

# ***IBM Worklight Foundation V6.2.0*** **入門**

## ネイティブ iOS アプリケーションでのプッシュ通知



## 商標

- IBM、IBM ロゴ、ibm.com および Worklight は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、<http://www.ibm.com/legal/copytrade.shtml> をご覧ください。
- Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。
- この資料は、事前に IBM の書面による許可を得ずにその一部または全部を複製することは禁じられています。

## IBM® について

- <http://www.ibm.com/ibm/us/en/> を参照してください。

# アジェンダ

- **プッシュ通知とは**
- プッシュ通知用の Worklight ネイティブ API の作成
- iOS ネイティブ・アプリケーションの作成および構成
- WLClient および WLPush の初期化
- サブスクリプション管理
- 通知 API
- タグ・ベース通知とブロードキャスト通知

# プッシュ通知とは

- プッシュ通知は、サーバーからプッシュされるメッセージを受信する、モバイル・デバイスの機能です。
- 通知は、アプリケーションが実行中であるかどうかにかかわらず受信されます。
- 通知には複数の形式を使用できます。
  - アラート: ポップアップ・テキスト・メッセージ
  - バッジ: アプリケーション・アイコンの横に表示される小さなバッジ・マーク
  - 音声アラート



# アジェンダ

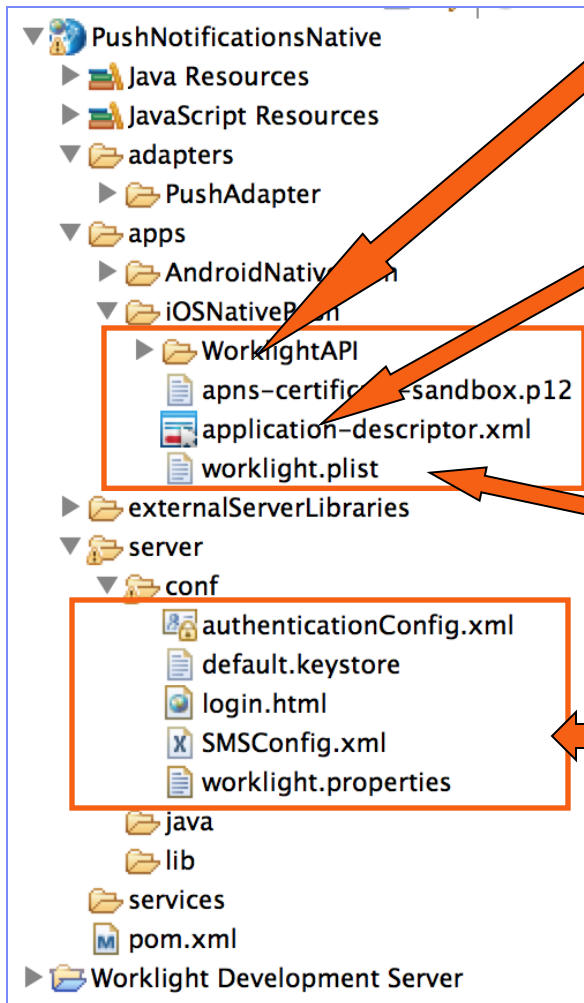
- プッシュ通知とは
- プッシュ通知用の Worklight ネイティブ API の作成
- iOS ネイティブ・アプリケーションの作成および構成
- WLClient および WLPush の初期化
- サブスクリプション管理
- 通知 API
- タグ・ベース通知とブロードキャスト通知

## プッシュ通知用の Worklight ネイティブ API の作成

- IBM Worklight Foundation ® では、ネイティブ iOS アプリケーションが Worklight ネイティブ API ライブラリーを使用することによって Worklight Server と通信できます。
- ネイティブ iOS アプリケーションでこの機能を使用できるようにするには、Worklight Server 側でこのアプリケーションを検出する必要があります。
- ネイティブ API フォルダーは、Worklight プロジェクトのアプリケーション・フォルダーにあります。
- ネイティブ API フォルダーには、ネイティブ API ライブラリーと構成ファイルが含まれており、これらをネイティブ iOS プロジェクトにコピーする必要があります。
- ネイティブ・アプリケーションには、application-descriptor.xml ファイルが含まれており、このファイルでアプリケーション・メタデータを構成できます。ネイティブ・アプリケーションはサーバーにデプロイされます。
- このモジュールでは、ネイティブ iOS アプリケーションにおける Worklight ネイティブ API の生成方法およびそのコンポーネントの使用方法について学習します。

