Copyright © 2012 NTT DATA INTRAMART CORPORATION

目次

変更年月日	変更内容
2012-10-01	初版
2012-12-21	第2版 下記を追加・変更しました ■ 「*認証機能*」を追加 ■ 「*ショートカットアクセス機能*」を追加 ■ 「*画面へアクセスするための認可設定方法*」で、認可設定画面を利用して認可リソースを登録する方法を追加 ■ 「エンターキー押下時のフォームの動作」を変更 ■ 「PageBuilder で実装可能な画面レイアウトの種類」に注意を追加
2013-04-01	第3版 下記を追加・変更しました ■ 「*ユーザコンテキストの所属組織を切り替える* 」を追加 ■ *UI（スマートフォン開発ガイドライン）* の 　■ *実装例：登録画面を作る* に *最終結果* を追加 　■ *実装例：一覧画面を作る* に *最終結果* を追加 　■ *実装例：参照画面を作る* に *最終結果* を追加 　■ *推奨画面構成* の *処理リンク* を修正 ■ 「PageBuilder で実装可能な画面レイアウトの種類」 に HeadWithContainerThemeBuilder についての記述を追加
2013-07-01	第4版 下記を追加・変更しました ■ 「タイムゾーン 」の「日時データを統一のタイムゾーンで変換して保存する」に、TIMESTAMP 型についての制約を追加
2013-10-01	第5版 下記を追加・変更しました ■ 「*ショートカット拡張検証機能* 」を追加 ■ 「*画面へアクセスするための認可設定方法*」に認可リソースマッパーの設定方法を追加 ■ 「*Cache サービス*」にサイズ計算に関する警告を追加
2014-01-01	第6版 下記を追加・変更しました ■ 「*Storage*」の 　■ 「*ファイルアップロード*」を修正 　■ 「*ファイルダウンロード*」を修正 ■ 「*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。
2014-04-01	第7版 下記を追加・変更しました ■ アプリケーションサーバ側のルートを示すパスの表記を「%CONTEXT_PATH%」に統一しました。
2014-08-01	第8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 ■ 「*データベース*」の 「*シェアードデータベースの利用*」を修正
2014-12-01	第9版 下記を追加・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 ■ 「*サーバサイド単体テスト*」の利用方法を追加 ■ 「*CSRF対策*」にCSRF対策の設定方法についての記述を追加 ■ 「*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。

変更年月日	変更内容
2015-04-01	第10版 下記を追加・変更しました ■ 「*データベース*」の 「*UserTransactionオブジェクトを使用したトランザクションの完全手動制御*」を修正
2015-08-01	第11版 下記を追加・変更しました ■ 「*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12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 ■ 「*UI（スマートフォン開発ガイドライン）*」を jQuery Mobile 1.4.5 ベースで作成したガイドに変更
2016-04-01	第13版 下記を追加・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。」）。
2017-12-01	第14版 下記を追加・変更しました ■ 「*はじめに*」に「*推奨開発方法について*」を追加
2018-08-01	第15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
2019-12-01	第16版 下記を追加・変更しました ■ 以下の設定ファイルの文書型定義に記述されているURLスキームを http から https に変更 ■ 「*データベース*」の 「*シェアードデータベースの利用*」 の jdbc.dicon, s2jdbc.dicon ■ 「*CSRF対策*」 の 「*app.diconファイル*」 の app.dicon ■ 「*基本的な画面の作り方*」 の 「*app.diconを編集する*」 の app.dicon ■ 「*基本的な画面の作り方*」 の 「*convention.diconを編集する*」 の convention.dicon ■ 「*基本（intra-mart Accel Platform での初めてのプログラミング）*」 の 「*ステップ2：convention.diconの修正*」 の convention.dicon
2020-08-01	第17版 下記を追加・変更しました ■ 「*ConfigurationLoader の設定ファイル読み込み*」 のサンプルプログラムを修正
2024-04-01	第18版 下記を追加・変更しました ■ 「*Storage*」の ■ 「*ファイルアップロード*」を修正 ■ 「*ファイルダウンロード*」を修正
2024-06-28	第19版　下記を追加・変更しました ■ 「タイムゾーン」の説明を「intra-mart Accel Platform タイムゾーン仕様書」へ転記 ■ 「日付と時刻の形式」の説明を「intra-mart Accel Platform タイムゾーン仕様書」へ転記 ■ 「*ユーザのタイムゾーンや日付と時刻の形式に対応するためのサンプルプログラム*」の説明を「intra-mart Accel Platform タイムゾーン仕様書」の記載にあわせて変更

変更年月日	変更内容
2024-10-01	第20版 下記を変更しました ■ 「*基本的な画面の作り方*」 の ■ 「*JSP ファイルを用意する* 」を修正

はじめに

本書の目的

本書ではアプリケーションを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。

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 の各機能の利用したい開発者
- 旧バージョンから移行したプログラムを intra-mart Accel Platform で動作させたい開発者

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。

- *旧バージョンで作成したプログラムの実行*

  旧バージョンから移行したプログラムを intra-mart Accel Platform で動作するための方法を説明します。

推奨開発方法について

2016/9/26にて、Seasar2 のサポートが終了となることが発表されました。
intra-mart Accel Platform では、現行通り、SAStruts+S2JDBC （Seasar2）を利用する事は可能ですが、新規にJava開発を行う場合は TERASOLUNA Server Framework for Java (5.x) での開発を推奨します。
TERASOLUNA Server Framework for Java (5.x) での開発方法は TERASOLUNA Server Framework for Java (5.x) プログラミングガイド を参照してください。

