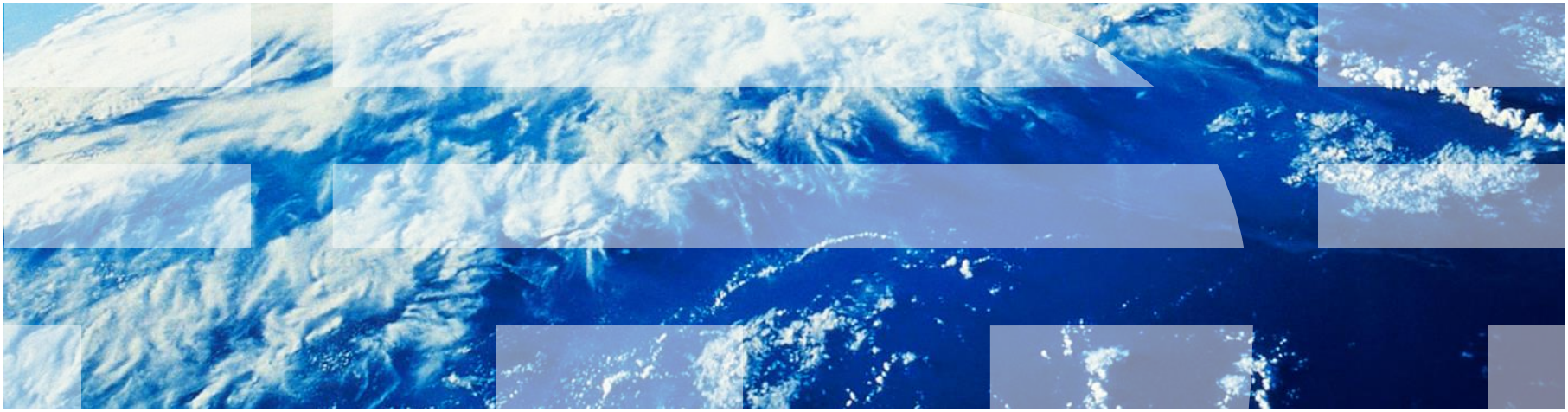
IBM Worklight V6.1.0
入門

カスタム・オーセンティケーターとログイン・モジュール



2014 年 3 月 7 日

商標

- IBM、IBM ロゴおよび ibm.com 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

- 認証の概要
- authenticationConfig.xml の構成
- カスタム Java オーセンティケーターの作成
- カスタム Java ログイン・モジュールの作成
- 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

認証の概要 (1/3)

- 認証プロセスは、対話式 (ユーザー名とパスワードなど) にすることも、非対話式 (ヘッダー・ベースの認証など) にすることもできます。
- これは、単一のステップ (単純なユーザー名/パスワード形式など) で構成することも、複数のステップ (例えば、最初のパスワードの発行後にチャレンジの追加が必要となるものなど) で構成することもできます。
- 認証レルムの定義には、オーセンティケーターのクラス名と、ログイン・モジュールへの参照が含まれます。
- オーセンティケーターは、ユーザー情報を収集するエンティティーです。
  - 例: ログイン・フォーム
- ログイン・モジュールは、取得したユーザー資格情報を検証し、ユーザー ID を作成するサーバー・エンティティーです。
- 認証の設定 (例えば、レルム、オーセンティケーター、ログイン・モジュールなど) は、Worklight Server 上にある `authenticationConfig.xml` ファイルで構成します。

非認証ユーザーが、認証レルムによって保護されたリソースへのアクセスを試みる。

↓

*オーセンティケーター*が呼び出されて、ユーザー資格情報 (すなわち、ユーザー名とパスワード) を収集するために使用される。

ログイン・モジュール が、収集された資格情報を受け取って検証する。

受け取った資格情報が検証を無事に通過すると、ログイン・モジュールが*ユーザー ID* オブジェクトを作成し、指定されたレルムで認証済みであることを示すフラグをセッションに設定する。

認証の概要 (2/3)

- オーセンティケーター、ログイン・モジュール、およびユーザー ID のインスタンスは、セッション範囲で保管されるため、セッションが維持されている間は存続します。
- デフォルトのログイン・モジュールとオーセンティケーターが要件を満たさない場合は、カスタム・ログイン・モジュールとカスタム・オーセンティケーターを作成できます。
- 前のモジュールでは、以下のことを行いました。
  - フォーム・ベースの認証を実装し、検証なしのログイン・モジュールを使用する。
  - ログイン・モジュールを追加せずにアダプター・ベースの認証を実装し、資格情報の検証を手動で実行する。
- 場合によっては、資格情報の検証をアダプター・レベルで実行できず、より複雑なコードが必要になることもありますが、そのような場合には追加のログイン・モジュールを実装できます。
  - 例: 企業のカスタムの資格情報の検証が必要な場合、あるいは、各クライアント要求からより多くの情報 (例えば、Cookie、ヘッダー、ユーザー・エージェントなど) を取得する必要がある場合。

認証の概要 (3/3)

- このモジュールでは、カスタム・オーセンティケーターとログイン・モジュールの作成方法について説明します。
  - 事前定義された URL への要求を使用してユーザー名とパスワードを収集するカスタム・オーセンティケーターの実装方法を学習します。
  - オーセンティケーターから受け取った資格情報を検査するカスタム・ログイン・モジュールの実装方法を学習します。
  - カスタム・オーセンティケーターとログイン・モジュールを使用するレルムの定義方法を学習します。
  - このレルムを使用してリソースを保護する方法を学習します。
- Worklight® での認証の概念について詳しくは、IBM Worklight ユーザー文書を参照してください。

アジェンダ

- 認証の概要
- authenticationConfig.xml の構成
- カスタム Java オーセンティケーターの作成
- カスタム Java ログイン・モジュールの作成
- 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

authenticationConfig.xml の構成 (1/2)

- 認証情報を **authenticationConfig.xml** ファイルに追加します。
- realms セクションで、**CustomAuthenticatorRealm** という名前のレルムを定義します。
  - このレルムが **CustomLoginModule** を使用することを確認します。
- **MyCustomAuthenticator** を className として指定します。この実装は後のスライドで行います。

```
<realm name="CustomAuthenticatorRealm" loginModule="CustomLoginModule">
    <className>com.mypackage.MyCustomAuthenticator</className>
</realm>
```

- loginModules セクションで、**CustomLoginModule** という名前の **loginModule** を追加します。

```
<loginModule name="CustomLoginModule">
    <className>com.mypackage.MyCustomLoginModule</className>
</loginModule>
```

- **MyCustomLoginModule** を className として指定します。この実装は後のスライドで行います。

authenticationConfig.xml の構成 (2/2)

- 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。
- 後で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ため、これを <customSecurityTest> にします。

```
<securityTests>
    <customSecurityTest name="CustomAuthSecurityTest">
        <test isInternalUserID="true" realm="CustomAuthenticatorRealm"/>
    </customSecurityTest>
</securityTests>
```

- セキュリティー・テスト名は以降のスライドで使用するために、覚えておいてください。

アジェンダ

- 認証の概要
- authenticationConfig.xml の構成
- カスタム Java オーセンティケーターの作成
- カスタム Java ログイン・モジュールの作成
- 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

カスタム Java™ オーセンティケーターの作成 (1/21)