## Worklight ネイティブ API の作成 (1/3)

- Worklight ネイティブ API には複数のコンポーネントが含まれています。



**WorklightAPI** フォルダーはライブラリーであり、ネイティブ iOS プロジェクトにコピーする必要があります。

**application-descriptor.xml** ファイルは、アプリケーション・メタデータを定義し、Worklight Server で実施されるセキュリティー設定を構成するために使用します。

**worklight.plist** ファイルには、ネイティブ iOS アプリケーションで使用される接続設定が含まれます。このファイルをネイティブ iOS プロジェクトにコピーします。

Worklight プロジェクトの場合と同様に、**server¥conf** フォルダー内にあるファイルを変更することによって、サーバー構成を作成します。

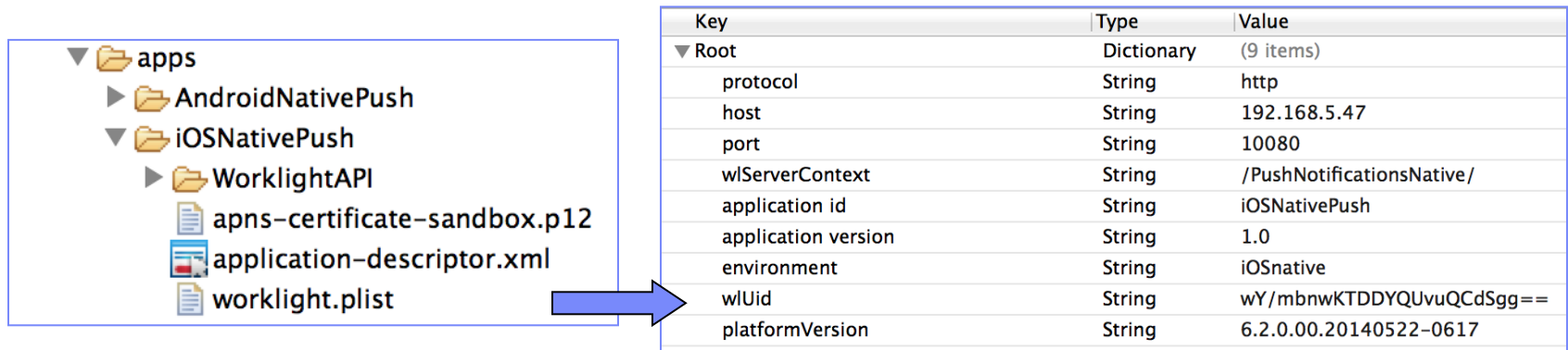
## Worklight ネイティブ API の作成 (2/3)

1. Worklight Studio で Worklight プロジェクトを作成し、Worklight ネイティブ API を追加します。
2. 「新規 Worklight ネイティブ API (New Worklight Native API)」ダイアログでアプリケーション名を入力し、「環境 (Environment)」フィールドで「iOS」を選択します。
3. Apple Push Notification Service (APNS) p12 キー (**apns-certificate-sandbox.p12** または **apns-certificate-production.p12** のいずれか) をアプリケーションのルート・フォルダーに追加します。
4. Worklight ネイティブ API フォルダーを右クリックし、「実行 (Run As)」> 「ネイティブ API のデプロイ (Deploy Native API)」をクリックします。



## Worklight ネイティブ API の作成 (3/3)

- サーバー構成を保持する worklight.plist ファイルを編集します。



The image shows a file explorer on the left with a tree view of project files. Under the 'iOSNativePush' folder, there is a 'WorklightAPI' folder containing 'apns-certificate-sandbox.p12', 'application-descriptor.xml', and 'worklight.plist'. A blue arrow points from 'worklight.plist' to a table on the right that displays the contents of the plist file.

Key	Type	Value
▼ Root	Dictionary	(9 items)
protocol	String	http
host	String	192.168.5.47
port	String	10080
wlServerContext	String	/PushNotificationsNative/
application id	String	iOSNativePush
application version	String	1.0
environment	String	iOSnative
wlUid	String	wY/mbnwKTDDYQUvuQCdSgg==
platformVersion	String	6.2.0.00.20140522-0617

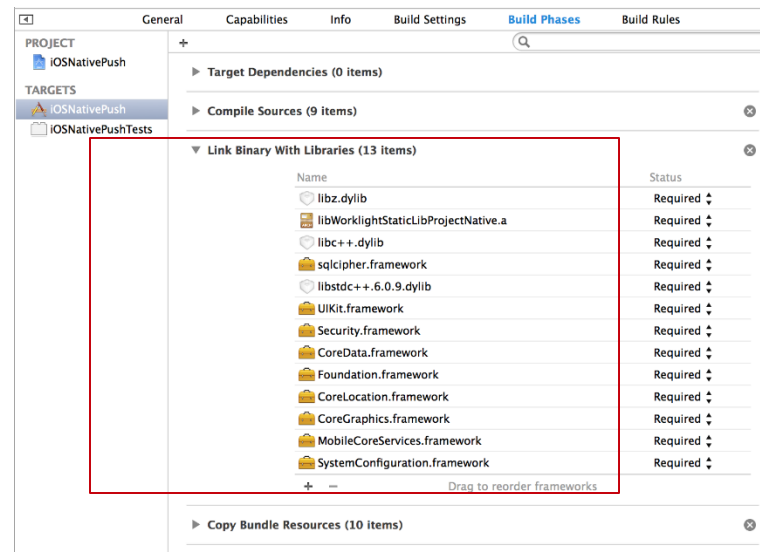
- **protocol** – Worklight Server に対する通信プロトコル (*http* または *https* のいずれかを指定できます)。
- **host** – Worklight Server のホスト名または IP アドレス。
- **port** – Worklight Server のポート。
- **wlServerContext** – Worklight Server 上のアプリケーションのコンテキスト・ルート・パス。
- **application id** – application-descriptor.xml ファイルで定義されているアプリケーション ID。
- **application version** – アプリケーションのバージョン番号。
- **environment** – ネイティブ・アプリケーションのターゲット環境 (Android または iOS)。

# アジェンダ

- プッシュ通知とは
- プッシュ通知用の Worklight ネイティブ API の作成
- iOS ネイティブ・アプリケーションの作成および構成
- WLClient および WLPush の初期化
- サブスクリプション管理
- 通知 API
- タグ・ベース通知とブロードキャスト通知

