目次

改訂情報

変更年月日	変更内容
2012-12-21	初版
2014-04-01	第2版 下記を追加・変更しました ■ 「*概要* 」-「基本認証フローについて 」の基本認証フロー図を更新 ■ 「*概要* 」-「処理対象のテナントについて 」を追加 ■ 「*認証プロバイダ・プラグイン仕様* 」-「認証プロバイダ・プラグイン一覧 」に「対象のテナント」を追加 ■ アプリケーションサーバ側のルートを示すパスの表記を「%CONTEXT_PATH%」に統一しました。
2014-08-01	第3版 下記を追加・変更しました ■ 「*二重ログイン防止機能* 」を追加 ■ 「*SSO（シングルサインオン）対応* 」の SSOユーザコードプロバイダプラグイン が 存在しないユーザコードを返却した場合の仕様を追加
2014-09-01	第4版 下記を追加・変更しました ■ 「*概要* 」「*認証プロバイダ・プラグイン仕様* 」の内容を「 認証仕様書 」に移動しました。
2015-12-01	第5版 下記を変更しました ■ 「*SSO（シングルサインオン）対応* 」に、サンプルの動作確認方法を追記 ■ 「*二重ログイン防止機能* 」の内容に、強制ログイン用認証リスナに関する情報を追記
2017-12-01	第6版 下記を変更しました ■ 「*ログインポートレット対応* 」を追加
2018-04-01	第7版 下記を変更しました ■ 「*多要素認証機能* 」を追加

はじめに

本書の目的

本書では 認証機能 を利用したアプリケーションを開発する場合の、基本的な方法や注意点などについて説明します。

対象読者

本書では次の開発者を対象としています。

- 認証機能 を利用して開発を行いたい。

次の内容を理解していることが必須です。

- JavaEE を利用したWebアプリケーションの基礎
- 認証機能について

  認証機能についての詳細は、「認証仕様書」 を参照してください。

本書の構成

- *概要*
  認証機能によるログインシーケンスと認証プロバイダ・プラグインを利用した拡張ポイントについて説明します。
- *認証プロバイダ・プラグイン仕様*
  認証プロバイダ・プラグインのインタフェースと実装方法について説明します。
- *認証プロバイダ・プラグインの開発*
  認証機能および認証プロバイダ・プラグインを利用して、認証方式を変更する手順と変更例について説明します。
- *その他の認証拡張機能*
  認証機能のその他の拡張ポイントについて説明します。

概要

このドキュメントでは、認証機能を利用した具体的な開発方法について説明します。
認証機能の詳細については、「 認証仕様書 」を参照してください。

「 *認証プロバイダ・プラグインの開発* 」 では、ログイン処理を拡張するために「 認証プロバイダ・プラグイン 」を作成する手順について説明します。
「 認証プロバイダ・プラグイン 」については、「 認証仕様書 」 の以下の章を参照してください。

- 一般ユーザのログイン処理の流れについて

  「 一般ユーザのログイン処理フロー 」

- 認証プロバイダ・プラグインについて

  「 認証プロバイダ・プラグインについて 」

認証プロバイダ・プラグイン仕様

各認証プロバイダ・プラグインの仕様については、「 認証仕様書 」 の以下の章を参照してください。

- 「 一般ユーザの認証プロバイダ・プラグインの詳細 」
- 「 システム管理者の認証プロバイダ・プラグインの詳細 」

認証プロバイダ・プラグインの開発

開発プロセス

この章では、認証プロバイダ・プラグインを利用して新しい認証方式に対応するための手順について説明します。

依存関係の設定

認証プロバイダ・プラグインを利用した実装を行うためには、モジュールプロジェクト直下に保存されている ＜module.xml＞ の「依存関係」に、以下のモジュールを追加します。

モジュールID	jp.co.intra_mart.im_certification
バージョン	8.0.7

開発の流れ

認証プロバイダ・プラグインを利用して新しい認証方式に対応するための手順は以下のような流れです。

1. ビジネスロジックの範囲を決定する
2. ビジネスロジックで必要となる認証プロバイダ・プラグインを決定する
3. 認証プロバイダ・プラグインを実装する
4. 認証プロバイダ・プラグインを登録する

次項より、各プロセスの概要について説明します。

ビジネスロジックの範囲を決定する

要件を定義し、必要な認証方式を決定します。

ビジネスロジックで必要となる認証プロバイダ・プラグインを決定する

認証フローの検討

認証方式が決定したら、認証方式を実現するために必要なプラグインを選定し、それぞれの役割を定義します。

ログイン処理における認証フローを確認しながら、決定した認証方式が実現可能かを検証します。
ログイン処理における認証フローについては、「 認証仕様書 」 - 「 ログイン処理フローについて 」 を参照してください。

プラグインの概要定義

実現方法が決まったら、必要なプロバイダを選択します。

各プロバイダのインタフェースは、「 認証仕様書 」 - 「 一般ユーザの認証プロバイダ・プラグインの詳細 」 で説明しています。

実装サンプルを参考にしながら、各プロバイダの役割を定義してください。

認証機能で定義されている認証プロバイダ・プラグインは、以下の通りです。
必要なプロバイダとその役割や処理概要を記述してください。

表 認証プロバイダ・プラグイン一覧

プロバイダ名	要否	概要
初期リクエスト解析プロバイダ		
リクエスト解析プロバイダ		
認証プロバイダ		
認証条件判定プロバイダ		
認証リスナ		
ログインページプロバイダ		
遷移先ページプロバイダ		

