか? できるとしたら何が必要なのでしょう? 答えは「採用できる」です。この作業は難しくありません。2つの方法 (Salesforce Mobile SDK を使用する方法と Salesforce Mobile SDK を使用しない方法) のいずれかで、ネイティブモバイルアプリケーションを Salesforce ID プロバイダーとして定義できます。この記事には、Salesforce Mobile SDK を使用せずにモバイル ID プロバイダーとそれらのクライアントを作成する手順が記載されています。

ID プロバイダー (IdP) 認証フローを使用すると、1つの信頼できるアプリケーションをすべてのデバイスの Salesforce ログイン要件の中央ハンドラーとして指定できます。任意のアプリケーションを IdP アプリケーショ

ンまたは IdP クライアントアプリケーション (サービスプロバイダー (SP) と呼ばれる) のいずれかになるように設定できます。クライアントアプリケーションは、ユーザー認証を指定された ID プロバイダーに委任します。ID プロバイダーは、ログインにユーザーの認証状態を使用します。

適切に設定すれば、IdP サービスを非 Mobile SDK アプリケーションに実装することもできます。Salesforce 側の要件は次のとおりです。

**❗ 重要:**

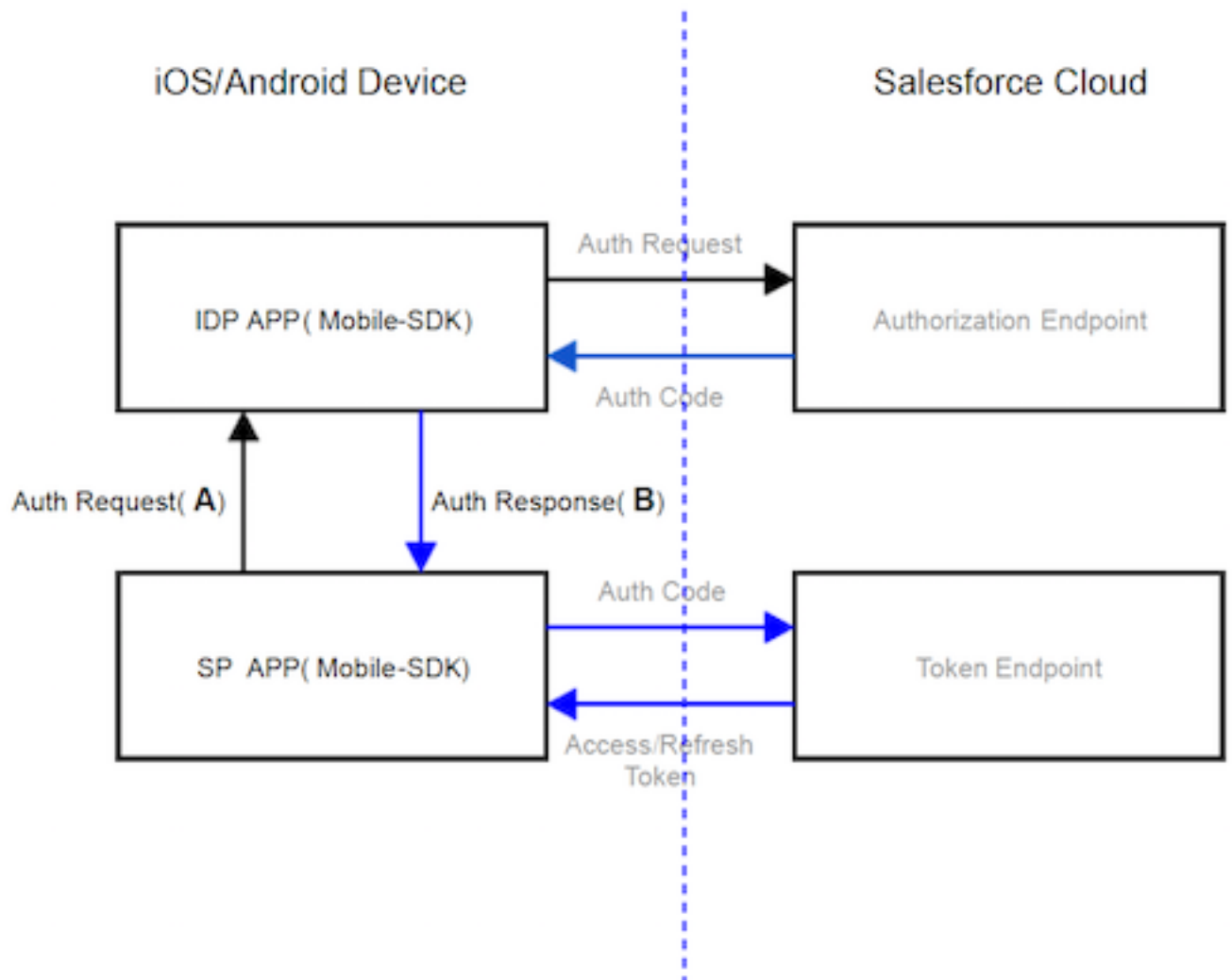
- これらの手順は、Mobile SDK を使用せずに Salesforce に対する認証を行うアプリケーションスイートを提供するモバイルアプリケーション開発者を対象にしています。この使用事例の ID プロバイダー実装では、シングルサインオン (SSO) をシミュレーションしてカスタマーエクスペリエンスを改善できます。ただし、ここではログイン認証の一元化に必要な最小限の情報交換のみを説明します。この方法では、モバイル実装の負荷がすべて開発者にかかります。
- 少ない労力で同じ結果を得るには、Salesforce Mobile SDK を使用します。『Salesforce Mobile SDK 開発ガイド』の「[ID プロバイダーとしての Mobile SDK アプリケーションの使用](#)」を参照してください。

## ID プロバイダー設定のルール

- IdP アプリケーションの ID は、クライアント SP アプリケーションに組み込まれています。
- IdP アプリケーションには、クライアント SP アプリケーションの予備知識はありません。
- IdP アプリケーションと SP アプリケーションは、両方のアプリケーションが理解できる定義済みの形式で要求情報と応答情報を交換します。この情報の交換モードは、モバイルオペレーティングシステムによって異なります。
- IdP アプリケーションには、現在ログインしているユーザーのリストが保持されます。
- 既知のユーザーの場合、IdP アプリケーションは Salesforce に認証コードを要求します。不明なユーザーの場合、IdP アプリケーションは Salesforce ログイン画面を表示します。
- SP アプリケーションは、IdP アプリケーションから認証コードを受信すると、そのコードを直接使用してユーザーの認証アーティファクトを更新します。SP アプリケーションとその IdP でトークンやパスワードは共有されません。
- IdP とその SP クライアントアプリケーションは、異なる開発者アカウントで公開できます。

## 認証フロー

ID プロバイダー認証のフローは Mobile SDK を使用するかどうかに関係なく同じです。基本フローダイアグラムを思い出してみましょう。



SP が開始するフローの場合、IdP アプリケーションが SP アプリケーションから受信した認証要求を処理します (A)。Salesforce サービスとのネゴシエーション後に、IdP アプリケーションは Salesforce から受信した認証コードを SP アプリケーションに送信します (B)。IdP アプリケーションは、カスタム URL スキームまたはパッケージ ID をクライアントアプリケーションと共有します。

次のセクションでは、(A) と (B) で送信されるメッセージについて説明します。

## (A) SP アプリケーションからの認証要求の処理

SP アプリケーションは、次の情報と共に認証要求を送信します。IdP アプリケーションはこの情報を待機し、この情報を読み込むためのコードを提供します。

**oauth\_client\_id**

SP アプリケーションのクライアント ID。

**code\_challenge**

SP アプリケーションの非公開鍵を使用した暗号化ハッシュ。「[Proof Key for Code Exchange by OAuth Public Clients \(OAuth 公開クライアントによるコード交換の証明鍵\)](#)」を参照してください。

**oauth\_redirect\_uri**

SP アプリケーションのリダイレクト URI。SP の接続アプリケーションに登録されている必要があります。

**scopes**

要求される範囲のカンマ区切りのリスト。

**user\_hint**

IDP が開始するフロー用です。選択したユーザーのループバックパラメーターとして使用されます。

**login\_host**

SP アプリケーションで選択された Salesforce ログインホスト。

**state**

コードフローの完了時に IDP から SP アプリケーションに返される状態値。

iOS の場合、この情報をパラメーターとして運ぶカスタムスキームは IDP アプリケーションが定義します。IDP アプリケーションのクライアント SP アプリケーションはこのスキームを認識しています。次に例を示します。

```
<IDP-APP-URL-SCHEME>://oauth2/v1.0/authrequest?
 oauth_client_id={ClientID}
 &code_challenge=base64({code challenge})
 &oauth_redirect_uri={SP APP Redirect URI}
 &scopes={requested scopes comma separated}
 &user_hint={orgid:userid}
 &login_host={loginHost}
 &state={state}
```

<IDP-APP-URL-SCHEME> は、Salesforce 接続アプリケーションで定義されるコールバック URL です。

Android の場合、SP アプリケーションはカスタムスキームの代わりに、キー-値ペアとしてこの情報を格納するバンドルを使用してインテントを起動します。

IDP アプリケーションが **user\_hint** で記述されたユーザーを認識している場合、受信した値を使用して Salesforce 認証エンドポイントに認証コードを要求します。認識していない場合、Salesforce ログイン画面を表示し、通常の Salesforce ログインフローに戻ります。

## (B) IDP アプリケーションからコール元 SP アプリケーションへの認証コード応答の送信

IDP アプリケーションは、Salesforce から認証コードを受信すると、次の情報と共に応答をコール元 SP アプリケーションに送信します。

**code**

認証要求 (A) の結果として IDP アプリケーションが受信した認証コード。

**login\_host**

認証要求 (A) で使用された Salesforce ログインホスト。

**state**

認証要求 (A) で IDP アプリケーションが SP アプリケーションから受信した状態値。

iOS の場合、これらの値をパラメーターとして受信するカスタムスキームは SP アプリケーションが定義します。SP アプリケーションは、`oauth_redirect_uri` 要求パラメーターを介してこのスキームを IdP アプリケーションに提供します。次に例を示します。

```
<SP-APP-URL-SCHEME>://oauth2/v1.0/authresponse?
 code={authcode}
 &state={state}
 &login_host={loginHost}
```

Android の場合、IdP アプリケーションはカスタムスキームの代わりに、これらのキー - 値ペアを格納するバンドルを使用して Intent を起動します。

## エラー処理

エラーには次の形式が使用されます。

```
<APP-URL-SCHEME>://oauth2/v1.0/error?
 error_code={error_code}
 &error_desc={description}
 &error_reason={reason}
 &state={state}
```

このセクションの内容:

### 実装の詳細とオプション

基本的な通信の詳細以外に、特定の標準シナリオが大部分の ID プロバイダーの設定に適用されます。

## 実装の詳細とオプション

基本的な通信の詳細以外に、特定の標準シナリオが大部分の ID プロバイダーの設定に適用されます。

## IdP アプリケーションからの認証の開始

IdP アプリケーションを設計して、ユーザーが SP アプリケーションではなく IdP アプリケーションを認証の開始点として使用できるようにすることもできます。この場合、IdP アプリケーションが次の情報を SP アプリケーションに送信します。

### user\_hint

ハンドルをユーザー参照に提供する値。SP アプリケーションはこのヒントを使用して、ユーザーの存在を検証できます。ユーザーアカウントが見つからない場合、SP アプリケーションはこの値を使用して IdP アプリケーションを呼び出します。その後、IdP アプリケーションは (A) の場合と同様に認証コードを取得します。一方、SP アプリケーションでユーザーアカウントを使用できる場合、SP アプリケーションは単純にそのユーザーに切り替えます。

### login\_host

認証要求 (A) で使用された選択済みログインホスト。

### start\_url

ユーザーが選択されたときに移動する URL。



iOS では SP アプリケーションのカスタムスキームを使用して、Android ではキー - 値ペアが含まれるバンドルを使用してintentを起動します。たとえば、iOS の場合は次のようになります。

```
<SP-APP-URL-SCHEME>://oauth2/v1.0/idpinit?
 user_hint={orgid:userId}
 &login_host={loginhost}
 &start_url={starturl}
```

## Salesforce 認証コードの取得

選択したユーザーがログインして他のセキュリティ要件を渡すと、IdP アプリケーションは Salesforce から認証コードを取得します。このコードの取得方法は、ID サービスの実装によって異なります。接続アプリケーションポリシーでユーザーデータへのアクセス権が要求される場合、IdP アプリケーションは Salesforce 権限の画面も表示する必要があります。

たとえば、JSP サーバーアプリケーションをコールする Web ビューを Mobile SDK が起動するとします。このサーバーページは、Salesforce 認証エンドポイントにアクセスして認証コードを取得し、アクセス権限画面を表示します。

## SP アプリケーションの検証

不正な SP アプリケーションから保護するために、IdP アプリケーションは次の操作を実行できます。

1. コール元アプリケーションのバンドル識別子を抽出する。
2. この識別子を `redirect_uri` と比較する。2つの値が一致しない場合、認証エラーが返されます。

本質的に、IdP アプリケーションに SP アプリケーションの予備知識があることは想定されていません。ただし、IdP アプリケーションは、サポートするアプリケーションの「ホワイトリスト」を維持するように選択できます。

## ユーザーの選択

ユーザーの選択では、IdP アプリケーションは次のいずれかの画面を表示します。

ユーザーの選択画面 — 1 人以上のユーザーがアカウントマネージャーであるが、どのユーザーも `user_hint` に一致しない場合に表示されます。

ログイン画面 — 次のいずれかの場合に表示されます。

- どのユーザーも IdP アプリケーションにログインしていない。
- 指定したユーザーが一度もログインしていない。

## 結果を返す

IdP アプリケーションは、`oauth_redirect_uri` に一致する要求を捕捉し、提供されたスキームまたはバンドル ID を使用して SP アプリケーションをコールします。

## 関連トピック

ID プロバイダーとして Salesforce の設定に関する下記のリンクでは、一般的な背景知識が提供されます。この情報の大部分は、モバイルアプリケーションを Salesforce IdP または SP として設定する場合には使用されません。

- Salesforce ヘルプの「[ID プロバイダーとサービスプロバイダー](#)」
- Salesforce ヘルプの「[シングルサインオン](#)」

Mobile SDK 8.2 以降、暗号化されたキー-値ストアによって、モバイルデバイス上で安全にデータを保存するための SmartStore に代わる機能が提供されます。キー-値ストアは、SmartStore と同一のものではなく、リレーショナルデータベースの機能をフル活用しない、単純なストレージ用途のために設計されたものです。これには、データのリレーションや構造を意識することなく、不透明な値のプールから迅速にデータを取得す必要があるレスポンスキャッシュなどがあります。

キー-値ストアは AES-256 暗号化を使用しており、デバイスのファイルシステム上で、それぞれ独自のディレクトリに格納されます。ストアごとに、ファイル名のプレフィックスとなる名前を指定します。ストア名には、文字、数字、アンダースコアのみを含めることができ、96 文字を超えることはできません。アプリケーションでは、デバイスの空き容量の限界まで、ストアを作成することができます。SmartStore インスタンスと同様に、キー-値ストアは、使用事例に応じて、ユーザーベースまたはグローバルのいずれかに設定できます。

キー-値ストアにはデータ型にとらわれず、さまざまな形態や形式を含めることができます。たとえば、1つの値は JavaScript ファイル、次の値は HTML、さらに次の値は PNG 画像などです。このストアでは、値のどうしのリレーションを認識する必要がなく、また要求されません。

バイナリデータを安全に保存するために、キー-値ストア API に特別なメソッドが用意されています。SmartStore での JSON エンベロープの作成や、Mobile SDK のファイル暗号化 API を使用したファイルの作成などの従来の手法の代わりに、これらのメソッドを使用します。

Android でより大きなデータセットの場合は、データをバッファリングして、値をキー-値ストアにストリーミングできます。たとえば、REST レスポンスキャッシュを作成し、`restResponse.asInputStream()` を `saveStream()` メソッドに渡すことでインポートできます。キー-値ストアは、データソースからのバッファをループで読み込み、ストアファイルに書き込みます。Mobile SDK for iOS では、同様のストリーミングはサポートされていません。

## キー - 値ストアと SmartStore の選択

次の場合には、キー-値ストアの使用について検討してください。

- キーによる単純な検索が、使用するデータセットに適している場合。
- 使用する値を JSON として保存したり、アトミックアクセスのために構造化したりしない場合。
- 大規模なデータセットを保存する場合。
- バイナリデータを保存しています。

- ネイティブアプリケーションを開発する場合。Mobile SDK 8.2 では、この機能をハイブリッドアプリケーションや ReactNative アプリケーションに使用することはできません。

次の場合には、SmartStore の方がより良い選択になります。

- 開発するアプリケーションに、複雑なクエリ処理能力が必要な場合。たとえば、複数の項目を絞り込むクエリ述語や、結合などのリレーショナルクエリなどが必要な場合です。
- 開発するアプリケーションで、データの一部分を取得できる必要がある場合。  
`select {soup:some_indexfield} from {soup}` のように SmartStore クエリでは可能な、値の小さな一部分の抽出ができません。
- 値が少なくとも半構造化されている場合。たとえば、データセットは JSON ですが、個々の値の形式が異なる場合です。
- ハイブリッドアプリケーションや ReactNative アプリケーションを開発している場合。

## キー - 値ストアのバージョン

---

Mobile SDK 8.2、8.3、9.0 では、キー - 値ストアのバージョン 1 が使用されます。Mobile SDK 9.1 では、バージョン 2 が導入されています。これらのバージョンでは、同じ基本機能が実装されていますが、バージョン 2 では、特定のストアのすべてのキーのリストを返す public アクセサーが追加されています。バージョン 1 でこのアクセサーを呼び出した場合、iOS では `nil` が返され、Android では例外が発生します。

Mobile SDK 9.1 では、両方のバージョンのストアがサポートされますが、作成されるストアはバージョン 2 のみです。Mobile SDK では、キーの一方方向のハッシュのみが認識され、キー自体は認識されません。このため、移行を自動的に行うことはできません。バージョン 1 ストアをバージョン 2 に変換するには、次の手順を実行します。

1. Mobile SDK 9.1 でストアを作成します。
2. 反復的なプロセスを使用して、キー - 値ペアを新しいストアに再作成します。

## キー - 値ストアクラス

---

キー - 値ストアファクトリーメソッドによって、ストアの作成、リスト、および削除が可能です。iOS の場合、これらのメソッドが、`KeyValueEncryptedFileStore` クラスに含まれています。Android の場合、次のようにこれらのメソッドを `SmartStoreSDKManager` からインポートします。

- **Android のみ:** `import com.salesforce.androidsdk.smartstore.app.SmartStoreSDKManager`

iOS および Android のストア管理メソッドは、それぞれの `KeyValueEncryptedFileStore` クラス内にあります。これらのメソッドを使用して、値の保存、アクセス、削除、またはキー - 値のペアの計数や削除を行います。

- **iOS (Swift):** `KeyValueEncryptedFileStore.swift` (SalesforceSDKCore に含まれています)
- **Android:** `KeyValueEncryptedFileStore.java` (com.salesforce.androidsdk.smartstore.store パッケージ)

 **メモ:** `KeyValueEncryptedFileStore` はネイティブ Swift API です。Objective-C 内からアクセスする場合、インポートに次の行を追加します。

```
#import <SalesforceSDKCore/SalesforceSDKCore-Swift.h>
```

## キー - 値ストアの設定

1. `KeyValueEncryptedFileStore` のインスタンスを作成します。

### iOS

このコンストラクターは、現在のユーザーのストアを作成します。

```
let kv = KeyValueEncryptedFileStore.shared(withName: "<some name>")
```

### Android

```
KeyValueEncryptedFileStore kv =
SmartStoreSDKManager.getKeyValueStore("<some name>")
```

2. 静的キー - 値ペアを追加します。

### iOS (Swift)

```
kv.saveValue(value, forKey: key)
```

### Android (Java)

```
kv.saveValue(key, value)
```

または、入力ストリームとして値を追加します。

### iOS (Swift)

未サポート

### Android (Java)

```
kv.saveStream(key, stream)
```

## バイナリデータの保存と取得 (キー - 値ストアのバージョン 2 以降)

バイナリデータをキー - 値ストアで管理するために、Mobile SDK 10.0 で新しい iOS メソッドが導入され、既存の Android メソッドが再利用されます。Mobile SDK のファイル暗号化 API や SmartStore の JSON エンベロープの代わりに、これらのメソッドを使用します。

### iOS (Swift)

```
/// Saving binary data to a key value store
/// Updates the data stored for the given key or adds a new entry
/// if the key does not exist.
/// - Parameters:
/// - data: Data to add to the store.
/// - key: Key associated with the data.
/// - Returns: True on success, false on failure.
@objc @discardableResult
public func saveData(_ data: Data, forKey key: String) → Bool

/// Accesses the data associated with the given key.
@objc public func readData(key: String) → Data?
```

#### 例

```
// Saving binary data to key value store
let sampleData = ...
store.saveData(sampleData, forKey:"key")

// Retrieving binary data back from key value store
let savedData = store.readData(key: "key")
```

### Android (Java)

```
/**
 * Save value given as an input stream for the given key.
 * Note: This method does not close the provided input stream
 *
 * @param key Unique identifier.
 * @param stream Stream to be persisted.
 * @return True - if successful, False - otherwise.
 */
public boolean saveStream(String key, InputStream stream) throws
IOException;

* Retrieving binary data from a key value store.

/**
 * Returns stream for value of given key.
 *
 * @param key Unique identifier.
 * @return stream to value for given key or null if key not found.
 */
public InputStream getStream(String key);
```

#### 例

```
// Saving binary data to key value store
//

byte[] arrayToWrite = ...;
```

```
// In real life, you probably would start from a stream
// (e.g. from a network call's response)
InputStream streamToWrite =
 new ByteArrayInputStream(arrayToWrite);
keyValueStore.saveStream("key", streamToWrite);

//
// Retrieving binary data back from key value store
//

InputStream streamToRead = keyValueStore.getStream("key");
byte[] arrayRead = Encryptor.
 getByteArrayStreamFromStream(streamToRead).toByteArray();
```

## ストア (キー - 値ストアのバージョン 2 以降) のすべてのキーの取得

---

次のメソッドは、特定のストアのすべてのキーを返します。

iOS (Swift)

```
/// All keys in the store
/// - Returns: all keys of stored values in a v2 store, nil if
/// it's a v1 store
@objc public func allKeys() -> [String]?
```

Android (Java)

```
/**
 * Get all keys.
 * NB: will throw UnsupportedOperationException for a v1 store
 */
public Set<String> keySet()
```

## キー - 値ストアのバージョンの取得

---

次の API では、実行時のストアのバージョンを確認できます。

iOS (Swift)

```
@objc public private(set) var storeVersion: Int
```

Android (Java)

```
public int getStoreVersion()
```

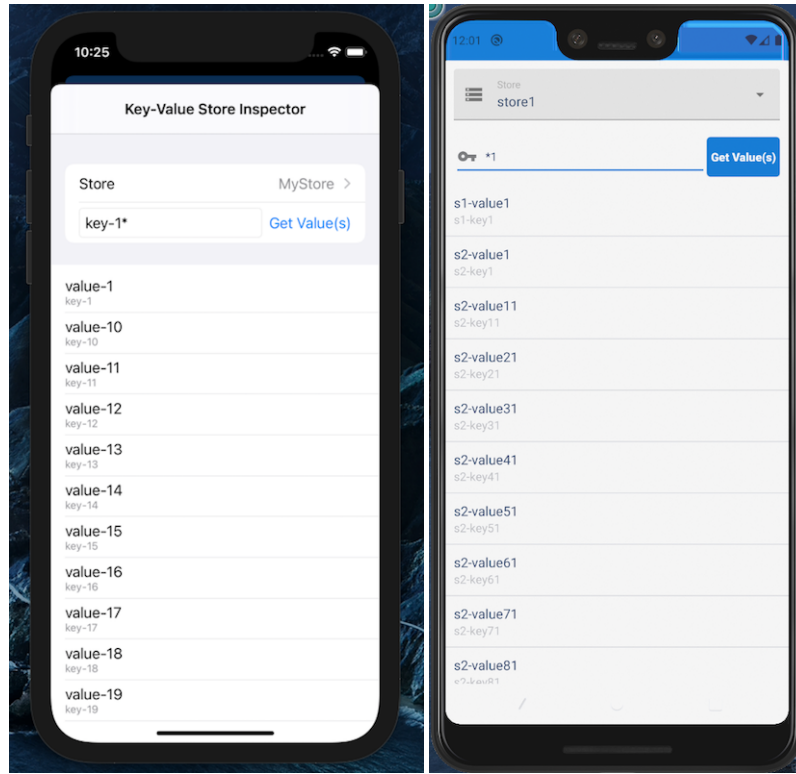
## キー - 値ストアの検査

---

キーと値のリストを表示するには、[Dev Support (開発サポート)] メニューの [Inspect Key-Value Store (キー - 値ストアを検査)] を選択します。このツールでは、ストアを検



索して、キー名の一部または全部がないかを調べることができます。一致する値があった場合は、それらの値がすべて返されます。



 **例:** 次に iOS Swift の例を示します。

```
...
writeToKv(value: "Joe", key: "Trader")
...

func writeToKv(value: String, key: String) {
 if let kv = KeyValueEncryptedFileStore.shared(
 withName: "testShared") {
 if kv.saveValue(value, forKey: key) {
 let numEntries = kv.count()
 SalesforceLogger.d(RootViewController.self,
 message: "\nValue added: \(value), " +
 "Number of entries: \(numEntries)")
 }
 }
}
```

関連トピック:

[アプリケーション内開発者サポート](#)

最近、ユーザーインターフェースでは、ダーク配色がよく使用されるようになりました。ダーク配色は、目の疲れを軽減できるほか、画面の読みやすさも向上するためです。最新バージョンの iOS および Android では、それぞれダークモードおよびダークテーマがサポートされるようになりました。こうした動向に対応するため、Mobile SDK 8.0 では、ネイティブ UI 要素のダークオプションが追加されました。

ダーク配色の影響を受ける Mobile SDK ダイアログボックスは、次のとおりです。

- 接続を選択 (サーバーピッカー)
- ユーザーの切り替え
- パスコードと生体認証入力

また、MobileSyncExplorer サンプルアプリケーションとネイティブテンプレートアプリケーションでもダーク UI がサポートされるようになりました。

ダーク設定は、次のように管理されます。

## iOS

Mobile SDK 8.2 以前の場合、ダークモードは、モバイルデバイスの iOS 設定に常に従います。アプリケーションがダークモードにまだ対応していない場合は、この機能を静的に無効にできます。これを行うには、アプリケーションの `Info.plist` ファイルを開き、`UIUserInterfaceStyle` キーを「Light」に設定します。この設定を使用するのは、アプリケーションのリソースを更新してダークモード対応にするためにさらに時間が必要な場合のみにするようにと Apple から強く推奨されています。

「[Choosing a Specific Interface Style for Your iOS App \(iOS アプリケーションの特定のインターフェーススタイルの選択\)](https://developer.apple.com/documentation/uikit/uiuserinterfacestyle)」(developer.apple.com) を参照してください。

Mobile SDK 8.3 では、`userInterfaceStyle` プロパティが `SFSDKWindowManager` に導入され、ユーザーインターフェースモードを実行時に変更することができます。この設定は、Mobile SDK が管理するすべてのウィンドウに次のように適用されます。

- ダーク、ライト、または未定義のモードを選択できます。デフォルトでは、Mobile SDK は未定義を使用します。
- アプリケーションが `plist` の `UIUserInterfaceStyle` エントリを指定し、さらに `userInterfaceStyle` プロパティも設定した場合は、プロパティが優先されます。
- このプロパティには、ベース SDK iOS 13 以降が必要です。

 **例:** この例では、iOS のダークテーマ API が使用されています。

```
// Swift
SFSDKWindowManager.shared().userInterfaceStyle = .dark
```

```
// Objective-C
[SFSDKWindowManager sharedManager].userInterfaceStyle =
UIUserInterfaceStyleDark;
```

## Android

Mobile SDK アプリケーションのダークテーマは、次のように決定されます。

- API 29 以降: モバイルデバイスの OS 設定にデフォルト設定されます。
- API 28 以前: オフにデフォルト設定されます。
- API 23 ~ 29: アプリケーションがダークテーマを強制的にオンまたはオフにできます。

`SalesforceSDKManager` クラスで Mobile SDK は、ダークテーマを照会および切り替える API を提供しています。ダークオプションがアプリケーションの既存の配色に合わない場合は、Mobile SDK リソースを明るい設定に戻すことができます。`Theme` enum では、状態を次のように定義します。

```
public enum Theme {
 LIGHT,
 DARK,
 SYSTEM_DEFAULT
}
```



**例:** この例では、Android のダークテーマ API が使用されています。

テーマ設定を照会するには、次のメソッドを使用します。

```
public boolean isDarkTheme()
```

ダークテーマを強制的にオンまたはオフにするには、`setTheme()` メソッドを使用します。次の例では、ダークテーマがオフにされています。

```
SalesforceSDKManager.getInstance().setTheme(SalesforceSDKManager.Theme.LIGHT);
```

## トピック:

- Experience Cloud サイトと Mobile SDK アプリケーション
- 「API の有効化」プロフィールを設定する
- 権限セットを設定する
- API アクセス権をユーザーに付与する
- ログインエンドポイントを設定する
- Experience Cloud サイトのブランド設定
- Experience Cloud サイトでのログイン、セルフ登録、およびパスワード管理のカスタマイズ
- ブランド名が入ったログインページの使用
- Experience Cloud サイトでの外部認証の使用
- 例: Mobile SDK アプリケーションからアクセスできるように Experience Cloud サイトを設定する
- 例: Facebook 認証できるように Experience Cloud サイトを設定する

ExperienceCloud サイトには、[Salesforce の制限](#)で許可されている百万人までのユーザーを含めることができます。適切に設定すれば、顧客は、ExperienceCloud サイトログイン情報を使用して、Mobile SDK アプリケーションにアクセスできます。ExperienceCloud サイトやログイン画面をブランド設定することもできます。

Experience Cloud サイトについての詳細は、Salesforce ヘルプの「[Experience Cloud サイトの設定および管理](#)」を参照してください。

## Experience Cloud サイトと Mobile SDK アプリケーション

Experience Cloud サイトのメンバーが Mobile SDK アプリケーションにログインできるようにするには、Salesforce で適切に権限を設定し、サイトの URL を認識するようにアプリケーションのログインサーバー設定を変更します。

Experience Cloud サイトでは、指定のメンバーが Mobile SDK アプリケーションを使用して Salesforce にアクセスできます。Experience Cloud サイトの独自のログインエンドポイントを定義し、Salesforce を使用して、仕様に従ってブランド設定されたサイトのログインページを作成します。また、一般的なプロバイダーのリストから認証プロバイダーおよび SAML ID プロバイダーを選択することもできます。

