Copyright © 2013 NTT DATA INTRAMART CORPORATION

目次

- 改訂情報
- はじめに
  - 本書の目的
  - 対象読者
  - 本書の構成
- 基本（ intra-mart Accel Platform での初めてのプログラミング）
  - 前提条件
  - intra-mart e Builder for Accel Platform で Hello World を作ろう
  - 登録と画面表示
  - 参考
- 応用（ intra-mart Accel Platform の機能を使いこなす）
  - 認可
  - アクセスコンテキスト
  - 認証機能
  - 国際化
  - データベース
  - ログ
  - UI（デザインガイドライン）
  - UI（スマートフォン開発ガイドライン）
  - エラー処理
  - Storage
  - 非同期処理
  - ジョブスケジューラ
  - Lockサービス
  - Cacheサービス
  - ショートカットアクセス機能
  - CSRF対策
  - RESTful Web Service
  - 設定ファイル
- 参考（ TERASOLUNA Server Framework for Java (5.x) for Accel Platform アーキテクチャ ）
  - TERASOLUNA Server Framework for Java (5.x) for Accel Platform の構成
  - TERASOLUNA Server Framework for Java (5.x) との対応
  - TERASOLUNA Server Framework for Java (5.x) for Accel Platform におけるBean定義の既定値

改訂情報

変更年月日	変更内容
2013-10-01	初版
2014-01-01	第2版 下記を追加・変更しました。 ▪ 「はじめに - 本書の目的」を修正しました。 ▪ 「応用 - 認可」に「メニュー設定と認可についての注意事項」を追記しました。 ▪ 「応用 - 国際化 - 国際化プログラミングのサンプル」に「レスポンスにJSONをセットして返す時のDate型プロパティのフォーマットを設定する。」を追記しました。 ▪ 「応用 - ログ - TERASOLUNA Global Framework の TraceLoggingInterceptor を利用する。」を修正しました。 ▪ 「応用 - UI（デザインガイドライン）」を別ドキュメント UIデザインガイドライン（PC版）に移行しました。 ▪ 「応用 - エラー処理」に「応用 - UI（デザインガイドライン） - 規約／ルール - エラー処理 - 汎用メッセージ画面」の項目を移しました。 ▪ 「応用 - Storage - ファイルダウンロード」を修正しました。 ▪ 「参考」を修正しました。 ▪ 「参考 - TERASOLUNA Global Framework on Accel Platform におけるBean定義の既定値 - applicationContext-im_tgfw_web.xml」を修正しました。 ▪ 「参考 - TERASOLUNA Server Framework for Java (5.x) との対応 」を追記しました。
2014-04-01	第3版 下記を変更しました。 ▪ 「応用 - 認可」のパブリックストレージパスの記述を修正しました。
2014-08-01	第4版 下記を追加・変更しました ▪ 「*アクセスコンテキスト*」の仕様の記述を「 アクセスコンテキスト仕様書 」へ移動
2014-09-01	第5版 下記を変更しました。 ▪ 「参考 - TERASOLUNA Global Framework との対応」のシステム時刻についての記述を修正しました。
2014-12-01	第6版 下記を変更しました。 ▪ 「*ジョブスケジューラ*」にジョブの実行方法についての記述を追加 ▪ 「*CSRF対策*」にCSRF対策の設定方法についての記述を追加 ▪ 「*参考（TERASOLUNA Server Framework for Java (5.x) for Accel Platform アーキテクチャ）*」のCSRFについての記述を修正しました。 ▪ 「参考 - TERASOLUNA Global Framework on Accel Platform におけるBean定義の既定値 - applicationContext-im_tgfw_common.xml」を intra-mart Accel Platform2014 Winter(Iceberg)にあわせて修正しました。 ▪ 「参考 - TERASOLUNA Global Framework on Accel Platform におけるBean定義の既定値 - applicationContext-im_tgfw_web.xml」を intra-mart Accel Platform2014 Winter(Iceberg)にあわせて修正しました。 ▪ 「*基本（intra-mart Accel Platform での初めてのプログラミング）*」の「入力画面の作成」にJSPタグの制限事項を追加しました。 ▪ 「*参考（TERASOLUNA Server Framework for Java (5.x) for Accel Platform アーキテクチャ）*」にjackson1、jackson2が混在する環境の場合での注意点の記述を追加しました。

変更年月日	変更内容
2015-04-01	第7版 下記を変更しました。 ▪ TERASOLUNA Global Framework を TERASOLUNA Server Framework for Java (5.x) に変更しました。 ▪ Spring Frameworkのリンクを 4.1.4.RELEASE に変更しました。 ▪ 「*データベース*」のガイドラインをTERASOLUNAのバージョンアップによりMyBatis2からMyBatis3に変更しました。 ▪ 「*データベース*」のガイドラインからSpring Data JPAを削除しました。 ▪ 「*実装例：登録画面を作る*」のデータアクセスをJPAからMyBatis3に変更しました。 ▪ 「*参考（TERASOLUNA Server Framework for Java (5.x) for Accel Platform アーキテクチャ）*」を「参考（ TERASOLUNA Global Framework on Accel Platform アーキテクチャ ）」から変更しました。 ▪ jacksonのバージョンを2に統一したため、「*参考（TERASOLUNA Server Framework for Java (5.x) for Accel Platform アーキテクチャ）*」からjackson1、jackson2が混在する環境の場合での注意点の記述を削除しました。
2015-08-01	第8版　下記を追加しました。 ▪ 「*応用（intra-mart Accel Platform の機能を使いこなす）*」に「*設定ファイル*」を追加 ▪ 「*概要*」に jQuery Mobile 1.3.0 ベースで作成したガイドであることの注意書きを追加 ▪ 「*スマートフォン版テーマ*」 に 「*テーマが読み込むライブラリ群の切り替え*」 を追加 ▪ 「*スマートフォン版テーマ*」 の 「*テーマとスウォッチ*」 をjQuery Mobile のバージョン別に表記するように修正 ▪ 「*推奨画面構成*」の「*処理リンク*」から「 CSS Sprite Image List のスマートフォン向け」へのリンクを追加
2015-12-01	第9版 下記を追加・変更しました ▪ 「*アクセスコンテキスト - プログラミング方法*」に「*一時的に実行ユーザを変更して処理を実行する*」を追加 ▪ 「*データベース*」の「*MyBatis-Springの設定*」と「*TypeAliasの設定*」の設定ファイルについての説明を追加しました。 ▪ 「*データベース*」の「*エンティティクラスの自動生成*」の説明を追記しました。
2016-04-01	第10版 下記を追加・変更しました ▪ Spring Frameworkのリンクを 4.2.4.RELEASE に変更しました。 ▪ 「*基本（intra-mart Accel Platform での初めてのプログラミング）*」内の制限事項を外しました（「この制限事項は 2015 Winter より外れました。」。） ▪ 「*RESTful Web Service*」を追加。 ▪ 「*TERASOLUNA Server Framework for Java (5.x) との対応*」に「RESTful Web Service」と「RESTクライアント（HTTPクライアント）」、「SOAP Web Service（サーバ/クライアント）」、「E-mail送信（SMTP）」を追加。
2016-12-01	第11版 下記を追加・変更しました ▪ 「*メニュー設定と認可についての注意事項*」に説明を追記しました。 ▪ Spring Frameworkのリンクを 4.2.7.RELEASE に変更しました。 ▪ 「*TERASOLUNA Server Framework for Java (5.x) との対応*」に「JSP Tag Library と EL Functions」と「ヘルスチェック」、「日付操作(JSR-310 Date and Time API)」、「文字列処理」、「JMS(Java Message Service)」を追加。 ▪ 「*TERASOLUNA Server Framework for Java (5.x) との対応*」の対応表の構成を変更。
2017-04-01	第12版 下記を追加・変更しました。 ▪ 「*コードリスト*」を追加しました。

変更年月日	変更内容
2017-12-01	第13版 下記を追加・変更しました。 ▪ Spring Frameworkのリンクを 4.3.5.RELEASE に変更しました。 ▪ 「*基本（intra-mart Accel Platform での初めてのプログラミング）*」の前提条件を変更しました。
2018-08-01	第14版 下記を追加・変更しました ▪ 「*スマートフォン版テーマ*」 の 「*テーマとスウォッチ*」 にスマートフォン標準テーマ白用のスウォッチイメージを追記
2019-08-01	第15版 下記を追加・変更しました。 ▪ Spring Frameworkのリンクを 5.1.4.RELEASE に変更しました。 ▪ 「*TERASOLUNA Server Framework for Java (5.x) for Accel Platform におけるBean定義の既定値*」の applicationContext-im_tgfw_common.xml を intra-mart Accel Platform2019 Summer(Waltz)にあわせて修正しました。
2019-12-01	第16版 下記を追加・変更しました。 ▪ 「*JSONでの日付の項目とJavaのオブジェクトのDate型のプロパティとを変換する。*」を AccountDateDeserializerの追加に合わせて修正しました。 ▪ 「応用 - コードリスト」の項目は、CodeListInterceptorのfallbackToに関する問題点が TERASOLUNA Server Framework for Java (5.x) 5.5.1.RELEASE で解決されたため、削除しました。
2022-12-01	第17版 下記を追加・変更しました。 ▪ Spring Frameworkのリンクを 5.3.18 に変更しました。 ▪ 「*TERASOLUNA Server Framework for Java (5.x) for Accel Platform におけるBean定義の既定値*」の applicationContext-im_tgfw_common.xml を intra-mart Accel Platform2022 Winter(Freesia)にあわせて修正しました。

はじめに

本書の目的

TERASOLUNA Server Framework for Java (5.x) (http://terasolunaorg.github.io/) は、Spring Framework (https://spring.io/projects/spring-framework) をベースに構成されるJavaフレームワークです。
本書では、TERASOLUNA Server Framework for Java (5.x) を利用したアプリケーションを intra-mart Accel Platform 上で開発する場合の基本的な方法、各機能仕様とそのプログラミング方法や注意点等について説明します。

TERASOLUNA Server Framework for Java (5.x) を活用したより詳細な開発手法に関しては TERASOLUNA Server Framework for Java (5.x) Development Guideline をご確認ください。

対象読者

次の開発者を対象としてます。

- intra-mart Accel Platform で初めてプログラミングを行う開発者
- intra-mart Accel Platform の各機能を利用したい開発者
- TERASOLUNA Server Framework for Java (5.x) for Accel Platform のアーキテクチャについて理解を深めたい開発者

本書の構成

本書は上記の対象読者に応じて次の３つの構成を取っています。

- *基本（intra-mart Accel Platform での初めてのプログラミング）*

  intra-mart Accel Platform で初めてプログラミングを行う場合について説明します。
- *応用（intra-mart Accel Platform の機能を使いこなす）*

  intra-mart Accel Platform の各機能仕様とそのプログラミング方法について説明します。
- *参考（TERASOLUNA Server Framework for Java (5.x) for Accel Platform アーキテクチャ）*

  intra-mart Accel Platform と TERASOLUNA Server Framework for Java (5.x) の連携によるフルスタックフレームワークについて説明します。

基本（ intra-mart Accel Platform での初めてのプログラミング）

本項では、 intra-mart Accel Platform での開発の導入として、intra-mart e Builder for Accel Platform で TERASOLUNA Server Framework for Java (5.x) を利用した Hello World を作成することによって intra-mart Accel Platform での TERASOLUNA Server Framework for Java (5.x) の開発の流れを体験します。

チュートリアルの流れ

- 前提条件
- intra-mart e Builder for Accel Platform で Hello World を作ろう
  - ステップ1：プロジェクトの作成と設定
  - ステップ2：Bean定義ファイル applicationContext.xml の作成
  - ステップ3：入力画面の作成
  - ステップ4：出力画面の作成
  - ステップ5：Formクラスの作成
  - ステップ6：Controllerクラスの作成
- 登録と画面表示
  - ステップ1：メニューの登録
  - ステップ2：Hello Worldの動作
- 参考
  - すぐ試してみたい方は作成済みのプロジェクトをインポートしてください

前提条件

- intra-mart e Builder for Accel Platform をインストール済みであること。
- intra-mart Accel Platform をインストールし、初期設定までが完了していること。
  Junglingのモジュール構成にてベースモジュールに TERASOLUNA Server Framework for Java (5.x) for Accel Platform を含めて作成してください。

intra-mart e Builder for Accel Platform で Hello World を作ろう

以下の手順で Hello World を作成していきます。

ステップ1：プロジェクトの作成と設定

intra-mart e Builder for Accel Platform 上にモジュール・プロジェクトを作成し、プロジェクトの設定を行います。
本項では、モジュール・プロジェクトは「hello_terasoluna」という名称で作成した場合を例に説明します。
プロジェクトの作成・設定の方法に関しては、『 intra-mart e Builder for Accel Platform アプリケーション開発ガイド』の『モジュール・プロジェクト作成』、および『プロジェクトの設定』の項を参照してください。

ステップ2：Bean定義ファイル applicationContext.xml の作成

プロジェクト固有のBean定義ファイルを作成します。
ここでは、src/main/resources/META-INF/spring/applicationContext-hello_terasoluna.xml として作成します。
この設定により、指定パッケージ配下のクラスがSpringのコンポーネント対象として扱われます。
なお、下記の例ではルートパッケージを”sample.tgfw”とします。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd">

   <!-- DIコンポーネントの対象とする要素のトップレベルパッケージ -->
   <context:component-scan base-package="sample.tgfw" />

</beans>
```

コラム

Bean定義ファイルを作成する際、以下のフォルダにapplicationContextで始まるxmlファイル名で配置してください。
フォルダ：src/main/resources/META-INF/spring/ または src/main/webapp/WEB-INF/classes/META-INF/spring/
ファイル名の例： applicationContext-foo.xml

ステップ3：入力画面の作成

入力画面（index.jsp）を作成します。
プロジェクトの src/main/webapp の下に /WEB-INF/views/sample/tgfw/hello という階層でフォルダを作成し、作成したフォルダ配下に「 index.jsp 」という名前でファイルを作成し、以下のソースを実装します。

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="imui"   uri="http://www.intra-mart.co.jp/taglib/imui"%>
<%@ taglib prefix="form"   uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<!-- HEADタグ -->
<imui:head>
 <title>Hello, World</title>
 <script type="text/javascript">
  $(function() {
   $('#button').click(function() {
    $('#helloForm').submit();
   });
  });
 </script>
</imui:head>

<!-- 画面上に表示されるタイトル -->
<div class="imui-title">
 <h1>Hello, World (TERASOLUNA)</h1>
</div>

<!-- 入力画面 -->
<div class="imui-form-container-narrow">
 <p>
  <label for="name">Please input the name. </label>
 </p>
 <!-- 入力フォームの設定
     actionに結果表示画面へのパスを入力(Helloアクションのoutputメソッドを呼び出す) -->
 <form:form action="sample/tgfw/hello/output" modelAttribute="helloForm">
  <div>
   <!-- テキストボックス -->
   <imui:textbox type="text" value='' id="name" name="name" size="10" />
  </div>
  <!-- submitボタン -->
  <imui:button name="button" value="Hello!!" class="mt-10" id="button" />
 </form:form>
</div>
```

注意

文字コードを UTF-8 にして保存してください。

ステップ4：出力画面の作成

出力画面（output.jsp）を作成します。
プロジェクトの src/main/webapp の下の /WEB-INF/views/sample/tgfw/hello の配下に
「 output.jsp 」という名前でファイルを作成し、以下のソースを実装します。

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="imui"   uri="http://www.intra-mart.co.jp/taglib/imui"%>
<%@ taglib prefix="form"   uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="f"      uri="http://terasoluna.org/functions" %>
<%@ taglib prefix="t"      uri="http://terasoluna.org/tags" %>

<!-- HEADタグ -->
<imui:head>
  <title>Hello, World</title>
</imui:head>

<!-- 画面上に表示されるタイトル -->
<div class="imui-title">
  <h1>Hello, World (TERASOLUNA)</h1>
</div>

<!-- ツールバー(前の画面へと戻るボタンを配置) -->
<div class="imui-toolbar-wrap">
  <div class="imui-toolbar-inner">
    <ul class="imui-list-toolbar">
      <li>
      <!-- 「戻る」ボタンのアイコン、HelloControllerのindexメソッドが呼び出される。 -->
        <a href="sample/tgfw/hello" class="imui-toolbar-icon" title="back">
          <span class="im-ui-icon-common-16-back"></span>
        </a>
      </li>
    </ul>
  </div>
</div>

<!-- 出力結果 -->
<div class="imui-form-container-narrow">
  <label>
    <imui:textbox type="text" value="${f:h(helloForm.name)}" id="name" name="name" size="10" class="imui-text-readonly"
readonly />
  </label>
</div>
```

コラム

- intra-mart Accel Platform 上で動作するHTMLファイルに記述するタグで <HTML>、<BODY>は記述せず、<HEAD>は<imui:head>を利用してください。 詳細は、 *UI（デザインガイドライン）* を参照してください。
- TERASOLUNA Server Framework for Java (5.x) for Accel Platform において、JSPなどのView用のファイルを置くディレクトリを/WEB-INF/classes/META-INF/spring/applicationContext-im_tgfw_web.xmlで設定しています。
  デフォルト値は/WEB-INF/viewsが指定してあります。

注意

文字コードを UTF-8 にして保存してください。

ステップ5：Formクラスの作成

1. 入力情報を保持する Form クラスを作成します。
   ここでは、パッケージ名は sample.tgfw.app.hello、クラス名は HelloForm とします。
   HelloForm に記述するソースは以下の通りです。

```java
package sample.tgfw.app.hello;

/**
 * index.jspで入力された情報を保持するFormクラスです。
 * @author intra-mart
 */
public class HelloForm {

    private String name; // テキストエリアに入力された値

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

注意

文字コードを UTF-8 にして保存してください。

ステップ6：Controllerクラスの作成

入力画面、および出力画面に対する処置を行う Controller クラスを作成します。
ここでは、パッケージ名はsample.tgfw.app.hello、クラス名は HelloController とします。
クラスには、 @Controller アノテーション、 @RequestMapping アノテーションを設定してください。
URLと対応させるメソッドには、 @RequestMapping アノテーションを設定してください。
Controller に記述するソースは以下の通りです。

```java
package sample.tgfw.app.hello;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("sample/tgfw/hello")
public class HelloController {

    @RequestMapping
    public String index() {
        return "sample/tgfw/hello/index.jsp";
    }

    @RequestMapping("output")
    public String output(HelloForm helloForm) {
        return "sample/tgfw/hello/output.jsp";
    }
}
```

注意

文字コードを UTF-8 にして保存してください。

コラム

リクエストとメソッドの対応は、 RequestMappingHandlerMapping と RequestMappingHandlerAdapter を利用しています。
applicationContext*.xml に mvc:annotation-driven の設定がある必要があります。
TERASOLUNA Server Framework for Java (5.x) for Accel Platform では、標準で applicationContext-im_tgfw_web.xml にこの設定をしています。

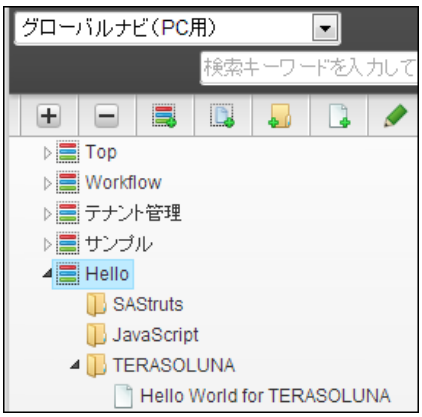
ここまでで Hello World の作成は完了です。
ここまで作成されたソースは、プロジェクトの設定によって Resin サーバ上のフォルダに適切にデプロイされています。

本番環境に移行する際は、作成したプロジェクトをユーザ定義モジュール化、または、ソースをデプロイして適用を行ってください。

登録と画面表示

ステップ1：メニューの登録

ここでは、上記で作成したサンプルをサイトマップに登録して実際に実行します。
Resin を起動し、テナント管理者で intra-mart Accel Platform にログインします。
ログイン画面の URL は「 http://<HOST>:<PORT>/<CONTEXT_PATH>/login 」です。
その後、メニューのメンテナンス画面を表示し、メニューを以下のように設定します。



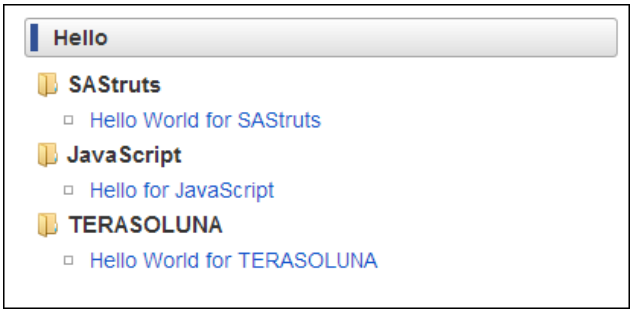
ここで作成する新規アイテムの情報に以下の情報を入力します

メニューアイテム名（日本語）	Hello World for TERASOLUNA
URL	sample/tgfw/hello

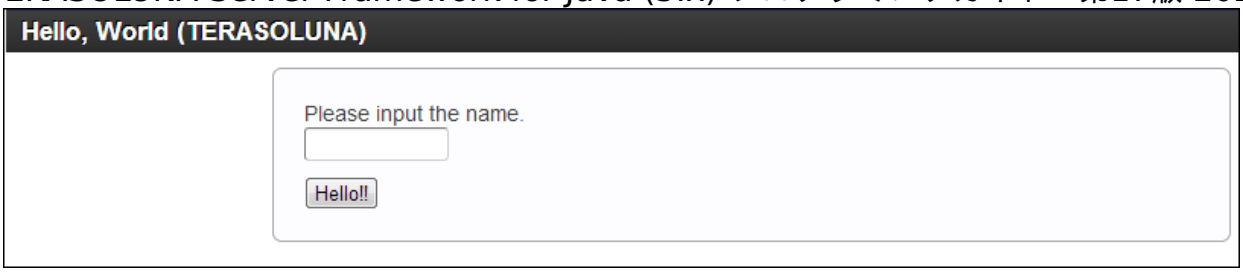
また、メニューを閲覧できるようにするために作成した「 Hello 」グループに以下のように認可を設定します。
本項では、Hello グループに対してゲストユーザと認証ユーザの参照を許可するように設定します。
詳細については *認可* を参照してください。

ステップ2：Hello Worldの動作

設定したメニューを反映させ、実行します。
下の画像のようにサイトマップから選択できます。



サイトマップで先ほど登録したメニューアイテムをクリックすると入力画面が表示されます。



テキストボックスに名前を入力すると結果画面に表示されます。

結果画面の上部にある「戻る」アイコンを押下すると、入力画面へと戻ります。

> **コラム**
>
> このチュートリアルでは、下記のポイントを確認しました。
>
> - TERASOLUNA Server Framework for Java (5.x) を用いて、簡単なアプリケーション（Hello World）の作成を通して、
>   intra-mart Accel Platform におけるアプリケーション作成からメニューに登録するまでの流れを把握しました。

参考

すぐ試してみたい方は作成済みのプロジェクトをインポートしてください

本チュートリアルで作成した Hello World は、モジュール・プロジェクト「hello_terasoluna」として **hello_terasoluna.zip** より入手できます。
このダウンロードした zip ファイルは、下記手順にて e Builder にインポートでき、すぐ実行確認できます。
はじめに、下記手順にて e Builder にインポートしてください。

1. e Builder を起動
2. ツールバーメニュー[ファイル]-[インポート]よりインポートウィザード画面を開く
3. 項目[General]-[既存プロジェクトをワークスペースへ]を選択し次へ
4. [アーカイブ・ファイルの選択]項目よりダウンロードしたzipファイルを選択し、[終了]ボタンをクリック

以上の手順で、モジュール・プロジェクト「hello_terasoluna」がインポートされます。
次に、モジュール・プロジェクトのプロパティにてWebアーカイブディレクトリを設定し、デバッグサーバを起動します。デバッグサーバ起動後、ブラウザより下記のように直接URLを指定してアクセスすると、入力画面が表示され、Hello World の動作確認が行えます。

```
http://<HOST>:<PORT>/<CONTEXT_PATH>/sample/tgfw/hello
```

応用（ intra-mart Accel Platform の機能を使いこなす）

認可

項目

- 画面にアクセス権限を設定するために
- 認可とは
  - 認可の概要
  - サブジェクト
  - リソースとリソースグループ
  - リソースタイプとアクション
  - ポリシー
- 画面へアクセスするための認可設定方法
  - 権限設定の流れ
  - ステップ1：@RequestMappingメソッドに認可を紐づける
  - ステップ2：認可のリソースグループ、リソースを登録する
  - ステップ2-1：認可設定画面を利用して認可のリソースグループ、リソースを登録する
  - ステップ2-2：ジョブを利用して認可のリソースグループ、リソースを登録する
  - ステップ3：リソースに対して権限を設定する
- メニュー設定と認可についての注意事項

画面にアクセス権限を設定するために

```
@RequestMapping()
public String index() {
   return "index.jsp";
}
```

Controllerクラスの設定により画面へのアクセスが可能になりました。
しかし、このままでは@RequestMappingメソッドの処理が行われる際に認可処理が省略されるため、誰でも画面を表示できます。

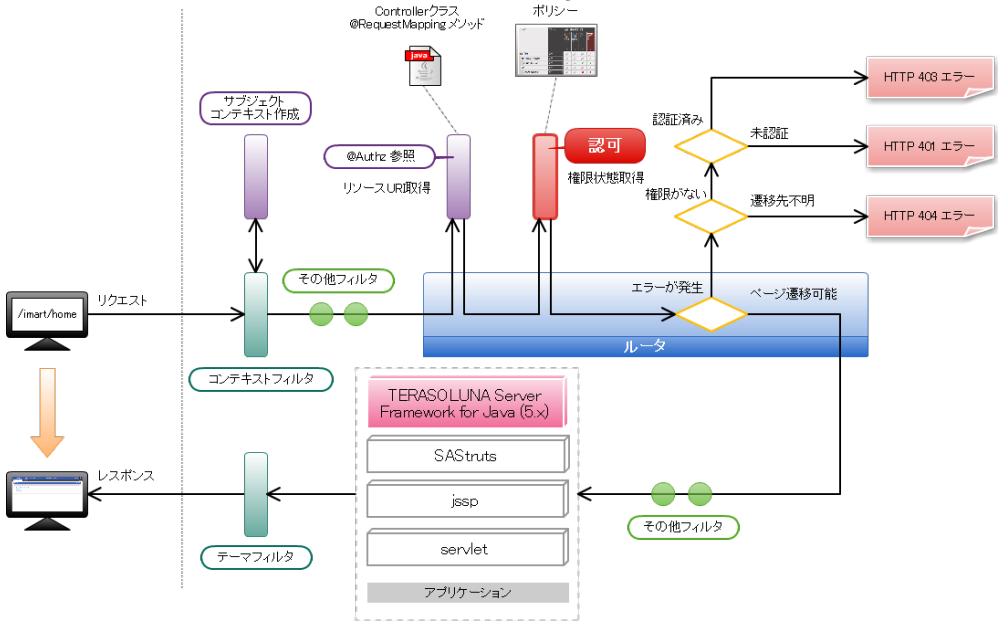
実際にシステムを運用する際は、アクセス権限を設定して特定のユーザのみに画面を表示させ、アクセスの制限をかける場合がほとんどです。
この章では、 intra-mart Accel Platform で用意されている認可機能を利用して、用意した画面を特定のユーザにのみ表示させる手順を説明します。

Controllerクラス、@RequestMappingメソッドの詳細については 「Annotated Controllers」 を参照してください。

認可とは

認可の概要

認可とは、認証機能によって特定されたユーザが要求するリソースへのアクセスを制御する機能です。
intra-mart Accel Platform では、 「誰が」「何を」「どうする」 を 「許可」「禁止」 で判定する共通的な認可機能の仕組みを提供しています。



サブジェクト

認可での 「サブジェクト」 は、「誰が」 の部分を示します。
intra-mart Accel Platform では 「ユーザ」「ロール」「会社・組織」「役職」「パブリックグループ」「パブリックグループ・役割」 の組み合わせで、権限を与える対象者を設定できます。

リソースとリソースグループ

認可での 「リソース」 は、「何を」 の部分を示します。
例えば、画面や Web サービスなどのサービス系、メニューやポータルなどのデータ系があります。
リソースは、 「リソースグループ」 によって親子階層を持たせることができます。

リソースタイプとアクション

認可での 「アクション」 は、「どうする」 の部分を示します。
アクションの内容はリソースの種類（リソースタイプ）によって決まります。
例えば、画面では 「実行」 というアクションを 1 つのみ持っています。

ポリシー

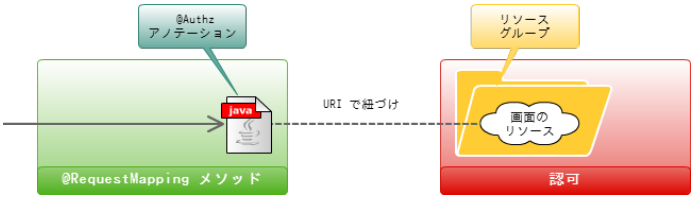
認可での 「ポリシー」 は、「許可」「禁止」 の部分を示します。
各サブジェクト・リソース・アクションの組み合わせ分、ポリシーを設定できます。

ポリシーが未設定状態の場合は、親階層のリソースグループの権限を引き継ぎます。
最上位階層のリソースグループのポリシーが未設定の場合、 「禁止」 として扱います。

画面へアクセスするための認可設定方法

権限設定の流れ

ルータでは @RequestMapping メソッドに設定された情報に基づいて認可に権限の問い合わせを行います。
問い合わせを行う際は、キーとして各リソース別に割り当てられた URI を使用します。



画面の権限設定をする場合の作業手順は以下の通りです。

1. @RequestMapping メソッドに認可を紐づける
2. 認可のリソースグループ、リソースを登録する
3. リソースに対して権限を設定する

ステップ1：@RequestMappingメソッドに認可を紐づける

認可に紐づけるため、 @RequestMapping メソッドに @Authz アノテーションを記述します。

```
@Authz(uri = "service://sample/foo", action = "execute")
@RequestMapping()
public String index() {
    return "index.jsp";
}
```

画面の場合、 uri プロパティには “service://” で始まる文字列を指定します。
ここの “service” が 「リソースタイプ」 を示す文字列で、 “service” は 「画面・処理」 を表します。

リソースタイプ “service” には、アクションとして “execute”（実行）が 1 つのみ用意されています。
そのため、 action 属性には “execute” を指定しておきます。

ステップ2：認可のリソースグループ、リソースを登録する

サンプルで作成した画面と認可を紐づけるため、認可に対して以下の構成でリソースグループとリソースを登録します。

リソースグループとリソースは 「認可設定画面から登録する方法」 と 「ジョブを利用してファイルから登録する方法」 があります。
認可設定画面からリソースを登録する場合は、 *ステップ2-1：認可設定画面を利用して認可のリソースグループ、リソースを登録する* の操作を行ってください。
ジョブを利用してファイルから登録するリソースを登録する場合は、 *ステップ2-2：ジョブを利用して認可のリソースグループ、リソースを登録する* の操作を行ってください。

ステップ2-1：認可設定画面を利用して認可のリソースグループ、リソースを登録する

テナント管理者で intra-mart Accel Platform にログインします。

```
http://<HOST>:<PORT>/<CONTEXT_PATH>/login
```

「サイトマップ」→「テナント管理」→「認可」の順にクリックします。

認可設定画面が開きますので、 「権限設定を開始する」ボタンをクリックします。

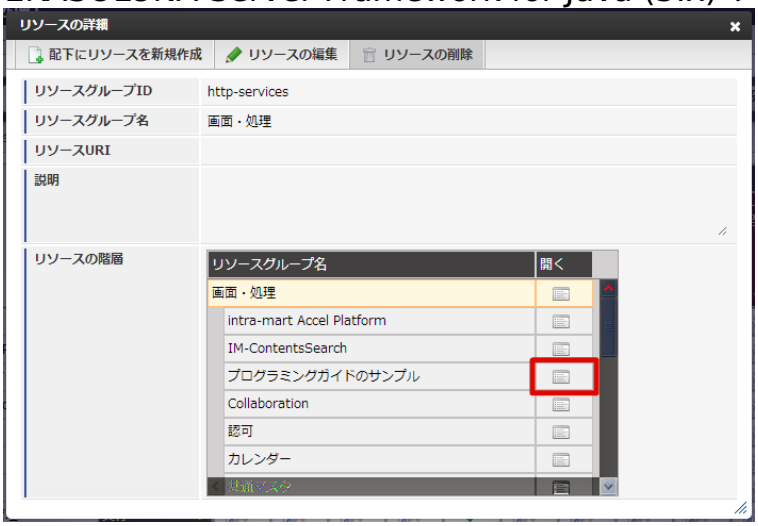
まずはリソースグループを登録します。
リソース列の 「画面・処理」 にマウスカーソルを合わせると右側にアイコンが表示されますので、アイコンをクリックします。

「リソースの詳細」 ダイアログの 「配下にリソースを新規作成」 をクリックします。

「リソースグループID」 のテキストボックスに 「guide-sample-service」 を入力します。
「リソースグループ名」 の最も上のテキストボックスに 「プログラミングガイドのサンプル」 を入力します。
「リソースURI」 と 「説明」 は未入力のままでかまいません。
「OK」 ボタンをクリックします。 これでリソースグループが登録されました。

> **コラム**
>
> 「リソースグループID」 には任意の ID を指定できます。
> 「リソースURI」 を未入力状態にして登録することで、リソースをグループ化するためのリソースグループを作成できます。
>
> 実際の開発時は、認可管理者に対して登録したリソースグループがどの画面・処理・データを表しているか明示するために、「説明」 を設定することをお勧めします。

「リソースの詳細」 ダイアログの 「リソースの階層」 から、作成した 「プログラミングガイドのサンプル」 を探します。
同じ行の右側にある 「開く」 アイコンをクリックします。
「プログラミングガイドのサンプル」 が選択されます。



> **注意**
>
> リソースグループの配下にリソースが登録されていない場合、権限設定のグリッド上には表示されません。
> 「リソースの階層」 には表示されます。

> **コラム**
>
> 「リソースの階層」 には選択されているリソース（グループ）の親階層と、子階層（1 階層）が表示されます。

次にリソースを登録します。
「リソースの詳細」 ダイアログの 「配下にリソースを新規作成」 をクリックします。

「リソースグループID」 のテキストボックスに 「guide-sample-foo-service」 を入力します。
「リソースグループ名」 の最も上のテキストボックスに 「Hello World」 を入力します。
「リソースURI」 のテキストボックスに 「service://sample/foo」 を入力します。
「説明」 は未入力のままでかまいません。
「OK」 ボタンをクリックします。 これでリソースが登録されました。

> **コラム**
>
> 「リソースグループID」 には任意の ID を指定できます。
> 「リソースURI」 にはルーティングテーブルで指定した authz タグの uri 属性と同じ値を指定します。
> 「リソースURI」 を入力して登録することで、画面などのリソースに紐づくリソースを作成できます。
>
> 実際の開発時は、認可管理者に対して登録したリソースがどの画面・処理・データを表しているか明示するために、「説明」 を設定することをお勧めします。

「リソースの詳細」 ダイアログの右上の 「×」 アイコンをクリックしてダイアログを閉じます。

これで認可に対して画面の表示に必要なリソースグループとリソースが登録できました。
次にステップ3 の操作を行います。

ステップ2-2：ジョブを利用して認可のリソースグループ、リソースを登録する

空の＜authz-resource-group.xml＞ファイルを作成して、以下を入力し保存します。

```
<?xml version="1.0" encoding="UTF-8"?>
<root
 xmlns="http://www.intra-mart.jp/authz/imex/resource-group"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intra-mart.jp/authz/imex/resource-group authz-resource-group.xsd">

 <authz-resource-group id="guide-sample-service">
  <display-name>
   <name locale="ja">プログラミングガイドのサンプル</name>
  </display-name>
  <resource-group-description>
   <description locale="ja">プログラミングガイドのサンプルです。</description>
  </resource-group-description>
  <parent-group id="http-services" />
 </authz-resource-group>

</root>
```

注意

文字コードを UTF-8 にして保存してください。

コラム

authz-resource-group タグの id 属性には任意の ID を指定できます。
parent-group タグの id 属性には “http-services” を指定してください。

resource-group-description タグにはリソースグループの説明を指定できます。 タグは省略可能です。
実際の開発時は、認可管理者に対して登録したリソースグループがどの画面・処理・データを表しているか明示するために、説明を指定することをお勧めします。

次に、空の＜authz-resource.xml＞ファイルを作成して、以下を入力し保存します。

```
<?xml version="1.0" encoding="UTF-8"?>
<root
 xmlns="http://www.intra-mart.jp/authz/imex/resource"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intra-mart.jp/authz/imex/resource authz-resource.xsd">

 <authz-resource id="guide-sample-foo-service" uri="service://sample/foo">
  <display-name>
   <name locale="ja">Hello World</name>
  </display-name>
  <resource-description>
   <description locale="ja">プログラミングガイドのサンプルです。</description>
  </resource-description>
  <parent-group id="guide-sample-service" />
 </authz-resource>

</root>
```

注意

文字コードを UTF-8 にして保存してください。

> **コラム**
>
> authz-resource タグの id 属性には任意の ID を指定できます。
> uri 属性には@RequestMappingメソッドで指定した @Authz アノテーションの uri プロパティと同じ値を指定します。
> parent-group タグの id 属性には、先ほど作成した authz-resource-group の id 属性と同じ値を指定します。
>
> resource-description タグにはリソースの説明を指定できます。 タグは省略可能です。
> 実際の開発時は、認可管理者に対して登録したリソースがどの画面・処理・データを表しているか明示するために、説明を指定することをお勧めします。

作成した＜authz-resource-group.xml＞と＜authz-resource.xml＞のファイルを、 %PUBLIC_STORAGE_PATH% 直下に配置します。

テナント管理者で intra-mart Accel Platform にログインします。

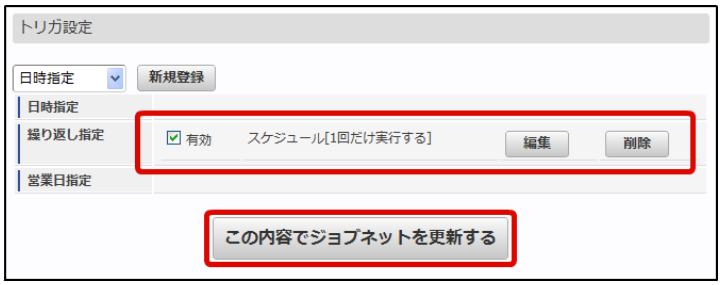
```
http://<HOST>:<PORT>/<CONTEXT_PATH>/login
```

「サイトマップ」→「テナント管理」→「ジョブ管理」→「ジョブネット設定」の順にクリックします。

「ジョブネット一覧」 から 「テナントマスタ」ー「インポート」ー「認可(リソースグループ)インポート」 を選択します。
画面下にある 「このジョブネットを編集する」 ボタンをクリックします。

「トリガ設定」 のプルダウンから 「繰り返し指定」 を選択して 「新規登録」 ボタンをクリックします。
「1回だけ実行する」 を選択状態にして 「決定」 ボタンをクリックします。

「有効」 のチェックボックスをクリックして、 「この内容でジョブネットを更新する」 ボタンをクリックします。
確認メッセージで 「決定」 ボタンをクリックします。



「ジョブネット一覧」 から 「テナントマスタ」ー「インポート」ー「認可(リソース)インポート」 を選択します。
同じ操作を行い、ジョブネットを更新します。

「サイトマップ」→「テナント管理」→「ジョブ管理」→「ジョブネットモニタ」の順にクリックします。
一覧に 「認可(リソースグループ)インポート」 「認可(リソース)インポート」 の2行が表示され、 「成功」 になっていることを確認します。

これで認可に対して画面の表示に必要なリソースグループとリソースが登録できました。
次にステップ3 の操作を行います。

ステップ3：リソースに対して権限を設定する

「サイトマップ」→「テナント管理」→「認可」の順にクリックします。
認可設定画面が開きますので、画面左上の 「検索」 アイコンをクリックします。
「リソース（縦軸）の絞込」 のテキストボックスに 「Hello」 を入力して 「検索」 ボタンをクリックします。

リソース列の 「画面・処理」 の下に 「プログラミングガイドのサンプル」、さらにその下に 「Hello World」 が表示されていることが確認できます。
これでこのサンプル画面に対するリソースの登録が完了しました。

この状態で以下のURLへアクセスしてみます。

```
http://<HOST>:<PORT>/<CONTEXT_PATH>/sample/hello
```

HTTP 403 でアクセスできなくなったことが確認できました。

それでは最後にこの認可設定画面から、 「Hello World」 に対して認可設定を行います。

「サイトマップ」→「テナント管理」→「認可」の順にクリックします。
認可設定画面が開きますので、画面左上の 「検索」 アイコンをクリックします。
「リソース（縦軸）の絞込」 のテキストボックスに 「Hello」 を入力して 「検索」 ボタンをクリックします。

「権限設定を開始する」ボタンをクリックします。
「Hello World」 の行と 「テナント管理者」 の列が交わるセルをクリックして、緑色のチェックに変更します。

この状態で、もう一度以下のURLへアクセスしてみます。

```
http://<HOST>:<PORT>/<CONTEXT_PATH>/sample/tgfw/hello
```

今度はアクセスできることが確認できました。
この場合、「テナント管理者」ロールを持つユーザのみがこのサンプル画面を表示できます。

> **コラム**
>
> このチュートリアルでは、下記のポイントを確認しました。
>
> - 画面へのアクセスを制御するために、認可を利用しました。
> - 画面の権限設定を認可で利用できるようにするために、認可と画面を紐づけるリソースとリソースグループを認可に登録しました。
> - 管理者が認可設定画面を開き、アクセスを制御したい画面のリソースに対して権限を設定しました。

メニュー設定と認可についての注意事項

作成した@RequestMappingメソッドをメニューに表示させるには、メニューアイテムとしてメニューに登録する必要があります。
メニュー登録の詳細については、「テナント管理者操作ガイド」の「メニューを設定する」を参照してください。