認証プロバイダ・プラグインを実装する

仕様に基づき、それぞれの認証プロバイダ・プラグインを実装します。

「 認証仕様書 」 -「 一般ユーザの認証プロバイダ・プラグインの詳細 」 の実装サンプルと APIドキュメントを参照して必要な機能を実装してください。

要件によっては、ログイン画面を変更する必要があるかもしれません。
画面の作成については、「UIデザインガイドライン（ PC 版）」を参照してください。

認証プロバイダ・プラグインを登録する

作成したプロバイダをアプリケーションで利用できるように、プラグイン定義を行います。

プラグイン定義のうち、プログラムから利用するノードの内容については実装時に決定しているでしょう。

ここでは、プラグインIDの定義、順序定義などを行って、*plugin.xml* を完成させます。

プロバイダによっては、システムに複数のプラグインが登録されます。
仕様にそって正しい順序で動作するように設定を行います。

プラグイン定義が完成したら、実際に Web Application Server 上で動作の確認を行います。
Web Application Server 上の適切な場所に作成した資材を配置して、動作の確認を行ってください。

サンプル

概要

ここでは、実際に開発プロセスに従って認証プロバイダ・プラグインの開発を行う手順を、例を利用しながら説明します。

変更する認証方式は、 「LDAP 認証」を利用した認証方式とします。

ここで例として利用するプロバイダは、「LDAP 認証モジュール」として intra-mart Accel Platform で実際に提供されているものです。
LDAP 認証モジュールでは、LDAP サーバを認証サーバとして利用し、LDAP で管理するユーザ情報を利用して認証を行うように認証方式が変更されます。

ビジネスロジックの範囲を決定する

今回対応する範囲は、以下とします。

- 認証サーバとして LDAP サーバを利用します。
- 認証で利用する LDAP 情報は、ユーザコードとパスワードのみとします。

実際に LDAP 認証を利用するためには以下の機能が必要ですが、認証プロバイダ・プラグインの説明の範囲外のため、ここでは扱いません。

- LDAP にアカウント情報を登録する。
- intra-mart Accel Platform のアカウント情報と LDAP のユーザ情報を同期する。

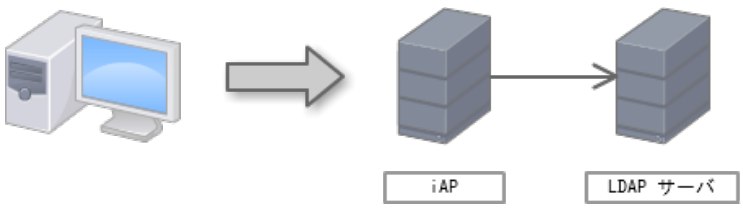

図 LDAP 構成

ビジネスロジックで必要となる認証プロバイダ・プラグインを決定する

ユーザコード、パスワードのみを扱うため、ログイン画面やリクエスト解析処理、画面遷移処理に変更はありません。
標準で提供されるプロバイダをそのまま利用可能です。

よって、認証を実行する「認証プロバイダ」のみ必要です。
また、LDAP の利用可否を検証するため、「認証条件判定プロバイダ」も作成します。

以上より、作成が必要なプロバイダは以下の通りです。

表 認証プロバイダ・プラグイン要否

プロバイダ名	要否	概要
初期リクエスト解析プロバイダ	×	
リクエスト解析プロバイダ	×	
認証プロバイダ	○	LDAP 認証サーバに認証を依頼します。
認証条件判定プロバイダ	○	LDAP 認証サーバが有効かどうかを判定します。
認証リスナ	×	
ログインページプロバイダ	×	
遷移先ページプロバイダ	×	

