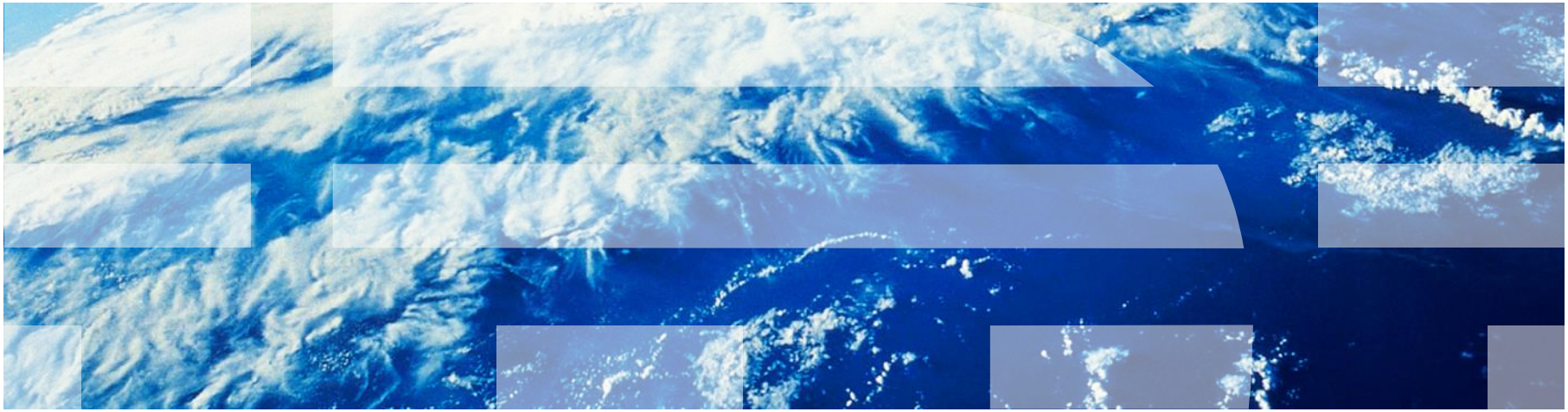
IBM Worklight Foundation V6.2.0
入門

ハイブリッド・アプリケーションでのアダプター・ベースの認証



2014 年 6 月 20 日

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

- アダプター・ベースの認証の概要
- authenticationConfig.xml ファイルの構成
- サーバー・サイドの認証コンポーネントの作成
- クライアント・サイドの認証コンポーネントの作成
- 結果の確認
- 演習
- クイズ

アダプター・ベースの認証の概要

- アダプター・ベースの認証は、実装時の柔軟性が最も高い認証タイプであり、Worklight® Server 認証フレームワークのあらゆるメリットが含まれています。
- アダプター・ベースの認証を使用する場合は、簡潔な JavaScript™ を使用して、資格情報の検証を含む認証ロジック全体をアダプターに実装できます。
- ただし、追加の認証レイヤーとして任意のログイン・モジュールを使用することもできます。
- このモジュールでは、ユーザー名とパスワードに依存するアダプター・ベースの認証メカニズムを実装します。

アジェンダ

- アダプター・ベースの認証の概要
- authenticationConfig.xml ファイルの構成
- サーバー・サイドの認証コンポーネントの作成
- クライアント・サイドの認証コンポーネントの作成
- 結果の確認
- 演習
- クイズ

authenticationConfig.xml ファイルの構成

- **authenticationConfig.xml** ファイルの `<realms>` セクションに 2 つの認証レルムを追加します。

```
<realm loginModule="AuthLoginModule" name="SingleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="SingleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="SingleStepAuthAdapter.onLogout"/>
</realm>
<realm loginModule="AuthLoginModule" name="DoubleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="DoubleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="DoubleStepAuthAdapter.onLogout"/>
</realm>
```

- これらのレルムでは `AuthLoginModule` ログイン・モジュールを使用しますが、これは後で定義します。

authenticationConfig.xml ファイルの構成

- **authenticationConfig.xml** ファイルの `<realms>` セクションに 2 つの認証レルムを追加します。

```
<realm loginModule="AuthLoginModule" name="SingleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="SingleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="SingleStepAuthAdapter.onLogout"/>
</realm>
<realm loginModule="AuthLoginModule" name="DoubleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="DoubleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="DoubleStepAuthAdapter.onLogout"/>
</realm>
```