- オーセンティケーター API は、以下のとおりです。
  - `void init(Map<String, String> options)`
  - `AuthenticationResult processRequest(HttpServletRequest request, HttpServletResponse response, boolean isAccessToProtectedResource)`
  - `AuthenticationResult processAuthenticationFailure(HttpServletRequest request, HttpServletResponse respo`
  - `AuthenticationResult processRequestAlreadyAuth request, HttpServletRespo`
  - `Map<String, Object> getAu`
  - `HttpServletRequest getReq uest request, HttpServletRespo userIdentity, LoginExtens`
  - `Boolean changeResponseOnS request, HttpServletResponse response)`
  - `WorkLightAuthenticator clone()`

オーセンティケーターの `init()` メソッドは、オーセンティケーター・インスタンスの作成時に呼び出されます。これは、**authenticationConfig.xml** のレルム定義で指定されているオプションを受け取ります。

カスタム Java オーセンティケーターの作成 (2/21)

- オーセンティケーター API は、以下のとおりです。
  - `void init(Map<String, String> options)`
  - `AuthenticationResult processRequest(HttpServletRequest request, HttpServletResponse response, boolean isAccessToProtectedResource)`
  - `AuthenticationResult processAuthenticationFailure(HttpServletRequest request, HttpServletResponse respo`
  - `AuthenticationResult processRequestAlreadyAuth request, HttpServletRespo`
  - `Map<String, Object> getAu`
  - `HttpServletRequest getReq uest request, HttpServletRespo userIdentity, LoginExtens`
  - `Boolean changeResponseOnS request, HttpServletResponse response)`
  - `WorkLightAuthenticator clone()`

`processRequest()` メソッドは、非認証セッションから要求があるたびに呼び出されます。

カスタム Java オーセンティケーターの作成 (3/21)

- オーセンティケーター API は、以下のとおりです。
  - `void init(Map<String, String> options)`
  - `AuthenticationResult processRequest(HttpServletRequest request, HttpServletResponse response, boolean isAccessToProtectedResource)`
  - `AuthenticationResult processAuthenticationFailure(HttpServletRequest request, HttpServletResponse response, String errorMessage)`
  - `AuthenticationResult processRequestAlreadyAuth request, HttpServletRespo`
  - `Map<String, Object> getAu`
  - `HttpServletRequest getRe request, HttpServletRespo userIdentity, LoginExtens`
  - `Boolean changeResponseOnS request, HttpServletRespo`
  - `WorkLightAuthenticator cl`

`processAuthenticationFailure()` メソッドは、ログイン・モジュールが資格情報検証の失敗を返した場合に呼び出されます。

カスタム Java オーセンティケーターの作成 (4/21)

- オーセンティケーター API は、以下のとおりです。
  - `void init(Map<String, String> options)`
  - `AuthenticationResult processRequest(HttpServletRequest request, HttpServletResponse response, boolean isAccessToProtectedResource)`
  - `AuthenticationResult processAuthenticationFailure(HttpServletRequest request, HttpServletResponse response, String errorMessage)`
  - `AuthenticationResult processRequestAlreadyAuthenticated(HttpServletRequest request, HttpServletResponse response)`
  - `Map<String, Object> ge`
  - `HttpServletRequest ge request, HttpServletRe userIdentity, LoginExt`
  - `Boolean changeResponse request, HttpServletRe`
  - `WorkLightAuthenticato`

`processRequestAlreadyAuthenticated()` メソッドは、認証済みセッションから要求があるたびに呼び出されます。

カスタム Java オーセンティケーターの作成 (5/21)

- オーセンティケーター API は、以下のとおりです。
  - void **init**(Map<String, Stri
  - AuthenticationResult **proce**
    request, HttpServletRespon
    isAccessToProtectedResourc
  - AuthenticationResult
    processAuthenticationFailu
    HttpServletResponse respon
  - AuthenticationResult
    processRequestAlreadyAuthe
    request, HttpServletResponse response)
  - Map<String, Object> **getAuthenticationData**()
  - HttpServletRequest **getRequestToProceed**(HttpServletRequest request, HttpServletResponse response, UserIdentity userIdentity, LoginExtension... loginExtension)
  - Boolean **changeResponseOnSuccess** (HttpServletRequest request, HttpServletResponse response)
  - WorkLightAuthenticator **clone**()

getAuthenticationData() メソッドは、オーセンティケーターによって収集された資格情報を取得するために、ログイン・モジュールによって使用されます。

カスタム Java オーセンティケーターの作成 (6/21)

- オーセンティケーター API は、以下のとおりです。
  - void **init**(Map<String, Stri
  - AuthenticationResult **proce**
    request, HttpServletRespon
    isAccessToProtectedResourc
  - AuthenticationResult
    processAuthenticationFailu
    HttpServletResponse respon
  - AuthenticationResult
    processRequestAlreadyAuthe
    request, HttpServletResponse response)
  - Map<String, Object> **getAuthenticationData**()
  - HttpServletRequest **getRequestToProceed**(HttpServletRequest request, HttpServletResponse response, UserIdentity userIdentity, LoginExtension... loginExtension)
  - Boolean **changeResponseOnSuccess** (HttpServletRequest request, HttpServletResponse response)
  - WorkLightAuthenticator **clone**()

`getRequestToProceed()` メソッドは、オーセンティケーターによって収集された資格情報をログイン・モジュールが検証して、検証が成功した後にのみ、呼び出されます。

`getRequestToProceed()` メソッドは、IBM Worklight V5.0.0.3 以降は**非推奨**となっています。

カスタム Java オーセンティケーターの作成 (7/21)

- オーセンティケーター API は、以下のとおりです。
  - void **init**(Map<String, Stri
  - AuthenticationResult **proce**
    request, HttpServletRespon
    isAccessToProtectedResourc
  - AuthenticationResult
    processAuthenticationFailu
    HttpServletResponse respon
  - AuthenticationResult
    processRequestAlreadyAuthe
    request, HttpServletResponse response)
  - Map<String, Object> **getAuthenticationData**()
  - HttpServletRequest **getRequestToProceed**(HttpServletRequest request, HttpServletResponse response, UserIdentity userIdentity, LoginExtension... loginExtension)
  - Boolean **changeResponseOnSuccess** (HttpServletRequest request, HttpServletResponse response)
  - WorkLightAuthenticator **clone**()

changeResponseOnSuccess() メソッドは、認証が成功した後に呼び出されます。認証が成功したら、このメソッドを使用して応答にデータを追加します。

カスタム Java オーセンティケーターの作成 (8/21)

- オーセンティケーター API は、以下のとおりです。
  - void **init**(Map<String, Stri
  - AuthenticationResult **proce**
    request, HttpServletRespon
    isAccessToProtectedResourc
  - AuthenticationResult
    processAuthenticationFailu
    HttpServletResponse respon
  - AuthenticationResult
    processRequestAlreadyAuthe
    request, HttpServletResponse response)
  - Map<String, Object> **getAuthenticationData**()
  - HttpServletRequest **getRequestToProceed**(HttpServletRequest request, HttpServletResponse response, UserIdentity userIdentity, LoginExtension... loginExtension)
  - Boolean **changeResponseOnSuccess** (HttpServletRequest request, HttpServletResponse response)
  - WorkLightAuthenticator **clone**()

clone() メソッドは、クラス・メンバーのディープ・コピーを作成するために使用します。

カスタム Java オーセンティケーターの作成 (9/21)