これにより、今回作成する認証プロバイダを利用した認証フローは、「*図 LDAP を利用した認証フロー*」です。

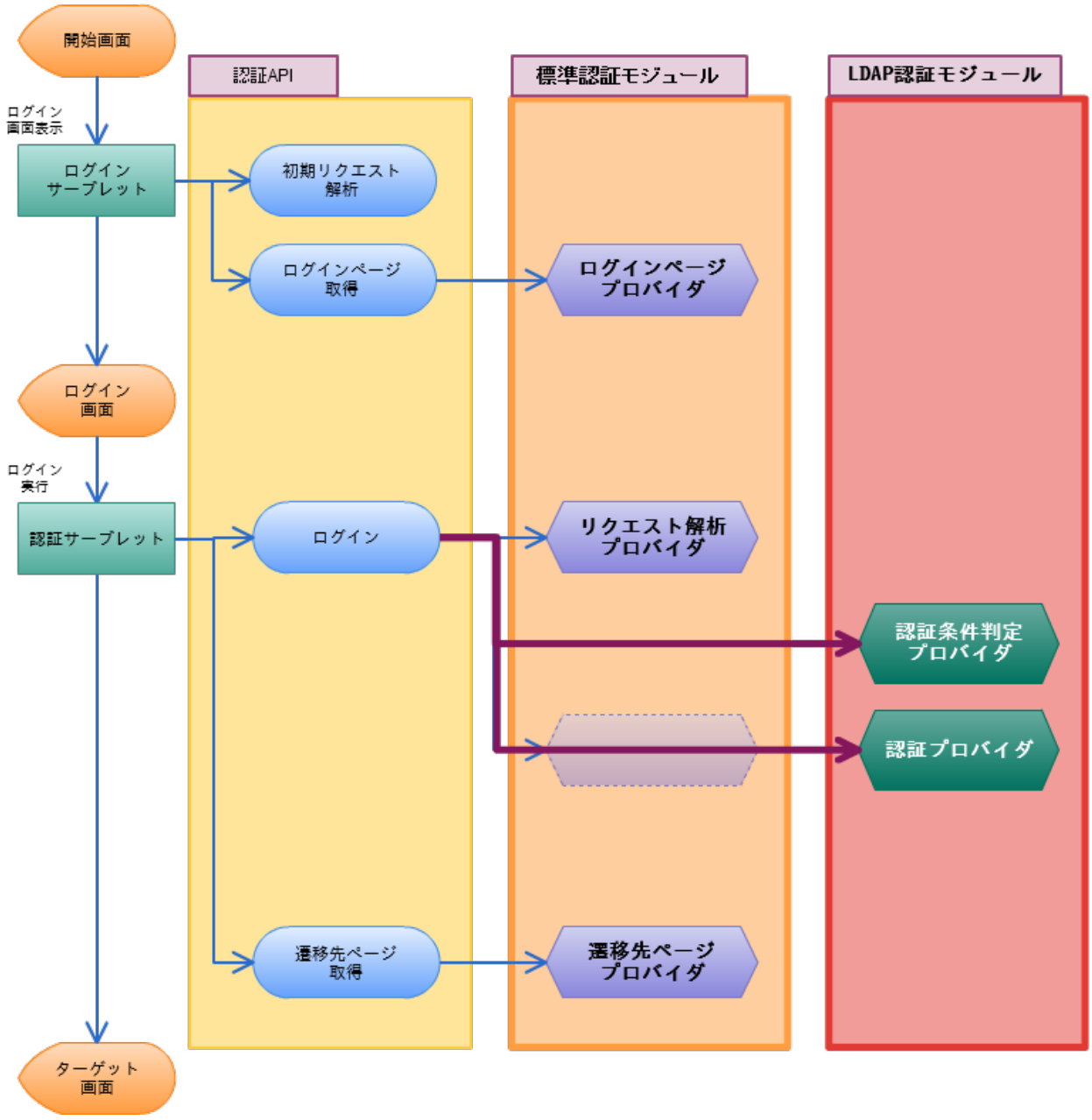

図 LDAP を利用した認証フロー

提供されないプロバイダは、標準認証モジュールのプロバイダを利用します。

認証プロバイダは、標準認証モジュールのものは利用されず、LDAP 認証モジュール で提供されるプロバイダを利用します。

認証プロバイダ・プラグインを実装する

仕様が決まりましたので、それぞれの認証プロバイダ・プラグインを実装します。

LDAP を利用するために、LDAP サーバの情報にアクセスするための API が必要となってきますが、認証プロバイダ・プラグインが扱う範囲ではありませんので、以下が既に作成済みとします。

表 LDAP アクセス関連API

APIクラス・インタフェース	概要
jp.co.intra_mart.system.security.certification.provider.ldap.LDAPContextInfoModel	LDAP 認証を行うための設定情報を保持します。

jp.co.intra_mart.system.security.certification.provider.ldap.LDAPServerInfoModel	LDAP サーバの接続情報を保持します。
jp.co.intra_mart.system.security.certification.provider.ldap.LDAPContextInfoBuilder	LDAP 設定情報を読み込みます。

このAPIでは、以下のような情報が取得可能です。

- LDAP 機能有効／無効フラグ
- LDAP サーバー覧
- LDAP サーバDN情報
- LDAP 検索情報

LDAP 認証モジュール では、これらの情報が以下の設定ファイルに記述されています。

```
%CONTEXT_PATH%/WEB-INF/conf/ldap-certification-config.xml
```

作成する認証プロバイダ・プラグイン一覧

以下の プロバイダクラスを作成します。

表 LDAP 認証モジュール 認証プロバイダー覧

プロバイダ名	実装クラス
認証条件判定プロバイダ	*jp.co.intra_mart.system.security.certification.provider.ldap.LDAPUserCertificationValidation*
認証プロバイダ	*jp.co.intra_mart.system.security.certification.provider.ldap.LDAPUserCertification*

認証条件判定プロバイダ

LDAP 認証サーバが有効かどうかを判定する処理を実装します。

以下に LDAP 認証モジュール で提供される認証条件判定プロバイダのソースの一部を例示します。

```
package jp.co.intra_mart.system.security.certification.provider.ldap;

/**
 * 一般ユーザ用 LDAP 認証条件判定プロバイダ<BR>
 */
public class LDAPUserCertificationValidation implements UserCertificationValidation {

    /** LDAP 認証情報のインスタンスを格納します。 */
    private LDAPContextInfoModel contextInfo;

    /**
     * コンストラクター
     */
    public LDAPUserCertificationValidation() {

        // LDAP 認証情報モデルを作成します。
        this.contextInfo = LDAPContextInfoBuilder.buildLDAPContextInfo(null);
    }

    /**
     * LDAP 認証サーバの利用可否を判定します。
     */
    public boolean validate(LoginInfo loginInfo, AccountInfo user, HttpServletRequest request,
HttpServletResponse response) {
        // LDAP 認証が利用可能かどうかを検証し、結果を返します。
        return this.contextInfo.isEnable();
    }

}
```

環境情報などを参照して、認証モジュールの利用可否を判定してください。

認証プロバイダ

LDAP サーバに認証を問い合わせる処理を実装します。

認証サーバへの問い合わせは、JNDI API を利用して行うことができます。
詳しい情報については、JDK の API ドキュメントを参照してください。

```
javax.naming.directory.InitialDirContext.InitialDirContext
```