メニュー表示時に行う認可チェックでは、設定したメニューアイテムのURLのみが使用され、呼び出し方法や引数は使われません。
そのため、メニューアイテムのURLに対応させる@RequestMappingメソッドは、@RequestMappingのvalueのみを使用し、URLから一意に@RequestMappingメソッドが決定できるようにしてください。

ここでは、メニューの設定でURLと引数を使っているパターンを例に挙げ、URLのみに修正します。

- 修正前
  メニューに引数を使っていて、@RequestMappingメソッドにparamsを使用して対応させています。
  メニュー表示時の解決にはURLのみが使用されるので、viewDog()でなく、view()の@Authzで認可チェックが行われてしまいます。
  メニューアイテムの設定内容

URL	pets	
引数	キー	kind
	値	dog

  Controllerクラス

```
@RequestMapping(value = "pets", params = "kind=dog")
@Authz(uri = "service://pets/dog")
public String viewDog() {
    return "dog.jsp";
}

@RequestMapping(value = "pets")
@Authz(uri = "service://pets")
public String view() {
    return "pets.jsp";
}
```

- 修正後
  URLと@RequestMappingのvalueのみでメニューと@RequestMappingメソッドを一意に対応できるように修正します。
  メニューアイテムの設定内容

URL	pets/dog
引数	なし

Controllerクラス

```
// paramsを使わず、valueに含めるように修正します。valueの値は、valueだけで見て他の@ReuqestMappingと重複しないように決定します。
@RequestMapping(value = "pets/dog")
@Authz(uri = "service://pets/dog")
public String viewDog() {
    return "dog.jsp";
}

@RequestMapping(value = "pets")
@Authz(uri = "service://pets")
public String view() {
    return "pets.jsp";
}
```

また、メニュー表示時に行う認可チェックは常に GETリクエスト が利用されます。
そのため、該当するメソッドに@RequestMapping(method = RequestMethod.POST) のようにPOSTのみを指定している場合、認可チェックが正しく行えません。

- 修正前
  @RequestMappingのmethodに POST のみを定義しています。
  メニュー表示時の認可チェックは常に GETリクエスト が利用されます。
  Controllerクラスには対応するメソッドがないため、認可チェックが正しく行えません。

  Controllerクラス

  ```
  @RequestMapping(method = RequestMethod.POST)
  @Authz(uri = "service://sample/foo")
  public String view() {
      return "view.jsp";
  }
  ```

- 修正後
  @RequestMappingにはGETリクエストも含めるように修正します。
  ここではmethodを省略し、GET, POSTの両方を受け付けるようにしています。

  Controllerクラス

  ```
  @RequestMapping()
  @Authz(uri = "service://sample/foo")
  public String view() {
      return "view.jsp";
  }
  ```

アクセスコンテキスト

仕様

アクセスコンテキストの仕様については、「 アクセスコンテキスト仕様書 」を参照してください。

定義済みアクセスコンテキスト

intra-mart Accel Platform ではあらかじめいくつかの種別のアクセスコンテキストが定義されています。
それぞれのアクセスコンテキストについては、「 アクセスコンテキスト仕様書 」-「 標準コンテキスト 」を参照してください。

プログラミング方法

ここでは、アクセスコンテキストの簡単な利用方法について説明します。

新しいアクセスコンテキストを作成する方法、および、アクセスコンテキストの詳細な利用方法については、「アクセスコンテキスト 拡張プログラミングガイド」を参照してください。

項目
- アクセスコンテキストの基本的な利用方法
  - アカウントコンテキストの状態アクセスユーティリティ
  - 認証状況を問い合わせる
  - ユーザ種別を問い合わせる
- 定義済みアクセスコンテキストの利用方法
  - アカウントコンテキストを利用する
  - ユーザコンテキストを利用する
  - ユーザコンテキストの所属組織を切り替える
  - 一時的に実行ユーザを変更して処理を実行する

アクセスコンテキストの基本的な利用方法

アクセスコンテキストを取得するためには、 `Contexts` API を利用します。
利用したいアクセスコンテキストの種別の短縮名を確認し、以下の方法で取得してください。

```
Contexts.get(<コンテキスト種別>)
```

コンテキスト種別は、利用したいアクセスコンテキストインタフェースのクラスオブジェクトです。
利用したいアクセスコンテキストのインタフェースを確認し、引数に指定して実行します。

以下にアカウントコンテキストを取得する例を示します。

```
// アカウントコンテキストを取得する。
AccountContext accountContext = Contexts.get(AccountContext.class);

// ユーザコードを取得する。
String userCd = accountContext.getUserCd();

// ユーザコンテキストを取得する。
UserContext userContext = Contexts.get(UserContext.class);
```

アカウントコンテキストの状態アクセスユーティリティ

よく利用される機能として、アカウントコンテキストの認証状況などを問い合わせることがあります。
そのためには、アカウントコンテキストの情報を問い合わせるためのユーティリティを利用できます。

```
// 認証状況を問い合わせる
boolean authenticated = ContextStatus.isAuthenticated();

if (authenticated) {
 // 認証済みユーザの処理
} else {
 // 未認証ユーザの処理
}
```

認証状況を問い合わせる

ログインしているかどうかは、以下のいずれかの方法により確認できます。

- アカウントコンテキストの認証状況を問い合わせる。

```
AccountContext accountContext = Contexts.get(AccountContext.class);
boolean authenticated = accountContext.isAuthenticated();
```

- ユーティリティを利用して認証状況を問い合わせる。

```
boolean authenticated = ContextStatus.isAuthenticated();
```

intra-mart Accel Platform では、ログインしていないユーザの場合も、ユーザコードが設定されています。
このユーザコードは、ログやデータベースの更新者等、アプリケーションに影響がない範囲で利用されることを想定しています。

ログインしているかどうかは、必ず上記の方法で確認してください。

ユーザ種別を問い合わせる

アクセスしているユーザが、一般ユーザなのか、または、システム管理者なのかは、以下のいずれかの方法により確認できます。

- アカウントコンテキストのユーザ種別を取得する。

```
AccountContext accountContext = Contexts.get(AccountContext.class);
UserType userType= accountContext.getUserType();

// システム管理者かどうか
boolean isAdministrator = UserType.ADMINISTRATOR  == userType;

// 一般ユーザかどうか
boolean isUser = UserType.USER  == userType;
```

- ユーティリティを利用してシステム管理者かどうかを問い合わせる。

```
boolean isAdministrator = ContextStatus.isAdministrator();
```

intra-mart Accel Platform では、システム管理者以外は一般ユーザとして扱います。
未認証ユーザの場合もユーザ種別は一般ユーザです。

定義済みアクセスコンテキストの利用方法

アカウントコンテキストを利用する

以下にアカウントコンテキストからユーザロケールを取得する例を示します。

```
AccountContext accountContext = Contexts.get(AccountContext.class);
Locale locale = accountContext.getLocale();

// メッセージを取得する。
String message = MessageManager.getInstance().getMessage(locale, "I.IWP.CERTIFICATION.SECURITYLOG.00200");
System.out.println("メッセージ = " + message);
```

アカウントコンテキストのロケールは、アカウント情報にロケールが設定されていない場合テナントのロケールが取得できます。
個人設定でロケールを変更し、ロケールが変更されることを確認してみてください。

多くのAPIでは、引数を省略することでアカウントコンテキストの情報を参照します。
上記の例は、以下のコードと同じ結果です。

```
// メッセージを取得する。
String message = MessageManager.getInstance().getMessage("I.IWP.CERTIFICATION.SECURITYLOG.00200");
```

ユーザコンテキストを利用する

以下にユーザコンテキストからプロファイル情報を取得する例を示します。

```
UserContext userContext = Contexts.get(UserContext.class);

// ユーザ名を取得します。
String userName = userContext.getUserProfile().getUserName();
```

このコードにより、現在アクセスしているユーザのユーザ名が取得できます。
未認証ユーザの場合は「ゲスト」が取得できます。

また、ユーザコンテキストからカレント組織を取得する例を示します。

```
UserContext userContext = Contexts.get(UserContext.class);

Department department = userContext.getCurrentDepartment();
if (department != null) {
  String departmentName = department.getDepartmentFullName();
  System.out.println("カレント組織名 = " + departmentName);
}
```

このコードにより、現在アクセスしている組織のフル名称が取得できます。
複数の所属先を持つユーザの場合、ヘッダのユーザ名をクリックして所属組織を切り替えることで、取得できる名称も変わりますので、確認してみてください。

ユーザコンテキストの所属組織を切り替える

以下にプログラムからユーザコンテキストの所属組織を切り替える例を示します。

```
Map<String, String> resource = new HashMap<String, String>();
resource.put("currentCompanyCd", companyCd);
resource.put("currentDepartmentSetCd", departmentSetCd);
resource.put("currentDepartmentCd", departmentCd);

Lifecycle lifecycle = LifecycleFactory.getLifecycle();
lifecycle.switchTo(new Resource("platform.current.department.switch", resource));
```

このコードにより、所属組織の切り替えを行うことができます。
所属組織の切り替えを行う場合は、以下の点に注意してください。

- リソースIDには「 `platform.current.department.switch` 」を指定してください。
- リソース情報（`Map`）には切り替え先の組織を指定してください。
  また、所属組織の切り替えを行うには、切り替えを行うユーザが所属している組織を指定する必要があります。
- 切り替え先の組織には、会社コード（`currentCompanyCd`）、組織セットコード（`currentDepartmentSetCd`）、組織コード（`currentDepartmentCd`）の３つを必ず指定してください。

一時的に実行ユーザを変更して処理を実行する

以下に一時的にユーザとカレント組織を変更して処理を実行する例を示します。

```
// 切り替えるユーザのユーザコードを指定します。
String userCd = "aoyagi";

// IM-共通マスタの組織ビジネスキーを指定します。
DepartmentBizKey departmentBizKey = new DepartmentBizKey();
department.setCompanyCd("comp_sample_01");
department.setDepartmentSetCd("comp_sample_01");
department.setDepartmentCd("dept_sample_21");

// 一時的にユーザを切り替えて処理を実行します。
UserSwitcher.switchTo(userCd, departmentBizKey, new UserSwitchProcedure() {

  public void process() throws UserSwitchException {
    try {

      // ユーザを切替えた場合の処理を記述します。
      doSomething();

    } catch (SomethingException e) {
      throw new UserSwitchException(e);
    }
  }

});
```

このコードにより、ジョブ実行環境での特定のユーザによる処理の実行や、Web実行環境でのログインユーザとは別のユーザによる処理の実行（ユーザを切り替えて実行）ができます。
ユーザを切り替えることで、そのユーザの認可情報やライセンス情報に従った処理を実行できます。

ユーザ切替処理の詳細は、「UserSwitcher クラスの APIドキュメント」を参照してください。

アクセスコンテキストとは

アクセスコンテキストとは、intra-mart Accel Platform に対して現在アクセスしている処理実行者（以下、利用者）に関連する共有情報を保持するオブジェクトです。

利用者のアクセスの開始から終了まで、利用者の情報や状態を保持します。
それらの情報は、システムおよびアプリケーションから自由に参照されます。
情報の種別ごとに複数のアクセスコンテキストが定義され、目的に応じて必要なアクセスコンテキストを利用可能です。

詳細は、「 アクセスコンテキスト仕様書 」を参照してください。

アクセスコンテキストの情報は、 intra-mart Accel Platform の基盤機能、または、アクセスコンテキストの情報を管理するモジュールによって設定されます。それ以外のモジュールからは参照のみ可能です。

intra-mart Accel Platform が提供するアクセスコンテキストには、以下のような種類があります。

- アカウントコンテキスト
- クライアントコンテキスト
- ユーザコンテキスト
- ジョブスケジューラコンテキスト
- 認可サブジェクトコンテキスト（システム用）

それぞれの詳細については「 アクセスコンテキスト仕様書 」-「 標準コンテキスト 」を参照してください。

認証機能

項目
- 認証機能とは
- プログラミング方法

認証機能とは

認証機能とはユーザ情報を検証して、システムへのログインやユーザの有効性確認を行うための機能です。

認証機能では、いくつかのプログラミングインタフェースが用意されています。
これにより、要件に応じて認証方式の変更や認証サーバとの連携が可能です。

プログラミング方法

認証機能のプログラミング方法については、「認証プログラミングガイド」を参照してください。

国際化

項目
- 概要
  - intra-mart Accel Platform が国際化に対応する理由
  - 国際化対応の要素と理由
- 国際化機能の仕様
- 国際化プログラミングのサンプル

概要

intra-mart Accel Platform が国際化に対応する理由

システムの国際化とは、様々な言語や地域の人たちが自分たちに合った言語やタイムゾーンでシステムを利用できることを意味します。
グローバル化を進める企業では、様々な言語や地域の人たちが協力して仕事をすることになるため、企業が利用するシステムも国際化に対応している必要があります。

国際化対応の要素と理由

intra-mart Accel Platform は言語、タイムゾーン、日付と時刻の形式について国際化対応しています。

- 多言語

  慣れない言語で画面を表示すると、そもそも表示内容を把握できなかったり、作業効率が落ちたり、ミスが多くなるため、日常的に使用している言語で画面を表示できることが必要です。

- タイムゾーン

  ユーザは現地時間に合わせて考え、行動します。そのため、ユーザが参照する日時データは、ユーザのタイムゾーンにおける日時データであることが要求されます。

- 日付と時刻の形式

  日付と時刻の形式は文化圏や地域によって異なっており、ユーザの地域における形式であることが必要です。
  地域に合った形式を提供しない場合、思わぬ誤解が生じる危険性があります。
  例えば、「12/09/10」と表示された日付は、地域によって以下の意味になります。

地域	表示結果
日本	2012年9月10日
英国	2010年9月12日
米国	2010年12月9日

国際化機能の仕様

多言語

項目
- 概要
  - 多言語対応とは？
  - intra-mart Accel Platform の多言語対応
- 言語の定義
  - 初期状態の言語の定義
- メッセージの定義
  - メッセージプロパティファイル

概要

多言語対応とは？

多言語対応とは、システムが 1 つの言語だけに対応しているのではなく、複数の言語を使い分けられることを意味します。

intra-mart Accel Platform の多言語対応

intra-mart Accel Platform における多言語対応の構成要素は、以下の４つです。

- 言語の定義

  intra-mart Accel Platform で利用する言語を定義しています。

- メッセージの定義

  ユーザに見せるためのメッセージは、intra-mart Accel Platform で定義している全言語について翻訳を用意しています。
  メッセージは、「言語 ID」と「メッセージコード」で管理しています。

- 言語解決

  ユーザは、使用したい言語を intra-mart Accel Platform に登録できます。
  登録しない場合、自動的に解決された言語が適用されます。

言語の解決順序については、*定義済みアクセスコンテキスト* を参照してください。

- メッセージの取得

intra-mart Accel Platform は、多言語対応されたメッセージを取得するための API (MessageManager) を提供しています。
MessageManager は「メッセージコード」と「ユーザの言語」から、該当するメッセージを取得します。
MessageManager の使い方については、*多言語化されたメッセージを取得する* を参照してください。

言語の定義

初期状態の言語の定義

intra-mart Accel Platform が初期状態で定義している言語は、以下の通りです。

言語	言語 ID
英語	en
日本語	ja
中国語（簡体字）	zh_CN

- 言語マスタファイル

intra-mart Accel Platform で定義されている言語は、「言語マスタファイル」という XML に記述されています。
初期状態のシステム・デフォルト言語は「英語」です。

- 設定ファイル

```
%CONTEXT_PATH%/WEB-INF/conf/locale-config/im-locale-default.xml
```

メッセージの定義

intra-mart Accel Platform におけるメッセージの構成要素は、以下の３つです。

- メッセージコード

メッセージの内容に紐付くコードです。
intra-mart Accel Platform から提供されているメッセージコードは、内部的な規約に基づいているため、コード体系を理解する必要はありません。

- 言語 ID

intra-mart Accel Platform が定義する言語は、言語 ID として言語マスタファイルに定義されています。

- 翻訳されたメッセージ

翻訳は、intra-mart Accel Platform が定義する言語分だけ用意されています。

メッセージプロパティファイル

メッセージが記述されているファイルです。

- ファイルの命名規約

```
{任意の名前} _ {言語ID} .properties
```

- 例

言語	ファイル名
英語	example_en.properties
日本語	example_ja.properties
中国語（簡体字）	example_zh_CN.properties

注意

{任意の名前} 部分にはアンダースコア(_)を入れないようにしてください。

（例） test_message_ja.properties

- ファイルの内容

```
{メッセージコード} = {翻訳されたメッセージ}
```

  - 例
    example_en.properties

```
I18N.MESSAGE.EXAMPLE=This is an example of message.
```

    example_ja.properties

```
I18N.MESSAGE.EXAMPLE=\u3053\u308c\u306f\u30e1\u30c3\u30bb\u30fc\u30b8\u306e\u4f8b\u3067\u3059\u3002
```

タイムゾーン

項目
- 概要
  - タイムゾーン対応とは？
  - intra-mart Accel Platform のタイムゾーン対応
- タイムゾーンの定義
  - 初期状態のタイムゾーンの定義
- 日時データを統一のタイムゾーンで変換して保存する
  - システム要件
  - 日時データをシステム・デフォルトのタイムゾーンで変換する
- 夏時間
  - 夏時間とは？
  - intra-mart Accel Platform の夏時間対応

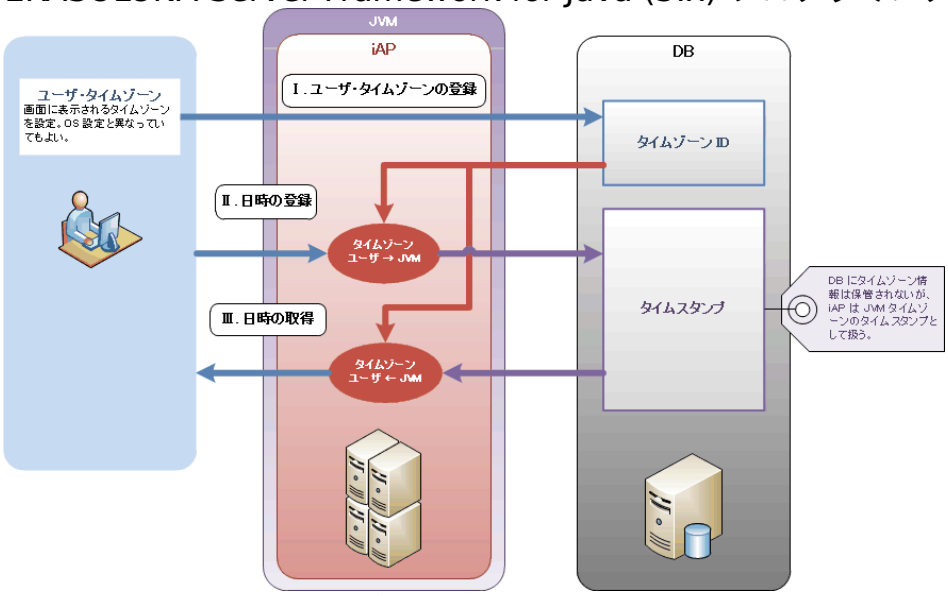
概要

タイムゾーン対応とは？

タイムゾーン対応とは、世界中で日時データを比較、変換して使えるようにするために、タイムゾーンを適切に変換する仕組みです。
この中には、システムが自動的に夏時間を認識して、日時データを変換する処理も含まれます。

ユーザが自分に合ったタイムゾーンで日時データを参照するためには、タイムゾーンをユーザごとに設定できる必要があります。

各ユーザの持つ日時データのタイムゾーンがバラバラでは、情報を有効に活用できません。
そのため、システム側では、統一のタイムゾーンによって変換された日時データを保管するようにします。



intra-mart Accel Platform のタイムゾーン対応

intra-mart Accel Platform におけるタイムゾーン対応の構成要素は、以下の５つです。

- タイムゾーンの定義

  intra-mart Accel Platform で利用するタイムゾーンを定義しています。

- 日時データを統一のタイムゾーンで変換して保存する

  日時データは、統一されたタイムゾーンで変換して DB に保存します。

- タイムゾーン解決

  ユーザは、使用したいタイムゾーンを intra-mart Accel Platform に登録できます。
  登録しない場合、自動的に解決されたタイムゾーンが適用されます。
  タイムゾーンの解決順序については、*定義済みアクセスコンテキスト* を参照してください。

- 日時データをユーザのタイムゾーンで変換する

  日時データを画面に表示する場合、ユーザのタイムゾーンで変換されている必要があります。
  ユーザのタイムゾーンに変換する方法については、*ユーザのタイムゾーン、日付と時刻の形式を利用する* を参照してください。

- 夏時間対応

タイムゾーンの定義

初期状態のタイムゾーンの定義

intra-mart Accel Platform が初期状態で定義しているタイムゾーンは、以下を参照してください。

初期状態のタイムゾーンの定義

項目
- 初期状態のタイムゾーン一覧

初期状態のタイムゾーン一覧

タイムゾーン ID	（時差）地域名
Pacific/Kiritimati	(GMT+14:00) キリバス / キリスィマスィ島
Pacific/Enderbury	(GMT+13:00) フェニックス諸島 / エンダーバリ島
Pacific/Tongatapu	(GMT+13:00) トンガ / トンガタプ島
Pacific/Chatham	(GMT+12:45) ニュージーランド / チャタム諸島
Asia/Kamchatka	(GMT+12:00) ロシア / 極東連邦管区 / カムチャツカ半島
Pacific/Auckland	(GMT+12:00) ニュージーランド / オークランド
Pacific/Fiji	(GMT+12:00) フィジー

タイムゾーン ID	（時差）地域名
Pacific/Norfolk	(GMT+11:30) ノーフォーク島
Pacific/Guadalcanal	(GMT+11:00) ソロモン諸島 / ガダルカナル島
Australia/Lord_Howe	(GMT+10:30) オーストラリア / ニューサウスウェールズ州 / ロード・ハウ島
Australia/Queensland	(GMT+10:00) オーストラリア / クィーンズランド州
Australia/NSW	(GMT+10:00) オーストラリア / ニューサウスウェールズ州
Australia/South	(GMT+09:30) オーストラリア / 南オーストラリア州
Australia/North	(GMT+09:30) オーストラリア / ノーザンテリトリー準州
Asia/Seoul	(GMT+09:00) 大韓民国 / ソウル
Asia/Tokyo	(GMT+09:00) 日本 / 東京
Asia/Hong_Kong	(GMT+08:00) 中華人民共和国 / 香港
Asia/Kuala_Lumpur	(GMT+08:00) マレーシア / クアラルンプール
Asia/Manila	(GMT+08:00) フィリピン / マニラ
Asia/Shanghai	(GMT+08:00) 中華人民共和国 / 上海
Asia/Singapore	(GMT+08:00) シンガポール / シンガポール
Asia/Taipei	(GMT+08:00) 台湾 / 台北
Antarctica/Casey	(GMT+08:00) ヴィンセンス湾 / ケーシー基地
Asia/Bangkok	(GMT+07:00) タイ王国 / バンコク
Asia/Jakarta	(GMT+07:00) インドネシア / ジャカルタ
Asia/Saigon	(GMT+07:00) ベトナム / ホーチミン
Asia/Rangoon	(GMT+06:30) ミャンマー / ヤンゴン
Asia/Dacca	(GMT+06:00) バングラデシュ / ダッカ
Asia/Katmandu	(GMT+05:45) ネパール / カトマンズ
Asia/Calcutta	(GMT+05:30) インド / コルカタ
Asia/Colombo	(GMT+05:30) スリランカ / コロンボ
Asia/Karachi	(GMT+05:00) パキスタン / シンド州 / カラーチー
Asia/Tashkent	(GMT+05:00) ウズベキスタン / タシュケント州 / タシュケント
Asia/Yekaterinburg	(GMT+06:00) ロシア / ウラル連邦管区 / エカテリンブルク
Asia/Kabul	(GMT+04:30) アフガニスタン / カーブル
Asia/Dubai	(GMT+04:00) アラブ首長国連邦 / ドバイ
Asia/Tbilisi	(GMT+04:00) グルジア / トビリシ
Asia/Tehran	(GMT+03:30) イラン / テヘラン州 / テヘラン
Africa/Nairobi	(GMT+03:00) ケニア / ナイロビ
Asia/Baghdad	(GMT+03:00) イラク / バグダード
Asia/Kuwait	(GMT+03:00) クウェート / クウェート
Asia/Riyadh	(GMT+03:00) サウジアラビア / ナジュド地方 / リヤド
Europe/Moscow	(GMT+04:00) ロシア / 中央連邦管区 / モスクワ
Africa/Cairo	(GMT+02:00) エジプト / カイロ
Africa/Johannesburg	(GMT+02:00) 南アフリカ共和国 / ハウテン州 / ヨハネスブルグ
Asia/Jerusalem	(GMT+02:00) イスラエル / エルサレム
Europe/Athens	(GMT+02:00) ギリシャ / アテネ

タイムゾーン ID	（時差）地域名
Europe/Bucharest	(GMT+02:00) ルーマニア / ワラキア / ブカレスト
Europe/Helsinki	(GMT+02:00) フィンランド / ヘルシンキ
Europe/Istanbul	(GMT+02:00) トルコ / イスタンブル県 / イスタンブル
Europe/Minsk	(GMT+03:00) ベラルーシ / ミンスク
Europe/Amsterdam	(GMT+01:00) オランダ / アムステルダム
Europe/Stockholm	(GMT+01:00) スウェーデン / ストックホルム県 / ストックホルム
Europe/Berlin	(GMT+01:00) ドイツ / ベルリン
Europe/Brussels	(GMT+01:00) ベルギー / ブリュッセル
Europe/Paris	(GMT+01:00) フランス / イル＝ド＝フランス地域圏 / パリ
Europe/Prague	(GMT+01:00) チェコ / プラハ
Europe/Rome	(GMT+01:00) イタリア / ラツィオ州 / ローマ
Europe/Dublin	(GMT+00:00) アイルランド / ダブリン
Europe/Lisbon	(GMT+00:00) ポルトガル / リスボン
Europe/London	(GMT+00:00) イギリス / ロンドン
GMT	(GMT+00:00) GMT
UTC	(GMT+00:00) UTC
Atlantic/Cape_Verde	(GMT-01:00) カーボベルデ
Atlantic/South_Georgia	(GMT-02:00) サウスジョージア・サウスサンドウィッチ諸島
America/Buenos_Aires	(GMT-03:00) アルゼンチン / ブエノスアイレス
America/Sao_Paulo	(GMT-03:00) ブラジル / サンパウロ州 / サンパウロ
America/St_Johns	(GMT-03:30) カナダ / ニューファンドランド・ラブラドール州 / セント・ジョンズ
America/Halifax	(GMT-04:00) カナダ / ノバスコシア州 / ハリファックス
America/Puerto_Rico	(GMT-04:00) 西インド諸島 / プエルトリコ
America/Santiago	(GMT-04:00) チリ / サンティアゴ
Atlantic/Bermuda	(GMT-04:00) バミューダ諸島
America/Caracas	(GMT-04:30) ベネズエラ / カラカス
America/Bogota	(GMT-05:00) コロンビア / ボゴタ
America/Indianapolis	(GMT-05:00) アメリカ合衆国 / インディアナ州 / インディアナポリス
America/Lima	(GMT-05:00) ペルー / リマ
America/New_York	(GMT-05:00) アメリカ合衆国 / ニューヨーク州 / ニューヨーク
America/Panama	(GMT-05:00) パナマ
America/Chicago	(GMT-06:00) アメリカ合衆国 / イリノイ州 / シカゴ
America/El_Salvador	(GMT-06:00) エルサルバドル
America/Mexico_City	(GMT-06:00) メキシコ / メキシコ連邦区 / メキシコシティ
America/Denver	(GMT-07:00) アメリカ合衆国 / コロラド州 / デンバー
America/Phoenix	(GMT-07:00) アメリカ合衆国 / アリゾナ州 / フェニックス
America/Los_Angeles	(GMT-08:00) アメリカ合衆国 / カリフォルニア州 / ロサンゼルス
America/Tijuana	(GMT-08:00) メキシコ / バハ・カリフォルニア州 / ティファナ
America/Anchorage	(GMT-09:00) アメリカ合衆国 / アラスカ州 / アンカレッジ
Pacific/Honolulu	(GMT-10:00) アメリカ合衆国 / ハワイ州 / ホノルル

タイムゾーン ID	（時差）地域名
Pacific/Niue	(GMT-11:00) ニウエ
Pacific/Pago_Pago	(GMT-11:00) アメリカ領サモア / パゴパゴ

- タイムゾーンマスタファイル

  intra-mart Accel Platform で定義されているタイムゾーンは、「タイムゾーンマスタファイル」という XML に記述されています。
  初期状態のシステム・デフォルト・タイムゾーンは、JDK のタイムゾーンです。

  - 設定ファイル

    ```
    %CONTEXT_PATH%/WEB-INF/conf/time-zone-config/im-time-zone-config.xml
    ```

日時データを統一のタイムゾーンで変換して保存する

システム要件

システム側で、日時データを同じタイムゾーンのデータとして処理するために、intra-mart Accel Platform の稼働環境には以下の制約があります。

- **日時データを保存する DB データ型には TIMESTAMP 型を使用する**

  SQL Server では datetime 、または、 datetime2 を使用します。

  > **注意**
  >
  > intra-mart Accel Platform では、タイムゾーン変換した際の日時データの整合性を保証するために、日時データを格納する DB カラムは TIMESTAMP 型を推奨しています。
  > タイムゾーン付き TIMESTAMP 型は、タイムゾーン変換が DB の仕様に依存しているため、 intra-mart Accel Platform 側で日時データの整合性を保証できませんので、タイムゾーン付き TIMESTAMP 型を使用しないでください。

- **日時データは、システム・デフォルトのタイムゾーンで変換して保存する**

  システム・デフォルトのタイムゾーンは、 JDK のタイムゾーンと同じです。

- **分散環境では全ての Java VM のタイムゾーンを統一する**
- **Java VM のタイムゾーンは運用開始後に変更しない**

日時データをシステム・デフォルトのタイムゾーンで変換する

ユーザのタイムゾーン、日付と時刻の形式を利用する を参照してください。

夏時間

夏時間とは？

夏の間、太陽の出ている時間帯を有効に利用する目的で、現行の時刻に一定時間を加えたタイムゾーンを採用する制度、またはその加えられた時刻のことを意味します。
その地域の政治的な事情に依存しており、実施期間や調整時間は一定ではありません。また、実施するかどうかさえ不確定な場合があります。

intra-mart Accel Platform の夏時間対応

intra-mart Accel Platform では、Java の提供している夏時間情報を利用しており、該当するタイムゾーンには、自動的に夏時間が適用されます。

日付と時刻の形式

項目
- 概要
  - 日付と時刻の形式対応とは？
  - intra-mart Accel Platform の日付と時刻の形式対応
- 日付と時刻の形式の定義
  - 初期状態の日付と時刻の形式の定義
  - 日付と時刻の形式マスタファイル
- 解決された日付と時刻の形式で日時データを整形、解析する
- 「言語」との違い

概要

日付と時刻の形式対応とは？

日付と時刻の形式対応とは、システムが１つの形式だけに対応しているのではなく、複数の形式を使い分けられることを意味します。
日常的に使用されている日付と時刻の形式は、地域によって異なります。

- （例）地域による標準的な日付表示形式の違い

地域	標準的な日付表示形式
日本	2012/09/19
英国	19-Sep-2012
米国	Sep 19, 2012

intra-mart Accel Platform の日付と時刻の形式対応

intra-mart Accel Platform における日付と時刻の形式対応の構成要素は、以下の３つです。

- 日付と時刻の形式の定義

  intra-mart Accel Platform で利用する日付と時刻の形式を定義しています。

- 日付と時刻の形式の解決

  ユーザは、使用したい形式を intra-mart Accel Platform に登録できます。
  登録しない場合、自動的に解決された形式が適用されます。
  日付と時刻の形式の解決順序については、「 アクセスコンテキスト仕様書 」-「 アカウントコンテキスト 」を参照してください。

- 日付と時刻の形式を使って、日時データを整形、解析する

  画面から入力された日時データは、文字列で送信された場合、サーバ側では文字列を解析して日時オブジェクトを生成します。
  画面には、ユーザの「日付と時刻の形式」で整形された日時データを表示します。
  日時データを整形、解析する方法については、*ユーザのタイムゾーン、日付と時刻の形式を利用する* で説明します。

日付と時刻の形式の定義

初期状態の日付と時刻の形式の定義

intra-mart Accel Platform が初期状態で定義している日付と時刻の形式は、以下を参照してください。

初期状態の日付と時刻の形式の定義

項目
- 概要
- 英語形式
- 日本語形式
- 中国語（簡体字）形式

概要

intra-mart Accel Platform は、初期状態で「英語形式」「日本語形式」「中国語（簡体字）形式」を用意しています。
各形式は、次の６種類のフォーマットから構成されます。

フォーマット名	フォーマット ID	説明
日付（標準表示）	IM_DATETIME_FORMAT_DATE_STANDARD	日付を表示する時の標準的な形式です。
日付（簡易表示）	IM_DATETIME_FORMAT_DATE_SIMPLE	日付を簡略して表示する時に使う形式です。
日付（入力）	IM_DATETIME_FORMAT_DATE_INPUT	日付を入力する時に使う形式です。
時刻（標準表示）	IM_DATETIME_FORMAT_TIME_STANDARD	時刻を表示する時の標準的な形式です。
時刻（タイムスタンプ表示）	IM_DATETIME_FORMAT_TIME_TIMESTAMP	時刻をタイムスタンプで表示する時に使う形式です。
時刻（入力）	IM_DATETIME_FORMAT_TIME_INPUT	時刻を入力する時に使う形式です。

各フォーマットには、形式に紐付いた複数のフォーマットパターンが用意されています。

- （例）日本語形式（一部のフォーマットパターンのみを表示しています。）

フォーマット名	パターン	表示例
日付（標準表示）	yyyy'年'M'月'd'日'	2012年9月23日
日付（簡易表示）	M'月'd'日'	9月23日
日付（入力）	yyyy/MM/dd	2012/09/23
時間（標準表示）	ah:mm	午前12:00
時間（タイムスタンプ表示）	ah:mm:ss	午前12:00:00
時間（入力）	HH:mm	00:00

「フォーマット ID」とは、各フォーマットに対してユーザが持つフォーマットパターンを参照するためのキーです。
フォーマット ID の使い方については、*ユーザのタイムゾーン、日付と時刻の形式を利用する* の中で説明します。

英語形式

フォーマット名	パターン	表示例
日付（標準表示）	MMM d, yyyy	Sep 23, 2012
	MMM dd, yyyy	Sep 23, 2012
	d/M/yyyy	23/9/2012
	d/MM/yyyy	23/09/2012
	dd/MM/yyyy	23/09/2012
	d-MMM-yyyy	23-Sep-2012
	dd-MMM-yyyy	23-Sep-2012
	d MMM, yyyy	23 Sep, 2012
	dd MMM, yyyy	23 Sep, 2012
	d MMM yyyy	23 Sep 2012
	dd MMM yyyy	23 Sep 2012
	yyyy-MM-dd	2012-09-23
日付（簡易表示）	MMM d	Sep 23
	MMM dd	Sep 23
	d/M	23/9
	d/MM	23/09
	dd/MM	23/09
	d-MMM	23-Sep
	dd-MMM	23-Sep
	d MMM	23 Sep

フォーマット名	パターン	表示例
	dd MMM	23 Sep
	MM-dd	09-23
日付（入力）	yyyy/MM/dd	2012/09/23
時間（標準表示）	h:mm a	12:00 AM
	hh:mm a	12:00 AM
	H:mm	0:00
	HH:mm	00:00
時間（タイムスタンプ表示）	h:mm:ss a	12:00:00 AM
	hh:mm:ss a	12:00:00 AM
	H:mm:ss	0:00:00
	HH:mm:ss	00:00:00
時間（入力）	HH:mm	00:00

日本語形式

フォーマット名	パターン	表示例
日付（標準表示）	yyyy’年’M’月’d’日’	2012年9月23日
	yyyy’年’MM’月’dd’日’	2012年09月23日
	yyyy/M/d	2012/9/23
	yyyy/MM/dd	2012/09/23
	yyyy-MM-dd	2012-09-23
日付（簡易表示）	M’月’d’日’	9月23日
	MM’月’dd’日’	09月23日
	M/d	9/23
	MM/dd	09/23
	MM-dd	09-23
日付（入力）	yyyy/MM/dd	2012/09/23
時間（標準表示）	ah:mm	午前12:00
	ahh:mm	午前12:00
	H:mm	0:00
	HH:mm	00:00
時間（タイムスタンプ表示）	ah:mm:ss	午前12:00:00
	ahh:mm:ss	午前12:00:00
	H:mm:ss	0:00:00
	HH:mm:ss	00:00:00
時間（入力）	HH:mm	00:00

中国語（簡体字）形式

フォーマット名	パターン	表示例
日付（標準表示）	yyyy’年’M’月’d’日’	2012年9月23日
	yyyy’年’MM’月’dd’日’	2012年09月23日
	yyyy/M/d	2012/9/23

フォーマット名	パターン	表示例
	yyyy/MM/dd	2012/09/23
	yyyy-M-d	2012-9-23
	yyyy-MM-dd	2012-09-23
	d MMM yyyy	23 九月 2012
	dd MMM yyyy	23 九月 2012
日付（簡易表示）	M’月’d’日’	9月23日
	MM’月’dd’日’	09月23日
	M/d	9/23
	MM/dd	09/23
	M-d	9-23
	MM-dd	09-23
	d MMM	23 九月
	dd MMM	23 九月
日付（入力）	yyyy/MM/dd	2012/09/23
時間（標準表示）	ah:mm	上午12:00
	ahh:mm	上午12:00
	H:mm	0:00
	HH:mm	00:00
時間（タイムスタンプ表示）	ah:mm:ss	上午12:00:00
	ahh:mm:ss	上午12:00:00
	H:mm:ss	0:00:00
	HH:mm:ss	00:00:00
時間（入力）	HH:mm	00:00

日付と時刻の形式マスタファイル

intra-mart Accel Platform で定義されている日付と時刻の形式は、「日付と時刻の形式マスタファイル」という XML に記述されています。
初期状態のシステム・デフォルトの形式は、「英語形式」です。

- 設定ファイル

```
%CONTEXT_PATH%/WEB-INF/conf/date-time-format-config/im-date-time-format-config.xml
```

解決された日付と時刻の形式で日時データを整形、解析する

画面から入力された日時データは、解決された「日付と時刻の形式」で解析されます。
画面には、ユーザの「日付と時刻の形式」で整形された日時データを表示します。
日時データを整形、解析する方法については、*ユーザのタイムゾーン、日付と時刻の形式を利用する* で説明します。

「言語」との違い

日付と時刻の形式は、言語とは別に登録できます。

- （例）ユーザ情報の「言語」に「日本語」、「日付と時刻の形式」に「英語形式」を登録した場合

  「現在時刻」というキャプションは日本語で表示され、日付と時刻は英語形式の「MMM d, yyyy h:mm a」で表示されています。

国際化プログラミングのサンプル

多言語化されたメッセージを取得する

項目

- 概要
- MessageManager によるメッセージの取得
  - メッセージコードを指定してメッセージを取得する
  - メッセージコードと言語を指定してメッセージを取得する
- タグによるメッセージの取得
  - メッセージコードを指定してメッセージを取得する
  - メッセージコードと言語を指定してメッセージを取得する
- MessageSource によるメッセージの取得
- spring.tldのmessageタグによるメッセージの取得

概要

MessageManager やMessageSource、タグを使って、多言語化されたメッセージを取得する方法を説明します。

MessageManager によるメッセージの取得

メッセージコードを指定してメッセージを取得する

メッセージコードと、現在ログインしているユーザの言語から、メッセージプロパティファイルに定義されたメッセージを取得します。

```
String exampleMsg =
jp.co.intra_mart.foundation.security.message.MessageManager.getInstance().getMessage("I18N.MESSAGE.EXAMPLE");
```

メッセージは、次のように自動解決されます。

1. アカウントコンテキスト言語のメッセージ
2. テナント言語のメッセージ
3. システム・デフォルト言語のメッセージ
4. 言語 ID の付いていないメッセージプロパティファイルのメッセージ
5. ユーザ言語で「未定義」を意味するメッセージ

上記のいずれにも該当しない場合は、文字列「**undefined**」が返却されます。

メッセージコードと言語を指定してメッセージを取得する

メッセージコードと、指定された言語から、メッセージプロパティファイルに定義されたメッセージを取得します。

```
String exampleMsg = jp.co.intra_mart.foundation.security.message.MessageManager.getInstance().getMessage(Locale.ENGLISH,
"I18N.MESSAGE.EXAMPLE");
```

メッセージは、次のように自動解決されます。

1. 指定された言語のメッセージ
2. 言語 ID の付いていないメッセージプロパティファイルのメッセージ

上記のいずれにも該当しない場合、文字列「**undefined**」が返却されます。

タグによるメッセージの取得

MessageManager に対応するタグが提供されています。
動作は MessageManager と同じです。

メッセージコードを指定してメッセージを取得する

```
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/core/standard" %>
<imarttag:message id="I18N.MESSAGE.EXAMPLE"></imarttag:message>
```

メッセージコードと言語を指定してメッセージを取得する

```
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/core/standard" %>
<imarttag:message id="I18N.MESSAGE.EXAMPLE" locale="en"></imarttag:message>
```

MessageSource によるメッセージの取得

MessageManager をラップした MessageSource の実装クラス IntramartMessageSource が提供されています。
applicationContext-im_tgfw_common.xml に IntramartMessageSource の bean を登録しています。

```
<bean id="messageSource" class="jp.co.intra_mart.framework.extension.spring.message.IntramartMessageSource" />
```