- **MyCustomAuthenticator** クラスを **server¥java** フォルダー内に作成します。
- 必ずこのクラスで **WorkLightAuthenticator** インターフェースを実装するようにします。

```
public class MyCustomAuthenticator implements WorkLightAuthenticator {
```

- 資格情報を保持するために、**authenticationData** マップをオーセンティケーターに追加します。
  - このオブジェクトは、ログイン・モジュールによって取得および使用されます。

```
private Map<String, Object> authenticationData = null;
```

カスタム Java オーセンティケーターの作成 (10/21)

- サーバー関連のクラス (例えば、**HttpServletRequest** など) を使用するには、サーバー・ランタイム・ライブラリーへの依存関係を追加する必要があります。
- Worklight プロジェクトを右クリックし、**「プロパティー (Properties)」**を選択します。
- **「Java ビルド・パス (Java Build Path)」**→**「ライブラリー (Libraries)」**を選択し、**「ライブラリーの追加 (Add Library)」**をクリックします。
- **「サーバー・ランタイム (Server Runtime)」**を選択し、**「次へ (Next)」**をクリックします。
- ご使用の Eclipse にインストールされているサーバー・ランタイムのリストが表示されます。
- いずれかを選択し、**「終了 (Finish)」**をクリックします。
- **「OK」**をクリックします。

カスタム Java オーセンティケーターの作成 (11/21)

- `init()` メソッドは、オーセンティケーターの作成時に呼び出されます。
- これは、**authenticationConfig.xml** のレルム定義で指定されているオプションのマップを受け取ります。

```
@Override
public void init(Map<String, String> options) throws MissingConfigurationOptionException {
    logger.info("init");
}
```

- オーセンティケーターの `clone()` メソッドは、オブジェクト・メンバーのディープ・コピーを作成します。

```
@Override
public WorkLightAuthenticator clone() throws CloneNotSupportedException {
    MyCustomAuthenticator otherAuthenticator = (MyCustomAuthenticator) super.clone();
    otherAuthenticator.authenticationData = new HashMap<String, Object>(authenticationData);
    return otherAuthenticator;
}
```

カスタム Java オーセンティケーターの作成 (12/21)

- processRequest() メソッドは、資格情報を収集するための非認証要求があるたびに呼び出されます。

```
@Override
public AuthenticationResult processRequest(HttpServletRequest request, HttpServletResponse response, boolean isAccessToProtected
    logger.info("MyCustomAuthenticator :: proces
    if (request.getRequestURI().contains("my_cus
        String username = request.getParameter("use
        String password = request.getParameter("

        if (null != username && null != password
            authenticationData = new HashMap<Str
            authenticationData.put("username", u
            authenticationData.put("password", p
            return AuthenticationResult.createFr
        } else {
            response.setContentType("application
            response.setHeader("Cache-Control",
            response.getWriter().print("{\"authS
            return AuthenticationResult.createFr
        }
    }

    if (!isAccessToProtectedResource)
        return AuthenticationResult.createFrom(A

    response.setContentType("application/json; c
    response.setHeader("Cache-Control", "no-cach
    response.getWriter().print("{\"authStatus\":
    return AuthenticationResult.createFrom(Authe
}
```

processRequest() メソッドは、request、response、および isAccessToProtectedResource の各引数を受け取ります。このメソッドは、要求からデータを取得したり、応答にデータを書き込んだりする場合もありますが、以降のスライドで説明するように、特定の AuthenticationResult 状況を返す必要があります。注意: オーセンティケーターは、ログイン・モジュールで使用する資格情報を収集しますが、資格情報の検証は行いません。

カスタム Java オーセンティケーターの作成 (13/21)

- `processRequest()` メソッドは、資格情報を収集するための非認証要求があるたびに呼び出されます。

```
@Override
public AuthenticationResult processRequest(HttpServletRequest request, HttpServletResponse response, boolean isAccessToProtectedR
    logger.info("MyCustomAuthenticator :: processRequest");
    if (request.getRequestURI().contains("my_custom_auth_request_url")){
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        if (null != username && null != password && username.length() > 0 && password.length() > 0){
            authenticationData = new HashMap<String, Object>();
            authenticationData.put("username", username);
            authenticationData.put("password", password);
            return AuthenticationResult.createFrom(Authentica
        } else {
            response.setContentType("application/json; charse
            response.setHeader("Cache-Control", "no-cache, mu
            response.getWriter().print("{\"authStatus\":\"req                                                    }");
            return AuthenticationResult.createFrom(Authentica
        }
    }

    if (!isAccessToProtectedResource)
        return AuthenticationResult.createFrom(Authentication

    response.setContentType("application/json; charset=UTF-8"
    response.setHeader("Cache-Control", "no-cache, must-reval
    response.getWriter().print("{\"authStatus\":\"required\"}
    return AuthenticationResult.createFrom(AuthenticationStat
}
```

アプリケーションが認証要求を特定の URL に送信します。この要求の URL には **my_custom_auth_request_url** というコンポーネントが含まれています。このコンポーネントは、この要求が確実に認証要求となるようにするために、オーセンティケーターが使用します。すべてのオーセンティケーターでそれぞれ異なる URL コンポーネントを使用することが推奨されています。

カスタム Java オーセンティケーターの作成 (14/21)

- `processRequest()` メソッドは、資格情報を収集するための非認証要求があるたびに呼び出されます。

```
@Override
public AuthenticationResult processRequest(HttpServletRequest request, HttpServletResponse response, boolean isAccessToProtectedR
    logger.info("MyCustomAuthenticator :: processRequest");
    if (request.getRequestURI().contains("my_custom_auth_request_url")){
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        if (null != username && null != password && username.length() > 0 && password.length() > 0){
            authenticationData = new HashMap<String, Object>();
            authenticationData.put("username", username);
            authenticationData.put("password", password);
            return AuthenticationResult.createFrom(AuthenticationStatus.SUCCESS);
        } else {
            response.setContentType("application/json; charset=UTF-8"
            response.setHeader("Cache-Control", "no-cache, must-reval
            response.getWriter().print("{\"authStatus\":\"required\",
            return AuthenticationResult.createFrom(AuthenticationStat
        }
    }

    if (!isAccessToProtectedResource)
        return AuthenticationResult.createFrom(AuthenticationStatus.

    response.setContentType("application/json; charset=UTF-8");
    response.setHeader("Cache-Control", "no-cache, must-revalidate");
    response.getWriter().print("{\"authStatus\":\"required\"}");
    return AuthenticationResult.createFrom(AuthenticationStatus.CLIEN
}
```

ssword\"}");

オーセンティケーターは、要求パラメーターとして渡されるユーザー名とパスワード資格情報を取得します。

カスタム Java オーセンティケーターの作成 (15/21)