Experience Cloud サイトのメンバーシップは、プロファイルと権限セットで決まります。サイトのメンバーが Mobile SDK アプリケーションを使用できるようにするには、次のように設定します。

- 各 Experience Cloud サイトメンバーに「API の有効化」権限があることを確認します。この権限は、プロファイルまたは権限セットを使用して設定できます。
- 「API の有効化」プロファイルまたは権限セットを含めるように Experience Cloud サイトを設定します。
- Experience Cloud サイトのログインエンドポイントを使用するように Mobile SDK アプリケーションを設定します。

これらの手順の概要の他に、必要な手順を実行してユーザーを適切に設定する必要があります。「例: Mobile SDK アプリケーションからアクセスできるように Experience Cloud サイトを設定する」では、Mobile SDK アプリケーションの Experience Cloud サイトプロセスについて説明しています。Experience Cloud サイトについての詳細は、Salesforce ヘルプを参照してください。

 **メモ:** Experience Cloud サイトのログインは、iOS や Android のネイティブおよびハイブリッドローカル Mobile SDK アプリケーションでサポートされています。現在、Visualforce を使用するハイブリッドリモートアプリケーションではサポートされていません。

## 「API の有効化」プロファイルを設定する

Experience Cloud サイトをはじめて使用する場合は、まず組織でデジタルエクスペリエンスを有効にします。「[デジタルエクスペリエンスの有効化](#)」を参照してください。ドメイン名を作成するように求められた場合、ドメイン名に SSL (https://) を使用しないでください。

Experience Cloud サイトを設定するには、「[Experience Cloud サイトの作成](#)」を参照してください。デジタルエクスペリエンスを有効にしたときに作成したドメイン名に基づいて、サイト URL を定義します。

次に、「API の有効化」権限のあるプロファイルを 1 つ以上設定します。これらのプロファイルを使用して、Experience Cloud サイトのメンバーの Mobile SDK アプリケーションを有効にできます。詳細は、「[例: Mobile SDK アプリケーションからアクセスできるように Experience Cloud サイトを設定する](#)」のチュートリアルを参照してください。

1. 新しいプロファイルを作成するか、既存のプロファイルを編集します。
2. プロファイルを詳細を編集して、[システム管理者権限] の [API の有効化] を選択します。
3. 変更を保存して、[設定] から、[クイック検索] ボックスに「デジタルエクスペリエンス」と入力し、[すべてのサイト] を選択して、Experience Cloud サイトを編集します。

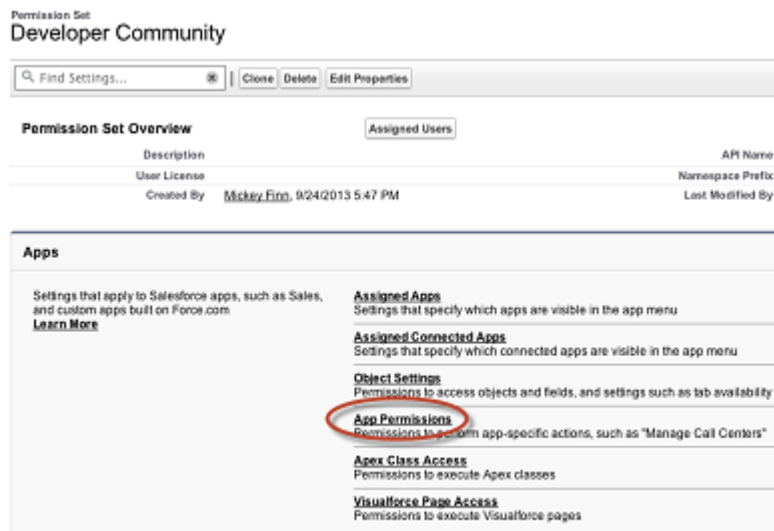
4. サイト名の横にある [ワークスペース] を選択します。次に [管理] > [メンバー] をクリックします。
5. 「API の有効化」プロファイルを [選択済みプロファイル] に追加します。

これで、これらのプロファイルが割り当てられているユーザーに API アクセス権が付与されました。プロファイルの概要は、Salesforce ヘルプの「[ユーザープロファイルの概要](#)」を参照してください。

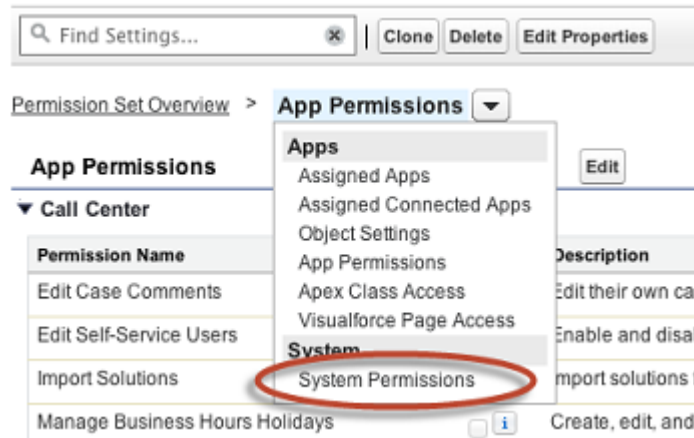
## 権限セットを設定する

権限セットを使用して、Experience Cloud サイトのモバイルアプリケーションを有効にすることもできます。

1. 「API の有効化」権限を既存の権限セットに追加するには、[設定] から、[クイック検索] ボックスに「権限セット」と入力し、[権限セット] を選択して権限セットを選択し、ステップ 6 に進みます。
2. 権限セットを作成するには、[設定] から、[クイック検索] ボックスに「権限セット」と入力し、[権限セット] を選択します。
3. [新規] をクリックします。
4. 権限セットに表示ラベルを指定して、*Return* を押すと、自動的に API 名が作成されます。
5. [次へ] をクリックします。
6. [アプリケーション] セクションで、[アプリケーション権限] をクリックします。



7. [アプリケーション権限] をクリックして [システム] > [システム権限] を選択します。



8. [システム権限] ページで [編集] をクリックして [API の有効化] を選択します。
  9. [保存] をクリックします。
  10. [設定] から、[クイック検索] ボックスに「デジタルエクスペリエンス」と入力し、[すべてのサイト] を選択して、コミュニティ名の横にある [ワークスペース] をクリックします。
  11. [管理] で、[メンバー] をクリックします。
  12. [権限セットを選択] で、「API の有効化」権限セットを [選択済みの権限セット] に追加します。
- これで、この権限セットのユーザーに API アクセス権が付与されました。

## API アクセス権をユーザーに付与する

API アクセス権を Experience Cloud サイトユーザーに拡張するには、「API の有効化」権限を設定するプロファイルまたは権限セットにコミュニティユーザーを追加します。この権限を含むプロファイルまたは権限セットをまだ設定していない場合は、「[「API の有効化」プロファイルを設定する](#)」および「[権限セットを設定する](#)」を参照してください。

## ログインエンドポイントを設定する

最後に、Experience Cloud サイトのログインエンドポイントを使用するようにアプリケーションを設定します。アプリケーションのモバイルプラットフォームにより、この設定方法が決まります。

### Android

Android では、ログインホストはサーバー接続と呼ばれます。SalesforceSDK プロジェクトの `res/xml/servers.xml` ファイルでサーバー接続の標準リストを参照できます。Mobile SDK では、このファイルを使用して本番サーバーと Sandbox サーバーを定義します。ネイティブ Android プロジェクトで独自の `res/xml/servers.xml` ファイルを作成することで、カスタムサーバーをランタイムリストに追加できます。servers.xml ファイルの最初にリストされているサーバーがデフォルトのログインサーバーとしてアプリケーションの起動時に使用されます。servers.xml のルート XML 要素は `<servers>` です。このルートには、`<server>` エントリをいくつでも含めることができます。各 `<server>` エントリには、name (わかりやすい任



意の表示ラベル) と url (「https://」プレフィックスが含まれるログインサーバーの Web アドレス) という 2 つの属性が必要です。

次に例を示します。

```
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <server name="XYZ.com Login" url="https://myloginserver.cloudforce.com"/>
</servers>
```

## iOS

バージョン 4.1 より前の Mobile SDK for iOS アプリケーションでは、アプリケーションの設定バンドルでカスタムログイン URI を定義していました。Mobile SDK 4.1 以降では、iOS アプリケーションの設定バンドルがなくなりました。代わりに、アプリケーションの info.plist ファイルの SFDCOAuthLoginHost プロパティを使用して、カスタムログイン URI を組み込むことができます。

顧客は、アプリケーションのランタイムで独自のカスタムログインホストを設定することもできます。手順は、次のとおりです。

1. ログインしないでアプリケーションを開始します。
2. ログイン画面で [設定] をタップするか、上部のナビゲーションバーの「歯車」アイコン  をタップします。
3. [接続を選択] 画面で、プラスアイコン  をタップします。
4. (省略可能ですが推奨) 今後アクセスしたときにこの設定を簡単に識別できるように、表示ラベルを入力します。
5. カスタムログインホストの URI を入力します。必ず https:// プレフィックスを削除してください。次に、Experience Cloud サイトの一般的な URI の入力方法の例を示します。

```
MyDomainName.my.site.com/fineapps
```

## Experience Cloud サイトのブランド設定

Salesforce タブ + Visualforce テンプレートを使用している場合は、エクスペリエンスワークスペースで Experience Cloud サイトのデザインをカスタマイズできます。会社のロゴ、色、著作権を追加してカスタマイズを行うことができます。これらの要素をカスタマイズすると、Experience Cloud サイトが会社のブランド設定と一致し、サイトのメンバーがただちに認識できるようになります。

 **警告:** Mobile SDK ではエクスペリエンスビルダーサイトをラップするアプリケーションの作成方法がサポートされていません。

1. [エクスペリエンスワークスペース](#)を開きます。
2. [管理]>[ブランド]をクリックします。
3. ルックアップを使用して、Experience Cloud サイトのヘッダーとフッターを選択します。  
ヘッダーとフッター用に選択するファイルは、[ドキュメント] タブにアップロード済みで、公開されている必要があります。ヘッダーには、.html、.gif、.jpg、または.pngを使用できます。フッターは、.html ファイ

ルである必要があります。.html ファイルの合計最大サイズは 100 KB です。.gif、.jpg、または .png ファイルの最大サイズは 20 KB です。ヘッダーの .html ファイルが 70 KB で、.html ファイルをフッターに使用するとします。フッターの .html ファイルは、30 KB までです。

選択したヘッダーは、グローバルヘッダーの下にある Salesforce ロゴに置き換わります。選択したフッターは、Salesforce 標準の著作権とプライバシーのフッターに置き換わります。

4. 定義済みの配色から選択する場合は、[配色を選択] をクリックします。カラーピッカーから色を選択する場合は、ページ選択項目の横にあるテキストボックスをクリックします。

選択した色の中には、Experience Cloud サイトのログインページと、Salesforce モバイルアプリケーションのサイトの外観にも影響を与えるものもあります。

| 色の選択肢  | 表示場所                                                                                                                |
|--------|---------------------------------------------------------------------------------------------------------------------|
| ヘッダー背景 | 黒のグローバルヘッダーの下のページ上部。[ヘッダー] 項目で HTML ファイルを選択した場合は、この色が上書きされます。<br>ログインページ上部。<br>Salesforce モバイルアプリケーションの [ログイン] ページ。 |
| ページ背景  | ログインページを含む、Experience Cloud サイトのすべてのページの背景色。                                                                        |
| 第 1    | 選択されたタブ。                                                                                                            |
| 第 2    | リストとテーブルの上境界線。<br>ログインページのボタン。                                                                                      |
| 第 3    | 編集と詳細ページのセクションヘッダーの背景色。                                                                                             |

5. [保存] をクリックします。

## Experience Cloud サイトでのログイン、セルフ登録、およびパスワード管理のカスタマイズ

Experience Cloud サイトの標準のログイン、ログアウト、パスワード管理、およびセルフ登録オプションを設定したり、Apex および Visualforce の動作をカスタマイズしたりします。

 **警告:** Mobile SDK ではエクスペリエンスビルダーサイトをラップするアプリケーションの作成方法がサポートされていません。

デフォルトでは、各 Experience Cloud サイトには、デフォルトのログイン、パスワード管理、およびセルフ登録ページと、内部でこの機能を実行する関連 Apex コントローラーが付属します。Visualforce または Apex を使用して、カスタムのブランド設定を作成したり、デフォルトの動作を変更したりすることができます。Salesforce ヘルプの次の手順を参照してください。

- [Experience Cloud サイトのログインページをブランド設定する。](#)

- [Experience Cloud サイトのログイン環境をカスタマイズする](#) (デフォルトのログインページ動作の変更、カスタムログインページの使用、および他の認証プロバイダーのサポート)。
- [ログアウト時にユーザーを異なる URL にリダイレクトする](#)。
- [\[パスワードの変更\] および \[パスワードを忘れた場合\] カスタムページを使用する](#)。
- [セルフ登録を設定する](#) (Experience Cloud サイトでライセンスのないゲストユーザーに対応するため)。

## ブランド名が入ったログインページの使用

Mobile SDK 5.2 以降は、クライアントアプリケーションに、ブランド名が入った Experience Cloud サイトのログインページを表示できます。

通常、ブランド名が入ったログインページの認証 URL はこの例のようになります。

```
https://MyDomainName.my.site.com/services/oauth2/authorize/<brand>?response_type=code&...
```

この URL の <brand> は、アプリケーションで再利用するブランド設定パラメーターです。この値は次のメソッドを使用して設定します。loginBrand は、Experience Cloud サイトのログインページのブランド設定パラメーターです。

### Android

```
SalesforceSDKManager.getInstance().setLoginBrand(brandedLoginPath);
```

### iOS

#### Swift

```
SalesforceManager.shared.brandLoginIdentifier = loginBrand
```

#### Objective-C

```
[SalesforceSDKManager sharedManager].brandLoginPath = loginBrand;
```

## Experience Cloud サイトでの外部認証の使用

Facebook®などの外部認証プロバイダーを使用して、Experience Cloud サイトユーザーを Mobile SDK アプリケーションにログインさせることができます。

- ❏ **メモ:** Salesforce では、Janrain を認証プロバイダーとして使用できますが、これは基本的に Salesforce で内部的に使用することを目的としています。ここで Janrain を説明した理由は、すべての情報を網羅するためです。

## Salesforce が証明書利用者である認証プロバイダー SSO

認証プロバイダーにより、ユーザーはサードパーティから提供されたログイン情報を使用して、シングルサインオン (SSO) で Salesforce 組織や Experience Cloud サイトにログインできます。認証プロバイダーは、保護されているサードパーティデータへのアクセスも承認します。Salesforce で認証プロバイダーを設定するには、OpenID Connect やカスタム OAuth 2.0 設定を使用するなど、いくつかの方法があります。使用できるプロトコルは、サードパーティによって異なります。

認証プロバイダーを設定するには、いくつかの方法があります。

- 定義済み認証プロバイダー
- Salesforce が管理する認証プロバイダー
- OpenID Connect 認証プロバイダー
- カスタム認証プロバイダー

Salesforce で認証プロバイダーを設定したら、その認証プロバイダーを Salesforce のログインページまたは Experience Cloud のログインページに追加できます。

### シングルサインオン認証と承認フロー

ほとんどの認証プロバイダーには2つの目的があります。SSOでのユーザー認証に加えて、ユーザーデータへのアクセス権の提供です。SSOでログインした後、このサードパーティデータへのアクセス権を使用して、ユーザーの Salesforce プロファイルを追加の情報で強化できます。

たとえば、あるユーザーが Facebook ログイン情報を使用して Salesforce にログインした場合、そのユーザーは Facebook データへのアクセスを承認できます。

Facebook データへのアクセスが承認された場合、Facebook は Salesforce にアクセストークンを送信します。このアクセストークンを使用して Facebook プロファイルデータにアクセスすることで、ユーザーの Salesforce ユーザープロファイルにデータを追加できます。

#### エディション

使用可能なインターフェース: Lightning Experience および Salesforce Classic

使用可能なエディション: Enterprise Edition、Performance Edition、Unlimited Edition、および Developer Edition

#### ユーザ権限

設定を参照する

- 「設定・定義の参照」

設定を編集する

- 「アプリケーションのカスタマイズ」

および

「認証プロバイダーの管理」

## 認証後にユーザーを Experience Cloud サイトに誘導

認証プロバイダーを使用してシングルサインオン (SSO) を設定する場合は、Experience Cloud サイト URL 要求パラメーターを使用して、ユーザーを認証後に特定のサイトに転送します。サイトパラメーターを使用することで、ユーザーを Salesforce 組織または Experience Cloud サイトのどちらにログインさせるかを指定できます。このパラメーターでは、登録ハンドラーが作成するユーザーの種別も変更できます。

たとえば、Google 認証プロバイダーを設定し、自分の Salesforce 組織を証明書利用者として SSO を設定するとします。Google のログイン情報でログインしたサイトユーザーはカスタムサイトに転送されるようにするために、`site` パラメーターを SSO クライアント設定 URL に追加します。ユーザーが組織のログインページを表示したら、Google のログインボタンをクリックし、Google のログイン情報を入力します。Google 認証が完了すると、Salesforce によってユーザーがカスタムサイトにリダイレクトされます。

`site` パラメーターを [シングルサインオン初期化 URL] に追加すると、サイトユーザーはログイン後にサイトに転送されます。このパラメーターを [既存ユーザーをリンクする URL] に追加した場合は、確認ページに [Salesforce に進む] リンクが表示され、ユーザーがこのリンクをクリックするとサイトに転送されます。

`site` パラメーターを追加しない場合、Salesforce は認証完了後にユーザーを `/home/home.jsp` (ポータルまたは標準アプリケーションの場合) またはデフォルトのサイトページ (サイトの場合) に転送します。

 **例:** 認証プロバイダーを作成すると、初期化 URL とコールバック URL は適切な [私のドメイン] ログイン URL またはサイト URL を指します。次の例の `URLsuffix` は、認証プロバイダーを定義したときに指定した値です。

```
https://MyDomainName.my.salesforce.com/services/auth/sso/URLsuffix
```

### エディション

使用可能なインターフェース: Lightning Experience および Salesforce Classic

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、**Unlimited** Edition、および **Developer** Edition

### ユーザー権限

設定を参照する

- 「設定・定義の参照」

設定を編集する

- 「アプリケーションのカスタマイズ」

および

「認証プロバイダーの管理」

## 証明書利用者のデータ要求のカスタマイズ

認証プロバイダーを使用してシングルサインオン (SSO) を設定する場合に、scope パラメーターを使用することで、サードパーティ (Facebook など) のデータへの要求をカスタマイズできます。たとえば、ユーザーの Facebook プロファイルに記載されているメールアドレスへのアクセスを要求できます。このパラメーターは、Janrain 以外のすべての認証プロバイダーで使用できます。

認証プロバイダー SSO フローでは、scope パラメーターで指定する範囲によって、証明書利用者が要求できるデータの種別を定義します。ユーザーがログインすると、証明書利用者は認証要求を送信します。サードパーティは、ユーザーを検証し、範囲が指定されたアクセストークンを返します。ユーザーが範囲で定義されるデータへのアクセスを承認すると、証明書利用者は要求したサードパーティデータにアクセスできるようになります。

たとえば、Google 認証プロバイダーを設定し、自分の Salesforce 組織を証明書利用者として SSO を設定するとします。そして、ユーザーが自分の Google ドライブのデータを Salesforce 組織で表示できるようにします。そのため、scope パラメーターを SSO クライアント設定 URL に追加し、要求した範囲が指定されたアクセストークンを取得するための Auth.AuthToken メソッドを実装します。ユーザーが組織にログインすると、Google にリダイレクトされて認証され、Salesforce から Google ドライブへのアクセスを承認します。その後、ユーザーの Google ドライブデータを Salesforce で表示できるようになります。

 **メモ:** 一部のサードパーティでは、要求するデータの範囲を事前に登録する必要があります。

ユーザーが指定する範囲に加えて、認証プロバイダーにはデフォルトの範囲も用意されています。デフォルトの範囲はサードパーティによって異なりますが、一般に、基本的なユーザー情報へのアクセスは制限されています。たとえば、Salesforce のデフォルト範囲は id であり、ユーザーの ID へのアクセスを許可します。デフォルトの範囲を上書きするには、範囲をスペースで区切って指定した文字列をサードパーティに送信します。

1. scope パラメーターをクライアント設定 URL に追加します。
2. Apex の Auth.AuthToken メソッドを使用してアクセストークンを取得します。詳細は、『Apex リファレンスガイド』の「AuthToken クラス」を参照してください。

 **例:** 次に、[シングルサインオン初期化 URL] に追加された、Salesforce の範囲 api および web を要求する scope パラメーターの例を示します。ここでは、Salesforce の api 範囲により、証明書利用者は Connect REST API リソースにアクセスできます。また、web 範囲により、証明書利用者はアクセストークンを Web 上で使用できます。

`https://login.salesforce.com/services/auth/sso/orgID/URLsuffix?scope=id+api+web`

- orgID は、認証プロバイダー ID です。
- URLsuffix は、認証プロバイダーを定義したときに指定した値です。

有効な範囲はサードパーティによって異なるため、サードパーティのドキュメントを参照してください。Salesforce は次の範囲をサポートします。

### エディション

使用可能なインターフェース: Lightning Experience および Salesforce Classic

使用可能なエディション: Enterprise Edition、Performance Edition、Unlimited Edition、および Developer Edition

### ユーザ権限

設定を参照する

- 「設定・定義の参照」

設定を編集する

- 「アプリケーションのカスタマイズ」

および

「認証プロバイダーの管理」



| 値                                                                     | 説明                                                                                                                                                                        |
|-----------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Customer Data Platform</b> データに対する ANSI SQL クエリを実行 (cdp_query_api) | ユーザーの代わりに Data Cloud データの ANSI SQL クエリを許可します。                                                                                                                             |
| <b>Pardot</b> サービスを管理 (pardot_api)                                    | ユーザーの代理として Marketing Cloud Account Engagement API サービスへのアクセスを許可します。Account Engagement でアクセス可能なすべてのサービスを管理します。(Pardot は Marketing Cloud Account Engagement という名前に変更されました。) |
| <b>Customer Data Platform</b> プロファイルデータを管理 (cdp_profile_api)          | Data Cloud REST API データへのアクセスを許可します。この範囲を使用して、プロファイルレコードを管理します。                                                                                                           |
| <b>Connect REST API</b> リソースにアクセス (chatter_api)                       | ユーザーの代理として Connect REST API リソースへのアクセスを許可します。                                                                                                                             |
| <b>Customer Data Platform</b> 取り込み API データを管理 (cdp_ingest_api)        | Data Cloud 取り込み API データへのアクセスを許可します。この範囲を使用して、Data Cloud で外部データセットをアップロードおよび管理します。この範囲は JSON Web トークン (JWT) でパッケージ化されます。                                                  |
| <b>Analytics REST API Charts Geodata</b> リソースにアクセス (eclair_api)       | Analytics REST API Charts Geodata リソースへのアクセスを許可します。                                                                                                                       |
| <b>Analytics REST API</b> リソースにアクセス (wave_api)                        | Analytics REST API リソースへのアクセスを許可します。                                                                                                                                      |
| <b>API</b> を使用してユーザーデータを管理 (api)                                      | REST API や Bulk API 2.0 などの API を使用して、現在のログインユーザーのアカウントへのアクセスを許可します。この範囲には、Connect REST API リソースへのアクセスを許可する chatter_api も含まれます。                                           |
| <b>カスタム権限</b> にアクセス (custom_permissions)                              | 接続アプリケーションに関連付けられた組織内のカスタム権限へのアクセスを許可します。この範囲は、現在のユーザーの各権限が有効になっているかどうかを示します。                                                                                             |
| <b>ID URL</b> サービスにアクセス (id, profile, email, address, phone)          | ID URL サービスへのアクセスを許可します。profile、email、address、phone を要求すれば、id を使用した場合と同じ結果を個別に得られます。これは、これらが同義であるためです。                                                                    |
| <b>Lightning</b> アプリケーションにアクセス (lightning)                            | ハイブリッドアプリケーションで OAuth 2.0 ハイブリッドアプリケーショントークンフローおよびハイブリッドアプリケーション更新トークンフローを介して Lightning の子セッションを直接取得できるようにします。                                                            |



| 値                                         | 説明                                                                                                                                                                                                                                        |
|-------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| コンテンツリソースにアクセス (content)                  | ハイブリッドアプリケーションで OAuth 2.0 ハイブリッドアプリケーショントークンフローおよびハイブリッドアプリケーション更新トークンフローを介してコンテンツの子セッションを直接取得できるようにします。                                                                                                                                  |
| 一意のユーザー識別子にアクセス (openid)                  | <p>OpenID Connect アプリケーションの現在のログインユーザーの一意の識別子へのアクセスを許可します。</p> <p>OAuth 2.0 ユーザーエージェントフローと OAuth 2.0 Web サーバーフローでは、<code>openid</code> 範囲を使用します。この範囲を使用すると、アクセストークンの他にも、「<a href="#">OpenID Connect の仕様</a>」に適合する署名付き ID トークンを受信できます。</p> |
| フルアクセス (full)                             | <p>ログインユーザーがアクセスできるすべてのデータへのアクセスを許可し、その他すべての範囲が対象となります。</p> <p><code>full</code> は更新トークンを返しません。更新トークンを取得するには、<code>refresh_token</code> の範囲を明示的に要求する必要があります。</p>                                                                          |
| いつでも要求を実行 (refresh_token, offline_access) | 更新トークンを受け取る資格が要求元クライアントにある場合に、更新トークンを返すことを許可します。更新トークンにより、ユーザーがオフラインのときにアプリケーションはユーザーのデータを操作できます。このトークンは、 <code>offline_access</code> を要求した場合と同じ意味になります。                                                                                  |
| Visualforce アプリケーションにアクセス (visualforce)   | <p>顧客が作成した Visualforce ページへのアクセスのみを許可します。この範囲では標準の Salesforce UI へのアクセスは許可されません。</p> <p>ハイブリッドアプリケーションで Visualforce の子セッションを直接取得できるようにするには、この範囲を OAuth 2.0 ハイブリッドアプリケーショントークンフローまたはハイブリッドアプリケーション更新トークンフローに含めます。</p>                      |
| Web ブラウザーを使用してユーザーデータを管理 (web)            | Web で <code>access_token</code> を使用することを許可します。この範囲には <code>visualforce</code> も含まれ、顧客が作成した Visualforce ページへのアクセスが許可されます。                                                                                                                  |
| チャットボットサービスにアクセス (chatbot_api)            | Einstein ボット API サービスへのアクセスを許可します。                                                                                                                                                                                                        |

| 値                                             | 説明                                                                                        |
|-----------------------------------------------|-------------------------------------------------------------------------------------------|
| ヘッドレス登録 API にアクセス (user_registration_api)     | ヘッドレス登録フローの API へのアクセスを許可します。認証を要求するようにフローを設定する場合、この範囲が含まれるアクセストークンを渡す必要があります。            |
| パスワードを忘れた場合のヘッドレス API にアクセス (forgot_password) | ヘッドレスのパスワードを忘れた場合のフローの API へのアクセスを許可します。認証を要求するようにフローを設定する場合、この範囲が含まれるアクセストークンを渡す必要があります。 |
| すべての Data Cloud API リソースにアクセス (cdp_api)       | すべての Data Cloud API リソースへのアクセスを許可します。                                                     |
| Salesforce API プラットフォームにアクセス (sfap_api)       | 将来の使用のために予約されています。                                                                        |
| インタラクション API リソースにアクセス (interaction_api)      | 将来の使用のために予約されています。                                                                        |

## Facebook 認証プロバイダーの設定

Facebook 認証プロバイダーを設定し、ユーザーが Facebook ログイン情報を使用して Salesforce にログインできるようにします。

Facebook を認証プロバイダーとして設定するには、次の手順を実行します。

1. Facebook アプリケーションを設定し、Salesforce をアプリケーションのドメインにします。
2. Salesforce で Facebook 認証プロバイダーを定義します。
3. Salesforce によって生成されるコールバック URL を Facebook の Web サイト URL として使用するよう、Facebook アプリケーションを更新します。
4. 接続をテストします。
5. ログインページに Facebook プロバイダーを追加します。

## Facebook アプリケーションの設定

Salesforce で Facebook を設定する前に、Facebook でアプリケーションを設定する必要があります。

 **メモ:** Salesforce に独自のデフォルトアプリケーションの使用を許可すると、この手順を省略できます。詳細は、「[Salesforce 管理認証プロバイダーの使用](#)」を参照してください。

1. Facebook の Web サイトに移動して、アプリケーションを作成します。
2. アプリケーション設定を変更し、アプリケーションドメインを Salesforce に設定します。

### エディション

使用可能なインターフェース: Lightning Experience および Salesforce Classic

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、**Unlimited** Edition、および **Developer** Edition

### ユーザ権限

設定を参照する

- 「設定・定義の参照」

設定を編集する

- 「アプリケーションのカスタマイズ」
- および
- 「認証プロバイダーの管理」

3. アプリケーション ID とアプリケーションの秘密をメモします。

## Salesforce での Facebook プロバイダーの定義

Facebook プロバイダーを設定するには、Facebook アプリケーションの ID とアプリケーションの秘密に関する情報が必要です。

生成された [認証プロバイダー ID] の値をメモします。この値は `Auth.AuthToken Apex` クラスで使します。

 **メモ:** Salesforce に値の管理を許可することで、この手順を省略できます。詳細は、「[Salesforce 管理認証プロバイダーの使用](#)」を参照してください。

1. [設定]の[クイック検索]ボックスに「認証プロバイダー」と入力して、[認証プロバイダー]>[新規]を選択します。
2. [プロバイダータイプ]に[Facebook]を選択します。
3. プロバイダーの名前を入力します。
4. クライアント設定 URL に使用する URL サフィックスを入力します。たとえば、プロバイダーの URL サフィックスが MyFacebookProvider である場合、シングルサインオン (SSO) URL は `https://mydomain_url または site_url/services/auth/sso/MyFacebookProvider` のようになります。
5. [コンシューマー鍵]に Facebook アプリケーション ID を使します。
6. [コンシューマーの秘密]に Facebook アプリケーションの秘密を使します。
7. 必要に応じて、次の項目を設定して作業内容を保存します。
  - [承認エンドポイント URL] に、Facebook から入手したベース URL を入力します。たとえば、`https://www.facebook.com/v2.2/dialog/oauth` です。この項目を空白のままにした場合は、アプリケーションが使用する Facebook API のバージョンが使されます。
  -  **ヒント:** 必要に応じて、クエリ文字列パラメーターをベース URL に追加できます。たとえば、オフラインアクセス用に Facebook から更新トークンを取得するには、`https://accounts.facebook.com/o/oauth2/auth?access_type=offline&approval_prompt=force` を使します。Facebook で最初の更新トークンの後にも更新トークンが引き続き提供されるように、更新アクションの承認をユーザーに求めるには `approval_prompt` パラメーターが必要となります。
  - [トークンエンドポイント URL] に、Facebook から入手した URL を入力します。たとえば、`https://www.facebook.com/v2.2/dialog/oauth` です。この項目を空白のままにした場合は、アプリケーションが使用する Facebook API のバージョンが使されます。
  - Facebook のプロファイル API から要求された値を変更するには、[ユーザー情報エンドポイント URL] を入力します。詳細は、[https://developers.facebook.com/docs/facebook-login/permissions/v2.0#reference-public\\_profile](https://developers.facebook.com/docs/facebook-login/permissions/v2.0#reference-public_profile) を参照してください。要求された項目は、要求された範囲に対応する必要があります。この項目を空白のままにした場合は、アプリケーションが使用する Facebook API のバージョンが使されます。
  - セキュリティを強化する OAuth 2.0 Proof Key for Code Exchange (PKCE) 拡張を自動的に有効にするには、[Use Proof Key for Code Exchange (PKCE) Extension (Proof Key for Code Exchange (PKCE) 拡張を使用)]を選択します。この設定がどのようにプロバイダーを保護するかについての詳細は、「[実装への Proof Key for Code Exchange \(PKCE\) 拡張の組み込み](#)」を参照してください。