- `com.worklight.integration.auth.AdapterAuthenticator` クラスを使用すると、オーセンティケーターのサーバー・サイド部分がアダプターで定義されることになります。

authenticationConfig.xml ファイルの構成

- **authenticationConfig.xml** ファイルの `<realms>` セクションに 2 つの認証レルムを追加します。

```
<realm loginModule="AuthLoginModule" name="SingleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="SingleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="SingleStepAuthAdapter.onLogout"/>
</realm>
<realm loginModule="AuthLoginModule" name="DoubleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="DoubleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="DoubleStepAuthAdapter.onLogout"/>
</realm>
```

- Worklight 認証フレームワークが保護リソースへのアクセスの試行を検出すると、そのたびに、**login-function パラメーター**で定義されているアダプター関数が自動的に呼び出されます。
- ログアウト (明示的またはセッション・タイムアウト) が検出されると、**logout-function** が自動的に呼び出されます。
- どちらの場合も、パラメーター値の構文は `adapterName.functionName` です。

authenticationConfig.xml ファイルの構成

- ログイン・モジュールを **authenticationConfig.xml** ファイルの `<loginModules>` セクションに追加し、`AuthLoginModule` という名前を付けます。

```
<loginModule name="AuthLoginModule">
    <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
</loginModule>
```

- `NonValidatingLoginModule` クラス名を使用すると、Worklight プラットフォームによる追加検証がまったく実行されないため、開発者の責任においてアダプター内の資格情報の検証を行う必要があります。
- 認証関連のアクションはすべてアダプター・コードで実行されるので、アダプター・ベースの認証では `NonValidatingLoginModule` を使用することが必須となります。

authenticationConfig.xml ファイルの構成

- セキュリティー・テストを **authenticationConfig.xml** ファイルの `<securityTests>` セクションに追加します。
- このセキュリティー・テストを使用してアダプター・プロシージャーを保護する必要があります。したがって、`<customSecurityTest>` エレメントを使用します。

```
<customSecurityTest name="SingleStepAuthAdapter-securityTest">
    <test isInternalUserID="true" realm="SingleStepAuthRealm"/>
</customSecurityTest>
<customSecurityTest name="DoubleStepAuthAdapter-securityTest">
    <test isInternalUserID="true" realm="DoubleStepAuthRealm"/>
</customSecurityTest>
```

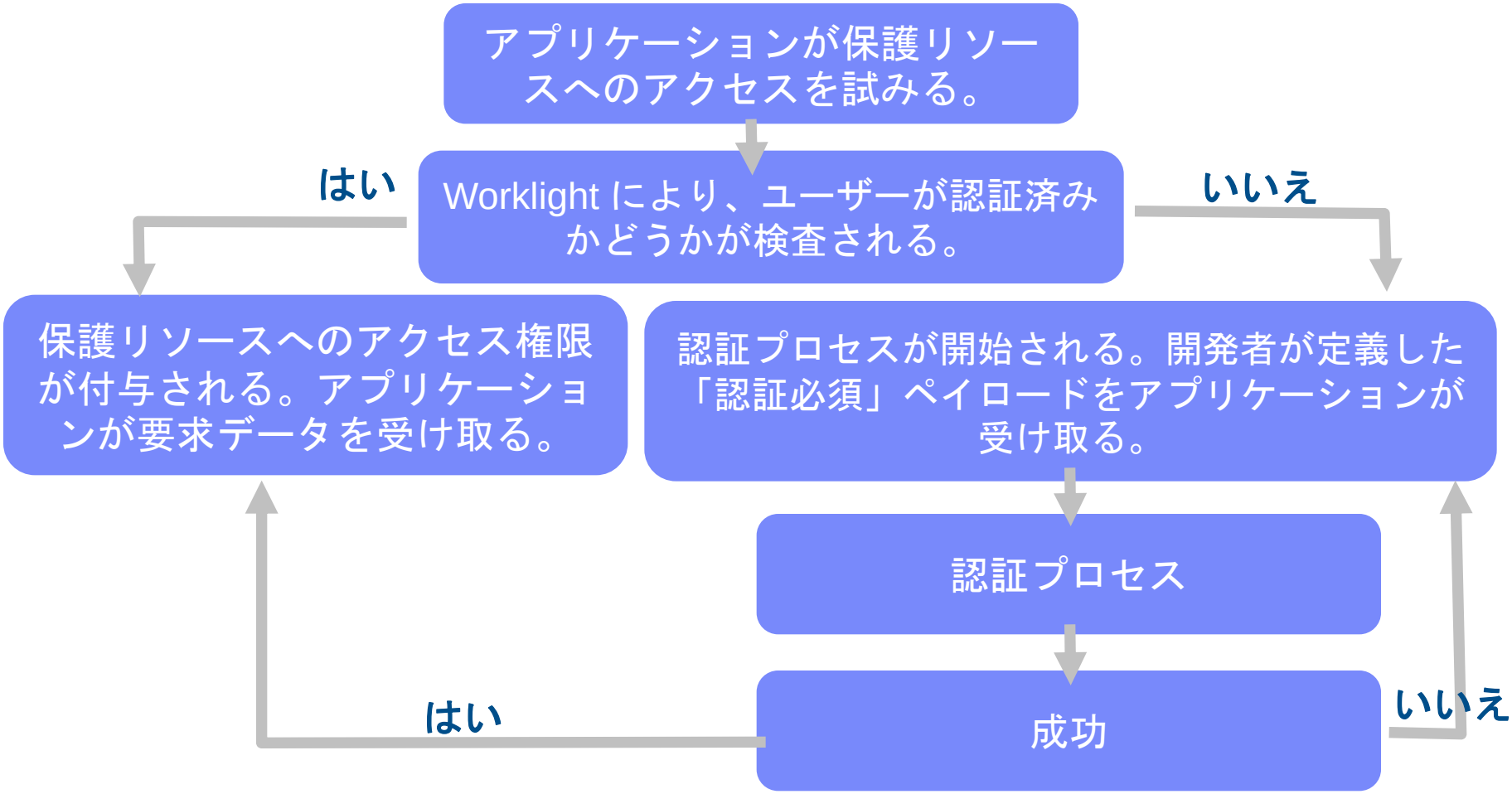
- セキュリティー・テスト名を覚えておいてください。以降のスライドで、それらの名前を使用する必要があります。