以下に LDAP 認証モジュール で提供される認証プロバイダのソースの一部を例示します。

```
1  package jp.co.intra_mart.system.security.certification.provider.ldap;
2
3  /**
4   * 一般ユーザ用 LDAP 認証プロバイダ<BR>
5   */
6  public class LDAPUserCertification implements UserCertification, XmlInitParamable {
7
8     /** LDAP 認証情報のインスタンスを格納します。 */
9     private LDAPContextInfoModel contextInfo;
10
11    /**
12     * ユーザログイン認証の初期化を行います。 <BR>
13     */
14    @Override
15    public void init(Node node) {
16       // LDAP 認証情報モデルを作成します。
17       this.contextInfo = LDAPContextInfoBuilder.buildLDAPContextInfo(node);
18    }
19
20    /**
21     * LDAP 認証サーバに認証を依頼します。
22     */
23    public CertificationStatus certification(LoginInfo loginInfo, AccountInfo user, HttpServletRequest request,
24 HttpServletResponse response) {
25
26       // 設定された LDAP 認証サーバの数だけループさせます。
27       for (LDAPServerInfoModel serverInfo : this.contextInfo.getLDAPServers()) {
28
29          DirContext dirContext = null;
30          try {
31             // 検索するための設定を取得します。
32             dirContext = new InitialDirContext(serverInfo.getSearchEnvironment());
33
34             // 検索フィルタにユーザコードを設定します。
35             String filter = serverInfo.getSearchFilter().replaceAll("\\?", loginInfo.getUserCd());
36
37             // LDAP でユーザの検索を実行します。
38             NamingEnumeration<SearchResult> answer = dirContext.search(serverInfo.getBaseDn(), filter,
39 serverInfo.getSearchControls());
40
41             // 検索結果が存在するかを判定します。
42             if (answer.hasMoreElements()) {
43
44                SearchResult sr = (SearchResult) answer.nextElement();
45
46                // ユーザの検索結果を基にパスワードの検証を行います。
47                // 入力されたパスワードを認証情報として LDAP サーバに認証を依頼します。
48                String name = sr.getName();
49                dirContext = new InitialDirContext(serverInfo.getCertifyEnvironment(name, ",",
50 loginInfo.getPassword()));
51
52                // ここまで来たら LDAP 認証成功したこととします。
53                return CertificationStatus.CR_OK;
54             }
55
56          } catch (CommunicationException e) {
57
58             // 通信例外が発生した場合、エラーとします。
59             return CertificationStatus.CR_ERROR;
60
61          } catch (NamingException e) {
62
63             // ユーザの検索、またはパスワードの検証に失敗した場合、次の LDAP サーバで検証を行います。
64             continue;
65          }
66       }
67
```

```
    return CertificationStatus.CR_NG;
  }
}
```

intra-mart Accel Platform の LDAP 認証モジュール では、複数の LDAP 認証サーバを登録することが可能です。
そのため、取得した LDAP サーバに対して順にユーザの問い合わせを行っています。

問い合わせは、以下の処理を順に行います。

- ユーザコードの問い合わせ
- 問い合わせたユーザを基にした、パスワードの検証

問い合わせた結果ユーザコードが見つからなかった場合やパスワードの検証に失敗した場合には *NamingException* が発生します。
その場合は、次の認証サーバで検証を行うように処理を継続しています。

処理結果により、以下を返却します。

表 認証処理結果

条件	処理結果
ユーザが存在し、パスワード検証が成功した場合	*CertificationStatus.CR_OK*
すべての認証サーバでユーザ検索・パスワード検証を行い、検証に失敗した場合	*CertificationStatus.CR_NG*
通信エラーなどで LDAP サーバのアクセスに失敗した場合	*CertificationStatus.CR_ERROR*

注意

ここで例として記述しているソースでは、実際のソースの省略可能な部分を省略しています。
例えば、*InitialDirContext* のクローズ処理は記述していません。

認証プロバイダ・プラグインを登録する

plugin.xml の作成

plugin.xml を作成します。

今回は、認証プロバイダのみですので、以下の 1 つのみを作成し、「認証プロバイダ」、「認証条件判定プロバイダ」を記述してください。

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <plugin>
3    <extension point="jp.co.intra_mart.security.user.certification">
4      <certification
5        name="standard"
6        id="jp.co.intra_mart.security.user.certification.ldap"
7        version="8.0"
8        rank="90">
9        <certification-
10 class>jp.co.intra_mart.system.security.certification.provider.ldap.LDAPUserCertification</certification-class>
11       <certification-validation>
12         <validation-
13 class>jp.co.intra_mart.system.security.certification.provider.ldap.LDAPUserCertificationValidation</validation-
14 class>
15       </certification-validation>
     </certification>
   </extension>
</plugin>
```

> **注意**
>
> intra-mart Accel Platform の 標準プロバイダの「*rank*」属性は、「*100*」が設定されています。
>
> プラグインは、「*rank*」属性の小さい順に読み込まれますので、「*100*」より小さい値を設定する必要があります。

Web Application Server で動作確認する

Web Application Server で動作を確認するためには、作成したクラスと *plugin.xml* を Web Application Server のデプロイ先にコピーします。

モジュールを作成することで、War ファイルの作成時に自動的に正しい場所に配置されますが、ここでは動作確認のために以下のパスに手動でコピーします。

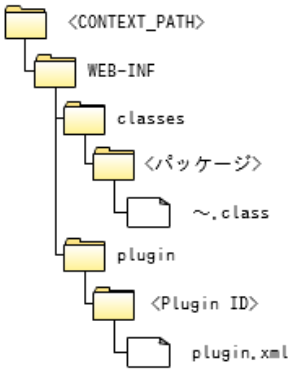

図 プロバイダの配置先

ファイルの配置後に、Web Application Server を起動します。
これにより、作成した認証モジュールが選択され、LDAP サーバによる認証方式が利用されます。

実際に LDAP 認証方式を利用するためには、LDAP サーバの構築と、LDAP の利用情報の設定が必要です。

intra-mart Accel Platform のLDAP 認証モジュールを利用した場合の LDAP の利用情報の設定は、「セットアップガイド 」-「 設定ファイル 」-「 LDAP認証設定ファイル 」を参照してください。

LDAP サーバの構築は、利用する LDAP サーバのドキュメントを参照してください。

その他の認証拡張機能

この章では、 その他の認証拡張機能について説明を行います。

SSO（シングルサインオン）対応

SSOユーザコードプロバイダを実装することで、intra-mart Accel Platform アクセス時にリクエスト情報（リクエストヘッダー、クッキー、リクエストパラメータなど）からユーザ情報を取得して自動ログインを行うことができます。
これにより、シングルサインオン（以下、SSO と記述します）を実現することができます。

intra-mart Accel Platform における SSO の仕様については、「認証仕様書」 - 「SSO（シングルサインオン）」を参照してください。

SSOユーザコードプロバイダ・プラグイン

一般ユーザのSSOを実現するには、次のプロバイダを実装します。

プラグイン一覧

プロバイダ名	概要	拡張ポイント
SSOユーザコードプロバイダ	リクエスト情報を元に、SSOさせたいユーザコードを返却します。	*jp.co.intra_mart.foundation.security.certification.sso.user.providers*

SSOユーザコードプロバイダ・プラグイン詳細

インタフェース定義