- [デフォルトの範囲] に、認証エンドポイントへの要求と共に送信する範囲を入力します。送信しない場合は、プロバイダータイプのハードコード化されたデフォルトが使用されます。デフォルトについては、[Facebook の開発者用ドキュメント](#)を参照してください。  
詳細は、「[Scope URL パラメーターの使用](#)」を参照してください。
- コンシューマー鍵とコンシューマーの秘密を入力すると、コンシューマーの秘密がデフォルトで SOAP API の応答に含まれます。SOAP API 応答で秘密を表示しないようにするには、[SOAP API 応答にコンシューマーの秘密を含める] の選択を解除してください。2022 年 11 月より、メタデータ API 応答では、秘密が常にプレースホルダーの値に置き換えられます。デプロイ時に、プレースホルダーをプレーンテキストのコンシューマーの秘密に置き換えるか、後から UI で値を変更してください。
- [カスタムエラー URL] に、プロバイダーがエラーの報告に使用する URL を入力します。
- [カスタムログアウト URL] に、ユーザーが SSO フローを使用して認証された場合に、ログアウト後の特定の移動先となる URL を入力します。この項目を使用して、ユーザーをブランド設定されたログアウトページか、またはデフォルトの Salesforce ログアウトページ以外の場所のいずれかに移動します。URL は、http または https プレフィックスを付けて完全修飾する必要があります (<https://acme.my.salesforce.com> など)。  
 **ヒント:** [シングルログアウト \(SLO\)](#) を設定して、Salesforce と ID プロバイダーの両方から自動的にユーザーをログアウトするようにします。Salesforce では、証明書利用者として、ユーザーが ID プロバイダーまたは Salesforce からログアウトした場合でも OpenID Connect SLO がサポートされます。
- 既存の Apex クラスを Registration Handler クラスとして選択します。登録ハンドラーの Apex クラステンプレートを作成するには [登録ハンドラーテンプレートを自動作成] を選択します。このクラスを後で編集してデフォルトのコンテンツを変更してから使用します。  
 **メモ:** Registration Handler クラスは、Salesforce で SSO 初期化 URL を生成するために必要となります。
- [他のアカウントで登録を実行] で、Apex ハンドラークラスを実行するユーザーを選択します。  
[他のアカウントで登録を実行] では、登録ハンドラーが動作するコンテキストが提供されます。既存の登録ハンドラークラスを指定しているか、テンプレートから登録ハンドラークラスを作成しているかに関係なく、ユーザーを選択します。本番環境では、通常、[他のアカウントで登録を実行] ユーザー用にシステムユーザーを作成します。このように、ハンドラーによって実行される操作は、その登録プロセスが簡単に特定されます。たとえば取引先責任者が作成される場合、システムユーザーがそれを作成します。
- プロバイダーでポータルを使用するには、[ポータル] ドロップダウンリストからポータルを選択します。
- [アイコン URL] に、サイトのログインページのボタンとして表示するアイコンのパスを追加します。このアイコンは Experience Cloud サイトにのみ適用されます。Salesforce のログインページまたは [私のドメイン] のログイン URL には表示されません。ユーザーは、このボタンをクリックし、サイトに関連付けられた認証プロバイダーを使用してログインします。  
独自の画像へのパスを指定するか、いずれかのサンプルアイコンの URL を項目にコピーしてください。
- ID プロバイダーの MFA サービスではなく、Salesforce 多要素認証 (MFA) 機能を使用するには、[この SSO プロバイダーで Salesforce MFA を使用] を選択します。この設定は、MFA が直接適用されているユーザーの MFA のみをトリガーします。詳細は、「[SSO での Salesforce MFA の使用](#)」を参照してください。

認証プロバイダーを定義した後、複数のクライアント設定 URL が生成されます。

- **テスト専用初期化 URL** — Salesforce システム管理者は、この URL を使用して、サードパーティプロバイダーが正しく設定されていることを確認します。システム管理者がブラウザでこの URL を開き、サードパーティにサインインすると、属性の対応付けにより Salesforce にリダイレクトされます。
- **シングルサインオン初期化 URL** — この URL を使用して、サードパーティからサードパーティのログイン情報を使用して Salesforce への SSO を実行します。ユーザーは、ブラウザでこの URL を開いて、サードパーティにログインします。このサードパーティは、ユーザーを作成するか既存のユーザーを更新します。次にサードパーティは、ユーザーを Salesforce にそのユーザーとしてサインインします。
- **既存ユーザーをリンクする URL** — この URL を使用して、既存の Salesforce ユーザーをサードパーティアカウントにリンクします。このユーザーはブラウザでこの URL を開いてサードパーティにサインインし、Salesforce にサインインしてリンクを承認します。
- **OAuth 専用初期化 URL** — この URL を使用して、サードパーティの OAuth アクセストークンを取得します。サードパーティサービスがトークンを取得するには、ユーザーが Salesforce で認証する必要があります。このフローでは、以降の SSO 機能は提供されません。
- **コールバック URL** — 認証プロバイダーが設定のためにコールバックするエンドポイントに、この URL を使用します。認証プロバイダーは、各クライアント設定 URL の情報と共に、コールバック URL にリダイレクトする必要があります。

クライアント設定 URL では、ユーザーの特定サイトへのログイン、サードパーティからのカスタマイズされた権限の取得、認証後の特定の場所への移動を行うための、追加の要求パラメーターをサポートしています。

## Facebook アプリケーションの更新

Facebook 認証プロバイダーを Salesforce に定義したら、Facebook に戻り、コールバック URL を Facebook の Web サイト URL として使用するようアプリケーションを更新します。

## SSO 接続のテスト

ブラウザの認証プロバイダーの詳細ページで [テスト専用初期化 URL] を開きます。ユーザーは Facebook にリダイレクトされ、サインインするよう求められます。そしてサインイン時に、アプリケーションを認証するよう求められます。認証後、ユーザーは Salesforce にリダイレクトされます。

## ログインページへの認証プロバイダーの追加

認証プロバイダーをログインオプションとして表示するようにログインページを設定します。組織の SSO を設定しているのか、Experience Cloud サイトの SSO を設定しているかに応じて、このステップは異なります。

- 組織の場合、「[組織のログインページへの認証プロバイダーの追加](#)」を参照してください。
- Experience Cloud サイトの場合、「[Experience Cloud サイトのログインページへの認証プロバイダーの追加](#)」を参照してください。



## Salesforce 認証プロバイダーの設定

ユーザーが Salesforce ログイン情報を使用してカスタム外部 Web アプリケーションにログインできるように Salesforce 認証プロバイダーを設定します。

Salesforce 認証プロバイダーとして設定するには、次の概要手順に従います。

1. 組織の Salesforce 認証プロバイダーを定義します。
2. 接続をテストします。
3. ログインページに認証プロバイダーを追加します。

### 組織の Salesforce 認証プロバイダーの定義

組織の認証プロバイダーを設定するには、接続アプリケーション定義の [コンシューマー鍵] 項目および [コンシューマーの秘密] 項目の値が必要です。

 **メモ:** Salesforce に値の管理を許可することで、この手順を省略できます。詳細は、「[Salesforce 管理認証プロバイダーの使用](#)」を参照してください。

1. [設定] から、[クイック検索] ボックスに「**認証プロバイダー**」と入力し、[認証プロバイダー] > [新規] を選択します。
2. [プロバイダータイプ] に [Salesforce] を選択します。
3. プロバイダーの名前を入力します。
4. クライアント設定 URL に使用する URL サフィックスを入力します。たとえば、プロバイダーの URL サフィックスが MySFDCProvider である場合、SSO URL は `https://mydomain_url または site_url /services/auth/sso/MySFDCProvider` のようになります。
5. 接続アプリケーション定義のコンシューマー鍵の値を [コンシューマー鍵] 項目に貼り付けます。
6. 接続アプリケーション定義のコンシューマーの秘密の値を [コンシューマーの秘密] 項目に貼り付けます。
7. 必要に応じて、次の項目を設定します。
  - [承認エンドポイント URL] に、OAuth 認証の URL を指定します。  
[承認エンドポイント URL] では、ホスト名に Sandbox または会社固有のカスタムドメインのログイン URL を含めることができます。URL は `.salesforce.com` で終わり、パスは `/services/oauth2/authorize` で終わる必要があります。たとえば、「`https://login.salesforce.com/services/oauth2/authorize`」などです。
  - [トークンエンドポイント URL] に、OAuth トークンの URL を指定します。  
[トークンエンドポイント URL] では、ホスト名に Sandbox やカスタムドメイン名を含めることができます。URL は `.salesforce.com` で終わり、パスは `/services/oauth2/token` で終わる必要があります。たとえば、「`https://login.salesforce.com/services/oauth2/token`」などです。
  - セキュリティを強化する OAuth 2.0 Proof Key for Code Exchange (PKCE) 拡張を自動的に有効にするには、[Use Proof Key for Code Exchange (PKCE) Extension (Proof Key for Code Exchange (PKCE) 拡張を使用)] を選択します。この設定がどのようにプロバイダーを保護するかについての詳細は、「[実装への Proof Key for Code Exchange \(PKCE\) 拡張の組み込み](#)」を参照してください。

#### エディション

使用可能なインターフェース: Lightning Experience および Salesforce Classic

使用可能なエディション: Enterprise Edition、Performance Edition、Unlimited Edition、および Developer Edition

#### ユーザ権限

設定を参照する

- 「設定・定義の参照」

設定を編集する

- 「アプリケーションのカスタマイズ」

および

「認証プロバイダーの管理」

- [デフォルトの範囲] に、認証エンドポイントへの要求と共に送信する範囲を入力します。送信しない場合は、ハードコード化されたデフォルトが使用されます。

詳細は、「[証明書利用者のデータ要求のカスタマイズ](#)」を参照してください。

- 認証プロバイダーが Winter '15 リリースより後に作成された場合、[サードパーティアカウントのリンクの ID 組織の組織 ID を含める] オプションは表示されなくなりました。Winter '15 より前は、異なる組織に同じユーザー ID をもつ複数のユーザーがいる場合に、対象組織がそれらのユーザーを区別できませんでした。たとえば、異なる組織にいる同じユーザー ID の 2 人のユーザーは同じユーザーとみなされていました。Winter '15 以降、ユーザー ID には組織 ID が含まれるため、このオプションは不要になりました。認証プロバイダーが旧式の場合は、対象組織で ID を別個に保つためにこのオプションを選択します。このオプションを有効にすると、ユーザーはすべてのサードパーティリンクを再承認する必要があります。リンクは、ユーザーの詳細ページの [サードパーティアカウントのリンク] セクションに一覧表示されます。
  - コンシューマー鍵とコンシューマーの秘密を入力すると、コンシューマーの秘密がデフォルトで SOAP API の応答に含まれます。SOAP API 応答で秘密を表示しないようにするには、[SOAP API 応答にコンシューマーの秘密を含める] の選択を解除してください。2022 年 11 月より、メタデータ API 応答では、秘密が常にプレースホルダーの値に置き換えられます。デプロイ時に、プレースホルダーをプレーンテキストのコンシューマーの秘密に置き換えるか、後から UI で値を変更してください。
  - [カスタムエラー URL] に、プロバイダーがエラーの報告に使用する URL を入力します。
  - [カスタムログアウト URL] に、ユーザーが SSO フローを使用して認証された場合に、ログアウト後の特定の移動先となる URL を入力します。この項目を使用して、ユーザーをブランド設定されたログアウトページか、またはデフォルトの Salesforce ログアウトページ以外の場所のいずれかに移動します。URL は、http または https プレフィックスを付けて完全修飾する必要があります (https://acme.my.salesforce.com など)。
-  **ヒント: シングルログアウト (SLO)** を設定して、Salesforce と ID プロバイダーの両方から自動的にユーザーをログアウトするようにします。Salesforce では、証明書利用者として、ユーザーが ID プロバイダーまたは Salesforce からログアウトした場合でも OpenID Connect SLO がサポートされます。

8. 既存の Apex クラスを Registration Handler クラスとして選択します。または、登録ハンドラーの Apex クラステンプレートを作成するには、[登録ハンドラーテンプレートを自動作成] を選択します。このクラスを後で編集してデフォルトのコンテンツを変更してから使用します。

 **メモ:** Registration Handler クラスは、Salesforce で SSO 初期化 URL を生成するために必要となります。

9. [他のアカウントで登録を実行] で、Apex ハンドラークラスを実行するユーザーを選択します。このユーザーは「ユーザーの管理」権限を持っている必要があります。既存の登録ハンドラークラスを指定しているか、テンプレートから登録ハンドラークラスを作成しているかに関係なく、ユーザーが要です。
10. プロバイダーでポータルを使用するには、[ポータル] ドロップダウンリストからポータルを選択します。
11. [アイコン URL] に、サイトのログインページのボタンとして表示するアイコンのパスを追加します。このアイコンは Experience Cloud サイトにのみ適用されます。Salesforce のログインページまたは [私のドメイン] のログイン URL には表示されません。ユーザーは、このボタンをクリックし、サイトに関連付けられた認証プロバイダーを使用してログインします。

独自の画像へのパスを指定するか、いずれかのサンプルアイコンの URL を項目にコピーしてください。



12. ID プロバイダーの MFA サービスではなく、Salesforce 多要素認証 (MFA) 機能を使用するには、[この SSO プロバイダーで Salesforce MFA を使用] を選択します。この設定は、MFA が直接適用されているユーザーの MFA のみをトリガーします。詳細は、「[SSO での Salesforce MFA の使用](#)」を参照してください。
13. [保存] をクリックします。  
クライアント設定 URL の値を確認します。このコールバック URL は、最後の手順を完了するために必要です。[テスト専用初期化 URL] を使用して、設定を確認します。Auth.AuthToken Apex クラスと共に [認証プロバイダー ID] 値を使用するため、その値もメモします。
14. 以前 [設定] から作成した接続アプリケーション定義に戻ります。コールバック URL の値を認証プロバイダーから [コールバック URL] 項目に貼り付けます。

認証プロバイダーを定義した後、複数のクライアント設定 URL が生成されます。

クライアント設定 URL では、ユーザーの特定サイトへのログイン、サードパーティからのカスタマイズされた権限の取得、認証後の特定の場所への移動を行うための、追加の要求パラメーターをサポートしています。

## SSO 接続のテスト

ブラウザーの認証プロバイダーの詳細ページで [テスト専用初期化 URL] を開きます。認証プロバイダーにリダイレクトされ、サインインするよう求められます。そしてサインイン時に、アプリケーションを認証するよう求められます。認証後、ユーザーは Salesforce にリダイレクトされます。

## ログインページへの認証プロバイダーの追加

- 組織の場合、「[組織のログインページへの認証プロバイダーの追加](#)」を参照してください。
- Experience Cloud サイトの場合、「[Experience Cloud サイトのログインページへの認証プロバイダーの追加](#)」を参照してください。

## OpenID Connect を使用した認証プロバイダーの設定

Salesforce をサードパーティ OpenID プロバイダーの証明書利用者としてシングルサインオン (SSO) を設定するには、OpenID Connect を実装する認証プロバイダーを設定します。この設定により、ユーザーは OpenID プロバイダーから Salesforce にログインできるようになり、Salesforce による保護されたデータへのアクセスを承認できます。

認証プロバイダーは、OpenID Connect プロトコルのサーバー側を実装する任意のサードパーティで設定できます。一般的な OpenID プロバイダーとして、次のようなプロバイダーがあります。

- [Amazon](#)
- [Google](#)
- [PayPal](#)

Salesforce を OpenID プロバイダーの証明書利用者として設定するには、次の手順を実行します。

- アプリケーションを登録して、Salesforce をアプリケーションドメインにします。
- Salesforce の OpenID Connect 認証プロバイダーを定義します。
- Salesforce で生成されたコールバック URL を使用するようにアプリケーションを更新します。
- 接続をテストします。
- ログインページに認証プロバイダーを追加します。

### OpenID プロバイダーでのアプリケーションの登録

Salesforce で認証プロバイダーを定義する前に、Web アプリケーションを OpenID プロバイダーに登録する必要があります。このプロセスは、OpenID プロバイダーによって異なります。たとえば、Google アプリケーションを登録するには、[OAuth 2.0 クライアント ID を作成](#)します。

1. OpenID プロバイダーの Web サイトでアプリケーションを登録します。
2. アプリケーション設定を変更し、アプリケーションドメイン、またはホームページ URL を Salesforce に設定します。
3. OpenID プロバイダーのドキュメントから、次の設定値を取得します。
  - クライアント ID
  - クライアントの秘密
  - 承認エンドポイント URL
  - トークンエンドポイント URL
  - ユーザー情報エンドポイント URL

#### エディション

使用可能なインターフェース: Lightning Experience および Salesforce Classic

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、**Unlimited** Edition、および **Developer** Edition

#### ユーザ権限

設定を参照する

- 「設定・定義の参照」

設定を編集する

- 「アプリケーションのカスタマイズ」

および

「認証プロバイダーの管理」

## Salesforce での認証プロバイダーの定義

生成された [認証プロバイダー ID] の値を確認しておいてください。この値は `Auth.AuthToken` Apex クラスで使用します。

1. [設定] の [クイック検索] ボックスに「認証」と入力して、[認証プロバイダー] を選択します。
2. [新規] をクリックします。
3. [プロバイダータイプ] に [OpenID Connect] を選択します。
4. プロバイダーの名前を入力します。
5. クライアント設定 URL に使用する URL サフィックスを入力します。たとえば、認証プロバイダーの URL 接尾辞が `MyOpenIDConnectProvider` である場合、SSO URL は `https://mydomain_url または site_url /services/auth/sso/OpenIDConnectProvider` のようになります。
6. [コンシューマー鍵] に OpenID プロバイダーから入手したクライアント ID を使用します。
7. [コンシューマーの秘密] に OpenID プロバイダーから入手したクライアントの秘密を使用します。
8. [承認エンドポイント URL] に、OpenID プロバイダーから入手したベース URL を入力します。

 **ヒント:** 必要に応じて、クエリ文字列パラメーターをベース URL に追加できます。たとえば、オフラインアクセス用に Google から更新トークンを取得するには、`https://accounts.google.com/o/oauth2/auth?access_type=offline&approval_prompt=force` を使用します。Google で最初の更新トークンの後にも更新トークンが引き続き提供されるように、更新アクションの承認をユーザーに求めるには `approval_prompt` パラメーターが必要となります。

9. OpenID プロバイダーから入手したトークンエンドポイント URL を入力します。
10. OpenID プロバイダーのユーザー情報エンドポイント URL を入力します。

 **メモ:** OAuth を使用してプロバイダーと API を統合する場合、ユーザー情報エンドポイントは必要ありません。

11. セキュリティを強化する OAuth 2.0 Proof Key for Code Exchange (PKCE) 拡張を自動的に有効にするには、[Use Proof Key for Code Exchange (PKCE) Extension (Proof Key for Code Exchange (PKCE) 拡張を使用)] を選択します。この設定でプロバイダーを保護する方法についての詳細は、「[実装への Proof Key for Code Exchange \(PKCE\) 拡張の組み込み](#)」を参照してください。

12. 必要に応じて、次の項目を設定します。

- [トークン発行者] 項目は、`https://URL` 形式で認証トークンの供給元を指定します。

OAuth 2.0 Web サーバー認証フローの場合、プロバイダーはトークンエンドポイントから ID トークンを応答に含める必要があります。必要に応じて、プロバイダーは更新トークンフローの応答に ID トークンを含めることができます。

この ID トークンは `Token Issuer` 値および `UserInfo` エンドポイントの情報と照らして検証されます。ID トークンの署名は検証されません。

この ID トークンの利用者は、認証プロバイダーに登録されたコンシューマー鍵です。他の利用者値を含めないでください。

- [デフォルトの範囲] に、認証エンドポイントへの要求と共に送信する範囲を入力します。送信しない場合は、認証プロバイダータイプのハードコード化されたデフォルトが使用されます。デフォルトについては、[OpenID Connect の開発者用ドキュメント](#)を参照してください。

[デフォルトの範囲] に、認証エンドポイントへの要求と共に送信する範囲を入力します。送信しない場合は、認証プロバイダータイプのハードコード化されたデフォルトが使用されます。デフォルトについては、[OpenID Connect の開発者用ドキュメント](#)を参照してください。

- コンシューマー鍵とコンシューマーの秘密を入力すると、コンシューマーの秘密がデフォルトで SOAP API の応答に含まれます。SOAP API 応答で秘密を表示しないようにするには、[SOAP API 応答にコンシューマーの秘密を含める] の選択を解除してください。2022 年 11 月より、メタデータ API 応答では、秘密が常にプレースホルダーの値に置き換えられます。デプロイ時に、プレースホルダーをプレーンテキストのコンシューマーの秘密に置き換えるか、後から UI で値を変更してください。

13. 必要に応じて、トークンをクエリ文字列ではなくヘッダーで送信する場合は、[ヘッダーでアクセストークンを送信] を選択します。

14. 必要に応じて、コンシューマー鍵と秘密をリクエストボディではなく基本ヘッダーで送信する場合は、[ヘッダーでクライアントログイン情報を送信] を選択します。

15. 必要に応じて、次の項目を設定します。

- [カスタムエラー URL] に、プロバイダーがエラーの報告に使用する URL を入力します。
- [カスタムログアウト URL] に、ユーザーが SSO フローを使用して認証された場合に、ログアウト後の特定の移動先となる URL を入力します。この項目を使用して、ユーザーをブランド設定されたログアウトページか、またはデフォルトの Salesforce ログアウトページ以外の場所のいずれかに移動します。URL は、http または https プレフィックスを付けて完全修飾する必要があります (https://acme.my.salesforce.com など)。

 **ヒント:** シングルログアウト (SLO) を設定して、Salesforce と ID プロバイダーの両方から自動的にユーザーをログアウトするようにします。Salesforce では、証明書利用者として、ユーザーが ID プロバイダーまたは Salesforce からログアウトした場合でも OpenID Connect SLO がサポートされます。

- 既存の Apex クラスを Registration Handler クラスとして選択します。登録ハンドラーの Apex クラステンプレートを作成するには [登録ハンドラーテンプレートを自動作成] を選択します。このクラスを後で編集してデフォルトのコンテンツを変更してから使用します。

 **メモ:** Registration Handler クラスは、Salesforce で SSO 初期化 URL を生成するために必要となります。

- [他のアカウントで登録を実行] で、Apex ハンドラークラスを実行するユーザーを選択します。このユーザーは「ユーザーの管理」権限を持っている必要があります。

[他のアカウントで登録を実行] では、登録ハンドラーが動作するコンテキストが提供されます。既存の登録ハンドラークラスを指定しているか、テンプレートから登録ハンドラークラスを作成しているかに関係なく、ユーザーを選択します。本番環境では、通常、[他のアカウントで登録を実行] ユーザー用にシステムユーザーを作成します。このように、ハンドラーによって実行される操作は、その登録プロセスが簡単に特定されます。たとえば取引先責任者が作成される場合、システムユーザーがそれを作成します。

- プロバイダーでポータルを使用するには、[ポータル] ドロップダウンリストからポータルを選択します。

- [アイコン URL] に、サイトのログインページのボタンとして表示するアイコンのパスを追加します。このアイコンは Experience Cloud サイトにのみ適用されます。Salesforce のログインページまたは [私のドメイン] のログイン URL には表示されません。ユーザーは、このボタンをクリックし、サイトに関連付けられた認証プロバイダーを使用してログインします。

独自の画像へのパスを指定するか、いずれかのサンプルアイコンの URL を項目にコピーしてください。

16. ID プロバイダーの MFA サービスではなく、Salesforce 多要素認証 (MFA) 機能を使用するには、[この SSO プロバイダーで Salesforce MFA を使用] を選択します。この設定は、MFA が直接適用されているユーザーの MFA のみをトリガーします。詳細は、「[SSO での Salesforce MFA の使用](#)」を参照してください。

17. 設定を保存します。

認証プロバイダーを定義した後、複数のクライアント設定 URL が生成されます。

クライアント設定 URL では、ユーザーの特定サイトへのログイン、サードパーティからのカスタマイズされた権限の取得、認証後の特定の場所への移動を行うための、追加の要求パラメーターをサポートしています。

## OpenID Connect アプリケーションの更新

認証プロバイダーを Salesforce で定義したら、OpenID プロバイダーに戻り、アプリケーションのコールバック URL を更新します。Google アプリケーションの場合、コールバック URL は承認済みのリダイレクト URI と呼ばれています。PayPal の場合は戻り URL と呼ばれています。

## SSO 接続のテスト

ブラウザの認証プロバイダー設定ページでテスト専用初期化 URL を開きます。OpenID プロバイダーにリダイレクトされ、サインインするよう求められます。そしてサインイン時に、アプリケーションを認証するよう求められます。認証後、ユーザーは Salesforce にリダイレクトされます。

## ログインページへの認証プロバイダーの追加

認証プロバイダーをログインオプションとして表示するようにログインページを設定します。組織の SSO を設定しているのか、Experience Cloud サイトの SSO を設定しているかに応じて、このステップは異なります。

- 組織の場合、「[組織のログインページへの認証プロバイダーの追加](#)」を参照してください。
- Experience Cloud サイトの場合、「[Experience Cloud サイトのログインページへの認証プロバイダーの追加](#)」を参照してください。

## 例: Mobile SDK アプリケーションからアクセスできるように Experience Cloud サイトを設定する

---

Mobile SDK アプリケーションからログインできるように Experience Cloud サイトを設定する場合、注意が必要です。このチュートリアルでは、詳細と適切な順序を直接確認できます。

モバイルアクセス用に Experience Cloud サイトのユーザーを設定する場合、設定の順序と方法が成功を左右します。たとえば、取引先責任者が関連付けられていないユーザーは、モバイルデバイスでログインできません。次に、留意すべき重要なガイドラインを示します。



- 取引先に属する取引先責任者からのみユーザーを作成します。ユーザーを作成してから、後で取引先責任者に関連付けることはできません。
- 使用する取引先の所有者にロールを割り当てていることを確認してください。割り当てていない場合、ユーザーがログインしようとするエラーが発生します。
- iOS アプリケーションでカスタムログインホストを定義する場合、必ず `http[s]://` プレフィックスを削除してください。プレフィックスは、実行時にiOS コアによって追加されます。明示的にプレフィックスを含めると、無効なアドレスになる可能性があります。

1. [プロファイルに権限を追加する](#)
2. [Experience Cloud サイトを作成する](#)
3. [Experience Cloud サイトに API ユーザープロファイルを追加する](#)
4. [新しい取引先責任者およびユーザーを作成する](#)
5. [新しい Experience Cloud サイトのログインをテストする](#)

## プロファイルに権限を追加する

「API の有効化」および「Chatter の有効化」権限のあるプロファイルを作成します。

1. [設定] から、[クイック検索] ボックスに「プロファイル」と入力し、[プロファイル] を選択します。
2. [新規プロファイル] をクリックします。
3. [既存のプロファイル] で、[Customer Community User (カスタマーコミュニティユーザー)] を選択します。
4. [プロファイル名] に「*FineApps API User*」(FineApps API ユーザー) と入力します。
5. [保存] をクリックします。
6. [FineApps API User (FineApps API ユーザー)] ページで、[編集] をクリックします。
7. [システム管理者権限] で、[API の有効化] および [Chatter を有効化] を選択します。

 **メモ:** 「Chatter の有効化」権限がないユーザーは、Salesforce の Experience Cloud サイトに正常にログインした直後に「アクセス権がありません」エラーが表示されます。

8. [保存] をクリックします。

 **メモ:** このチュートリアルではプロファイルを使用しますが、必要な権限がある権限セットを使用することもできます。

## Experience Cloud サイトを作成する

Experience Cloud サイトと、サイトのログイン URL を作成します。

次の手順についての詳細は、Salesforce ヘルプの「[デジタルエクスペリエンスの有効化](#)」および「[Experience Cloud サイトを作成する](#)」を参照してください。

1. [設定] から、[クイック検索] ボックスに「デジタルエクスペリエンス」と入力します。
2. [すべてのサイト] が表示されない場合、次の手順を実行します。
  - a. [設定] をクリックします。

- b. [デジタルエクスペリエンスを有効化] を選択します。
  - c. [ドメイン名] に一意のドメイン名 (*fineapps.<あなたの名前>.force.com* など) を入力します。
  - d. [使用可能か調べる] をクリックして、ドメイン名がまだ使用されていないことを確認します。
  - e. [保存] をクリックします。
3. [設定] から、[クイック検索] ボックスに「デジタルエクスペリエンス」と入力し、[すべてのサイト] を選択します。
  4. [新規サイト] をクリックします。
  5. テンプレートを選択し、新しい Experience Cloud サイトに「*FineApps Users*」(FineApps ユーザー) という名前を付けます。
  6. サフィックス編集ボックスで、[URL] に「*customers*」(顧客) と入力します。  
サフィックスを含んだ完全な URL が表示され、これが Experience Cloud サイトの新しい URL になります。
  7. [サイトを作成] をクリックします。

## Experience Cloud サイトに API ユーザープロフィールを追加する

API ユーザープロフィールを [メンバー] ページの Experience Cloud サイト設定に追加します。

1. [管理] > [メンバー] をクリックします。
2. [検索] で、[すべて] を選択します。
3. [選択可能なプロフィール] リストで [FineApps API ユーザー] を選択し、[追加] をクリックします。
4. [保存] をクリックします。
5. [公開] をクリックします。
6. 確認ダイアログボックスを閉じて、[閉じる] をクリックします。