メッセージは、次のように自動解決されます。

1. 指定された言語のメッセージ
2. 言語 ID の付いていないメッセージプロパティファイルのメッセージ

上記のいずれにも該当しない場合、文字列「**undefined**」が返却されます。 NoSuchMessageException はスローされません。

spring.tldのmessageタグによるメッセージの取得

設定されている messageSource により、メッセージを取得します。

ユーザのタイムゾーン、日付と時刻の形式を利用する

項目

- 前提
- 画面から送信された日時を DB に保存する
- 画面から送信された日付を DB に保存する
- DB に保存されている日時をユーザの画面に表示する
- DB に保存されている日付をユーザの画面に表示する
- クライアント側で、ユーザ・タイムゾーンの今日から３日間の日付を生成し、最後の日付をサーバ側へ送信する
  - 1. ユーザ・タイムゾーンの今日を取得する
  - 2. 今日から３日間の日付を生成する
  - 3. 最後の日付をサーバ側へ送信する
  - 4. クライアント側から送信された日付文字列から Date を生成する
- @AccountDateFormat アノテーションを利用して、クライアント側から送信された文字列を Date で受け取る。
- Data binding を利用して、クライアント側から送信された文字列を Date で受け取る。
- JSONでの日付の項目とJavaのオブジェクトのDate型のプロパティとを変換する。
  - 1. bean定義xmlファイルの設定
  - 2. Controllerクラスの設定
  - 3. フォーム（モデル）クラスの設定
  - 4. JSPの設定

前提

- 各種設定値

設定項目	設定値	
ユーザ・タイムゾーン	(GMT+09:00) 日本 / 東京	
システム・タイムゾーン	(GMT+00:00) UTC	
日付と時刻の形式	英語形式	
	日付（標準表示）	MMM d, yyyy
	日付（簡易表示）	MMM d
	日付（入力）	yyyy/MM/dd
	時刻（標準表示）	h:mm a
	時刻（タイムスタンプ表示）	h:mm:ss a
	時刻（入力）	HH:mm

- テーブル定義

```
CREATE TABLE example_table (
    user_cd     VARCHAR(100) NOT NULL,
    update_date TIMESTAMP  NOT NULL,
    PRIMARY KEY (user_cd)
);
```

- DB アクセス

example_table テーブルの参照、更新を行うために、次のクラスが用意されていることを前提とします。

example_table のレコード情報を格納するためのモデルクラスです。

```
ExampleTableModel
```

example_table の参照、更新を行うクラスです。

```
SampleService
```

画面から送信された日時を DB に保存する

画面から入力された日時をサーバ側で解析して、DB に保存するまでの流れを説明します。

画面には、ユーザの日付と時刻の入力形式で次の値が入力され、サーバに送信されたとします。

```
2012/09/19 03:46
```

サーバ側のプログラムは以下の通りです。

```
import java.util.Date;

import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.i18n.datetime.format.AccountDateTimeFormatter;
import jp.co.intra_mart.foundation.i18n.datetime.format.DateTimeFormatIds;
import jp.co.intra_mart.foundation.i18n.datetime.format.DateTimeFormatterException;
import jp.co.intra_mart.foundation.i18n.sa.sample.model.ExampleTableModel;
import jp.co.intra_mart.foundation.i18n.sa.sample.service.SampleService;
import jp.co.intra_mart.foundation.i18n.sa.sample.service.SampleServiceException;

/**
 * サンプル
 */
public class SampleDateTime {

    /**
     * 画面から送信された日時を DB に保存します。
     *
     * @param inputDate 画面から送信された日時文字列
     * @throws DateTimeFormatterException
     * @throws SampleServiceException
     */
    public void sample(final String inputDate) throws DateTimeFormatterException, SampleServiceException {

        /*
         * 1. 画面から送信された日時文字列を解析します。
         */
        final Date date = AccountDateTimeFormatter.parse(inputDate, Date.class,
                DateTimeFormatIds.IM_DATETIME_FORMAT_DATE_INPUT,
                DateTimeFormatIds.IM_DATETIME_FORMAT_TIME_INPUT);

        /*
         * 2. モデルを利用して、DB へ日時を保存します。
         */
        final ExampleTableModel model = new ExampleTableModel();
        model.setUserCd(Contexts.get(AccountContext.class).getUserCd());
        model.setDate(date);

        SampleService.getInstance().update(model);

    }
}
```

ユーザの入力形式に沿った日時文字列を解析するためには、AccountDateTimeFormatter を使用します。
AccountDateTimeFormatter は、ログインユーザのタイムゾーンを使って解析します。

DB には、システム・デフォルト・タイムゾーンに変換された日時を保存します。
Date 型オブジェクトは、タイムゾーンを持たないため、DB の TIMESTAMP 型カラムには JDK のタイムゾーンに変換された日時が保存されます。
JDK のタイムゾーンとシステム・デフォルト・タイムゾーンは同じです。

画面から送信された日付を DB に保存する

画面から入力された日付をサーバ側で解析して、DB に保存するまでの流れを説明します。

> **注意**
>
> 画面から日付のみを入力させる場合でも、次の例のように、通常は時刻まで考慮しなければなりません。
>
> （例）交通費申請の締め切り日
>
> 「締め切り日」には、23時59分59秒（または、営業時間）まで、という時刻に関する意味が含まれています。
> 例えば、本社のある日本（GMT+09:00）の9月19日が締め切り日だとすると、ホノルル（GMT-10:00）支社では、9月19日 午前5時までに交通費申請を終わらせなければなりません。

画面には、ユーザの日付入力形式で次の値が入力され、サーバに送信されたとします。

```
2012/09/19
```

サーバ側のプログラムは以下の通りです。

```
import java.util.Date;

import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.i18n.datetime.format.AccountDateTimeFormatter;
import jp.co.intra_mart.foundation.i18n.datetime.format.DateTimeFormatIds;
import jp.co.intra_mart.foundation.i18n.datetime.format.DateTimeFormatterException;
import jp.co.intra_mart.foundation.i18n.sa.sample.model.ExampleTableModel;
import jp.co.intra_mart.foundation.i18n.sa.sample.service.SampleService;
import jp.co.intra_mart.foundation.i18n.sa.sample.service.SampleServiceException;

/**
 * サンプル
 */
public class SampleDateTime {

   /**
    * 画面から送信された日時を DB に保存します。
    *
    * @param inputDate 画面から送信された日時文字列
    * @throws DateTimeFormatterException
    * @throws SampleServiceException
    */
   public void sample(final String inputDate) throws DateTimeFormatterException, SampleServiceException {

      /*
       * 1. 画面から送信された日時文字列を解析します。
       */
      final Date date = AccountDateTimeFormatter.parse(inputDate, Date.class,
            DateTimeFormatIds.IM_DATETIME_FORMAT_DATE_INPUT,
            DateTimeFormatIds.IM_DATETIME_FORMAT_TIME_INPUT);

      /*
       * 2. モデルを利用して、DB へ日時を保存します。
       */
      final ExampleTableModel model = new ExampleTableModel();
      model.setUserCd(Contexts.get(AccountContext.class).getUserCd());
      model.setDate(date);

      SampleService.getInstance().update(model);

   }
}
```

日付文字列の場合、時刻部分は「00:00:00」として解析されます。

DB への保存は、「画面から送信された日付を DB に保存する」と同様です。

DB に保存されている日時をユーザの画面に表示する

DB に保存されている日時を、ユーザのタイムゾーン、日付と時刻の表示形式を使って日時文字列に整形し、画面に表示するまでの流れを説明します。

サーバ側のプログラムは以下の通りです。

```
import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.i18n.datetime.format.AccountDateTimeFormatter;
import jp.co.intra_mart.foundation.i18n.datetime.format.DateTimeFormatIds;
import jp.co.intra_mart.foundation.i18n.sa.sample.model.ExampleTableModel;
import jp.co.intra_mart.foundation.i18n.sa.sample.service.SampleService;
import jp.co.intra_mart.foundation.i18n.sa.sample.service.SampleServiceException;

/**
 * サンプル
 */
public class SampleDateTimeView {

   /**
    * DB に保存されている日時をユーザの画面に表示します。
    *
    * @return String 日時文字列
    * @throws SampleServiceException
    */
   public String sample() throws SampleServiceException {

      /*
       * 1. DB から日時を取得します。
       */
      final ExampleTableModel model = SampleService.getInstance().get(Contexts.get(AccountContext.class).getUserCd());

      /*
       * 2. 日時文字列に整形します。
       */
      return AccountDateTimeFormatter.format(model.getDate(),
            DateTimeFormatIds.IM_DATETIME_FORMAT_DATE_STANDARD,
            DateTimeFormatIds.IM_DATETIME_FORMAT_TIME_STANDARD);
   }
}
```

DB には、システム・デフォルト・タイムゾーンに変換された日時が保存されています。

ユーザの表示形式に沿った日時文字列に整形するためには、AccountDateTimeFormatter を使用します。
AccountDateTimeFormatter は、ログインユーザのタイムゾーンにおける時刻を計算します。
以下の結果が得られます。

```
Sep 19, 2012 3:46 AM
```

DB に保存されている日付をユーザの画面に表示する

DB に保存されている日付を、ユーザのタイムゾーン、日付の表示形式を使って日付文字列に整形し、画面に表示するまでの流れを説明します。

サーバ側のプログラムは以下の通りです。

```
import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.i18n.datetime.format.AccountDateTimeFormatter;
import jp.co.intra_mart.foundation.i18n.datetime.format.DateTimeFormatIds;
import jp.co.intra_mart.foundation.i18n.sa.sample.model.ExampleTableModel;
import jp.co.intra_mart.foundation.i18n.sa.sample.service.SampleService;
import jp.co.intra_mart.foundation.i18n.sa.sample.service.SampleServiceException;

/**
 * サンプル
 */
public class SampleDateView {

    /**
     * DB に保存されている日付をユーザの画面に表示します。
     *
     * @return String 日付文字列
     * @throws SampleServiceException
     */
    public String sample() throws SampleServiceException {

        /*
         * 1. DB から日時を取得します。
         */
        final ExampleTableModel model = SampleService.getInstance().get(Contexts.get(AccountContext.class).getUserCd());

        /*
         * 2. 日付文字列に整形します。
         */
        return AccountDateTimeFormatter.format(model.getDate(), DateTimeFormatIds.IM_DATETIME_FORMAT_DATE_STANDARD);
    }
}
```

次の結果が得られます。

```
Sep 19, 2012
```

クライアント側で、ユーザ・タイムゾーンの今日から３日間の日付を生成し、最後の日付をサーバ側へ送信する

このサンプルを通して、csjs で日時データを扱う際に注意する点を理解します。

1. ユーザ・タイムゾーンの今日を取得する

ユーザ・タイムゾーンとは、ユーザが intra-mart Accel Platform に登録したタイムゾーンのことです。
ユーザ・タイムゾーンにおける「今日」を csjs で取得するためには、intra-mart Accel Platform から提供されている ImDate を使用します。

注意

ユーザ・タイムゾーンにおける「今日」の取得に、csjs の new Date を利用しないでください。
csjs の new Date はクライアント OS のタイムゾーンにおける現在日時データを返しますが、ユーザ・タイムゾーンがクライアント OS のタイムゾーンと一致しているとは限りません。

```
<script type="text/javascript" src="im_i18n/timezone/im_date_timezone.js"></script>
<script type="text/javascript">
var firstDate = ImDate.now();
</script>
```

2. 今日から３日間の日付を生成する

Date に標準で用意されているメソッドを使用して構いません。

```
var dateArray = new Array();
var date = firstDate;
for (var i = 0; i < 3; i++) {
    dateArray[i] = date;
    date.setDate(date.getDate() + 1);
}
```

3. 最後の日付をサーバ側へ送信する

年月日の値から日付文字列を作ります。

```
var lastDate = dateArray[2];
var lastDateStr = lastDate.getFullYear() + "-" + (lastDate.getMonth() + 1) + "-" + lastDate.getDate();
```

注意

ImDate.now() で生成した Date のエポックミリ秒は送信しないでください。

ImDate.now() の返す Date は、ユーザ・タイムゾーンにおける「今日」の年月日時分秒を持っていますが、エポックミリ秒は正しいとは限りません。
理由は、クライアント側で Date を生成しているためです。
Date の持つエポックミリ秒は、クライアント OS のタイムゾーンで計算された値となり、ユーザ・タイムゾーンで計算された値と一致しない可能性があります。

注意

サーバ側でユーザ・タイムゾーンの日時データを生成する場合は、Date ではなく、DateTime を使用してください。
Date はシステム・デフォルトのタイムゾーンで計算するため、ユーザ・タイムゾーンとの時差を考慮して扱う必要があります。
DateTime は指定されたタイムゾーンで計算できます。

```
jp.co.intra_mart.foundation.i18n.datetime.DateTime dateTime = new
jp.co.intra_mart.foundation.i18n.datetime.DateTime(Contexts.get(AccountContext.class).getTimeZone(), 1996,
Calendar.SEPTEMBER, 19, 3, 47, 0);
```

4. クライアント側から送信された日付文字列から Date を生成する

サーバ側のプログラムは以下の通りです。

```
import java.sql.Date;
import java.util.TimeZone;

import javax.servlet.http.HttpServletRequest;

import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.i18n.datetime.format.DateTimeFormatter;
import jp.co.intra_mart.foundation.i18n.datetime.format.DateTimeFormatterException;

/**
 * サンプル
 */
public class SampleDTF {

   /**
    * クライアント側から送信された日付文字列から Date を生成します。
    * @param request リクエスト
    * @throws DateTimeFormatterException
    */
   public void sample(final HttpServletRequest request) throws DateTimeFormatterException {

      /*
       * 1. クライアント側から送信された日付文字列を取得します。
       */
      final String inputDateStr = request.getParameter("inputDate");

      /*
       * 2. DateTimeFormatter を使用して、日付文字列から Date を生成します。
       */
      final TimeZone userTimeZone = Contexts.get(AccountContext.class).getTimeZone();
      final DateTimeFormatter formatter = DateTimeFormatter.withPattern("yyyy-MM-dd");
      formatter.setTimeZone(userTimeZone);
      final Date inputDate = formatter.parse(inputDateStr, Date.class);
   }
}
```

日時文字列がユーザの日付と時刻の入力形式に沿っていない場合、DateTimeFormatter を使用します。
DateTimeFormatter は、直接フォーマットパターンを指定できます。
DateTimeFormatter は、システム・デフォルトのタイムゾーンを使って解析を行います。
ユーザ・タイムゾーンを使って解析を行う場合は、parse メソッドを実行する前に、ユーザ・タイムゾーンをセットします。

@AccountDateFormat アノテーションを利用して、クライアント側から送信された文字列を Date で受け取る。

画面から入力された日付、日時、時刻の文字列を Date型のフィールドにバインドします。
日付、日時、時刻に対応するアノテーションは以下の通りです。

パターン	アノテーション	フォーマット	タイムゾーン変換(デフォルト値)
日付	@AccountDateFormat	日付(入力)	なし
日時	@AccountDateTimeFormat	日付(入力) 時刻(入力)	あり
時刻	@AccountTimeFormat	時刻(入力)	なし

- フォーマットについて
  日付のフォーマットについては *日付と時刻の形式* を参照してください。
- タイムゾーン変換について
  入力された文字列をアカウントのタイムゾーンでの日時とし、サーバ側でのDate型プロパティにセットするときにシステムタイムゾーンにします。タイムゾーン変換なしの場合は、入力された文字列、サーバ側でのDate型プロパティもシステムタイムゾーンでの値です。

画面から入力された文字列をアカウントのタイムゾーンとし、Date型プロパティに設定する例を次に示します。

```
import java.util.Date;

import jp.co.intra_mart.framework.extension.spring.format.annotation.AccountDateFormat;
import jp.co.intra_mart.framework.extension.spring.format.annotation.AccountDateTimeFormat;
import jp.co.intra_mart.framework.extension.spring.format.annotation.AccountTimeFormat;

public class SampleForm {

    // ここではシステムタイムゾーンはJSTとします。

    // 日付フォーマットのリクエスト文字列をDate型プロパティにバインドします。
    // (String) 2000/01/01 が (Date) 2000/01/01 00:00:00 JST になります。
    // デフォルトではタイムゾーン変換しません。
    @AccountDateFormat
    private Date date;

    // 日時フォーマットのリクエスト文字列をDate型プロパティにバインドします。
    // アカウントのタイムゾーンがGMTの場合、システムタイムゾーンのJST(+9)に変換するので、
    // (String) 2000/01/01 00:00 が (Date) 2000/01/01 09:00:00 JST になります。
    @AccountDateTimeFormat
    private Date datetime;

    // 時刻フォーマットのリクエスト文字列をDate型プロパティにバインドします。
    // (String) 01:00 が (Date) 1970/01/01 01:00:00 JST になります。
    // デフォルトではタイムゾーン変換しません。
    @AccountTimeFormat
    private Date time;

    // 日付フォーマットでタイムゾーン変換ありに設定した場合
    // (String) 2000/01/01 が (Date) 2000/01/01 09:00:00 JST になります。
    @AccountDateFormat(convertTimeZone = true)
    private Date dateTimezone;

    // getter, setter ...
}
```

コラム

フォーマット変換に失敗した場合のエラーメッセージコードの解決については Spring Frameworkの仕様に従います。
「Resolving Codes to Error Messages」 を参照してください。

Data binding を利用して、クライアント側から送信された文字列を Date で受け取る。

Controller クラスに @InitBinder を設定したメソッドを作成し、データバインドの定義を登録します。
以下に、@InitBinderを使用したデータバインドの例を示します。

```
import java.util.Date;

public class SampleForm {

    private Date date;

    private Date datetime;

    private Date time;

    // getter, setter ...
}
```

```
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

import jp.co.intra_mart.foundation.i18n.datetime.format.AccountDateTimeFormatter;
import jp.co.intra_mart.foundation.i18n.datetime.format.DateTimeFormatIds;

import org.springframework.beans.propertyeditors.CustomDateEditor;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("sample/tgfw/datetime")
public class DateTimeController {

    @RequestMapping("register")
    public String register(SampleForm form, BindingResult result) {

        // 登録して、入力画面を再表示

        // 登録処理など ...

        return "sample/tgfw/datetime/input.jsp";
    }

    @InitBinder
    public void initBinder(final WebDataBinder binder) {
        // プロパティ"date"に対するデータバインド設定
        // "日付(入力)" フォーマットパターンを取得します。
        final String datePattern = AccountDateTimeFormatter.getPattern(DateTimeFormatIds.IM_DATETIME_FORMAT_DATE_INPUT);
        // DateFormatを生成します。
        final DateFormat dateFormat = new SimpleDateFormat(datePattern);
        // プロパティ"date"に対するデータバインドを登録します。
        binder.registerCustomEditor(Date.class, "date", new CustomDateEditor(dateFormat, true));

        // プロパティ"datetime"に対するデータバインド設定
        final String datetimePattern = AccountDateTimeFormatter.getPattern(DateTimeFormatIds.IM_DATETIME_FORMAT_DATE_INPUT,
DateTimeFormatIds.IM_DATETIME_FORMAT_TIME_INPUT);
        final DateFormat datetimeFormat = new SimpleDateFormat(datetimePattern);
        binder.registerCustomEditor(Date.class, "datetime", new CustomDateEditor(datetimeFormat, true));

        // プロパティ"time"に対するデータバインド設定
        final String timePattern = AccountDateTimeFormatter.getPattern(DateTimeFormatIds.IM_DATETIME_FORMAT_TIME_INPUT);
        final DateFormat timeFormat = new SimpleDateFormat(timePattern);
        binder.registerCustomEditor(Date.class, "time", new CustomDateEditor(timeFormat, true));
    }
}
```

画面から入力された文字列を、date, datetime, timeのDate型のフィールドにセットしています。
この例では、タイムゾーンの変換をしていません。入力された文字列、サーバ側でのDate型プロパティもシステムタイムゾーンでの値です。

JSONでの日付の項目とJavaのオブジェクトのDate型のプロパティとを変換する。

Spring Framework MVCでは、リクエストやレスポンスにJSONをセットして送る場合に、Jacksonを利用してJSONとJavaのオブジェクトとを変換しています。
Jacksonでは、デフォルトでDate型のプロパティはエポックミリ秒で出力され、日付項目の文字列は yyyy-MM-dd'T'HH:mm:ss.SSSZ 形式でパースされます。
これをアカウントに設定された日付フォーマットで変換するようにします。

Date型プロパティとJSONとの変換のフォーマットを指定できるようにするために、以下のようにJacksonの設定を行います。
これにより、(フォーム)モデルクラスに @AccountDateFormatアノテーションなどを設定して日付をフォーマットして出力できます。

1. bean定義xmlファイルの設定
   mvc:message-convertersを設定します。

2. Controllerクラスの設定
   リクエストをJSONで受け取るようにします。
   レスポンスにJSONをセットするようにします。

3. フォーム(モデル)クラスの設定
   Date型のプロパティにアノテーションを設定します。

4. JSPの設定
   送信データをjsonデータにします。
   JSP側でjsonデータを受け取ります。

1. bean定義xmlファイルの設定

applicationContext-im_tgfw_web.xml のmvc:annotation-drivenタグに mvc:message-convertersタグを追加します。
Message converterにMappingJackson2HttpMessageConverterを設定し、そのobjectMapperプロパティに AccountDateObjectMapper を設定します。
AccountDateObjectMapper により、アノテーションが動作します。

```
<mvc:annotation-driven conversion-service="conversionService">

    <!-- 略 -->

    <!-- jackson message converter -->
    <mvc:message-converters register-defaults="true">
      <bean class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">
        <property name="objectMapper">
          <bean class="jp.co.intra_mart.framework.extension.spring.http.converter.json.AccountDateObjectMapper" />
        </property>
      </bean>
    </mvc:message-converters>

</mvc:annotation-driven>
```

2. Controllerクラスの設定

SpringFramework MVCに従って、リクエスト、レスポンスにJSONをセットするように設定します。
この例では、メソッドの引数に@RequestBodyアノテーションを設定し、メソッドの戻り値にJSONにするクラスを定義しメソッドに @ResponseBodyアノテーションを設定します。

```
@Controller
@RequestMapping("path/to")
public class SampleController {

    @RequestMapping("json")
    @ResponseBody
    public SampleModel json(@RequestBody DateJsonForm form) {
        // form.date には入力した日付のDateがセットされる。

        SampleModel sampleModel = new SampleModel();

        // sampleModelに値を設定する処理

        return sampleModel;
    }
}
```

3. フォーム（モデル）クラスの設定

必要に応じて、Date型のプロパティに @AccountDateFormatアノテーションを設定します。

```
import java.util.Date;

import jp.co.intra_mart.framework.extension.spring.format.annotation.AccountDateFormat;

public class DateJsonForm {

    // AccountDateFormatアノテーションを設定した例
    @AccountDateFormat
    private Date date;


    // setter & getter
}
```

```
import java.util.Date;

import jp.co.intra_mart.framework.extension.spring.format.annotation.AccountDateFormat;

public class SampleModel {

    // アノテーションなし
    private Date date;

    // AccountDateFormatアノテーションを設定した例
    @AccountDateFormat
    private Date dateDefault;

    // フォーマットタイプをSIMPLEに設定した例
    @AccountDateFormat(type = AccountDateFormat.TYPE.SIMPLE)
    private Date dateSimple;

    // AccountDateTimeFormatアノテーションを設定した例
    @AccountDateTimeFormat
    private Date datetimeDefault;


    // setter & getter
}
```

4. JSPの設定

AjaxでJSONデータを送信、受信する場合の例を示します。

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui"%>

<imui:head>
<title>sample</title>
<script type="text/javascript">
(function ($) {
  $(document).ready(function () {
    $('#link').on('click', function () {

      $.ajax({
        dataType : 'json',
        url : '<c:url value="path/to/json"/>',
        data : JSON.stringfy({ date : "2000/01/01" }), // アカウントの日付フォーマットに従ったフォーマット
        type :'GET',
        success : function (data, textStatus, jqXHR) {
          // 受け取ったデータ(data)の処理の例
          $('#date').val(data.date);
          $('#dateDefault').val(data.dateDefault);
          $('#dateSimple').val(data.dateSimple);
          $('#datetimeDefault').val(data.datetimeDefault);
        }
      });

    });
  });
})(jQuery);
</script>
</imui:head>

<div class="imui-title">
  <h1>json - index page</h1>
</div>

<div class="imui-form-container">

<a href="javascript:void(0);" id="link">fetch data</a>

<ul>
<li>アノテーションなし - エポックミリ秒で表示<br/><input type="text" id="date" name="date" value=""></li>
<li>@AccountDateFormat - yyyy/MM/dd形式で表示<br/><input type="text" id="dateDefault" name="dateDefault" value=""></li>
<li>@AccountDateFormat(type = SIMPLE) - MM/dd形式で表示<br/><input type="text" id="dateSimple" name="dateSimple"
value=""></li>
<li>@AccountDateTimeForma - yyyy/MM/dd HH:mm形式で表示<br/><input type="text" id="datetimeDefault"
name="datetimeDefault" value=""></li>
</ul>

</div>
```

データベース

項目

- 概要
- MyBatis3について
- プログラミング方法
  - MyBatis-Springの設定
  - TypeAliasの設定
  - リポジトリインタフェースの作成
  - マッピングファイルの作成
  - コントローラ
  - エンティティクラスの自動生成
  - トランザクション制御
    - Rollbackする例外を定義する
    - 明示的トランザクション
- シェアードデータベースの利用

概要

TERASOLUNA Server Framework for Java (5.x) （intra-mart Accel Platform 2015 Spring(Juno) 以降） では、永続化フレームワークとしてMyBatis3を使用します。

- MyBatis3(3.5)

当項ではMyBatis3を使用したデータベースプログラミングの方法を説明します。

MyBatis3について

MyBatis3はデータベースで管理されているレコードとオブジェクトをマッピングするという考え方ではなく、SQLとオブジェクトをマッピングという考え方で開発されたO/Rマッピングフレームワークです。
複雑な結合条件、細かいクエリチューニングを行う必要がある場合などに向いています。
MyBatis3の詳細については下記を参照してください。
http://www.mybatis.org/mybatis-3/
http://www.mybatis.org/spring/

> **注意**
>
> MyBatisは2.x系と3.x系で実装が異なります。
> TERASOLUNA Server Framework for Java (5.x) では、MyBatis3.5系を利用して実装します。

プログラミング方法

MyBatis-Springの設定

Springの設定にMyBatis3の設定を追加します。
juggling project の <classes/META-INF/spring/applicationContext-im_tgfw_mybatis3.xml> に以下の設定を追加、修正します。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mybatis="http://mybatis.org/schema/mybatis-spring"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-
beans.xsd
        http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd
        http://mybatis.org/schema/mybatis-spring
        https://mybatis.org/schema/mybatis-spring.xsd">

    <!-- DIコンポーネントの対象とする要素のトップレベルパッケージ -->
    <context:component-scan base-package="my.terasoluna.domain" />

    <bean id="sqlSessionFactory"
        class="org.mybatis.spring.SqlSessionFactoryBean">  <!-- ① -->
        <property name="dataSource" ref="dataSource" />  <!-- ② -->
        <property name="configLocation"
            value="classpath:/META-INF/mybatis/mybatis-config.xml" />  <!-- ③ -->
    </bean>

    <mybatis:scan base-package="my.terasoluna.domain.repository" />  <!-- ④ -->

</beans>
```

1. SqlSessionFactory を生成するためのコンポーネントとして、SqlSessionFactoryBean をbean定義します。
2. dataSource プロパティに、設定済みのデータソースのbeanを指定します。
   MyBatis3の処理の中でSQLを発行する際は、ここで指定したデータソースからコネクションが取得されます。
3. configLocation プロパティに、MyBatis設定ファイルのパスを指定します。
   ここで指定したファイルがSqlSessionFactory を生成する時に読み込まれます。
4. リポジトリインタフェースをスキャンするためにmybatis:scan要素を定義し、base-package属性には、リポジトリインタフェースが格納されている基底パッケージを指定します。

TypeAliasの設定

MyBatis3の機能を利用するにあたり、マッピングファイルで指定するJavaクラスに対して、エイリアス名（短縮名）を割り当てる事ができます。
TypeAliasを使用しない場合、マッピングファイルで指定する属性にJavaクラスの完全修飾クラス名を指定する必要があるため、マッピングファイルの記述効率の低下、記述ミスの増加などが懸念されます。
juggling project の <classes/META-INF/mybatis/mybatis-config.xml> に設定を記述します。

- パッケージ単位で指定する例

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
 PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
 "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <package name="my.terasoluna.domain.model" />
    </typeAliases>
</configuration>
```

package要素のname属性に、エイリアスを設定するクラスが格納されているパッケージ名を指定します。
指定したパッケージ配下に格納されているクラスは、パッケージの部分が除去された部分がエイリアス名として割り当てられます。
上記例だと、my.terasoluna.domain.model.MyCompanyクラスのエイリアス名は、mycompanyです。

- クラス単位で指定する例

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
 PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
 "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <typeAliases>
    <typeAlias alias="MyCompany" type="my.terasoluna.domain.model.MyCompany" />
  </typeAliases>
</configuration>
```

typeAlias要素のtype属性に、エイリアスを設定するJavaクラスの完全修飾クラス名を指定します。
指定したクラスの、 alias属性に設定した値がエイリアス名として割り当てられます。
上記例だと、my.terasoluna.domain.model.MyCompanyクラスのエイリアス名は、MyCompanyです。

> **コラム**
>
> 記述効率の向上、記述ミスの削減、マッピングファイルの可読性向上などを目的として、TypeAliasを使用することが推奨されています。

リポジトリインタフェースの作成

任意のエンティティに対し、以下のようにリポジトリインタフェースを作成します。

```
package my.terasoluna.domain.repository;

import my.terasoluna.domain.model.MyCompany;
import java.util.List;
import org.apache.ibatis.session.RowBounds;

public interface MyCompanyRepository {

    public void insert(MyCompany entity); // ①

    public void update(MyCompany entity); // ②

    public void delete(String companyId); // ③

    public MyCompany selectOne(String companyId); // ④

    public List<MyCompany> selectAll(RowBounds rowBounds); // ⑤

    public List<MyCompany> selectLessId(String companyId); // ⑥

    public List<MyCompany> selectLessIdByInt(int companyId); // ⑦

}
```

1. マッピングファイルのidの”insert”にマッピングしています。
   引数はクエリの要求するパラメータクラスのオブジェクトをセットします。
2. マッピングファイルのidの”update”にマッピングしています。
   引数はクエリの要求するパラメータクラスのオブジェクトをセットします。
3. マッピングファイルのidの”delete”にマッピングしています。
   引数はクエリの要求する文字列をセットします。
4. マッピングファイルのidの”selectOne”にマッピングしています。
   引数はクエリの要求する文字列をセットします。
5. マッピングファイルのidの”selectAll”にマッピングしています。
   引数はページネーションするためのオブジェクト（取得範囲の情報(offsetとlimit)を保持するRowBounds）をセットします。
   検索結果から該当範囲のレコードを抽出する処理は、MyBatis3が行うため、SQLで取得範囲のレコードを絞り込む必要はありません。
6. マッピングファイルのidの”selectLessId”にマッピングしています。
   引数はクエリの要求する文字列をセットします。
7. マッピングファイルのidの”selectLessIdByInt”にマッピングしています。
   引数はクエリの要求する数値をセットします。

マッピングファイルの作成

MyBatis3の機能を利用するにあたり、実行するクエリを定義するマッピングファイルを作成する必要があります。
例として、以下のようなXMLファイルを作成し、マッピングされるリポジトリインタフェースとresourcesの同階層に配置します。
%プロジェクト%/src/main/javaの配下 my.terasoluna.domain.repository.MyCompanyRepository.java
%プロジェクト%/src/main/resourcesの配下 my.terasoluna.domain.repository.MyCompanyRepository.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
   "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="my.terasoluna.domain.repository.MyCompanyRepository">  <!-- ① -->

 <resultMap id="resultMapMyCompnay" type="MyCompany">  <!-- ② -->
  <id property="companyId" column="company_id" />
  <result property="name" column="name" />
 </resultMap>

 <insert id="insert" parameterType="MyCompany">  <!-- ③ -->
  INSERT INTO my_company (company_id, name)
  VALUES(#{companyId}, #{name})
 </insert>

 <update id="update" parameterType="MyCompany">  <!-- ④ -->
  UPDATE my_company SET
  name = #{name}
  WHERE company_id = #{companyId}
 </update>

 <delete id="delete" parameterType="string">  <!-- ⑤ -->
  DELETE FROM my_company WHERE company_id = #{companyId}
 </delete>

 <select id="selectOne" parameterType="string" resultMap="resultMapMyCompnay">  <!-- ⑥ -->
  SELECT * FROM my_company WHERE company_id = #{companyId}
 </select>

 <select id="selectAll" resultMap="resultMapMyCompnay">  <!-- ⑦ -->
  SELECT * FROM my_company
 </select>

 <select id="selectLessId" parameterType="string" resultMap="resultMapMyCompnay">  <!-- ⑧ -->
 <![CDATA[
  SELECT * FROM my_company WHERE company_id <  TO_NUMBER(#{companyId}, '999')
 ]]>
 </select>

 <select id="selectLessIdByInt" parameterType="_int" resultMap="resultMapMyCompnay">  <!-- ⑨ -->
 <![CDATA[
  SELECT * FROM my_company WHERE company_id <  #{companyId}
 ]]>
 </select>

</mapper>
```

1. mapper要素のnamespace属性に、リポジトリインタフェースの完全修飾クラス名を指定します。

2. resultMap要素に、検索結果(ResultSet)とJavaBeanのマッピング定義を行います。
   id属性にマッピングを識別するためのキーを、type属性にマッピングするJavaBeanのクラス名（またはエイリアス名）を指定します。
   id要素とresult要素は、 どちらも検索結果(ResultSet)のカラムとJavaBeanのプロパティをマッピングするための要素ですが、
   ID(PK)カラムに対してマッピングは、id要素を使うことが推奨されます。
   理由としては、ID(PK)カラムに対してid要素を使用してマッピングを行いますと、MyBatis3が提供しているオブジェクトのキャッシュ制御の処理や、 関連オブジェクトへのマッピングの処理のパフォーマンスを、全体的に向上させることができるためです。

3. insert要素の中に、INSERTするSQLを実装します。
   id属性には、リポジトリインタフェースに定義したメソッドのメソッド名を指定します。
   VALUE句にバインドする値は、#{variableName}形式のバインド変数として指定します。
   上記例では、リポジトリインタフェースの引数としてJavaBean(MyCompany)を指定しているため、 バインド変数名には

JavaBeanのプロパティ名を指定します。

4. update要素の中に、UPDATEするSQLを実装します。
   id属性には、リポジトリインタフェースに定義したメソッドのメソッド名を指定します。
   SET句にバインドする値は、#{variableName}形式のバインド変数として指定します。
   上記例では、リポジトリインタフェースの引数としてJavaBean(MyCompany)を指定しているため、 バインド変数名には JavaBeanのプロパティ名を指定します。

5. delete要素の中に、DELETEするSQLを実装します。
   id属性には、リポジトリインタフェースに定義したメソッドのメソッド名を指定します。
   WHERE句に削除条件を指定します。
   削除条件にバインドする値は、#{variableName}形式のバインド変数として指定します。上記例では、 #{companyId}がバインド変数として指定されます。
   Stringのような単純型の場合は、バインド変数名に制約はありませんが、メソッドの引数名と同じ値にしておくことが推奨されます。

6. select要素の中に、SELECTするSQLを実装します。
   id属性には、リポジトリインタフェースに定義したメソッドのメソッド名を指定します。
   WHERE句に検索条件を指定します。
   検索条件にバインドする値は、#{variableName}形式のバインド変数として指定します。上記例では、 #{companyId}がバインド変数として指定されます。
   select要素のresultMap属性に、適用するマッピング定義のIDを指定します。

7. select要素の中に、SELECTするSQLを実装します。
   id属性には、リポジトリインタフェースに定義したメソッドのメソッド名を指定します。
   リポジトリインタフェースにて、ページネーションするためのオブジェクト（取得範囲の情報(offsetとlimit)を保持する RowBounds）がセットされていますが、検索結果から該当範囲のレコードを抽出する処理は、MyBatis3が行うため、SQLで取得範囲のレコードを絞り込む必要はありません。

8. select要素の中に、SELECTするSQLを実装します。
   id属性には、リポジトリインタフェースに定義したメソッドのメソッド名を指定します。
   WHERE句に検索条件を指定します。
   検索条件にバインドする値は、#{variableName}形式のバインド変数として指定します。上記例では、 #{companyId}がバインド変数指定されます。
   SQL内にXMLのエスケープが必要な文字（”<”や”>”など）を指定する場合は、 CDATAセクションを使用すると、SQLの可読性を保つことができます。
   CDATAセクションを使用しない場合は、”<”や”>”といったエンティティ参照文字を指定する必要があり、 SQLの可読性を損なう可能性があります。

9. select要素の中に、SELECTするSQLを実装します。
   8と同一結果となるSQLですが、parameterType属性に_intを指定しています。
   parameterTypeのstringや_intの他にもdateやcollection等存在します。詳しくはリファレンスを参照してください。

コラム

その他様々なタグや機能がありますので、詳しくは リファレンス を参照してください。

コントローラ

コントローラ（呼び出し）の例です。

```
@Inject
MyCompanyRepository myCompanyRepository; // ①

public void insert(MyCompanyForm myCompanyForm) { // ②
    MyCompany myCompany = new MyCompany();

...

    myCompanyRepository.insert(myCompany);
}

public void update(MyCompanyForm myCompanyForm) { // ③
    MyCompany myCompany = new MyCompany();

...

    myCompanyRepository.update(myCompany);
}

public void delete(MyCompanyForm myCompanyForm) { // ④
    String deleteCompanyId = "";

...

    myCompanyRepository.delete(deleteCompanyId);
}

public MyCompany selectOne(MyCompanyForm myCompanyForm) { // ⑤
    String companyId = "10";

...

    return myCompanyRepository.selectOne(companyId);
}

public List<MyCompany> selectAll(MyCompanyForm myCompanyForm) { // ⑥
    RowBounds rowBounds = new RowBounds(5, 10);

...

    return myCompanyRepository.selectAll(rowBounds);
}

public List<MyCompany> selectLessId(MyCompanyForm myCompanyForm) { // ⑦
    String companyId = "10";

...

    return myCompanyRepository.selectLessId(companyId);
}

public List<MyCompany> selectLessIdByInt(MyCompanyForm myCompanyForm) { // ⑧
    String companyId = "10";

...

    return myCompanyRepository.selectLessIdByInt(Integer.parseInt(companyId));
}
```

1. リポジトリインタフェースをDIします。
2. リポジトリインタフェースのメソッドを呼び出し、INSERTします。
3. リポジトリインタフェースのメソッドを呼び出し、UPDATEします。
4. リポジトリインタフェースのメソッドを呼び出し、DELETEします。
5. リポジトリインタフェースのメソッドを呼び出し、1件取得します。
6. リポジトリインタフェースのメソッドを呼び出し、件数を制限して取得します。
   上記の例では、6件目から10件取得します。

1件目から取得する場合は、第1引数(offset)に0を指定します。

```
RowBounds rowBounds = new RowBounds(0, n);
```

7. リポジトリインタフェースのメソッドを呼び出し、条件の範囲内を取得します。

8. リポジトリインタフェースのメソッドを呼び出し、条件の範囲内を取得します。
   検索条件をStringからintに変換して取得します。

エンティティクラスの自動生成

手で作成してもよいのですが、テーブルからエンティティクラスを自動生成するのが簡単です。
Eclipse DTP には、データベースに接続して選択したテーブルからエンティティクラスを生成する機能があります。
詳細については下記ページを参照してください。

- *Eclipseからエンティティを生成*

トランザクション制御

トランザクション制御の設定は任意のクラス・メソッドに@Transactionalアノテーションを付加して設定します。
例えば、サービスクラスのクラス単位にトランザクション境界を設定する場合、以下のようにクラスに@Transactionalアノテーションを付加します。

```
import org.springframework.transaction.annotation.Transactional;
...

@Service
@Transactional
public class HogeServiceImpl implements HogeService {
  ...
```

> **コラム**
>
> @Transactionalはメソッド単位に設定することもできます。

Rollbackする例外を定義する

トランザクション境界内で例外が発生した場合、デフォルトでは、RuntimeExceptionを継承する例外のみロールバックされます。
ロールバックする例外を追加定義する場合、以下のように@TransactionalにrollbackFor属性を付加します。

```
@Transactional(rollbackFor=java.lang.Exception.class)
```

> **コラム**
>
> 逆に、noRollbackFor属性を設定することで、ロールバックしない例外を設定することもできます。
> 詳しくは、「Transaction Management」 を参照してください。

明示的トランザクション

アノテーションを用いず、ソースコード内に直接トランザクション処理を記述する場合、以下のようにします。

```
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.DefaultTransactionDefinition;

...

@Inject
PlatformTransactionManager transactionManager;// ①

...