アジェンダ

- アダプター・ベースの認証の概要
- authenticationConfig.xml ファイルの構成
- サーバー・サイドの認証コンポーネントの作成
- クライアント・サイドの認証コンポーネントの作成
- 結果の確認
- 演習
- クイズ

サーバー・サイドの認証コンポーネントの作成

- 以下のダイアグラムは、アダプター・ベースの認証プロセスを示しています。



サーバー・サイドの認証コンポーネントの作成

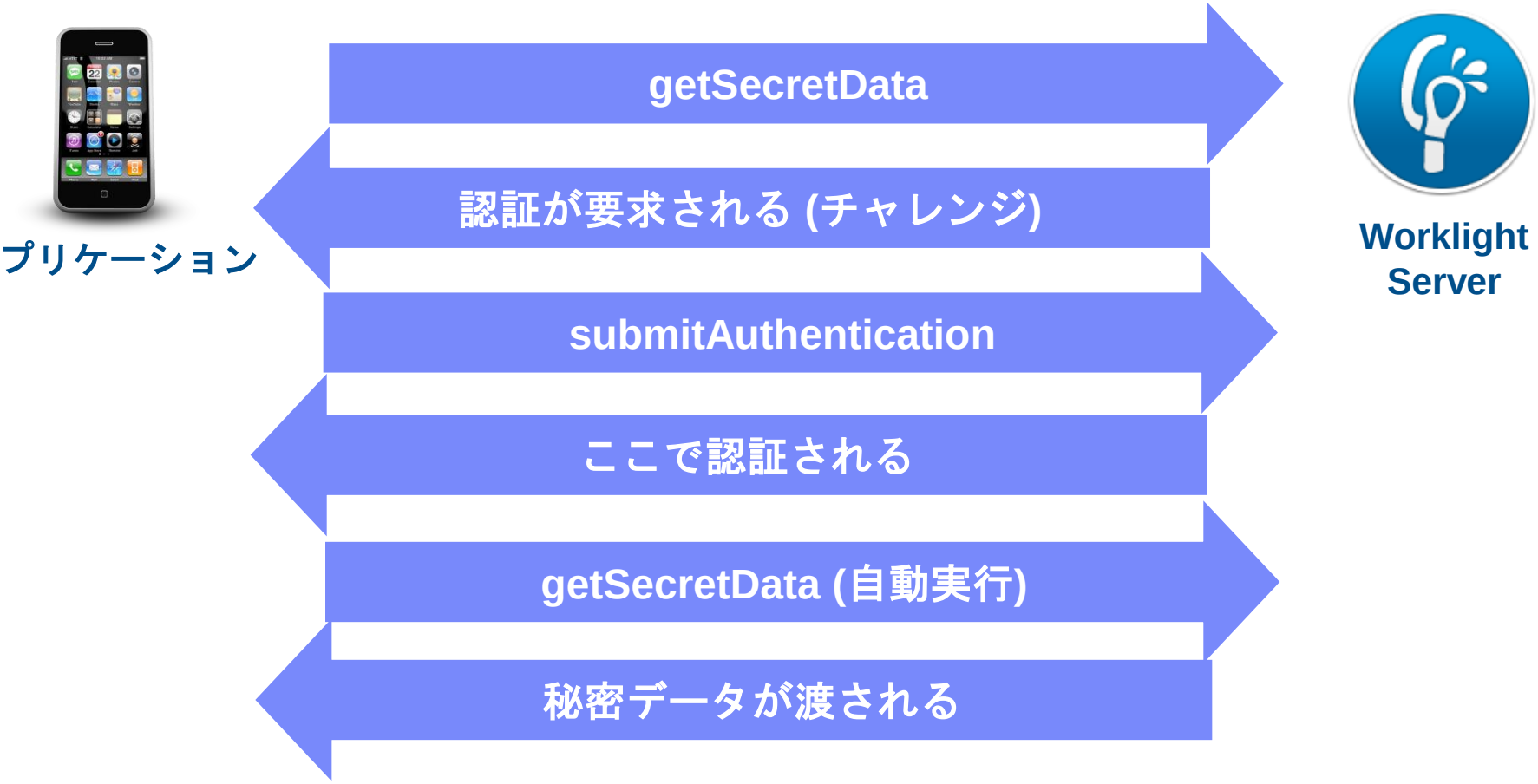
- このトレーニング・モジュールで紹介する例では、2 つのアプリケーションと 2 つのアダプターを使用します。次ページ以降のスライドでは、SingleStepAuth アプリケーションおよびアダプターに焦点を当てます。DoubleStepAuth アプリケーションおよびアダプターは、同じ手法を拡張したものにすぎません。
- 認証プロセスを扱うアダプターを作成し、**SingleStepAuthAdapter** という名前を付けます。
- **SingleStepAuthAdapter** には、以下の 2 つのプロシージャーが含まれます。

```
<procedure name="submitAuthentication"/>

<procedure name="getSecretData" securityTest="AdapterSecurityTest"/>
```

- `submitAuthentication` プロシージャーは認証プロセスを扱います。このプロシージャーの呼び出しに認証は必要ありません。
- ただし、2 番目のプロシージャーは、認証されたユーザーしか使用できません。

サーバー・サイドの認証コンポーネントの作成

- 以下のダイアグラムは、実装フローを示しています。



サーバー・サイドの認証コンポーネントの作成

- Worklight フレームワークが保護リソースへの非認証アクセスの試行を検出すると、そのたびに `onAuthRequired` 関数が呼び出されます (この動作は、**authenticationConfig.xml** ファイルでの定義に基づいています)。

```
function onAuthRequired(headers, errorMessage){
    errorMessage = errorMessage ? errorMessage : null;
    return {
        authRequired: true,
        errorMessage: errorMessage
    };
}
```

このオブジェクトは、アプリケーションに送信される**カスタム**・チャレンジ・オブジェクトです。

- この関数は、応答ヘッダーとオプションの `errorMessage` パラメーターを受け取ります。この関数によって返されるオブジェクトは、クライアント・アプリケーションに送信されます。
- `authRequired: true` というプロパティーに注意してください。チャレンジ・ハンドラーでこのプロパティーを使用して、サーバーが認証を要求していることを検出します。

サーバー・サイドの認証コンポーネントの作成

- クライアント・アプリケーションによって `submitAuthentication` 関数が呼び出され、ユーザー名とパスワードが検証されます。

```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        var userIdentity = {
                userId: username,
                displayName: username,
                attributes: {
                    foo: "bar"
                }
        };

        WL.Server.setActiveUser("SingleStepAuthRealm", userIdentity);

        return {
            authRequired: false
        };
    }

    return onAuthRequired(null, "Invalid login credentials");
}
```