## 新しい取引先責任者およびユーザーを作成する

ユーザーを直接作成する代わりに、取引先の取引先責任者を作成した後でその取引先責任者からユーザーを作成します。

現在、取引先がない場合、次の手順を実行します。

1. [取引先] タブをクリックします。
2. 組織にまだ取引先がない場合、次の手順を実行します。
  - a. [簡易作成] の [取引先名] に「*My Test Account*」(私のテスト取引先) と入力します。
  - b. [保存] をクリックします。
3. [最近の取引先] で、[My Test Account (私のテスト取引先)] またはその他の取引先名をクリックします。取引先所有者の名前をメモします。
4. [設定] から、[クイック検索] ボックスに「ユーザー」と入力し、[ユーザー] を選択して、取引先所有者名の横にある [編集] をクリックします。



5. [ロール] が CEO などの管理ロールに設定されていることを確認します。
6. [保存] をクリックします。
7. [取引先] タブをクリックして、取引先の名前を再度クリックします。
8. [取引先責任者] で、[新規取引先責任者] をクリックします。
9. [名] に「Jim」、[姓] に「Parker」と入力します。[保存] をクリックします。
10. Jim Parker の [取引先責任者] ページで、[外部ユーザーの管理] をクリックし、[カスタマーユーザーを有効化] を選択します。
11. [ユーザーライセンス] で、[Customer Community (カスタマーコミュニティ)] を選択します。
12. [プロファイル] で、[FineApps API User (FineApps API ユーザー)] を選択します。
13. その他の必須項目に次の値を使用します。

| 項目     | 値                      |
|--------|------------------------|
| メール    | 有効なメールアドレスを入力します。      |
| ユーザー名  | jimparker@fineapps.com |
| ニックネーム | jimmyp                 |

必須でない情報は、ブラウザーによって自動的に入力される場合は削除できます。

14. [保存] をクリックします。
15. Jim Parker への歓迎メールが受信箱に届くのを待ってから、そのメールのリンクをクリックしてパスワードを作成します。パスワードを「mobile333」に設定します。

## 新しい Experience Cloud サイトのログインをテストする

新しい取引先責任者として Mobile SDK ネイティブまたはハイブリッドローカルアプリケーションにログインし、Experience Cloud サイト設定をテストします。

Mobile SDK アプリケーションから Experience Cloud サイトにログインするには、サイトログイン URL を認識するようにアプリケーションを設定します。

1. Android の場合:
  - a. Android Studio で Android プロジェクトを開きます。
  - b. プロジェクトエクスプローラーで、res フォルダーに移動し、新しい xml フォルダーを作成するか、既存のフォルダーを選択します。
  - c. xml フォルダーで、テキストファイルを作成します。これは、[ファイル] メニューまたは **Ctrl**-クリック (または右クリック) メニューを使用して行うことができます。

- d. 新しいテキストファイルで、次の XML を追加します。サーバー URL を Experience Cloud サイトのログイン URL で置き換えます。

```
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <server name="Experience Cloud Site Login" url=
 "https://fineapps-dev-ed.my.site.com/fineapps">
 </server>
</servers>
```

- e. ファイルを `servers.xml` として保存します。

## 2. iOS の場合:

- a. ログインしないでアプリケーションを開始します。
- b. ログイン画面で [設定] をタップするか、上部のナビゲーションバーの「歯車」アイコン  をタップします。
- c. [接続を選択] 画面で、プラスアイコン  をタップします。
- d. (省略可能ですが推奨) 今後アクセスしたときにこの設定を簡単に識別できるように、表示ラベルを入力します。
- e. カスタムログインホストの URI を入力します。必ず `https://` プレフィックスを削除してください。次に、Experience Cloud サイトの一般的な URI の入力方法の例を示します。

```
MyDomainName.my.site.com/fineapps
```

または、設定に MDM を使用している場合、MDM を介してログイン画面を設定します。

3. デバイス、シミュレーター、またはエミュレーターでアプリケーションを起動し、ユーザー名 `jimparker@fineapps.com`、パスワード `mobiletest1234` でログインします。

 **メモ:** モバイルアプリケーションで長期間ログイン画面のままでいると、ログイン時に「アクセス権がありません」エラーが発生する可能性があります。この場合、アプリケーションを閉じてから再度開き、すぐにログインします。

## 例: Facebook 認証できるように Experience Cloud サイトを設定する

Experience Cloud サイトのログインを処理するように外部認証プロバイダーを設定して、サイトへのアクセスを拡張できます。

この例では、Facebook を認証フロントエンドとして使用するように前の例を拡張します。この簡単なシナリオでは、認証された Facebook ユーザーの Experience Cloud サイトへの参加を許可できるように外部認証プロバイダーを設定します。

モバイルアプリケーションにログインできるように Experience Cloud サイトがすでに設定されている場合、外部認証を使用するようにモバイルアプリケーションまたは接続アプリケーションを変更する必要はありません。代わりに、Facebook アプリケーション、Salesforce 認証プロバイダー、および認証プロバイダー Apex クラスを定義します。また、Experience Cloud サイトの設定も若干変更します。

## Facebook アプリケーションを作成する

Facebook を介した Experience Cloud サイトログインを有効にするには、まず Facebook アプリケーションを作成します。

Facebook アプリケーションは、Salesforce 接続アプリケーションに相当します。接続アプリケーションは、モバイルデバイスのアプリケーションの接続および認証を管理する設定用コンテナです。

1. 「Facebook アプリケーションを作成する」に移動します。
2. Facebook 開発者アカウントでログインします。Facebook 開発者として登録していない場合は、登録します。
3. [アプリケーション] > [新規アプリケーションを作成] に移動します。
4. 表示名を「FineApps Experience Cloud Site Test」（FineApps Experience Cloud サイトテスト）に設定します。
5. 必要に応じて、名前空間を追加します。Facebook の要件に従って、小文字、ダッシュ、およびアンダースコアのみで 20 文字以下の名前空間表示ラベルを入力する必要があります。たとえば、「my\_fb\_goodapps」のようになります。
6. [カテゴリ] で、[公益事業] を選択します。
7. 後で使用するためにアプリ ID とアプリのシークレットキーをコピーして保存します。

次の URL を使用してアプリケーションにログインできます。

`https://developers.facebook.com/apps/<App ID>/dashboard/`

## Salesforce 認証プロバイダーを定義する

Salesforce で外部認証を有効にするには、認証プロバイダーを作成します。

Facebook を介した外部認証には、前のステップで作成した Facebook アプリケーションのアプリケーション ID とアプリケーションのシークレットキーが必要です。

1. [設定] で、[クイック検索] ボックスに「認証プロバイダー」と入力し、[認証プロバイダー] を選択します。
2. [新規] をクリックします。
3. 次の表に示すように、認証プロバイダーの項目を設定します。

| 項目        | 値                                                                                                                                                                |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| プロバイダータイプ | [Facebook] を選択します。                                                                                                                                               |
| 名前        | 「FB Community Login」（FB コミュニティログイン）と入力します。                                                                                                                       |
| URL 接尾辞   | デフォルトを受け入れます。<br> <b>メモ:</b> URL 構文に適合するその他の文字列を入力することもできますが、この例ではデフォルトが最適です。 |
| コンシューマー鍵  | Facebook アプリケーションのアプリケーション ID を入力します。                                                                                                                            |

| 項目          | 値                                     |
|-------------|---------------------------------------|
| コンシューマーの秘密  | Facebook アプリケーションのアプリのシークレットキーを入力します。 |
| カスタムエラー URL | 空白のままにします。                            |

4. [登録ハンドラー] で、[登録ハンドラーテンプレートを自動作成] をクリックします。

- 5.

[他のアカウントで登録を実行] で、[検索]  をクリックし、管理権限のある Experience Cloud サイトのメンバーを選択します。

6. [ポータル] は空白のままにします。

7. [保存] をクリックします。

Salesforce によって、RegistrationHandler を拡張する新しい Apex クラスが作成されます。このクラス名は、AutocreatedRegHandlerxxxxx... の形式を取ります。

8. 後で使用するために認証プロバイダー ID をコピーします。

9. 新しい認証プロバイダーの詳細ページの [クライアント設定] で、後で使用するためにコールバック URL をコピーします。

このコールバック URL は、

`https://login.salesforce.com/services/authcallback/<id>/<Auth.Provider_URL_Suffix>` の形式を取ります。

## Facebook アプリケーションを設定する

次に、ログインに Salesforce 認証プロバイダーを使用するように Experience Cloud サイトを設定します。

Salesforce 認証プロバイダーを定義したので、Facebook アプリケーションを認証プロバイダーにリンクして認証プロトコルを完成させます。Salesforce ログイン URL およびコールバック URL を指定します。この URL には、認証プロバイダー ID と認証プロバイダーの URL サフィックスが含まれます。

1. Facebook アプリケーションで、[設定] に移動します。
2. [App Domains (アプリのドメイン)] に、`「MyDomainName.my.salesforce.com」` と入力します。
3. [+プラットフォームを追加] をクリックします。
4. [Web サイト] を選択します。
5. [サイト URL] に、認証プロバイダーのコールバック URL を入力します。
6. [連絡先メールアドレス] に、有効なメールアドレスを入力します。
7. 左パネルで、[Status & Review (ステータス & レビュー)] を [はい] に設定します。この設定により、すべての Facebook ユーザーは Facebook ログインを使用して、Experience Cloud サイトのユーザーアカウントを作成できます。
8. [変更を保存] をクリックします。

9. [確認] をクリックします。

## 認証プロバイダーの Apex クラスをカスタマイズする

認証プロバイダーの Apex クラスを使用して、Experience Cloud サイトに参加できるユーザーを制御する条件ロジックを定義します。

1. [設定] から、[クイック検索] ボックスに「Apex クラス」と入力し、[Apex クラス] を選択します。
2. 認証プロバイダークラスの横にある [編集] をクリックします。デフォルトのクラス名は、「AutocreatedRegHandlerxxxxx...」で始まります。
3. `canCreateUser()` メソッドを実装するには、`true` を返します。

```
global boolean canCreateUser(Auth.UserData data) {
 return true;
}
```

この実装により、Facebook を介してログインするすべてのユーザーが Experience Cloud サイトに参加できます。

 **メモ:** 既存のメンバーのみが Experience Cloud サイトにアクセスできるようにするには、検索条件を実装してすべての有効ユーザーを識別できるようにします。Facebook パケットの一意のデータ (ユーザー名やメールアドレスなど) の条件に基づいて、Experience Cloud サイトメンバーのレコードの同様の項目に対してそのデータを検証します。

4. `createUser()` コードを変更します。
  - a. アカウント名クエリの「Acme」を「FineApps」に置き換えます。
  - b. ユーザー名のサフィックス (「@acmecorp.com」) を「@fineapps.com」に置き換えます。
  - c. プロファイルクエリのプロファイル名 (「カスタマーポータルユーザー」) を「API の有効化」に変更します。
5. `updateUser()` コードで、ユーザー名 (「myorg.com」) のサフィックスを「@fineapps.com」に置き換えます。
6. [保存] をクリックします。

## Experience Cloud サイトの設定

最後のステップでは、ログインに Salesforce 認証プロバイダーを使用するように Experience Cloud サイトを設定します。

1. [設定] から、[クイック検索] ボックスに「デジタルエクスペリエンス」と入力し、[すべてのサイト] を選択します。
2. サイト名の横にある [管理] をクリックします。
3. [管理] > [ログイン & 登録] をクリックします。
4. [ログイン] で、新しい認証プロバイダーを選択します。
5. [保存] をクリックします。

これで完了です。Experience Cloud サイトのログイン URL を使用してモバイルアプリケーションにログインするときに、Facebook を使用したログインを促す追加のボタンが表示されます。そのボタンをクリックして画面の指示に従い、ログインがどのように機能するのかを確認します。

ブラウザーで外部認証設定をテストするには、(認証プロバイダーから) シングルサインオン初期化 URL をカスタマイズします。

```
https://MyDomainName.my.salesforce.com/services/auth/sso/orgID/
URLsuffix?community=ExperienceCloudSite_login_url
```

次に例を示します。

```
https://MyDomainName.my.salesforce.com/services/auth/sso/00Da0000000TPNEAA4/
FB_Community_Login?community=
https://MyDomainName.my.site.com/fineapps
```

既存ユーザーをリンクする URL を作成するには、sso を link に置き換えます。

```
https://MyDomainName.my.salesforce.com/services/auth/link/00Da0000000ABCDEF9/
FB_Community_Login?community=
https://MyDomainName.my.site.com/fineapps
```

トピック:

- マルチユーザーのサポートについて
- マルチユーザーのサポートの実装

複数のユーザーが同時にログインできるようにするために、Mobile SDK には、ユーザーの切り替えの基本的な実装と API が用意されています。

Mobile SDK には、認証されたアカウントから選択できるデフォルトのダイアログボックスが用意されています。アプリケーションでこのダイアログボックスを起動する手段を実装し、ユーザーの切り替えフローを開始する API をコールします。



## マルチユーザーのサポートについて

バージョン 2.2 以降、Mobile SDK では、複数のユーザーアカウントから同時にログインできます。これらのアカウントは、同じ組織の異なるユーザーか、異なる組織 (本番組織と Sandbox 組織など) の異なるユーザーを表します。

ユーザーがログインすると、サーバーに対して再認証することなくアカウント間のシームレスな切り替えを行うことができるようにそのユーザーのログイン情報が保存されます。複数のログインをサポートしない場合は、アプリケーションを変更する必要はありません。既存の Mobile SDK API は、シングルユーザーの場合と同じように機能します。

Mobile SDK では、各ユーザーアカウントが、認証された他のユーザーアカウントに関連していないと想定しています。それに応じて、Mobile SDK では、各アカウントに関連付けられたデータと他のすべてのユーザーのデータが分離されます。これにより、アカウント間でデータが混じることを回避できます。データの分離により、アカウントに関連付けられた SharedPreferences ファイル、SmartStore データベース、AccountManager データ、およびその他のフラットファイルが保護されます。

 **例:** ネイティブ Android の場合、RestExplorer サンプルアプリケーションでマルチユーザーの切り替えが示されます。

ネイティブ iOS の場合、RestAPIExplorer サンプルアプリケーションでマルチユーザーの切り替えが示されます。

ハイブリッドの場合、次のハイブリッドサンプルアプリケーションでマルチユーザーの切り替えが示されます。

- SmartStore なし: ContactExplorer
- SmartStore あり: AccountEditor

## マルチユーザーのサポートの実装

Mobile SDK では、ネイティブ Android、ネイティブ iOS、およびハイブリッドアプリケーションでマルチユーザーのサポートを有効にする API が用意されています。

Mobile SDK では基礎となる機能は実装されますが、アプリケーションで次のいずれかの API をコールしない限り、実行時にマルチユーザーの切り替えが初期化されません。

**Android ネイティブ (UserAccountManager クラスメソッド)**

```
public void switchToUser(UserAccount user)
public void switchToNewUser()
```

**iOS ネイティブ (SFUserAccountManager クラスメソッド)**

```
- (void)switchToUser:(SFUserAccount *)newCurrentUser
- (void)switchToNewUser
```

**ハイブリッド (JavaScript メソッド)**

```
switchToUser
```

ユーザーが別のアカウントに切り替えるようにするには、ユーザーインターフェースのボタン、メニュー、またはその他のコントロールから選択画面を起動します。Mobile SDK では、現在認証されているすべてのアカウ

ントをラジオボタンリストに表示する標準のマルチユーザーの切り替え画面が用意されています。この画面をカスタマイズすることも、デフォルトバージョンをそのまま表示することもできます。ユーザーが選択すると、マルチユーザーフローを起動する Mobile SDK メソッドをコールします。

API の使用を開始する前に、Mobile SDK とアプリケーションの分業を理解しておくことが重要です。次のリストに、Mobile SDK が実行するタスクと、複数のユーザーがいる状況でアプリケーションが実行する必要のあるタスクを示します。特に、以下の管理方法を考慮してください。

- [プッシュ通知](#) (アプリケーションでサポートしている場合)
- [SmartStore スープ](#) (アプリケーションで SmartStore を使用している場合)
- [アカウント管理](#)

## プッシュ通知のタスク

Mobile SDK (すべてのアカウント):

- ログイン時にプッシュ通知を登録する
- ログアウト時にプッシュ通知を登録解除する
- プッシュ通知を配信する

アプリケーション

- 対象ユーザーアカウントに応じて通知を識別する
- 適切なユーザーコンテキストを起動して各通知を表示する

## SmartStore のタスク

Mobile SDK (すべてのアカウント):

- 認証されたユーザーアカウントごとに個別の SmartStore データベースを作成する
- ユーザーの切り替えが発生するたびに適切なバックエンドデータベースに切り替える

アプリケーション

- ユーザーの切り替えまたはログアウトが発生するたびに、キャッシュされたログイン情報(メモリに保持された SmartStore のインスタンスなど)を更新する

## アカウント管理のタスク

Mobile SDK (すべてのアカウント):

- ユーザーの切り替えが発生するたびに適切なアカウントログイン情報を読み込む

アプリケーション

- ユーザーの切り替えまたはログアウトが発生するたびに、キャッシュされたログイン情報(メモリに保持された認証済み REST クライアントなど)を更新する

## Android ネイティブ API

Mobile SDK for Android のネイティブクラスは、マルチユーザーのサポートの大半の作業を行います。アプリケーションは、いくつかの簡単なコールを行い、メモリにキャッシュされたすべてのデータを処理します。ユーザーの切り替えアクティビティをカスタマイズすることもできます。

Mobile SDK for Android では、ユーザーの切り替えをサポートするために、`com.salesforce.androidsdk.accounts`、`com.salesforce.androidsdk.ui`、および `com.salesforce.androidsdk.util` パッケージのネイティブクラスを定義します。`com.salesforce.androidsdk.accounts` パッケージには、次のクラスがあります。

- `UserAccount`
- `UserAccountManager`

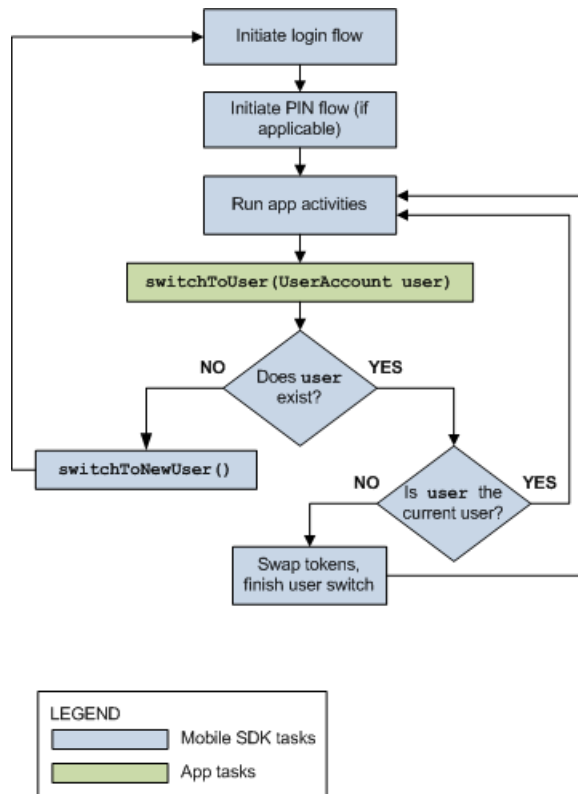
`com.salesforce.androidsdk.ui` パッケージには、`AccountSwitcherActivity` クラスがあります。このクラスを拡張して、アカウントの切り替えアクティビティを高度にカスタマイズできます。

`com.salesforce.androidsdk.util` パッケージには、`UserSwitchReceiver` 抽象クラスがあります。アプリケーションでトークン以外のデータをキャッシュする場合、このクラスを実装する必要があります。

次セクションでは、これらのクラスについて簡単に説明します。完全な API リファレンスドキュメントについては、<https://forcedotcom.github.io/SalesforceMobileSDK-Android/index.html> を参照してください。

## マルチユーザーフロー

ネイティブ Android アプリケーションの場合、`UserAccountManager.switchToUser()` Mobile SDK メソッドがマルチユーザーフローを起動します。アプリケーションでこのメソッドがコールされると、Mobile SDK コアがすべてのパスで実行フローを処理します。次のダイアグラムに、このフローを示します。



このセクションの内容:

#### UserAccount クラス

UserAccount クラスは、現在認証されている単一のユーザーアカウントを表します。このクラスは、ユーザーアカウントを一意に識別するために使用できるデータをカプセル化します。

#### UserAccountManager クラス

UserAccountManager クラスは、認証されたアカウントへのアクセス、新しいアカウントの追加、既存のアカウントのログアウト、既存のアカウント間の切り替えを行うメソッドを提供します。

#### AccountSwitcherActivity クラス

AccountSwitcherActivity クラスを使用または拡張して、ユーザーの切り替えのインターフェースを表示します。

#### UserSwitchReceiver クラス

ネイティブ Android アプリケーションがトークン以外のデータをキャッシュする場合、UserSwitchReceiver 抽象クラスを実装して、ユーザーの切り替えイベントの通知を受信します。

## UserAccount クラス

UserAccount クラスは、現在認証されている単一のユーザーアカウントを表します。このクラスは、ユーザーアカウントを一意に識別するために使用できるデータをカプセル化します。

## コンストラクター

JSON オブジェクトまたはバンドルから直接 `UserAccount` オブジェクトを作成できます。

| コンストラクター                                                                                                                                                                                                                                                                                                    | 説明                                                            |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|
| <pre>public UserAccount(     String authToken,     String refreshToken,     String loginServer,     String idUrl,     String instanceServer,     String orgId,     String userId,     String username,     String accountName,     String clientId,     String communityId,     String communityUrl )</pre> | 指定した値を使用して <code>UserAccount</code> オブジェクトを作成します。             |
| <pre>public UserAccount(JSONObject object)</pre>                                                                                                                                                                                                                                                            | JSON 文字列から <code>UserAccount</code> オブジェクトを作成します。             |
| <pre>public UserAccount(Bundle bundle)</pre>                                                                                                                                                                                                                                                                | Android アプリケーションバンドルから <code>UserAccount</code> オブジェクトを作成します。 |

## メソッド

| メソッド                                               | 説明                                                                                                                                                                                                                                                      |
|----------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>public String getOrgLevelStoragePath()</pre>  | このユーザーアカウントの組織レベルのストレージパス (アプリケーションデータの上位レベルのディレクトリに対する相対パス) を返します。上位レベルのディレクトリは <code>files</code> などになります。出力の形式は、 <code>/ {orgID} /</code> になります。これは、同じ組織の複数のユーザーで共有できるデータのストレージパスです。                                                                 |
| <pre>public String getUserLevelStoragePath()</pre> | このユーザーアカウントのユーザーレベルのストレージパス (アプリケーションデータの上位レベルのディレクトリに対する相対パス) を返します。上位レベルのディレクトリは <code>files</code> などになります。出力の形式は、 <code>/ {orgID} / {userID} /</code> になります。これは、組織の特定のユーザーに固有のデータのストレージパスですが、そのユーザーがメンバーであるその組織内のすべてのコミュニティで共有されるデータのストレージパスでもあります。 |

| メソッド                                                                         | 説明                                                                                                                                                                                                                                                                                                                               |
|------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>public String getCommunityLevelStoragePath(String communityId)</pre>    | <p>このユーザーアカウントのコミュニティレベルのストレージパス (アプリケーションデータの上位レベルのディレクトリに対する相対パス) を返します。上位レベルのディレクトリは <code>files</code> などになります。出力の形式は、<code>/{orgID}/{userID}/{communityID}/</code> になります。<code>communityID</code> が <code>null</code> の場合、出力は <code>/{orgID}/{userID}/internal/</code> のようになります。これは、特定のコミュニティの特定のユーザーに固有のデータのストレージパスです。</p> |
| <pre>public String getOrgLevelFilenameSuffix()</pre>                         | <p>このユーザーアカウントの一意のサフィックスを返します。このサフィックスをファイルに追加して、組織レベルでこのアカウントを一意に識別できます。出力の形式は、<code>_{orgID}</code> になります。これは、同じ組織の複数のユーザーで共有できるデータのサフィックスです。</p>                                                                                                                                                                              |
| <pre>public String getUserLevelFilenameSuffix()</pre>                        | <p>このユーザーアカウントの一意のサフィックスを返します。このサフィックスをファイルに追加して、ユーザーレベルでこのアカウントを一意に識別できます。出力の形式は、<code>_{orgID}_{userID}</code> になります。これは、組織の特定のユーザーに固有のデータのサフィックスですが、そのユーザーがメンバーであるその組織内のすべてのコミュニティで共有されるデータのサフィックスでもあります。</p>                                                                                                                 |
| <pre>public String getCommunityLevelFilenameSuffix(String communityId)</pre> | <p>このユーザーアカウントの一意のサフィックスを返します。このサフィックスをファイルに追加して、コミュニティレベルでこのアカウントを一意に識別できます。出力の形式は、<code>_{orgID}_{userID}_{communityID}</code> になります。<code>communityID</code> が <code>null</code> の場合、出力は <code>_{orgID}_{userID}_internal</code> のようになります。これは、特定のコミュニティの特定のユーザーに固有のデータのサフィックスです。</p>                                          |

## UserAccountManager クラス

`UserAccountManager` クラスは、認証されたアカウントへのアクセス、新しいアカウントの追加、既存のアカウントのログアウト、既存のアカウント間の切り替えを行うメソッドを提供します。

`UserAccountManager` のインスタンスは直接作成しません。代わりに、次のコールを使用してインスタンスを取得します。

```
SalesforceSDKManager.getInstance().getUserAccountManager();
```

## メソッド

| メソッド                                                                                                  | 説明                                                                                                    |
|-------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| <code>public <a href="#">UserAccount</a> getCurrentUser()</code>                                      | 現在有効なユーザーアカウントを返します。                                                                                  |
| <code>public List&lt;<a href="#">UserAccount</a>&gt; getAuthenticatedUsers()</code>                   | 認証されたユーザーアカウントのリストを返します。                                                                              |
| <code>public boolean doesUserAccountExist(<a href="#">UserAccount</a> account)</code>                 | 指定されたユーザーアカウントがすでに認証されているかどうかを確認します。                                                                  |
| <code>public void switchToUser(<a href="#">UserAccount</a> user)</code>                               | 指定されたユーザーアカウントにアプリケーションコンテキストを切り替えます。指定されたユーザーアカウントが無効または <code>null</code> の場合、このメソッドはログインフローを起動します。 |
| <code>public void switchToNewUser()</code>                                                            | 新しいユーザーがログインするためのログインフローを起動します。                                                                       |
| <code>public void signoutUser(<a href="#">UserAccount</a> userAccount, Activity frontActivity)</code> | 指定されたユーザーをアプリケーションからログアウトさせて、そのユーザーのログイン情報を消去します。                                                     |

## AccountSwitcherActivity クラス

`AccountSwitcherActivity` クラスを使用または拡張して、ユーザーの切り替えのインターフェースを表示します。

`AccountSwitcherActivity` クラスは、マルチユーザーログインを処理する画面を提供します。この画面では、既存のユーザーアカウントのリストが表示され、ユーザーは既存のアカウントを切り替えたり、新しいアカウントにサインインしたりできます。マルチユーザーログインを有効にするには、次のコードを使用してアプリケーションの任意の場所からアクティビティを起動します。

```
final Intent i = new Intent(this, SalesforceSDKManager.getInstance().
 getAccountSwitcherActivityClass());
i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
this.startActivity(i);
```

たとえば、ユーザーインターフェースの「ユーザーの切り替え」ボタンからこのアクティビティを起動できます。例については、`SampleApps/RestExplorer` を参照してください。

必要に応じて、XML を介して `AccountSwitcherActivity` をカスタマイズおよびスタイル設定できます。

さらに詳細に制御するために、`AccountSwitcherActivity` を拡張したり、独自のカスタムサブクラスに置き換えたりできます。デフォルトクラスを置き換えるには、

`SalesforceSDKManager.setAccountSwitcherActivityClass()` をコールします。置き換わるアクティビティクラス (`AccountSwitcherActivity.class` など) のクラスファイルに参照を渡します。



## UserSwitchReceiver クラス

ネイティブ Android アプリケーションがトークン以外のデータをキャッシュする場合、UserSwitchReceiver 抽象クラスを実装して、ユーザーの切り替えイベントの通知を受信します。

ユーザーの切り替えが発生するたびに、Mobile SDK は Intent をブロードキャストします。Intent アクションは、次のように UserAccountManager クラスで宣言されます。

```
public static final String USER_SWITCH_INTENT_ACTION =
 "com.salesforce.USERSWITCHED";
```

このブロードキャストイベントにより、キャッシュされたリソースをアプリケーションで適切に更新して、ユーザーの切り替えに対応できます。アプリケーションがこのイベントをリスンできるように、Mobile SDK には、UserSwitchReceiver 抽象クラスが用意されています。このクラスは、次の Salesforce アクティビティクラスで実装されます。

- SalesforceActivity
- SalesforceListActivity
- SalesforceExpandableListActivity

メインアクティビティでいずれかの Salesforce アクティビティクラスを拡張する場合、UserSwitchReceiver を実装する必要はありません。

メモリにトークンのみをキャッシュしている場合、何もする必要はありません。Mobile SDK が自動的にトークンを更新します。

トークン以外のユーザーデータをキャッシュしている場合、カスタム更新アクションでアクティビティの refreshIfUserSwitched() メソッドを上書きします。

メインアクティビティでいずれかの Salesforce アクティビティクラスを拡張しない場合、UserSwitchReceiver を実装して、ユーザーの切り替え時にキャッシュされたデータを処理します。

ブロードキャストレシーバーを設定する手順は、次のとおりです。

1. UserSwitchReceiver のサブクラスを実装します。
2. アクティビティの onCreate() メソッドでサブクラスをレシーバーとして登録します。
3. アクティビティの onDestroy() メソッドでレシーバーを登録解除します。

例については、RestExplorer サンプルアプリケーションの ExplorerActivity クラスを参照してください。

アプリケーションがハイブリッドアプリケーションの場合、作業は必要ありません。

SalesforceDroidGapActivity クラスは、ユーザーの切り替えの発生時に必要に応じてキャッシュを更新します。

## メソッド

1つのメソッドで実装が必要になります。

| メソッド名                                   | 説明                                                       |
|-----------------------------------------|----------------------------------------------------------|
| protected abstract void onUserSwitch(); | このメソッドを実装して、ユーザーの切り替えの発生時にキャッシュされたユーザーデータ(トークン以外)を処理します。 |

## iOS ネイティブ API

MobileSDKforiOSのネイティブクラスは、マルチユーザーのサポートの大半の作業を行います。アプリケーションは、いくつかの簡単なコールを行い、メモリにキャッシュされたすべてのデータを処理します。ユーザーの切り替えアクティビティをカスタマイズすることもできます。