 @Override
 public void store(MyCompany entity) {

   DefaultTransactionDefinition def = new DefaultTransactionDefinition();  // ②
   TransactionStatus status = transactionManager.getTransaction(def);     // ③

   try {

    //ビジネスロジック

   } catch (RuntimeException e) {
      transactionManager.rollback(status);  // ④
      throw e;
   }

   transactionManager.commit(status);  // ⑤
```

1. PlatformTransactionManagerをフィールド定義します。
   @InjectによりBeanが生成されます。
2. トランザクション定義情報を設定する
   DefaultTransactionDefinitionクラスを生成します。
3. transactionManager#getTransactionでトランザクションを開始します。
4. ロールバックする場合、transactionManager#rollbackを使用します。
5. コミットする場合、transactionManager#commitを使用します。

シェアードデータベースの利用

applicationContext-im_tgfw_common.xml のsharedDataSourceのコメントアウトを外します。
このとき、connectIdに対象シェアードデータベースの接続IDを指定します。

```
<bean id="dataSource"   class="jp.co.intra_mart.framework.extension.spring.datasource.TenantDataSource" />

<bean id="sharedDataSource" class="jp.co.intra_mart.framework.extension.spring.datasource.SharedDataSource">
  <constructor-arg name="connectId" value="sharedA" />
</bean>
```

MyBatis-Springの設定にシェアードデータベースの定義を追加します。

```
<bean id="sqlSessionFactory"
  class="org.mybatis.spring.SqlSessionFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <property name="configLocation"
    value="classpath:/META-INF/mybatis/mybatis-config.xml" />
</bean>

<bean id="sharedSqlSessionFactory"
  class="org.mybatis.spring.SqlSessionFactoryBean">  <!-- ① -->
  <property name="dataSource" ref="sharedDataSource" />  <!-- ② -->
  <property name="configLocation"
    value="classpath:/META-INF/mybatis/mybatis-config.xml" />  <!-- ③ -->
</bean>

<mybatis:scan base-package="my.terasoluna.domain.repository" factory-ref="sqlSessionFactory" />  <!-- ④ -->

<mybatis:scan base-package="my.terasoluna.domain.shared.repository" factory-ref="sharedSqlSessionFactory"/>  <!-- ⑤ -->
```

1. SqlSessionFactory を生成するためのコンポーネントとして、SqlSessionFactoryBean をbean定義します。
2. dataSource プロパティに、設定済みのシェアードデータベースのデータソースのbeanを指定します。
3. configLocation プロパティに、MyBatis設定ファイルのパスを指定します。
   ここで指定したファイルがシェアードデータベースのSqlSessionFactory を生成する時に読み込まれます。
4. リポジトリインタフェースをスキャンするためにmybatis:scan要素を定義し、base-package属性には、リポジトリインタフェースが格納されている基底パッケージを指定します。
   SqlSessionFactoryBeanのbeanを複数設定した場合、factory-ref属性の指定が必要です。
5. リポジトリインタフェースをスキャンするためにmybatis:scan要素を定義し、base-package属性には、リポジトリインタフェースが格納されている基底パッケージを指定します。
   factory-ref属性にsharedSqlSessionFactoryを指定します。

> **注意**
>
> mybatis:scan要素を使用する場合は、リポジトリインタフェースのパッケージをテナントデータベースとシェアードデータベースで分ける必要があります。

> **コラム**
>
> Jugglingプロジェクトの「追加リソースの選択」から TERASOLUNA Server Framework for Java (5.x) (...)設定ファイルを選択すると上記ファイルを含む設定ファイルがJugglingプロジェクト上に配置されます。
> チームで開発を行う場合、モジュールプロジェクト上に上記編集ファイルを配置すると他のモジュールプロジェクトに影響を与える可能性があるので、予めJugglingプロジェクト上から設定ファイルの編集を行ってください。

ログ

項目

- 概要
- Logger APIを利用する
  - ログレベル
  - Loggerオブジェクトを取得する
  - ログを出力する
- MDC APIを利用する
  - MDC
  - MDCを利用したログを出力する
- TERASOLUNA Server Framework for Java (5.x) for Accel Platform のログ出力設定について
- TERASOLUNA Server Framework for Java (5.x) の TraceLoggingInterceptor を利用する。

概要

内部統制、セキュリティ確保や保守などの目的からログを出力します。

この項目では TERASOLUNA Server Framework for Java (5.x) for Accel Platform で行うログの実装について記述します。

Logger APIを利用する

Logger(jp.co.intra_mart.common.platform.log.Logger)の使用方法について記述します。

ログレベル

Logger APIでは5つのログレベルが提供されています。

trace（最も軽微）
debug
info
warn
error（最も重大）

Loggerオブジェクトを取得する

```
public int add(final int value1, final int value2) {
    // ロガーインスタンスを取得
    // ロガー名を指定していないため、クラス名がロガー名となる
    // クラス名：tutorial.controller.AddController
    // ロガー名：tutorial.controller.AddController
    final Logger logger = Logger.getLogger();

    return 0;
}
```

Logger#getLogger()メソッドを利用してLoggerオブジェクトを取得します。引数に渡す文字列がロガーの名前として使用されます。
引数に何も渡さない場合は、呼び出したクラス名(FQCN)がロガーの名前として使用されます。

ログを出力する

```
public int add(final int value1, final int value2) {
    // ロガーインスタンスを取得
    // ロガー名を指定していないため、クラス名がロガー名となる
    // クラス名：tutorial.controller.AddController
    // ロガー名：tutorial.controller.AddController
    final Logger logger = Logger.getLogger();

    logger.debug("arguments=[{}, {}]", value1, value2);
    final int result = value1 + value2;
    logger.trace("result={}", result);

    return result;
}
```

Loggerオブジェクトを利用してログの出力を行います。
上記の関数では、引数に渡された値がdebugレベルで、計算結果がtraceレベルで出力しています。

ログレベルdebugでadd(1, 2)を実行した場合は以下のように出力されます。

```
[DEBUG] t.c.AddController - arguments=[1, 2]
```

ログレベルtraceでadd(1, 2)を実行した場合は以下のように出力されます。

```
[DEBUG] t.c.AddController - arguments=[1, 2]
[TRACE] t.c.AddController - result=3
```

MDC APIを利用する

MDC(jp.co.intra_mart.common.platform.log.MDC)の使用方法について記述します。

MDC

Mapped Diagnostic Context（マップ化された診断コンテキスト）を利用することにより、ログ設定ファイルのレイアウト設定で独自に定義したkeyで保存した情報のログ出力が可能です。
MDC APIを利用することにより、独自に定義したkeyへの情報の書き込みが可能です。

MDCを利用したログを出力する

```
// MDCのキーを定義
private static final String MKC_FUNC_KEY = "application.func";

public int add(final int value1, final int value2) {
    // ロガーインスタンスを取得
    // ロガー名を指定していないため、クラス名がロガー名となる
    // クラス名：tutorial.controller.AddController
    // ロガー名：tutorial.controller.AddController
    final Logger logger = Logger.getLogger();

    // MDCに値を設定
    MDC.put(MKC_FUNC_KEY, "add");

    logger.debug("arguments=[{}, {}]", value1, value2);
    final int result = value1 + value2;
    logger.trace("result={}", result);

    // MDCの値を初期化
    MDC.remove(MKC_FUNC_KEY);

    return result;
}
```

実行中の関数名をMDCに設定しています。
MDC#put(key, value)メソッドで、MDCのキー “application.func” に実行中の関数名 “add” を設定しています。
MDCに保存した内容は、明示的に初期化が行われない限り値が初期化されることがありません。
そのため、目的のログ出力処理が完了後にMDC#remove(key)メソッドで、MDCのキー “application.func” の値を初期化しています。

出力例

%CONTEXT_PATH%/WEB-INF/conf/log/im_logger.xmlの<configuration>/<appender name=”STDOUT”>/<encoder>/<pattern>の内容を以下に変更し、アプリケーションサーバを再起動します。

```
[%level] %logger{10} - %X{application.func} %msg%n
```

ログレベルtraceでadd(1, 2)を実行した時のログ出力

```
[DEBUG] t.c.AddController - add arguments=[1, 2]
[TRACE] t.c.AddController - add result=3
```

TERASOLUNA Server Framework for Java (5.x) for Accel Platform のログ出力設定について

WEB-INF/conf/log/im_logger_tgfw.xml にログ出力の設定を記載しています。
標準で、 WEB-INF/log/platform/im_tgfw.log ファイルにログを出力します。
運用環境に合わせて設定を変更してください。
標準の設定は以下の通りです。

ロガー名	ログレベル
jp.co.intra_mart.framework.extension.spring	warn
jp.co.intra_mart.system.router.spring	warn
org.springframework	warn
org.springframework.web.servlet	info

ロガー名	ログレベル
org.terasoluna.gfw	info
org.terasoluna.gfw.common.exception.ExceptionLogger	info
org.terasoluna.gfw.common.exception.ExceptionLogger.Monitoring	none

TERASOLUNA Server Framework for Java (5.x) の TraceLoggingInterceptor を利用する。

TraceLoggingInterceptorは、リクエストURLにマッピングされたメソッドの処理時間を出力する機能です。
TERASOLUNA Server Framework for Java (5.x) for Accel Platform では、 applicationContext-im_tgfw_web.xml と im_logger_tgfw.xml に設定をすることによって、この機能を利用できます。
TraceLoggingInterceptor の詳細については、 TERASOLUNA Server Framework for Java (5.x) Development Guideline の TraceLoggingInterceptorの項 を参考にしてください。

applicationContext-im_tgfw_web.xml の設定
mvc:interceptorsにTraceLoggingInterceptorの設定を追加します。

```
<!-- MVC interceptors -->
<mvc:interceptors>

  <!-- 中略 -->

  <!-- TraceLoggingInterceptorを追加する。 -->
  <mvc:interceptor>
    <mvc:mapping path="/**" />
    <bean class="org.terasoluna.gfw.web.logging.TraceLoggingInterceptor">
      <property name="warnHandlingNanos" value="1000000000" />
    </bean>
  </mvc:interceptor>
</mvc:interceptors>
```

> **コラム**
>
> mvc:mapping の path 属性は、トレースログを出力する対象の Controller クラスのパッケージに絞って設定してください。

im_logger_tgfw.xml の設定
org.terasoluna.gfw.web.logging.TraceLoggingInterceptor のログ出力設定を追加します。

```
<logger name="org.terasoluna.gfw.web.logging.TraceLoggingInterceptor" additivity="false">
  <level value="trace" />
  <appender-ref ref="IM_TGFW" />
</logger>
```

UI（デザインガイドライン）

「UI（デザインガイドライン）」は、別ドキュメントの UIデザインガイドライン（ PC 版） へ移行しました。

UI（スマートフォン開発ガイドライン）

概要

項目

- IM-Mobile Frameworkについて
- jQuery Mobile

IM-Mobile Frameworkについて

IM-Mobile Frameworkは、スマートフォン向けに最適化したWebサイト作成のためのモジュールです。
本機能を利用することで、入力部品やページデザイン等に jQuery Mobile を利用し、統一的なデザインでスマートフォン向けWeb サイトを

構築できます。

jQuery Mobile

jQuery Mobileとは、jQueryのプラグインとして稼働するスマートフォン用ライブラリです。
jQuery Mobileを利用すると、開発者がインタフェースを意識しなくてもスマートフォン用に最適化されたWeb画面を作れます。
IM-Mobile Frameworkをインストールするとjquery Mobileを利用する環境が整いますので、開発者は改めてjQuery Mobileをインストールする必要はありません。

jQuery Mobileの詳細については下記サイトを参照してください。

- jQuery Mobile

http://jquerymobile.com/
http://jquerymobile.com/demos/

> **注意**
> 本ガイドは jQuery Mobile のバージョン 1.3.0 をベースに作成しています。
> jQuery Mobile のバージョン 1.4.5 を利用した場合とレイアウトが異なる場合があります。
> また、1.4.5 で非推奨となったメソッドもサンプルコードにて利用しています。
> http://jquerymobile.com/upgrade-guide/1.4 を参照し、非推奨メソッドなどは適宜置き換えて実装してください。

クライアントタイプとテーマ

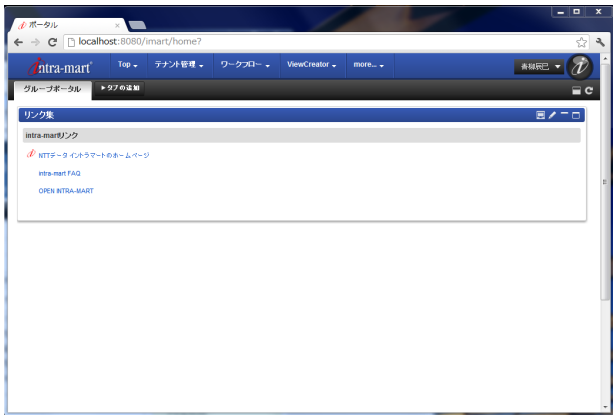
項目

- クライアントタイプとスマートフォン版テーマ
- クライアントタイプの変更
  - PCブラウザからスマートフォン版画面を表示する
  - スマートフォンからPC版画面を表示する

クライアントタイプとスマートフォン版テーマ

IM-Mobile Framework ではブラウザのユーザエージェントを参照し、PCブラウザおよびスマートフォンのブラウザの２種類に判別します。
クライアントタイプがスマートフォンである場合、IM-Mobile Frameworkではスマートフォン版テーマが適用されます。

- PCブラウザからHOME画面にアクセスした例



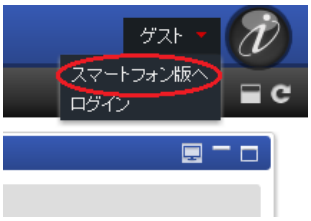
- スマートフォンブラウザからHOME画面にアクセスした例。スマートフォン版テーマが適用されている

クライアントタイプの変更

クライアントタイプはユーザが明示的に変更することもできます。

PCブラウザからスマートフォン版画面を表示する

ヘッダ右のユーザ名ドリルダウンから「スマートフォン版へ」リンクを押下します。



スマートフォンからPC版画面を表示する

HOME画面のフッタバー右下「...」を押下し、「PC版へ」ボタンを押下します。

スマートフォン版テーマ

項目

- スマートフォン版テーマとは
  - テーマが読み込むライブラリ群の切り替え
- テーマとスウォッチ
  - スウォッチ
    - jQuery Mobile 1.3.0
    - jQuery Mobile 1.4.5

スマートフォン版テーマとは

スマートフォン版テーマとは、スマートフォン用に最適化されたテーマのことを指します。
<HTML>タグや<HEAD>タグなど冗長的な記述を省き、予め必要なライブラリ群をロードした環境下で開発できるため、 開発者は改めてjQuery/jQuery Mobileなどのライブラリ群を定義する必要はありません。

- 一般的なjQuery Mobileのコーディング例。

```
<!DOCTYPE html>
<html>
 <head>
  <title>My Page</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.css" />
  <script type="text/javascript" src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
  <script type="text/javascript" src="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.js"></script>
 </head>
 <body>
  <div data-role="page">
   <div data-role="header">
    <h1>My Title</h1>
   </div>
   <div data-role="content">
    <p>Hello world</p>
   </div>
   <div data-role="footer">
    <h1>My Title</h1>
   </div>
  </div>
 </body>
</html>
```

- スマートフォン版テーマを利用した場合のコーディング例。
<HEAD>タグの一部のみ<imui:head>タグで定義し、その他はBODYタグの内部のみ記述します。

```
<imui:head>
 <title>My Page</title>
</imui:head>
<div data-role="page">
 <div data-role="header">
  <h1>My Title</h1>
 </div>
 <div data-role="content">
  <p>Hello world</p>
 </div>
 <div data-role="footer">
  <h1>My Title</h1>
 </div>
</div>
```

コラム
テーマの詳細は、別ドキュメント テーマ仕様書 の PageBuilder を参照してください。

テーマが読み込むライブラリ群の切り替え

intra-mart Accel Platform 2015 Summer(Karen) では、jQuery Mobile 1.4.5 が導入されたため、読み込むライブラリ群のバージョンの切り替えが可能になりました。
jQuery Mobile 1.4.5 とそれに対応するライブラリ群と、jQuery Mobile 1.3.0 とそれに対応するライブラリ群を、画面ごとに切り替えることができます。
例えば、%CONTEXT_PATH%/sample/mobile_fw 配下をすべて jQuery Mobile 1.4.5 で実装したい場合は以下のような設定ファイルを用意します。

- %CONTEXT_PATH%/WEB-INF/conf/theme-full-theme-path-config/mobile_fw.xml

```
<theme-full-theme-path-config
  xmlns="http://www.intra-mart.jp/theme/theme-full-theme-path-config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.intra-mart.jp/theme/theme-full-theme-path-config theme-full-theme-path-config.xsd ">

  <path regex="true" client-type="sp" libraries-version="iap-8.0.11">/sample/mobile_fw/.*</path>

</theme-full-theme-path-config>
```

上記の設定ファイルを読み込み、*http://<HOST>:<PORT>/<CONTEXT_PATH>/sample/mobile_fw* 配下のjspへアクセスすると、jQuery Mobile 1.4.5 とそれに対応するライブラリ群を読み込んだ画面が表示されます。

> **コラム**
> 設定ファイルの詳細は、別ドキュメント テーマの適用方法設定 FullThemeBuilder を参照してください。

テーマとスウォッチ

スマートフォン版テーマのデザインは、jQuery Mobileのテーマを拡張することによって実現しています。
jQuery Mobileのテーマには「スウォッチ」と呼ばれる複数のカラーデザインパターンが含まれており、開発者は指定の要素にスウォッチを指定することによって、用途に応じてスウォッチを使い分けながら画面を作成できます。

スウォッチ

intra-mart Accel Platform 2015 Summer(Karen) から、jQuery Mobile 1.4.5 が導入されたため、バージョンによってレイアウトの違いが生じるようになりました。
バージョンアップによる主な違いは以下です。

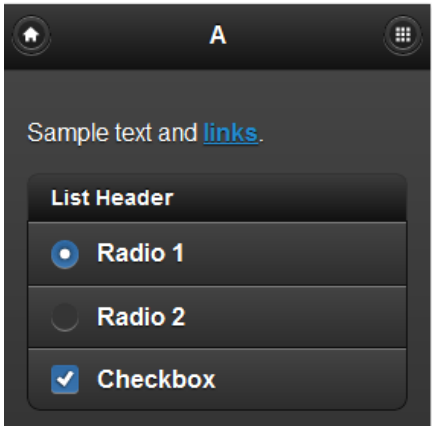
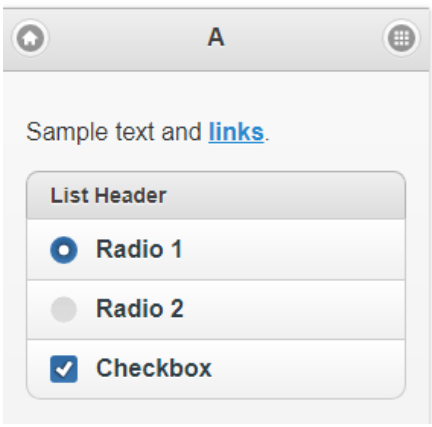
- デフォルトテーマが c から a 変更になったため、data-theme を指定していない画面は a（黒色）のテーマが適用されるようになった
- フラットデザインになったため、ボーダーやグラデーションがほとんどなくなった

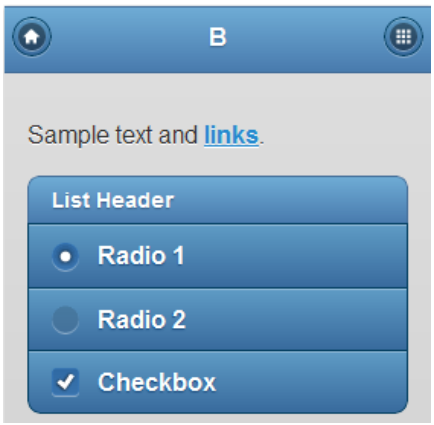
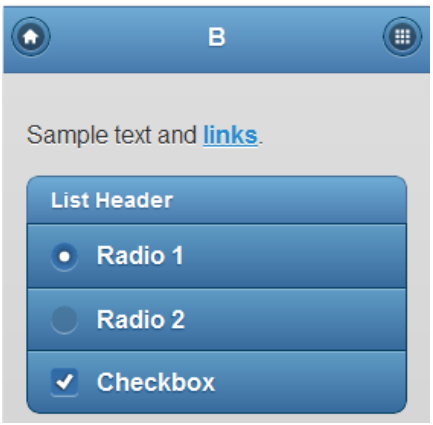
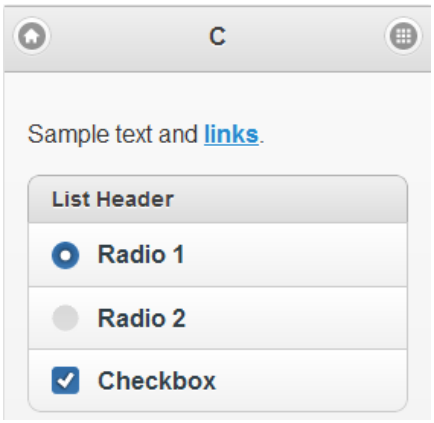
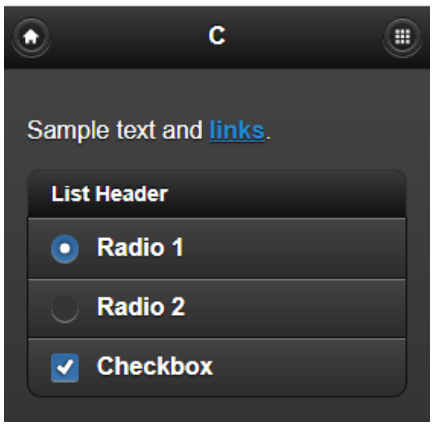
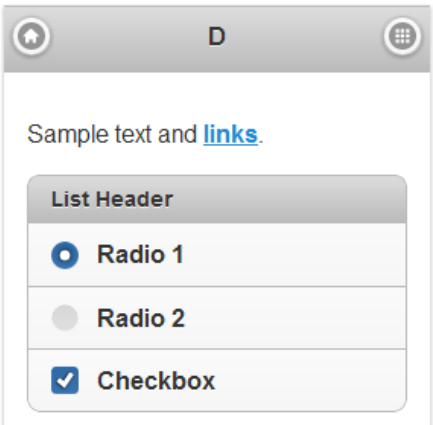
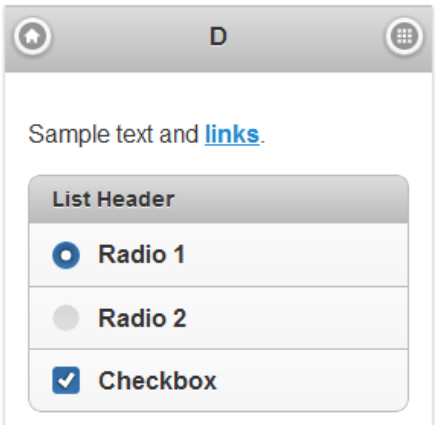
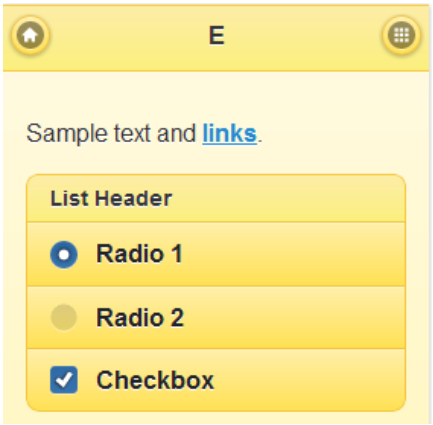
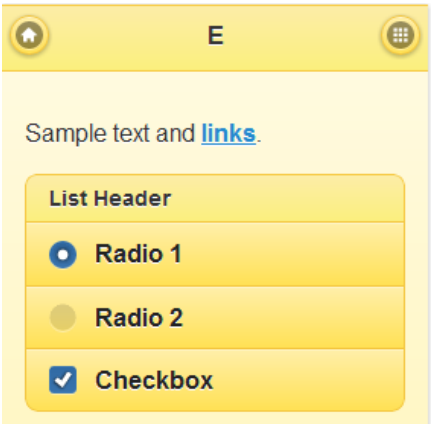
> **コラム**
> その他の変更点や変更点の詳細は http://jquerymobile.com/upgrade-guide/1.4 を参照してください。

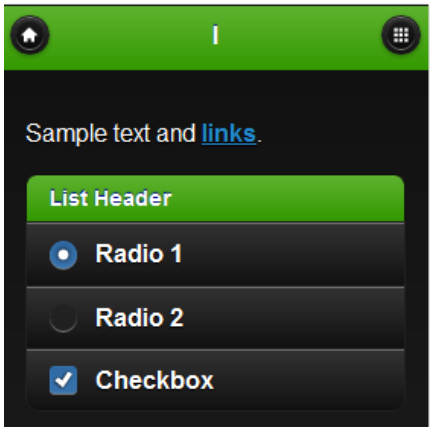
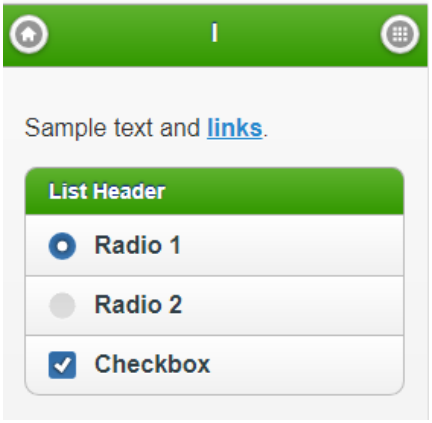
> **コラム**
> 2018 Summer(Tiffany) より標準テーマ白が追加されました。既存の標準テーマ黒を利用している場合とスウォッチの色が異なります。
> 各テーマのスウォッチについては下記を参照してください。

jQuery Mobile 1.3.0

intra-mart Accel Platform では、jQuery Mobileデフォルトスウォッチの「A」～「E」のほかに、拡張スウォッチ「F」「I」を使用できます。

設定	説明	黒テーマイメージ	白テーマイメージ
A	jQueryMobile、intra-mart Accel Platform 標準色		

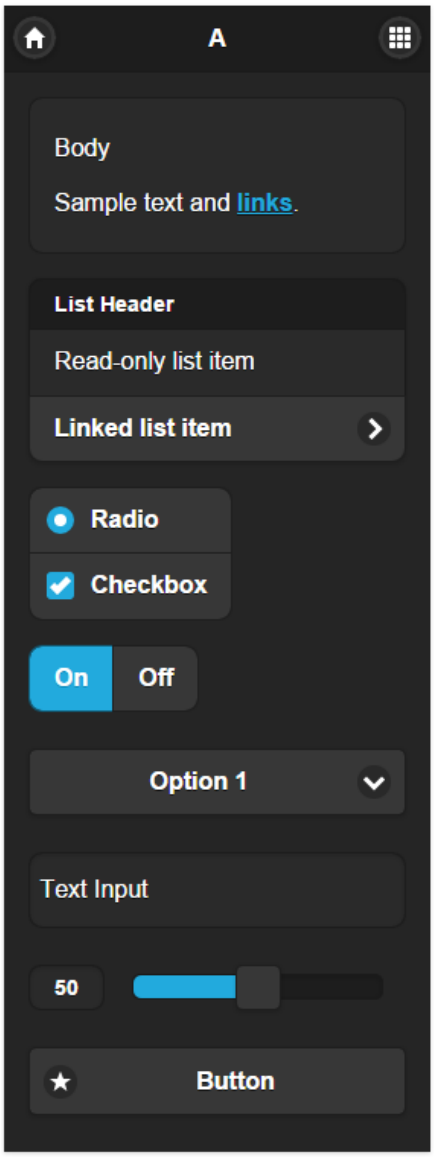
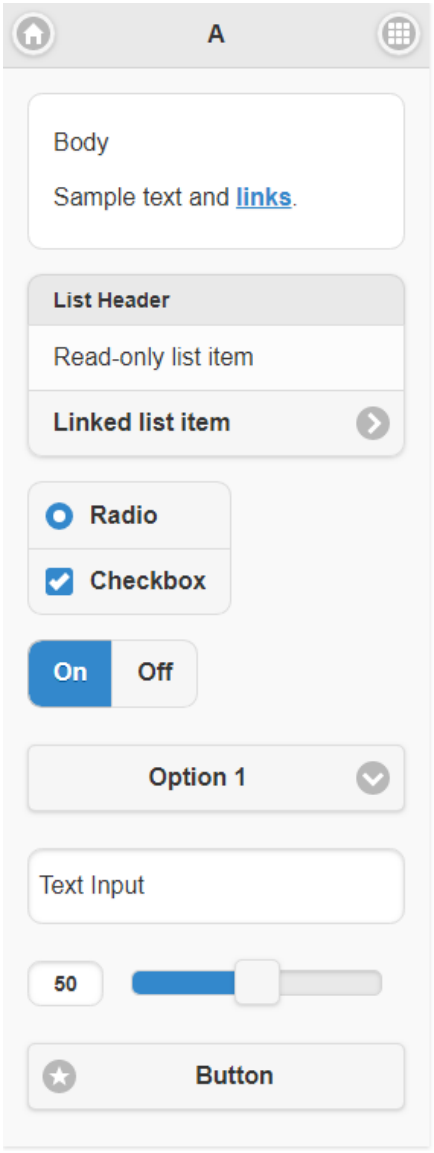
設定	説明	黒テーマイメージ	白テーマイメージ
B			
C	jQueryMobile標準色		
D			
E			

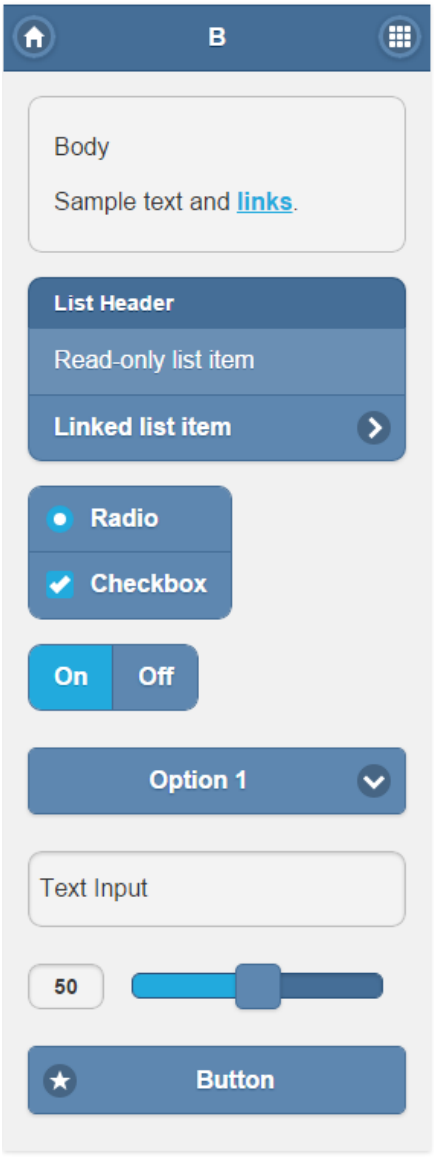
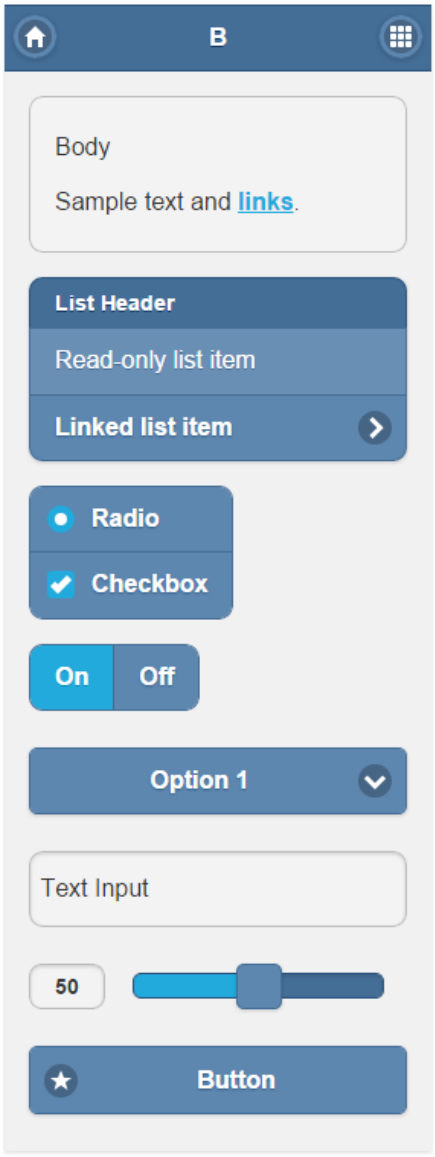
設定	説明	黒テーマイメージ	白テーマイメージ
F	追加スウォッチ		
I	追加スウォッチ		

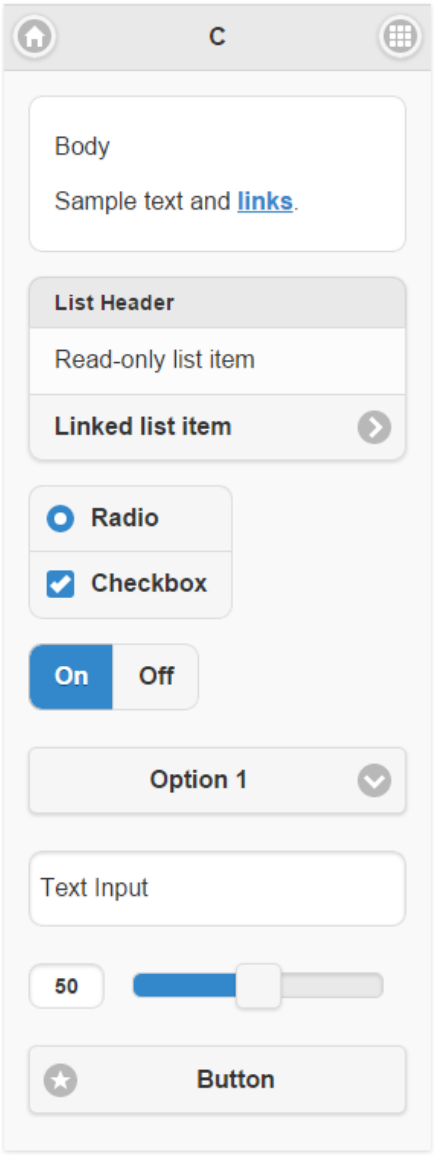
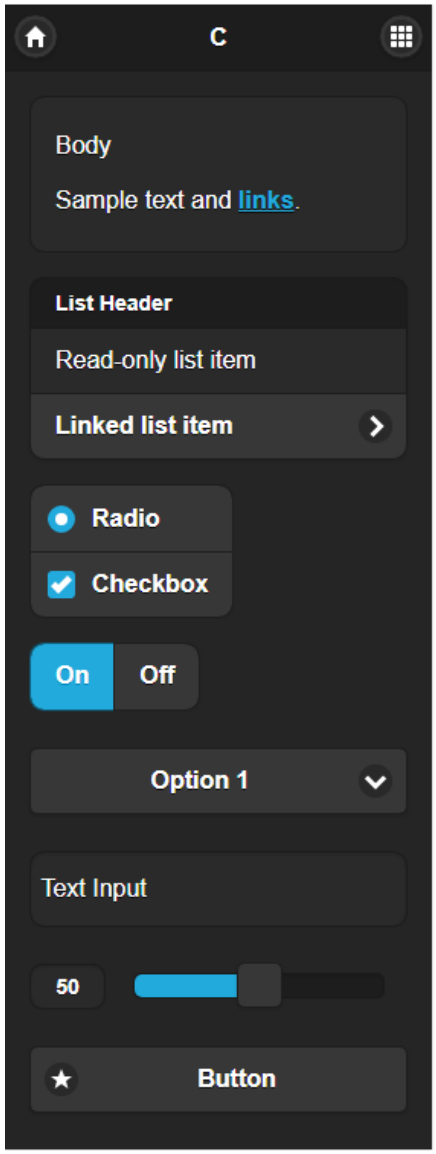
jQuery Mobile 1.4.5

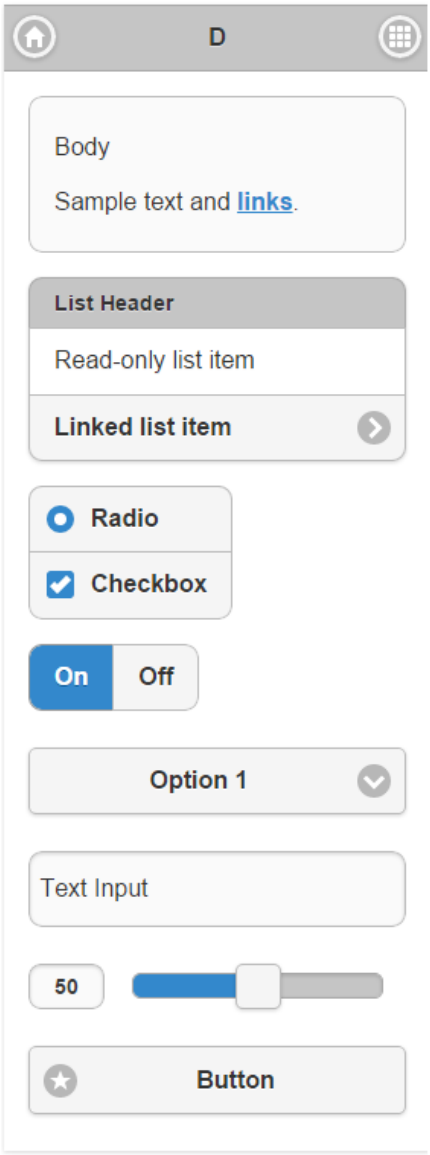
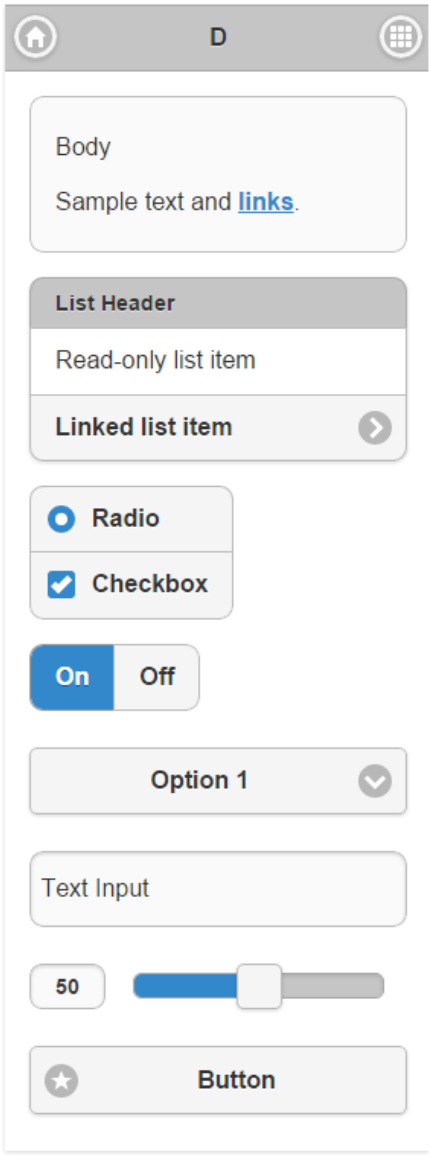
intra-mart Accel Platform では、jQuery Mobileデフォルトスウォッチの「A」～「E」のほかに、拡張スウォッチ「F」「I」を使用できます。
「A」が jQueryMobile標準色へ変更になりました。

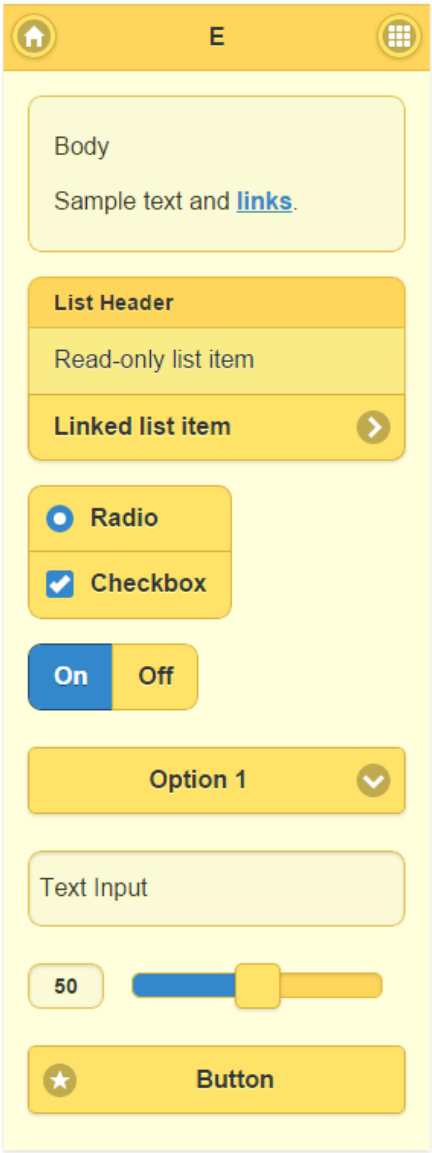
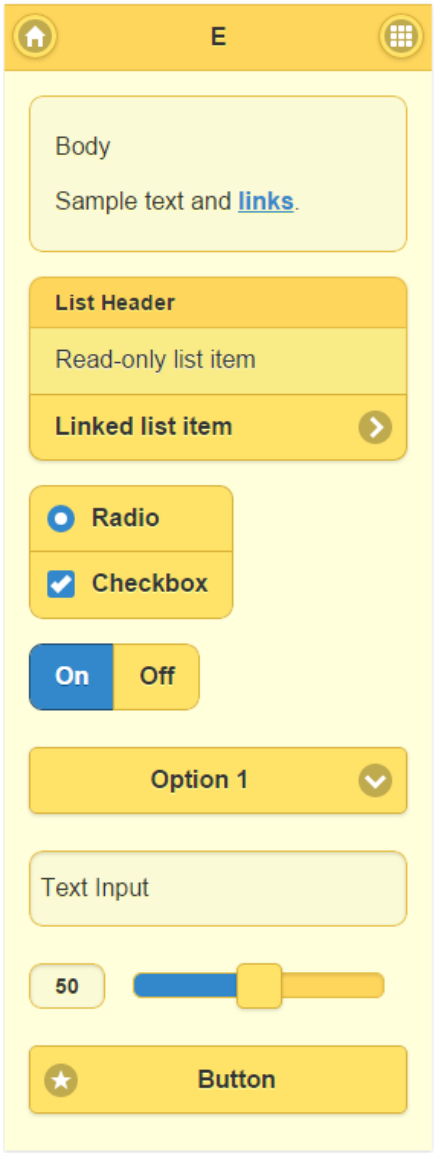
設定	説明	黒テーマイメージ	白テーマイメージ

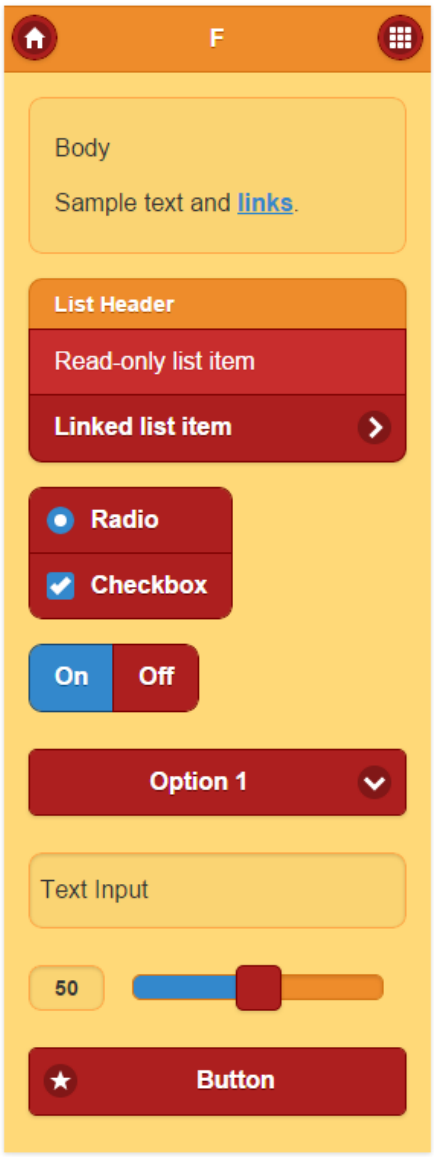
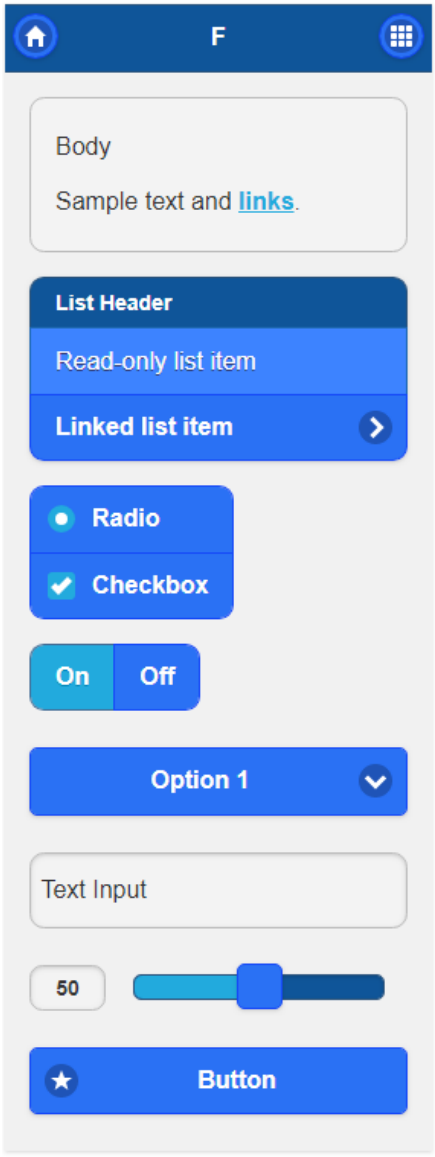
設定	説明	黒テーマイメージ	白テーマイメージ
A	jQueryMobile、intra-mart Accel Platform 標準色		

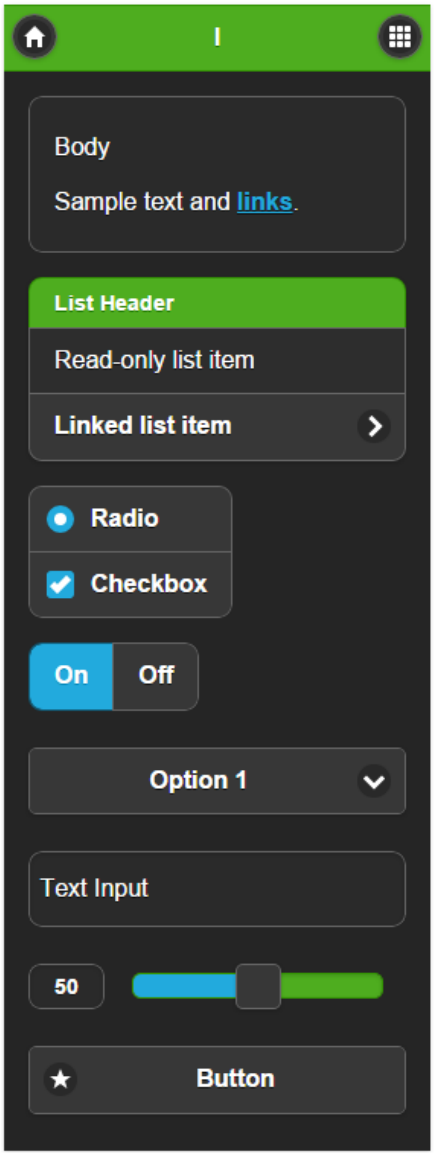
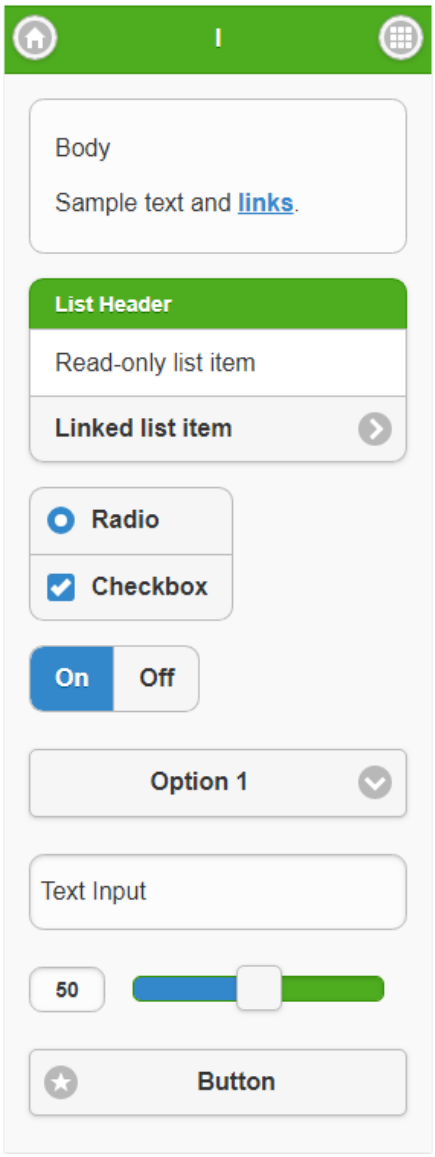
設定	説明	黒テーマイメージ	白テーマイメージ
B			

設定	説明	黒テーマイメージ	白テーマイメージ
C			

設定	説明	黒テーマイメージ	白テーマイメージ
D			

設定	説明	黒テーマイメージ	白テーマイメージ
E			

設定	説明	黒テーマイメージ	白テーマイメージ
F	追加スウォッチ		

設定	説明	黒テーマイメージ	白テーマイメージ
l	追加スウォッチ		

基本的な画面の作り方

本項では IM-Mobile Frameworkを利用した基本的な画面開発方法について説明します。

> **コラム**
>
> 本項では IM-Mobile Frameworkを利用する上で最低限知っておくべきjQuery Mobileの使用方法を紹介しています。
> jQuery Mobileのより具体的な使用方法についてはjQuery Mobileのリファレンスを参照してください。

項目

- Hello IM-Mobile Framework!を作る
  - アクションクラスを用意する
  - JSPファイルを用意する
  - プロジェクト固有のapplicationContext.xmlの作成
  - マークアップの説明
- スウォッチを指定する

Hello IM-Mobile Framework!を作る

アクションクラスを用意する

- 以下のようなControllerクラスを作成します。