```java
package jp.co.intra_mart.foundation.security.certification.sso;

import javax.servlet.http.HttpServletRequest;

/**
 * シングルサインオンの自動ログイン用ユーザ情報の取得を行うインタフェースです。
 */
public interface SSOUserProvider {

   /**
    * 自動ログインを行うユーザコードを返却します。<br>
    * ユーザコードを取得できなかった場合はnullを返却します。
    * @param request サーブレットリクエスト
    * @return ユーザコード
    */
   String getUserCd(final HttpServletRequest request);

}
```

標準実装

初期状態で提供されるプロバイダはありません。

実装サンプル

リクエストパラメータ「sample_user_code」からユーザコードを取得し、SSOするサンプルです。
ユーザをリクエストパラメータ「sample_user_code」として送信することにより、自動ログインを実現します。
なお、本コードはサンプルであるため、セキュリティを考慮していません。

- サンプル Java コード

```java
package sample;

import javax.servlet.http.HttpServletRequest;

import jp.co.intra_mart.foundation.security.certification.sso.SSOUserProvider;

public class SampleSSOUserProvider implements SSOUserProvider {
   @Override
   public String getUserCd(final HttpServletRequest request) {
      // リクエストパラメータからSSOユーザコードを取得します
      final String userCd = request.getParameter("sample_user_code");

      if (userCd != null) {
         return userCd;
      }

      // nullを返却することで別のプロバイダに処理を委譲できます
      return null;
   }
}
```

- サンプルプラグイン設定ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
 <extension point="jp.co.intra_mart.foundation.security.certification.sso.user.providers">
  <sso-user-providers
   id="sample.programming_guide.sso_user_provider"
   name="Sample SSO User Provider"
   version="1.0"
   rank="50">
   <sso-user-provider class="sample.SampleSSOUserProvider"/>
  </sso-user-providers>
 </extension>
</plugin>
```

引数のリクエスト情報から、ユーザコードを返却するようにします。null を返却することで、別のプラグインに処理を委譲することができます。
全てのSSOユーザコードプロバイダプラグインが null を返却した場合、未認証としてアクセスされます。
SSOユーザコードプロバイダプラグインが 存在しないユーザコードを返却した場合、 intra-mart Accel Platform 2013 Winter(Felicia) までは未認証としてアクセスされ、intra-mart Accel Platform 2014 Spring(Granada) 以降のバージョンでは HTTP500 エラーが出力されます。

プラグインとして設定された順番にプロバイダが実行されます。
順番を考慮した実装および設定を行ってください。

動作確認を行うために以下のURLにアクセスしてください。ユーザ「aoyagi」でログインしていればサンプルは正しく動作しています。

- `http://<HOST>:<PORT>/<CONTEXT_PATH>/home?sample_user_code=aoyagi`

> **注意**
> SSOユーザコードプロバイダの実装において、ユーザコードの取得方法が容易に推測可能であるような方式は、セキュリティ上好ましくありません。 ユーザコードの暗号化またはデータの改ざん防止等を考慮した実装を行うことをお奨めします。

二重ログイン防止機能

項目

二重ログイン防止機能 動作仕様

二重ログイン防止機能とは、既にログインしているユーザと同じユーザコードを利用した別の環境（ブラウザ）からのログインを防止する機能です。（二重ログイン防止機能）
また、システム管理者およびテナント管理者がユーザのログイン状況を参照・無効化する機能を提供します。（ログインセッション管理機能）

二重ログイン防止機能を利用した際の認証処理は以下の動作を行います。

1. 有効な認証かどうか判定を行う。
   - ここでWebサービスや外部ソフトウェア接続モジュールなどの外部連携APIを利用した認証処理等は二重ログイン防止機能の対象外です。
   - 二重ログイン防止機能が無効な場合は通常のログイン処理が行われます。
2. ログイン情報とアカウント情報のユーザコード、パスワードの比較を行う。
   - ログイン情報とアカウント情報が異なる場合は、この時点で認証NGです。
3. データベースにログインセッション情報が登録されているかチェックを行う。
   - ログインセッション情報が存在する場合は、二重ログインとして認証NGです。
4. ログイン処理を行う。
5. ログインしたユーザのログインセッション情報をデータベースに登録する。

> **コラム**
> ログインセッション管理機能の画面操作に関しては「テナント管理者操作ガイド」 - 「ログインセッションを無効化する」を参照してください。

二重ログイン防止機能 標準認証プロバイダ・プラグイン

二重ログイン防止機能は以下の認証プロバイダ・プラグインで構成されています。

- 認証プロバイダ

  実装クラス

  jp.co.intra_mart.foundation.security.login_session.certification.LoginSessionUserCertification

  処理概要

  ログイン情報とアカウント情報のユーザコード、パスワードを比較します。
  ユーザコード、パスワードに間違いがなければ、ログインセッション情報を参照して、二重ログインチェックを行います。

- 認証リスナ

  実装クラス

  jp.co.intra_mart.foundation.security.login_session.listener.LoginSessionUserCertificationListener