Mobile SDK for iOS では、ユーザーの切り替えをサポートするために、SalesforceSDKCore ライブラリの Security フォルダのネイティブクラスを定義します。クラスには、以下が含まれます。

- SFUserAccount
- SFUserAccountManager

次セクションでは、これらのクラスについて簡単に説明します。完全な API リファレンスドキュメントについては、『[SalesforceSDKCore Reference](#)』を参照してください。

このセクションの内容:

### SFUserAccount クラス

SFUserAccount クラスは、現在認証されている単一のユーザーアカウントを表します。このクラスは、ユーザーアカウントを一意に識別するために使用できるデータをカプセル化します。

### SFUserAccountManager クラス

SFUserAccountManager クラスは、認証されたアカウントへのアクセス、新しいアカウントの追加、アカウントのログアウト、アカウント間の切り替えを行うメソッドを提供します。

## SFUserAccount クラス

SFUserAccount クラスは、現在認証されている単一のユーザーアカウントを表します。このクラスは、ユーザーアカウントを一意に識別するために使用できるデータをカプセル化します。

## プロパティ

JSON オブジェクトまたはバンドルから直接 SFUserAccount オブジェクトを作成できます。

| プロパティ                                                                     | 説明                                                                     |
|---------------------------------------------------------------------------|------------------------------------------------------------------------|
| <pre>@property (nonatomic, copy) NSSet *accessScopes</pre>                | このユーザーのアクセス範囲。                                                         |
| <pre>@property (nonatomic, strong) SFOAuthCredentials *credentials;</pre> | このユーザーに関連付けられたログイン情報。                                                  |
| <pre>@property (nonatomic, strong) SFIdentityData *idData;</pre>          | このユーザーに関連付けられた ID データ。                                                 |
| <pre>@property (nonatomic, copy, readonly) NSURL *apiUrl;</pre>           | サーバー側の API を呼び出すために使用できる URL。<br>この URL では、現在のコミュニティが考慮されます (使用可能な場合)。 |

| プロパティ                                                                                  | 説明                                                                                                                      |
|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <pre>@property (nonatomic, copy) NSString *email;</pre>                                | ユーザーのメールアドレス。                                                                                                           |
| <pre>@property (nonatomic, copy) NSString *organizationName;</pre>                     | ユーザーの組織の名前。                                                                                                             |
| <pre>@property (nonatomic, copy) NSString *fullName;</pre>                             | ユーザーの名と姓。                                                                                                               |
| <pre>@property (nonatomic, copy) NSString *userName;</pre>                             | ユーザーのユーザー名。                                                                                                             |
| <pre>@property (nonatomic, strong) UIImage *photo;</pre>                               | ユーザーの写真(通常、ユーザーのサムネイル)。このクラスのコンシューマーは、写真を使用するために少なくとも1回はこのプロパティを設定する必要があります。このクラスは、サーバーから写真を取得しません。写真はローカルで保存および取得されます。 |
| <pre>@property (nonatomic) SFUserAccountAccessRestriction accessRestrictions;</pre>    | このユーザーに関連付けられたアクセス制限。                                                                                                   |
| <pre>@property (nonatomic, copy) NSString *communityId;</pre>                          | 現在のコミュニティ ID (ユーザーがコミュニティにログインしている場合)。コミュニティにログインしていない場合、このプロパティは nil になります。                                            |
| <pre>@property (nonatomic, readonly, getter = isSessionValid) BOOL sessionValid;</pre> | ユーザーにアクセストークンがあり、有効なセッションだと想定される場合、YES を返します。                                                                           |
| <pre>@property (nonatomic, copy) NSDictionary *customData;</pre>                       | ユーザーのカスタムデータ。このデータは逐次化できるため、customData に含まれるオブジェクトは NSCoder プロトコルに準拠する必要があります。                                          |

## グローバル関数

| 関数名                                                                                        | 説明                                                                                                 |
|--------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| <pre>NSString *SFKeyForUserAndScope (SFUserAccount *user, SFUserAccountScope scope);</pre> | 特定の範囲でこのユーザーアカウントを一意に識別するキーを返します。scope を SFUserAccountScopeGlobal に設定すると、ユーザーアカウントに関係なく同じキーが返されます。 |

## SFUserAccountManager クラス

SFUserAccountManager クラスは、認証されたアカウントへのアクセス、新しいアカウントの追加、アカウントのログアウト、アカウント間の切り替えを行うメソッドを提供します。

シングルトン SFUserAccountManager インスタンスにアクセスするには、次のメッセージを送信します。

```
[SFUserAccountManager sharedInstance]
```

## プロパティ

| プロパティ                                                        | 説明                                                                                                                                                   |
|--------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| @property (nonatomic, strong) SFUserAccount *currentUser     | 現在のユーザーアカウント。ユーザーがログインしていない場合、このプロパティは nil になる可能性があります。                                                                                              |
| @property (nonatomic, readonly) NSString *currentUserId      | 現在のユーザーの ID を取得する便利なプロパティ。このプロパティは、currentUser.credentials.userId の別名です。                                                                             |
| @property (nonatomic, readonly) NSString *currentCommunityId | 現在のユーザーのコミュニティ ID を取得する便利なプロパティ。このプロパティは、currentUser.communityId の別名です。                                                                              |
| @property (nonatomic, readonly) NSArray *allUserAccounts     | アプリケーションのすべての SFUserAccount インスタンスの NSArray。                                                                                                         |
| @property (nonatomic, readonly) NSArray *allUserIds          | すべてのユーザー ID を含む配列を返します。                                                                                                                              |
| @property (nonatomic, copy) NSString *activeUserId           | 最後の有効なユーザー ID。activeUserId によって指定されたユーザーがアカウントリストから削除されていると、このユーザーは現在のユーザーとは一時的に異なる場合があります。                                                          |
| @property (nonatomic, strong) NSString *loginHost            | ログインに使用されるホスト。                                                                                                                                       |
| @property (nonatomic, assign) BOOL retryLoginAfterFailure    | ログインプロセスが失敗した場合にログインプロセスを再起動するかどうかを制御するフラグ。デフォルト値は YES です。                                                                                           |
| @property (nonatomic, copy) NSString *oauthCompletionUrl     | OAuth ログインプロセスに使用される OAuth コールバック URL。このプロパティは、アプリケーションによってカスタマイズされている場合があります。デフォルトでは、プロパティの値は、メインバンドルの SFDCOAuthRedirectUri プロパティからコピーされます。デフォルト値は、 |

| プロパティ                                     | 説明                                            |
|-------------------------------------------|-----------------------------------------------|
|                                           | @"testsfdc:///mobilesdk/detect/oauth/done"です。 |
| @property (nonatomic, copy) NSSet *scopes | アプリケーションに関連付けられた OAuth 範囲。                    |

## メソッド

| メソッド                                                                  | 説明                                                                                                                                                   |
|-----------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| - (NSString*)<br>userAccountPlistFileForUser:(SFUserAccount*)user     | 指定されたユーザーアカウントの .plist ファイルのパスを返します。                                                                                                                 |
| - (void)<br>addDelegate:(id<SFUserAccountManagerDelegate>)delegate    | このユーザーアカウントマネージャーに代理を追加します。                                                                                                                          |
| - (void)<br>removeDelegate:(id<SFUserAccountManagerDelegate>)delegate | このユーザーアカウントマネージャーから代理を削除します。                                                                                                                         |
| - (SFLoginHostUpdateResult*)updateLoginHost                           | アプリケーション設定でアプリケーションレベルのログインホストをこの値に設定します。                                                                                                            |
| - (BOOL)loadAccounts:(NSError**)error                                 | すべてのアカウントを読み込みます。                                                                                                                                    |
| - (SFUserAccount*)createUserAccount                                   | すべてのアカウント情報を自分自身で設定する場合に空のユーザーアカウントを作成するために使用できます。それ以外の場合は、必要に応じて<br>[SFAuthenticationManager loginWithCompletion:failure:]<br>を使用し、アカウントを自動的に作成します。 |
| - (SFUserAccount*)<br>userAccountForUserId:(NSString*)userId          | 特定のユーザーIDに関連付けられたユーザーアカウントを返します。                                                                                                                     |
| - (NSArray*)<br>accountsForOrgId:(NSString*)orgId                     | 特定の組織にアクセスできるすべてのアカウントを返します。                                                                                                                         |

| メソッド                                                                                   | 説明                                                                                                                                                                                                                     |
|----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>- (NSArray *) accountsForInstanceURL: (NSString *)instanceURL</pre>               | 特定のインスタンス URL に一致するすべてのアカウントを返します。                                                                                                                                                                                     |
| <pre>- (void)addAccount: (SFUserAccount *)acct</pre>                                   | ユーザーアカウントを追加します。                                                                                                                                                                                                       |
| <pre>- (BOOL) deleteAccountForUserId: (NSString*)userId error: (NSError **)error</pre> | 特定のユーザー ID に関連付けられたユーザーアカウントを削除します。                                                                                                                                                                                    |
| <pre>- (void)clearAllAccountState</pre>                                                | メモリ内のアカウントの状態をクリアします (ただし、ディスクの内容は何も変更されません)。                                                                                                                                                                          |
| <pre>- (void) applyCredentials: (SFOAuthCredentials*)credentials</pre>                 | 指定されたログイン情報を現在のユーザーに適用します。ユーザーが存在しない場合、ユーザーが作成されます。                                                                                                                                                                    |
| <pre>- (void)applyCustomDataToCurrentUser: (NSDictionary*)customData</pre>             | <p>そのユーザーの Sandbox の外部からアクセスできる SFUserAccount にカスタムデータを適用します。このデータは、アプリケーションの起動ごとに変わることはありません。このデータは逐次化されるため、customData に含まれるオブジェクトが NSCoder プロトコルに準拠していることを確認してください。</p> <p><b>⚠ 重要:</b> このメソッドは、機密情報以外にのみに使用します。</p> |
| <pre>- (void)switchToNewUser</pre>                                                     | 現在のユーザーから新しいユーザーコンテキストに切り替えます。                                                                                                                                                                                         |
| <pre>- (void)switchToUser: (SFUserAccount *)newCurrentUser</pre>                       | 現在のユーザーを指定されたユーザーアカウントに切り替えます。                                                                                                                                                                                         |

| メソッド                                                          | 説明                                                          |
|---------------------------------------------------------------|-------------------------------------------------------------|
| <pre>- (void) userChanged: (SFUserAccountChange) change</pre> | 現在のユーザーに何らかの変更があったことを<br>SFUserAccountManager オブジェクトに通知します。 |

## ハイブリッド API

ハイブリッドアプリケーションは、Mobile SDK JavaScript API を介して、マルチユーザーのサポートを有効にできます。これらの API は、SFAccountManagerPlugin Cordova ベースのモジュールにあります。

### SFAccountManagerPlugin メソッド

これらのメソッドをコールする前に、sfaccountmanager プラグインを読み込む必要があります。次に例を示します。

```
cordova.require("com.salesforce.plugin.sfaccountmanager").logout();
```

| メソッド名          | 説明                                                                   |
|----------------|----------------------------------------------------------------------|
| getUsers       | すでにログインしているユーザーのリストを返します。                                            |
| getCurrentUser | 現在の有効なユーザーを返します。                                                     |
| logout         | ユーザーが渡された場合は指定されたユーザーをログアウトさせます。引数を指定しないでコールした場合は、現在のユーザーをログアウトさせます。 |
| switchToUser   | 指定したユーザーにアプリケーションコンテキストを切り替えます。ユーザーが指定されていない場合は、アカウントの切り替え画面を起動します。  |

ハイブリッドアプリケーションでは、マルチユーザーの切り替えのブロードキャストイベントのレシーバーを実装する必要はありません。このハンドラーは、SalesforceDroidGapActivity クラスで実装されます。



## 第 21 章

## 開発者向け Mobile SDK ツール

トピック:

- [アプリケーション  
内開発者サポート](#)

Mobile SDK では、開発者が実行時のアプリケーションの状況を確認するのに役立つツールが提供されています。

たとえば、SmartStore スープの検査とクエリ、認証済みユーザーの参照、ユーザーエージェント文字列の表示などができます。これらのツールは、デバッグを主目的として設計されています。ほとんどの場合、本番のビルドではこの情報を無効にします。

## アプリケーション内開発者サポート

Mobile SDK 6.0 では、ネイティブアプリケーションにアプリケーション開発時のデバッグ用の新規画面が複数導入されました。これらの機能は、通常はコーディングを必要とせず、デバッグビルド用に設計されています。

[DevSupport (開発サポート)] ダイアログボックスから、すべての使用可能なサポート画面を起動できます。このダイアログボックスには、実行中のアプリケーションの種別に関連するオプションのみが表示されます。デバッグ中、[Dev Support (開発サポート)] にはキーボードショートカットまたはジェスチャーでアクセスできます。

デフォルトでは、これらのツールはデバッグビルドでのみ有効になっています。ただし、必要に応じて、SDK コールを使用して本番ビルドでツールを有効または無効にすることもできます。

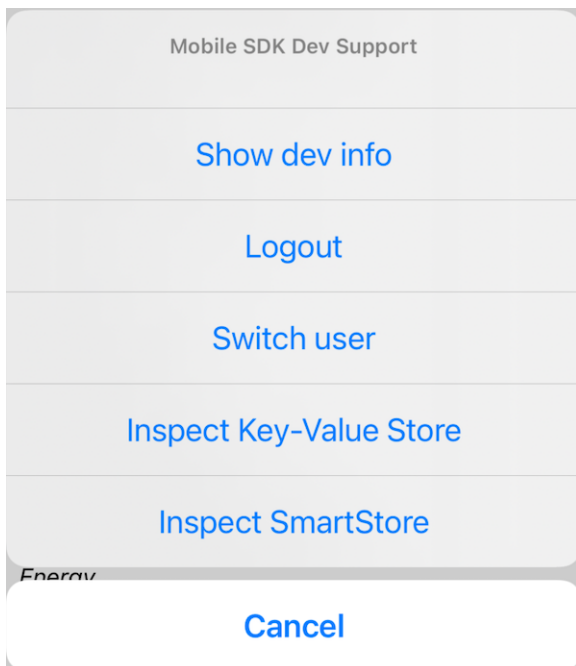
### [Dev Support (開発サポート)] ダイアログボックスの起動

#### iOS

[Dev Support (開発サポート)] ダイアログボックスを起動するには、次のいずれかのオプションを使用します。

- 物理的なデバイスをシェイクする。
- [iOS Simulator (iOS シミュレーター)] メニューで、[Hardware (ハードウェア)] > [Shake Gesture (デバイスのシェイク)] を選択する。
- iOS Simulator でアプリケーションを実行中に、`⌘+Command+z` キーボードショートカットを使用する。

次に、iOS 画面を示します。

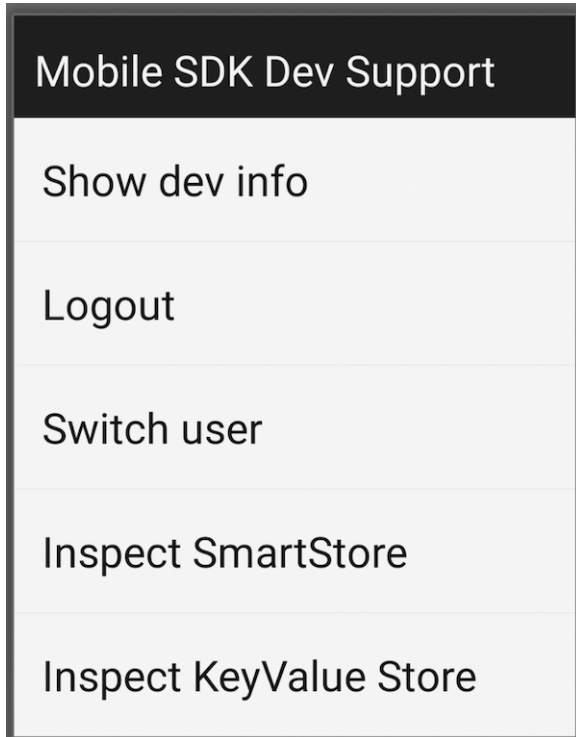


#### Android

[Dev Support (開発サポート)] ダイアログボックスを起動するには、次のいずれかのオプションを使用します。

- Android Simulator でアプリケーションを実行中に、*Command+m* (Mac) または *Ctrl+m* (Windows) キーボードショートカットを使用する。
- システムコマンドシェルで、*adb shell input keyevent 82* を実行する。

次に、Android 画面を示します。



## 本番ビルドでの開発サポートツールの有効化

本番ビルドで開発サポートツールが必要になった場合、`[DevSupport(開発サポート)]`が有効になっていることを示すプロパティを設定します。この用途は一般的ではありません。

### iOS

```
// To enable
[SalesforceSDKManager sharedManager].isDevSupportEnabled = YES;
// To disable
[SalesforceSDKManager sharedManager].isDevSupportEnabled = NO;
```

### Android

```
// To enable
SalesforceSDKManager.getInstance().setDevSupportEnabled(true);
// To disable
SalesforceSDKManager.getInstance().setDevSupportEnabled(false);
```

**⚠ 重要:** アプリケーションを公開して配布する前に、必ずこの機能を無効にしてください。

## サポートオプション

[DevSupport(開発サポート)] ダイアログボックスには、アプリケーション種別に基づいてオプションが表示されます。次のオプションはすべての場合に表示されます。

- **Show dev info** (開発情報を表示) — 開発情報画面を表示します。
- **Logout** (ログアウト) — 現在のユーザーをログアウトします。使用を開始して、まだログアウトボタンまたはアクションを UI に追加する時間がない場合に便利です。
- **Switch user** (ユーザーの切り替え) — 別のユーザーに切り替えます。使用を開始して、まだユーザーの切り替えボタンまたはアクションを UI に追加する時間がない場合に便利です。
- **Inspect Key-Value Store** (キー - 値ストアを検査) — 暗号化されたキー - 値ストアに保存されている値を検査できます。

アプリケーションで SmartStore を使用している場合、次のオプションも表示されます。

**Inspect SmartStore** (SmartStore の検査) — SmartStore インスペクター画面を表示します。

SmartStore インスペクターでは表示目的でデータが復号化されるため、開発中に便利です。この復号化はインスペクターの表示にのみ適用され、ディスク上のデータは暗号化された状態のままです。

アプリケーションが React Native アプリケーションの場合、次のオプションも表示されます。

**React native dev support** (React Native 開発サポート) — React Native 開発メニュー。

## [Dev Info (開発情報)] 画面

[Dev Info (開発情報)] 画面には、アプリケーション種別に基づいて、アプリケーション、その設定などに関する情報のコレクションが表示されます。次のオプションはすべての場合に表示されます。

- SDK バージョン
- アプリケーション種別
- ユーザーエージェント
- ネイティブブラウザでのログインが有効 (高度な認証が設定されているかどうかを示します)
- IDプロバイダーのログインが有効 (アプリケーションが別のアプリケーションを使用してログインを提供できるかどうかを示します)
- 現在のユーザー
- 認証済みユーザー
- ブート構成設定
- 管理 (管理アプリケーションかどうかを示します)
- 管理設定 (該当する場合は、MDM プロバイダーによってアプリケーションに転送されるアプリケーション設定)

SmartStore アプリケーションの場合、次の追加の情報が表示されます。

- SQLCipher バージョン
- SQLCipher コンパイルオプション
- ユーザーストアの名前
- グローバルストアの名前

## SmartStore インスペクター

SmartStore インスペクター画面は従来の機能で、ユーザーは SmartStore スープとそのインデックスのリストを表示し、カスタムクエリを実行できます。Mobile SDK 6.0 では、この画面にストアピッカーが追加され、検査するグローバルストアまたはユーザーストアを選択できます。

## [Dev Info (開発情報)] 画面の拡張

[Dev Info (開発情報)] 画面は、画面に表示する情報を `SalesforceSDKManager` の `getDevSupportInfos` というメソッドから取得します。`SalesforceSDKManager` の一部のサブクラスはこのメソッドを上書きします。独自のカスタム情報を表示するために、このメソッドを上書きすることもできます。そのためには、`SalesforceSDKManager` のサブクラスを実装し、そのメソッドを上書きして、アプリケーションで使用します。

### iOS

```
/**
 * @return Array of name1, value1, name2, value2, etc.) to show in
 SFSDKDevInfoViewController
 */
- (NSArray*) getDevSupportInfos;
```

### Android

```
/**
 * @return Dev info (list of name1, value1, name2, value2, etc.) to show in DevInfoActivity
 */
public List<String> getDevSupportInfos();
```

## 第 22 章      ログイングと分析

トピック:

- [iOS のコンパイラレベルのログイング](#)
- [Android のログイングフレームワーク](#)
- [計測およびイベント収集](#)

Mobile SDK は分析やログイングを独自の目的で使用するほか、アプリケーションのログイングフレームワークも提供します。分析機能はアプリケーションで無効にでき、ログイングフレームワークを使用してカスタムコンポーネントのメッセージを出力できます。

## iOS のコンパイラレベルのロギング

MobileSDK7.0 では、iOS のロギングが簡略化され、基盤となるオペレーティングシステムフレームワークに委託します。

ロギングシステムにアクセスするには、`os_log()` 関数をコールします。この関数では、Apple の統合ロギングシステムにアクセスできます。必要に応じて、カスタムコンポーネントログオブジェクトを渡し、ログレベルを設定することもできます。詳細は <https://developer.apple.com/documentation/os/logging> を参照してください。

 **メモ:** 現在、iOS での Salesforce ロギングフレームワークは、外部ユーザーにはお勧めしません。SFSDKLogger を使用する従来のコードがある場合、次のようにして引き続き使用できます。

1. SFSDKLogger を使用する各ソースファイルで次の行を置き換えます。

```
#import <SalesforceAnalytics/SFSDKLogger.h>
```

置換後

```
#import <SalesforceSDKCommon/SFLogger.h>
```

2. Xcode Refactor を使用して、SFSDKLogger のすべてのインスタンスを SFLogger に置き換えます。

 **例:** Swift forceios テンプレートの SalesforceLogger コールを次のように置き換えることができます。これらの単純化した例では、デフォルトのコンポーネントローガーを使用して Xcode コンソールのデバッグメッセージをログに記録しています。

```
RestClient.shared.send(request: request)
{ [weak self] (result) in
 switch result {
 case .success(let response):
 self?.handleSuccess(response: response, request: request)
 case .failure(let error):
 // SalesforceLogger.d(RootViewController.self,
 // message:"Error invoking: \(request) , \(error)")
 os_log("\nError invoking: %@", log: .default, type: .debug, request)
 }
}
```

## Android のロギングフレームワーク

Mobile SDK には、開発者が簡単にアプリケーションコンポーネントのログを作成できるロギングフレームワークが含まれています。

このフレームワークを使用すると、次のことを実行できます。

- ログをコンソールとファイルの両方に出力する。
- カスタムコンポーネントと Mobile SDK コンポーネントのログを出力する。
- コンポーネントレベルでログを記録する。たとえば、Mobile Sync と SmartStore に別々のローガーを使用することや、独自のコンポーネント用のカスタムローガーを作成することができます。
- ロギングレベル (ERROR、DEBUG、VERBOSE) を設定する。



- コンポーネントごとにロギングレベルを設定する。たとえば、Mobile Sync は ERROR レベル、SmartStore は VERBOSE というように設定できます。

ロギングフレームワークを使用するということは、ジュークボックスの曲を流すようなものです。ロガーインスタンスを取得して使用する手順は、次のとおりです。

1. 曲目を調べる—アプリケーションの使用可能なコンポーネントのリストを取得します。コンポーネントの名前を把握している場合は、ランタイムのみのこの手順は必要ありません。
2. 曲を選定する—コンポーネントの名前をフレームワークに渡します。選んだものを「再生」するために、Mobile SDK が選択されたコンポーネントのロガーインスタンスを返します。ロガーが存在するかどうかは関係ありません。必要があれば Mobile SDK が作成します。
3. さぁ踊りましょう!—ロギングレベルを設定し、ログに行を書き込んで、ログの行数を制限し、ロギングをオフまたはオンにします。

さほど上手い喻えではありませんでしたが、要点はつかめたでしょうか? ここから詳しく説明していきます。

## コンポーネントのロガーの作成

コンポーネントは、アプリケーションの機能の仮想ドメインを表します。コンポーネントがアプリケーション全体を表すこともあれば、機能セットの一部のみを表すこともあります。ロガーを作成するときにこのコンポーネントの名前を考えます。次にこのロガーを使用して、コンポーネントのドメインで生じている顕著な状態を記録します。

独自のコンポーネントのロガーを作成するために、何かを上書き、拡張、実装する必要はありません。代わりに、コンポーネントの名前を使用して、ロギングフレームワークからロガーを要求します。ロガーがコンポーネントリストにまだない場合は、ロギングフレームワークがシングルトンロガーを作成し、そのコンポーネントをリストに追加します。リストになれば、既存のロガーが返されます。すべてのスレッドがこのロガーを共有します。

1つのロガーを作成することも、複数のロガーを作成することもできます。たとえば、アプリケーションのすべてのメッセージを同じレベルでログに記録する場合は、アプリケーション名を使用するロガーを1つ作成します。通常は、アプリケーションにロガーが1つあれば十分です。他方、アプリケーションの部分ごとにロギングレベルを変える場合は、複数のロガーを作成して、コンポーネントに基づく名前を付けます。ただし、やり過ぎないように注意します。ロギングアクティビティが過剰になれば、アプリケーションのパフォーマンスが低下しかねません。

次のオブジェクト経由でロガーを要求します。

- **Android:** `com.salesforce.androidsdk.analytics.logger.SalesforceLogger`

カスタムコンポーネントや標準の Mobile SDK コンポーネントのロガーを取得する場合は、次の API を使用します。

### Android

```
public synchronized static SalesforceLogger getLogger(String componentName, Context context);
```

次に例を示します。

```
SalesforceLogger.getLogger("MyComponent", context);
```

## すべてのコンポーネントの取得

次の `SalesforceLogger` メソッドは、現在ロガーに関連付けられているすべてのコンポーネントの名前のリストを返します。Mobile SDK の内部コンポーネントの名前を知らなくても、実行時にコードがこのリストから見つけ出します。このリストを使用して、すべてのコンポーネントのロギングをオフにすることもできます。アプリケーションがこの情報をどのように使用するかは、各自のビジネスロジックによって異なります。

### Android

```
public synchronized static Set<String> getComponents();
```

## ロガーの出力のカスタマイズ

ロガーインスタンスを取得したら、ロガーが出力する情報の種類と量を制御できます。

- コンポーネントのログレベルの設定

アプリケーションコンポーネントにそれぞれのロガーインスタンスがあり、独自のログレベルが設定されています。デフォルトのログレベルは、デバッグビルドが `DEBUG`、リリースビルドが `ERROR` です。コンポーネントのログレベルを設定する場合は、次の API を使用します。

### Android

```
public void setLogLevel(Level level);
```

Android ロガーは、Android のデフォルトのログレベルをそのまま反映する内部公開列挙を使用します。次に例を示します。

```
logger.setLogLevel(Level.INFO);
```

- ログ行の書き込み

新しいフレームワークを使用してログを書き込む場合は、次の API を使用できます。コンポーネントのファイルロギングが無効になっている場合を除き、ログ行がコンソールとファイルの両方に自動的に書き込まれます。

### Android

```
public void log(Level level, String tag, String message);
```

```
public void log(Level level, String tag, String message, Throwable e);
```

- ファイルロギングの有効化または無効化

ロギングフレームワークはデフォルトで、メッセージをコンソールとファイルの両方のログに記録します。ファイルロギングは必要に応じて、次の API を使用して無効にできます。

### Android

```
public synchronized void disableFileLogging();
```

ファイルロギングの無効化:

```
public synchronized void enableFileLogging(int maxSize);
```

この API の `maxSize` は、ファイルが保持可能なログの最大行数を表します。この数を超えるとログ行が集約されます。

## Mobile SDK のロギングコンポーネント

Mobile SDK では、標準コンポーネントごとにデフォルトのロガーを用意しています。こうしたアーキテクチャにより、各コンポーネントのログレベルを他のコンポーネントとは関係なく管理できます。たとえば、Mobile Sync のログレベルは INFO、SmartStore のログレベルは ERROR というように設定できます。

以下は、Mobile SDK の標準コンポーネントのロガーのリストです。

### Android

- `SalesforceAnalyticsLogger`
- `SalesforceSDKLogger`
- `SmartStoreLogger`
- `MobileSyncLogger`
- `SalesforceReactLogger`
- `SalesforceHybridLogger`

上記のロギングコンポーネントのそれぞれに、これらのコアコンポーネントに関連付けられたログレベルとロギングメッセージを調整するための便利なメソッドが用意されています。

## 計測およびイベント収集

Mobile SDK 5.0 には、分析計測を Mobile SDK アプリケーションに追加する新しいフレームワークが導入されています。アプリケーションはこの計測を使用して、使用中のアプリケーションによる Mobile SDK の使用状況を説明するイベントデータを収集します。Mobile SDK は、これらのイベントログを定期的に Salesforce クラウドにアップロードします。この情報は、顧客にとって特に重要な機能を絞り込むうえで役立ちます。ユーザーまたはその Salesforce 組織に固有のデータが収集されるわけではありません。

Mobile SDK アプリケーションのユーザーおよび開発者は、Salesforce が収集する情報にアクセスできません。

Salesforce は、独自の使用のみを目的に情報を収集します。データを収集するソフトウェアは、

[github.com/forcedotcom](https://github.com/forcedotcom) にある Mobile SDK オープンソースリポジトリに保持されます。

Mobile SDK 5.0 以降は、計測がデフォルトで有効になっています。Mobile SDK は、収集されたフレームワークイベントを次のスケジュールで Salesforce クラウドに自動的に公開します。

- **iOS:** アプリケーションがバックグラウンドに移動したとき。
- **Android:** 8 時間ごと。

アプリケーションでイベントのログ記録のオン/オフを切り替えることができます。Android の場合は、アプリケーションで収集のアップロード頻度も変更できます。

イベントのログ記録サービスを管理するには、次の API を使用します。アプリケーションの現在のユーザーアカウントを使用して初期化する、分析マネージャーオブジェクトのインスタンスで各 API をコールします。

## イベントのログ記録の切り替え

### Android

Android の場合は、`enableLogging(boolean enabled)` メソッドをコールします。

```
final UserAccount curAccount = UserAccountManager.getInstance().getCurrentUser();
final SalesforceAnalyticsManager sfAnalyticsManager =
 SalesforceAnalyticsManager.getInstance(curAccount);
sfAnalyticsManager.enableLogging(false);
```