```
package jp.co.intra_mart.sample.spring.imsp.app.hello;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("sample/spring/sp/hello")
public class HelloController {

    @RequestMapping({"", "/"})
    public String index(Model model) {
        return "sample/spring/imsp/hello/index.jsp";
    }
}
```

JSPファイルを用意する

以下のファイルを作成し、%CONTEXT_PATH%/WEB-INF/views/sample/spring/imsp/helloフォルダに保存します。

- index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="imsp" uri="http://www.intra-mart.co.jp/taglib/imsp" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<imui:head>
  <title>Hello Mobile Framework!</title>
</imui:head>
<div data-role="page">
  <div data-role="header">
    <h3>Header</h3>
  </div>
  <div data-role="content">
    <p>Hello IM-Mobile Framework!</p>
  </div>
  <div data-role="footer">
    <h3>Footer</h3>
  </div>
</div>
```

プロジェクト固有のapplicationContext.xmlの作成

- 以下のファイルを作成し、%CONTEXT_PATH%/WEB-INF/classes/META-INF/springに保存します。
  applicationContext-guideline-imsp.xml

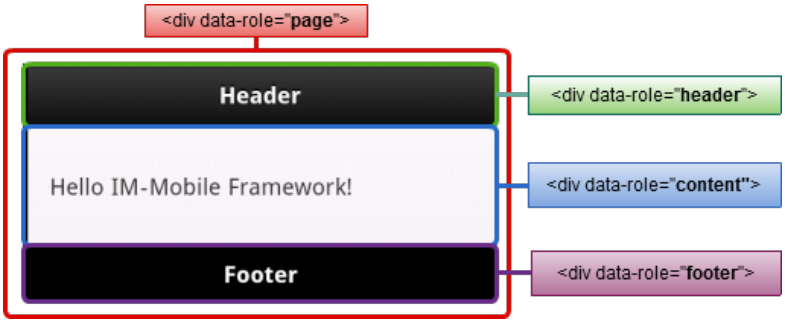
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd">

    <!-- DIコンポーネントの対象とする要素のトップレベルパッケージ -->
    <context:component-scan base-package="jp.co.intra_mart.sample.spring.imsp" />

</beans>
```

- スマートフォンから以下のURLにアクセスします。
  `http://<HOST>:<PORT>/<CONTEXT_PATH>/sample/spring/sp/hello`

コラム

- PCブラウザから動作確認する場合はスマートフォン版テーマに切り替える必要があります。
  PCブラウザからスマートフォン版画面を表示する

マークアップの説明



-
  1ページ当たりのブロック要素であることを定義します。
  最も上位のブロック要素であり、以下３つの要素は必ずこの要素内に定義する必要があります。
-
  ヘッダのブロック要素であることを定義します。
  任意の要素です。
-
  コンテンツのブロック要素であることを定義します。
  必須の要素です。
-
  フッタのブロック要素であることを定義します。
  任意の要素です。

スウォッチを指定する

jQuery Mobileでは各要素に *スウォッチ* を指定できます。
先ほどの作成したプレゼンテーションページの各要素に属性data-theme=”b”を追記します。

```
<div data-role="page" data-theme="b">
  <div data-role="header" data-theme="b">
    <h3>Header</h3>
  </div>
  <div data-role="content">
    <p>Hello IM-Mobile Framework!</p>
  </div>
  <div data-role="footer" data-theme="b">
    <h3>Footer</h3>
  </div>
</div>
```

再表示すると、以下の様に各要素が青系色で装飾されます。

コラム

この項目では、下記のポイントを確認しました。

- ページ要素は で定義する
- ヘッダ要素は で定義する
- コンテンツ要素は で定義する
- フッタ要素は で定義する

実装例：登録画面を作る

この項では、スマートフォンでTODOを登録する画面の実装例を紹介します。

項目

- 前提条件
  - 下準備 テーブル作成
- 画面を表示できるようにする
  - ソースの準備と配置
  - メニューの作成
  - 認可の設定
  - 画面を表示する
- 画面に要素を配置する
- 登録処理を実装する
- 入力チェック処理を実装する（クライアントサイド）
- 非同期で登録処理を実行する
- 最終結果

前提条件

- intra-mart Accel Platform をインストールし、初期設定までが完了していること。
- ベースモジュールに TERASOLUNA Server Framework for Java (5.x) for Accel Platform およびIM-Mobile Frameworkモジュールを含めて環境を作成してください。
  実行環境は単体テスト用で作成してください。

下準備 テーブル作成

以下手順を行う前に、以下のテーブルを作成してください。

- mfw_sample

列名	データ型	主キー	NOT NULL	説明
todo_id	VARCHAR(20)	○	○	レコードのID
user_cd	VARCHAR(20)		○	登録ユーザID
user_nm	VARCHAR(20)		○	登録ユーザ名
limit_date	VARCHAR(20)			TODOの期限
title	VARCHAR(100)			TODOのタイトル
comment	VARCHAR(1000)			コメント
progress	NUMBER(3)			進捗度
complete	VARCHAR(1)			完了/未完了
priority	VARCHAR(1)			重要度
timestmp	VARCHAR(20)			タイムスタンプ

サンプルテーブルのCREATE文（PostgreSQL）

※その他のデータベースの場合は環境に合わせて調整してください。

画面を表示できるようにする

ソースの準備と配置

まず、以下２点のファイルを作成します。

- jp.co.intra_mart.sample.spring.imsp.app.store.StoreController

```
package jp.co.intra_mart.sample.spring.imsp.app.store;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("sample/spring/sp/store")
public class StoreController {

   @RequestMapping
   public String index(Model model) {
      return "sample/spring/imsp/store/index.jsp";
   }
}
```

- %CONTEXT_PATH%/WEB-INF/views/sample/spring/imsp/store/index.jsp

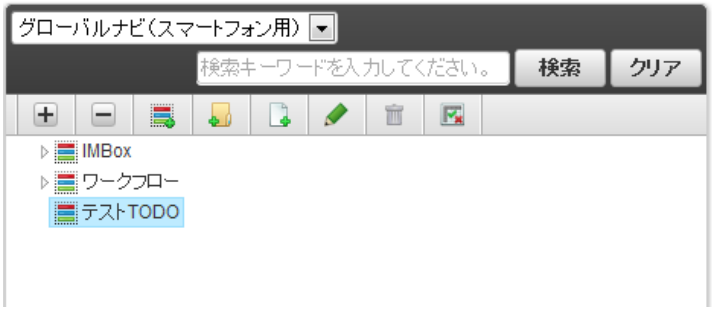
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="imsp" uri="http://www.intra-mart.co.jp/taglib/imsp" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<imui:head>
  <title>TODO登録</title>
</imui:head>
<div data-role="page" id="main" data-theme="a">
  <imsp:headerWithLink headerText="TODO登録" />
  <div data-role="content">
  </div>
  <imsp:commonFooter dataPosition="fixed" />
</div>
```

> **コラム**
>
> - スマートフォン向けタグライブラリを使用するには以下のtaglibディレクティブを指定してください。
>   `<%@ taglib prefix="imsp" uri="http://www.intra-mart.co.jp/taglib/imsp" %>`
> - <imsp:headerWithLink> - ヘッダ部左端に、任意のページに遷移のボタンを備えたヘッダを表示します。
> - <imsp:spCommonFooter> - フッタ部にHOMEボタンとログアウトボタンを表示します。

メニューの作成

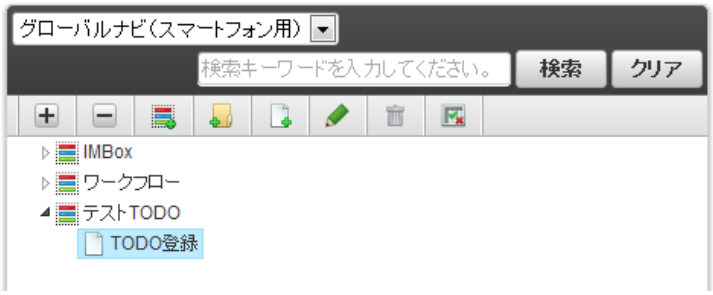
メニューの設定を行います。

- PCブラウザからテナント管理者でログインし、「メニュー設定」画面を表示します。
- グローバルナビ（スマートフォン用）を選択し、新規メニューグループ「テストTODO」を作成します。



新規メニューアイテムを作成します。

- メニューアイテム名を「TODO登録」とし、URLを”sample/spring/sp/store”とします。



認可の設定

認可を設定します。

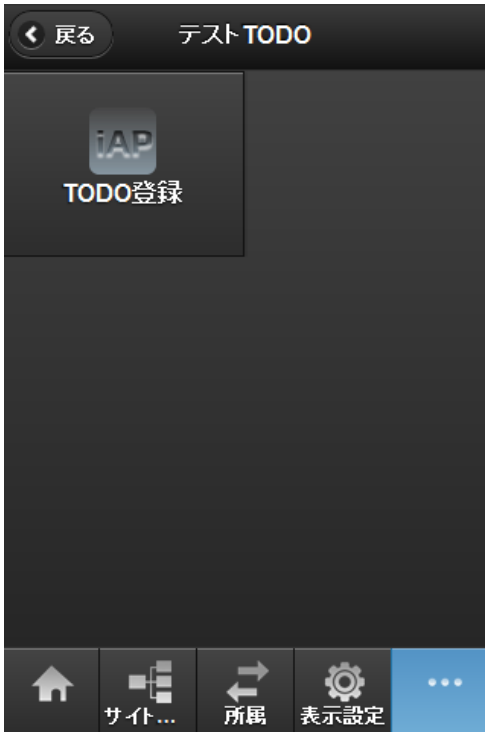
- 「権限設定」ボタンを押下し権限設定（グローバルナビ（スマートフォン用））を表示します。
- 「権限設定を開始する」ボタンをクリックします。
- テストTODOの権限の「参照」権限を認証済みユーザに付与します。

> **コラム**
> - 認可の詳細については *認可* を参照してください。

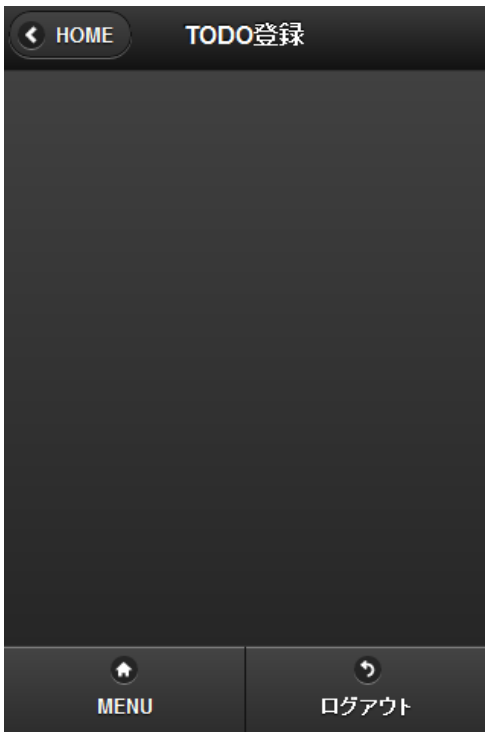
画面を表示する

作成したページをメニューから表示します。

- クライアントタイプをスマートフォン版へ切り替え、スマートフォン版グローバルナビを表示します。
- メニューから「テストTODO」を選択すると、先ほど作成されたメニュー「TODO登録」が表示されます。



- TODO登録を選択します。目的の画面を表示できました。



> **コラム**
>
> この項目では、下記のポイントを確認しました。
>
> - スマートフォン用グローバルナビにメニューを表示するには、メニューカテゴリ「グローバルナビ（スマートフォン用）」に設定する

画面に要素を配置する

内に要素を配置します。
例としてテキストボックスとラベルを配置してみます。
ラベルを配置するには<imsp:fieldContain>タグを使用します。

- store.jsp

```
<imsp:fieldContain label="TODO名" required="true">
  <input type="text" name="title" />
</imsp:fieldContain>
```

- マークアップ結果。テーマにより最適化されたテキストボックスが表示されました。

同様に他の要素も配置していきます。

- imsp:datePicker–日付文字列を参照入力するためのインタフェースを提供します。

```
<imsp:fieldContain label="期限" required="true">
  <imsp:datePicker name="limitDate" />
</imsp:fieldContain>
```

- textarea–テキストエリアを提供します。

```
<imsp:fieldContain label="コメント">
  <textarea name="comment"></textarea>
</imsp:fieldContain>
```

- imsp:controlGroup–フォーム要素をグループ化します。
- imsp:radioButton–jQuery mobileで最適化されたラジオボタンを提供します。

```
<imsp:controlGroup label="重要度">
  <imsp:radioButton name="priority" id="radio1" value="0" label="低" />
  <imsp:radioButton name="priority" id="radio2" value="1" label="中" />
  <imsp:radioButton name="priority" id="radio3" value="2" label="高" />
</imsp:controlGroup>
```

- imsp:slider–スライダーを提供します。

```
<imsp:fieldContain label="進捗">
  <imsp:slider name="progress" min="<%=0 %>" max="<%=100 %>" />
</imsp:fieldContain>
```

- imsp:toggle–トグルスイッチを提供します。

```
<imsp:fieldContain label="完了">
  <imsp:toggle name="complete" onLabel="完了" offLabel="未完了" onValue="1" offValue="0" />
</imsp:fieldContain>
```

- マークアップ結果。各要素が表示されました。

その他使用可能な要素についてはAPIリストを参照してください。

> **コラム**
>
> この項目では、下記のポイントを確認しました。
>
> - フォーム要素は 内に配置する
> - ラベルを与える場合は<imsp:fieldContain>タグを使用する

登録処理を実装する

まず、フォームを受け取るためのFormクラスを作成します。

- jp.co.intra_mart.sample.spring.imsp.app.store.StoreForm

```
package jp.co.intra_mart.sample.spring.imsp.app.store;

public class StoreForm {
    private String title;
    private String limitDate;
    private String comment;
    private String priority;
    private String progress;
    private String complete;

    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getLimitDate() {
        return limitDate;
    }
    public void setLimitDate(String limitDate) {
        this.limitDate = limitDate;
    }
    public String getComment() {
        return comment;
    }
    public void setComment(String comment) {
        this.comment = comment;
    }
    public String getPriority() {
        return priority;
    }
    public void setPriority(String priority) {
        this.priority = priority;
    }
    public String getProgress() {
        return progress;
    }
    public void setProgress(String progress) {
        this.progress = progress;
    }
    public String getComplete() {
        return complete;
    }
    public void setComplete(String complete) {
        this.complete = complete;
    }
}
```

次に登録処理用のエンティティクラスを作成します。

- jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample

```
package jp.co.intra_mart.sample.spring.imsp.domain.model;

import java.io.Serializable;
import java.math.BigDecimal;

public class MfwSample implements Serializable {

    private static final long serialVersionUID = 1L;

    private String todoId;

    private String comment;

    private String complete;

    private String limitDate;

    private String priority;
```

```
    private BigDecimal progress;

    private String timestmp;

    private String title;

    private String userCd;

    private String userNm;

    public MfwSample() {
    }

    public String getComment() {
        return comment;
    }

    public String getComplete() {
        return complete;
    }

    public String getLimitDate() {
        return limitDate;
    }

    public String getPriority() {
        return priority;
    }

    public BigDecimal getProgress() {
        return progress;
    }

    public String getTimestmp() {
        return timestmp;
    }

    public String getTitle() {
        return title;
    }

    public String getTodoId() {
        return todoId;
    }

    public String getUserCd() {
        return userCd;
    }

    public String getUserNm() {
        return userNm;
    }

    public void setComment(final String comment) {
        this.comment = comment;
    }

    public void setComplete(final String complete) {
        this.complete = complete;
    }

    public void setLimitDate(final String limitDate) {
        this.limitDate = limitDate;
    }

    public void setPriority(final String priority) {
        this.priority = priority;
    }

    public void setProgress(final BigDecimal progress) {
        this.progress = progress;
    }
```

```
    public void setTimestmp(final String timestmp) {
        this.timestmp = timestmp;
    }

    public void setTitle(final String title) {
        this.title = title;
    }

    public void setTodoId(final String todoId) {
        this.todoId = todoId;
    }

    public void setUserCd(final String userCd) {
        this.userCd = userCd;
    }

    public void setUserNm(final String userNm) {
        this.userNm = userNm;
    }
}
```

mybatis-config.xml の typeAliasesにエンティティのパッケージを指定します。

```
<typeAliases>
    <package name="jp.co.intra_mart.sample.spring.imsp.domain.model" />
</typeAliases>
```

登録処理用のリポジトリインタフェースを定義します。

```
package jp.co.intra_mart.sample.spring.imsp.domain.repository;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;

public interface MfwSampleRepository {

    void create(MfwSample entity);
}
```

コラム

MyBatis3については、 *データベース* を参照してください。

リポジトリのパッケージと同じ構成のディレクトリにMfwSampleRepository.xmlを作成します。

```
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="jp.co.intra_mart.sample.spring.imsp.domain.repository.MfwSampleRepository">

  <insert id="create" parameterType="MfwSample">
    <![CDATA[
    INSERT INTO iap.mfw_sample (
      todo_id,
      user_cd,
      user_nm,
      title,
      comment,
      timestmp,
      priority,
      limit_date,
      progress,
      complete)
    VALUES (
      #{todoId},
      #{userCd},
      #{userNm},
      #{title},
      #{comment},
      #{timestmp},
      #{priority},
      #{limitDate},
      #{progress},
      #{complete})
    ]]>
  </insert>
</mapper>
```

applicationContext-im_tgfw_mybatis3.xml のbase-packageの属性値をリポジトリのパッケージに修正し、定義したリポジトリのパッケージをスキャン対象に設定します。

```
<!-- setting 'base-package' which is the package in which mapper interface is. -->
<mybatis:scan base-package="jp.co.intra_mart.sample.spring.imsp.domain.repository" />
```

ビジネスロジック処理を実現するため、サービスクラスのインタフェースと実装クラスを定義します。

- jp.co.intra_mart.sample.spring.imsp.domain.service.MfwSampleService

```
package jp.co.intra_mart.sample.spring.imsp.service;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;

public interface MfwSampleService {
  void store(MfwSample entity);
}
```

- jp.co.intra_mart.sample.spring.imsp.domain.service.MfwSampleServiceImpl

```
package jp.co.intra_mart.sample.spring.imsp.service;

import javax.inject.Inject;

import org.springframework.stereotype.Service;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;
import jp.co.intra_mart.sample.spring.imsp.domain.repository.MfwSampleRepository;

@Service
public class MfwSampleServiceImpl implements MfwSampleService {

    @Inject
    MfwSampleRepository mfwSampleRepository;

    @Override
    public void store(MfwSample entity) {
        mfwSampleRepository.create(entity);
    }
}
```

- StoreControllerクラスを以下のように修正します。

```
package jp.co.intra_mart.sample.spring.imsp.app.store;

import java.io.IOException;

import javax.inject.Inject;

import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.i18n.datetime.DateTime;
import jp.co.intra_mart.foundation.i18n.datetime.format.DateTimeFormatter;
import jp.co.intra_mart.foundation.service.client.information.Identifier;
import jp.co.intra_mart.foundation.user_context.model.UserContext;
import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;
import jp.co.intra_mart.sample.spring.imsp.service.MfwSampleService;

import net.arnx.jsonic.JSON;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("sample/spring/sp/store")
public class StoreController {

    // ①
    @Inject
    MfwSampleService mfwSampleService;

    // ②
    @ModelAttribute
    public StoreForm setupStoreForm() {
        return new StoreForm();
    }

    @RequestMapping
    public String index(Model model) {
        return "sample/spring/imsp/store/index.jsp";
    }

    // ③
    @RequestMapping("store")
    public String store(Model model, StoreForm storeForm) throws IOException {

        //ユーザコンテキストを取得します。
        UserContext userProfile = Contexts.get(UserContext.class);
        //アカウントコンテキストからタイムゾーンを取得します。
        AccountContext accountContext = Contexts.get(AccountContext.class);
        DateTimeFormatter formatter = DateTimeFormatter.withPattern("yyyy/MM/dd HH:mm");

        //登録値の設定
        MfwSample insertData = new MfwSample();
        // 略
        // storeFormからinsertDataへコピー

        insertData.setTodoId(new Identifier().get());
        insertData.setUserCd(userProfile.getUserProfile().getUserCd());
        insertData.setUserNm(userProfile.getUserProfile().getUserName());
        insertData.setTimestmp(formatter.format(DateTime.now(accountContext.getTimeZone())));

        mfwSampleService.store(insertData);

        return "redirect:/sample/spring/sp/store";
    }
}
```

1. サービスクラスをバインドします。

2. StoreFormを返却するメソッドを定義します。
   @ModelAttributeにより、jspから変数”storeForm”として

フォームの値を参照できます。

3. 登録処理のメソッドです。
   引数にStoreFormを設定します。

> **コラム**
> - データベースの詳細については *データベース* を参照してください。
> - コンテキストの詳細については *アクセスコンテキスト* を参照してください。

- 画面にFormタグと登録ボタンを配置します。 formタグのaction属性は先ほどstoreメソッドに設定したリクエストマッピングに合わせます。

```
<form name="storeForm" id="storeForm" method="POST" action="sample/spring/sp/store/store" data-ajax="false">
  <imsp:fieldContain label="TODO名" required="true">
    <input type="text" name="title" />
  </imsp:fieldContain>
  <imsp:fieldContain label="期限" required="true">
    ...
  <imsp:fieldContain label="完了">
    <imsp:toggle name="complete" onLabel="完了" offLabel="未完了" onValue="1" offValue="0" />
  </imsp:fieldContain>
    <a data-role="button" data-theme="b" id="storeButton">登録</a>
</form>
```

> **コラム**
> - リンクにdata-role=”button”属性を付加するとリンクをボタン表示します。
> - リンクまたはFORM要素にdata-ajax=”false”属性を付加することで 明示的にAjax画面遷移をキャンセルできます。

- 最後に登録ボタン押下時のイベントを実装します。
  このとき、記述箇所は 内に実装することに
  気を付けてください。

```
<div data-role="page" id="main" data-theme="a">
<script>
 (function($){
   $('#main').bind("pagecreate create", function() {
     $("#storeButton").unbind().tap(function() {
       $("#storeForm").submit();
     });
   });
 })(jQuery);
</script>
```

> **注意**
> jQuery Mobileでは、Ajaxを使って画面遷移をする場合遷移先画面の 要素のみ取得し、表示中画面に挿入します。
> そのため、<HEAD>タグ内にスクリプト、およびスタイルシートを宣言すると画面遷移時に読み込まれないため、不正動作をする場合があります。

- サーバを再起動して画面を再表示します。 登録ボタン押下時、登録処理を経て画面が再表示されます。

> **コラム**
> この項目では、下記のポイントを確認しました。
> - フォームの入力内容を受け取るにはFormクラスを定義する
> - 任意のアクションに遷移するにはパスにアクション名を指定する

入力チェック処理を実装する（クライアントサイド）

store.jspにバリデーションルールを定義します。
rulesオブジェクトは入力チェックのルール、messagesオブジェクトは各ルールに対応したメッセージを定義します。

- JSP

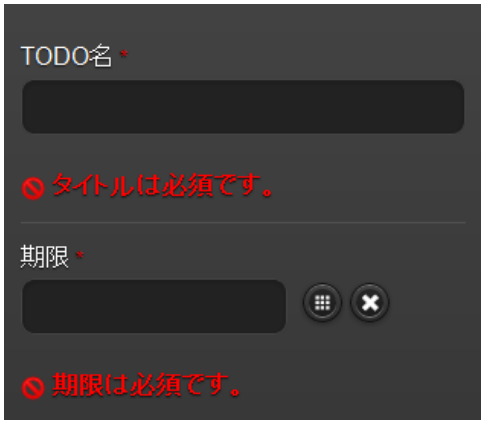
```
<script>
 var rules = {
   "title": {
     required:true,
     maxlength:20
   },
   "limit_date": {
     required:true,
     date:true
   }
 };
 var messages = {
    "title": {
      required:"タイトルは必須です。",
      maxlength:"タイトルは20文字以内で入力してください"
    },
    "limit_date": {
      required:"期限は必須です。",
      maxlength:"期限は日付形式で入力してください"
    }
 };
</script>
```

登録ボタン押下時のスクリプト処理を修正します。

- JSP

```
<script>
 (function($){
   $('#main').bind("pagecreate create", function() {
     //登録ボタン押下時のイベント
     $("#storeButton").unbind().tap(function() {
       //入力チェック実行
       if (imspValidate('#storeForm', rules, messages)) {
         //正常時
         imspAlert('入力エラーはありませんでした');
       } else {
         imspAlert('入力エラーが発生しました', 'エラー');
       }
       return false;
     });
   });
 })(jQuery);
</script>
```

- マークアップ結果。タイトルと日付未入力の状態で登録ボタンを押下すると、
  エラーダイアログが表示され警告が出力されます。



> **コラム**
> エラー仕様の詳細は、別ドキュメント クライアントサイド JavaScript の imspValidate を参照してください。

コラム

この項目では、下記のポイントを確認しました。

- クライアントサイドで入力チェックを実装するにはバリデーションルールオブジェクトを定義する

非同期で登録処理を実行する

登録ボタン押下時のスクリプト処理を修正します。

- store.jsp

```
<script>
 (function($){
  $('#main').bind("pagecreate create", function() {
    // Formのダブルクリック防止
    $('#storeForm').imspDisableOnSubmit();
    $("#storeButton").unbind().tap(function() {
      //入力チェック実行
      if (imspValidate('#storeForm', rules, messages)) {
        //Ajaxでのデータ送信
        imspAjaxSend('#storeForm', 'POST', 'json');
        //バリデーションのリセット
        imspResetForm('#storeForm');
      } else {
        imspAlert('入力エラーが発生しました', 'エラー');
      }
    });
  });
 })(jQuery);
</script>
```

非同期で返却するレスポンスオブジェクトを定義します。

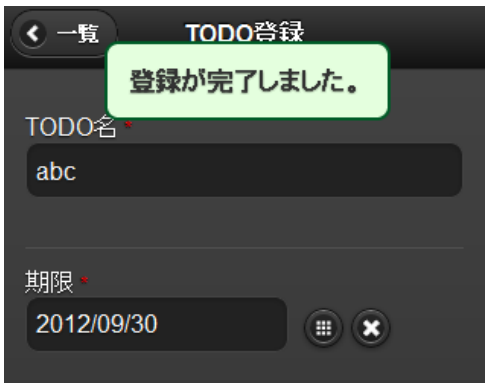
- jp.co.intra_mart.sample.spring.imsp.app.store.MyAjaxResponse

```
package jp.co.intra_mart.sample.spring.imsp.app.store;

public class MyAjaxResponse {
    private String result;
    private boolean error;
    private String errorMessage;
    private String successMessage;
    private String[] detailMessages;

    public String getResult() {
        return result;
    }
    public void setResult(String result) {
        this.result = result;
    }
    public boolean isError() {
        return error;
    }
    public void setError(boolean error) {
        this.error = error;
    }
    public String getErrorMessage() {
        return errorMessage;
    }
    public void setErrorMessage(String errorMessage) {
        this.errorMessage = errorMessage;
    }
    public String getSuccessMessage() {
        return successMessage;
    }
    public void setSuccessMessage(String successMessage) {
        this.successMessage = successMessage;
    }
    public String[] getDetailMessages() {
        return detailMessages;
    }
    public void setDetailMessages(String[] detailMessages) {
        this.detailMessages = detailMessages;
    }


}
```

- storeメソッドを以下のように修正します。
  返却値をMyAjaxResponseクラスに設定し@ResponseBodyを付加することでレスポンスをオブジェクトで返却できるようにします。

```
@RequestMapping("store")
public @ResponseBody MyAjaxResponse store(Model model, StoreForm storeForm) throws IOException {

    //ユーザコンテキストを取得します。
    UserContext userProfile = Contexts.get(UserContext.class);
    //アカウントコンテキストからタイムゾーンを取得します。
    AccountContext accountContext = Contexts.get(AccountContext.class);
    DateTimeFormatter formatter = DateTimeFormatter.withPattern("yyyy/MM/dd HH:mm");

    //登録値の設定
    MfwSample insertData = new MfwSample();
    // 略
    // storeFormからinsertDataへコピー

    insertData.setTodoId(new Identifier().get());
    insertData.setUserCd(userProfile.getUserProfile().getUserCd());
    insertData.setUserNm(userProfile.getUserProfile().getUserName());
    insertData.setTimestmp(formatter.format(DateTime.now(accountContext.getTimeZone())));

    MyAjaxResponse responseObject = new MyAjaxResponse();
    try {
      //登録処理実行
      mfwSampleService.store(insertData);

      //成功時
      responseObject.setResult("success");
      responseObject.setError(false);
      responseObject.setSuccessMessage("登録が完了しました。");
    } catch (Exception e) {
      responseObject.setError(true);
      responseObject.setErrorMessage("データ登録時にエラーが発生しました。");
      responseObject.setDetailMessages(new String[] {"管理者にお問い合わせください。"});
    }

    return responseObject;
}
```

- マークアップ結果。登録ボタン押下後にダイアログが表示され、登録処理が正常終了したことが確認できるようになりました。



> **コラム**
>
> この項目では、下記のポイントを確認しました。
>
> - クライアントから非同期でリクエストを送信するにはimspAjaxSend関数を使う

最終結果

- jp.co.intra_mart.sample.spring.imsp.app.store.StoreController.java

```
package jp.co.intra_mart.sample.spring.imsp.app.store;

import java.io.IOException;

import javax.inject.Inject;

import jp.co.intra_mart.foundation.context.Contexts;
```

```
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.i18n.datetime.DateTime;
import jp.co.intra_mart.foundation.i18n.datetime.format.DateTimeFormatter;
import jp.co.intra_mart.foundation.service.client.information.Identifier;
import jp.co.intra_mart.foundation.user_context.model.UserContext;
import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;
import jp.co.intra_mart.sample.spring.imsp.service.MfwSampleService;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping("sample/spring/sp/store")
public class StoreController {

    @Inject
    MfwSampleService mfwSampleService;

    @ModelAttribute
    public StoreForm setupStoreForm() {
        return new StoreForm();
    }

    @RequestMapping
    public String index(Model model) {
        return "sample/spring/imsp/store/index.jsp";
    }

    @RequestMapping("store")
    public @ResponseBody MyAjaxResponse store(Model model, StoreForm storeForm) throws IOException {

        //ユーザコンテキストを取得します。
        UserContext userProfile = Contexts.get(UserContext.class);
        //アカウントコンテキストからタイムゾーンを取得します。
        AccountContext accountContext = Contexts.get(AccountContext.class);
        DateTimeFormatter formatter = DateTimeFormatter.withPattern("yyyy/MM/dd HH:mm");

        //登録値の設定
        MfwSample insertData = new MfwSample();
        // 略
        // storeFormからinsertDataへコピー

        insertData.setTodoId(new Identifier().get());
        insertData.setUserCd(userProfile.getUserProfile().getUserCd());
        insertData.setUserNm(userProfile.getUserProfile().getUserName());
        insertData.setTimestmp(formatter.format(DateTime.now(accountContext.getTimeZone())));

        MyAjaxResponse responseObject = new MyAjaxResponse();
        try {
            //登録処理実行
            mfwSampleService.store(insertData);

            //成功時
            responseObject.setResult("success");
            responseObject.setError(false);
            responseObject.setSuccessMessage("登録が完了しました。");
        } catch (Exception e) {
            responseObject.setError(true);
            responseObject.setErrorMessage("データ登録時にエラーが発生しました。");
            responseObject.setDetailMessages(new String[] {"管理者にお問い合わせください。"});
        }

        return responseObject;
    }
}
```

- %CONTEXT_PATH%/WEB-INF/views/sample/spring/imsp/store/index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="imsp" uri="http://www.intra-mart.co.jp/taglib/imsp" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<imui:head>
<title>TODO登録</title>
</imui:head>
<div data-role="page" id="main" data-theme="a">
  <script>
    var rules = {
      "title": {
        required:true,
        maxlength:20
      },
      "limit_date": {
        required:true,
        date:true
      }
    };
    var messages = {
      "title": {
        required:"タイトルは必須です。",
        maxlength:"タイトルは20文字以内で入力してください"
      },
      "limit_date": {
        required:"期限は必須です。",
        maxlength:"期限は日付形式で入力してください"
      }
    };
  </script>
  <script>
    (function($){
      $('#main').bind("pagecreate create", function() {
        // Formのダブルクリック防止
        $('#storeForm').imspDisableOnSubmit();
        $("#storeButton").unbind().tap(function() {
          //入力チェック実行
          if (imspValidate('#storeForm', rules, messages)) {
            //Ajaxでのデータ送信
            imspAjaxSend('#storeForm', 'POST', 'json');
            //バリデーションのリセット
            imspResetForm('#storeForm');
          }
        });
      });
    })(jQuery);
  </script>
  <imsp:headerWithLink headerText="TODO登録" />
  <div data-role="content">
    <form name="storeForm" id="storeForm" method="POST" action="sample/spring/sp/store/store" data-ajax="false">
      <imsp:fieldContain label="TODO名" required="true">
      <input type="text" name="title" />
      </imsp:fieldContain>

      <imsp:fieldContain label="期限" required="true">
      <imsp:datePicker name="limitDate" />
      </imsp:fieldContain>

      <imsp:fieldContain label="コメント">
      <textarea name="comment"></textarea>
      </imsp:fieldContain>

      <imsp:controlGroup label="重要度">
      <imsp:radioButton name="priority" id="radio1" value="0" label="低" />
      <imsp:radioButton name="priority" id="radio2" value="1" label="中" />
      <imsp:radioButton name="priority" id="radio3" value="2" label="高" />
      </imsp:controlGroup>

      <imsp:fieldContain label="進捗">
      <imsp:slider name="progress" min="<%=0 %>" max="<%=100 %>" />
      </imsp:fieldContain>

      <imsp:fieldContain label="完了">
```

```
        <imsp:toggle name="complete" onLabel="完了" offLabel="未完了" onValue="1" offValue="0" />
        </imsp:fieldContain>

        <a data-role="button" data-theme="b" id="storeButton">登録</a>
      </form>
    </div>
    <imsp:commonFooter dataPosition="fixed" />
</div>
```

- jp.co.intra_mart.sample.spring.imsp.app.store.StoreForm.java

```
package jp.co.intra_mart.sample.spring.imsp.app.store;

public class StoreForm {
    private String title;
    private String limitDate;
    private String comment;
    private String priority;
    private String progress;
    private String complete;

    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getLimitDate() {
        return limitDate;
    }
    public void setLimitDate(String limitDate) {
        this.limitDate = limitDate;
    }
    public String getComment() {
        return comment;
    }
    public void setComment(String comment) {
        this.comment = comment;
    }
    public String getPriority() {
        return priority;
    }
    public void setPriority(String priority) {
        this.priority = priority;
    }
    public String getProgress() {
        return progress;
    }
    public void setProgress(String progress) {
        this.progress = progress;
    }
    public String getComplete() {
        return complete;
    }
    public void setComplete(String complete) {
        this.complete = complete;
    }
}
```

- jp.co.intra_mart.sample.spring.imsp.app.store.MyAjaxResponse.java