- `processRequest()` メソッドは、資格情報を収集するための非認証要求があるたびに呼び出されます。

```
@Override
public AuthenticationResult processRequest(HttpServletRequest request, HttpServletResponse response, boolean isAccessToProtectedR
    logger.info("MyCustomAuthenticator :: processRequest");
    if (request.getRequestURI().contains("my_custom_auth_request_url")){
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        if (null != username && null != password && username.length() > 0 && password.length() > 0){
            authenticationData = new HashMap<String, Object>();
            authenticationData.put("username", username);
            authenticationData.put("password", password);
            return AuthenticationResult.createFrom(AuthenticationStatus.SUCCESS);
        } else {
            response.setContentType("application/json; charset=UTF-8");
            response.setHeader("Cache-Control", "no-cache, must
            response.getWriter().print("{\"authStatus\":\"requi
            return AuthenticationResult.createFrom(Authenticati
        }
    }

    if (!isAccessToProtectedResource)
        return AuthenticationResult.createFrom(AuthenticationSt

    response.setContentType("application/json; charset=UTF-8");
    response.setHeader("Cache-Control", "no-cache, must-revali
    response.getWriter().print("{\"authStatus\":\"required\"}");
    return AuthenticationResult.createFrom(AuthenticationStatus
}
```

"}");

オーセンティケーターは資格情報に基本的な妥当性があるか検査し、`authenticationData` オブジェクトを作成して、SUCCESS を返します。SUCCESS が意味するのは、資格情報の収集に成功したことのみです。その後で、資格情報を検証するためにログイン・モジュールが呼び出されます。

カスタム Java オーセンティケーターの作成 (16/21)

- `processRequest()` メソッドは、資格情報を収集するための非認証要求があるたびに呼び出されます。

```
@Override
public AuthenticationResult processRequest(HttpServ
    logger.info("MyCustomAuthenticator :: processRe
    if (request.getRequestURI().contains("my_custom
        String username = request.getParameter("use
        String password = request.getParameter("pas

        if (null != username && null != password &&
            authenticationData = new HashMap<String
            authenticationData.put("username", user
            authenticationData.put("password", pass
            return AuthenticationResult.createFrom(
        } else {
            response.setContentType("application/json; charset=UTF-8");
            response.setHeader("Cache-Control", "no-cache, must-revalidate");
            response.getWriter().print("{\"authStatus\":\"required\", \"errorMessage\":\"Please enter username and password\"}");
            return AuthenticationResult.createFrom(AuthenticationStatus.CLIENT_INTERACTION_REQUIRED);
        }
    }

    if (!isAccessToProtectedResource)
        return AuthenticationResult.createFrom(AuthenticationStatus.REQUEST_NOT_RECOGNIZED);

    response.setContentType("application/json; charset=UTF-8");
    response.setHeader("Cache-Control", "no-cache, must-revalidate");
    response.getWriter().print("{\"authStatus\":\"required\"}");
    return AuthenticationResult.createFrom(AuthenticationStatus.CLIENT_INTERACTION_REQUIRED);
}
```

受け取った資格情報に問題があった場合、オーセンティケーターは errorMessage を応答に追加し、CLIENT_INTERACTION_REQUIRED を返します。この場合でも、クライアントは認証データを提供する必要があります。

カスタム Java オーセンティケーターの作成 (17/21)

- `processRequest()` メソッドは、資格情報を収集するための非認証要求があるたびに呼び出されます。

```
@Override
public AuthenticationResult processRequest(HttpServletRequest re
    logger.info("MyCustomAuthenticator :: processRequest");
    if (request.getRequestURI().contains("my_custom_auth_request
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        if (null != username && null != password && username.len
            authenticationData = new HashMap<String, Object>();
            authenticationData.put("username", username);
            authenticationData.put("password", password);
            return AuthenticationResult.createFrom(Authenticatio
        } else {
            response.setContentType("application/json; charset=U
            response.setHeader("Cache-Control", "no-cache, must-
            response.getWriter().print("{\"authStatus\":\"requir
            return AuthenticationResult.createFrom(Authenticatio
        }
    }

    if (!isAccessToProtectedResource)
        return AuthenticationResult.createFrom(AuthenticationStatus.REQUEST_NOT_RECOGNIZED);

    response.setContentType("application/json; charset=UTF-8");
    response.setHeader("Cache-Control", "no-cache, must-revalidate");
    response.getWriter().print("{\"authStatus\":\"required\"}");
    return AuthenticationResult.createFrom(AuthenticationStatus.CLIENT_INTERACTION_REQUIRED);
}
```

`isAccessToProtectedResource` 引数は、保護リソースへのアクセスが試行されたかどうかを示します。試行されていない場合、このメソッドは REQUEST_NOT_RECOGNIZED を返します。この戻り値はオーセンティケーターによる処理が不要であることを意味しており、メソッドはそのまま要求の処理を続行します。

カスタム Java オーセンティケーターの作成 (18/21)

- `processRequest()` メソッドは、資格情報を収集するための非認証要求があるたびに呼び出されます。

```
@Override
public AuthenticationResult processRequest(HttpServletRequest request, HttpServletResponse response, boolean isAccessToProtectedR
    logger.info("MyCustomAuthenticator :: processRequest");
    if (request.getRequestURI().contains("my_custom_auth_request_url")){
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        if (null != username && null != password && username.leng
            authenticationData = new HashMap<String, Object>();
            authenticationData.put("username", username);
            authenticationData.put("password", password);
            return AuthenticationResult.createFrom(Authentication
        } else {
            response.setContentType("application/json; charset=UT
            response.setHeader("Cache-Control", "no-cache, must-r
            response.getWriter().print("{\"authStatus\":\"require
            return AuthenticationResult.createFrom(Authentication
        }
    }

    if (!isAccessToProtectedResource)
        return AuthenticationResult.createFrom(AuthenticationStat

    response.setContentType("application/json; charset=UTF-8");
    response.setHeader("Cache-Control", "no-cache, must-revalidate");
    response.getWriter().print("{\"authStatus\":\"required\"}");
    return AuthenticationResult.createFrom(AuthenticationStatus.CLIENT_INTERACTION_REQUIRED);
}
```

保護リソースに対して行われた要求に認証データが含まれていない場合、オーセンティケーターは `authStatus:required` プロパティーを応答に追加し、さらに CLIENT_INTERACTION_REQUIRED 状況を返します。

カスタム Java オーセンティケーターの作成 (19/21)

- オーセンティケーターの `getAuthenticationData()` メソッドは、収集された資格情報を取得するために、ログイン・モジュールで使用します。

```
@Override
public Map<String, Object> getAuthenticationData() {
    logger.info("getAuthenticationData");
    return authenticationData;
}
```

- 認証済みセッションが確立されると、それ以後はすべての要求が **`changeResponseOnSuccess()`** メソッドと **`processRequestAlreadyAuthenticated()`** メソッドを介してトランスポートされます。
- これらのメソッドを使用すると、要求からデータを取得し、応答を更新することができます。

カスタム Java オーセンティケーターの作成 (20/21)

- `changeResponseOnSuccess()` メソッドは、ログイン・モジュールによる資格情報の検証が成功した後に呼び出されます。
- このメソッドを使用して、応答をクライアントに返す前に応答を変更することができます。
- このメソッドは、応答が変更された場合は true、変更されていない場合は false を返す必要があります。
- このメソッドを使用して、認証が成功したことをクライアント・アプリケーションに通知します。

```
@Override
public boolean changeResponseOnSuccess(HttpServletRequest request, HttpServletResponse response) throws IOException {
    Logger.info("MyCustomAuthenticator :: changeResponseOnSuccess");
    if (request.getRequestURI().contains("my_custom_auth_request_url")){
        response.setContentType("application/json; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache, must-revalidate");
        response.getWriter().print("{\"authStatus\":\"complete\"}");
        return true;
    }
    return false;
}
```