ユーザー名とパスワードは、アプリケーションからパラメーターとして受け取ります。

サーバー・サイドの認証コンポーネントの作成

- クライアント・アプリケーションによって `submitAuthentication` 関数が呼び出され、ユーザー名とパスワードが検証されます。

```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        var userIdentity = {
                userId: username,
                displayName: username,
                attributes: {
                    foo: "bar"
                }
        };

        WL.Server.setActiveUser("SingleStepAuthRealm", userIdentity);

        return {
            authRequired: false
        };
    }

    return onAuthRequired(null, "Invalid login credentials");
}
```

この例では、ハードコーディングされた値との照合によって資格情報が検証されますが、他の任意の検証モード (例えば、SQL や Web サービスなどを使用した検証) も有効です。

サーバー・サイドの認証コンポーネントの作成

- クライアント・アプリケーションによって `submitAuthentication` 関数が呼び出され、ユーザー名とパスワードが検証されます。

```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        var userIdentity = {
                userId: username,
                displayName: username,
                attributes: {
                    foo: "bar"
                }
        };

        WL.Server.setActiveUser("SingleStepAuthRealm", userIdentity);

        return {
            authRequired: false
        };
    }

    return onAuthRequired(null, "Invalid login credentials");
}
```

検証が正常に行われると、WL.Server.setActiveUser メソッドが呼び出され、userIdentity オブジェクトに格納されているユーザー・データを使用して SingleStepAuthRealm の認証済みセッションが作成されます。独自のカスタム・プロパティーをユーザー ID 属性に追加できます。

サーバー・サイドの認証コンポーネントの作成

- クライアント・アプリケーションによって `submitAuthentication` 関数が呼び出され、ユーザー名とパスワードが検証されます。

```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        var userIdentity = {
                userId: username,
                displayName: username,
                attributes: {
                    foo: "bar"
                }
        };

        WL.Server.setActiveUser("SingleStepAuthRealm", userIdentity);

        return {
            authRequired: false
        };
    }

    return onAuthRequired(null, "Invalid login credentials");
}
```

認証画面が不要になったことを示すオブジェクトがアプリケーションに送信されます。

サーバー・サイドの認証コンポーネントの作成

- クライアント・アプリケーションによって `submitAuthentication` 関数が呼び出され、ユーザー名とパスワードが検証されます。

```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        var userIdentity = {
                userId: username,
                displayName: username,
                attributes: {
                    foo: "bar"
                }
        };

        WL.Server.setActiveUser("SingleStepAuthRealm", userIdentity);

        return {
            authRequired: false
        };
    }

    return onAuthRequired(null, "Invalid login credentials");
}
```

資格情報の検証に失敗すると、onAuthRequired 関数によって作成されたオブジェクトが、適切なエラー・メッセージと共にアプリケーションに返されます。

サーバー・サイドの認証コンポーネントの作成

- トレーニングを目的としているため、この例の `getSecretData` 関数は、ハードコーディングされた値を返します。なお、`getSecretData` がセキュリティー・テストにより保護されている (このことはアダプター XML で定義されている) ことに注意してください。
- `onLogout` 関数は **authenticationConfig.xml** ファイル内での定義に従って、ログアウト時に自動的に呼び出され、例えばクリーンアップなどを実行します。

```
function getSecretData(){
    return {
        secretData: "A very very very very secret data"
    };
}

function onLogout(){
    WL.Logger.debug("Logged out");
}
```

アジェンダ

- アダプター・ベースの認証の概要
- authenticationConfig.xml ファイルの構成
- サーバー・サイドの認証コンポーネントの作成
- クライアント・サイドの認証コンポーネントの作成
- 結果の確認
- 演習
- クイズ

クライアント・サイドの認証コンポーネントの作成

- Worklight アプリケーションを作成します。
- このアプリケーションは、以下の 2 つの `<div>` エレメントで構成されます。
  - **`<div id="AppDiv">`** エレメントは、アプリケーション・コンテンツの表示に使用します。
  - **`<div id="AuthDiv">`** エレメントは、認証フォームに使用します。
- 認証が要求されると、アプリケーションは `AppDiv` エレメントを非表示にして `AuthDiv` エレメントを表示します。
- 認証が完了すると、アプリケーションは表示と非表示を逆にします。

クライアント・サイドの認証コンポーネントの作成

- 最初に AppDiv エレメントを作成します。
- これには以下の基本構造と関数が含まれています。

```
<div id="AppDiv">
    <div class="header">
        <h1>Single Step Adapter Based Authentication</h1>
    </div>
    <input type="button" value="Get secret data" onclick="getSecretData()" />
    <input type="button" value="Logout" onclick="WL.Client.logout('SingleStepAuthRealm', {onSuccess:WL.Client.reloadApp})" />
    <div id="ResponseDiv"></div>
</div>
```