# iOS ネイティブ・アプリケーションの作成および構成 (1/2)

- Xcode プロジェクトを作成するか、または既存のプロジェクトを使用します。
- WorklightAPI フォルダおよび worklight.plist ファイルを、Eclipse Worklight ネイティブ API からネイティブ・プロジェクトのルートにコピーします。
- ネイティブ iOS アプリケーション内で以下のライブラリーをリンクします。  
**SystemConfiguration.framework、MobileCoreServices.framework、CoreData.framework、Security.framework、libz.dylib、sqlcipher.framework、libc++.dylib、libstdc++.6.dylib、および CoreLocation.framework**



## iOS ネイティブ・アプリケーションの作成および構成 (2/2)

- 「ビルド設定 (Build Settings)」で、以下のようにします。
  - 項目 `$(SRCROOT)/WorklightAPI/include` for `HEADER_SEARCH_PATH` を追加します。
  - 「その他のリンカー・フラグ (Other Linker Flags)」 フィールドで、値 `-ObjC` を入力します。
  - 「デプロイメント (Deployment)」 セクションで、「iOS デプロイメント・ターゲット (iOS Deployment Target)」 フィールドに 5.0 以上の値を選択します。

# アジェンダ

- プッシュ通知とは
- プッシュ通知用の Worklight ネイティブ API の作成
- iOS ネイティブ・アプリケーションの作成および構成
- WLClient および WLPush の初期化
- サブスクリプション管理
- 通知 API
- タグ・ベース通知とブロードキャスト通知

## WLClient および WLPush の初期化 (1/4)

- アプリケーション内の任意の場所で **[WLClient sharedInstance]** を使用して、**WLClient** 機能にアクセスします。
- **wlConnectWithDelegate** メソッドを使用して、サーバーへの接続を初期化します。
- ほとんどのアクションでは、次の例の **MyConnectListener** インスタンスのようなデリゲート・オブジェクトを指定する必要があります。

```
MyConnectListener *connectListener = [[MyConnectListener alloc] initWithController:self];  
[[WLClient sharedInstance] wlConnectWithDelegate:connectListener];  
[connectListener release];
```

- 以降のスライドでは、この作成方法について学習します。
- 必ず **WLClient.h** と **WLDelegate.h** をヘッダー・ファイルにインポートしてください。

## WLClient および WLPush の初期化 (2/4)

- 前のスライドで説明したように、Worklight Server を呼び出すメソッドに接続デリゲート (リスナー) を指定する必要があります。
- **wlConnectWithDelegate** メソッドで使用するデリゲートを作成し、Worklight Server からの応答を受け取ります。クラスに **MyConnectListener** という名前を付けます。
- ヘッダー・ファイルに、**WLDelegate** プロトコルを実装することを指定する必要があります。

```
#import <Foundation/Foundation.h>
#import "WLClient.h"
#import "WLDelegate.h"
#import "ViewController.h"

@interface MyConnectListener : NSObject <WLDelegate> {
    @private
    ViewController *vc;
}

- (id)initWithController: (ViewController *)mainView;
@end
```

- **WLDelegate** プロトコルにより、クラスに以下のメソッドを実装することを指定します。
  - **onSuccess (WLResponse \*)response**
  - **onFailure (WLFailResponse \*)response**

## WLClient および WLPush の初期化 (3/4)

- `wlConnectWithDelegate` が終了すると、指定された `MyConnectListener` インスタンスの `onSuccess` メソッドまたは `onFailure` メソッドのいずれかが呼び出されます。
- どちらの場合も、応答オブジェクトが引数として送信されます。
- このオブジェクトを使用して、サーバーから取得されたデータを操作します。

```
-(void)onSuccess:(WLResponse *)response{
    NSLog(@"\nConnection Success: %@", response);
    NSString *resultText = @"Connection success. ";

    if ([response responseText] != nil){
        resultText = [resultText stringByAppendingString:[response responseText]];
    }
    [vc updateView:resultText];
}

-(void)onFailure:(WLFailResponse *)response{
    NSString *resultText = @"Connection failure. ";

    if ([response responseText] != nil){
        resultText = [resultText stringByAppendingString:[response responseText]];
    }
    [vc updateView:resultText];
}
```



## WLClient および WLPush の初期化 (4/4)

- アプリケーション内の任意の場所で **[WLPush sharedInstance]** を使用して、**WLPush** 機能にアクセスします。
- **onReadyToSubscribeListener** のインスタンスを作成します。

```
ReadyToSubscribeListener *readyToSubscribeListener = [[ReadyToSubscribeListener alloc]
                                                       initWithController:self];
readyToSubscribeListener.alias = self.alias;
readyToSubscribeListener.adapterName = self.adapterName;
readyToSubscribeListener.eventSourceName = self.eventSourceName;
```

- **WLPush** に、**onReadyToSubscribeListener** を設定します。

```
[[WLPush sharedInstance] setOnReadyToSubscribeListener:readyToSubscribeListener];
```

- **WLPush** にトークンを渡します。

```
[[WLPush sharedInstance] setTokenFromClient:self.myToken];
```

# アジェンダ

- プッシュ通知とは
- プッシュ通知用の Worklight ネイティブ API の作成
- iOS ネイティブ・アプリケーションの作成および構成
- WLClient および WLPush の初期化
- サブスクリプション管理
- 通知 API
- タグ・ベース通知とブロードキャスト通知

# サブスクリプション管理 – ユーザー・サブスクリプション

## ■ ユーザー・サブスクリプション

- ユーザー ID、デバイス ID、およびイベント・ソース ID を含むエンティティ。ユーザー・サブスクリプションは、特定のイベント・ソースから通知を受信するユーザーのインテントを表します。