```
package jp.co.intra_mart.sample.spring.imsp.app.store;

public class MyAjaxResponse {
    private String result;
    private boolean error;
    private String errorMessage;
    private String successMessage;
    private String[] detailMessages;

    public String getResult() {
        return result;
    }
    public void setResult(String result) {
        this.result = result;
    }
    public boolean isError() {
        return error;
    }
    public void setError(boolean error) {
        this.error = error;
    }
    public String getErrorMessage() {
        return errorMessage;
    }
    public void setErrorMessage(String errorMessage) {
        this.errorMessage = errorMessage;
    }
    public String getSuccessMessage() {
        return successMessage;
    }
    public void setSuccessMessage(String successMessage) {
        this.successMessage = successMessage;
    }
    public String[] getDetailMessages() {
        return detailMessages;
    }
    public void setDetailMessages(String[] detailMessages) {
        this.detailMessages = detailMessages;
    }


}
```

- jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample.java

```
package jp.co.intra_mart.sample.spring.imsp.domain.model;

import java.io.Serializable;
import java.math.BigDecimal;

public class MfwSample implements Serializable {

    private static final long serialVersionUID = 1L;

    private String todoId;

    private String comment;

    private String complete;

    private String limitDate;

    private String priority;

    private BigDecimal progress;

    private String timestmp;

    private String title;

    private String userCd;
```

```
    private String userNm;

    public MfwSample() {
    }

    public String getComment() {
        return comment;
    }

    public String getComplete() {
        return complete;
    }

    public String getLimitDate() {
        return limitDate;
    }

    public String getPriority() {
        return priority;
    }

    public BigDecimal getProgress() {
        return progress;
    }

    public String getTimestmp() {
        return timestmp;
    }

    public String getTitle() {
        return title;
    }

    public String getTodoId() {
        return todoId;
    }

    public String getUserCd() {
        return userCd;
    }

    public String getUserNm() {
        return userNm;
    }

    public void setComment(final String comment) {
        this.comment = comment;
    }

    public void setComplete(final String complete) {
        this.complete = complete;
    }

    public void setLimitDate(final String limitDate) {
        this.limitDate = limitDate;
    }

    public void setPriority(final String priority) {
        this.priority = priority;
    }

    public void setProgress(final BigDecimal progress) {
        this.progress = progress;
    }

    public void setTimestmp(final String timestmp) {
        this.timestmp = timestmp;
    }

    public void setTitle(final String title) {
        this.title = title;
    }
```

```
    }

    public void setTodoId(final String todoId) {
        this.todoId = todoId;
    }

    public void setUserCd(final String userCd) {
        this.userCd = userCd;
    }

    public void setUserNm(final String userNm) {
        this.userNm = userNm;
    }
}
```

- jp.co.intra_mart.sample.spring.imsp.domain.repository.MfwSampleRepository.java

```
package jp.co.intra_mart.sample.spring.imsp.domain.repository;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;

public interface MfwSampleRepository {

    void create(MfwSample entity);
}
```

- jp.co.intra_mart.sample.spring.imsp.domain.service.MfwSampleService.java

```
package jp.co.intra_mart.sample.spring.imsp.service;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;

public interface MfwSampleService {
    void store(MfwSample entity);
}
```

- jp.co.intra_mart.sample.spring.imsp.domain.service.MfwSampleServiceImpl.java

```
package jp.co.intra_mart.sample.spring.imsp.service;

import javax.inject.Inject;

import org.springframework.stereotype.Service;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;
import jp.co.intra_mart.sample.spring.imsp.domain.repository.MfwSampleRepository;

@Service
public class MfwSampleServiceImpl implements MfwSampleService {

    @Inject
    MfwSampleRepository mfwSampleRepository;

    @Override
    public void store(MfwSample entity) {
        mfwSampleRepository.create(entity);
    }
}
```

実装例：一覧画面を作る

この項では、TODO登録画面で登録したTODOをスマートフォンで一覧表示する画面の実装例を紹介します。

項目

- 前提条件
- 画面の用意
  - ソースの準備と配置
  - メニューの設定
- 一覧表示部品を配置する
  - 一覧のマークアップ例
  - 検索処理の実装
- ページング処理を実装する
  - ページ情報の定義
  - ページング用タグの配置
  - ページング処理の実装
- 書式をカスタマイズする
- 最終結果

前提条件

- intra-mart Accel Platform をインストールし、初期設定までが完了していること。
- ベースモジュールに TERASOLUNA Server Framework for Java (5.x) for Accel Platform 、および、IM-Mobile Frameworkモジュールを含めて環境を作成してください。
  実行環境は単体テスト用で作成してください。
- *実装例：登録画面を作る* を参考に事前にテーブル作成、および、サンプル画面を作成しておいてください。

画面の用意

ソースの準備と配置

まず、以下２点のファイルを作成します。

- jp.co.intra_mart.sample.spring.imsp.app.list.ListController.java

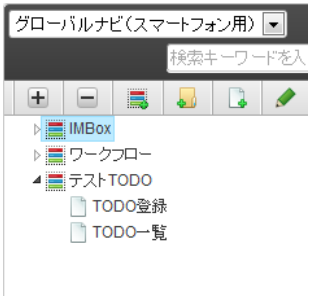
```
package jp.co.intra_mart.sample.spring.imsp.app.list;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("sample/spring/sp/list")
public class ListController {

    @RequestMapping({"", "/"})
    public String index(Model model) {
        return "sample/spring/imsp/list/index.jsp";
    }
}
```

- %CONTEXT_PATH%/WEB-INF/views/sample/spring/imsp/list/index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="imsp" uri="http://www.intra-mart.co.jp/taglib/imsp" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<imui:head>
  <title>TODO一覧</title>
</imui:head>
<div data-role="page" id="main" data-theme="a">
  <imsp:headerWithLink headerText="TODO一覧" />
  <div data-role="content">
  </div>
  <imsp:commonFooter dataPosition="fixed"/>
</div>
```

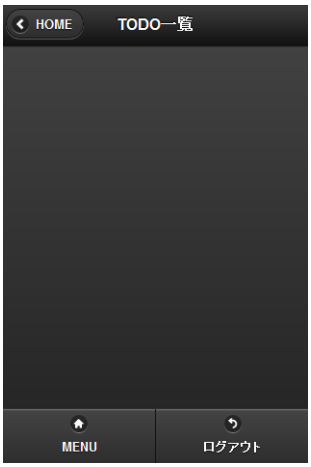
メニューの設定

登録画面と同様に一覧画面をメニューに追加します。パスは”sample/spring/sp/list”とします。



クライアントタイプをスマートフォンに切り替えグローバルナビ画面を表示します。

TODO一覧を選択します。先ほど作ったブランク画面が表示されました。



一覧表示部品を配置する

一覧表示を実現するには、<ul>タグ、および、<li>タグを使用します。

一覧のマークアップ例

例として見出し行、および、３行の一覧を表示してみます。
list.jspに以下の記述を追加します。

```
<div data-role="content">
  <ul data-role="listview">
    <li data-role="list-divider">テストの一覧</li>
    <li>一行目です。</li>
    <li>二行目です。</li>
    <li>三行目です。</li>
  </ul>
</div>
```

コラム

- リスト表示する場合、ulタグにdata-role="listview"属性を付加します。
- liタグにdata-role="list-divider"属性を付加することで強調表現になります。

- マークアップ結果。ulタグに囲まれたliタグが１行づつ表示されました。

それでは実際にTODO一覧を表示する処理を実装していきます。

検索処理の実装

前項で作成したサービスクラスに検索処理を追加します。

- jp.co.intra_mart.sample.spring.imsp.service.MfwSampleService.java

```
package jp.co.intra_mart.sample.spring.imsp.service;

import java.util.List;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;

public interface MfwSampleService {
    public void store(MfwSample entity);

    public List<MfwSample> findAll();
}
```

- jp.co.intra_mart.sample.spring.imsp.service.MfwSampleServiceImpl.java

```
package jp.co.intra_mart.sample.spring.imsp.service;

import java.util.List;

import javax.inject.Inject;

import org.springframework.stereotype.Service;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;
import jp.co.intra_mart.sample.spring.imsp.domain.repository.MfwSampleRepository;

@Service
public class MfwSampleServiceImpl implements MfwSampleService {

    @Inject
    MfwSampleRepository mfwSampleRepository;

    @Override
    public void store(MfwSample entity) {
        mfwSampleRepository.save(entity);
    }

    @Override
    public List<MfwSample> findAll() {
        return mfwSampleRepository.findAll();
    }
}
```

ListControllerにサービスクラスをバインドし、検索処理を追加します。

```
@Inject
MfwSampleService mfwSampleService;

@RequestMapping({"", "/"})
public String index(Model model) {

    List<MfwSample> result = mfwSampleService.findAll();

    model.addAttribute("list", result);

    return "sample/spring/imsp/list/index.jsp";
}
```

コラム

- データベースの詳細については データベース を参照してください。

- list.jspのulタグの内部を修正します。

```
<ul data-role="listview">
  <li data-role="list-divider">テストの一覧</li>
  <c:forEach var="record" items="${list}">
  <li><c:out value="${record.title}"/></li>
  </c:forEach>
</ul>
```

- マークアップ結果。登録されている全件のTODOのタイトルが表示されました。

コラム

この項目では、下記のポイントを確認しました。

- 一覧表示するには<ul data-role=”listview”>を使用する
- 行の表示は<li>タグを使用する

ページング処理を実装する

TODO一覧にページング処理を追加します。

ページ情報の定義

サービスクラスのfindAllメソッドを以下のように修正します。

- jp.co.intra_mart.sample.spring.imsp.service.MfwSampleService.java

```
import org.springframework.data.domain.Pageable;

...

public List<MfwSample> findAll(Pageable pageable);

...

public long count();
```

- jp.co.intra_mart.sample.spring.imsp.service.MfwSampleServiceImpl.java

```
import org.springframework.data.domain.Pageable;

...

public List<MfwSample> findAll(Pageable pageable) {
    return mfwSampleRepository.findAll(pageable).getContent();
}

...

public long count() {
    return mfwSampleRepository.count();
}
```

- ListControllerクラスを以下のように修正します。

```
    import org.springframework.data.domain.PageRequest;
    import org.springframework.data.domain.Pageable;

    ...

    @RequestMapping({"", "/"})
    public String index(Model model, Integer currentPage) {

    currentPage = (currentPage == null)? 1 : currentPage;

    //ページング情報オブジェクト
    Pageable pageable = new PageRequest(currentPage - 1, 10, Direction.ASC, "limitDate");

    List<MfwSample> result = mfwSampleService.findAll(pageable);

    model.addAttribute("currentPage", pageable.getPageNumber() + 1);
    model.addAttribute("pageLine", pageable.getPageSize());
    model.addAttribute("maxRecord", mfwSampleService.count());
    model.addAttribute("list", result);

    return "sample/spring/imsp/list/index.jsp";
}
```

ページング用タグの配置

- list.jspのulタグの内部に<imsp:pagingButton>タグを表示します。 またページ送り用のフォームも設置します。

```
<ul data-role="listview">
  <li data-role="list-divider">テストの一覧</li>
  <c:forEach var="record" items="${list}">
  <li><c:out value="${record.title}"/></li>
  </c:forEach>
  <imsp:pagingButton currentPage="${currentPage}" pageLine="${pageLine}" maxRecord="${maxRecord}" />
</ul>
<form id="pageForm" method="POST">
  <input type="hidden" name="currentPage" />
</form>
```

- 使用するタグについて

jspタグ名	機能概要	マークアップ例
pagingButton	リストのページを移動するためのボタンを提供します。	

ページング処理の実装

- 最後に、list.jspにページ遷移ボタン押下時の処理を追加します。

```
<div data-role="page">
  <script>
    function onPageLinkFunc(page) {
      $("input[name=currentPage]").val(page);
      $("#pageForm").submit();
    }
  </script>
```

> **コラム**
>
> pagingButtonタグのページ遷移ボタンは未実装関数「onPageLinkFunc」を呼び出しますので
> 必要に応じて実装してください。

- マークアップ結果。一覧下部にページ遷移ボタンが表示され、一覧が5ページづつ表示されます。

- ページ遷移ボタン押下時。2ページ目が表示されました。

> **コラム**
>
> この項目では、下記のポイントを確認しました。
>
> - ページング処理を実装するには<imsp:spPagingButton>タグを使用する

書式をカスタマイズする

より使いやすいレイアウトにするために、一覧表示の書式を修正します。

- タグの内部を以下のように修正します。

```
<ul data-role="listview">
  <li data-role="list-divider"><c:out value="${currentPage}" />ページ目</li>
  <c:forEach var="record" items="${list}">
    <li>
      <p class="ui-li-aside"><c:out value="${record.limitDate}"/></p>
      <h3><c:out value="${record.title}"/></h3>
      <p class="ui-li-desc"><c:out value="${record.comment}"/></p>
    </li>
  </c:forEach>
  <imsp:pagingButton currentPage="${currentPage}" pageLine="${pageLine}" maxRecord="${maxRecord}" />
</ul>
```

> **コラム**
>
> タグ内の<h>タグは主題として扱われます。
> <p class=”ui-li-aside”>は右端装飾部として表示されます。
> <p class=”ui-li-desc”>は副題として主題下部に表示されます。複数配置可能です。

- マークアップ結果。日付とコメントが一覧に表示されるようになりました。

最終結果

- jp.co.intra_mart.sample.spring.imsp.app.list.ListController.java

```
package jp.co.intra_mart.sample.spring.imsp.app.list;

import java.util.List;

import javax.inject.Inject;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;
import jp.co.intra_mart.sample.spring.imsp.service.MfwSampleService;

import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort.Direction;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("sample/spring/sp/list")
public class ListController {

    @Inject
    MfwSampleService mfwSampleService;

    @RequestMapping({"", "/"})
    public String index(Model model, Integer currentPage) {

        currentPage = (currentPage == null)? 1 : currentPage;

        //ページング情報オブジェクト
        Pageable pageable = new PageRequest(currentPage - 1, 10, Direction.ASC, "limitDate");

        List<MfwSample> result = mfwSampleService.findAll(pageable);

        model.addAttribute("currentPage", pageable.getPageNumber() + 1);
        model.addAttribute("pageLine", pageable.getPageSize());
        model.addAttribute("maxRecord", mfwSampleService.count());
        model.addAttribute("list", result);

        return "sample/spring/imsp/list/index.jsp";
    }
}
```

- %CONTEXT_PATH%/WEB-INF/views/sample/spring/imsp/list/index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="imsp" uri="http://www.intra-mart.co.jp/taglib/imsp" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<imui:head>
 <title>TODO一覧</title>
</imui:head>
<div data-role="page" id="main" data-theme="a">
 <script>
   function onPageLinkFunc(page) {
     $("input[name=currentPage]").val(page);
     $("#pageForm").submit();
   }
 </script>
 <imsp:headerWithLink headerText="TODO一覧" />
 <div data-role="content">
  <ul data-role="listview">
  <li data-role="list-divider"><c:out value="${currentPage}" />ページ目</li>
  <c:forEach var="record" items="${list}">
   <li>
     <p class="ui-li-aside"><c:out value="${record.limitDate}"/></p>
     <h3><c:out value="${record.title}"/></h3>
     <p class="ui-li-desc"><c:out value="${record.comment}"/></p>
   </li>
  </c:forEach>
  <imsp:pagingButton currentPage="${currentPage}" pageLine="${pageLine}" maxRecord="${maxRecord}" />
 </ul>
   <form id="pageForm" method="POST">
     <input type="hidden" name="currentPage" />
   </form>
 </div>
 <imsp:commonFooter dataPosition="fixed"/>
</div>
```

- jp.co.intra_mart.sample.spring.imsp.service.MfwSampleService.java

```
package jp.co.intra_mart.sample.spring.imsp.service;

import java.util.List;

import org.springframework.data.domain.Pageable;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;

public interface MfwSampleService {
   public void store(MfwSample entity);

   public List<MfwSample> findAll(Pageable pageable);

   public long count();
}
```

- jp.co.intra_mart.sample.spring.imsp.service.MfwSampleServiceImpl.java

```
package jp.co.intra_mart.sample.spring.imsp.service;

import java.util.List;

import javax.inject.Inject;

import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;
import jp.co.intra_mart.sample.spring.imsp.domain.repository.MfwSampleRepository;

@Service
public class MfwSampleServiceImpl implements MfwSampleService {

    @Inject
    MfwSampleRepository mfwSampleRepository;

    @Override
    public void store(MfwSample entity) {
        mfwSampleRepository.save(entity);
    }

    @Override
    public List<MfwSample> findAll(Pageable pageable) {
        return mfwSampleRepository.findAll(pageable).getContent();
    }

    @Override
    public long count() {
        return mfwSampleRepository.count();
    }

}
```

実装例：参照画面を作る

この項では、TODO登録画面で登録したTODOをスマートフォンで参照表示する画面の実装例を紹介します。

項目

- 前提条件
- 画面の用意
- 画面遷移処理を実装する
- 参照項目を表示する
- レイアウトをカスタマイズする
- 最終結果

前提条件

- intra-mart Accel Platform をインストールし、初期設定までが完了していること。
- ベースモジュールに TERASOLUNA Server Framework for Java (5.x) for Accel Platform およびIM-Mobile Frameworkモジュールを含めて環境を作成してください。
  実行環境は単体テスト用で作成してください。
- *実装例：一覧画面を作る* を参考に事前にサンプル画面を作成しておいてください。

画面の用意

まず、以下のファイルを作成します。

- jp.co.intra_mart.sample.spring.imsp.app.reference.ReferenceController.java

```
package jp.co.intra_mart.sample.spring.imsp.app.reference;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("sample/spring/sp/reference/{todoId}")
public class ReferenceController {

    @RequestMapping({"", "/"})
    public String index(Model model, @PathVariable("todoId") String todoId) {

        model.addAttribute("todoId", todoId);

        return "sample/spring/imsp/reference/index.jsp";
    }
}
```

- %CONTEXT_PATH%/WEB-INF/views/sample/spring/imsp/reference/index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="imsp" uri="http://www.intra-mart.co.jp/taglib/imsp" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<imui:head>
  <title>TODO参照</title>
</imui:head>
<div data-role="page" id="main" data-theme="a">
  <div data-role="header">
    <h3>TODO参照</h3>
    <a data-rel="back" data-icon="arrow-l">戻る</a>
  </div>
  <div data-role="content">
    <c:out value="${todoId}" />
  </div>
  <imsp:commonFooter dataPosition="fixed"/>
</div>
```

> **コラム**
> <a>タグにdata-rel=”back”属性を与えると、ボタン押下時に前画面へ戻ります。

画面遷移処理を実装する

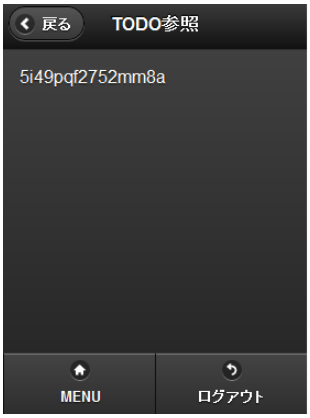
実装例：一覧画面を作る で作成した一覧画面から、参照画面へ遷移するように修正します。

- list.jspの<ul>タグにリンクを追加します。

```
<c:forEach var="record" items="${list}">
    <li>
      <a href="<c:url value="sample/spring/sp/reference/" /><c:out value="${record.todoId}" />">
        <p class="ui-li-aside"><c:out value="${record.limitDate}"/></p>
        <h3><c:out value="${record.title}"/></h3>
        <p class="ui-li-desc"><c:out value="${record.comment}"/></p>
      </a>
    </li>
</c:forEach>
```

- マークアップ結果。一覧右部に右矢印アイコンが表示されます。

- 一覧押下時。参照画面へ遷移し、選択されたTODOのIDが渡されたことが分かります。



参照項目を表示する

TODO参照画面に表示項目を追加します。

前項で作成したサービスクラスにメソッドを追加します。

- MfwSampleService.java

```
public interface MfwSampleService {

  ...

  public MfwSample findOne(String todoId);
```

- MfwSampleServiceImpl.java

```
public class MfwSampleServiceImpl implements MfwSampleService {

...

 public MfwSample findOne(String todoId) {
    return mfwSampleRepository.findOne(todoId);
 }
```

- ReferenceController.javaを以下のように修正します。

```
@Controller
@RequestMapping("sample/spring/sp/reference/{todoId}")
public class ReferenceController {

    @Inject
    MfwSampleService mfwSampleService;

    @RequestMapping({"", "/"})
    public String index(Model model, @PathVariable("todoId") String todoId) {

        MfwSample result = mfwSampleService.findOne(todoId);

        model.addAttribute("record", result);

        return "sample/spring/imsp/reference/index.jsp";
    }
}
```

コラム

リクエストマッピングに設定した変数を受け取るには
変数に@PathValiableを設定します。

- ref.jspの 以下を以下のように修正します。

```
<div data-role="content">
  <imsp:fieldContain label="登録者">
    <c:out value="${record.userCd}" />
  </imsp:fieldContain>
  <imsp:fieldContain label="登録日">
    <c:out value="${record.timestmp}" />
  </imsp:fieldContain>
  <imsp:fieldContain label="TODO名">
    <c:out value="${record.title}" />
  </imsp:fieldContain>
  <imsp:fieldContain label="期限">
    <c:out value="${record.limitDate}" />
  </imsp:fieldContain>
  <imsp:fieldContain label="コメント">
    <c:out value="${record.comment}" />
  </imsp:fieldContain>
  <imsp:fieldContain label="重要度">
    <c:if test='${record.priority.equals("0")}'>
      低
    </c:if>
    <c:if test='${record.priority.equals("1")}'>
      中
    </c:if>
    <c:if test='${record.priority.equals("2")}'>
      高
    </c:if>
  </imsp:fieldContain>
  <imsp:fieldContain label="進捗">
    <c:out value="${record.progress}" />
  </imsp:fieldContain>
  <imsp:fieldContain label="完了">
    <c:if test='${record.complete.equals("0")}'>
      未完了
    </c:if>
    <c:if test='${record.complete.equals("1")}'>
      完了
    </c:if>
  </imsp:fieldContain>
</div>
```

- マークアップ結果。各項目が表示されました。

レイアウトをカスタマイズする

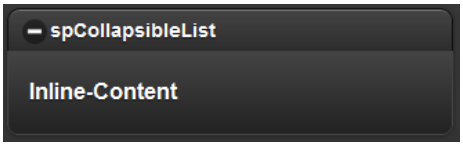
より参照画面を見やすくするために、画面をカスタマイズします。

- ref.jspの登録者・登録日項目を以下の様に修正します。

```
...
<div data-role="content">
  <imsp:collapsibleList title="登録者情報" dividerTheme="b">
    <imsp:fieldContain label="登録者">
            <c:out value="${record.userCd}" />
        </imsp:fieldContain>
        <imsp:fieldContain label="登録日">
          <c:out value="${record.timestmp}" />
        </imsp:fieldContain>
  </imsp:collapsibleList>
  <imsp:fieldContain label="TODO名">
    ...
```

- マークアップ結果。開閉リストに登録者と登録日が埋め込まれました。

- 使用するタグについて

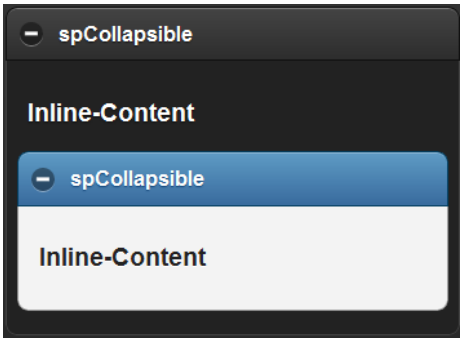
jspタグ名	機能概要	マークアップ例
spCollapsibleList	開閉可能なリストを提供します。	

- さらに、残りの項目を以下の様に修正します。

```
<div data-role="collapsible-set">
 <imsp:collapsible title="TODO内容" dataTheme="b" contentTheme="a" collapse="false">
        <imsp:fieldContain label="TODO名">
          <c:out value="${record.title}" />
        </imsp:fieldContain>
      ...
   <imart type="spFieldContain" label="重要度">
    ...
   </imart>
 </imsp:collapsible>
 <imsp:collapsible title="進捗状況" dataTheme="b" contentTheme="a">
   <imart type="spFieldContain" label="進捗">
    <imart type="string" value=$record.progress />
   </imart>
   <imart type="spFieldContain" label="完了">
    ...
   </imart>
 </imsp:collapsible>
</div>
```

- マークアップ結果。TODO登録内容が開閉ブロックに囲まれました。

- 使用するタグについて

jspタグ名	機能概要	マークアップ例
spCollapsible	開閉可能なブロックを提供します。	

> **コラム**
>
> spCollapsibleタグを タグで囲むと
> アコーディオン表示になります。

最終結果

- jp.co.intra_mart.sample.spring.imsp.app.reference.ReferenceController.java

```
package jp.co.intra_mart.sample.spring.imsp.app.reference;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;
import jp.co.intra_mart.sample.spring.imsp.service.MfwSampleService;

import javax.inject.Inject;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("sample/spring/sp/reference/{todoId}")
public class ReferenceController {

    @Inject
    MfwSampleService mfwSampleService;

    @RequestMapping({"", "/"})
    public String index(Model model, @PathVariable("todoId") String todoId) {

        MfwSample result = mfwSampleService.findOne(todoId);

        model.addAttribute("record", result);

        return "sample/spring/imsp/reference/index.jsp";
    }
}
```

- jp.co.intra_mart.sample.spring.imsp.service.MfwSampleService.java

```
package jp.co.intra_mart.sample.spring.imsp.service;

import java.util.List;

import org.springframework.data.domain.Pageable;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;

public interface MfwSampleService {
    public void store(MfwSample entity);

    public List<MfwSample> findAll(Pageable pageable);

    public MfwSample findOne(String todoId);

    public long count();
}
```

- jp.co.intra_mart.sample.spring.imsp.service.MfwSampleServiceImpl.java

```
package jp.co.intra_mart.sample.spring.imsp.service;

import java.util.List;

import javax.inject.Inject;

import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

import jp.co.intra_mart.sample.spring.imsp.domain.model.MfwSample;
import jp.co.intra_mart.sample.spring.imsp.domain.repository.MfwSampleRepository;

@Service
public class MfwSampleServiceImpl implements MfwSampleService {

    @Inject
    MfwSampleRepository mfwSampleRepository;

    @Override
    public void store(MfwSample entity) {
        mfwSampleRepository.save(entity);
    }

    @Override
    public List<MfwSample> findAll(Pageable pageable) {
        return mfwSampleRepository.findAll(pageable).getContent();
    }

    @Override
    public long count() {
        return mfwSampleRepository.count();
    }

    @Override
    public MfwSample findOne(String todoId) {
        return mfwSampleRepository.findOne(todoId);
    }
}
```

- %CONTEXT_PATH%/WEB-INF/views/sample/spring/imsp/reference/index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="imsp" uri="http://www.intra-mart.co.jp/taglib/imsp" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<imui:head>
 <title>TODO参照</title>
</imui:head>
<div data-role="page" id="main" data-theme="a">
 <div data-role="header">
  <h3>TODO参照</h3>
  <a data-rel="back" data-icon="arrow-l">戻る</a>
 </div>
 <div data-role="content">
  <imsp:collapsibleList title="登録者情報" dividerTheme="b">
   <imsp:fieldContain label="登録者">
    <c:out value="${record.userCd}" />
   </imsp:fieldContain>
   <imsp:fieldContain label="登録日">
    <c:out value="${record.timestmp}" />
   </imsp:fieldContain>
  </imsp:collapsibleList>
  <div data-role="collapsible-set">
   <imsp:collapsible title="TODO内容" dataTheme="b" contentTheme="a" collapse="false">
    <imsp:fieldContain label="TODO名">
     <c:out value="${record.title}" />
    </imsp:fieldContain>
    <imsp:fieldContain label="期限">
     <c:out value="${record.limitDate}" />
    </imsp:fieldContain>
    <imsp:fieldContain label="コメント">
     <c:out value="${record.comment}" />
    </imsp:fieldContain>
    <imsp:fieldContain label="重要度">
     <c:if test='${record.priority.equals("0")}'>
         低
     </c:if>
     <c:if test='${record.priority.equals("1")}'>
         中
     </c:if>
     <c:if test='${record.priority.equals("2")}'>
         高
     </c:if>
    </imsp:fieldContain>
    </imsp:collapsible>
    <imsp:collapsible title="進捗状況" dataTheme="b" contentTheme="a">
     <imsp:fieldContain label="進捗">
      <c:out value="${record.progress}" />
     </imsp:fieldContain>
     <imsp:fieldContain label="完了">
      <c:if test='${record.complete.equals("0")}'>
         未完了
      </c:if>
      <c:if test='${record.complete.equals("1")}'>
         完了
      </c:if>
     </imsp:fieldContain>
    </imsp:collapsible>
   </div>
  </div>
  <imsp:commonFooter dataPosition="fixed"/>
</div>
```

- %CONTEXT_PATH%/WEB-INF/views/sample/spring/imsp/list/index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="imsp" uri="http://www.intra-mart.co.jp/taglib/imsp" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<imui:head>
 <title>TODO一覧</title>
</imui:head>
<div data-role="page" id="main" data-theme="a">
 <script>
   function onPageLinkFunc(page) {
     $("input[name=currentPage]").val(page);
     $("#pageForm").submit();
   }
  </script>
 <imsp:headerWithLink headerText="TODO一覧" />
 <div data-role="content">
  <ul data-role="listview">
  <li data-role="list-divider"><c:out value="${currentPage}" />ページ目</li>
  <c:forEach var="record" items="${list}">
  <li>
    <a href="<c:url value="sample/spring/sp/reference/" /><c:out value="${record.todoId}" />">
      <p class="ui-li-aside"><c:out value="${record.limitDate}"/></p>
      <h3><c:out value="${record.title}"/></h3>
      <p class="ui-li-desc"><c:out value="${record.comment}"/></p>
    </a>
  </li>
   </c:forEach>
  <imsp:pagingButton currentPage="${currentPage}" pageLine="${pageLine}" maxRecord="${maxRecord}" />
 </ul>
   <form id="pageForm" method="POST">
     <input type="hidden" name="currentPage" />
   </form>
 </div>
 <imsp:commonFooter dataPosition="fixed"/>
</div>
```

推奨画面構成

推奨画面構成に従って頂くことで、 intra-mart Accel Applications 基盤画面部品、およびPlatform上の各種アプリケーションと互換性のとれた、統一的な画面デザインを作成できます。

項目

- スマートフォン版テーマ
- ページ
- ヘッダ
- フッタ
- ボタン
- フォーム要素
- クライアントJavaScript
- イベント

スマートフォン版テーマ

積極的に利用してください。
使用することで intra-mart Accel Applications 基盤画面部品、およびPlatform上の各種アプリケーションと同じ画面デザインを利用できます。

ページ

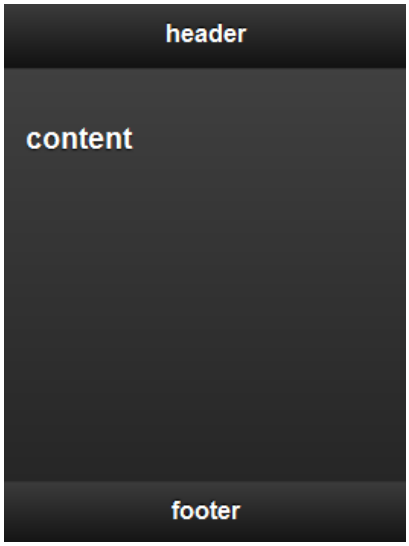
スウォッチ

明暗所の視認性を考慮し、基本的に”A”としてください。

- マークアップ例。 にdata-theme=”a”を割り当てます。

```
<div data-role="page" data-theme="a" id="a">
 <div data-role="header" data-position="fixed"> <h3>header</h3></div>
 <div data-role="content"><h3>content</h3></div>
 <div data-role="footer" data-position="fixed"> <h3>footer</h3></div>
</div>
```

- マークアップ結果。黒を基調としたデザインに調整されます。



ページ切替効果

端末やOSで差異の出にくい”fade”（デフォルト）を基本としてください。
アニメーションを伴う効果の場合、端末によって効果的にアニメーションが動作しない場合があります。

配置する要素

HOME画面へ遷移する処理をページ要素内に一つ配置してください。基本的にはフッタに配置します。

ヘッダ

通常ページの場合一律設けるようにします。
を使用するか、<imsp:headerWithLink>タグを使用してください。
固定ポジションモード（data-position=”fixed”）は積極的に使用してください。

左ボタン

任意で配置します。

- アプリケーションや各機能の最上位階層の場合、戻るボタンは不要です。
- 上記以外の場合、基本的に戻るボタンを配置します。
- 戻るボタンは、アイコン＋文字列とします。基本的にdata-icon=”arrow-l”を指定してください。

右ボタン

任意で配置します。

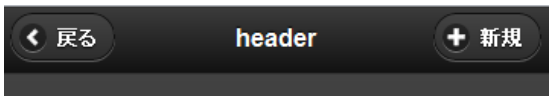
- 基本的に、入力操作系ボタン（新規ボタン、編集ボタン、投稿ボタン）やトランザクション処理ボタン（登録ボタン、更新ボタン）を配置します。
- 入力操作系ボタンは、文字列とします。

マークアップ例

- JSP

```
<div data-role="header" data-position="fixed">
 <h3>header</h3>
 <a data-role="button" data-rel="back" data-icon="arrow-l">戻る</a>
 <a data-role="button" href="hoge/hoge" data-icon="plus">新規</a>
</div>
```

- マークアップ結果



フッタ

通常ページの場合一律設けるようにします。
以下に示すナビゲーションバーを利用するかまたは <imsp:commonFooter>を利用してください。
ナビゲーションバーの利用方法については実装例を参考に実装してください。

ナビゲーションバーを用いたフッタの実装（推奨）

NO	ラベル名	役割	備考
1	なし	HOMEリンク	アカウントコンテキストから取得したホームURLを設定します。
2～4	個別	画面個別機能	各画面で個別に設定します。
5	なし	その他	個別機能が4つ以上の場合に配置します。

フッタ要素

フッタは jQueryMobile の標準のフッタとして定義します。また data-position=”fixed” と指定することで固定フッタとします。
なお、フッタには必ず class=”imui-smart-footer” を指定してください。
フッタの中に navbar を定義し、処理リンクなどを配置します。
一度に表示するリンクは最大5つまでとします。
コピーライトは各画面のフッタから必ず呼び出せるようにしてください。

HOMEリンク

フッタの最左部に配置します。 以下属性を設定します。

- data-ajax=”false”
- data-icon=”custom”
- class=”im-smart-icon-common-32-home-navbar-navbar”
- data-iconpos=”top”

処理リンク

各画面で個別に設定します。 HOMEリンク含め総リンク数が５つを超える場合は その他リンクで補完します。

ボタンの属性には以下を指定します。

- data-icon=”custom”
- class=”アイコンのクラス名 + -navbar”
- data-iconpos=”top”
- リンクにはテキストを指定してください。

使用可能なアイコンは CSS Sprite Image List のスマートフォン向け を参照してください。

> **コラム**
>
> 推奨しているアイコン画像のサイズは32pxです。

その他リンク

HOMEリンク含め総リンク数が５つを超える場合に配置します。 ボタン押下時はポップアップ表示でその他の機能を表示します。

ボタンの属性には以下を指定します。

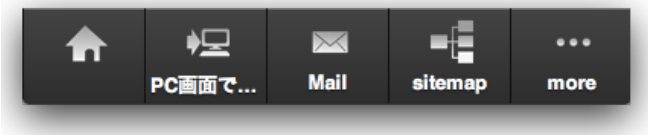
- data-icon=”custom”
- class=”im-smart-icon-common-32-more-navbar”
- data-iconpos=”top”

マークアップ例

- JSP

```
<footer data-role="footer" class="imui-smart-footer" data-position="fixed">
 <div data-role="navbar">
  <ul>
   <li><a href="home" data-icon="custom" class="im-smart-icon-common-32-home-navbar" data-iconpos="top" data-ajax="false"></a></li>
   <li><a href="index.html" data-icon="custom" class="im-smart-icon-common-32-desktop-site-navbar" data-iconpos="top">PC画面で表示する</a></li>
   <li><a href="index.html" data-icon="custom" class="im-smart-icon-common-32-mail-navbar" data-iconpos="top">Mail</a></li>
   <li><a href="index.html" data-icon="custom" class="im-smart-icon-common-32-sitemap-navbar" data-iconpos="top">sitemap</a></li>
   <li><a href="index.html" data-icon="custom" class="im-smart-icon-common-32-more-navbar" data-iconpos="top">more</a></li>
  </ul>
 </div>
</footer>
```

- マークアップ結果



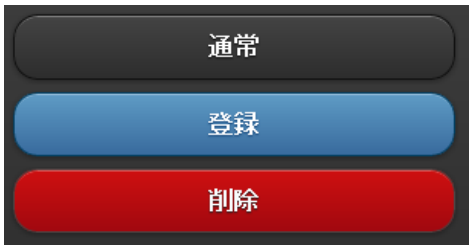
ボタン

スウォッチ

登録・更新などトランザクション処理など、画面の主目的となるボタンは強調するため”B”としてください。
ただし、削除系処理の場合は intra-mart Accel Platform 拡張スウォッチの”F”を指定してください。

その他の場合は特に制限しませんが、特段理由がなければ指定しない方が画面全体の統一感がとれてよいでしょう。

- 各スウォッチ別マークアップ例



ボタンアイコン

ボタンの用途に合うものがあれば積極的に利用してください。

フォーム要素

data-role=”none”について

要素をブラウザのネイティブ表示にする属性指定です。 特段理由がない限り使用しないでください。

配置

基本的に 1 行 1 要素としてください。

ラベル

<imsp:fieldContain>タグまたは 、<imsp:controlGroup>タグまたは タグを使用してください。

必須項目の強調

<imsp:fieldContain>または<imsp:controlGroup>タグの場合、required="true"を指定します。 その他の場合はスタイルのクラスに"imui-smart-ui-required:after"を指定してください。

- マークアップ例 画面上には、「タイトル＊」と表示されます。

```
<label class="imui-smart-ui-required:after">タイトル</label>
```

クライアントJavaScript

スクリプトの配置

画面固有動作の場合、 タグ内に配置してください。 jQuery Mobileでは、Ajaxを使用した画面遷移を利用すると遷移先画面のページ要素のみを取得しますので、従来通り<HEAD>タグ内に配置すると不正動作を起こす可能性があります。

$(document).ready()について

使用をお勧めしません。 Ajaxを使用した画面遷移を利用する場合に遷移先画面のreadyイベントがコールされない場合があります。
$(document).bind("pageinit", function() {});を使用してください。

window.alert() について

imspAlertを使用してください。

- マークアップ例

```
<script>
 $(document).bind("pageinit",function() {
  //ボタンのタップイベントをバインド
  $("#someButton").bind("tap", function() {
   imspAlert('aaa', '警告', function() {alert("ok");});
  });
 });
</script>
...
<a data-role="button" id="someButton">ボタン</a>
```

- マークアップ結果。
ボタンをクリックすると以下警告ダイアログが表示され、「決定」をクリックするとコールバック関数が呼び出されます。

window.confirm()について

上記同様、imspConfirmを使用してください。

イベント

クリックイベント

clickイベントはタッチデバイスを考慮していません。 タッチデバイスが考慮されているtap、またはvclickイベントを使用してください。

イベントのバインド

以下のようにタグに直接イベントハンドラを記述することはお勧めしません。

```
<input type="button" name="someButton" onclick="someFunction()" />
```

ページ初期化イベント時等でjQueryのイベントバインド関数を使用して各要素にイベントをバインドしてください。

```
//ページ初期化処理。要素にイベントをバインドします
$(document).bind("pageinit",function() {
 //ボタンのタップイベントをバインド
 $("input[name=someButton]").bind("tap", function() {
   ...
 });
 ...
});
```

エラー処理

項目

- クライアントサイドのエラー処理
- TERASOLUNA Server Framework for Java (5.x) for Accel Platform での汎用メッセージ画面
  - View名
    - 実装例

クライアントサイドのエラー処理

クライアントサイドのエラー処理は UIデザインガイドライン（ PC 版） のエラー処理 を参照してください。

TERASOLUNA Server Framework for Java (5.x) for Accel Platform での汎用メッセージ画面

アプリケーション内で発生したエラーによっては、回復不能な場合があります。
このような場合に表示する画面の view を提供します。

View名

表示できるメッセージ表示画面の種類、その画面に遷移するための view名を定義している定数を以下の通りです。

- エラー
  - jp.co.intra_mart.framework.extension.spring.web.servlet.view.TransferView.VIEW_NAME_ERROR
- 警告
  - jp.co.intra_mart.framework.extension.spring.web.servlet.view.TransferView.VIEW_NAME_WARN
- インフォメーション
  - jp.co.intra_mart.framework.extension.spring.web.servlet.view.TransferView.VIEW_NAME_INFO

メッセージ表示画面に表示できるのは、

- タイトル
- メッセージ
- 詳細なエラーメッセージ

の３つです。

また、メッセージ表示画面表示後の遷移先を指定することも可能です。
遷移先 URL、遷移先 URL のラベルを指定すると、画面遷移のボタンが表示されます。
遷移先 URL へ引き渡すパラメータを指定すると、そのパラメータが遷移先に引き渡されます。

メッセージ表示画面へは jp.co.intra_mart.foundation.ui.page.Transfer を利用しています。 view からTransferを呼び出してメッセージ画面へリダイレクトします。