- ボタンは、getSecretData プロシージャーの呼び出しとログアウトに使用します。
- エレメントは、getSecretData の応答の表示に使用します。

クライアント・サイドの認証コンポーネントの作成

- AuthDiv エレメントには以下のサブエレメントが含まれます。

```
<div id="AuthDiv" style="display:none">
    <div class="header">
        <h1>Single Step Adapter Based Authentication</h1>
    </div>
    <p id="AuthInfo"></p>
    <input type="text" placeholder="Enter username" id="AuthUsername"/><br />
    <input type="password" placeholder="Enter password" id="AuthPassword"/><br />
    <input type="button" value="Submit" id="AuthSubmitButton" />
    <input type="button" value="Cancel" id="AuthCancelButton" />
</div>
```

  - AuthInfo は、エラー・メッセージを表示するために使用します。
  - AuthUsername および AuthPassword は、エレメントを入力するために使用します。
  - AuthSubmitButton および AuthCancelButton は、認証要求を送信またはキャンセルするために使用します。
- AuthDiv エレメントのスタイルは、display:none と指定されています。これは、サーバーによって認証が要求される前に表示されてはならないからです。

クライアント・サイドの認証コンポーネントの作成

- 最後に、チャレンジ・ハンドラーを作成します。
- 以下の API を使用してこのハンドラーを作成し、ハンドラーの機能を実装します。

```
var singleStepAuthRealmChallengeHandler = WL.Client.createChallengeHandler("SingleStepAuthRealm");

singleStepAuthRealmChallengeHandler.isCustomResponse = function(response) {
    if (!response || !response.responseJSON || response.responseText === null) {
        return false;
    }
    if (typeof(response.responseJSON.authRequired) !== 'undefined'){
        return true;
    } else {
        return false;
    }
};
```

WL.Client.createChallengeHandler メソッドを使用してチャレンジ・ハンドラー・オブジェクトを作成します。
レルム名をパラメーターとして指定します。

チャレンジ・ハンドラーを作成して、カスタマイズされた認証フローを定義します。
認証フローと関係のない変更をユーザー・インターフェースに対して行うコードは、
チャレンジ・ハンドラーに追加しないでください。

クライアント・サイドの認証コンポーネントの作成

- 最後に、チャレンジ・ハンドラーを作成します。
- 以下の API を使用してこのハンドラーを作成し、ハンドラーの機能を実装します。

```
var singleStepAuthRealmChallengeHandler = WL.Client.createChallengeHandler("SingleStepAuthRealm");

singleStepAuthRealmChallengeHandler.isCustomResponse = function(response) {
    if (!response || !response.responseJSON || response.responseText === null) {
        return false;
    }
    if (typeof(response.responseJSON.authRequired) !== 'undefined'){
        return true;
    } else {
        return false;
    }
};
```

チャレンジ・ハンドラーの isCustomResponse 関数は、サーバーから応答を受け取るたびに呼び出されます。この関数は、このチャレンジ・ハンドラーに関連するデータが応答に含まれているかどうかを検出するために使用します。
戻り値は、`true` または `false` です。

クライアント・サイドの認証コンポーネントの作成

- 最後に、チャレンジ・ハンドラーを作成します。
- 以下の API を使用してこのハンドラーを作成し、ハンドラーの機能を実装します。

```
singleStepAuthRealmChallengeHandler.handleChallenge = function(response){
    var authRequired = response.responseJSON.authRequired;

    if (authRequired == true){
        $("#AppDiv").hide();
        $("#AuthDiv").show();
        $("#AuthPassword").empty();
        $("#AuthInfo").empty();

        if (response.responseJSON.errorMessage)
            $("#AuthInfo").html(response.responseJSON.errorMessage);

    } else if (authRequired == false){
        $("#AppDiv").show();
        $("#AuthDiv").hide();
        singleStepAuthRealmChallengeHandler.submitSuccess();
    }
};
```

`isCustomResponse` 関数が `true` を返した場合、フレームワークは `handleChallenge` 関数を呼び出します。この関数は、必要なアクション (例えば、アプリケーション画面の非表示、ログイン画面の表示など) を実行するために使用します。