## ■ 作成

- イベント・ソースのユーザー・サブスクリプションは、ユーザーが任意のデバイスから初めてそのイベント・ソースにサブスクライブしたときに作成されます。

## ■ 削除

- ユーザー・サブスクリプションは、ユーザーがそのイベント・ソースからアンサブスクライブしたときに、所有するすべてのデバイスから削除されます。

## ■ 通知

- ユーザー・サブスクリプションが存在している間、Worklight Server は、サブスクライブ済みユーザーのプッシュ通知を生成することができます。これらの通知は、アダプター・コードによって、ユーザーがサブスクライブしたデバイスのすべてまたは一部に送信可能です。

## サブスクリプション管理 – デバイス・サブスクリプション

- デバイス・サブスクリプションは、ユーザー・サブスクリプションに属し、特定のユーザーおよびイベント・ソースの範囲内に存在します。1つのユーザー・サブスクリプションで複数のデバイス・サブスクリプションを持つことができます。
- デバイス・サブスクリプションは、デバイス上のアプリケーションが **[[WIPush sharedInstance]subscribe]** メソッドを呼び出したときに作成されます。
- デバイス・サブスクリプションは、**[[WIPush sharedInstance]unsubscribe]** を呼び出すアプリケーションによって削除されるか、またはデバイスが永続的にアクセスできないことをプッシュ・メディエーターが Worklight Server に通知した場合に削除されます。

# アジェンダ

- プッシュ通知とは
- プッシュ通知用の Worklight ネイティブ API の作成
- iOS ネイティブ・アプリケーションの作成および構成
- WLClient および WLPush の初期化
- サブスクリプション管理
- 通知 API
- タグ・ベース通知とブロードキャスト通知

通知 API の実装は、以下のメイン・ステップで構成されています。

- サーバー・サイド:
  - イベント・ソースの作成。
  - 通知の送信。
  
- クライアント・サイド:
  - トークンの送信および WLPush クラスの初期化。
  - イベント・ソースの登録。
  - イベント・ソースへのサブスクライブ/アンサブスクライブ。

## 通知 API: サーバー・サイド (1/9)

- イベント・ソースの作成。
  - アダプター JavaScript™ コードで通知イベント・ソースをグローバル・レベル (JavaScript 関数外) で宣言します。

### 通知はバックエンドによってプッシュされる

```
WL.Server.createEventSource({  
  name: 'PushEventSource',  
  onDeviceSubscribe: 'deviceSubscribeFunc',  
  onDeviceUnsubscribe: 'deviceUnsubscribeFunc',  
  securityTest: 'PushApplication-strong-mobile-se  
});
```

### 通知はバックエンドからポーリングされる

```
WL.Server.createEventSource({  
  name: 'PushEventSource',  
  onDeviceSubscribe: 'deviceSubscribeFunc',  
  onDeviceUnsubscribe: 'deviceUnsubscribeFunc',  
  securityTest: 'PushApplication-strong-mobile-securityTest',  
  poll: {  
    interval: 3,  
    onPoll: getNotificationsFromBackend  
  }  
});
```

- **name** – イベント・ソースの参照に使用する名前。
- **onDeviceSubscribe** – ユーザー・サブスクリプションの要求の受信時に呼び出されるアダプター関数。
- **onDeviceUnsubscribe** – ユーザー・アンサブスクリプションの要求の受信時に呼び出されるアダプター関数。
- **securityTest** – イベント・ソースを保護するために使用される、authenticationConfig.xml ファイルからのセキュリティー・テスト。

## 通知 API: サーバー・サイド (2/9)

- イベント・ソースの作成 (続き)。
  - アダプター JavaScript コードで通知イベント・ソースをグローバル・レベル (JavaScript 関数外) で宣言します。

### 通知はバックエンドによってプッシュされる

```
WL.Server.createEventSource({
  name: 'PushEventSource',
  onDeviceSubscribe: 'deviceSubscribeFunc',
  onDeviceUnsubscribe: 'deviceUnsubscribeFunc',
  securityTest: 'PushApplication-strong-mobile-se
});
```

### 通知はバックエンドからポーリングされる

```
WL.Server.createEventSource({
  name: 'PushEventSource',
  onDeviceSubscribe: 'deviceSubscribeFunc',
  onDeviceUnsubscribe: 'deviceUnsubscribeFunc',
  securityTest: 'PushApplication-strong-mobile-securityTest',
  poll: {
    interval: 3,
    onPoll: getNotificationsFromBackend
  }
});
```

- **poll** – 通知を取得するためのメソッド。以下のパラメーターが必須です。
  - **interval** – ポーリング間隔 (秒)。
  - **onPoll** – ポーリングの実装、つまり指定された間隔で呼び出されるアダプター関数。



## 通知 API: サーバー・サイド (3/9)

- 通知の送信。

```
function submitNotification(userId, notificationText){
  var userSubscription =
    WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);

  if (userSubscription==null){
    return { result: "No subscription found for user :: " +
  }

  var deviceSubscriptions =
    userSubscription.getDeviceSubscriptions();

  WL.Logger.debug("submitNotification >> userId :: " + userId);

  WL.Server.notifyAllDevices(userSubscription, {
    badge: 1,
    sound: "sound.mp3",
    activateButtonLabel: "ClickMe",
    alert: notificationText,
    payload: {
      foo : 'bar'
    }
  });

  return { result: "Notification sent to user :: " + userId };
}
```

前述したように、通知はバックエンドからポーリングすることもバックエンドによってプッシュすることもできます。この例では、**submitNotification()** アダプター関数が、通知を送信する外部 API としてバックエンド・システムによって呼び出されます。