カスタム Java オーセンティケーターの作成 (21/21)

- `processRequestAlreadyAuthenticated()` メソッドは、認証された要求に対して AuthenticationResult を返します。

```
@Override
public AuthenticationResult processRequestAlreadyAuthenticated(HttpServletRequest request,
    logger.info("processRequestAlreadyAuthenticated");
    return AuthenticationResult.REQUEST_NOT_RECOGNIZED;
}
```

- ログイン・モジュールが認証障害を返した場合には、`processAuthenticationFailure()` が呼び出されます。このメソッドは、エラー・メッセージを応答の本文に書き込み、CLIENT_INTERACTION_REQUIRED 状況を返します。

```
@Override
public AuthenticationResult processAuthenticationFailure(HttpServletRequest request, HttpServletResponse response,
        String errorMessage) throws IOException, ServletException {

    logger.info("processAuthenticationFailure");
    response.setContentType("application/json; charset=UTF-8");
    response.setHeader("Cache-Control", "no-cache, must-revalidate");
    response.getWriter().print("{\"authRequired\":true, \"errorMessage\":\"" + errorMessage + "\"}");
    return AuthenticationResult.CLIENT_INTERACTION_REQUIRED;
}
```

アジェンダ

- 認証の概要
- authenticationConfig.xml の構成
- カスタム Java オーセンティケーターの作成
- カスタム Java ログイン・モジュールの作成
- 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

カスタム Java ログイン・モジュールの作成 (1/20)

- ログイン・モジュール API は、以下のとおりです。
  - `void **init**(Map<String, String> options)`
  - `boolean **login**(Map<String, Object> authenticationData)`
  - `UserIdentity **createIdentity**(String loginModule)`
  - `void **logout**()`
  - `void **abort**()`
  - `WorkLightAuthLoginModu`

ログイン・モジュールの `init()` メソッドは、ログイン・モジュール・インスタンスの作成時に呼び出されます。このメソッドは、**authenticationConfig.xml** ファイルのログイン・モジュール定義で指定されているオプションを受け取ります。

カスタム Java ログイン・モジュールの作成 (2/20)

- ログイン・モジュール API は、以下のとおりです。
  - `void **init**(Map<String, String> options)`
  - `boolean **login**(Map<String, Object> authenticationData)`
  - `UserIdentity **createIdentity**(String loginModule)`
  - `void **logout**()`
  - `void **abort**()`
  - `WorkLightAuthLoginModu`

ログイン・モジュールの `login()` メソッドは、オーセンティケーターによって収集された資格情報を検証するために使用されます。

カスタム Java ログイン・モジュールの作成 (3/20)

- ログイン・モジュール API は、以下のとおりです。
  - `void init(Map<String, String> options)`
  - `boolean login(Map<String, Object> authenticationData)`
  - `UserIdentity createIdentity(String loginModule)`
  - `void logout()`
  - `void abort()`
  - `WorkLightAuthLoginModu`

ログイン・モジュールの `createIdentity()` メソッドは、資格情報の検証が成功した後で `userIdentity` オブジェクトを作成するために使用されます。

カスタム Java ログイン・モジュールの作成 (4/20)

- ログイン・モジュール API は、以下のとおりです。
  - `void init(Map<String, String> options)`
  - `boolean login(Map<String, Object> authenticationData)`
  - `UserIdentity createIdentity(String loginModule)`
  - `void logout()`
  - `void abort()`
  - `WorkLightAuthLoginModu`

`logout()` メソッドと `abort()` メソッドは、ログアウトまたは認証の異常終了が発生した後に、キャッシュ・データをクリーンアップするために使用されます。

カスタム Java ログイン・モジュールの作成 (5/20)

- ログイン・モジュール API は、
  - `void init(Map<String, `
  - `boolean login(Map<Stri authenticationData)`
  - `UserIdentity createIde`
  - `void logout()`
  - `void abort()`
  - `WorkLightLoginModule clone()`

`clone()` メソッドは、クラス・メンバーのディープ・コピーを作成するために使用されます。

カスタム Java ログイン・モジュールの作成 (6/20)

- **MyCustomLoginModule** クラスを **server¥java** フォルダー内に作成します。
- このクラスが **WorkLightAuthLoginModule** インターフェースを実装することを確認します。

```
public class MyCustomLoginModule implements WorkLightAuthLoginModule {
```

- 資格情報を保持するために、USERNAME と PASSWORD という2つのプライベート・クラス・メンバーを追加します。

```
private String USERNAME;
private String PASSWORD;
```

カスタム Java ログイン・モジュールの作成 (7/20)

- `init()` メソッドは、ログイン・モジュール・インスタンスの作成時に呼び出されます。これは、**authenticationConfig.xml** file ファイルのログイン・モジュール定義で指定されたオプションのマップを受け取ります。

```
@Override
public void init(Map<String, String> options) throws MissingConfigurationOptionException {
    logger.info("init");
}
```

- ログイン・モジュールの `clone()` メソッドは、オブジェクト・メンバーのディープ・コピーを作成します。

```
@Override
public MyCustomLoginModule clone() throws CloneNotSupportedException {
    return (MyCustomLoginModule) super.clone();
}
```

カスタム Java ログイン・モジュールの作成 (8/20)

- `login()` メソッドは、オーセンティケーターが SUCCESS 状況を返した後に呼び出されます。

```
@Override
public boolean login(Map<String, Object> authenticationData) {
    logger.info("MyCustomLoginModule :: login");
    USERNAME = (String) authenticationData.get("username");
    PASSWORD = (String) authenticationData.get("password");

    if (USERNAME.equals("wluser") &
        return true;
    else
        throw new RuntimeException(
}
```

`login()` メソッドは、呼び出されたときに、`authenticationData` オブジェクトをオーセンティケーターから取得します。

カスタム Java ログイン・モジュールの作成 (9/20)

- `login()` メソッドは、オーセンティケーターが SUCCESS 状況を返した後に呼び出されます。

```
@Override
public boolean login(Map<String, Object> authenticationData) {
    Logger.info("MyCustomLoginModule :: login");
    USERNAME = (String) authenticationData.get("username");
    PASSWORD = (String) authenticationData.get("password");

    if (USERNAME.equals("wluser") && PASSWORD.equals("12345"))
        return true;
    else
        throw new RuntimeException("
}
```

`login()` メソッドは、オーセンティケーターが以前に格納していたユーザー名とパスワード資格情報を取得します。

カスタム Java ログイン・モジュールの作成 (10/20)

- `login()` メソッドは、オーセンティケーターが SUCCESS 状況を返した後に呼び出されます。

```
@Override
public boolean login(Map<String, Object> authenticationData) {
    logger.info("MyCustomLoginModule :: login");
    USERNAME = (String) authenticationData.get("username");
    PASSWORD = (String) authenticationData.get("password");

    if (USERNAME.equals("wluser") && PASSWORD.equals("12345"))
        return true;
    else
        throw new RuntimeException(
}
```

この例では、ログイン・モジュールが、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を検証します。独自の検証ルールを実装できます。`login()` メソッドは、資格情報が有効な場合は **true** を返します。

カスタム Java ログイン・モジュールの作成 (11/20)