  処理概要

  この認証リスナではログインセッション情報をデータベースへ登録します。
  登録されたログインセッション情報はユーザがログアウトした場合やセッションの有効期限が切れた場合等、セッションが無効になった時点で削除されます。

二重ログイン防止機能を無効化する方法

二重ログイン防止機能を無効化したい場合は、以下の plugin.xml の enable 属性に false を設定し、 intra-mart Accel Platform を再起動してください。
または、IM-Juggling のプロジェクトに以下のように設定した <plugin.xml> ファイルを含む plugin フォルダをプロジェクト直下にコピーして、WARファイルを作成し再デプロイしてください。

設定内容

ログイン用認証リスナ設定

```
%CONTEXT_PATH%/WEB-INF/plugin/jp.co.intra_mart.security.user.certification.login_session/plugin.xml
```

```
<plugin>
 <extension point="jp.co.intra_mart.security.user.certification">
  <certification
     name="standard"
     id="jp.co.intra_mart.security.user.certification.login_session"
     version="8.0"
     rank="90"
     enable="false">
    <certification-
class>jp.co.intra_mart.foundation.security.login_session.certification.LoginSessionUserCertification</certification-
class>
    <certification-listener>
     <listener-
class>jp.co.intra_mart.foundation.security.login_session.listener.LoginSessionUserCertificationListener</listener-
class>
    </certification-listener>
  </certification>
 </extension>
</plugin>
```

強制ログイン用認証リスナ設定

%CONTEXT_PATH%/WEB-INF/plugin/jp.co.intra_mart.security.user.certification.login_session_8.0.4/plugin.xml

```
<plugin>
 <extension point="jp.co.intra_mart.security.user.force_login">
  <certification
     name="standard"
     id="jp.co.intra_mart.security.user.certification.login_session"
     version="8.0.4"
     rank="90"
     enable="false">
    <certification-listener>
     <listener-
class>jp.co.intra_mart.foundation.security.login_session.listener.LoginSessionUserCertificationListener</listener-
class>
    </certification-listener>
  </certification>
 </extension>
</plugin>
```

また、二重ログイン防止機能を無効化してログインセッション管理機能のみ利用する場合は、ログイン用認証リスナ設定の <certification-class> をコメントアウトし認証リスナのみ有効にします。

設定内容

```
<plugin>
 <extension point="jp.co.intra_mart.security.user.certification">
  <certification
    name="standard"
    id="jp.co.intra_mart.security.user.certification.login_session"
    version="8.0"
    rank="90">
   <!-- certification-class をコメントアウトすると二重ログインチェックは行われません。
   <certification-class>jp.co.intra_mart.foundation.security.login_session.certification.LoginSessionUserCertification</certification-class>
    -->
   <certification-listener>
    <listener-class>jp.co.intra_mart.foundation.security.login_session.listener.LoginSessionUserCertificationListener</listener-class>
   </certification-listener>
  </certification>
 </extension>
</plugin>
```

二重ログインチェックを行わずにログイン処理を行う方法

UserCertificationManager を使用してログイン処理を実行する場合に、 forceLogin メソッドを使用して強制ログインを行うか、以下の拡張属性をログイン情報に設定することで、二重ログインチェックを行わずにログイン処理を実行する事ができます。

- 拡張属性名

  im_login_session.certification.validation

- 拡張属性値

  false

実装例は以下です。

```
loginRequestInfo.setAttribute("im_login_session.certification.validation", "false");
UserCertificationManager.getInstance().login(loginRequestInfo);
```

個別に作成した認証プロバイダで二重ログイン防止機能を実現する方法

新しい認証方式に対応するために開発した認証プロバイダ・プラグインで二重ログイン防止機能を実現するには以下の作業が必要です。
ここでは、 *サンプル* の「LDAP 認証」を利用した認証方式のプロバイダに二重ログインチェックを追加する例を利用しながら説明します。

ステップ1：作成した認証プロバイダを拡張した二重ログインチェックを行う認証プロバイダを作成する

新しい認証方式に対応する認証プロバイダを継承して、二重ログインチェックを行う認証プロバイダを作成します。
継承元の認証処理が成功した場合、二重ログインチェックを行うように実装します。
二重ログインチェックを行うには、以下のAPIを使用します。

APIクラス名

jp.co.intra_mart.foundation.security.login_session.certification.DuplicationLoginChecker

実装例

```
public class LoginSessionLDAPUserCertification extends LDAPUserCertification {
   @Override
   public CertificationStatus certification(final LoginInfo loginInfo, final AccountInfo user, final
HttpServletRequest request, final HttpServletResponse response) {
      // 継承元の認証処理を実行します。
      final CertificationStatus status = super.certification(loginInfo, user, request, response);
      if (status != CertificationStatus.CR_OK) {
         // 認証失敗の場合は、そのまま認証ステータスを返却します。
         return status;
      }
      // 二重ログインチェックを行います。
      return DuplicationLoginChecker.certification(loginInfo, user, request, response);
   }
}
```

ステップ2：plugin.xml の設定を追加する

plugin.xml に作成した二重ログインチェックを行う認証プロバイダの設定を変更します。
新しい認証方式に対応する認証プロバイダの設定を記述した plugin.xml の以下の内容を変更します。

<certification> の rank 属性

標準の二重ログイン防止認証プロバイダよりも小さい値（90未満）を指定します。

<certification>/<certification-class>

設定内容

jp.co.intra_mart.system.security.certification.provider.ldap.LoginSessionLDAPUserCertification

ステップ1で作成した認証プロバイダクラスを設定します。

<certification>/<certification-listener>/<listener-class>

設定内容

jp.co.intra_mart.foundation.security.login_session.listener.LoginSessionUserCertificationListener