## 通知 API: サーバー・サイド (4/9)

- 通知の送信 (続き)。
  - 通知データを取得します。

```
function submitNotification(userId, notificationText) {  
  var userSubscription =  
    WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);  
  
  if (userSubscription==null){  
    return { result: "No subscription found for user :: " + userId };  
  }  
  
  var deviceSubscriptions =  
    userSubscription.getDeviceSubscriptions();  
  
  WL.Logger.debug("submitNotification >> userId :: " + userId + ",  
  
  WL.Server.notifyAllDevices(userSubscription, {  
    badge: 1,  
    sound: "sound.mp3",  
    activateButtonLabel: "ClickMe",  
    alert: notificationText,  
    payload: {  
      foo : 'bar'  
    }  
  });  
  
  return { result: "Notification sent to user :: " + userId };  
}
```

**submitNotification** 関数は、通知の送信先の **userId** および **notificationText** を受け取ります。これらの引数はバックエンドによって指定され、これにより、この関数が呼び出されます。

## 通知 API: サーバー・サイド (5/9)

- 通知の送信 (続き)。
  - アクティブ・ユーザーを検索し、それを使用してユーザー・サブスクリプション・データを取得します。

```
function submitNotification(userId, notificationText){  
    var userSubscription =  
        WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);  
  
    if (userSubscription==null){  
        return { result: "No subscription found for user :: " + userId };  
    }  
  
    var deviceSubscriptions =  
        userSubscription.getDeviceSubscriptions();  
  
    WL.Logger.debug("submitNotification >> userId: " + userId);  
  
    WL.Server.notifyAllDevices(userSubscription,  
        badge: 1,  
        sound: "sound.mp3",  
        activateButtonLabel: "ClickMe",  
        alert: notificationText,  
        payload: {  
            foo : 'bar'  
        }  
    );  
  
    return { result: "Notification sent to user :: " + userId };  
}
```

ユーザー・サブスクリプション・オブジェクトにはすべてのユーザーのサブスクリプションに関する情報が含まれています。各ユーザー・サブスクリプションは複数のデバイス・サブスクリプションを持つことができます。オブジェクト構造は次のとおりです。

```
{  
    userId: 'bjones',  
    state: {  
        customField: 3  
    },  
    getDeviceSubscriptions: function(){}  
};
```

## 通知 API: サーバー・サイド (6/9)

- 通知の送信 (続き)。
  - ユーザー・サブスクリプション・データを取得します。

```
function submitNotification(userId, notificationText){
  var userSubscription =
    WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);

  if (userSubscription==null){
    return { result: "No subscription found for user :: " + userId };
  }

  var deviceSubscriptions =
    userSubscription.getDeviceSubscriptions();

  WL.Logger.debug("submitNotification >> userId :: " + userId);

  WL.Server.notifyAllDevices(userSubscription, {
    badge: 1,
    sound: "sound.mp3",
    activateButtonLabel: "ClickMe",
    alert: notificationText,
    payload: {
      foo : 'bar'
    }
  });

  return { result: "Notification sent to user :: " + userId };
}
```

指定されたイベント・ソースに対するサブスクリプションをユーザーが持っていない場合、ヌル・オブジェクトが返されます。

## 通知 API: サーバー・サイド (7/9)

- 通知の送信 (続き)。
  - ユーザー・サブスクリプション・データを取得します。

```
function submitNotification(userId, notificationText){
    var userSubscription =
        WL.Server.getUserNotificationSubscription('PushAdapter.PushEvent');

    if (userSubscription==null){
        return { result: "No subscription found for user :: " + userId };
    }

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions();

    WL.Logger.debug("submitNotification >> userId :: " + userId + ", t");

    WL.Server.notifyAllDevices(userSubscription, {
        badge: 1,
        sound: "sound.mp3",
        activateButtonLabel: "ClickMe",
        alert: notificationText,
        payload: {
            foo : 'bar'
        }
    });

    return { result: "Notification sent to user :: " + userId };
}
```

ユーザーのデバイスごとのサブスクリプション・データを個別に取得するには、**getDeviceSubscriptions** メソッドを使用します。この結果は、次の構造を含むオブジェクトの配列となります。

```
[
  {
    alias: "myPush",
    device: "4AooAq83gUSoas.....",
    token: 'KQz0srTUXs0qh.....',
    applicationId: 'PushApp',
    platform: 'Android',
    options: {
      customOption: 'aaa',
      alert: true,
      badge: true,
      sound: true
    }
  }
]
```

## 通知 API: サーバー・サイド (8/9)

- 通知の送信 (続き)。
  - ユーザー・デバイス (複数可) に通知を送信します。

```
function submitNotification(userId, notificationText){
    var userSubscription =
        WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);

    if (userSubscription==null){
        return { result: "No subscription found for user :: " + userId };
    }

    var deviceSubscriptions =
        userSubscription.getDeviceSubscriptions();

    WL.Logger.debug("submitNotification >> userId :: " + userId);

    WL.Server.notifyAllDevices(userSubscription, {
        badge: 1,
        sound: "sound.mp3",
        activateButtonLabel: "ClickMe",
        alert: notificationText,
        payload: {
            foo : 'bar'
        }
    });

    return { result: "Notification sent to user :: " + userId };
}
```