- login() メソッドは、オーセンティケーターが SUCCESS 状況を返した後に呼び出されます。

```
@Override
public boolean login(Map<String, Object> authenticationData) {
    logger.info("MyCustomLoginModule :: login");
    USERNAME = (String) authenticationData.get("username");
    PASSWORD = (String) authenticationData.get("password");

    if (USERNAME.equals("wluser") && PASSWORD.equals("12345"))
        return true;
    else
        throw new RuntimeException("Invalid credentials");
}
```

資格情報の検証が失敗した場合、login() メソッドは、**false** を返すか、または、errorMessage パラメーターとしてオーセンティケーターに返されるテキストとともに RuntimeException を スローします。

カスタム Java ログイン・モジュールの作成 (12/20)

- `createIdentity()` メソッドは、login() メソッドが **true** を返したときに呼び出されます。この使用目的は、認証済みのユーザー ID オブジェクトを作成することです。

```
@Override
public UserIdentity createIdentity(String loginModule) {
    logger.info("MyCustomLoginModule :: createIdentity");

    HashMap<String, Object> customAttributes = new HashMap<String, Object>();
    customAttributes.put("AuthenticationDate", new Date());

    UserIdentity identity = new UserIdentity(loginModule, USERNAME, null, null, customAttributes, PASSWORD);
    return identity;
}
```

`login()` メソッドが **true** を返した後に、`createIdentity()` メソッドが呼び出されます。このメソッドは、`UserIdentity` オブジェクトを作成するために使用されます。この中に独自のカスタム属性を格納して、後で Java コードやアダプター・コードに使用することができます。

カスタム Java ログイン・モジュールの作成 (13/20)

- `createIdentity()` メソッドは、`login()` メソッドが **true** を返したときに呼び出されます。この使用目的は、認証済みのユーザー ID オブジェクトを作成することです。

```
@Override
public UserIdentity createIdentity(String loginModule) {
    logger.info("MyCustomLoginModule :: createIdentity");

    HashMap<String, Object> customAttributes = new HashMap<String, Object>();
    customAttributes.put("AuthenticationDate", new Date());

    UserIdentity identity = new UserIdentity(loginModule, USERNAME, null, null, customAttributes, PASSWORD);
    return identity;
}
```

`UserIdentity` オブジェクトには、ユーザー情報が含まれます。そのコンストラクターは以下のとおりです。

```
public UserIdentity(String loginModule,
                    String name,
                    String displayName,
                    Set<String> roles,
                    Map<String, Object> attributes,
                    Object credentials)
```

カスタム Java ログイン・モジュールの作成 (14/20)

- createIdentity() メソッドは、login() メソッドが **true** を返したときに呼び出されます。この使用目的は、認証済みのユーザー ID オブジェクトを作成することです。

ユーザーの設定対象となるログイン・モジュール名

```
@Override
public UserIdentity createIdentity(
    logger.info("MyCustomLoginModule :: createIden ity");

    HashMap<String, Object> customAttributes = new HashMap<String, Object>();
    customAttributes.put("AuthenticationDate", new Date());

    UserIdentity identity = new UserIdentity(loginModule, USERNAME, null, null, customAttributes, PASSWORD);
    return identity;
}
```

UserIdentity オブジェクトには、ユーザー情報が含まれます。そのコンストラクターは以下のとおりです。

```
public UserIdentity(String loginModule,
                    String name,
                    String displayName,
                    Set<String> roles,
                    Map<String, Object> attributes,
                    Object credentials)
```

カスタム Java ログイン・モジュールの作成 (15/20)

- `createIdentity()` メソッドは、`login()` メソッドが **true** を返したときに呼び出されます。この使用目的は、認証済みのユーザー ID オブジェクトを作成することです。

固有のユーザー ID

```
@Override
public UserIdentity createIdentity(String logi
    logger.info("MyCustomLoginModule :: createIdentity");

    HashMap<String, Object> customAttributes = new HashMap<S ring, Object>();
    customAttributes.put("AuthenticationDate", new Date());

    UserIdentity identity = new UserIdentity(loginModule, USERNAME, null, null, customAttributes, PASSWORD);
    return identity;
}
```

`UserIdentity` オブジェクトには、ユーザー情報が含まれます。そのコンストラクターは以下のとおりです。

```
public UserIdentity(String loginModule,
                    String name,
                    String displayName,
                    Set<String> roles,
                    Map<String, Object> attributes,
                    Object credentials)
```

カスタム Java ログイン・モジュールの作成 (16/20)

- `createIdentity()` メソッドは、`login()` メソッドが **true** を返したときに呼び出されます。この使用目的は、認証済みのユーザー ID オブジェクトを作成することです。

ユーザーの表示名

```
@Override
public UserIdentity createIdentity(String loginModule)
    logger.info("MyCustomLoginModule :: createIdentity");

    HashMap<String, Object> customAttributes = new HashMap<String, Object>();
    customAttributes.put("AuthenticationDate", new Date());

    UserIdentity identity = new UserIdentity(loginModule, USERNAME, null, null, customAttributes, PASSWORD);
    return identity;
}
```

`UserIdentity` オブジェクトには、ユーザー情報が含まれます。そのコンストラクターは以下のとおりです。

```
public UserIdentity(String loginModule,
                    String name,
                    String displayName,
                    Set<String> roles,
                    Map<String, Object> attributes,
                    Object credentials)
```

カスタム Java ログイン・モジュールの作成 (17/20)

- `createIdentity()` メソッドは、`login()` メソッドが **true** を返したときに呼び出されます。この使用目的は、認証済みのユーザー ID オブジェクトを作成することです。

ユーザーの Java セキュリティー・ロール

```
@Override
public UserIdentity createIdentity(String loginModule) {
    logger.info("MyCustomLoginModule :: createIdentity");

    HashMap<String, Object> customAttributes = new HashMap<String, Object>( );
    customAttributes.put("AuthenticationDate", new Date());

    UserIdentity identity = new UserIdentity(loginModule, USERNAME, null, null, customAttributes, PASSWORD);
    return identity;
}
```

`UserIdentity` オブジェクトには、ユーザー情報が含まれます。そのコンストラクターは以下のとおりです。

```
public UserIdentity(String loginModule,
                    String name,
                    String displayName,
                    Set<String> roles,
                    Map<String, Object> attributes,
                    Object credentials)
```

カスタム Java ログイン・モジュールの作成 (18/20)

- `createIdentity()` メソッドは、`login()` メソッドが **true** を返したときに呼び出されます。この使用目的は、認証済みのユーザー ID オブジェクトを作成することです。

```
@Override
public UserIdentity createIdentity(String loginModule) {
    logger.info("MyCustomLoginModule :: createIdentity");

    HashMap<String, Object> customAttributes = new HashMap<String, Object>();
    customAttributes.put("AuthenticationDate", new Date());

    UserIdentity identity = new UserIdentity(loginModule, USERNAME, null, null, customAttributes, PASSWORD);
    return identity;
}
```

カスタム・ユーザー属性

`UserIdentity` オブジェクトには、ユーザー情報が含まれます。そのコンストラクターは以下のとおりです。

```
public UserIdentity(String loginModule,
                    String name,
                    String displayName,
                    Set<String> roles,
                    Map<String, Object> attributes,
                    Object credentials)
```

カスタム Java ログイン・モジュールの作成 (19/20)