実装例

エラー画面へ遷移するための例を示します。

TransferErrorView に指定するパラメータの詳細は、API リファレンスを参照してください。

ここでは、 *基本（ intra-mart Accel Platform での初めてのプログラミング）* のステップ 7 で作成したクラスを修正する形で紹介します。
テキストボックスに “error” と入力された場合、エラーページへ遷移します。

```
import jp.co.intra_mart.foundation.ui.page.Transfer.Message;
import jp.co.intra_mart.framework.extension.spring.web.servlet.view.TransferView;

// (中略)

@RequestMapping
public String index(@ModelAttribute HelloForm helloForm) {
    // 入力した項目を再表示するため、@ModelAttribute HelloFormを引数に設定します。
    return "sample/tgfw/hello/index.jsp";
}

/**
 * 結果表示画面のパスを返却します。
 * @return 結果表示画面のパス
 */
@RequestMapping("output")
public String output(@ModelAttribute HelloForm helloForm, Model model) {
    // メッセージ画面のviewへ情報を渡すために、引数にModelを設定します。

    if ("error".equals(helloForm.getName())) {
        final Message message = new Message();
        message.setTitle("入力エラー");
        message.setMessage("入力エラーが発生しました。再入力してください。");
        message.setDetails(new String[] { "詳細メッセージ01", "詳細メッセージ02" });
        message.setReturnUrl("sample/tgfw/hello");        // index画面へのURL
        message.setReturnUrlLabel("入力画面へ戻る");

        final Map<String, Object> parameters = new HashMap<String, Object>();
        parameters.put("name", "re-enter!");              // name に値を設定する。
        message.setParameters(parameters);

        model.addAttribute(TransferView.MESSAGE, message);  // modelのTransferView.MESSAGE属性にMessageオブジェクトをセットする。
        return TransferView.VIEW_NAME_ERROR;              // エラー画面のview名TransferView.VIEW_NAME_ERROR を返す。
    }

    return "sample/tgfw/hello/output.jsp";
}
```

コラム

intra-mart Accel Platform では、メッセージ表示画面へ遷移するメソッドTransfer が提供されています。
Transferメソッドを実行すると内部でリダイレクト処理が実行されますが、Spring MVC の Controllerの処理メソッド内でこのTransferのメソッドを利用すると、Transferメソッドのリダイレクト処理の後に、さらに処理メソッドの返却されるView名に応じてフォワード処理が実行されてしまいます。
そのため、エラー画面/警告画面/インフォメーション画面に遷移できるように、下記の通り View 名を提供しています。

- エラー画面：TransferView.VIEW_NAME_ERROR
- 警告画面：TransferView.VIEW_NAME_WARN
- インフォメーション画面：TransferView.VIEW_NAME_INFO

コラム

上記パラメータのメッセージは多言語対応していません。多言語対応したい場合は、MessageManager から取得したメッセージをセットしてください。

コラム

テーマを適用させたくない場合には、API を呼び出す前に HttpServletRequest に対し、setAttribute(‘imui-theme-builder-module’, ‘notheme’) を実行し、テーマを適用しないようにしてください。

Storage

項目
- Storageとは
- Storageの種類
- ストリーミング
- プログラミング方法
  - ファイルアップロード
  - ファイルダウンロード

Storageとは

Storageは分散システムで intra-mart Accel Platform を利用しているときに、アップロードされたファイルやシステムで
共有したいファイル（主にデータファイル）を一元管理します。

> **コラム**
>
> 分散システムを構築する場合、各 Web Application Server で共有するディレクトリを設定しておく必要があります。
> 詳細はセットアップガイドを参照してください。

Storageの種類

- SystemStorage

  システムで利用するファイルを保存する領域です。
  主に、 intra-mart Accel Platform の基盤APIやアプリケーション内の処理で利用されます。

- PublicStorage

  アップロードされたファイルや利用者間で共有したいファイルを保存する領域です。
  Storageにファイルを保存する場合は、基本的にPublicStorageに保存します。

- SessionScopeStorage

  一時的にファイルを保存する領域です。
  処理の途中でアップロードされたファイルや保存されたデータを一時的に保存したい場合に利用します。
  SessionScopeStorageに保存されたファイルは、セッションの有効期間が切れたタイミングで自動的に削除されます。

ストリーミング

intra-mart Accel Platform ではStorageの保存されているデータをストリームで扱うことができるようになりました。

ストリームを利用することで、従来のintra-mart WebPlatformのように一度 Web Application Server のメモリ上に
ファイルデータを持つ必要がなくなり、サイズの大きいファイルのアップロードやダウンロードが行えます。

> **コラム**
>
> Storage APIのload(),read(),save(),write()メソッドを使用するとファイルデータがAPサーバのメモリ上に展開されるため、メモリを圧迫します。
> その為、これらのメソッドの利用は推奨しません。

プログラミング方法

ファイルアップロード

ここでは、PublicStorage内の”sample”ディレクトリにファイルをアップロードする例を示します。

- Controller クラス

```
import java.io.IOException;
import java.io.OutputStream;

import jp.co.intra_mart.common.aid.jdk.java.io.IOUtil;
import jp.co.intra_mart.foundation.service.client.file.PublicStorage;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.multipart.MultipartFile;
import org.terasoluna.gfw.common.exception.SystemException;

@Controller
@RequestMapping("upload")
public class UploadController {

    @RequestMapping
    public String index() {
        return "index.jsp";
    }

    @RequestMapping(value = "store", method = RequestMethod.POST)
    public String upload(final MultipartFile uploadFile) {

        if (!uploadFile.isEmpty()) {
            // ファイル保存先のpublic storageインスタンスを生成する。
            final PublicStorage storage = new PublicStorage("sample", "samplefile");
            OutputStream os = null;
            try {
                os = storage.create();
                IOUtil.transfer(uploadFile.getInputStream(), os);
            } catch (final IOException e) {
                throw new SystemException("io error code", e);
            } finally {
                try {
                    os.close();
                } catch (final IOException e) {
                    throw new SystemException("io error code", e);
                }
            }
            return "redirect:/upload";
        }
        return "redirect:/upload";
    }
}
```

ファイルダウンロード

ここでは、PublicStorage内のファイル”sample.txt”をダウンロードする例を示します。
AbstractFileDownloadViewの詳細については TERASOLUNA Server Framework for Java (5.x) Development Guideline の任意のファイルのダウンロードの項 を参照してください。

- Controller クラス

```
import static org.springframework.web.bind.annotation.RequestMethod.GET;

import java.io.FileNotFoundException;
import java.io.IOException;

import jp.co.intra_mart.foundation.service.client.file.PublicStorage;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.terasoluna.gfw.common.exception.SystemException;
import org.terasoluna.gfw.common.message.ResultMessage;
import org.terasoluna.gfw.common.message.ResultMessages;

@Controller
@RequestMapping("sample/tgfw/download")
public class DownloadController {

    @RequestMapping(value = "fetch", method = GET)
    public String download(final Model model) {

        // sample.txtを指定したPublicStorageのインスタンスを生成します。
        final PublicStorage storage = new PublicStorage("sample.txt");
        try {
            if (!storage.isFile()) {
                // ファイルが存在しない場合、エラーとしてindex.jspへ戻す。
                final ResultMessages rs = ResultMessages.warning().add(ResultMessage.fromText("file not found."));
                model.addAttribute(rs);
                return "sample/tgfw/download/index.jsp"; // index.jspへ戻す。
            }
            model.addAttribute("storage", storage);
            // download用のviewを指定する。
            return "sample.tgfw.app.download.DownloadView";
        } catch (final IOException e) {
            throw new SystemException("io error code", e);
        }
    }

    @RequestMapping
    public String index() {
        return "sample/tgfw/download/index.jsp";
    }
}
```

- View クラス

```
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import jp.co.intra_mart.foundation.http.ResponseUtil;
import jp.co.intra_mart.foundation.service.client.file.Storage;

import org.springframework.stereotype.Component;
import org.terasoluna.gfw.common.exception.SystemException;
import org.terasoluna.gfw.web.download.AbstractFileDownloadView;

// view name = sample.tgfw.app.download.DownloadView として登録する。
@Component("sample.tgfw.app.download.DownloadView")
public class DownloadView extends AbstractFileDownloadView {

    @Override
    protected void addResponseHeader(final Map<String, Object> model, final HttpServletRequest request, final
HttpServletResponse response) {
        // textファイル用のヘッダを設定する。
        final String disposition;
        try {
            // サーバのcharacter encodingを第二引数に設定してください。
            disposition = ResponseUtil.encodeFileName(request, "UTF-8", "sample.txt");
        } catch (final UnsupportedEncodingException e) {
            throw new SystemException("download view error code", e);
        }
        response.setHeader("Content-Disposition", "attachment;" + disposition);
        response.setContentType("text/plain");
    }

    @Override
    protected InputStream getInputStream(final Map<String, Object> model, final HttpServletRequest request) throws
IOException {
        // modelに設定した storage を取得し、InputStreamを返す。
        final Storage<?> storage = (Storage<?>) model.get("storage");
        return storage.open();
    }
}
```

非同期処理

項目

- 非同期処理とは
- 仕様
- プログラミング方法

非同期処理とは

非同期処理はビジネスロジックを呼び出して処理を行うための一つの方法です。
ビジネスロジックの実行結果の取得が重要ではない場合、非同期処理機能を利用することによって、全体的な応答を早めることが可能です。

以下のような条件を満たす処理を行う場合、処理の呼び出し側とは別のスレッドで処理を行うとユーザインタフェースの応答を早くできる場合があります。

- 処理時間そのものは比較的短いが、ユーザインタフェースの観点からすると応答時間が長い。（数秒～数十秒程）
- 処理の実行はリアルタイムである必要はないが、処理要求後にできるだけ早く実行したい。
- 処理の呼び出し側では、処理の完了については特に気にする必要はない。

仕様

非同期処理の仕様については、「非同期仕様書」を参照してください。

プログラミング方法

非同期処理のプログラミング方法については、「非同期処理プログラミングガイド」を参照してください。

ジョブスケジューラ

項目

- ジョブスケジューラとは
- 仕様
- サンプルプログラム
  - サンプル内容
  - プログラムソース
- ジョブの実行方法
  - ジョブ・ジョブネットを登録する
  - ジョブを実行する

ジョブスケジューラとは

ジョブスケジューラとは、ジョブと呼ばれる処理単位に事前にいつ実行するかというスケジュールを定義しておくことで
自動的に実行する機能です。 1 つの業務を定期的に複数回実行する場合や、大量データを扱うため時間かかる業務処理を
夜間に実行する場合などに利用します。

intra-mart Accel Platform のジョブスケジューラは、このような要求を実現するためサーバ上の Java や
サーバサイドJavaScript で構成された任意の業務処理をスケジュールで定義されたタイミングで
自動的に実行する機能や、実行状況の監視や実行結果の管理を行うための機能を提供します。

仕様

ジョブスケジューラの仕様については、「ジョブスケジューラ仕様書」を参照してください。

サンプルプログラム

サンプル内容

固定文字列”Hello.”に、”message”というキーに設定されたパラメータ値を連結して出力します。
Spring Framework の Application Context から GreetingService の bean を取得し、メッセージを出力します。

プログラムソース

```
package sample.tgfw.job.hello;

import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.job_scheduler.Job;
import jp.co.intra_mart.foundation.job_scheduler.JobResult;
import jp.co.intra_mart.foundation.job_scheduler.JobSchedulerContext;
import jp.co.intra_mart.foundation.job_scheduler.annotation.Parameter;
import jp.co.intra_mart.foundation.job_scheduler.annotation.Parameters;
import jp.co.intra_mart.foundation.job_scheduler.exception.JobExecuteException;
import jp.co.intra_mart.framework.extension.spring.context.ApplicationContextProvider;
import sample.tgfw.domain.greeting.GreetingService;

public class HelloJob implements Job {

  public HelloJob() {
  }

  @Override
  @Parameters(@Parameter(key = "message", value = "world!"))
  public JobResult execute() throws JobExecuteException {
    try {
      // アカウントコンテキスト
      final AccountContext accountContext = Contexts.get(AccountContext.class);
      System.out.println("Account context : " + accountContext.toString());

      // ジョブスケジューラコンテキスト
      final JobSchedulerContext jobSchedulerContext = Contexts.get(JobSchedulerContext.class);
      System.out.println("Job scheduler context : " + jobSchedulerContext.toString());

      // パラメータの取得
      final String message = jobSchedulerContext.getParameter("message");
      if (null == message) {
        // 処理結果：異常
        return JobResult.error("パラメータにメッセージが存在しません。");
      } else if (message.trim().isEmpty()) {
        // 処理結果：警告
        return JobResult.waring("メッセージが空です。");
      }
      // メッセージの表示
      System.out.println("Hello. " + message);

      // application contextから service beanを取得
      final GreetingService greetingService = ApplicationContextProvider.getApplicationContext().getBean(GreetingService.class);
      System.out.println(greetingService.hello());

      // 処理結果：正常
      return JobResult.success("ジョブが正常に実行されました。");
    } catch (final Exception e) {
      // 処理結果：異常（例外による処理結果の返却）
      throw new JobExecuteException("予期しないエラーが発生しました。", e);
    }
  }
}
```

sample.tgfw.domain.greeting.GreetingService の bean を Spring Framework の Application Context に登録するために、 モジュールプロジェクトの applicationContext*.xml に以下を追記しておきます。

```
<context:component-scan base-package="sample.tgfw.domain" />
```

- Jobの実装クラスの作成
  Java のジョブプログラムは、jp.co.intra_mart.foundation.job_scheduler.Jobの実装クラスを作成します。
  このインタフェースには、ジョブの実行処理を記述するためのexecuteメソッドが定義されています。
  ジョブ開発者は、このexecuteメソッドにジョブ処理で実行したいプログラムを記述します。
- Spring Framework の Application Context の取得
  Spring Framework の Application Context を取得する場合には、
  jp.co.intra_mart.framework.extension.spring.context.ApplicationContextProvider.getApplicationContext() を使用してください。
  この例では、 Spring Framework の Application Context に登録されている GreetingService を取得しています。

> **コラム**
>
> ジョブ処理では、アカウントコンテキストとジョブスケジューラコンテキストが取得可能です。
> この2つのコンテキストを利用して任意の業務処理を記述します。
>
> **アカウントコンテキスト**
>
> アカウントコンテキストには、ジョブスケジューラから実行されたことを表すアカウント情報が格納されています。
>
> **ジョブスケジューラコンテキスト**
>
> ジョブスケジューラコンテキストには、ジョブ、ジョブネット、トリガの定義情報と実行日時などの
> 実行情報が格納されています。

ジョブの実行方法

> **コラム**
>
> ジョブ管理画面からジョブを実行する方法に関しては、「テナント管理者操作ガイド」 -「ジョブを設定する」 を参照してください。

ジョブ・ジョブネットを登録する

ここでは、ジョブを登録する例を示します。

```
// ジョブビルダの作成
JobBuilder jobBuilder = new JobBuilder("sample-job");
jobBuilder = jobBuilder.withLocalize("Sample Job", "This is a sample job.");// ローカライズ情報の設定
jobBuilder = jobBuilder.ofJobClass(HelloJob.class); // ジョブクラス名の登録
final JobDetail job = jobBuilder.build(); // ジョブの生成

// ジョブスケジューラマネージャの作成
final JobSchedulerManager manager = JobSchedulerManagerFactory.getJobSchedulerManager();

// ジョブを登録する
manager.insertJob(job);
```

ここでは、ジョブネットを登録する例を示します。

```
// ジョブネットビルダの作成
JobnetBuilder jobnetBuilder = new JobnetBuilder("sample-jobnet");
jobnetBuilder = jobnetBuilder.withLocalize("Sample Jobnet", "This is a sample jobnet."); // ローカライズ情報の設定
jobnetBuilder = jobnetBuilder.ofJobIdList("sample-job"); // 直列実行するジョブIDの設定
jobnetBuilder = jobnetBuilder.allowConcurrent(); // ジョブネットの同時実行の許可
jobnetBuilder = jobnetBuilder.usingParameters("message", "world!"); // パラメータの設定
final Jobnet jobnet = jobnetBuilder.build(); // ジョブネットの生成

// ジョブスケジューラマネージャの作成
final JobSchedulerManager manager = JobSchedulerManagerFactory.getJobSchedulerManager();

// ジョブネットの登録
manager.insertJobnet(jobnet);
```

ジョブを実行する

ここでは、指定したジョブネットを即時実行する例を示します。

```
final String triggerId = "sample-trigger";
try {
    final JobSchedulerManager manager = JobSchedulerManagerFactory.getJobSchedulerManager();
    try {
        final Trigger trigger = manager.findTrigger(triggerId);
        // 登録済みのトリガーを再登録して実行
        manager.updateTrigger(trigger);
    } catch (DataNotFoundException e) {
        // トリガーが存在しない場合は新規登録
        final Trigger trigger = new TriggerBuilder(triggerId).forJobnetId("sample-jobnet").onceSchedule().build();
        manager.insertTrigger(trigger);
    }
} catch (JobSchedulerException e) {
    // 例外処理
}
```

intra-mart Accel Platform 2014 Winter以降のジョブの即時実行方法

以下の方法でジョブの即時実行をスケジュールできます。

```
// ジョブスケジューラマネージャの作成
final JobSchedulerManager manager = JobSchedulerManagerFactory.getJobSchedulerManager();

// 指定したジョブネットの即時実行
manager.execute("sample-jobnet");
```

Lockサービス

項目

- Lockとは
- サンプルプログラム
- リクエストに紐付けたロック

Lockとは

Lockサービスはintra-martシステム全体で一意のロックを行う機能です。
特定の機能を使用不可能にしたい場合や処理の直列化を行いたい場合等に利用します。

> **コラム**
>
> ロックの利用状況は[システム管理者] - [アプリケーションロック一覧]画面より確認できます。
> 詳細は、「システム管理者操作ガイド」を参照してください。

サンプルプログラム

ロックを開始した後、処理ロジックを実行して、最後にロックを開放する場合は以下のように実装します。
（このサンプルでは５秒間ロックが開始できなかった場合は例外をスローします。）

```
// ロックの開始
NewLock lock = new NewLock("lock_key");
if(!lock.tryLock(5, TimeUnit.SECONDS)) {
    // ロックの開始に失敗
    throw new Exception();
}
try {
    // 処理ロジック

} finally {
    // ロックの開放処理
    lock.unlock();
}
```

リクエストに紐付けたロック

リクエストに紐付けたロックを開始する場合は、以下のメソッドを利用してください。

```
NewLock lock = new NewLock("lock_key");
boolean result = lock.tryLockRequestScope(5, TimeUnit.SECONDS);
```

この関数を利用して開始されたロックは、レスポンスを返却する際に自動的に開放されます。
また、unlock()関数を使用して任意のタイミングで開放することもできます。
ロックの解放漏れを防ぎたい場合などは、この関数を利用してロックを開始してください。

> **注意**
>
> この関数はリクエストにロックを紐付けて自動開放を行うものなので、非同期タスクや
> ジョブスケジューラで実行されるジョブの処理内で利用することはできません。

Cacheサービス

項目

- Cacheとは
- 仕様

Cacheとは

Cacheはアプリケーションサーバ上のメモリを利用して、オブジェクトの保存を行うことが可能な機能です。
データベースアクセスや、ファイルアクセス等の取得結果をキャッシュすることによりアプリケーションのパフォーマンス向上を図ることが可能です。

仕様

標準では、Cache実装としてEHCacheが利用されます。
EHCacheに関しては、 http://ehcache.org を参照してください。

Cacheに登録したオブジェクトは、設定ファイルに指定した要素数、またはサイズの上限を超えた場合に破棄されます。
また、有効期間を過ぎたオブジェクトも破棄対象です。
Cacheを利用する場合、そのCacheに対する設定は%CONTEXT_PATH%/WEB-INF/conf/im-ehcache-config/フォルダ配下に任意の名前のxmlファイルを配置する必要があります。
以下に設定ファイル例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<im-ehcache-config xmlns="http://www.intra-mart.jp/cache/ehcache/config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.intra-mart.jp/cache/ehcache/config im-ehcache-config.xsd ">

  <cache name="myCache"
      enable="true"
      max-bytes-memory="10m"
      max-elements-on-memory="100"
      overflow-to-disk="true"
      max-bytes-disk="50m"
      max-elements-on-disk="500"
      time-to-idle-seconds="600"
      time-to-live-seconds="3600" />

</im-ehcache-config>
```

> **注意**
>
> 文字コードを UTF-8 にして保存してください。

各設定に関する詳細は以下の通りです。

属性名	説明
name	Cache名を設定します。
enable	trueまたはfalseを指定します。 falseが指定された場合は該当のCacheは無効です。
max-bytes-memory	メモリ上にオブジェクトを格納する際の最大サイズを指定します。 1k, 10M, 50G等の表記が可能です。
max-elements-on-memory	メモリ上にキャッシュするオブジェクトの最大数を指定します。
overflow-to-disk	メモリ上にキャッシュするの領域の上限を超えた場合にディスクに書き出すか設定します。
max-bytes-disk	ディスク上にオブジェクトを格納する際の最大サイズを指定します。 1k, 10M, 50G等の表記が可能です。
max-elements-on-disk	ディスク上にキャッシュするオブジェクトの最大数を指定します。
time-to-idle-seconds	アイドル時間（秒）を指定します。指定された時間対象となるオブジェクトが参照されなかった場合、そのオブジェクトは破棄されます。
time-to-live-seconds	生存期間（秒）を指定します。指定された生存期間を超えた場合そのオブジェクトは破棄されます。

コラム

「max-bytes-memory」及び、「max-bytes-disk」属性が設定されている場合、Cacheにオブジェクトを登録する際に、そのオブジェクトのサイズの計算処理が行われます。
この際、登録するオブジェクトが、別のオブジェクトの参照を大量に持つ場合、計算処理に時間がかかりパフォーマンスの低下の原因となる可能性があります。
登録するオブジェクトが1000以上の参照を持つ場合、下記のようなメッセージがログに出力されます。

```
The configured limit of 1,000 object references was reached while attempting to calculate the size of the object graph.
Severe performance degradation could occur if the sizing operation continues.
This can be avoided by setting the CacheManger or Cache <sizeOfPolicy> elements maxDepthExceededBehavior to
"abort" or adding stop points with @IgnoreSizeOf annotations.
If performance degradation is NOT an issue at the configured limit, raise the limit value using the CacheManager or
Cache <sizeOfPolicy> elements maxDepth attribute.
For more information, see the Ehcache configuration documentation.
```

このログが出力される場合は、キャッシュに格納するオブジェクトの構成を見直すか、「max-bytes-memory」または、「max-bytes-disk」の代わりに、「max-elements-on-memory」または「max-elements-on-disk」の利用を検討して下さい。

ショートカットアクセス機能

項目

- ショートカットアクセス機能とは
- ショートカットアクセス URL
- ショートカット ID の作成
- ショートカット拡張検証機能
  - 検証プログラムの作成
  - 検証コードと検証プログラムの設定
  - 標準の検証コードについて

ショートカットアクセス機能とは

ショートカットアクセス機能は、初期アクセス URL にショートカットアクセス用の URL を指定することによって、ログイン後の画面を任意の画面に取り替えることができる機能です。
ショートカットアクセス機能を用いると、ログイン後の画面で任意のページを表示することが可能になります。

ショートカットアクセス URL

ショートカットアクセス用の URL は、ショートカットIDを含む以下の URL になります。

```
http:// <server> / <context-path> /user/shortcut/ <ショートカットID>
```

<記述例>

```
http://localhost/imart/user/shortcut/5i4deh98wou5uuc
```

ショートカット ID の作成

ショートカット ID は、表示するページの情報およびセキュリティの情報に紐づく ID となります。
ショートカット IDは、表示するページの情報およびセキュリティの情報を指定してAPI を用いて作成します。
ログイン後に表示する画面に、/sample/shortcut を指定する場合のショートカット IDの作成手順を説明します。

```
// ショートカットマネージャの作成
ShortCutManager manager = new ShortCutManager();

// ショートカット情報の作成
ShortCutInfo shortCutInfo = new ShortCutInfo();

// 表示する URL
shortCutInfo.setUrl("/sample/shortcut");

// 表示する URL に渡すパラメータの設定(任意指定)
shortCutInfo.setUrlParam("arg1","value1");
shortCutInfo.setUrlParam("arg2","value2");

// 表示許可を行うユーザ
shortCutInfo.setAllowsUsers(new String[]{"guest","ueda"});

// ログイン認証が必要かどうか？
shortCutInfo.setAuth(true);

// この情報の有効期限(作成時から１０日間有効)
shortCutInfo.setValidEndDate(manager.addValidEndDate(10));

// ショートカットID 作成
String shortCutId = manager.createShortCut(shortCutInfo);
```

ショートカット拡張検証機能

拡張検証機能では、ショートカット情報の URL の表示を許可ユーザ以外で、ログインしたユーザに対して検証プログラム利用して、ページの表示許可を与えるかどうかの判定を追加で行う機能です。
拡張検証コードは、検証プログラムのコード名を表します。
拡張検証パラメータは、検証プログラムに渡されるパラメータとなります。

```
// ショートカットマネージャの作成
ShortCutManager manager = new ShortCutManager();

// ショートカット情報の作成
ShortCutInfo shortCutInfo = new ShortCutInfo();

// 表示する URL
shortCutInfo.setUrl("/sample/shortcut");

// 表示する URL に渡すパラメータの設定(任意指定)
shortCutInfo.setUrlParam("arg1","value1");
shortCutInfo.setUrlParam("arg2","value2");

// 表示許可を行うユーザ（検証機能のみを利用する場合は、設定しなくてもよい。）
shortCutInfo.setAllowsUsers(new String[]{"guest","ueda"});

// ログイン認証が必要かどうか？（検証機能を利用する場合は、設定した値にかかわらず、trueとなります。）
shortCutInfo.setAuth(true);

// この情報の有効期限(作成時から１０日間有効)
shortCutInfo.setValidEndDate(manager.addValidEndDate(10));

shortCutInfo.setValidationCode("RoleUser"); // 追加でユーザが許可されるかどうかの検証コード
shortCutInfo.setValidationParam(""); // 検証処理に渡される追加のパラメータ

// ショートカットID 作成
String shortCutId = manager.createShortCut(shortCutInfo);
```

検証プログラムの作成

検証プログラムの作成は **jp.co.intra_mart.foundation.security.shortcut.ShortCutValidator** を実装して作成します。

実装するメソッド ： boolean isAllowUser(String groupId, ShortCutInfo shortcutInfo, String userId) throws AccessSecurityException

このメソッドがtrueを返す場合は、許可されたユーザとなります。

```
package jp.co.intra_mart.foundation.security.shortcut;

import java.util.regex.Pattern;

import jp.co.intra_mart.foundation.security.exception.AccessSecurityException;

/**
 * 正規表現で許可ユーザを検証するショートカット検証クラス。 <br>
 * <br>
 * ショートカット情報の拡張検証パラメータに指定されている<br>
 * 正規表現とユーザ名が一致している場合に許可します。
 * @author INTRAMART
 * @version 8.0
 * @since 7.0
 */
public class RegExpUserShortCutValidator implements ShortCutValidator {

   /**
    * ショートカット情報の拡張検証パラメータに指定した正規表現がユーザIDと一致するか判定します。<br>
    * @param groupId ログイングループID
    * @param shortcutInfo ショートカット情報
    * @param userId ユーザID
    * @return 許可されたユーザであるなら true。
    * @throws AccessSecurityException 検証中にエラーが発生した場合スローされます。
    */
   @Override
   public boolean isAllowUser(final String groupId, final ShortCutInfo shortcutInfo, final String userId) throws
AccessSecurityException {
       return Pattern.matches(shortcutInfo.getValidationParam(), userId);
   }

}
```

検証コードと検証プログラムの設定

IM-Jugglingよりショートカットアクセス設定（short-cut-config.xml）に対して、以下の設定を追加します。

<validator>を追加します。

- 属性 code には、検証コードを設定します。
- 属性 class には、検証プログラムのクラスを設定ます。

```
<?xml version="1.0" encoding="UTF-8"?>
<short-cut-config>
  <short-cut-accessor>
    <short-cut-accessor-class>jp.co.intra_mart.foundation.security.shortcut.StandardShortCutAccessor</short-cut-
accessor-class>
    <error-page>/user/shortcut/error</error-page>
    <main-page>/home</main-page>
    <validator code="RegExpUser" class="jp.co.intra_mart.foundation.security.shortcut.RegExpUserShortCutValidator"/>
    <validator code="RoleUser" class="jp.co.intra_mart.foundation.security.shortcut.RoleUserShortCutValidator"/>
    <validator code="Script" class="jp.co.intra_mart.foundation.security.shortcut.ScriptShortCutValidator"/>
    <validator code="sample" class="jp.co.intra_mart.foundation.security.shortcut.SampleValidator"/>
  </short-cut-accessor>
</short-cut-config>
```

注意

IM-Jugglingよりショートカットアクセス設定（short-cut-config.xml）を編集するには、intra-mart Accel Platfom 2013 Autumn 以降が必要となります。
intra-mart Accel Platfom 2013 Autumn 以前の場合は、作成されたwarファイルより直接編集してください。

標準の検証コードについて

標準で3種類の検証コードが登録されています。

- 拡張検証パラメータに指定された正規表現にマッチするユーザ ID のユーザを許可ユーザとします。

検証コード	検証プログラム
RegExpUser	jp.co.intra_mart.foundation.security.shortcut.RegExpUserShortCutValidator

- 拡張検証パラメータに指定されたロール ID を持つユーザを許可ユーザとします。

検証コード	検証プログラム
RoleUser	jp.co.intra_mart.foundation.security.shortcut.RoleUserShortCutValidator

- スクリプトを実行して許可ユーザを検証します。

検証コード	検証プログラム
Script	jp.co.intra_mart.foundation.security.shortcut.ScriptShortCutValidator

ショートカット情報の拡張検証パラメータに指定したスクリプトの isAllowUserメソッドで判定します。
このモジュールを利用する場合は、ショートカット情報の拡張検証パラメータにスクリプトのパスを指定します。
拡張子はつけません。

例 ： shortcut/validator

また、スクリプトのisAllowUserメソッドに対して、パラメータ（param）を渡したい場合は、ショートカット情報の
拡張検証パラメータのスクリプトのパスに続いて、カンマ区切りで、パラメータを指定します。
例 ： shortcut/validator,パラメータ値
パラメータ値が指定されていない場合は、スクリプトの isAllowUser メソッドの param 引数には空文字列が渡されます。
スクリプトの isAllowUser メソッドのインタフェース

Boolean isAllowUser(String groupId, ShortCutInfo shortcutInfo,String userId,String param)

CSRF対策

項目

- 概要
- imSecureTokenタグを使ったCSRF対策
- AOPを使ってトークンチェック処理を実行する方法
  - 入力フォームのjsp
  - 登録処理のController
  - AOPを設定するjavaクラス
  - bean定義xmlファイル
  - applicationContext-im_tgfw_common.xml
  - applicationContext-im_tgfw_web.xml

概要

Cross-Site Request Forgery（以下 CSRF）対策として、 intra-mart Accel Platform ではimSecureTokenタグ を用意しています。 このタグにより生成されるトークンに対してサーバ側でトークンチェック処理を行うことにより、CSRF対策を行います。 ここでは、CSRF対策を実施したいメソッドにAOPを使ってトークンチェック処理を織り込む方法について説明します。

imSecureTokenタグを使ったCSRF対策

imSecureTokenタグでは、CSRF対策の方法としてトークンを使います。 jspのformにimSecureTokenタグを記述する、ajaxで送信するデータにトークンをセットするなどをして、 サーバに送信するリクエストにトークンをセットします。 サーバ側でトークンチェック処理を実装し、リクエストのトークンの検証を行い、CSRF対策を行います。

imSecureTokenタグでは、トークンチェック処理について2つの方法を紹介しています。

- token-filtering-target-configのxmlに設定を記述し、トークンチェック処理を自動で行う方法
- ユーザでトークンチェック処理を実装する方法

ここでは、後者の「ユーザでトークンチェック処理を実装する方法」をAOPを使って実行する方法を紹介します。

AOPを使ってトークンチェック処理を実行する方法

ここでは単純な登録処理を例にCSRF対策の方法を説明します。

対象となるファイルは以下の通りです。

- 入力フォームのjsp
- 登録処理のControllerクラス
- AOPを設定するjavaクラス
- bean定義xmlファイル
- applicationContext-im_tgfw_common.xml （2014 Winter(Iceberg)の変更点の確認）
- applicationContext-im_tgfw_web.xml （2014 Winter(Iceberg)の変更点の確認）

注意
ここでは、 intra-mart Accel Platform 2014 Winter(Iceberg) で導入された SecureTokenValidator、SecureTokenExceptionを利用しています。 お使いの intra-mart Accel Platform が 2014 Winter(Iceberg) 以上であることを確認してください。

入力フォームのjsp

登録する入力フォームに imSecureTokenタグを追記します。

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui"%>
<%@ taglib prefix="imst" uri="http://www.intra-mart.co.jp/taglib/imst" %>

<imui:head>
 <title>something creation</title>
</imui:head>

<div class="imui-title">
 <h1>something creation</h1>
</div>

<div class="imui-form-container-narrow">
 <form id="form" action="myapp/something/creation/create" method="POST">
 <!-- 入力フォームにimSecureTokenを追加します。 -->
 <imst:imSecureToken />

 ...

 </form>
</div>
```

登録処理のController

AOPでのPointcutの記述を簡単にするために、メソッド名やメソッド引数をパターン化しておきます。 また、トークンチェックに必要なHttpServletRequestを引数に含めます。

ここでは、登録処理のメソッド名をcreateとし、HttpServletRequestが引数の最後になるパターンにします。

```java
package sample.myapplication.app.something;

import javax.servlet.http.HttpServletRequest;
import javax.validation.Valid;

import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

@Controller
@RequestMapping("myapp/something/creation")
public class CreationController {

    /**
     * 登録フォーム表示
     */
    @RequestMapping(value = "create", params = "form", method = RequestMethod.GET)
    public String createForm() {
        return "myapp/something/createForm.jsp";
    }

    /**
     * 登録処理
     * AOP定義に合わせてメソッド名をcreateにしています。
     * また、引数の最後に HttpServletRequest を追加しています。
     */
    @RequestMapping(value = "create", method = RequestMethod.POST)
    public String create(@ModelAttribute @Valid final SimpleForm form, final BindingResult result,
            final RedirectAttributes model, final HttpServletRequest request) {
        if (result.hasErrors()) {
            // エラー処理
            // ...
            return "myapp/something/createForm.jsp";
        }

        // 登録処理

        model.addFlashAttribute(form);
        return "redirect:create?complete";
    }

    /**
     * 登録完了画面
     */
    @RequestMapping(value = "create", params = "complete", method = RequestMethod.GET)
    public String createComplete(@ModelAttribute final SimpleForm form) {
        return "myapp/something/createComplete.jsp";
    }

    @ModelAttribute
    public SimpleForm setUpForm() {
        final SimpleForm form = new SimpleForm();
        return form;
    }
}
```

AOPを設定するjavaクラス

ここでは汎用的なpointcut定義とCSRF対策に特化したpointcut定義の2クラスに分けています。

- 汎用的なpointcut定義

  登録処理のメソッド名に対してpointcutを定義します。
  ここでは登録処理のメソッド名としてcreateとしているので、それに合わせた設定にします。

```
1  package sample.myapplication.csrf;
2
3  import org.aspectj.lang.annotation.Aspect;
4  import org.aspectj.lang.annotation.Pointcut;
5
6  /**
7   * 汎用的なPointcutを定義します。
8   */
9  @Aspect
10 public class ApplicationArchitecture {
11
12    @Pointcut("execution(* sample.myapplication.app.*.*.create(..))")
13    public void createOperation() {
14    }
15
16    @Pointcut("execution(* sample.myapplication.app.*.*.remove(..))")
17    public void removeOperation() {
18    }
19
20    @Pointcut("execution(* sample.myapplication.app.*.*.update(..))")
21    public void updateOperation() {
22    }
23 }
```

- CSRF対策のAOP定義

  CSRF対策のトークチェックを実行するpointcutを定義します。
  メソッドの引数の最後にHttpServletRequestがあるパターンを追加しています。

```java
package sample.myapplication.csrf;

import javax.inject.Inject;
import javax.servlet.http.HttpServletRequest;

import jp.co.intra_mart.foundation.secure_token.SecureTokenException;
import jp.co.intra_mart.framework.extension.spring.web.csrf.SecureTokenValidator;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

/**
 * セキュアトークン検証処理を行うaspectを定義したクラスです。
 */
@Aspect
public class CsrfAspect {

    @Inject
    private SecureTokenValidator secureTokenValidator;

    /**
     * beforeでトークンチェック処理を実行します。
     */
    @Before("create(request) || update(request) || remove(request)")
    public void doSecureTokenCheck(final HttpServletRequest request) throws SecureTokenException {
        // トークンチェック処理を実行します。
        secureTokenValidator.validate(request);
    }

    /**
     * 登録処理でCSRF対策を行うPointcutです。
     * ApplicationArchitectureに定義した登録処理のPointcutに、トークンチェックのために引数の条件を追加しています。
     */
    @Pointcut("sample.myapplication.csrf.ApplicationArchitecture.createOperation() && args(..,request)")
    private void create(final HttpServletRequest request) {
    }

    /**
     * 削除処理でCSRF対策を行うPointcutです。
     * ApplicationArchitectureに定義した削除処理のPointcutに、トークンチェックのために引数の条件を追加しています。
     */
    @Pointcut("sample.myapplication.csrf.ApplicationArchitecture.removeOperation() && args(..,request)")
    private void remove(final HttpServletRequest request) {
    }

    /**
     * 更新処理でCSRF対策を行うPointcutです。
     * ApplicationArchitectureに定義した更新処理のPointcutに、トークンチェックのために引数の条件を追加しています。
     */
    @Pointcut("sample.myapplication.csrf.ApplicationArchitecture.updateOperation() && args(..,request)")
    private void update(final HttpServletRequest request) {
    }
}
```

SpringのAOPの詳細については、「Aspect Oriented Programming with Spring」を参照してください。

bean定義xmlファイル

AOPを定義したjavaクラスのbean定義をbean定義xmlに記述します。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/aop
https://www.springframework.org/schema/aop/spring-aop.xsd
        http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- @AspectJ Support の有効化 -->
    <aop:aspectj-autoproxy />
    <!-- AOPの汎用的な定義 -->
    <bean class="sample.myapplication.csrf.ApplicationArchitecture" />
    <!-- AOPのCSRF対策用の定義 -->
    <bean class="sample.myapplication.csrf.CsrfAspect" />

</beans>
```

applicationContext-im_tgfw_common.xml

intra-mart Accel Platformを2014 Summer(Honoka)以前から2014 Winter(Iceberg)以降へとバージョンアップした場合、applicationContext-im_tgfw_common.xmlに、以下の変更が反映されていることを確認してください。

- exceptionCodeResolverのbean定義
  InvalidSecureTokenExceptionを追加しています。

```
    <!-- Exception Code Resolver. -->
    <bean id="exceptionCodeResolver"
      class="org.terasoluna.gfw.common.exception.SimpleMappingExceptionCodeResolver">
      <!-- Setting and Customization by project. -->
      <property name="exceptionMappings">
        <map>
          <entry key="ResourceNotFoundException" value="w.im.fw.0001" />
          <entry key="InvalidTransactionTokenException" value="w.im.fw.0004" />
          <entry key="InvalidSecureTokenException" value="w.im.fw.0005" />
          <entry key="BusinessException" value="w.im.fw.0002" />
        </map>
      </property>
      <property name="defaultExceptionCode" value="e.im.fw.0001" />
    </bean>
```

applicationContext-im_tgfw_web.xml

intra-mart Accel Platformを2014 Summer(Honoka)以前から2014 Winter(Iceberg)以降へとバージョンアップした場合、applicationContext-im_tgfw_web.xmlに、以下の変更が反映されていることを確認してください。

- secureTokenValidatorのbean定義
  secureTokenValidatorのbean定義をしています。

```
    <!-- secure token validator -->
    <bean id="secureTokenValidator"
class="jp.co.intra_mart.framework.extension.spring.web.csrf.SecureTokenValidator" />
```

- SystemExceptionResolverのbean定義
  secureTokenErrorの設定を追加しています。

```
    <!-- Setting Exception Handling. -->
    <!-- Exception Resolver. -->
    <bean class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">
      <property name="exceptionCodeResolver" ref="exceptionCodeResolver" />
      <!-- Setting and Customization by project. -->
      <property name="order" value="3" />
      <property name="exceptionMappings">
        <map>
          <entry key="ResourceNotFoundException" value="im_tgfw/common/error/resourceNotFoundError.jsp" />
          <entry key="BusinessException" value="im_tgfw/common/error/businessError.jsp" />
          <entry key="InvalidTransactionTokenException" value="im_tgfw/common/error/transactionTokenError.jsp"
/>
          <entry key="InvalidSecureTokenException" value="im_tgfw/common/error/secureTokenError.jsp" />
        </map>
      </property>
      <property name="statusCodes">
        <map>
          <entry key="im_tgfw/common/error/resourceNotFoundError" value="404" />
          <entry key="im_tgfw/common/error/businessError" value="200" />
          <entry key="im_tgfw/common/error/transactionTokenError" value="409" />
          <entry key="im_tgfw/common/error/secureTokenError" value="403" />
        </map>
      </property>
      <property name="excludedExceptions">
        <array>
          <value>org.springframework.web.util.NestedServletException</value>
        </array>
      </property>
      <property name="defaultErrorView" value="im_tgfw/common/error/systemError.jsp" />
      <property name="defaultStatusCode" value="500" />
    </bean>
```

RESTful Web Service

項目

- 概要
- REST APIの作成
- REST APIの実装
  - Web API Maker のファクトリクラスの作成
  - Web API Maker のサービスクラスの作成
  - packagesファイルの作成

概要

ここでは intra-mart Accel Platform 上で TERASOLUNA Server Framework for Java (5.x) のREST APIの実装方法について説明します。

TERASOLUNA Server Framework for Java (5.x) でREST APIを作成する場合には、@RestControllerアノテーションを付けたControllerクラスを作ります。 intra-mart Accel Platform ではControllerクラスの替わりに Web API Maker を使います。 Web API Maker のサービスクラスを intra-mart Accel Platform のControllerクラスに対応させ、 intra-mart Accel Platform のServiceクラスを呼出しビジネスロジックを実行します。

Web API Maker についての詳細は、 「Web API Maker プログラミングガイド」を参照してください。

REST APIの作成

TERASOLUNA Server Framework for Java (5.x) Development Guideline の 「チュートリアル（Todoアプリケーション REST編）」 と同じようにtodoを公開するためのREST APIを作成します。

API名	HTTP メソッド	パス	ステータス コード	説明
GET Todos	GET	/api/v1/todos	200 (OK)	Todoリソースを全件取得する。
POST Todos	POST	/api/v1/todos	201 (Created)	Todoリソースを新規作成する。
Get Todo	GET	/api/v1/todos/{todoId}	200 (OK)	Todoリソースを一件取得する。
PUT Todo	PUT	/api/v1/todos/{todoId}	200 (OK)	Todoリソースを完了状態に更新する。
DELETE Todo	DELETE	/api/v1/todos/{todoId}	204 (No Content)	Todoリソースを削除する。

REST APIの実装

以下の手順で実装を進めます。

- Web API Maker のファクトリクラスの作成
  TodoServiceFactory を作成します。
- Web API Maker のサービスクラスの作成
  TodoApiService, TodoApiServiceImpl を作成します。
- packagesファイルの作成
  src/main/resources/META-INF/im_web_api_maker/packages を作成します。

intra-mart Accel Platform のServiceクラスとして、TodoService, TodoServiceImplが既にあるとします。

Web API Maker のファクトリクラスの作成

Web API Maker のサービスクラスに intra-mart Accel Platform の Service クラスをDIするために、 getService() メソッドでは ApplicationContextProvider.getApplicationContext() から Web API Maker のサービスクラスのビーンを取得しています。