**notifyAllDevices** メソッドが、ユーザーにサブスクライブされたすべてのデバイスに通知を送信します。カスタム・プロパティーを **payload** オブジェクトで送信できます。

## 通知 API: サーバー・サイド (9/9)

- WL.Server クラスの以下の通知メソッドが使用可能です。
  - すべてのユーザー・デバイスに通知を送信するには、前のスライドに説明されているように、**notifyAllDevices(userSubscription, options)** を使用します。
  - 特定のユーザー・サブスクリプションに属する特定のデバイスに通知を送信するには、**notifyDevice(userSubscription, device, options)** を使用します。
  - 特定のデバイスに通知を送信するには、**notifyDeviceSubscription(deviceSubscription, options)** を使用します。

## 通知 API: クライアント・サイド (1/5)

### クライアントへのトークンの送信および WLPush の初期化

- ユーザーは、アプリケーションの ViewController ロード・メソッド内の WLPush sharedInstance を初期化する必要があります。

```
AppDelegate *appDelegate = [[UIApplication sharedApplication]delegate];
appDelegate.appDelegateVC = self;

[[WLPush sharedInstance]init];
```

- ユーザーは、このメソッドをアプリケーション・デリゲートに追加して、トークンを取得する必要があります。

```
- (void)application:(UIApplication*)application
  didRegisterForRemoteNotificationsWithDeviceToken:(NSData*)deviceToken
{
    NSLog(@"My token from APNS : %@", deviceToken);
    _appDelegateVC.myToken = deviceToken.description;
}
```

- このメソッドによって受け取ったトークンを WLPush メソッドに渡す必要があります。[[WLPush sharedInstance] setTokenFromClient]:

```
[[WLPush sharedInstance] setTokenFromClient:self.myToken];
```



## 通知 API: クライアント・サイド (2/5)

### イベント・ソース – 登録

1. アプリケーション内でイベント・ソースを登録します。  
IBM Worklight Foundation には、イベント・ソースの登録に使用するカスタマイズ可能な `onReadyToSubscribe` 関数が用意されています。
2. `onReadyToSubscribe` 関数をリスナーにセットアップすると、`WLOnReadyToSubscribeListener` が実装されます。この関数は、認証の終了時に呼び出されます。

```
#import "ReadyToSubscribeListener.h"
#import "MyEventListener.h"

@implementation ReadyToSubscribeListener

- (id)initWithController: (ViewController *) mainView{
    if ( self = [super init] )
    {
        vc = mainView;
    }
    return self;
}

-(void)OnReadyToSubscribe{
    [vc updateMessage:@"\nPreparing to subscribe"];
    MyEventListener *eventSourceListener=[[MyEventListener alloc]init];
    [[WLPush sharedInstance] registerEventSourceCallback:self.alias :self.adapterName
                                                         :self.eventSourceName :eventSourceListener];
    [vc updateMessage:@"Ready to subscribe..."];
}

@end
```

## 通知 API: クライアント・サイド (3/5)

### イベント・ソースへのサブスクライブ

前提条件: サブスクライブするためには、ユーザーは認証を行う必要があります。

- イベント・ソースをサブスクライブするには、次の API を使用します。

```
- (IBAction)subscribe:(id)sender {
    self.result.text=@"Trying to subscribe ...";
    MySubscribeListener *mySubscribeListener = [[MySubscribeListener alloc] initWithController:self];
    [[WLPush sharedInstance]subscribe:self.alias :nil :mySubscribeListener];
}
```

- `[[WLPush sharedInstance] subscribe]` は、以下のパラメーターを使用します。
  - `[[WLPush sharedInstance] registerEventSourceCallback]` 内で宣言された別名
  - 任意指定の `onSuccess` デリゲート
  - 任意指定の `onFailure` デリゲート
- デリゲートは、必要に応じて応答オブジェクトを受け取ります。

## 通知 API: クライアント・サイド (4/5)

### イベント・ソースからのアンサブスクライブ

- イベント・ソースからアンサブスクライブするには、次の API を使用します。

```
- (IBAction)unsubscribe:(id)sender {
    self.result.text = @"Trying to unsubscribe ... ";
    MyUnsubscribeListener *myUnsubscribeListener = [[MyUnsubscribeListener alloc]
                                                    initWithController:self];
    [[WLPush sharedInstance]unsubscribe:self.alias :myUnsubscribeListener];
}
```

```
-(void) onSuccess:(WLResponse *)response{
    [vc updateMessage:@"Successfully got a response for Unsubscribe"];
    [vc updateMessage:response.responseText];
}

-(void) onFailure:(WLFailResponse *)response{
    [vc updateMessage:@"Failed get a response for Unsubscribe"];
    [vc updateMessage:response.responseText];
}
```

- `[[WLPush sharedInstance] unsubscribe]` は、以下のパラメーターを使用します。
  - `WL.Client.Push.registerEventSourceCallback` 内で宣言された別名
  - 任意指定の `onSuccess` デリゲート
  - 任意指定の `onFailure` デリゲート
- デリゲートは、必要に応じて応答オブジェクトを受け取ります。