クライアント・サイドの認証コンポーネントの作成

- 最後に、チャレンジ・ハンドラーを作成します。
- 以下の API を使用してこのハンドラーを作成し、ハンドラーの機能を実装します。

```
singleStepAuthRealmChallengeHandler.handleChallenge = function(response){
    var authRequired = response.responseJSON.authRequired;

    if (authRequired == true){
        $("#AppDiv").hide();
        $("#AuthDiv").show();
        $("#AuthPassword").empty();
        $("#AuthInfo").empty();

        if (response.responseJSON.errorMessage)
            $("#AuthInfo").html(response.responseJSON.errorMessage);

    } else if (authRequired == false){
        $("#AppDiv").show();
        $("#AuthDiv").hide();
        singleStepAuthRealmChallengeHandler.submitSuccess();
    }
};
```

authRequired が true である場合には、ログイン画面が表示され、パスワード・フィールドがクリーンアップされ、さらにエラー・メッセージがあればそれが表示されます。

クライアント・サイドの認証コンポーネントの作成

- 最後に、チャレンジ・ハンドラーを作成します。
- 以下の API を使用してこのハンドラーを作成し、ハンドラーの機能を実装します。

```
singleStepAuthRealmChallengeHandler.handleChallenge = function(response){
    var authRequired = response.responseJSON.authRequired;

    if (authRequired == true){
        $("#AppDiv").hide();
        $("#AuthDiv").show();
        $("#AuthPassword").empty();
        $("#AuthInfo").empty();

        if (response.responseJSON.errorMessage)
            $("#AuthInfo").html(response.responseJSON.errorMes

    } else if (authRequired == false){
        $("#AppDiv").show();
        $("#AuthDiv").hide();
        singleStepAuthRealmChallengeHandler.submitSuccess();
    }
};
```

`authRequired` が `false` である場合には、AppDiv が表示され、AuthDiv が非表示にされ、さらに認証が正常に完了したことが Worklight フレームワークに通知されます。

クライアント・サイドの認証コンポーネントの作成

- チャレンジ・ハンドラーには、開発者が実装しなければならないメソッドに加え、開発者が必要に応じて使用できる以下のような機能も含まれています。
  - `submitAdapterAuthentication` 関数は、収集した資格情報を特定のアダプター・プロシージャーに送信するために使用します。この関数は、`WL.Client.invokeProcedure` API と同じシグニチャーを持ちます。
  - `submitSuccess` 関数は、認証プロセスが正常に完了したことを Worklight フレームワークに通知します。Worklight フレームワークはその後で、認証をトリガーした元の要求を自動的に発行します。
  - `submitFailure` 関数は、認証プロセスが失敗に終わったことを Worklight フレームワークに通知します。Worklight フレームワークはその後で、認証をトリガーした元の要求を破棄します。

**** 注: これらの各関数は対応するオブジェクトに付加する必要があります。例えば、次のように使用します。***

`myChallengeHandler.submitSuccess()`

クライアント・サイドの認証コンポーネントの作成

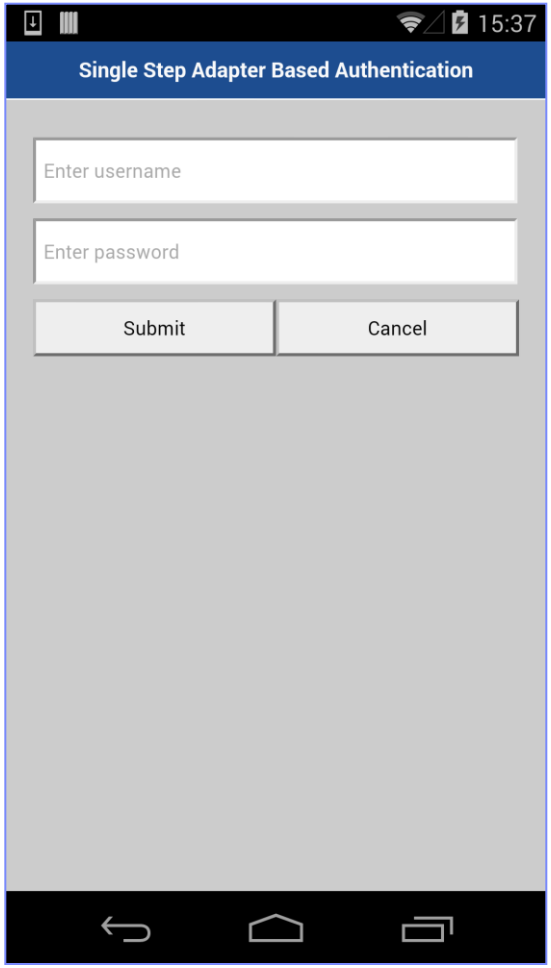
- 送信ボタンをクリックすると、HTML 入力フィールドからユーザー名とパスワードを収集してアダプターに送信する関数がトリガーされます。
- チャレンジ・ハンドラーが `submitAdapterAuthentication` メソッドを使用する点に注意してください。

```
$("#AuthSubmitButton").bind('click', function () {
    var username = $("#AuthUsername").val();
    var password = $("#AuthPassword").val();

    var invocationData = {
        adapter : "SingleStepAuthAdapter",
        procedure : "submitAuthentication",
        parameters : [ username, password ]
    };

    singleStepAuthRealmChallengeHandler.submitAdapterAuthentication(invocationData, {});
});
```