ログインセッションを登録する認証リスナクラスを設定します。

設定例

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
  <extension point="jp.co.intra_mart.security.user.certification">
    <certification
      name="standard"
      id="jp.co.intra_mart.security.user.certification.ldap"
      rank="80">
      <certification-class>jp.co.intra_mart.system.security.certification.provider.ldap.LoginSessionLDAPUserCertification</certification-class>
      <!-- 変更前の認証プロバイダクラス
      <certification-class>jp.co.intra_mart.system.security.certification.provider.ldap.LDAPUserCertification</certification-class>
       -->
      <!-- ログインセッション情報を登録するための認証リスナクラス -->
      <certification-listener>
        <listener-class>jp.co.intra_mart.foundation.security.login_session.listener.LoginSessionUserCertificationListener</listener-class>
      </certification-listener>
      <certification-validation>
        <validation-class>jp.co.intra_mart.system.security.certification.provider.ldap.LDAPUserCertificationValidation</validation-class>
      </certification-validation>
    </certification>
  </extension>
</plugin>
```

作成したクラスと plugin.xml を含めたWARファイルを作成し再デプロイを行うと、二重ログイン防止機能が動作します。

ログインポートレット対応

項目

コラム

ログインポートレットは2017 Winter(Rebecca)より利用できます。

注意

本設定を利用する場合は、以下のモジュールが必要です。

- モジュール名：「ポータル」
  モジュールID：jp.co.intra_mart.im_portal
- モジュール名：「ログインポートレット」
  モジュールID：jp.co.intra_mart.im_login_portlet

ログインポートレットは、一般ユーザ向けのログイン処理が行えるログイン機能を持つポートレットです。
ログインポートレットを利用する場合は、「ログイン画面表示設定」画面から設定を有効にする必要があります。
詳細は「テナント管理者操作ガイド」-「ログイン画面の表示を切り替える」を参照してください。

本ページでは、ログインポートレットの拡張機能の対応方法について説明します。
また、本ページでは以下の赤枠の項目を「ログインフォーム」と称します。

ログインポートレットに任意のページを差し込む方法

ログインポートレットでは、ログインポートレット内に任意のページを差し込むことができます。
お客様が作成したページや画像をログインポートレット内の任意の位置に差し込み、ログインポートレットの画面を自由にカスタマイズすることができます。
設定ファイルについての詳細は、「設定ファイルリファレンス」-「ログインポートレット表示設定」を参照してください。
画面作成の際は、「ポートレット プログラミングガイド」-「注意事項」も併せて確認してください。

ログインポートレットの画面構成

ログインポートレットの画面は、以下で構成されています。

```
<div id="login-portlet-view">

  <div id="login-portlet-view-top">
    <!-- position要素がTOPの場合は、ここに差し込まれます。 -->
  </div>

  <div id="login-portlet-view-middle">

    <div id="login-portlet-view-left">
      <!-- position要素がLEFTの場合は、ここに差し込まれます。 -->
    </div>

    <div id="login-portlet-view-center">
      <div>
        <!-- 通常のログイン画面はここに差し込まれます。 -->
      </div>
      <!-- position要素がCENTERの場合は、ここに差し込まれます。 -->
    </div>

    <div id="login-portlet-view-right">
      <!-- position要素がRIGHTの場合は、ここに差し込まれます。 -->
    </div>

  </div>

  <div id="login-portlet-view-bottom">
    <!-- position要素がBOTTOMの場合は、ここに差し込まれます。 -->
  </div>
</div>
```

表示位置設定の値によって、以下のid属性が指定されたdiv要素内にページが差し込まれます。

- CENTER（中心）の場合： id属性 login-portlet-view-center
- TOP（上）の場合： id属性 login-portlet-view-top
- BOTTOM（下）の場合： id属性 login-portlet-view-bottom
- LEFT（左）の場合： id属性 login-portlet-view-left
- RIGHT（右）の場合： id属性 login-portlet-view-right

表示位置に任意のページを差し込むサンプル実装

全ての表示位置に差し込みを行った場合のサンプルは以下です。

上記のサンプルの設定方法を紹介します。

まず、差し込みたいページを作成します。以下は作成例です。

- プレゼンテーション・ページ（sample_top.html）

```
<style type="text/css">
 .sample-top {
   text-align: center;
   border: 1px solid skyblue;
 }
</style>
<div class="sample-top">TOPの場合</div>
```

- ファンクション・コンテナ（sample_top.js）

```
function init() {
 // 任意の処理を実装してください。
}
```

- プレゼンテーション・ページ（sample_top2.html）

```
<style type="text/css">
#sample-top2 {
  font-size: xx-small;
  border: 1px solid cyan;
  text-align: center;
}
</style>
<div id="sample-top2">ソート数によって差し込まれる位置の並び順を指定できます。</div>
```

- ファンクション・コンテナ（sample_top2.js）

```
function init() {
 // 任意の処理を実装してください。
}
```

- プレゼンテーション・ページ（sample_bottom.html）

```
<style type="text/css">
 .sample-bottom {
   border: 1px solid tomato;
   text-align: center;
   margin-top: 10px;
 }
</style>
<div class="sample-bottom">BOTTOMの場合</div>
```

- ファンクション・コンテナ（sample_bottom.js）

```
function init() {
 // 任意の処理を実装してください。
}
```

- プレゼンテーション・ページ（sample_right.html）

```
<style type="text/css">
 .sample-right {
   text-align: right;
   border: 1px solid lime;
 }
</style>
<div class="sample-right">RIGHTの場合</div>
```

- ファンクション・コンテナ（sample_right.js）

```
function init() {
 // 任意の処理を実装してください。
}
```

- プレゼンテーション・ページ（sample_left.html）

```
<style type="text/css">
 #login-portlet-view-middle {
   align-items: center;
 }
 .sample-left {
   border: 1px solid olive;
 }
</style>
<div class="sample-left">LEFTの場合</div>
```

- ファンクション・コンテナ（sample_left.js）