## 通知 API: クライアント・サイド (5/5)

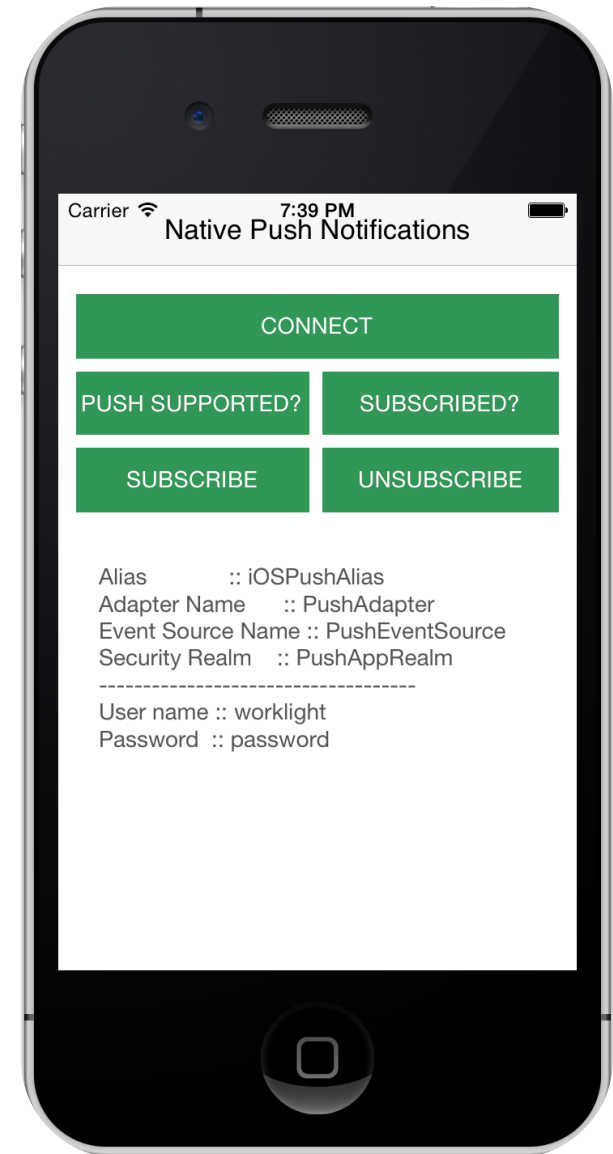
- その他のクライアント・サイド・メソッド:
  - **[[WLPush sharedInstance]isPushSupported]** – プッシュ通知がプラットフォームでサポートされている場合には `true` を返し、それ以外の場合には `false` を返します。
  - **[[WIPush sharedInstance]isSubscribed :alias]** – 現在ログインしているユーザーが指定のイベント・ソース別名にサブスクライブしているかどうかを返します。
- プッシュ通知をデバイスで受信したときに、アプリケーション・デリゲート内の **didReceiveRemoteNotification** メソッドが呼び出されます。

```
- (void)application:(UIApplication*)application didReceiveRemoteNotification
    |:(NSDictionary*)userInfo
{
    _appDelegateVC.result.text = userInfo.description;
}
```

- プッシュ通知の受信時にアプリケーションがバックグラウンド・モード (非アクティブ) であった場合、アプリケーションがフォアグラウンドに戻ると、このコールバックが呼び出されます。

## プロシージャー応答の受信

- このトレーニング・モジュールのサンプルは、IBM Worklight Foundation 文書 Web サイト (<http://www.ibm.com/mobile-docs>) の「入門」ページにあります。
- このサンプルには以下の 2 つのプロジェクトが含まれます。
  - **PushNotificationsNative.zip** には、Worklight Server にデプロイする Worklight ネイティブ API が含まれます。
  - **iOSNativePush.zip** には、Worklight ネイティブ API ライブラリーを使用して Worklight Server と通信するネイティブ iOS アプリケーションが含まれます。
- 関連するサーバー設定を使用して、iOSNativeApp 内の wlclient.plist ファイルを必ず更新するようにしてください。



# アジェンダ

- プッシュ通知とは
- プッシュ通知用の Worklight ネイティブ API の作成
- iOS ネイティブ・アプリケーションの作成および構成
- WLClient および WLPush の初期化
- サブスクリプション管理
- 通知 API
- タグ・ベース通知とブロードキャスト通知

## タグ・ベース通知

- タグは、ユーザーが興味を示すトピックを表し、選択した興味あるトピックに従って通知を受けられる機能を提供します。
- この通知タイプは、タグによりフィルタリングされたメッセージをデバイスが送信および受信できるようにします。
- タグ・ベース通知の受信を開始するには、デバイスがまずアプリケーションでプッシュ通知タグにサブスクライブする必要があります。
- タグは application-descriptor.xml ファイルで次のように定義されています。

```
<tags>
  <tag>
    <name>PushTag1</name>
    <description>About PushTag1</description>
  </tag>
  <tag>
    <name>PushTag2</name>
    <description>About PushTag2</description>
  </tag>
</tags>
```

- このような通知は、アプリケーションでタグにサブスクライブしているすべてのデバイスをターゲットにしています。

## タグ・ベース通知

- クライアント・サイド・メソッド:

- `[[WLPush sharedInstance]subscribeTag:tagName:options]`

指定されたタグ名にデバイスをサブスクライブします。

- `[[WLPush sharedInstance]unsubscribeTag:tagName:options]`

指定されたタグ名からデバイスをアンサブスクライブします。

- `[[WLPush sharedInstance]isTagSubscribed:tagName]`

指定されたタグ名にデバイスがサブスクライブされているかどうかを返します。



## タグ・ベース通知

タグ・ベース通知についての詳細は、「IBM Worklight Foundation ユーザー文書」の「[タグ・ベース通知](#)」を参照してください。

## ブロードキャスト通知

- ブロードキャスト通知は、プッシュ対応の Worklight アプリケーションの場合にデフォルトで有効です。予約されたタグ `Push.ALL` へのサブスクリプションが、すべてのデバイスに対して作成されます。
- ブロードキャスト通知は、予約されたタグ `Push.ALL` からアンサブスクライブすることにより、使用不可にできます。