```java
package com.sample.todo.api.todo;

import jp.co.intra_mart.foundation.web_api_maker.annotation.ProvideFactory;
import jp.co.intra_mart.foundation.web_api_maker.annotation.ProvideService;
import jp.co.intra_mart.foundation.web_api_maker.annotation.WebAPIMaker;
import jp.co.intra_mart.framework.extension.spring.context.ApplicationContextProvider;

@WebAPIMaker
public class TodoServiceFactory {

    @ProvideFactory
    public static TodoServiceFactory getFactory() {
        return new TodoServiceFactory();
    }

    @ProvideService
    public TodoApiService getService() {
        return ApplicationContextProvider.getApplicationContext().getBean(TodoApiService.class);
    }
}
```

ファクトリクラスには @WebAPIMaker を記述します。
ファクトリ取得メソッドに @ProvideFactory 、サービスクラス取得メソッドに @ProvideService を記述します。
サービスクラス取得メソッドでは、ApplicationContextProvider.getApplicationContext() から Web API Maker のサービスクラスのビーンを取得します。

Web API Maker のサービスクラスの作成

TodoApiService インタフェースと TodoApiServiceImpl 実装クラスを作成します。 Web API Makerのアノテーションはインタフェース側に記述します。

```java
package com.sample.todo.api.todo;

import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import jp.co.intra_mart.foundation.web_api_maker.annotation.BasicAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Body;
import jp.co.intra_mart.foundation.web_api_maker.annotation.DELETE;
import jp.co.intra_mart.foundation.web_api_maker.annotation.GET;
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.PUT;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Response;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Variable;
import com.sample.todo.api.NotFoundException;

@BasicAuthentication(pathPrefix = "")
@Todo
public interface TodoApiService {

    @GET(summary = "Todo情報を取得します。", description = "Todo情報の一覧を取得します。")
    @Path("/api/v1/todos")
    @TodoTag
    public List<TodoResource> getTodos();

    @POST(summary = "Todo情報を登録します。", description = "Todo情報を登録します。idは自動採番します。")
    @Path("/api/v1/todos")
    @Response(code = 201)
    @TodoTag
    public TodoResource postTodos(@Required @Body TodoResource todoResource, HttpServletRequest request,
HttpServletResponse response);

    @GET(summary = "Todo情報を取得します。", description = "指定したTodo情報を取得します。指定したTodo情報が無い場合、
404を返します。")
    @Path("/api/v1/todos/{todoId}")
    @TodoTag
    public TodoResource getTodo(@Required @Variable(name = "todoId") String todoId) throws NotFoundException;

    @PUT(summary = "Todoを完了します。", description = "Todo情報の完了フラグをONにします。")
    @Path("/api/v1/todos/{todoId}")
    @TodoTag
    public TodoResource putTodo(@Required @Variable(name = "todoId") String todoId);

    @DELETE(summary = "Todo情報を削除します。", description = "Todo情報を削除します。")
    @Path("/api/v1/todos/{todoId}")
    @Response(code = 204)
    @TodoTag
    public void deleteTodo(@Required @Variable(name = "todoId") String todoId);
}
```

@GET, @POSTなどHTTPメソッドを表しています。 @PathでURIと各メソッドの対応を表しています。
@Responseはステータスコードを指定する時に使います。省略時は200になります。

@BasicAuthentication はベーシック認証を使うことを示すアノテーションです。
@Todo, @TodoTag は APIドキュメントアノテーションです。
それぞれのアノテーションについては、「Web API Maker プログラミングガイド」を参照してください。

```
package com.sample.todo.api.todo;

import java.net.URI;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import javax.inject.Inject;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.github.dozermapper.core.Mapper;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;
import org.terasoluna.gfw.common.exception.ResourceNotFoundException;

import com.sample.todo.api.NotFoundException;
import com.sample.todo.domain.model.Todo;
import com.sample.todo.domain.service.todo.TodoService;

@Component
public class TodoApiServiceImpl implements TodoApiService {

    @Inject
    TodoService todoService;

    @Inject
    Mapper mapper;

    @Override
    public List<TodoResource> getTodos() {
        Collection<Todo> todos = todoService.findAll();
        List<TodoResource> todoResources = new ArrayList<>();
        for (Todo todo : todos) {
            todoResources.add(mapper.map(todo, TodoResource.class));
        }
        return todoResources;
    }

    @Override
    public TodoResource postTodos(TodoResource todoResource, HttpServletRequest request, HttpServletResponse
response) {
        Todo createdTodo = todoService.create(mapper.map(todoResource, Todo.class));
        TodoResource createdTodoResponse = mapper.map(createdTodo, TodoResource.class);
        ServletUriComponentsBuilder uriBuilder = ServletUriComponentsBuilder.fromRequest(request);
        URI uri = uriBuilder.pathSegment(createdTodoResponse.getTodoId()).build().toUri();
        response.addHeader("Location", uri.toASCIIString());
        return createdTodoResponse;
    }

    @Override
    public TodoResource getTodo(String todoId) throws NotFoundException {
        try {
            Todo todo = todoService.findOne(todoId);
            TodoResource todoResource = mapper.map(todo, TodoResource.class);
            return todoResource;
        } catch (ResourceNotFoundException e) {
            throw new NotFoundException(e.getMessage(), e);
        }
    }

    @Override
    public TodoResource putTodo(String todoId) {
        Todo finishedTodo = todoService.finish(todoId);
        TodoResource finishedTodoResource = mapper.map(finishedTodo, TodoResource.class);
        return finishedTodoResource;
    }

    @Override
    public void deleteTodo(String todoId) {
        todoService.delete(todoId);
    }
```

```
}
```

@Component をつけて TodoApiServiceImplをビーン登録します。
@Inject TodoService で Service をDIします。

```
package com.sample.todo.api;

import jp.co.intra_mart.foundation.web_api_maker.annotation.Response;

@Response(code = 404)
public class NotFoundException extends Exception {

    private static final long serialVersionUID = 1L;

    public NotFoundException() {
        super();
    }

    public NotFoundException(String message, Throwable cause, boolean enableSuppression, boolean writableStackTrace)
{
        super(message, cause, enableSuppression, writableStackTrace);
    }

    public NotFoundException(String message, Throwable cause) {
        super(message, cause);
    }

    public NotFoundException(String message) {
        super(message);
    }

    public NotFoundException(Throwable cause) {
        super(cause);
    }
}
```

例外時に特定のステータスコードを返したい場合には @Response アノテーションを記述した例外クラスを使います。
ここでは TodoApiServiceImpl#getTodo(String) で NotFoundException がスローされた場合、 404のステータスコードを返します。

packagesファイルの作成

「src/main/resources/META-INF/im_web_api_maker」 ディレクトリに 「packages」 ファイルを作成し、ファクトリクラスのパッケージを記述します。 このサンプルでは、 「com.sample.todo.api.todo」 になります。

設定ファイル

項目

- 設定ファイルについて
- ConfigurationLoader

設定ファイルについて

WEB-INF/conf 配下の設定ファイルは WEB-INF/schema 配下のスキーマ定義に基づいており、ConfigurationLoader を使用して読み込んでいます。スキーマ定義を追加すれば intra-mart が提供している設定だけでなく、新しい設定を読み込むこともできます。

ConfigurationLoader

ConfigurationLoader（jp.co.intra_mart.foundation.config.ConfigurationLoader）の設定ファイル読み込みについて説明します。
ConfigurationLoader クラスの詳細については、「ConfigurationLoader クラスの APIドキュメント」を参照してください。

ConfigurationLoader の設定ファイル読み込み

項目

- 概要
- 設定ファイルの読み込み
- サンプル
  - 読み込み対象のサンプル
  - 読み込み処理のサンプル

概要

ConfigurationLoader（jp.co.intra_mart.foundation.config.ConfigurationLoader）を使用した設定ファイルの読み込みについて説明します。

設定ファイルの読み込み

設定ファイルの読み込みには ConfigurationLoader#load or #loadAll メソッドを使用し、スキーマ定義に基づいた Java オブジェクトに変換されて取得できます。

```
// 設定ファイルのデータを読み込み、スキーマ定義に基づいた Java オブジェクトを作成します
JaxbConfig config = ConfigurationLoader.load(JaxbConfig.class);
```

コラム

ConfigurationLoader#load or #loadAll メソッドの設定ファイルから Java オブジェクトへの変換は
JAXB（Java Architecture for XML Binding）を利用しています。
intra-mart が提供している設定以外を ConfigurationLoader で読み込む場合は以下のファイルが必要です。

- スキーマ定義（xsd）
- スキーマに基づく設定ファイル（xml）
- スキーマに基づくクラス
- スキーマに基づくクラスのインスタンスを生成するクラス

サンプル

ConfigurationLoader を使用して設定ファイルのデータを取得するサンプルです。

読み込み対象のサンプル

- スキーマの定義（xsd）

  WEB-INF/schema ディレクトリ配下に配置します。

  スキーマのサンプル（WEB-INF/schema/jaxb-config.xsd）

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://intra_mart.co.jp/jaxb/sample/jaxb-config"
 xmlns:tns="http://intra_mart.co.jp/jaxb/sample/jaxb-config"
 xmlns:jaxb="http://java.sun.com/xml/ns/jaxb" jaxb:version="2.0"
 elementFormDefault="qualified">

  <xs:annotation>
    <xs:appinfo>
      <jaxb:schemaBindings>
        <jaxb:package name="jp.co.intra_mart.jaxb.sample" />
      </jaxb:schemaBindings>
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="jaxb-config">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="jaxbNestedConfig">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="value" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

- スキーマに基づく設定ファイル（xml）

WEB-INF/conf ディレクトリ配下に配置します。

設定ファイルのサンプル（WEB-INF/conf/jaxb-config.xml）

```
<?xml version="1.0" encoding="UTF-8"?>
<jaxb-config
  xmlns="http://intra_mart.co.jp/jaxb/sample/jaxb-config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://intra_mart.co.jp/jaxb/sample/jaxb-config">

  <jaxbNestedConfig>
    <value>sample_value</value>
  </jaxbNestedConfig>

</jaxb-config>
```

- スキーマに基づくクラス

クラスパスの通っているディレクトリ配下に配置します。

スキーマに基づくクラスのサンプル（jp.co.intra_mart.jaxb.sample.JaxbConfig）

```
package jp.co.intra_mart.jaxb.sample;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "jaxbNestedConfig"
}, factoryClass = ObjectFactory.class , factoryMethod = "createJaxbConfig")
@XmlRootElement(name = "jaxb-config")
public class JaxbConfig {

    @XmlElement(required = true)
    protected JaxbConfig.JaxbNestedConfig jaxbNestedConfig;

    public JaxbConfig.JaxbNestedConfig getJaxbNestedConfig() {
        return jaxbNestedConfig;
    }

    public void setJaxbNestedConfig(JaxbConfig.JaxbNestedConfig value) {
        this.jaxbNestedConfig = value;
    }

    @XmlAccessorType(XmlAccessType.FIELD)
    @XmlType(name = "", propOrder = {
        "value"
    }, factoryClass = ObjectFactory.class , factoryMethod = "createJaxbConfigJaxbNestedConfig")
    public static class JaxbNestedConfig {

        @XmlElement(required = true)
        protected String value;

        public String getValue() {
            return value;
        }

        public void setValue(String value) {
            this.value = value;
        }
    }
}
```

- スキーマに基づくクラスのインスタンスを生成するクラス

クラスパスの通っているディレクトリ配下に配置します。

インスタンスを生成するクラスのサンプル（jp.co.intra_mart.jaxb.sample.ObjectFactory）

```
package jp.co.intra_mart.jaxb.sample;

import javax.xml.bind.annotation.XmlRegistry;

@XmlRegistry
public class ObjectFactory {

    public ObjectFactory() {
    }

    public static JaxbConfig createJaxbConfig() {
        return new JaxbConfig();
    }

    public static JaxbConfig.JaxbNestedConfig createJaxbConfigJaxbNestedConfig() {
        return new JaxbConfig.JaxbNestedConfig();
    }

}
```

読み込み処理のサンプル

ConfigurationLoader#load メソッドを使用して追加した設定を読み込み、設定ファイルのデータを取得します。

```
// jaxb-config.xml を読み込んでオブジェクトを取得します。
JaxbConfig config = ConfigurationLoader.load(JaxbConfig.class);
JaxbNestedConfig nestedConfig = config.getJaxbNestedConfig();

// jaxb-config.xml に定義した「sample_value」を取得できます。
String value = nestedConfig.getValue();
```

ConfigurationLoader 使用時の注意事項

項目

- 概要
- 注意事項
- スキーマ定義から自動生成したクラスの修正
  - 修正対象
  - 修正方法
    - @XmlRegistry アノテーションが付与されているクラスの修正
    - @XmlType アノテーションが付与されているクラスの修正
  - 修正例
    - @XmlRegistry アノテーションが付与されているクラス
    - @XmlType アノテーションが付与されているクラス

概要

ConfigurationLoader#load or #loadAll メソッドを使用するにあたって注意すべき点について説明します。

注意事項

ConfigurationLoader#load or #loadAll メソッドは、JAXB（Java Architecture for XML Binding）を利用して設定ファイルからデータを読み込んでいるため、使用するには設定ファイル、スキーマ定義、および、スキーマ定義に基づいたクラスが必要となります。
しかし、JAXB を利用してスキーマ定義から自動生成したクラスをConfigurationLoader で使用すると ThreadLocal が開放されずに**メモリリーク** します。

ConfigurationLoader を使用して設定ファイルを読み込む場合はスキーマ定義から自動生成したクラスを修正してください。

スキーマ定義から自動生成したクラスの修正

修正対象

スキーマ定義から自動生成したクラスがメモリリークしないようにするため、以下のクラスを修正する必要があります。

- @XmlRegistry アノテーションが付与されているクラス
- @XmlType アノテーションが付与されているクラス

修正方法

@XmlRegistry アノテーションが付与されているクラスの修正

インスタンスを生成しているメソッドを **static メソッド** に修正してください。
通常は自動生成時に @XmlType アノテーションが付与されているクラスのインスタンスを生成するメソッドが定義されていますが
不足している場合はインスタンスを生成して返却する static メソッドを作成してください。

```
@XmlRegistry
public class ObjectFactory {

    public static JaxbConfig createJaxbConfig() {
        return new JaxbConfig();
    }

    public static JaxbConfig.JaxbNestedConfig createJaxbConfigJaxbNestedConfig() {
        return new JaxbConfig.JaxbNestedConfig();
    }

}
```

コラム

自動生成時のインスタンス生成メソッドは以下の命名規則に従って定義されます。

- クラス名の先頭に「create」を加えた「create（クラス名）」で定義されます。
- ネストクラスに @XmlType アノテーションが付与されている場合は
  「create（外側のクラス名）（ネストクラス名）」で定義されます。

@XmlType アノテーションが付与されているクラスの修正

@XmlType アノテーションに **factoryClass**、および、**factoryMethod** を指定してください。

- **factoryClass**
  同一パッケージ内のクラスから @XmlRegistry アノテーションが付与されているクラスを指定してください。
- **factoryMethod**
  factoryClass で指定したクラスのメソッドから自身のインスタンスを生成するメソッドを文字列で指定してください。

```
@XmlType(name = "", propOrder = {
    "jaxbNestedConfig"
}, factoryClass = ObjectFactory.class , factoryMethod = "createJaxbConfig")
@XmlRootElement(name = "jaxb-config")
public class JaxbConfig {

    @XmlType(name = "", propOrder = {
        "value"
    }, factoryClass = ObjectFactory.class , factoryMethod = "createJaxbConfigJaxbNestedConfig")
    public static class JaxbNestedConfig {
```

修正例

以下のスキーマ定義から自動生成されたクラスの修正例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://intra_mart.co.jp/jaxb/sample/jaxb-config"
 xmlns:tns="http://intra_mart.co.jp/jaxb/sample/jaxb-config"
 xmlns:jaxb="http://java.sun.com/xml/ns/jaxb" jaxb:version="2.0"
 elementFormDefault="qualified">

  <xs:annotation>
    <xs:appinfo>
      <jaxb:schemaBindings>
        <jaxb:package name="jp.co.intra_mart.jaxb.sample" />
      </jaxb:schemaBindings>
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="jaxb-config">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="jaxbNestedConfig">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="value" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

@XmlRegistry アノテーションが付与されているクラス

自動生成時はメソッドに static 修飾子がありません。処理内容は変更せず、static メソッドに修正するのみです。

自動生成時

```
package jp.co.intra_mart.jaxb.sample;

import javax.xml.bind.annotation.XmlRegistry;

@XmlRegistry
public class ObjectFactory {

  public ObjectFactory() {
  }

  public JaxbConfig createJaxbConfig() {
    return new JaxbConfig();
  }

  public JaxbConfig.JaxbNestedConfig createJaxbConfigJaxbNestedConfig() {
    return new JaxbConfig.JaxbNestedConfig();
  }

}
```

修正例

```
package jp.co.intra_mart.jaxb.sample;

import javax.xml.bind.annotation.XmlRegistry;

@XmlRegistry
public class ObjectFactory {

    public ObjectFactory() {
    }

    public static JaxbConfig createJaxbConfig() {
        return new JaxbConfig();
    }

    public static JaxbConfig.JaxbNestedConfig createJaxbConfigJaxbNestedConfig() {
        return new JaxbConfig.JaxbNestedConfig();
    }

}
```

@XmlType アノテーションが付与されているクラス

自動生成時は @XmlType アノテーションに factoryClass , factoryMethod の指定がありません。
@XmlType アノテーションは JaxbConfig クラス、および JaxbNestedConfig クラスに付与されているため
二箇所 factoryClass , factoryMethod を指定する修正が必要です。

- **factoryClass**
  @XmlRegistry アノテーションは ObjectFactory クラスに付与されているため、両クラスとも
  @XmlType アノテーションの factoryClass には「factoryClass = ObjectFactory.class」と指定します。
- **factoryMethod**
  ObjectFactory クラスには両クラスのインスタンスを生成するメソッドが定義されているため
  各 @XmlType アノテーションの factoryMethod に自身のインスタンスを生成するメソッドを指定します。
  JaxbConfig クラスは「factoryMethod = “createJaxbConfig”」と指定します。
  JaxbNestedConfig クラスは「factoryMethod = “createJaxbConfigJaxbNestedConfig”」と指定します。

  自動生成時

```
package jp.co.intra_mart.jaxb.sample;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "jaxbNestedConfig"
})
@XmlRootElement(name = "jaxb-config")
public class JaxbConfig {

    @XmlElement(required = true)
    protected JaxbConfig.JaxbNestedConfig jaxbNestedConfig;

    public JaxbConfig.JaxbNestedConfig getJaxbNestedConfig() {
        return jaxbNestedConfig;
    }

    public void setJaxbNestedConfig(JaxbConfig.JaxbNestedConfig value) {
        this.jaxbNestedConfig = value;
    }

    @XmlAccessorType(XmlAccessType.FIELD)
    @XmlType(name = "", propOrder = {
        "value"
    })
    public static class JaxbNestedConfig {

        @XmlElement(required = true)
        protected String value;

        public String getValue() {
            return value;
        }

        public void setValue(String value) {
            this.value = value;
        }
    }
}
```

修正例

```
package jp.co.intra_mart.jaxb.sample;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "jaxbNestedConfig"
}, factoryClass = ObjectFactory.class , factoryMethod = "createJaxbConfig")
@XmlRootElement(name = "jaxb-config")
public class JaxbConfig {

    @XmlElement(required = true)
    protected JaxbConfig.JaxbNestedConfig jaxbNestedConfig;

    public JaxbConfig.JaxbNestedConfig getJaxbNestedConfig() {
        return jaxbNestedConfig;
    }

    public void setJaxbNestedConfig(JaxbConfig.JaxbNestedConfig value) {
        this.jaxbNestedConfig = value;
    }

    @XmlAccessorType(XmlAccessType.FIELD)
    @XmlType(name = "", propOrder = {
        "value"
    }, factoryClass = ObjectFactory.class , factoryMethod = "createJaxbConfigJaxbNestedConfig")
    public static class JaxbNestedConfig {

        @XmlElement(required = true)
        protected String value;

        public String getValue() {
            return value;
        }

        public void setValue(String value) {
            this.value = value;
        }
    }
}
```

参考（ TERASOLUNA Server Framework for Java (5.x) for Accel Platform アーキテクチャ ）

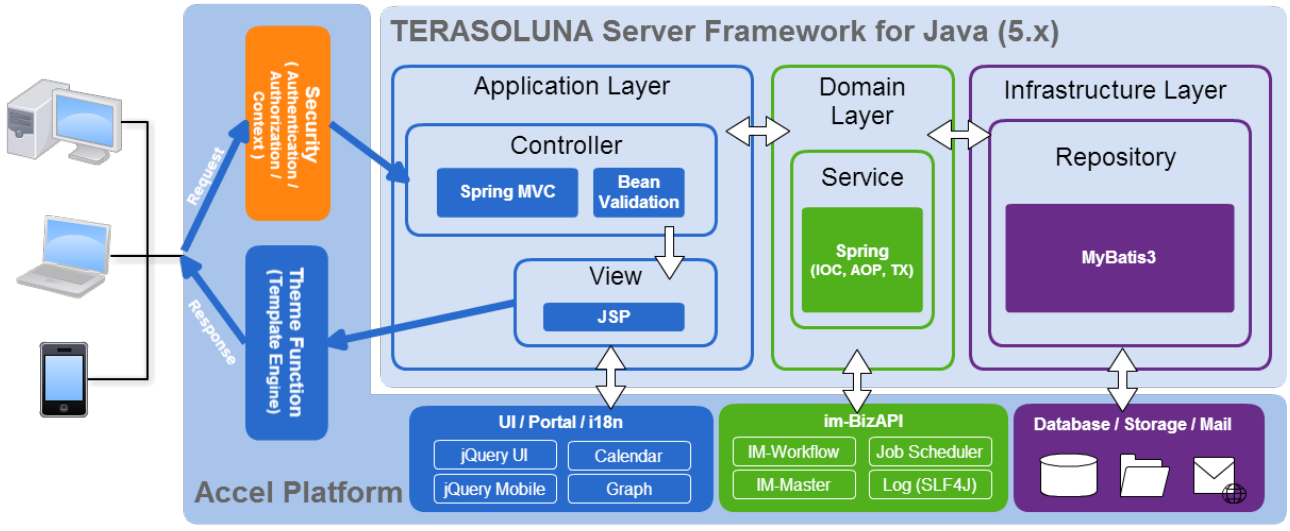
TERASOLUNA Server Framework for Java (5.x) (http://terasolunaorg.github.io/) は、Spring Framework (http://projects.spring.io/spring-framework/) をベースに構成されるJavaフレームワークです。
TERASOLUNA Server Framework for Java (5.x) の詳細については TERASOLUNA Server Framework for Java (5.x) Development Guideline にて公開されています。
本項では、 intra-mart Accel Platform と TERASOLUNA Server Framework for Java (5.x) によるフルスタックフレームワークについて説明します。

TERASOLUNA Server Framework for Java (5.x) for Accel Platform アーキテクチャ

- TERASOLUNA Server Framework for Java (5.x) for Accel Platform の構成
- TERASOLUNA Server Framework for Java (5.x) との対応
  - Webアプリ開発機能
  - Web Service
  - データアクセス
  - アプリケーション形態に依存しない汎用機能
  - メッセージ連携
  - セキュリティ対策
- TERASOLUNA Server Framework for Java (5.x) for Accel Platform におけるBean定義の既定値
  - web.xml
  - applicationContext-im_tgfw_common.xml
  - applicationContext-im_tgfw_web.xml
  - SpringMVCServlet-servlet.xml
  - applicationContext-im_tgfw_mybatis3.xml

TERASOLUNA Server Framework for Java (5.x) for Accel Platform の構成

intra-mart Accel Platform 上に TERASOLUNA Server Framework for Java (5.x) を組み込んだ「フルスタックフレームワーク」のイメージは下図の通りです。



TERASOLUNA Server Framework for Java (5.x) は、下記の通り、Spring Framework がベースで構成され、トランザクショントークンチェック、コードリスト、例外ハンドリングやロギング等の機能を提供し、その活用方法を「 TERASOLUNA Server Framework for Java (5.x) Development Guideline 」として提供しています。

アプリケーション層	Spring MVC / Bean Validation
ドメイン層	Spring Framework
インフラストラクチャ層	MyBatis3

TERASOLUNA Server Framework for Java (5.x) for Accel Platform では、intra-mart Accel Platform が提供する以下の機能を利用して、TERASOLUNA Server Framework for Java (5.x) のアプリケーションを効率よく開発できます。

- *認可*
- *アクセスコンテキスト*
- *認証機能*
- *国際化*
- *データベース*
- *ログ*
- *UI（デザインガイドライン）*
- *UI（スマートフォン開発ガイドライン）*
- *エラー処理*
- *Storage*
- *非同期処理*
- *ジョブスケジューラ*
- *Lockサービス*
- *Cacheサービス*
- *ショートカットアクセス機能*
- *ポートレット*
- IM-Workflow
- etc...

コラム

intra-mart Accel Platform が提供する各機能の使い方については上記リンクより参照にしてください。

TERASOLUNA Server Framework for Java (5.x) との対応

TERASOLUNA Server Framework for Java (5.x) Development Guidelineで想定している TERASOLUNA Server Framework for Java (5.x) の TERASOLUNA Server Framework for Java (5.x) for Accel Platform 上での対応は以下の通りです。

表中の記号は以下の意味を示します。
✔：対応、⚠：注意点有り、✖：非対応

Webアプリ開発機能

機能	対応	備考
入力チェック	✔	
例外ハンドリング	✔	
セッション管理	⚠	Spring Securityを使うものについては対象外です。
ページネーション	✖	imuiTableを使用してください。
二重送信防止	✔	
メッセージ管理	✖	IntramartMessageSource を使用してください。 また、「*多言語化されたメッセージを取得する*」を参照してください。
国際化	⚠	IntramartMessageSource, AccountLocaleResolverを使用してください。 また、「*国際化*」を参照してください。
コードリスト	✔	Bean定義 applicationContext-im_tgfw_web.xml にコードリストの設定を記述しています。 <mvc:mapping>のpath属性は、適用対象のパスを設定してください。
ファイルアップロード	⚠	Servlet 3.0の機能は非対応です。 multipartResolverにはCommonsMultipartResolverを設定しています。 また、「*Storage*」を参照してください。

機能	対応	備考
ファイルダウンロード	✔	「*Storage*」を参照してください。
Tiles	✖	intra-mart Accel Platform のテーマ機能を使用してください。
JSP Tag Library と EL Functions	✔	
Ajax	✔	
ヘルスチェック	✖	

Web Service

機能	対応	備考
RESTful Web Service	✖	@RestControllerの代わりにWeb API Makerを使用してください。 service層は、applicationContextからbeanを取得してください。 Web API Makerについては「Web API Maker プログラミングガイド」を参照してください。
RESTクライアント（HTTPクライアント）	✔	
SOAP Web Service（サーバ/クライアント）	✖	「Webサービス Java開発プログラミングガイド」を参照してください。

データアクセス

機能	対応	備考
データアクセス（共通編）	✔	シンプルなCRUD操作で動作確認を行っております。
データアクセス（JPA編）	✖	intra-mart Accel Platform では対応していません。
データアクセス（Mybatis3編）	✔	シンプルなCRUD操作で動作確認を行っております。
排他制御	✔	

アプリケーション形態に依存しない汎用機能

機能	対応	備考
ロギング	⚠	Logger, MDCは intra-mart Accel Platform のものを使用してください。 また、「*ログ*」を参照してください。 TraceLoggingInterceptor, ExceptionLoggerについては、 「TERASOLUNA Server Framework for Java (5.x) Development Guideline」を参照してください。
プロパティ管理	✔	
日付操作(JSR-310 Date and Time API)	✔	
日付操作(Joda Time)	✔	
システム時刻	⚠	org.terasoluna.gfw.common.date.JdbcAdjustedDateFactory を使用する場合、 dataSourceプロパティにはシェアードデータソースを指定してください。 シェアードデータソースの設定は、 「DataSource」や 「*データベース*」を参照してください。
文字列処理	✔	
Beanマッピング(Dozer)	✔	

メッセージ連携

機能	対応	備考
E-mail送信(SMTP)	✖	
JMS(Java Message Service)	✖	

セキュリティ対策

Spring Security は非対応です。

セキュリティ対策	対応	備考
Spring Security （認証、認可、パスワードハッシュ化）	✖	認証、認可、パスワードハッシュ化機能については、 intra-mart Accel Platform の認証、認可を使用してください。 intra-mart Accel Platform でもパスワードのハッシュ化を行っています。
XSS対策	✔	
CSRF対策	✖	intra-mart Accel PlatformのimSecureTokenタグを使用してください。 「imSecureTokenタグ」、「*CSRF対策*」を参照してください。

TERASOLUNA Server Framework for Java (5.x) for Accel Platform におけるBean定義の既定値

Spring MVC では、DispatcherServlet というサーブレットクラスが提供されています。
DispatcherServletは、クライアントからの要求に対して、要求情報からコントローラを探し、コントローラを呼び出す役割を担います。
intra-mart Accel Platform では、クライアントからの要求に対して、該当するプログラムを探し、認可のチェックをし、見つかったプログラムを呼び出す「ルーティング機構」があります。
intra-mart Accel Platform では、この「ルーティング機構」からSpring MVCのコントローラを呼び出す機構が組み込まれています。

> **コラム**
>
> 「ルーティング機構」の仕組みについては、「*認可*」ページを参考にしてください。

Spring MVCでは通常、 1 つのWebアプリケーションに複数のDispatcherServletを登録し、サーブレットごとにBean定義を行うことができますが、 intra-mart Accel Platform では、DispatcherServletを拡張したサーブレットクラスが1つだけ登録されており、このサーブレットクラスを複数登録することはできません。
そのため、intra-mart Accel Platform ではすべてのBean定義はWebアプリケーション上で共有されます。
intra-mart Accel Platform では、あらかじめ下記ファイルにてBean定義が行われています。

- <Jugglingプロジェクト>/classes/META-INF/spring/applicationContext-im_tgfw_common.xml
- <Jugglingプロジェクト>/classes/META-INF/spring/applicationContext-im_tgfw_web.xml
- <Jugglingプロジェクト>/classes/META-INF/spring/SpringMVCServlet-servlet.xml
- <Jugglingプロジェクト>/classes/META-INF/spring/applicationContext-im_tgfw_mybatis3.xml

これらのBean定義は、intra-mart Accel Platform のテナント上で共有されるため、セットアップガイドに記載のない内容以外は基本変更しないようにしてください。
以下、intra-mart Accel Platform へのSpring MVCの組込み内容と上記Bean定義ファイルの初期値について示します。

web.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        metadata-complete="false"
        version="3.0"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

 : (省略)

 <!-- Web アプリケーション起動時に、クラスパス上の/META-INF/spring/内のapplicationContextで始まるBean定義ファイルをロード させる設定-->
 <context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>classpath*:/META-INF/spring/applicationContext*.xml</param-value>
 </context-param>
 <listener>
   <listener-class>jp.co.intra_mart.framework.extension.spring.web.servlet.SpringServletContextListener</listener-class>
 </listener>

 : (省略)

 <!-- 「ルーティング機構」用にDispatcherServletを拡張したサーブレットを登録 -->
 <servlet>
   <servlet-name>SpringMVCServlet</servlet-name>
   <servlet-class>jp.co.intra_mart.framework.extension.spring.web.servlet.IntramartDispatcherServlet</servlet-class>
   <init-param>
     <param-name>contextConfigLocation</param-name>
     <param-value>classpath:/META-INF/spring/SpringMVCServlet-servlet.xml</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
   <multipart-config/>
 </servlet>

 : (省略)

</web-app>
```

applicationContext-im_tgfw_common.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx https://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- a bean which holds the root application context -->
    <bean class="jp.co.intra_mart.system.extension.spring.context.ApplicationContextHolder" />

    <!-- message source which loads from 'WEB-INF/conf/message/**/*.properties' -->
    <bean id="messageSource" class="jp.co.intra_mart.framework.extension.spring.message.IntramartMessageSource" />

    <!-- bean validation (JSR-303 / JSR-349) -->
    <bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
        <property name="validationMessageSource" ref="messageSource" />
    </bean>

    <!-- transaction manager (JTA) -->
    <bean id="transactionManager" class="org.springframework.transaction.jta.JtaTransactionManager" />

    <!-- enable the configuration of transactional behavior based on annotations -->
    <tx:annotation-driven transaction-manager="transactionManager"/>

    <!-- intra-mart tenant datasource -->
    <bean id="dataSource" class="jp.co.intra_mart.framework.extension.spring.datasource.TenantDataSource" />

    <!-- intra-mart shared datasource -->
    <!--
        [Note] If you use the shared datasource, please uncomment/enable the following setting
            and set the bean's id and the connectId's value.
    -->
    <!--
    <bean id="sharedDataSource" class="jp.co.intra_mart.framework.extension.spring.datasource.SharedDataSource">
        <constructor-arg name="connectId" value="xxxxxxxxxxxxx" />
    </bean>
     -->

    <!-- property placeholder setting -->
    <context:property-placeholder location="classpath*:/META-INF/spring/*.properties" />

    <!-- dozer setting -->
    <bean class="com.github.dozermapper.spring.DozerBeanMapperFactoryBean">
        <property name="mappingFiles" value="classpath*:/META-INF/dozer/**/*-mapping.xml" />
    </bean>

    <!-- Exception Code Resolver. -->
    <bean id="exceptionCodeResolver"
        class="org.terasoluna.gfw.common.exception.SimpleMappingExceptionCodeResolver">
        <!-- Setting and Customization by project. -->
        <property name="exceptionMappings">
            <map>
                <entry key="ResourceNotFoundException" value="w.im.fw.0001" />
                <entry key="InvalidTransactionTokenException" value="w.im.fw.0004" />
                <entry key="InvalidSecureTokenException" value="w.im.fw.0005" />
                <entry key="BusinessException" value="w.im.fw.0002" />
            </map>
        </property>
        <property name="defaultExceptionCode" value="e.im.fw.0001" />
    </bean>

    <!-- Exception Logger. -->
    <bean id="exceptionLogger"
        class="org.terasoluna.gfw.common.exception.ExceptionLogger">
        <property name="exceptionCodeResolver" ref="exceptionCodeResolver" />
    </bean>

</beans>
```

applicationContext-im_tgfw_web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:util="http://www.springframework.org/schema/util"
   xmlns:context="http://www.springframework.org/schema/context"
xmlns:aop="http://www.springframework.org/schema/aop"
   xsi:schemaLocation="http://www.springframework.org/schema/aop https://www.springframework.org/schema/aop/spring-
aop.xsd
      http://www.springframework.org/schema/mvc https://www.springframework.org/schema/mvc/spring-mvc.xsd
      http://www.springframework.org/schema/beans https://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/util https://www.springframework.org/schema/util/spring-util.xsd
      http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-
context.xsd">

   <!-- locale resolver -->
   <bean id="localeResolver" class="jp.co.intra_mart.framework.extension.spring.web.servlet.i18n.AccountLocaleResolver"
/>

   <!-- SpringMVCRoute needs HandlerSelector bean. -->
   <bean id="handlerSelector" class="jp.co.intra_mart.system.router.spring.HandlerSelector" init-method="init" />

   <!-- view components -->
   <context:component-scan base-package="jp.co.intra_mart.framework.extension.spring.web.servlet.view" />

   <bean class="org.springframework.web.servlet.view.BeanNameViewResolver">
      <property name="order" value="0" />
   </bean>
   <!-- prefix for InternalResourceViewResolver -->
   <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/views/" />
      <property name="order" value="1" />
   </bean>

   <!-- multipart resolver -->
   <bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver" />

   <!-- support for annotation-driven MVC controllers -->
   <mvc:annotation-driven conversion-service="conversionService">
      <!-- transaction token -->
      <mvc:argument-resolvers>
         <bean class="org.terasoluna.gfw.web.token.transaction.TransactionTokenContextHandlerMethodArgumentResolver"
/>
      </mvc:argument-resolvers>
      <!-- jackson message converter -->
      <mvc:message-converters register-defaults="true">
         <bean class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">
            <property name="objectMapper">
               <bean class="jp.co.intra_mart.framework.extension.spring.http.converter.json.AccountDateObjectMapper" />
            </property>
         </bean>
      </mvc:message-converters>
   </mvc:annotation-driven>

   <bean id="conversionService"
       class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
      <property name="formatters">
         <set>
            <bean
class="jp.co.intra_mart.framework.extension.spring.format.datetime.AccountDateFormatAnnotationFormatterFactory"/>
            <bean
class="jp.co.intra_mart.framework.extension.spring.format.datetime.AccountDateTimeFormatAnnotationFormatterFactory"/>
            <bean
class="jp.co.intra_mart.framework.extension.spring.format.datetime.AccountTimeFormatAnnotationFormatterFactory"/>
         </set>
      </property>
   </bean>

   <!-- MVC interceptors -->
   <mvc:interceptors>
      <!-- transaction token -->
      <mvc:interceptor>
         <mvc:mapping path="/**" /> <!-- configure the path to specify the package of controllers. -->
         <bean class="org.terasoluna.gfw.web.token.transaction.TransactionTokenInterceptor">
```

```
            <constructor-arg value="10" />
        </bean>
    </mvc:interceptor>
    <!-- code list -->
    <mvc:interceptor>
        <mvc:mapping path="/**" /> <!-- configure the path to specify the package of controllers. -->
        <bean class="org.terasoluna.gfw.web.codelist.CodeListInterceptor">
            <property name="codeListIdPattern" value="CL_.+" />
        </bean>
    </mvc:interceptor>
</mvc:interceptors>

<!-- transaction token -->
<bean name="requestDataValueProcessor"
class="org.terasoluna.gfw.web.mvc.support.CompositeRequestDataValueProcessor">
    <constructor-arg>
        <util:list>
            <bean class="org.terasoluna.gfw.web.token.transaction.TransactionTokenRequestDataValueProcessor" />
        </util:list>
    </constructor-arg>
</bean>

<!-- secure token validator -->
<bean id="secureTokenValidator" class="jp.co.intra_mart.framework.extension.spring.web.csrf.SecureTokenValidator" />

<!-- Setting Exception Handling. -->
<!-- Exception Resolver. -->
<bean class="org.terasoluna.gfw.web.exception.SystemExceptionResolver">
    <property name="exceptionCodeResolver" ref="exceptionCodeResolver" />
    <!-- Setting and Customization by project. -->
    <property name="order" value="3" />
    <property name="exceptionMappings">
        <map>
            <entry key="ResourceNotFoundException" value="im_tgfw/common/error/resourceNotFoundError.jsp" />
            <entry key="BusinessException" value="im_tgfw/common/error/businessError.jsp" />
            <entry key="InvalidTransactionTokenException" value="im_tgfw/common/error/transactionTokenError.jsp" />
            <entry key="InvalidSecureTokenException" value="im_tgfw/common/error/secureTokenError.jsp" />
        </map>
    </property>
    <property name="statusCodes">
        <map>
            <entry key="im_tgfw/common/error/resourceNotFoundError" value="404" />
            <entry key="im_tgfw/common/error/businessError" value="200" />
            <entry key="im_tgfw/common/error/transactionTokenError" value="409" />
            <entry key="im_tgfw/common/error/secureTokenError" value="403" />
        </map>
    </property>
    <property name="excludedExceptions">
        <array>
            <value>org.springframework.web.util.NestedServletException</value>
        </array>
    </property>
    <property name="defaultErrorView" value="im_tgfw/common/error/systemError.jsp" />
    <property name="defaultStatusCode" value="500" />
</bean>
<!-- AOP. -->
<bean id="handlerExceptionResolverLoggingInterceptor"
    class="org.terasoluna.gfw.web.exception.HandlerExceptionResolverLoggingInterceptor">
    <property name="exceptionLogger" ref="exceptionLogger" />
</bean>
<aop:config>
    <aop:advisor advice-ref="handlerExceptionResolverLoggingInterceptor"
        pointcut="execution(* org.springframework.web.servlet.HandlerExceptionResolver.resolveException(..))" />
</aop:config>

</beans>
```

SpringMVCServlet-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-
context.xsd
        http://www.springframework.org/schema/mvc https://www.springframework.org/schema/mvc/spring-mvc.xsd">

</beans>
```

applicationContext-im_tgfw_mybatis3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mybatis="http://mybatis.org/schema/mybatis-spring"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans.xsd
    http://mybatis.org/schema/mybatis-spring https://mybatis.org/schema/mybatis-spring.xsd">

    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="configLocation" value="classpath:/META-INF/mybatis/mybatis-config.xml" />
    </bean>

    <!-- setting 'base-package' which is the package in which mapper interface is. -->
<!--
    <mybatis:scan base-package="xxxxxx.yyyyyy.zzzzzz.domain.repository" />
 -->

</beans>
```