- `createIdentity()` メソッドは、`login()` メソッドが **true** を返したときに呼び出されます。この使用目的は、認証済みのユーザー ID オブジェクトを作成することです。

```
@Override
public UserIdentity createIdentity(String loginModule) {
    logger.info("MyCustomLoginModule :: createIdentity");

    HashMap<String, Object> customAttributes = new HashMap<String, Object>();
    customAttributes.put("AuthenticationDate", new Date());

    UserIdentity identity = new UserIdentity(loginModule, USERNAME, null, null, customAttributes, PASSWORD);
    return identity;
}
```

保持されてはならない機密性の高いユーザー資格情報

`UserIdentity` オブジェクトには、ユーザー情報が含まれます。そのコンストラクターは以下のとおりです。

```
public UserIdentity(String loginModule,
                    String name,
                    String displayName,
                    Set<String> roles,
                    Map<String, Object> attributes,
                    Object credentials)
```

カスタム Java ログイン・モジュールの作成 (20/20)

- `logout()` メソッドと `abort()` メソッドは、ユーザーがログアウトした後または認証フローを異常終了した後に、クラス・メンバーをクリーンアップするために使用します。

```
@Override
public void logout() {
    logger.info("MyCustomLoginModule :: logout");
    USERNAME = null;
    PASSWORD = null;
}

@Override
public void abort() {
    logger.info("MyCustomLoginModule :: abort");
    USERNAME = null;
    PASSWORD = null;
}
```

アジェンダ

- 認証の概要
- authenticationConfig.xml の構成
- カスタム Java オーセンティケーターの作成
- カスタム Java ログイン・モジュールの作成
- 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

クライアント・サイドの認証コンポーネントの作成 (1/13)

- 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 エレメントで構成されます。
  - `<div id="AppBody">` 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="AuthBody">` エレメントは、認証フォームのために使用します。
- 認証が要求されると、アプリケーションは AppBody を非表示にして AuthBody 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 (2/13)

- 最初に AppBody を作成します。
- これには基本構造と関数が含まれています。

```
<div id="AppBody">
    <div class="header">
        <h1>Custom Login Module</h1>
    </div>
    <div class="wrapper">
        <input type="button" value="Call protected adapter proc" onclick="getSecretData()" />
        <input type="button" value="Logout"
               onclick="WL.Client.logout('CustomAuthenticatorRealm',{onSuccess: WL.Client.reloadApp})" />
    </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。

クライアント・サイドの認証コンポーネントの作成 (3/13)

- AuthBody には、以下のエレメントが含まれています。

```
<div id="AuthBody" style="display: none">
    <div id="loginForm">
        Username:<br/>
        <input type="text" id="usernameInputField"  /><br />
        Password:<br/>
        <input type="password" id="passwordInputField" /><br/>
        <input type="button" id="loginButton" value="Login" />
        <input type="button" id="cancelButton" value="Cancel" />
    </div>
</div>
```

  - 「ユーザー名 (Username)」と「パスワード (Password)」の各入力フィールド
  - 「ログイン (Login)」ボタンと「キャンセル (Cancel)」ボタン
- AuthBody 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 (4/13)

- 以下の API は、チャレンジ・ハンドラーを作成し、その機能を実装する方法を示しています。

```
var myChallengeHandler = WL.Client.createChallengeHandler("realm-name");

myChallengeHandler.isCustomResponse = function (response){
    return false;
};

myChallengeHandler.handleChallenge = function (response){
};
```

`WL.Client.createChallengeHandler()` 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 (5/13)

- 以下の API は、チャレンジ・ハンドラーを作成し、その機能を実装する方法を示しています。

```
var myChallengeHandler = WL.Client.createChallengeHandler("realm-name");

myChallengeHandler.isCustomResponse = function (response){
    return false;
};

myChallengeHandler.handleChallenge = function (response){
};
```

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。この関数を使用して、このチャレンジ・ハンドラーに関するデータが応答に含まれているかどうかを検出します。戻り値は **true** または **false** でなければなりません。

クライアント・サイドの認証コンポーネントの作成 (6/13)

- 以下の API は、チャレンジ・ハンドラーを作成し、その機能を実装する方法を示しています。

```
var myChallengeHandler = WL.Client.createChallengeHandler("realm-name");

myChallengeHandler.isCustomResponse = function (response){
    return false;
};

myChallengeHandler.handleChallenge = function (response){
};
```

`isCustomResponse` 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。この関数を使用して、必要なアクション (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) が実行されます。

クライアント・サイドの認証コンポーネントの作成 (7/13)

- 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機能も含まれています。
  - `submitLoginForm()` は、収集した資格情報を特定の URL に送信するために使用されます。開発者は、要求パラメーター、ヘッダー、およびコールバックを指定することもできます。
  - `submitSuccess()` は、認証が正常に終了したことを 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()` は、認証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。

  これらの各関数を対応するオブジェクトに付加する必要がある点に注意してください。例: *`myChallengeHandler.submitSucces()`*
- これらの関数は、以降のスライドでチャレンジ・ハンドラーを実装する際に使用します。

クライアント・サイドの認証コンポーネントの作成 (8/13)

- チャレンジ・ハンドラーを作成します。

```
var customAuthenticatorRealmChallengeHandler = WL.Client.createChallengeHandler("CustomAuthenticatorRealm");

customAuthenticatorRealmChallengeHandler.isCustomResponse = function(response) {
    if (!response || !response.responseJSON) {
        return false;
    }

    if (response.responseJSON.authStatus)
        return true;
    else
        return false;
};

customAuthenticatorRealmChallengeHandler.handleChallenge = fu
    var authStatus = response.responseJSON.authStatus;

    if (authStatus == "required"){
        $('#AppBody').hide();
        $('#AuthBody').show();
        $('#passwordInputField').val('');
        if (response.responseJSON.errorMessage){
            alert(response.responseJSON.errorMessage);
        }
    } else if (authStatus == "complete"){
        $('#AppBody').show();
        $('#AuthBody').hide();
        customAuthenticatorRealmChallengeHandler.submitSu
    }
};
```

チャレンジ JSON に `authStatus` プロパティーが含まれている場合には、**true** を返し、そうでない場合には、**false** を返します。

クライアント・サイドの認証コンポーネントの作成 (9/13)

- チャレンジ・ハンドラーを作成します。

```
var customAuthenticatorRealmChallengeHandler =

customAuthenticatorRealmChallengeHandler.isCust
    if (!response || !response.responseJSON) {
        return false;
    }

    if (response.responseJSON.authStatus)
        return true;
    else
        return false;
};

customAuthenticatorRealmChallengeHandler.handleChallenge = function(response){
    var authStatus = response.responseJSON.authStatus;

    if (authStatus == "required"){
        $('#AppBody').hide();
        $('#AuthBody').show();
        $('#passwordInputField').val('');
        if (response.responseJSON.errorMessage){
            alert(response.responseJSON.errorMessage);
        }
    } else if (authStatus == "complete"){
        $('#AppBody').show();
        $('#AuthBody').hide();
        customAuthenticatorRealmChallengeHandler.submitSuccess();
    }
};
```

`authStatus` プロパティーの値が「required」である場合には、ログイン・フォームを表示して、パスワードの入力フィールドをクリーンアップし、さらにエラー・メッセージがあればそれを表示します。

クライアント・サイドの認証コンポーネントの作成 (10/13)

- チャレンジ・ハンドラーを作成します。