```
function init() {
 // 任意の処理を実装してください。
}
```

- プレゼンテーション・ページ（sample_center.html）

```
<style type="text/css">
 .sample-center {
   border: 1px solid orange;
   text-align: center;
   font-size: x-small;
 }
</style>
<div class="sample-center">
 CENTERの場合は<br>
 ログインフォームの下側に<br>
 差し込まれます。
</div>
```

- ファンクション・コンテナ（sample_center.js）

```
function init() {
 // 任意の処理を実装してください。
}
```

今回は例として、 jssp/src/sample に配置します。
次に、設定ファイルに作成したページ、および、表示位置などの情報を設定します。
今回は例として、 conf/login-portlet-view-config/login-portlet-view-config_sample.xml に設定ファイルを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<login-portlet-view-config xmlns="http://www.intra-mart.jp/im_login_portlet/login-portlet-view-config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.intra-
mart.jp/im_login_portlet/login-portlet-view-config ../../schema/login-portlet-view-config.xsd ">
   <view>
      <page>/sample/sample_top</page>
      <position>TOP</position>
      <sort>100</sort>
   </view>
   <view>
      <page>/sample/sample_top2</page>
      <position>TOP</position>
      <sort>200</sort>
   </view>
   <view>
      <page>/sample/sample_bottom</page>
      <position>BOTTOM</position>
      <sort>100</sort>
   </view>
   <view>
      <page>/sample/sample_right</page>
      <position>RIGHT</position>
      <sort>100</sort>
   </view>
   <view>
      <page>/sample/sample_left</page>
      <position>LEFT</position>
      <sort>100</sort>
   </view>
   <view>
      <page>/sample/sample_center</page>
      <position>CENTER</position>
      <sort>100</sort>
   </view>
</login-portlet-view-config>
```

設定後、ユーザモジュールを作成、および、設定しアプリケーションサーバを再起動してください。
ログインポートレットに作成したページが差し込まれることを確認してください。
このように、ページ内にJavaScriptやCSSなどの任意の処理を実装することで、ログインポートレットを自由にカスタマイズすることができます。

ログインフォームの表示を制御する方法

ログインポートレットでは、ログインフォームを表示するかどうかを制御できます。
先述の通り、ログインポートレットでは任意のページを差し込むことができますが、ログインフォームはカスタマイズすることができません。
ログインフォームを利用せずに独自のログインフォームを用意したい場合や、何らかの理由により、ログインフォームを非表示にしたい場合は本対応を行ってください。

以下のインタフェースを実装したクラスを作成してください。

`jp.co.intra_mart.foundation.portal.portlet.login.LoginViewController`

標準の動作仕様は以下です。

- 実装されたクラスが存在しない場合は、ログインフォームを表示する。
- ログインフォームを非表示にするクラスが1つでも存在する場合、ログインフォームを非表示にする。

ログインポートレットモジュールには、標準で実装済みの機能、および、クラスはありません。

> **コラム**
>
> SAML認証機能 ログインポートレット連携モジュールを利用している場合は、SAML認証機能側の表示制御が機能します。
> SAML認証機能側の表示制御については、「SAML認証プログラミングガイド」-「ログインフォームの表示方法を制御する」を参照してください。

リクエストパラメータによってログインフォームの表示を制御するサンプル実装

ログインフォームを非表示にする場合のサンプルの実装方法を紹介します。
今回は例として、パラメータ「login_form_disabled」がリクエストに付与されている場合に、ログインフォームを非表示にします。
まず、インタフェースを実装したクラスを作成します。以下は作成例です。

```java
package jp.co.sample.portal.portlet.login;

import javax.servlet.http.HttpServletRequest;

import jp.co.intra_mart.foundation.portal.portlet.login.LoginViewController;

/**
 * サンプルの実装です。<br>
 * login_form_disabledパラメータが付与されている場合は、ログインフォームを非表示にします。
 * @author INTRAMART
 * @version 8.0.0
 */
public class SampleLoginViewController implements LoginViewController {

    @Override
    public boolean isView(final HttpServletRequest request) {
        if (request.getParameter("login_form_disabled") != null) {
            // login_form_disabledパラメータが付与されている場合はログインフォームを非表示にします。
            return false;
        }
        return true;
    }
}
```

次に作成したクラスをサービス定義ファイルに追加します。以下は作成例です。

- resources/META-INF/services/jp.co.intra_mart.foundation.portal.portlet.login.LoginViewController

```
jp.co.sample.portal.portlet.login.SampleLoginViewController
```

作成後、ユーザモジュールを作成、および、設定しアプリケーションサーバを再起動してください。
以下のURLにアクセスすることで、ログインフォームが非表示になることが確認できます。
http://<HOST>:<PORT>/<CONTEXT_PATH>/login?login_form_disabled

多要素認証機能

項目
- 多要素認証を行わずにログイン処理を行う方法

多要素認証機能は、ログインを行う際に通常のユーザコード/パスワードでの認証以外に、複数の要素をユーザに要求し認証する機能です。
intra-mart Accel Platform では認証アプリケーションを用いた追加認証に対応しています。

> **コラム**
>
> 多要素認証機能は 2018 Spring(Skylark) より利用できます。

多要素認証を行わずにログイン処理を行う方法

多要素認証機能が有効になっている場合、UserCertificationManager を利用してログイン処理を実行する際には、多要素認証処理が実行されます。
この際、多要素認証処理を行わず通常の認証のみ行いたい場合、以下の拡張属性をログイン情報に設定してください。
または、forceLogin メソッドを使用して強制ログインを行ってください。

- 拡張属性名

  im_multi_factor_authentication.certification

- 拡張属性値

false

実装例は以下です。

```
loginRequestInfo.setAttribute("im_multi_factor_authentication.certification", "false");
UserCertificationManager.getInstance().login(loginRequestInfo);
```