## ブロードキャスト通知

ブロードキャスト通知についての詳細は、「IBM Worklight Foundation ユーザー文書」の「[ブロードキャスト通知](#)」を参照してください。

## タグ・ベース通知とブロードキャスト通知の共通 API (1/2)

- クライアント・サイド API:

- 通知をデバイスで受信したときに、アプリケーション・デリゲート内の **didReceiveRemoteNotification** メソッドが呼び出されます。

```
- (void)application:(UIApplication*)application didReceiveRemoteNotification
    :(NSDictionary*)userInfo
{
    _appDelegateVC.result.text = userInfo.description;
}
```

`userInfo - payload` フィールドを含む JSON ブロック。このフィールドは、Worklight Server から送信される他のデータを保持します。タグ通知およびブロードキャスト通知のタグ名も含んでいます。タグ名は `tag` エlement に表示されます。ブロードキャスト通知では、デフォルトのタグ名は `Push.ALL` です。

## タグ・ベース通知とブロードキャスト通知の共通 API (2/2)

- サーバー・サイド API:
  - `WL.Server.sendMessage(applicationId, notificationOptions)`

このメソッドは、指定されたターゲット・パラメーターに基づいて通知を送信し、次の2つの必須パラメーターを使用します。

- `applicationId` - Worklight アプリケーションの名前。
  - `notificationOptions` - メッセージ・プロパティを含む JSON ブロック。
- メッセージ・プロパティの完全なリストは、「IBM Worklight Foundation ユーザー文書」を参照してください。

# 特記事項

- これらの資料は、以下のご使用条件に同意していただける場合に限りご使用いただけます。
- 本書は米国 IBM が提供する製品およびサービスについて作成したものです。
- 本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。
- IBM は、本書に記載されている内容に関して特許権（特許出願中のものを含む）を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。
  - 〒103-8510  
東京都中央区日本橋箱崎町19番21号  
日本アイ・ビー・エム株式会社  
法務・知的財産  
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態を提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

- この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。
- 本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。
- IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。
- 本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。
  - IBM Corporation  
Dept F6, Bldg 1  
294 Route 100  
Somers NY 10589-3216  
USA

- 本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。
- 本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。
- IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っていません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確認できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお問い合わせください。

## 著作権使用許諾:

- 本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほめめしたり、保証することはできません。
- それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。
  - © (お客様の会社名) (西暦年) このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。© Copyright IBM Corp. 年を入れる。 All rights reserved.

## プライバシー・ポリシーの考慮事項

- サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品（「ソフトウェア・オファリング」）では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。ご使用の「ソフトウェア・オファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的な事項を確認ください。
- このソフトウェア・オファリングは、展開される構成に応じて、(アプリケーション・サーバーが生成する) セッション情報を収集するセッションごとの Cookie を使用場合があります。これらの Cookie は個人情報を含まず、セッション管理のために要求されるものです。加えて、匿名ユーザーの認識および管理のために持続的な Cookie が無作為に生成される場合があります。これらの Cookie も個人情報を含まず、要求されるものです。
- この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。このような目的での Cookie を含む様々なテクノロジーの使用の詳細については、IBM の『IBM オンラインでのプライバシー・ステートメント』(<http://www.ibm.com/privacy/details/jp/ja/>) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』および『IBM Software Products and Software-as-a-Service Privacy Statement』(<http://www.ibm.com/software/info/product-privacy>) を参照してください。

# サポートおよびコメント

- IBM Worklight の一連の文書、トレーニング資料、および質問をポストできるオンライン・フォーラムはすべて、次の IBM Web サイトからご覧になれます。
  - <http://www.ibm.com/mobile-docs>
- サポート
  - ソフトウェア・サブスクリプション & サポート (ソフトウェア・メンテナンスと呼ばれる場合もあります) は、パスポート・アドバンテージおよびパスポート・アドバンテージ・エクスプレスから購入されたライセンスに含まれています。International Passport Advantage Agreement および IBM International Passport Advantage Express Agreement の追加情報については、次のパスポート・アドバンテージ Web サイトを参照してください。
    - <http://www.ibm.com/software/passportadvantage>
  - ソフトウェア・サブスクリプション & サポートが有効になっている場合、IBM は、インストールおよび使用法 (ハウツー) に関する短期間の FAQ に対するサポートや、コード関連の質問に対するサポートを提供します。詳しくは、次の IBM ソフトウェア・サポート・ハンドブックを参照してください。
    - <http://www.ibm.com/support/handbook>
- ご意見
  - 本資料に関するご意見をお寄せください。本資料の具体的な誤りや欠落、正確性、編成、題材、または完成度に関するご意見をお寄せください。お寄せいただくご意見は、本マニュアルまたは製品の情報、およびその情報の提示方法に関するもののみとしてください。
  - 製品の技術的な質問および情報、および価格については、担当の IBM 営業所、IBM ビジネス・パートナー、または認定リマーカーターにお問い合わせください。
  - IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。IBM またはいかなる組織も、お客様から提示された問題についてご連絡を差し上げる場合にのみ、お客様が提供する個人情報を使用するものとします。
  - どうぞよろしくお願いいたします。
  - 次の IBM Worklight Developer Edition サポート・コミュニティにご意見をお寄せください。
    - <https://www.ibm.com/developerworks/mobile/worklight/connect.html>
  - IBM からの回答を希望される場合は、以下の情報をご連絡ください。
    - 氏名
    - 住所
    - 企業または組織
    - 電話番号
    - E メール・アドレス

ありがとうございました