```
var customAuthenticatorRealmChallengeHandler = WL.Client.createChallengeHandler("CustomAuthenticatorRealm");

customAuthenticatorRealmChallengeHandler.isCust
    if (!response || !response.responseJSON) {
        return false;
    }

    if (response.responseJSON.authStatus)
        return true;
    else
        return false;
};

customAuthenticatorRealmChallengeHandler.handl
    var authStatus = response.responseJSON.auth

    if (authStatus == "required"){
        $('#AppBody').hide();
        $('#AuthBody').show();
        $('#passwordInputField').val('');
        if (response.responseJSON.errorMessage){
            alert(response.responseJSON.errorMessage);
        }
    } else if (authStatus == "complete"){
        $('#AppBody').show();
        $('#AuthBody').hide();
        customAuthenticatorRealmChallengeHandler.submitSuccess();
    }
};
```

authStatus の値が「complete」である場合には、ログイン画面を非表示にして、アプリケーションに戻り、認証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。

クライアント・サイドの認証コンポーネントの作成 (11/13)

- チャレンジ・ハンドラーを作成します。

```
$('#loginButton').bind('click', function () {
    var reqURL = '/my_custom_auth_request_url';
    var options = {};
    options.parameters = {
        username : $('#usernameInputField').val(),
        password : $('#passwordInputField').val()
    };
    options.headers = {};
    customAuthenticatorRealmChallengeHandler.submitLoginForm(reqURL, options,
            customAuthenticatorRealmChallengeHandler.submitLoginFormCallback);
});

$('#cancelButton').bind('click', funct
    $('#AppBody').show();
    $('#AuthBody').hide();
    customAuthenticatorRealmChallengeH
});
```

「ログイン (login)」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、サーバーに送信する関数がトリガーされます。ここで要求ヘッダーを設定し、コールバック関数を指定できます。

クライアント・サイドの認証コンポーネントの作成 (12/13)

- チャレンジ・ハンドラーを作成します。

```
$('#loginButton').bind('click', function
    var reqURL = '/my_custom_auth_request
    var options = {};
    options.parameters = {
        username : $('#usernameInputFiel
        password : $('#passwordInputFiel
    };
    options.headers = {};
    customAuthenticatorRealmChallengeHandl
            customAuthenticatorRealmChallengeHandler.submitLoginFormCallback);
});

$('#cancelButton').bind('click', function () {
    $('#AppBody').show();
    $('#AuthBody').hide();
    customAuthenticatorRealmChallengeHandler.submitFailure();
});
```

「キャンセル (cancel)」ボタンをクリックすると、authBody が非表示になり、appBody が表示され、認証に失敗したことが 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 (13/13)

- チャレンジ・ハンドラーを作成します。

```
customAuthenticatorRealmChallengeHandler.submitLoginFormCallback = function(response) {
    var isLoginFormResponse = customAuthenticatorRealmChallengeHandler.isCustomResponse(response);
    if (isLoginFormResponse){
        customAuthenticatorRealmChallengeHandler.handleChallenge(response);
    }
};
```

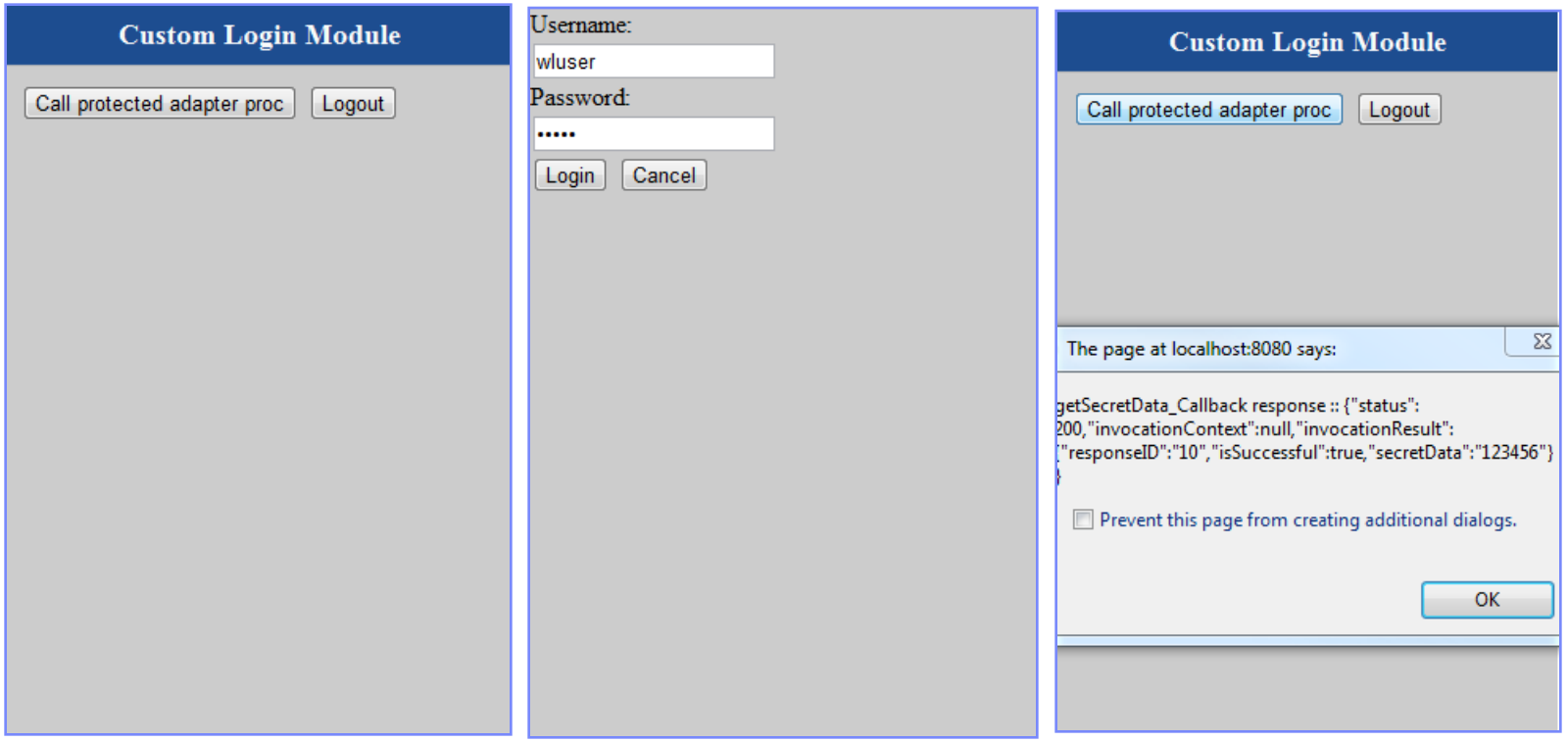
コールバック関数は、含まれているサーバー・チャレンジの応答を再び検査します。チャレンジが検出された場合は、`handleChallenge()` 関数が再び呼び出されます。

アジェンダ

- 認証の概要
- authenticationConfig.xml の構成
- カスタム Java オーセンティケーターの作成
- カスタム Java ログイン・モジュールの作成
- 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

結果の確認

- このトレーニング・モジュールのサンプルは、IBM Worklight 文書 Web サイト (http://www.ibm.com/mobile-docs) の「入門」ページにあります。
- *wluser と 12345* をユーザー資格情報として入力してください。



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