### iOS

iOS の場合は、`BOOL loggingEnabled` プロパティを設定します。

```
SFUserAccount *account = [SFUserAccountManager
sharedInstance].currentUser;
SFSDKSalesforceAnalyticsManager *sfAnalyticsManager =
 [SFSDKSalesforceAnalyticsManager
sharedInstanceWithUser:account];
sfAnalyticsManager.loggingEnabled = NO;
```

## イベントのログ記録状況の確認

### Android

Android の場合は、`isLoggingEnabled(boolean enabled)` メソッドをコールします。

```
final UserAccount curAccount = UserAccountManager.getInstance().getCurrentUser();
final SalesforceAnalyticsManager sfAnalyticsManager =
 SalesforceAnalyticsManager.getInstance(curAccount);
boolean enabled = sfAnalyticsManager.isLoggingEnabled();
```

### iOS

iOS の場合は、`BOOL isLoggingEnabled` プロパティを確認します。

```
SFUserAccount *account = [SFUserAccountManager sharedInstance].currentUser;
SFSDKSalesforceAnalyticsManager *sfAnalyticsManager =
 [SFSDKSalesforceAnalyticsManager sharedInstanceWithUser:account];
BOOL enabled = sfAnalyticsManager.isLoggingEnabled;
```

## アップロード頻度の設定 (Android のみ)

Android では、イベントログのアップロード頻度を設定できます (時間単位)。デフォルト値は、8 です。

```
final UserAccount curAccount = UserAccountManager.getInstance().getCurrentUser();
final SalesforceAnalyticsManager sfAnalyticsManager =
 SalesforceAnalyticsManager.getInstance(curAccount);
sfAnalyticsManager.setPublishFrequencyInHours(numHours); // numHours is the desired upload
interval in hours.
```

## 第 23 章      以前のリリースからの移行

トピック:

- [11.0 から 11.1 へのすべてのアプリケーションの移行](#)
- [以前のリリースからの移行](#)

Salesforce Mobile SDK 11.0 を使用して作成されたアプリケーションをアップグレードする場合は、次の手順に従ってアプリケーションをバージョン 11.1 に更新します。

Salesforce Mobile SDK 11.0 より前のバージョンを使用して作成されたアプリケーションをアップグレードする場合は、「[以前のリリースからの移行](#)」を参照してアップグレードを開始します。

## 11.0 から 11.1 へのすべてのアプリケーションの移行

Mobile SDK 11.1 を最も簡単に移行できるのは、Salesforce CLI プラグインまたは Mobile SDK npm スクリプトを使用した場合です。

アップグレードを始める前に、「[Mobile SDK 11.1 の新機能](#)」(ページ 11)で 11.1 の新機能についてご確認ください。Mobile SDK 11.1 でコードを変更する必要はありません。

### ネイティブ iOS (Swift、Objective-C)

- サポートされるバージョンの iOS と Xcode がインストールされていることを確認します。「[iOS の基本的な要件](#)」を参照するか、「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
- 次のいずれかのオプションを選択します。
  - (推奨) Salesforce CLI Mobile SDK プラグインまたは `forceios` npm スクリプトを使用する: アプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。  

```
sf mobilesdk:ios:create help
```

  
`forceios` の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。
  - CocoaPods を使用する: CocoaPods を使用して手動でアプリケーションを作成した場合は、「[Mobile SDK Pod の更新](#)」を参照してください。
  - 手動で行う: Swift の新しいネイティブテンプレートアプリケーションを手動で作成します。「[iOS Swift プロジェクトの手動作成](#)」の手順に従います。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処します。「[iOS で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

### ネイティブ Android (Java、Kotlin)

- サポートされるバージョンの Android SDK と Android Studio がインストールされていることを確認します。「[ネイティブ Android の要件](#)」を参照するか、「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
- 次のいずれかのオプションを選択します。
  - (推奨) Salesforce CLI Mobile SDK プラグインまたは `forcedroid` npm スクリプトを使用する: アプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。  

```
sf mobilesdk:android:create help
```

  
`forcedroid` の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。

- **Maven** を使用する: Maven 使用して手動でアプリケーションを作成した場合は、「[Maven を使用した Android アプリケーションの Mobile SDK ライブラリの更新](#)」を参照してください。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処するという作業を繰り返して、ビルドを成功させます。「[Android で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

#### React Native

- Salesforce CLI Mobile SDK プラグインまたは forcereact npm スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:reactnative:create help
```

forcereact の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

#### ハイブリッド

- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
- Salesforce CLI Mobile SDK プラグインまたは forcehybrid npm スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:hybrid:create help
```

forcehybrid の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ハイブリッドアプリケーションの更新」の手順に従います。

- アプリケーションを再作成したら、次を行います。
  - アプリケーションのアーティファクトを新しいテンプレートに移行します。
  - サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
  - 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。



## 関連トピック

- [Salesforce DX の設定](#)

## 以前のリリースからの移行

---

以前のリリースよりも古いバージョンから移行するには、現在のバージョンから始めて、介在するリリースごとにコードのアップグレード手順を実行します。

### 10.2 から 11.0 へのすべてのアプリケーションの移行

Mobile SDK 11.0 を最も簡単に移行できるのは、Salesforce CLI プラグインまたは Mobile SDK npm スクリプトを使用した場合です。

アップグレードを始める前に、「[Mobile SDK 11.1 の新機能](#)」(ページ 11)で 11.0 の新機能についてご確認ください。Mobile SDK 11.0 でコードを変更する必要はありません。

ネイティブ iOS (Swift、Objective-C)

- サポートされるバージョンの iOS と Xcode がインストールされていることを確認します。「[iOS の基本的な要件](#)」を参照するか、「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
- 次のいずれかのオプションを選択します。
  - (推奨) Salesforce CLI Mobile SDK プラグインまたは `forceios npm` スクリプトを使用する: アプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。  

```
sf mobilesdk:ios:create help
```

  
`forceios` の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。
  - CocoaPods を使用する: CocoaPods を使用して手動でアプリケーションを作成した場合は、「[Mobile SDK Pod の更新](#)」を参照してください。
  - 手動で行う: Swift の新しいネイティブテンプレートアプリケーションを手動で作成します。「[iOS Swift プロジェクトの手動作成](#)」の手順に従います。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処します。「[iOS で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

### ネイティブ Android (Java、Kotlin)

- サポートされるバージョンの Android SDK と Android Studio がインストールされていることを確認します。  
「[ネイティブ Android の要件](#)」を参照するか、「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
- 次のいずれかのオプションを選択します。
  - (推奨) **Salesforce CLI Mobile SDK** プラグインまたは **forcedroid npm** スクリプトを使用する: アプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。  

```
sf mobilesdk:android:create help
```

  
forcedroid の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。
  - **Maven** を使用する: Maven 使用して手動でアプリケーションを作成した場合は、「[Maven を使用した Android アプリケーションの Mobile SDK ライブラリの更新](#)」を参照してください。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処するという作業を繰り返して、ビルドを成功させます。「[Android で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

### React Native

- Salesforce CLI Mobile SDK プラグインまたは **forcereact npm** スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。  

```
sf mobilesdk:reactnative:create help
```

  
forcereact の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

### ハイブリッド

- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。

- Salesforce CLI Mobile SDK プラグインまたは forcehybrid npm スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:hybrid:create help
```

forcehybrid の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ハイブリッドアプリケーションの更新」の手順に従います。

- アプリケーションを再作成したら、次を行います。
  - アプリケーションのアーティファクトを新しいテンプレートに移行します。
  - サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
  - 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

## 関連トピック

- [Salesforce DX の設定](#)

## 10.1 から 10.2 へのすべてのアプリケーションの移行

Mobile SDK 10.2 を最も簡単に移行できるのは、Salesforce CLI プラグインまたは Mobile SDK npm スクリプトを使用した場合です。

アップグレードを始める前に、「[Mobile SDK 11.1 の新機能](#)」(ページ 11)で 10.2 の新機能についてご確認ください。Mobile SDK 10.2 でコードを変更する必要はありません。

### ネイティブ iOS (Swift、Objective-C)

- サポートされるバージョンの iOS と Xcode がインストールされていることを確認します。「[iOS の基本的な要件](#)」を参照するか、「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
- 次のいずれかのオプションを選択します。
  - (推奨) Salesforce CLI Mobile SDK プラグインまたは forceios npm スクリプトを使用する: アプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:ios:create help
```

forceios の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。
  - CocoaPods を使用する: CocoaPods を使用して手動でアプリケーションを作成した場合は、「[Mobile SDK Pod の更新](#)」を参照してください。
  - 手動で行う: Swift の新しいネイティブテンプレートアプリケーションを手動で作成します。「[iOS Swift プロジェクトの手動作成](#)」の手順に従います。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。

- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処します。「[iOS で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

#### ネイティブ Android (Java、Kotlin)

- サポートされるバージョンの Android SDK と Android Studio がインストールされていることを確認します。「[ネイティブ Android の要件](#)」を参照するか、「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
- 次のいずれかのオプションを選択します。
  - (推奨) **Salesforce CLI Mobile SDK** プラグインまたは **forcedroid npm** スクリプトを使用する: アプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:android:create help
```

forcedroid の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「[ネイティブアプリケーションと React Native アプリケーションの更新](#)」の手順に従います。
  - **Maven** を使用する: Maven 使用して手動でアプリケーションを作成した場合は、「[Maven を使用した Android アプリケーションの Mobile SDK ライブラリの更新](#)」を参照してください。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処するという作業を繰り返して、ビルドを成功させます。「[Android で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

#### React Native

- Salesforce CLI Mobile SDK プラグインまたは **forcereact npm** スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:reactnative:create help
```

forcereact の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「[ネイティブアプリケーションと React Native アプリケーションの更新](#)」の手順に従います。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

## ハイブリッド

- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
- Salesforce CLI Mobile SDK プラグインまたは `forcehybrid` npm スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:hybrid:create help
```

`forcehybrid` の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ハイブリッドアプリケーションの更新」の手順に従います。

- アプリケーションを再作成したら、次を行います。
  - アプリケーションのアーティファクトを新しいテンプレートに移行します。
  - サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
  - 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

## 関連トピック

- [Salesforce DX の設定](#)

## 10.0 から 10.1 へのすべてのアプリケーションの移行

Mobile SDK 10.1 を最も簡単に移行できるのは、Salesforce CLI プラグインまたは Mobile SDK npm スクリプトを使用した場合です。

アップグレードを始める前に、「[Mobile SDK 11.1 の新機能](#)」(ページ 11)で 10.1 の新機能についてご確認ください。Mobile SDK 10.1 でコードを変更する必要はありません。

### ネイティブ iOS (Swift、Objective-C)

- サポートされるバージョンの iOS と Xcode がインストールされていることを確認します。「[iOS の基本的な要件](#)」を参照するか、「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
- 次のいずれかのオプションを選択します。

- (推奨) Salesforce CLI Mobile SDK プラグインまたは `forceios` npm スクリプトを使用する: アプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:ios:create help
```

`forceios` の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。

- CocoaPods を使用する: CocoaPods を使用して手動でアプリケーションを作成した場合は、「[Mobile SDK Pod の更新](#)」を参照してください。
- 手動で行う: Swift の新しいネイティブテンプレートアプリケーションを手動で作成します。「[iOS Swift プロジェクトの手動作成](#)」の手順に従います。



アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処します。「[iOS で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

#### ネイティブ Android (Java、Kotlin)

- サポートされるバージョンの Android SDK と Android Studio がインストールされていることを確認します。「[ネイティブ Android の要件](#)」を参照するか、「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
- 次のいずれかのオプションを選択します。
  - (推奨) Salesforce CLI Mobile SDK プラグインまたは `forcedroid npm` スクリプトを使用する: アプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:android:create help
```

`forcedroid` の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。
  - `Maven` を使用する: `Maven` 使用して手動でアプリケーションを作成した場合は、「[Maven を使用した Android アプリケーションの Mobile SDK ライブラリの更新](#)」を参照してください。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処するという作業を繰り返して、ビルドを成功させます。「[Android で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

#### React Native

- Salesforce CLI Mobile SDK プラグインまたは `forcereact npm` スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:reactnative:create help
```

`forcereact` の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。

- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

#### ハイブリッド

- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
- Salesforce CLI Mobile SDK プラグインまたは `forcehybrid` npm スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:hybrid:create help
```

`forcehybrid` の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ハイブリッドアプリケーションの更新」の手順に従います。

- アプリケーションを再作成したら、次を行います。
  - アプリケーションのアーティファクトを新しいテンプレートに移行します。
  - サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
  - 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

## 関連トピック

- [Salesforce DX の設定](#)

## 9.2 から 10.0 へのすべてのアプリケーションの移行

Mobile SDK 10.0 を最も簡単に移行できるのは、Salesforce CLI プラグインまたは Mobile SDK npm スクリプトを使用した場合です。

アップグレードを始める前に、「[Mobile SDK 11.1 の新機能](#)」(ページ 11)で 10.0 の新機能についてご確認ください。

#### ネイティブ iOS (Swift、Objective-C)

- サポートされるバージョンの iOS と Xcode がインストールされていることを確認します。「[iOS の基本的な要件](#)」または「[Mobile SDK 11.1 でサポートされるバージョンのツールおよびコンポーネント](#)」を参照してください。
- 次のいずれかのオプションを選択します。
  - Salesforce CLI Mobile SDK プラグインまたは `forceios` npm スクリプトを使用する (推奨): アプリケーションを再作成し、アプリケーションのアーティファクトを新しいテンプレートに移行します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:ios:create help
```



forceios の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。

- **CocoaPods**を使用する:CocoaPodsを使用して手動でアプリケーションを作成した場合は、「[Mobile SDK Pod の更新](#)」を参照してください。
- **手動で行う**:Swift のネイティブテンプレートアプリケーションを手動で作成してからアプリケーションのアーティファクトを新しいテンプレートに移行します。「[iOS Swift プロジェクトの手動作成](#)」の手順に従います。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- Mobile SDK 10.0 で削除される API のリストを確認し、コードベースに影響を与える項目があれば、それに対処します。「[Mobile SDK 11.0 で削除された iOS API](#)」を参照してください。
- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処します。「[iOS で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

#### ネイティブ Android (Java、Kotlin)

- サポートされるバージョンの Android SDK と Android Studio がインストールされていることを確認します。「[ネイティブ Android の要件](#)」または「[Mobile SDK 11.1 でサポートされるバージョンのツールおよびコンポーネント](#)」を参照してください。
- 次のいずれかのオプションを選択します。
  - **Salesforce CLI Mobile SDK** プラグインまたは **forcedroid npm** スクリプト (推奨) を使用する: プラグインまたはスクリプトを使用してアプリケーションを再作成し、アプリケーションのアーティファクトを新しいテンプレートに移行します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:android:create help
```

forcedroid の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。

- **Maven** を使用する:Maven 使用して手動でアプリケーションを作成した場合は、「[Maven を使用した Android アプリケーションの Mobile SDK ライブラリの更新](#)」を参照してください。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- Mobile SDK 10.0 で削除される API のリストを確認し、コードベースに影響を与える項目があれば、それに対処します。「[Mobile SDK 11.0 で削除された Android API](#)」を参照してください。
- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処するという作業を繰り返して、ビルドを成功させます。「[Android で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。

- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

### React Native

Mobile SDK 10.0 でコードを変更する必要はありません。

- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Mobile SDK 11.1 でサポートされるバージョンのツールおよびコンポーネント](#)」を参照してください。
- Salesforce CLI Mobile SDK プラグインまたは `forcereact npm` スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:reactnative:create help
```

`forcereact` ヘルプには次のように入力します。

```
forcereact
```

- アプリケーションを再作成したら、アプリケーションのアイテムを新しいテンプレートに移行します。

### ハイブリッド

Mobile SDK 10.0 でコードを変更する必要はありません。

- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Mobile SDK 11.1 でサポートされるバージョンのツールおよびコンポーネント](#)」を参照してください。
- Salesforce CLI Mobile SDK プラグインまたは `forcehybrid npm` スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:hybrid:create help
```

`forcehybrid` の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ハイブリッドアプリケーションの更新」の手順に従います。

## 関連トピック

- [Salesforce DX の設定](#)

## 9.1 から 9.2 へのすべてのアプリケーションの移行

Mobile SDK 9.2 を最も簡単に移行できるのは、Salesforce CLI プラグインまたは Mobile SDK npm スクリプトを使用した場合です。

アップグレードを始める前に、「[Mobile SDK 11.1 の新機能](#)」(ページ 11)で 9.2 の新機能についてご確認ください。

### ネイティブ iOS (Swift、Objective-C)

- サポートされるバージョンの iOS と Xcode がインストールされていることを確認します。「[iOS の基本的な要件](#)」を参照するか、「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
- 次のいずれかのオプションを選択します。

- **Salesforce CLI Mobile SDK** プラグインまたは **forceios npm** スクリプトを使用する (推奨): アプリケーションを再作成し、アプリケーションのアーティファクトを新しいテンプレートに移行します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:ios:create help
```

forceios の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。

- **CocoaPods** を使用する: CocoaPods を使用して手動でアプリケーションを作成した場合は、「[Mobile SDK Pod の更新](#)」を参照してください。
- **手動で行う**: Swift のネイティブテンプレートアプリケーションを手動で作成してからアプリケーションのアーティファクトを新しいテンプレートに移行します。「[iOS Swift プロジェクトの手動作成](#)」の手順に従います。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処します。「[iOS で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

キーストアの暗号化のアップグレード

Mobile SDK 9.2 では、キーストアの暗号化がアップグレードされます。ほとんどのアプリケーションでは、アップグレードは自動で行われますが、一部のアプリケーションでは追加手順が必要になります。「[アプリケーションでの暗号化のアップグレード](#)」(ページ 489)を参照してください。

ネイティブ Android (Java、Kotlin)

- サポートされるバージョンの Android SDK と Android Studio がインストールされていることを確認します。「[ネイティブ Android の要件](#)」を参照するか、「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
- 次のいずれかのオプションを選択します。
  - **Salesforce CLI Mobile SDK** プラグインまたは **forcedroid npm** スクリプト (推奨) を使用する: プラグインまたはスクリプトを使用してアプリケーションを再作成し、アプリケーションのアーティファクトを新しいテンプレートに移行します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:android:create help
```

forcedroid の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。

- **Maven** を使用する: Maven を使用して手動でアプリケーションを作成した場合は、「[Maven を使用した Android アプリケーションの Mobile SDK ライブラリの更新](#)」を参照してください。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。

- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処するという作業を繰り返して、ビルドを成功させます。「[Android で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

### React Native

Mobile SDK 9.2 でコードを変更する必要はありません。

- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
- Salesforce CLI Mobile SDK プラグインまたは forcereact npm スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:reactnative:create help
```

forcereact には次のように入力します。

```
forcereact create
```

- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
- アプリケーションを再作成したら、アプリケーションのアイテムを新しいテンプレートに移行します。

### ハイブリッド

Mobile SDK 9.2 でコードを変更する必要はありません。

- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」 Trailhead プロジェクトを使用してください。
- Salesforce CLI Mobile SDK プラグインまたは forcehybrid npm スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:hybrid:create help
```

forcehybrid の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ハイブリッドアプリケーションの更新」の手順に従います。

## 関連トピック

- [Salesforce DX の設定](#)

## 9.0 から 9.1 へのすべてのアプリケーションの移行

Mobile SDK 9.1 を最も簡単に移行できるのは、Salesforce CLI プラグインまたは Mobile SDK npm スクリプトを使用した場合です。

アップグレードを始める前に、「[Mobile SDK 11.1 の新機能](#)」(ページ 11)で 9.1 の新機能についてご確認ください。

#### ネイティブ iOS (Swift、Objective-C)

- サポートされるバージョンの iOS と Xcode がインストールされていることを確認します。「[iOS の基本的な要件](#)」を参照するか、「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
- 次のいずれかのオプションを選択します。
  - **Salesforce CLI Mobile SDK プラグイン**または **forceios npm スクリプト**を使用する (推奨): アプリケーションを再作成し、アプリケーションのアーティファクトを新しいテンプレートに移行します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:ios:create help
```

forceios の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。

- **CocoaPods**を使用する:CocoaPodsを使用して手動でアプリケーションを作成した場合は、「[Mobile SDK Pod の更新](#)」を参照してください。
- 手動で行う:Swift のネイティブテンプレートアプリケーションを手動で作成してからアプリケーションのアーティファクトを新しいテンプレートに移行します。「[iOS Swift プロジェクトの手動作成](#)」の手順に従います。

アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処します。「[iOS で現在非推奨のもの](#)」を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。「[Mobile SDK 11.1 の新機能](#)」を参照してください。

#### ネイティブ Android (Java、Kotlin)

- サポートされるバージョンの Android SDK と Android Studio がインストールされていることを確認します。「[ネイティブ Android の要件](#)」を参照するか、「[Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)](#)」Trailhead プロジェクトを使用してください。
- 次のいずれかのオプションを選択します。
  - **Salesforce CLI Mobile SDK プラグイン**または **forcedroid npm スクリプト** (推奨)を使用する: プラグインまたはスクリプトを使用してアプリケーションを再作成し、アプリケーションのアーティファクトを新しいテンプレートに移行します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:android:create help
```

forcedroid の場合は、「[Mobile SDK アプリケーション \(5.0 以降\) の更新](#)」の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。

- **Maven**を使用する:Maven 使用して手動でアプリケーションを作成した場合は、「[Maven を使用した Android アプリケーションの Mobile SDK ライブラリの更新](#)」を参照してください。



アプリケーションを再作成したら、次を行います。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- 今後削除されるために非推奨になっていた API のリストを確認し、コードベースに影響を与える項目があれば、それに対処するという作業を繰り返して、ビルドを成功させます。[「Android で現在非推奨のもの」](#)を参照してください。
- ビルドが成功したら、コンパイラーの警告を確認し、非推奨の項目や Mobile SDK に関するその他の問題の見逃しがないかを調べます。
- 新機能を採用することを検討します。[「Mobile SDK 11.1 の新機能」](#)を参照してください。

## React Native

Mobile SDK 9.1 でコードを変更する必要はありません。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- Salesforce CLI Mobile SDK プラグインまたは forcereact npm スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:reactnative:create help
```

forcereact の場合は、[「Mobile SDK アプリケーション \(5.0 以降\) の更新」](#)の「ネイティブアプリケーションと React Native アプリケーションの更新」の手順に従います。

- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。[「Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)」](#) Trailhead プロジェクトを使用してください。
- 新機能を採用することを検討します。[「Mobile SDK 11.1 の新機能」](#)を参照してください。

## ハイブリッド

Mobile SDK 9.1 でコードを変更する必要はありません。

- アプリケーションのアーティファクトを新しいテンプレートに移行します。
- サポートされるバージョンの対象となるモバイルプラットフォームがインストールされていることを確認します。[「Set Up Your Mobile SDK Developer Tools \(Mobile SDK 開発者ツールの設定\)」](#) Trailhead プロジェクトを使用してください。
- Salesforce CLI Mobile SDK プラグインまたは forcehybrid npm スクリプトを使用してアプリケーションを再作成します。Salesforce CLI の場合は、次のように入力して、コマンドラインの指示に従います。

```
sf mobilesdk:hybrid:create help
```

forcehybrid の場合は、[「Mobile SDK アプリケーション \(5.0 以降\) の更新」](#)の「ハイブリッドアプリケーションの更新」の手順に従います。

## 関連トピック

- [Salesforce DX の設定](#)

## 第 24 章      リファレンス

トピック:

- サポート対象の Salesforce API
- Files API リファレンス
- iOS アーキテクチャ
- Android アーキテクチャ
- Mobile SDK 11.1 でサポートされるバージョンのツールおよびコンポーネント

リファレンスドキュメントは、GitHub で提供されています。

- iOS の場合:
  - *SalesforceAnalytics* ライブラリリファレンス:  
<https://forcedotcom.github.io/SalesforceMobileSDK-iOS/Documentation/SalesforceAnalytics/html/index.html>
  - *SalesforceSDKCommon* ライブラリリファレンス:  
<https://forcedotcom.github.io/SalesforceMobileSDK-iOS/Documentation/SalesforceSDKCommon/html/index.html>
  - *SalesforceSDKCore* ライブラリリファレンス:  
<https://forcedotcom.github.io/SalesforceMobileSDK-iOS/Documentation/SalesforceSDKCore/html/index.html>
  - *SmartStore* ライブラリリファレンス:  
<https://forcedotcom.github.io/SalesforceMobileSDK-iOS/Documentation/SmartStore/html/index.html>
  - *Mobile Sync* ライブラリリファレンス:  
<https://forcedotcom.github.io/SalesforceMobileSDK-iOS/Documentation/MobileSync/html/index.html>
- Android の場合: <https://forcedotcom.github.io/SalesforceMobileSDK-Android/index.html>



## サポート対象の Salesforce API

`RestRequest` クラスには、Salesforce API コールをラップするファクトリーメソッドと拡張メソッドがあります。これらのメソッドは、ネットワークコールを構成するために指定する要求パラメーターを使用します。

| Swift、Java、Kotlin        | Objective-C                |
|--------------------------|----------------------------|
| <code>RestRequest</code> | <code>SFRestRequest</code> |

`RestRequest` は、パラメーターを反映したそれ自体の専用コピーを返します。要求を Salesforce に送信するには、カスタマイズしたこのオブジェクトを Mobile SDK REST クライアントに渡します。

`RestRequest` では、次の Salesforce API がサポートされています。

このセクションの内容:

### [Batch Request](#)

サブ要求のバッチを実行します。

### [ブリーフケースプライミングレコード](#)

接続アプリケーションに割り当てられたブリーフケースからレコード ID を取得する要求オブジェクトを返します。

### [ブリーフケースプライミングレコード応答](#)

すべての Mobile SDK ブリーフケースプライミング要求の応答を処理します。

### [コレクションの作成](#)

指定したオブジェクト種別のレコードのコレクションを作成します。

### [コレクションの削除](#)

指定したオブジェクト ID に一致する、コレクション内のオブジェクトを削除します。

### [コレクションの取得](#)

指定したオブジェクト ID に一致する、指定したオブジェクト種別のオブジェクトのコレクションを取得します。

### [コレクションの更新](#)

要求したコレクションを、指定したレコードで更新します。

### [コレクションの更新/挿入](#)

外部データのオブジェクトのコレクションを更新または挿入します。

### [コレクション応答](#)

すべての Mobile SDK コレクション要求の応答を処理します。

### [Composite Request](#)

Composite 要求の実行に使用する `RestRequest` オブジェクトを返します。

### [Create](#)

指定したオブジェクト種別のレコードを作成します。

[Delete](#)

指定した種別かつ指定した ID のオブジェクトを削除します。

[Describe](#)

項目、URL、子リレーションなど、オブジェクトのあらゆるレベルのメタデータについて詳しく説明します。

[Describe Global](#)

組織で利用できるすべてのオブジェクトとそのメタデータのリストを返します。

[メタデータ](#)

指定したオブジェクトの sObject Basic Information によって提供されたメタデータについて説明します。

[通知](#)

通知 ID を基に 1 件の通知を取得します。

[Notification Update](#)

指定された ID を持ち、「read」 (null でない場合) と「seen」 (null でない場合) 状況の通知を更新します。

[通知](#)

指定された日付の「前」または「後」について、指定された数(最大 20)のアーカイブされた Notification Builder 通知を取得します。

[Notifications Status](#)

未読数や未表示数など、現在のユーザーの通知の状況を取得します。

[Notifications Update](#)

指定された ID を持ち、「read」 (null でない場合) と「seen」 (null でない場合) 状況の通知、または指定された日付よりも前の通知を更新します。

[Object Layout](#)

指定したオブジェクト種別とパラメーターのレイアウトメタデータを取得します。

[リソース](#)

リソース名および URI を含む、指定された API バージョンで使用可能なリソースを取得します。

[Retrieve](#)

オブジェクト ID に基づいて 1 つの sObject レコードを取得します。

[SOSL Search](#)

指定した SOSL 検索を実施します。

[Search Result Layout](#)

1 回のクエリのオブジェクトの最大数が 100 となる検索結果レイアウトを取得します。

[Search Scope and Order](#)

現在のユーザーのデフォルトのグローバル検索範囲内にあるオブジェクトの順序付きリストを取得します。

[SObject Tree](#)

指定したオブジェクト種別のルートレコードを持つ 1 つ以上の sObject ツリーを作成します。

[SOQL Query](#)[SOQL Query All](#)

指定した SOQL 文字列を実行します。結果には、クエリを満たす、現在のオブジェクトおよび削除済みオブジェクトがすべて含まれます。

### ユーザー情報

現在のユーザーに関連付けられている情報を返します。

#### Update

要求されたレコードの指定された項目を指定の値で更新します。また、特定の日付以降にレコードが変更されている場合には更新が行われないようにすることもできます。

#### Upsert

外部データのオブジェクトを更新または挿入します。

#### Versions

Salesforce API の現在使用可能な各バージョンのサマリー情報を取得します。

関連トピック:

[Mobile SDK での Salesforce REST API の使用](#)

[REST API の使用](#)

## Batch Request

サブ要求のバッチを実行します。

`RestRequest` オブジェクトのリストで指定された最大 25 個のサブ要求のバッチを含む `RestRequest` オブジェクトを返します。各サブ要求は、レート制限に含まれます。

## パラメーター

- `requests` (配列/リスト)
- `haltOnError` (Boolean)
- `apiVersion` (文字列)

## iOS

### Swift

```
RestClient.shared.batchRequest(requests:haltOnError:apiVersion:)
```

### Objective-C

```
- (SFRestRequest *) batchRequest:(NSArray<SFRestRequest *> *) requests
 haltOnError:(BOOL) haltOnError
 apiVersion:(nullable NSString *) apiVersion;
```

## Android

### Java

```
public static BatchRequest getBatchRequest(String apiVersion, boolean haltOnError,
List<RestRequest> requests) throws JSONException
```

## 関連トピック

- [Batch 要求および Composite 要求](#) (ページ 104) (iOS)
- [Batch 要求および Composite 要求](#) (ページ 162) (Android)
- 『[REST API 開発者ガイド](#)』の「[Composite Batch](#)」

## ブリーフケースプライミングレコード

接続アプリケーションに割り当てられたブリーフケースからレコード ID を取得する要求オブジェクトを返します。

Mobile SDK では、ブリーフケースプライミングレコード要求の結果を解析するためのカスタム応答オブジェクトが提供されています。「[ブリーフケースプライミングレコード応答](#)」を参照してください。

## リレートークンを使用したレコード ID の取得

次のように、リレートークンにより制御される反復ループでレコード ID のバッチを取得します。

- 最初の要求で `relayToken` を `null` に設定します。この要求の応答には新しいリレートークンが含まれます。
- 後続の各要求で `relayToken` を前の応答のリレートークン値に設定します。

要求でタイムスタンプを設定すると、特定の時間以降に変更されたレコードのみを取得することもできます。

## パラメーター

- API バージョン (文字列)
- リレートークン (文字列、`null` 可能)
- 「変更」タイムスタンプ (ISO タイムスタンプ、`null` 可能)