アジェンダ

- アダプター・ベースの認証の概要
- authenticationConfig.xml ファイルの構成
- サーバー・サイドの認証コンポーネントの作成
- クライアント・サイドの認証コンポーネントの作成
- 結果の確認
- 演習
- クイズ

結果の確認



アジェンダ

- アダプター・ベースの認証の概要
- authenticationConfig.xml ファイルの構成
- サーバー・サイドの認証コンポーネントの作成
- クライアント・サイドの認証コンポーネントの作成
- 結果の確認
- 演習
- クイズ

演習

- このトレーニング・モジュールの説明に従ってアダプター認証を実装します。
- 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。

アジェンダ

- アダプター・ベースの認証の概要
- authenticationConfig.xml ファイルの構成
- サーバー・サイドの認証コンポーネントの作成
- クライアント・サイドの認証コンポーネントの作成
- 結果の確認
- 演習
- クイズ

クイズ
確認テスト – 答えは次のスライドにあります

- authenticationConfig.xml でアダプター・ベースの認証を使用するレルムを定義するときに、必須となる 2 つのパラメーターは次のどれですか。
  - login-function、logout-function
  - adapter-name、realm-name
  - adapter-name、login-function
  - login-function、login-module
- 認証レルムによって保護されるアダプター・プロシージャーを開発者が指定する方法として、正しいのは次のどれですか。
  - 認証レルムがアダプター XML ファイルに指定されている場合は、すべてのアダプター・プロシージャーがその認証レルムによって保護される。
  - 開発者が指定する必要はない。WL.Client.invokeProcedure を使用してこのプロシージャーが動作するようにすれば、クライアント・サイドで認証資格情報が追加される。
  - securityTest プロパティーをアダプター XML 内のプロシージャー定義に追加する。
  - 認証レルムによってアダプター・プロシージャーを保護することはできない。保護の対象はアプリケーションのみである。
- サーバーがクライアント要求に対して認証を必要としていることを検出するために使用するクライアント・サイド・メカニズムは次のどれですか。
  - challengeHandler.isAuthenticationRequired
  - challengeHandler.isUserAuthenticated
  - challengeHandler.analyzeServerResponse
  - challengeHandler.isCustomResponse

クイズ- 答え

- authenticationConfig.xml でアダプター・ベースの認証を使用するレルムを定義するときに、必須となる 2 つのパラメーターは次のどれですか。
  - login-function、logout-function
  - adapter-name、realm-name
  - adapter-name、login-function
  - login-function、login-module
- 認証レルムによって保護されるアダプター・プロシージャーを開発者が指定する方法として、正しいのは次のどれですか。
  - 認証レルムがアダプター XML ファイルに指定されている場合は、すべてのアダプター・プロシージャーがその認証レルムによって保護される。
  - 開発者が指定する必要はない。WL.Client.invokeProcedure を使用してこのプロシージャーが動作するようにすれば、クライアント・サイドで認証資格情報が追加される。
  - securityTest プロパティーをアダプター XML 内のプロシージャー定義に追加する。
  - 認証レルムによってアダプター・プロシージャーを保護することはできない。保護の対象はアプリケーションのみである。
- サーバーがクライアント要求に対して認証を必要としていることを検出するために使用するクライアント・サイド・メカニズムは次のどれですか。
  - challengeHandler.isAuthenticationRequired
  - challengeHandler.isUserAuthenticated
  - challengeHandler.analyzeServerResponse
  - challengeHandler.isCustomResponse

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で提供し、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おりません。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証できません。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願いします。

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 あることをほのめかしたり、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 を使用する場合があります。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、または認定リマーケ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・コミュニティー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