## iOS

### Swift

要求ファクトリーメソッド:

```
let request = RestClient.shared.requestForPrimingRecords(relayToken: relayToken,
 changedAfter: changedAfter, apiVersion: nil)
```

### Objective-C

要求ファクトリーメソッド:

```
- (SFRestRequest*) requestForPrimingRecords:(nullable NSString *)relayToken
 changedAfterTimestamp:(nullable NSNumber *)timestamp apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

要求ファクトリーメソッド:

```
val request =
 RestRequest.getRequestForPrimingRecords(ApiVersionStrings.getVersionNumber(this),
 relayToken, changedAfter)
```

### Java

要求ファクトリーメソッド:

```
public static RestRequest getRequestForPrimingRecords(
 String apiVersion,
 String relayToken,
 Long changedAfterTime)
```

## 関連トピック

- [『Connect REST API 開発者ガイド』の「Briefcase Priming Records リソース」](#)
- [『Connect REST API 開発者ガイド』の「Priming Record Collection」\(プライミングレコード応答\)](#)

## ブリーフケースプライミングレコード応答

すべての Mobile SDK ブリーフケースプライミング要求の応答を処理します。

要求ファクトリーメソッドに加えて、Mobile SDK ではカスタムブリーフケース応答オブジェクトが提供されています。このオブジェクトのプロパティを使用して、解析されたブリーフケース応答値を取得します。

## プロパティ

- プライミングレコード (対応付けまたは辞書。レコード ID とその更新タイムスタンプが含まれます)
- リレートークン (文字列。null の場合、それ以上のレコードはありません)
- ルールエラー (文字列配列。処理されてエラーとなったプライミングルールの ID が含まれます)
- 統計 (ルール数、レコード数、処理されたルール数、処理されたレコード数を含むオブジェクト)

## iOS

### Swift

応答処理:

```
let parsedResponse =
 PrimingRecordsResponse(try response.asJson() as! [AnyHashable : Any])
let records = parsedResponse.primingRecords
let token = parsedResponse.relayToken
let ruleErrors = parsedResponse.ruleErrors
let stats = parsedResponse.stats
```

**Objective-C**

応答処理:

```

SFSDKPrimingRecordsResponse* parsedResponse = [[SFSDKPrimingRecordsResponse alloc]
 initWith:response];
NSDictionary<NSString*, NSDictionary<NSString*,
 NSArray<SFSDKPrimingRecord*>>*>* primingRecords =
 parsedResponse.primingRecords;
NSString *relayToken = parsedResponse.relayToken;
NSArray<SFSDKPrimingRuleError*>* ruleErrors = parsedResponse.ruleErrors;
SFSDKPrimingStats * stats = parsedResponse.stats;

```

**Android****Kotlin**

要求ファクトリーメソッド:

```

val parsedResponse = PrimingRecordsResponse(result.asJSONArray())
val primingRecords = parsedResponse.primingRecords
val relayToken = parsedResponse.relayToken
val ruleErrors = parsedResponse.ruleErrors
val stats = parsedResponse.stats

```

**Java**

応答処理:

```

PrimingRecordsResponse parsedResponse =
 new PrimingRecordsResponse(result.asJSONObject());
String stats = parsedResponse.stats;
Map<String, Map<String, List<PrimingRecordsResponse.PrimingRecord>>> records =
 parsedResponse.primingRecords;
String relayToken = parsedResponse.relayToken;
List<PrimingRecordsResponse.PrimingRuleError> ruleErrors = parsedResponse.ruleErrors;

```

**関連トピック**

- SalesforceMobileSDK-iOS リポジトリの [SFSDKPrimingRecordsResponse.h](#)。
- SalesforceMobileSDK-Android リポジトリの [PrimingRecordsResponse.java](#)。
- プライミングレコード要求: 『[Connect REST API 開発者ガイド](#)』の「[Briefcase Priming Records リソース](#)」
- 『[Connect REST API 開発者ガイド](#)』の「[Priming Record Collection](#)」(プライミングレコード応答)

**コレクションの作成**

指定したオブジェクト種別のレコードのコレクションを作成します。

MobileSDKでは、コレクション要求結果を解析するためのカスタム応答オブジェクトが提供されています。「[コレクション応答](#)」を参照してください。

## パラメーター

コレクションでは、オールオアナッシングパラメーターを指定することで、部分的に成功した結果を却下することができます。このパラメーターを `true` に設定すると、いずれかのレコードの作成に失敗した場合に Mobile SDK は要求全体をロールバックします。

- API バージョン (文字列)
- オールオアナッシング (Boolean)
- 項目 (配列)

## iOS

### Swift

```
let request = RestClient.shared.requestForCreate(allOrNone: allOrNone, records: records,
 apiVersion: nil)
```

### Objective-C

```
- (SFRestRequest*) requestForCollectionCreate:(BOOL)allOrNone
 records:(NSArray<NSDictionary*>*)records
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
val request =
 RestRequest.getRequestForCollectionCreate(ApiVersionStrings.getVersionNumber(this),
 allOrNone, records)
```

### Java

```
public static RestRequest getRequestForCollectionCreate(String apiVersion,
 boolean allOrNone, JSONArray records)
```

## React Native

```
collectionCreate = <T>({
 allOrNone: boolean,
 records: Array<Record<string, unknown>>,
 successCB: ExecSuccessCallback<T>,
 errorCB: ExecErrorCallback,
}): void
```

## 関連トピック

- [『REST API 開発者ガイド』の「sObject コレクション」](#)



## コレクションの削除

指定したオブジェクト ID に一致する、コレクション内のオブジェクトを削除します。

Mobile SDK では、コレクション要求結果を解析するためのカスタム応答オブジェクトが提供されています。「[コレクション応答](#)」を参照してください。

### パラメーター

コレクションでは、オールオアナッシングパラメーターを指定することで、部分的な更新を却下することができます。このパラメーターを `true` に設定すると、いずれかのレコードの削除に失敗した場合に Mobile SDK は要求全体をロールバックします。

- API バージョン (文字列)
- オールオアナッシング (Boolean)
- オブジェクト ID (配列)

### iOS

#### Swift

```
let request = RestClient.shared.requestForDelete(allOrNone: allOrNone,
 withObjects: objectIds, apiVersion: nil)
```

#### Objective-C

```
- (SFRestRequest*)
requestForCollectionDelete: (BOOL)allOrNone
 objectIds: (NSArray<NSString*>*)objectIds
 apiVersion: (nullable NSString *)apiVersion;
```

### Android

#### Kotlin

```
val request =
 RestRequest.getRequestForCollectionDelete(ApiVersionStrings.getVersionNumber(this),

 allOrNone, objectIds)
```

#### Java

```
public static RestRequest getRequestForCollectionDelete(String apiVersion,
 boolean allOrNone, List<String> objectIds)
 throws UnsupportedOperationException
```

### React Native

```
collectionDelete = <T>({
 ids: Array<string>,
 successCB: ExecSuccessCallback<T>,
```

```
errorCB: ExecErrorCallback,
) : void
```

## 関連トピック

- [『REST API 開発者ガイド』の「sObject コレクション」](#)

## コレクションの取得

指定したオブジェクト ID に一致する、指定したオブジェクト種別のオブジェクトのコレクションを取得します。

Mobile SDK では、コレクション要求結果を解析するためのカスタム応答オブジェクトが提供されています。「[コレクション応答](#)」を参照してください。

## パラメーター

項目リストを指定すると、Mobile SDK はその項目のみを取得します。指定しない場合は、アクセス可能な標準およびカスタムのすべての項目が返されます。

- API バージョン (文字列、省略可能)
- オブジェクト種別 (文字列)
- オブジェクト ID (配列)
- 項目リスト (配列)

## iOS

### Swift

```
let request = RestClient.shared.requestForRetrieve(withObjectType: objectType,
objectIds: objectIds!, fieldList: fieldList!, apiVersion: nil)
```

### Objective-C

```
- (SFRestRequest*) requestForCollectionRetrieve:(NSString*) objectType
objectIds: (NSArray<NSString*>*) objectIds
fieldList: (NSArray<NSString*>*) fieldList
apiVersion: (nullable NSString *) apiVersion;
```

## Android

### Kotlin

```
val request =
 RestRequest.getRequestForCollectionRetrieve(ApiVersionStrings.getVersionNumber(this),
objectType, objectIds, fieldList)
```

## Java

```
RestRequest getRequestForCollectionRetrieve(String apiVersion, String objectType,
List<String> objectIds, List<String> fieldList)
```

## React Native

```
collectionRetrieve = <T>(
 objectType: string,
 ids: Array<string>,
 fields: Array<string>,
 successCB: ExecSuccessCallback<T>,
 errorCB: ExecErrorCallback,
) : void
```

## 関連トピック

- [『REST API 開発者ガイド』の「sObject コレクション」](#)

## コレクションの更新

要求したコレクションを、指定したレコードで更新します。

コレクションでは、オールオアナッシングパラメーターを指定することで、部分的な更新を却下することができます。このパラメーターを `true` に設定すると、いずれかのレコードの更新に失敗した場合に Mobile SDK は要求全体をロールバックします。

Mobile SDK では、コレクション要求結果を解析するためのカスタム応答オブジェクトが提供されています。「[コレクション応答](#)」を参照してください。

## パラメーター

- API バージョン (文字列、省略可能)
- 「オールオアナッシング」設定 (Boolean)
- レコード (sObject の配列)

## iOS

## Swift

```
let request = RestClient.shared.requestForCollectionUpdate(allOrNone,
records:records, apiVersion: apiVersion)
```

## Objective-C

```
- (SFRestRequest*) requestForCollectionUpdate:(BOOL)allOrNone
 records:(NSArray<NSDictionary*>*)records
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
val request =
 RestRequest.getRequestForCollectionUpdate(ApiVersionStrings.getVersionNumber(this),

 allOrNone, records)
```

### Java

```
public static RestRequest getRequestForCollectionUpdate(String apiVersion,
 boolean allOrNone, JSONArray records) throws JSONException
```

## React Native

```
collectionUpdate= <T>(
 allOrNone: boolean,
 records: Array<Record<string, unknown>>,
 successCB: ExecSuccessCallback<T>,
 errorCB: ExecErrorCallback,
): void
```

## 関連トピック

- [『REST API 開発者ガイド』の「sObject コレクション」](#)

## コレクションの更新/挿入

外部データのオブジェクトのコレクションを更新または挿入します。

Mobile SDK では、コレクション要求結果を解析するためのカスタム応答オブジェクトが提供されています。[「コレクション応答」](#)を参照してください。

## パラメーター

Salesforce では、外部IDが現在外部ID項目に存在するかどうかに基づいて、レコードを更新または挿入します。Salesforce で強制的に新規レコードを作成するには、外部ID項目の名前を「Id」、外部ID値を null に設定します。

コレクションでは、オールオアナッシングパラメーターを指定することで、部分的な更新/挿入を却下することができます。このパラメーターを true に設定すると、いずれかのレコードの更新/挿入に失敗した場合に Mobile SDK は要求全体をロールバックします。

- API バージョン (文字列、省略可能)
- 「オールオアナッシング」設定 (Boolean)
- オブジェクト種別 (文字列)
- 外部ID項目名 (文字列)
- レコード (配列)

## iOS

### Swift

```
let request = RestClient.shared.requestForCollectionUpsert(allOrNone: allOrNone,
 withObjectType: objectType, externalIdField: externalIdFieldName!,
 records: records!, apiVersion: nil)
```

### Objective-C

```
- (SFRestRequest*)
requestForCollectionUpsert:(BOOL)allOrNone
 objectType:(NSString*)objectType
 externalIdField:(NSString*)externalIdField
 records:(NSArray<NSDictionary*>*)records
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
val request =
 RestRequest.getRequestForCollectionUpsert(ApiVersionStrings.getVersionNumber(this),

 allOrNone, objectType, externalIdFieldName, records)
```

### Java

```
public static RestRequest getRequestForCollectionUpsert(
 String apiVersion, boolean allOrNone, String objectType, String externalIdField,
 JSONArray records)
 throws JSONException
```

## React Native

```
collectionUpsert = <T>(
 allOrNone: boolean,
 objectType: string,
 externalIdField: string,
 records: Array<Record<string, unknown>>,
 successCB: ExecSuccessCallback<T>,
 errorCB: ExecErrorCallback,
) : void
```

## 関連トピック

- [『REST API 開発者ガイド』の「Object コレクション」](#)

## コレクション応答

すべての Mobile SDK コレクション要求の応答を処理します。

プロパティ:

- サブ応答 (配列)
- サブ応答プロパティ:
  - オブジェクト ID (文字列)
  - 成功 (Boolean)
  - エラー (配列)
  - JSON (辞書)

## iOS

## Swift、Objective-C

## Swift

## 応答処理:

```
let parsedResponse = CollectionResponse(try response.asJson() as! [Any])
let objId = parsedResponse.subResponses[0].objectId // String; can be nil
let success = parsedResponse.subResponses[0].success // Boolean
let errors = parsedResponse.subResponses[0].errors // Array of
CollectionErrorResponse objects with
// status code, message and
fields
```

## Objective-C

## 応答処理:

```
SFSDKCollectionResponse* parsedResponse = [[SFSDKCollectionResponse alloc]
initWith:response];
parsedResponse.subResponses[0].objectId
parsedResponse.subResponses[0].success
parsedResponse.subResponses[0].errors // NSArray<SFSDKCollectionErrorResponse*>
with status
// code, message and fields
```

## Android

## Kotlin

## 応答処理:

```
val parsedResponse = CollectionResponse(response.asJSONArray())
val objId = parsedResponse.subResponses[0].id // a string or null
val success = parsedResponse.subResponses[0].success // a boolean
val errors = parsedResponse.subResponses[0].errors // a
CollectionSubResponse.ErrorResponse object
// with status code, message
and fields
```

## Java

## 応答処理:

```
CollectionResponse parsedResponse =
 new CollectionResponse(response.asJSONArray());
String objId = parsedResponse.subResponses.get(0).id; // can be null
Boolean success =
 parsedResponse.subResponses.get(0).success;
```

```
List<CollectionResponse.ErrorResponse> errors =
 parsedResponse.subResponses.get(0).errors;
```

## 関連トピック

- SalesforceMobileSDK-iOS リポジトリの [SFSDKCollectionResponse.h](#)。
- SalesforceMobileSDK-Android リポジトリの [CollectionResponse.java](#)。
- 『REST API 開発者ガイド』の「Object コレクション」

## Composite Request

Composite 要求の実行に使用する `RestRequest` オブジェクトを返します。

- 📌 **メモ:** サブ要求の数に関係なく、各 Composite 要求は 1 つの API コールと見なされます。

## パラメーター

- `apiVersion` (文字列)
- `requests`
  - iOS
    - `requests` (配列) — サブ要求の配列
    - `refIds` (配列) — 要求の参照 ID の配列。要素の数が要求の数と一致している必要があります。
  - Android
    - `refIdToRequests` (対応付け) — `RestRequest` オブジェクトへの参照 ID の `LinkedHashMap`。対応付けられた順に要求が実行されます。
- `allOrNone` (Boolean) — エラー条件ですべての要求を 1 つのトランザクションブロックとして処理するかどうかを示すフラグを設定します。

## iOS

### Swift

```
RestClient.shared.compositeRequest(requests:refIds:allOrNone:apiVersion:)
```

### Objective-C

```
- (SFRestRequest *)compositeRequest:(NSArray<SFRestRequest *> *) requests
 refIds:(NSArray<NSString *> *) refIds
 allOrNone:(BOOL)allOrNone
 apiVersion:(nullable NSString *)apiVersion;
```



## Android

### Kotlin

```
@Throws(JSONException::class)
fun getCompositeRequest(apiVersion: String?, allOrNone: Boolean,
 refIdToRequests: LinkedHashMap<String?, RestRequest?>): CompositeRequest
```

### Java

```
public static CompositeRequest getCompositeRequest(String apiVersion,
 boolean allOrNone, LinkedHashMap<String, RestRequest> refIdToRequests)
 throws JSONException
```

## 関連トピック

- [Batch 要求および Composite 要求](#) (ページ 104) (iOS)
- [Batch 要求および Composite 要求](#) (ページ 162) (Android)
- 『[REST API 開発者ガイド](#)』の「[Composite](#)」

## Create

指定したオブジェクト種別のレコードを作成します。

## パラメーター

- API バージョン (文字列)
- オブジェクト種別 (文字列)
- (省略可能) 各項目の名前 (文字列) を、その値を含むオブジェクトに対応付けます。

## iOS

### Swift

```
RestClient.shared.requestForCreate(withObjectType:fields:)
```

### Objective-C

```
- (SFRestRequest *)requestForCreateWithObjectType:(NSString *)objectType
 fields:(nullable NSDictionary<NSString*, id> *)fields
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
fun getRequestForCreate(apiVersion: String?, objectType: String?,
 fields: Map<String?, Any?>?): RestRequest
```

## Java

```
public static RestRequest getRequestForCreate(String apiVersion,
 String objectType, Map<String, Object> fields)
```

## 関連トピック

- [『REST API 開発者ガイド』の「sObject Basic Information」](#)

## Delete

指定した種別かつ指定した ID のオブジェクトを削除します。

## パラメーター

- API バージョン (文字列)
- オブジェクト種別 (文字列)
- オブジェクト ID (文字列)

## iOS

## Swift

```
RestClient.shared.requestForDelete(withObjectType:objectId:)
```

## Objective-C

```
- (SFRestRequest *)requestForDeleteWithObjectType:(NSString *)objectType
 objectId:(NSString *)objectId
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

## Kotlin

```
fun getRequestForDelete(apiVersion: String?, objectType: String?, objectId: String?):
 RestRequest
```

## Java

```
public static RestRequest getRequestForDelete(String apiVersion, String objectType,
 String objectId)
```

## 関連トピック

- [『REST API 開発者ガイド』の「sObject Rows」](#)

## Describe

項目、URL、子リレーションなど、オブジェクトのあらゆるレベルのメタデータについて詳しく説明します。

## パラメーター

- API バージョン (文字列)
- オブジェクト種別 (文字列)

## iOS

### Swift

#### Delegate メソッド

```
RestClient.shared.requestForDescribe(withObjectType:)
```

#### Block メソッド

```
describe(_:onFailure:onSuccess:)
```

### Objective-C

#### Delegate メソッド

```
- (SFRestRequest *)
 requestForDescribeWithObjectType:(NSString *)objectType
 apiVersion:(nullable NSString *)apiVersion;
```

#### Block メソッド

```
- (SFRestRequest *) performDescribeWithObjectType:(NSString *)objectType
 failBlock:(SFRestFailBlock) failBlock

completeBlock:(SFRestDictionaryResponseBlock) completeBlock;
```

## Android

### Kotlin

```
fun getRequestForDescribe(apiVersion: String?, objectType: String?): RestRequest
```

### Java

```
public static RestRequest getRequestForDescribe(String apiVersion, String objectType)
```

## 関連トピック

- [『REST API 開発者ガイド』の「sObject Describe」](#)

## Describe Global

組織で利用できるすべてのオブジェクトとそのメタデータのリストを返します。

## パラメーター

- API バージョン (文字列)

## iOS

### Swift

```
RestClient.shared.requestForDescribeGlobal()
```

### Objective-C

```
- (SFRestRequest *)requestForDescribeGlobal
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
fun getRequestForDescribeGlobal(apiVersion: String?): RestRequest
```

### Java

```
public static RestRequest getRequestForDescribeGlobal(String apiVersion)
```

## 関連トピック

- [『REST API 開発者ガイド』の「Describe Global」](#)

## メタデータ

指定したオブジェクトの sObject Basic Information によって提供されたメタデータについて説明します。

## パラメーター

- API バージョン (文字列)
- オブジェクト種別 (文字列)

## iOS

### Swift

#### Delegate メソッド

```
RestClient.shared.requestForMetadata(withObjectType:apiVersion:)
```

## Objective-C

### Delegate メソッド

```
- (SFRestRequest *)
 requestForMetadataWithObjectType:(NSString *)objectType
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
fun getRequestForMetadata(apiVersion: String?, objectType: String?): RestRequest
```

### Java

```
public static RestRequest getRequestForMetadata(String apiVersion, String objectType)
```

## 関連トピック

- [『REST API 開発者ガイド』の「sObject Basic Information」](#)

## 通知

通知 ID を基に 1 件の通知を取得します。

## パラメーター

- API バージョン (文字列)
- 通知 ID (文字列)

## iOS

### Swift

```
RestClient.shared.request(forNotification:apiVersion:)
```

### Objective-C

```
#import <SalesforceSDKCore/SFRestAPI+Notifications.h>
...
- (SFRestRequest *)requestForNotification:(NSString *)notificationId apiVersion:(NSString *)apiVersion;
```

## Android

### Kotlin

```
fun getRequestForNotification(apiVersion: String?, notificationId: String?): RestRequest
```

## Java

```
public static RestRequest getRequestForNotification(String apiVersion, String notificationId)
```

## 関連トピック

- 『[Connect REST API 開発者ガイド](#)』の「Notification」

## Notification Update

指定された ID を持ち、「read」 (null でない場合) と「seen」 (null でない場合) 状況の通知を更新します。

## iOS

iOS で更新要求を作成するには、Swift の `UpdateNotificationsRequestBuilder` オブジェクト、または Objective-C の `SFSDKUpdateNotificationsRequestBuilder` オブジェクトを使用します。

通知の ID を `notificationId` プロパティに渡します。

## Swift

```
let builder = UpdateNotificationsRequestBuilder.init()
builder.setNotificationId("<some_id>")
builder.setBefore(Date.init())
builder.setRead(true)
builder.setSeen(true)
let request = builder.buildUpdateNotificationsRequest(SFRestDefaultAPIVersion)
```

## Objective-C

```
#import <SalesforceSDKCore/SFRestAPI+Notifications.h>
...

SFSDKUpdateNotificationsRequestBuilder *builder =
 [[SFSDKUpdateNotificationsRequestBuilder alloc] init];
[builder setNotificationId:"<some_id>"]
[builder setRead:true];
[builder setSeen:true];
[builder setBefore: [NSDate date]];
SFRestRequest *updateRequest = [builder
 buildUpdateNotificationsRequest:kSFRestDefaultAPIVersion];
```

## Android

### パラメーター

- `apiVersion` (文字列)
- `notificationId` (文字列)
- `read` (Boolean)
- `seen` (Boolean)

### Kotlin

```
fun getRequestForNotificationUpdate(apiVersion: String?, notificationId: String?,
 read: Boolean?, seen: Boolean?): RestRequest
```

### Java

```
public static RestRequest getRequestForNotificationUpdate(String apiVersion,
 String notificationId, Boolean read, Boolean seen)
```

## 関連トピック

- [『Connect REST API 開発者ガイド』の「Notification」](#)

## 通知

指定された日付の「前」または「後」について、指定された数 (最大 20) のアーカイブされた Notification Builder 通知を取得します。

## パラメーター

- API バージョン (文字列)
- バッチサイズ (整数)
- 前の日付 (日付、省略可能)
- 後の日付 (日付、省略可能)

## iOS

iOS で通知の GET 要求を作成するには、FetchNotificationsRequestBuilder オブジェクト、または Objective-C の SFSDKFetchNotificationsRequestBuilder を使用します。

### Swift

```
let builder = FetchNotificationsRequestBuilder.init()
builder.setSize(10)
builder.setBefore(Date.init())
let request = builder.buildFetchNotificationsRequest(SFRestDefaultAPIVersion)
```

### Objective-C

```
#import <SalesforceSDKCore/SFRestAPI+Notifications.h>
...

SFSDKFetchNotificationsRequestBuilder *builder =
 [[SFSDKFetchNotificationsRequestBuilder alloc] init];
[builder setBefore: [NSDate date]];
[builder setSize:10];
SFRestRequest *fetchRequest =
 [builder buildFetchNotificationsRequest:kSFRestDefaultAPIVersion];
```



## Android

### Kotlin

```
fun getRequestForNotifications(apiVersion: String?, size: Int?,
 before: Date?, after: Date?): RestRequest
```

### Java

```
public static RestRequest getRequestForNotifications(String apiVersion,
 Integer size, Date before, Date after)
```

## 関連トピック

- [『REST API 開発者ガイド』の「Query」](#)

## Notifications Status

未読数や未表示数など、現在のユーザーの通知の状況を取得します。

## パラメーター

- API バージョン (文字列)

## iOS

### Swift

デフォルトの `forNotificationsStatus` パラメーターについては、API バージョン文字列を渡します。

```
RestClient.shared.request(forNotificationsStatus:)
```

### Objective-C

```
#import <SalesforceSDKCore/SFRestAPI+Notifications.h>
...
- (SFRestRequest *)requestForNotificationsStatus:(NSString *)apiVersion;
```

## Android

### Kotlin

```
fun getRequestForNotificationsStatus(apiVersion: String?): RestRequest
```

### Java

```
public static RestRequest getRequestForNotificationsStatus(String apiVersion)
```

## 関連トピック

- [『Connect REST API 開発者ガイド状況』の「Notifications Status」](#)

## Notifications Update

指定された ID を持ち、「read」(null でない場合) と「seen」(null でない場合) 状況の通知、または指定された日付よりも前の通知を更新します。

### iOS

iOS で更新要求を作成するには、Swift の `UpdateNotificationsRequestBuilder` オブジェクト、または Objective-C の `SFSDKUpdateNotificationsRequestBuilder` オブジェクトを使用します。

影響を受ける通知の範囲を定義するには、通知 ID の配列または「前」の日付を渡します。ID 配列と「前」の日付は相互に排他的なパラメーターです。

#### Swift

```
let builder = UpdateNotificationsRequestBuilder.init()
// builder.setNotificationIds("<array_of_ids>")
// OR
// builder.setBefore(Date.init())
builder.setRead(true)
builder.setSeen(true)
let request = .
 builder.buildUpdateNotificationsRequest(SFRestDefaultAPIVersion)
```

#### Objective-C

```
#import <SalesforceSDKCore/SFRestAPI+Notifications.h>
...

SFSDKUpdateNotificationsRequestBuilder *builder =
 [[SFSDKUpdateNotificationsRequestBuilder alloc] init];
// [builder setNotificationIds:<array_of_ids>]
// OR
// [builder setBefore: [NSDate date]];
[builder setRead:true];
[builder setSeen:true];
SFRestRequest *updateRequest =
 [builder buildUpdateNotificationsRequest:kSFRestDefaultAPIVersion];
```

### Android

#### Kotlin

```
fun getRequestForNotificationUpdate(apiVersion: String?, notificationId: String?,
 read: Boolean?, seen: Boolean?): RestRequest
```

#### Java

##### パラメーター

- `apiVersion` (文字列)
  - `notificationIds` (配列)
- OR

- before (日付)
- read (Boolean)
- seen (Boolean)

```
public static RestRequest getRequestForNotificationsUpdate(String apiVersion,
 List<String> notificationIds, Date before, Boolean read, Boolean seen)
```

## 関連トピック

- [『Connect REST API 開発者ガイド』の「Notifications」](#)

## Object Layout

指定したオブジェクト種別とパラメーターのレイアウトメタデータを取得します。

### パラメーター

- API バージョン (文字列、省略可能)
- オブジェクト API 参照名 (文字列、必須)
- フォーム要素 (文字列、省略可能) — 「大」 (デフォルト)、 「中」、 「小」
- レイアウト種別 (文字列、省略可能) — 「フル」 (デフォルト)、 「コンパクト」
- モード (文字列、省略可能) — 「表示」 (デフォルト)、 「作成」、 「編集」
- レコードタイプ ID (文字列、省略可能) — 新規レコードの RecordType オブジェクトの ID。指定されていない場合は、デフォルトのレコードタイプが使用されます。

## iOS

### Swift

```
requestForLayout (withObjectAPIName:formFactor:layoutType:mode:recordTypeId:apiVersion:)
```

### Objective-C

```
- (SFRestRequest *)
requestForLayoutWithObjectAPIName:(nonnull NSString *)objectAPIName
 formFactor:(nullable NSString *)formFactor
 layoutType:(nullable NSString *)layoutType
 mode:(nullable NSString *)mode
 recordTypeId:(nullable NSString *)recordTypeId
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
fun getRequestForObjectLayout(apiVersion: String?, objectAPIName: String?,
 formFactor: String?, layoutType: String?, mode: String?,
 recordTypeId: String?): RestRequest
```

### Java

```
public static RestRequest getRequestForObjectLayout(
 String apiVersion, String objectAPIName, String formFactor,
 String layoutType, String mode, String recordTypeId)
```

## 関連トピック

- 『[User Interface API Developer Guide \(ユーザーインターフェース API 開発者ガイド\)](#)』の「[Get Record Layout Metadata \(レコードレイアウトメタデータの取得\)](#)」

## リソース

リソース名および URI を含む、指定された API バージョンで使用可能なリソースを取得します。

## パラメーター

- API バージョン (文字列、省略可能)

## iOS

### Swift

```
RestClient.shared.requestForResources()
```

### Objective-C

```
- (SFRestRequest *)requestForResources
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
fun getRequestForResources(apiVersion: String?): RestRequest
```

### Java

```
public static RestRequest getRequestForResources(String apiVersion)
```

## 関連トピック

- 『[REST API 開発者ガイド](#)』の「[Resources by Version](#)」

## Retrieve

オブジェクト ID に基づいて 1 つの sObject レコードを取得します。

項目のリストを指定すると、Mobile SDK がその項目のみを取得します。指定しない場合は、アクセス可能な標準およびカスタムのすべての項目が返されます。

## パラメーター

- API バージョン (文字列、省略可能)
- オブジェクト種別 (文字列)
- オブジェクト ID (文字列)
- 項目リスト (文字列リスト、省略可能)

## iOS

iOS の場合、fieldList パラメーターはカンマで区切られた項目名のリストまたは nil になります。

### Swift

```
RestClient.shared.requestForRetrieve(withObjectType:objectId:fieldList:apiVersion:)
```

### Objective-C

```
- (SFRestRequest *)requestForRetrieveWithObjectType:(NSString *)objectType
 objectId:(NSString *)objectId
 fieldList:(nullable NSString *)fieldList
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
fun getRequestForRetrieve(apiVersion: String?, objectType: String?,
 objectId: String?, fieldList: List<String>?): RestRequest
```

### Java

```
public static RestRequest getRequestForRetrieve(String apiVersion, String objectType,
 String objectId, List<String> fieldList) throws UnsupportedOperationException
```

## 関連トピック

- 項目データの取得に関する条件については、『[REST API 開発者ガイド](#)』の「[標準オブジェクトレコードから項目値を取得する](#)」を参照してください。

## SOSL Search

指定した SOSL 検索を実施します。

指定した SOQL クエリを実行し、要求されたレコード項目のうち、クエリに一致したものを返します。

`batchSize` パラメーターの範囲は、200～2,000(デフォルト値)です。このパラメーターは、実行時に実際にそのサイズになることは保証されていません。デフォルトでは、一度に最大 2,000 レコードが返されます。この要求でバッチサイズを指定した場合は、レコードがそのサイズ以下のバッチで返されます。バッチサイズを指定しても、返されるバッチがそのサイズになるという保証はありません。

## パラメーター

- API バージョン (文字列、省略可能)
- SOSL クエリ (文字列)

## iOS

### Swift

```
RestClient.shared.request(forSearch:apiVersion:)
```

### Objective-C

```
- (SFRestRequest *)requestForSearch:(NSString *)sosl
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
@Throws(UnsupportedEncodingException::class)
fun getRequestForSearch(apiVersion: String?, q: String?): RestRequest
```

### Java

```
public static RestRequest getRequestForSearch(String apiVersion, String q)
 throws UnsupportedOperationException
```

## 関連トピック

- [『REST API 開発者ガイド』の「Search」](#)

## Search Result Layout

1 回のクエリのオブジェクトの最大数が 100 となる検索結果レイアウトを取得します。

## パラメーター

- API バージョン (文字列、省略可能)
- オブジェクトリスト (文字列リスト)

## iOS

### Swift

オブジェクトリストについては、`forSearchResultLayout` をカンマ区切りのオブジェクト名の文字列に設定します。

```
RestClient.shared.request(forSearchResultLayout:apiVersion:)
```

### Objective-C

```
- (SFRestRequest *)
requestForSearchResultLayout:(NSString*)objectList
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
@Throws (UnsupportedEncodingException::class)
fun getRequestForSearchResultLayout (apiVersion: String?,
 objectList: List<String>): RestRequest
```

### Java

```
public static RestRequest getRequestForSearchResultLayout (String apiVersion, List<String>
 objectList) throws UnsupportedEncodingException
```

## 関連トピック

- [『REST API 開発者ガイド結果レイアウト』の「Search Result Layouts」](#)

## Search Scope and Order

現在のユーザーのデフォルトのグローバル検索範囲内にあるオブジェクトの順序付きリストを取得します。

## パラメーター

- API バージョン (文字列、省略可能)

## iOS

### Swift

```
RestClient.shared.requestForSearchScopeAndOrder (apiVersion:)
```

### Objective-C

```
- (SFRestRequest *)requestForSearchScopeAndOrder
 apiVersion:(nullable NSString *)apiVersion;
```



## Android

### Kotlin

```
fun getRequestForSearchScopeAndOrder(apiVersion: String?): RestRequest
```

### Java

```
public static RestRequest getRequestForSearchScopeAndOrder(String apiVersion)
```

## 関連トピック

- [『REST API 開発者ガイド結果レイアウト』の「Search Scope and Order」](#)

## SObject Tree

指定したオブジェクト種別のルートレコードを持つ1つ以上の sObject ツリーを作成します。

## パラメーター

- API バージョン (文字列、省略可能)
- オブジェクト種別 (文字列)
- オブジェクトツリー (sObject ツリーオブジェクトのリストまたは配列)

## iOS

### Swift

```
RestClient.shared.request(forSObjectTree:objectTrees:apiVersion:)
```

### Objective-C

```
- (SFRestRequest*) requestForSObjectTree:(NSString*) objectType
 objectTrees:(NSArray<SFObjectTree*>*) objectTrees
 apiVersion:(nullable NSString *) apiVersion;
```

## Android

### Kotlin

```
@Throws(JSONException::class)
fun getRequestForSObjectTree(apiVersion: String?, objectType: String?,
 objectTrees: List<SObjectTree>): RestRequest
```

### Java

```
public static RestRequest getRequestForSObjectTree(String apiVersion, String objectType,
 List<SObjectTree> objectTrees) throws JSONException
```

## 関連トピック

- [『REST API 開発者ガイド』の「sObject Tree」](#)

## SOQL Query

指定した SOQL クエリを実行し、要求されたレコード項目のうち、クエリに一致したものを返します。

batchSize パラメーターの範囲は、200～2,000(デフォルト値)です。このパラメーターは、実行時に実際にそのサイズになることは保証されていません。デフォルトでは、一度に最大 2,000 レコードが返されます。この要求でバッチサイズを指定した場合は、レコードがそのサイズ以下のバッチで返されます。バッチサイズを指定しても、返されるバッチがそのサイズになるという保証はありません。

## パラメーター

- API バージョン (文字列、省略可能)
- クエリ (文字列)
- バッチサイズ (整数)

## iOS

### Swift

```
RestClient.shared.request(forQuery:apiVersion:batchSize:)
```

### Objective-C

```
- (SFRestRequest *)requestForQuery:(NSString *)soql
 apiVersion:(nullable NSString *)apiVersion
 batchSize:(NSInteger)batchSize;
```

## Android

### Kotlin

```
@Throws(UnsupportedEncodingException::class)
fun getRequestForQuery(apiVersion: String?, q: String?): RestRequest
```

### Java

```
public static RestRequest
 getRequestForQuery(String apiVersion, String q)
 throws UnsupportedOperationException

public static RestRequest
 getRequestForQuery(String apiVersion, String q, int batchSize)
 throws UnsupportedOperationException
```

## 関連トピック

- [『REST API 開発者ガイド』の「Query」](#)

## SOQL Query All

指定したSOQL文字列を実行します。結果には、クエリを満たす、現在のオブジェクトおよび削除済みオブジェクトがすべて含まれます。

`batchSize` パラメーターの範囲は、200～2,000(デフォルト値)です。このパラメーターは、実行時に実際にそのサイズになることは保証されていません。デフォルトでは、一度に最大2,000レコードが返されます。この要求でバッチサイズを指定した場合は、レコードがそのサイズ以下のバッチで返されます。バッチサイズを指定しても、返されるバッチがそのサイズになるという保証はありません。

## パラメーター

- API バージョン (文字列、省略可能)
- クエリ (文字列)

## iOS

### Swift

```
RestClient.shared.request(forQueryAll:apiVersion:)
```

### Objective-C

```
- (SFRestRequest *)requestForQueryAll:(NSString *)soql
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

サポートされていません。

## 関連トピック

- [『REST API 開発者ガイド』の「QueryAll」](#)

## ユーザー情報

現在のユーザーに関連付けられている情報を返します。

## iOS

### Swift

```
RestClient.shared.requestForUserInfo()
```

### Objective-C

```
- (SFRestRequest *)requestForUserInfo;
```

## Android

### Kotlin

```
val requestForUserInfo: RestRequest
```

### Java

```
public static RestRequest getRequestForUserInfo()
```

## 関連トピック

- [Salesforce ヘルプの「ユーザー情報のクエリ」](#)

## Update

要求されたレコードの指定された項目を指定の値で更新します。また、特定の日付以降にレコードが変更されている場合には更新が行われないようにすることもできます。

## パラメーター

- API バージョン (文字列、省略可能)
- オブジェクト種別 (文字列)
- オブジェクト ID (文字列)
- 項目 (対応付け、省略可能) — 更新する項目を新しい値に対応付けます。
- 「指定した日付以降にデータが変更されていない場合」の日付 (日付、省略可能) — 指定の日付以降にレコードが変更されていない場合にのみ要求を実行します。

## iOS

### Swift

```
RestClient.shared.requestForUpdate(withObjectType:objectId:fields:)
```

```
RestClient.shared.requestForUpdate(withObjectType:objectId:fields:ifUnmodifiedSince:)
```

### Objective-C

```
- (SFRestRequest *)requestForUpdateWithObjectType:(NSString *)objectType
 objectId:(NSString *)objectId
 fields:(nullable NSDictionary<NSString*, id> *)fields
 apiVersion:(nullable NSString *)apiVersion;

- (SFRestRequest *)requestForUpdateWithObjectType:(NSString *)objectType
 objectId:(NSString *)objectId
 fields:(nullable NSDictionary<NSString*, id> *)fields
 ifUnmodifiedSinceDate:(nullable NSDate *) ifUnmodifiedSinceDate
 apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
fun getRequestForUpdate(apiVersion: String?, objectType: String?,
 objectId: String?, fields: Map<String?, Any?>?): RestRequest

fun getRequestForUpdate(apiVersion: String?, objectType: String?,
 objectId: String?, fields: Map<String?, Any?>?,
 ifUnmodifiedSinceDate: Date?): RestRequest
```

### Java

```
public static RestRequest getRequestForUpdate(String apiVersion,
 String objectType, String objectId, Map<String, Object> fields)

public static RestRequest getRequestForUpdate(String apiVersion,
 String objectType, String objectId, Map<String, Object> fields,
 Date ifUnmodifiedSinceDate)
```

## 関連トピック

- [『REST API 開発者ガイド』の「Update a Record」](#)

## Upsert

外部データのオブジェクトを更新または挿入します。

Salesforce では、外部IDが現在外部ID項目に存在するかどうかに基づいて、レコードを更新または挿入します。Salesforce で強制的に新規レコードを作成するには、外部ID項目の名前を「Id」、外部IDを null に設定します。

## パラメーター

- API バージョン (文字列、省略可能)
- オブジェクト種別 (文字列)
- 外部ID項目 (文字列)
- 外部ID (文字列、省略可能)
- 項目 (対応付け、省略可能) — 各項目名を、その値を含むオブジェクトに対応付けます。

`fields` が null の場合は、更新/挿入されたレコードが空になります。

## iOS

### Swift

```
RestClient.shared.requestForUpsert(withObjectType:externalIdField:externalId:fields:)
```

### Objective-C

```
- (SFRestRequest *)requestForUpsertWithObjectType:(NSString *)objectType
 externalIdField:(NSString *)externalIdField
```

```
externalId:(nullable NSString *)externalId
fields:(NSDictionary<NSString*, id> *)fields
apiVersion:(nullable NSString *)apiVersion;
```

## Android

### Kotlin

```
fun getRequestForUpsert(apiVersion: String?, objectType: String?,
 externalIdField: String?, externalId: String?,
 fields: Map<String?, Any?>?): RestRequest
```

### Java

```
public static RestRequest getRequestForUpsert(String apiVersion,
 String objectType, String externalIdField, String externalId,
 Map<String, Object> fields)
```

## 関連トピック

- [『REST API 開発者ガイド』の「外部 ID を使用してレコードを挿入/更新 \(Upsert\) する」](#)

## Versions

Salesforce API の現在使用可能な各バージョンのサマリー情報を取得します。

## iOS

### Swift

```
RestClient.shared.requestForVersions()
```

### Objective-C

```
- (SFRestRequest *)requestForVersions;
```

## Android

### Kotlin

```
val requestForVersions: RestRequest
```

### Java

```
public static RestRequest getRequestForVersions()
```

## 関連トピック

- [『REST API 開発者ガイド』の「Versions」](#)

## Files API リファレンス

---

Files API の使用法については、「[ファイルとネットワーキング](#)」を参照してください。API リファレンスのリンクを次に示します。

### Android

「[Package com.salesforce.androidsdk.rest.files](#) ([パッケージ com.salesforce.androidsdk.rest.files](#))」を参照してください。

### iOS

<https://forcedotcom.github.io/SalesforceMobileSDK-iOS/Documentation/SalesforceSDKCore/html/Classes/SFRestAPI.html> の「Files Methods (ファイルメソッド)」を参照してください。

### ハイブリッド

ハイブリッドファイル要求ラッパーは、[force+files.js](#) JavaScript ライブラリにあります。

## iOS アーキテクチャ

---

Mobile SDK は基本的に 1 つのライブラリで、次のモジュールが含まれています。

- [SalesforceSDKCommon](#) — 他の iOS コンポーネントで使用するユーティリティやカスタムデータ型を実装します。
- [SalesforceAnalytics](#) — Mobile SDK のログ記録および計測フレームワークを実装します。
- [SalesforceSDKCore](#) — OAuth 認証、パスコード、ネットワークを実装します。
- [SmartStore](#) — Mobile SDK の安全なオフラインストレージソリューション。
- [MobileSync](#) — Mobile SDK のオフライン同期ソリューション。

次の iOS アーキテクチャは、iOS ネイティブライブラリに依存しますが、別個の GitHub リポジトリに保持されています。

- [React Native](#) — Mobile SDK 機能に対するネイティブブリッジ。React Native アプリケーションでのみ使用します。
- [ハイブリッド](#) — Mobile SDK の機能に対する Mobile SDK Cordova プラグインを定義します。ハイブリッドアプリケーションでのみ使用します。

[forceios](#) を使用してネイティブアプリケーションを作成する場合、CocoaPods は指定したアプリケーション種別に基づいて必要なモジュールを組み込みます。SalesforceMobileSDK-iOS Git リポジトリのコピーを使用してネイティブアプリケーションを作成する場合、プロジェクトはこれらのモジュールを動的ライブラリとして使用します。

関連トピック:

[ログインと分析](#)

## サポートされる REST サービス

Mobile SDK REST API は、Salesforce Platform REST および SOAP API で提供される標準のオブジェクト操作をサポートしています。大半の操作種別は、ファクトリーメソッドまたはファクトリーオブジェクトによって、その操作専用の REST 要求オブジェクトが作成されます。Mobile SDK REST API を使用してこの要求オブジェクトを Salesforce に送信し、非同期で応答を受信します。



要求を Salesforce に送信すると、応答が非同期で各自のアプリケーションに届きます。これらの応答を処理するには、要求の送信時にコールバック代行業を指定するか、クロージャーを定義します (Swift のみ)。

## Notification

通知を取得します。

**Delegate メソッド**

**Swift**

```
RestClient.shared.request(forNotification:apiVersion:)
```

**Objective-C**

```
- (SFRestRequest *)requestForNotification:(NSString *)notificationId apiVersion:(NSString *)apiVersion;
```

**Block メソッド**

利用できません。

## Notifications Status

未読数や未表示数などのさまざまな通知の状況を取得できます。

**Delegate メソッド**

**Swift**

```
RestClient.shared.request(forNotificationsStatus:)
```

**Objective-C**

```
- (SFRestRequest *)requestForNotificationsStatus:(NSString *)apiVersion;
```

**Block メソッド**

利用できません。

## Notifications

指定された日付の「前」または「後」について、指定された数 (最大 20) のアーカイブされた Notification Builder 通知を取得します。Mobile SDK で通知の GET 要求を作成するには、Swift `FetchNotificationsRequestBuilder` オブジェクト、または Objective-C `SFSDKFetchNotificationsRequestBuilder` を使用します。

**Delegate メソッド**

**Swift**

```
let builder = FetchNotificationsRequestBuilder.init()
builder.setSize(10)
builder.setBefore(Date.init())
let request = builder.buildFetchNotificationsRequest(SFRestDefaultAPIVersion)
```

## Objective-C

```
SFSDKFetchNotificationsRequestBuilder *builder =
 [[SFSDKFetchNotificationsRequestBuilder alloc] init];
[builder setBefore: [NSDate date]];
[builder setSize:10];
SFRestRequest *fetchRequest =
 [builder buildFetchNotificationsRequest:kSFRestDefaultAPIVersion];
```

## Block メソッド

利用できません。

## Notifications Update

特定の Notification Builder の通知セットの「read」と「seen」の状況を更新します。Mobile SDK で更新要求を作成するには、Swift UpdateNotificationsRequestBuilder オブジェクト、または Objective-C SFSDKUpdateNotificationsRequestBuilder オブジェクトを使用します。

1つの通知を更新するには、notificationId プロパティを設定します。通知の範囲を更新し、notificationIds または before プロパティのいずれかを設定します。これらの notificationId、notificationIds および before プロパティは相互に排他的です。

## Delegate メソッド

### Swift

```
let builder = UpdateNotificationsRequestBuilder.init()
builder.setBefore(Date.init())
builder.setRead(true)
builder.setSeen(true)
let request = builder.buildUpdateNotificationsRequest(SFRestDefaultAPIVersion)
```

## Objective-C

```
SFSDKUpdateNotificationsRequestBuilder *builder =
 [[SFSDKUpdateNotificationsRequestBuilder alloc] init];
[builder setRead:true];
[builder setSeen:true];
[builder setBefore: [NSDate date]];
SFRestRequest *updateRequest = [builder
 buildUpdateNotificationsRequest:kSFRestDefaultAPIVersion];
```

## Block メソッド

利用できません。



例: サンプルコールは、[github.com/forcedotcom/SalesforceMobileSDK-iOS](https://github.com/forcedotcom/SalesforceMobileSDK-iOS) の

/libs/SalesforceSDKCore/SalesforceSDKCoreTests/SalesforceRestAPITests.m を参照してください。

## 関連トピック:

[サポート対象の Salesforce API](#)

## Android アーキテクチャ

---

Salesforce Mobile SDK for Android は、Mobile SDK のコアコンポーネント、SmartStore、および Mobile Sync が含まれているライブラリプロジェクトです。このプロジェクトには、Android用のハイブリッドおよびReactNative ネイティブブリッジも含まれています。

Android アプリケーションは、アプリケーションプロジェクトから SalesforceSDK プロジェクトを参照します。「[Android アーキテクチャ](#)」を参照してください。

## Android パッケージおよびクラス

Android Mobile SDK の Java ソースファイルは、`libs/SalesforceSDK/src` にあります。

## 最上位パッケージのカタログ

パッケージとクラスの説明は、『[Salesforce Mobile SDK Android Reference \(Salesforce Mobile SDK Android リファレンス\)](#)』を参照してください。

## Android リソース

Mobile SDK プロジェクトの場合、リソースは `/res` フォルダーの下にあります。SalesforceMobileSDK-Android GitHub リポジトリの場合、リソースは `libs/SalesforceSDK/res` にあります。

## Mobile SDK 11.1 でサポートされるバージョンのツールおよびコンポーネント

---

### すべてのプラットフォーム

| ツールまたはコンポーネント | サポートされるバージョン                | インストールの詳細                                              |
|---------------|-----------------------------|--------------------------------------------------------|
| Node.js       | 最新                          | <a href="https://nodejs.org">nodejs.org</a> からインストールする |
| npm           | 3.10                        | Node.js によってインストールされる                                  |
| shelljs       | 0.8.5                       | Node.js によってインストールされる                                  |
| SQLite        | 3.41.2                      | Mobile SDK によってインストールされる                               |
| SQLCipher     | Android は 4.5.4、iOS は 4.5.4 | Mobile SDK によってインストールされる                               |
| 全文検索 (FTS)    | FTS5                        | Mobile SDK によってインストールされる                               |

## iOS

| ツールまたはコンポーネント | サポートされるバージョン     | インストールの詳細                                                           |
|---------------|------------------|---------------------------------------------------------------------|
| Xcode         | 14               | Mac App Store からインストールする                                            |
| iOS リリース対象    | 15               | Xcode によってインストールされる                                                 |
| iOS ベース SDK   | 16               | Xcode によってインストールされる                                                 |
| CocoaPods     | 1.8 から最大値 (指定なし) | <a href="https://cocoapods.org">cocoapods.org</a> からインストールする        |
| forceios      | 11.1             | コマンドラインまたはターミナルプロンプトで、 <code>「npm install -g forceios」</code> と入力する |

## Android

| ツールまたはコンポーネント                           | サポートされるバージョン                                                  | インストールの詳細                                                                                    |
|-----------------------------------------|---------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| Java JDK                                | 11.0.11+9                                                     | <a href="https://oracle.com">oracle.com</a> からインストールする                                       |
| Android Studio                          | 最新                                                            | <a href="https://developer.android.com/studio/">developer.android.com/studio/</a> からインストールする |
| Gradle                                  | 7.2.1                                                         | Android Studio によってインストールされる                                                                 |
| Android SDK minApi                      | Android Nougat (API 24)                                       | Android Studio の Android SDK マネージャーを使用してインストールする                                             |
| Android SDK targetApi                   | Android 13 (API 33)                                           | Android Studio の Android SDK マネージャーを使用してインストールする                                             |
| ハイブリッドアプリケーションのデフォルトの Android SDK バージョン | 対象バージョンは Android 13 (API 33)、最小バージョンは Android Nougat (API 24) | Android Studio の Android SDK マネージャーを使用してインストールする                                             |
| OkHttp                                  | 3.12.1                                                        | Mobile SDK によってインストールされる                                                                     |
| forcedroid                              | 11.1                                                          | コマンドラインまたはターミナルプロンプトで、 <code>「npm install -g forcedroid」</code> と入力する                        |

## ハイブリッド

| ツールまたはコンポーネント                           | サポートされるバージョン                                                  | インストールの詳細                                                              |
|-----------------------------------------|---------------------------------------------------------------|------------------------------------------------------------------------|
| Cordova                                 | 12.0.1 (Android)、7.0.1 (iOS)                                  | <a href="https://cordova.apache.org">cordova.apache.org</a> からインストールする |
| Cordova コマンドライン                         | 12.0.0                                                        | コマンドラインまたはターミナルプロンプトで、 <code>npm install -g cordova</code> と入力する       |
| ハイブリッドアプリケーションのデフォルトの Android SDK バージョン | 対象バージョンは Android 13 (API 33)、最小バージョンは Android Nougat (API 24) | Android Studio の Android SDK マネージャーを使用してインストールする                       |

## React Native

| ツールまたはコンポーネント | サポートされるバージョン | インストールの詳細                                                           |
|---------------|--------------|---------------------------------------------------------------------|
| React Native  | 0.70.14      | Mobile SDK によってインストールされる                                            |
| React         | 18.1.0       | Mobile SDK によってインストールされる                                            |
| forcereact    | 11.1         | コマンドラインまたはターミナルプロンプトで、 <code>npm install -g forcereact</code> と入力する |

## 第 25 章      iOS で現在非推奨のもの

iOS で現在非推奨になっている Mobile SDK オブジェクトおよびアイテムを次に示します。これらは、ソースファイル内でアノテーションが付加されています。この情報を参照し、これらのアイテムが、示されているリリースで削除されることに備えてください。

### SFApplication

---

```
@interface SFApplication : UIApplication
```

- 12.0 で削除されるため、11.1 で非推奨になっています。

### SFInactivityTimerCenter

---

```
@interface SFInactivityTimerCenter : NSObject
```

- 12.0 で削除されるため、11.1 で非推奨になっています。

Mobile SDK for iOS のオブジェクトおよびアイテムのうち、Mobile SDK 10.0 で削除されたものを次に示します。

## SFCryptChunks

---

```
@interface SFCryptChunks : NSObject
```

## SFDecryptStream

---

```
@interface SFDecryptStream : NSInputStream <SFCryptChunksDelegate>
```

- 代わりに `SFSDKDecryptStream` を使用します。このクラスは、アップグレードの手順でのみ使用します。

## SFEncryptionKey

---

```
@interface SFEncryptionKey : NSObject <NSCoding, NSCopying>
```

## SFKeyStoreManager

---

```
@interface SFKeyStoreManager : NSObject
```

## SFSecureEncryptionKey

---

```
@interface SFSecureEncryptionKey : SFEncryptionKey
```

## SFSoupSpec

---

```
@interface SFSoupSpec : NSObject
```

- 外部ストレージとスープ指定は 11.0 で削除されました。



## SFAlterSoupLongOperation

```
- (id) initWithStore: (SFSmartStore*) store
 soupName: (NSString*) soupName
 newSoupSpec: (nullable SFSoupSpec*) newSoupSpec
 newIndexSpecs: (NSArray*) newIndexSpecs
 reIndexData: (BOOL) reIndexData;
```

- 外部ストレージとスープ指定は 11.0 で削除されました。代わりに、他のコンストラクターを使用します。

## SFSmartStore

```
- (nullable SFSoupSpec*) attributesForSoup: (NSString*) soupName
NS_SWIFT_NAME(specification(forSoupNamed:));
```

- 外部ストレージとスープ指定は 11.0 で削除されました。

```
- (BOOL) registerSoupWithSpec: (SFSoupSpec*) soupSpec
 withIndexSpecs: (NSArray<SFSoupIndex*>*) indexSpecs
 error: (NSError**) error
NS_SWIFT_NAME(registerSoup(withSpecification:withIndices:));
```

- 外部ストレージとスープ指定は 11.0 で削除されました。代わりに `soupName` を指定した `registerSoup` を使用します。

```
- (unsigned long
long) getExternalFileStorageSizeForSoup: (NSString*) soupName
NS_SWIFT_NAME(externalFileStorageSize(forSoupNamed:));
```

- 外部ストレージとスープ指定は 11.0 で削除されました。

```
- (NSUInteger) getExternalFilesCountForSoup: (NSString*) soupName
NS_SWIFT_NAME(externalFilesCount(forSoupNamed:));
```

- 外部ストレージとスープ指定は 11.0 で削除されました。

```
- (BOOL) alterSoup: (NSString*) soupName
 withSoupSpec: (SFSoupSpec*) soupSpec
 withIndexSpecs: (NSArray<SFSoupIndex*>*) indexSpecs
 reIndexData: (BOOL) reIndexData
NS_SWIFT_NAME(alterSoup(named:soupSpec:indexSpecs:reIndexData:));
```

- 外部ストレージとスープ指定は 11.0 で削除されました。代わりに `alterSoup` メソッドを使用します。

Android で現在非推奨になっている Mobile SDK オブジェクトおよびアイテムを次に示します。これらは、ソースファイル内でアノテーションが付加されています。この情報を参照し、これらのアイテムが、示されているリリースで削除されることに備えてください。

Mobile SDK for Android では、11.1 リリースで非推奨の API のタグは付けられていません。ただし、Mobile SDK 12.0 で計画されている次の変更に備えてください。

### Gradle 8 へのアップグレード

---

Mobile SDK 12.0 で Gradle 8 にアップグレードするには、ビルドファイルを Gradle 8 にアップグレードし、JDK 17 を使用する必要があります。

### Firebase Cloud Messaging 20.1.0 以降へのアップグレード

---

Mobile SDK 12.0 は、FCM を 20.1.0 以降の最新バージョンにアップグレードします。アプリケーションをアップグレードする手順は、次のとおりです。

1. `androidPushNotificationClientId` bootconfig エントリを削除します。
2. `google-services.json` をプロジェクトのルートに追加します。

Mobile SDK for Android のオブジェクトおよびアイテムのうち、Mobile SDK 10.0 で削除されたものを次に示します。

Mobile SDK 10.0 for Android で、次の API が削除されました。

## SoupSpec

```
public class SoupSpec
```

- 外部ストレージとスープ指定は 11.0 で削除されました。

## AlterSoupLongOperation

```
public AlterSoupLongOperation(SmartStore store, String soupName,

 SoupSpec newSoupSpec, IndexSpec[] newIndexSpecs, boolean
 reIndexData)
 throws JSONException
```

- 外部ストレージとスープ指定は 11.0 で削除されました。代わりに、他のコンストラクターを使用します。

## SmartStore

```
public void registerSoupWithSpec(final SoupSpec soupSpec,
 final IndexSpec[] indexSpecs)
```

- 外部ストレージとスープ指定は 11.0 で削除されました。代わりに `registerSoup(String soupName, IndexSpec[] indexSpecs)` を使用します。

```
public void alterSoup(String soupName, SoupSpec soupSpec,
 IndexSpec[] indexSpecs, boolean reIndexData)
 throws JSONException
```

- 外部ストレージとスープ指定は 11.0 で削除されました。代わりに `alterSoup(String soupName, IndexSpec[] indexSpecs)` を使用します。

```
public SoupSpec getSoupSpec(String soupName)
```

- 外部ストレージとスープ指定は 11.0 で削除されました

```
public boolean usesExternalStorage(String soupName)
```

- 外部ストレージとスープ指定は 11.0 で削除されました

## 第 29 章 商標の帰属

App Store、iOS、iTunes、MacBook、Objective-C、OS X、Swift、TV、Watch、および Xcode は、Apple Inc. の商標です。

Google、Google+、Chrome、Play、および Android は、Alphabet Inc. の商標です。

Microsoft、Windows、Visual Studio、Access、Azure は、Microsoft, Inc. の商標です。

Amazon は、Amazon.com, Inc. の商標です。

Facebook は、Facebook, Inc. の商標です。

PayPal は、PayPal Holdings, Inc. の商標です。

LinkedIn は、LinkedIn, Inc. の商標です。

Twitter は、Twitter, Inc. の商標です。

Apple は 2018 年に UIWebView を廃止し、WKWebView を採用しました。先日 App Store は、まだ UIWebView を使用しているアプリケーションを正式に削除する計画を発表しました。これらの要件を満たすために、Mobile SDK では UIWebView への参照がすべて削除されています。

Mobile SDK では、3 段階のリリースサイクルで UIWebView の置き換えが実施されます。ただし、Mobile SDK 8.0 の開発当初、Mobile SDK ハイブリッドアプリケーションの基盤技術である Cordova では、そのコードベースで UIWebView が参照されていました。Mobile SDK 8.0 では、この未使用の参照を削除する手段がないため、新しい App Store の要件を満たすことができませんでした。

Mobile SDK 8.1 では新しい Cordova 5.1.1 リリースが採用されており、クライアントではその UIWebView コードを「コンパイルから除外」できます。このため、Mobile SDK 8.1 は App Store への提出に対応しています。

## よくある質問 (FAQ)

---

App Store の期限はいつですか？

- **2020 年 4 月:** App Store は、UIWebView への参照を含んだ新規 iOS アプリケーションの受け入れを終了。
- **2020 年 12 月:** App Store は、UIWebView への参照を含んだ iOS アプリケーションの更新の受け入れを終了。

どのようなアプリケーションが影響を受けますか？

この変更は、iOS の Mobile SDK ハイブリッドアプリケーションに影響します。Android の Mobile SDK ハイブリッドアプリケーション、Mobile SDK ネイティブアプリケーション、Mobile SDK React Native アプリケーションには影響しません。

どのようなことを行う必要がありますか？

- 新規ハイブリッドアプリケーションを App Store に提出する場合は、2020 年 4 月までに Mobile SDK 8.1 にアップグレードしてください。
- 更新されたハイブリッドアプリケーションを App Store に提出する場合は、2020 年 12 月までに Mobile SDK 8.1 (以降) にアップグレードしてください。
- すべてのハイブリッドアプリケーションで考慮すること:
  - カスタムコード内の UIWebView 参照をすべて WKWebView に更新してください。
  - 8.1 への更新中に、他の API が廃止されたり、削除されたり、置き換えられた場合は、必要に応じてコードを変更してください。

## iOSハイブリッドアプリケーションからのUIWebViewの 削除

### UIWebView とは何ですか?

UIWebView は、Apple の UIKit フレームワークで廃止された iOS ユーザーインターフェースコントロールです。HTML ファイルや Web コンテンツをアプリケーションビューに読み込み、ブラウザウィンドウでの表示用にレンダリングします。

[developer.apple.com/documentation/uikit/uiwebview](https://developer.apple.com/documentation/uikit/uiwebview) を参照してください。

## 関連トピック

---

[Apple の開発者向けニュースリリース \(英語\)](#)