【参考】: [FAQ]「Seasar2 のサポートが、2016/9/26 にて終了するアナウンスがありました、intra-mart製品の方針について教えてください。」

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 SAStruts フレームワークを利用した Hello World を作成することによって intra-mart Accel Platform での SAStruts フレームワーク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 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。
  ベースモジュールに SAStruts 開発フレームワーク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。

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。
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：convention.diconの修正

convention.dicon でルートパッケージの設定をします。
SAStruts フレームワークでは、ルートパッケージをパッケージ名の先頭に指定して、action などのパッケージを作成して必要なファイルを格納し、URL を指定します。
以下にAction クラスを作成する際に、パッケージ名と URL の例を示します。
ex: Action クラス名：ルートパッケージ.action.foo.BarActionとした場合、URL は *http://ホスト名/war名/foo/bar/* となる。
<% war_home %>/WEB-INF/classes/convention.dicon を src/main/resource 配下に配置し、ルートパッケージを決めて、convention.dicon に以下の様に記述します。
なお、下記の例ではルートパッケージを”sample.sastruts”とします。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC "-//SEASAR//DTD S2Container 2.4//EN"
 "https://www.seasar.org/dtd/components24.dtd">
<components>
 <component class="jp.co.intra_mart.framework.extension.seasar.convention.IMNamingConventionImpl">
  <initMethod name="addRootPackageName">
   <arg>"org.seasar.framework.container.warmdeploy"</arg>
  </initMethod>
  <initMethod name="addRootPackageName">
   <arg>"tutorial"</arg>
  </initMethod>
  <initMethod name="addRootPackageName">
   <arg>"sample.sastruts"</arg>
  </initMethod>
 </component>
 <component class="org.seasar.framework.convention.impl.PersistenceConventionImpl"/>
</components>
```

> **コラム**
>
> convention.diconなどの各種設定を行うdiconファイルの仕様に関しては、seasar2のホームページを参照してください。

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/view/sample/sa/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 pageEncoding="UTF-8"%>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui"%>
<%@ taglib prefix="s" uri="http://sastruts.seasar.org" %>
<%@ taglib prefix="f" uri="http://sastruts.seasar.org/functions" %>

<!-- 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 (SAStruts)</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) -->
 <s:form action="/sample/hello/output/" styleId="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"></imui:button>
 </s: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/WEB-INF/view/sample/sa/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="s" uri="http://sastruts.seasar.org" %>
<%@ taglib prefix="f" uri="http://sastruts.seasar.org/functions" %>

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

<!-- 画面上に表示されるタイトル -->
<div class="imui-title">
  <h1>Hello, World (SAStruts)</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、HelloActionのindexメソッドが呼び出される。 -->
        <a href=${f:url("/sample/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(helloDto.outputString)}" 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。
> - SAStruts+S2JDBCフレームワークにおいて、JSPなどのView用のファイルを 置くディレクトリを/WEB-INF/web.xmlにてsastruts.VIEW_PREFIXで指定します。 sastruts.VIEW_PREFIXのデフォルト値は/WEB-INF/viewが指定してあります。本ガイドではこのsastruts.VIEW_PREFIXの値がデフォルト値である /WEB-INF/viewであることを前提に説明をしています。

> **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.sastruts.form.sample、クラス名は HelloForm とします。
   以下に HelloForm に記述するソースを示します。

```
package sample.sastruts.form.sample;

/**
 * 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 {

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

}
```

> **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：Dtoクラスの作成

出力情報を保持する Dto クラスを作成します。
ここでは、パッケージ名は sample.sastruts.dto.sample、クラス名は HelloDto とします。
以下に HelloDto に記述するソースを示します。

```
package sample.sastruts.dto.sample;

/**
 * 結果表示画面に渡すDTOクラス
 * @author intra-mart
 */
public class HelloDto {

    public String outputString;     // 結果を保持する文字列

}
```

> **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7：Action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 Action クラスを作成します。
ここでは、パッケージ名は sample.sastruts.action.sample、クラス名は HelloAction とします。
以下に HelloAction に記述するソースを示します。

```java
package sample.sastruts.action.sample;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;

import org.seasar.struts.annotation.ActionForm;
import org.seasar.struts.annotation.Execute;

import sample.sastruts.dto.sample.HelloDto;
import sample.sastruts.form.sample.HelloForm;

/**
 * Hello　Worldの処理を行うActionクラス
 * @author intra-mart
 */
public class HelloAction {

    @Resource
    @ActionForm
    public HelloForm helloForm;      // 入力情報を保持するFormクラス

    @Resource
    public HelloDto helloDto;       // 出力情報を保持するDTOクラス(publicにすること)

    /**
     * 入力ページのパスを返却します。
     * @return 入力ページのパス
     */
    @Execute(validator = false)
    public String index() {
        return "/sample/sa/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
     */
    @Execute(validator = false)
    public String output() {
        helloDto.outputString = "Hello, " + helloForm.name;     // 結果の情報を格納
        return "/sample/sa/hello/output.jsp";
    }
}
```

> **注意**
>
> 文字コードを UTF-8 にして保存してください。

これで 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、「メニュー設定」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 SA
URL	sample/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。

- SAStruts フレームワークを用いて、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。

認可

項目

画面にアクセス権限を設定するために

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

アクション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、このままでは実行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、誰でも画面を表示することができます。

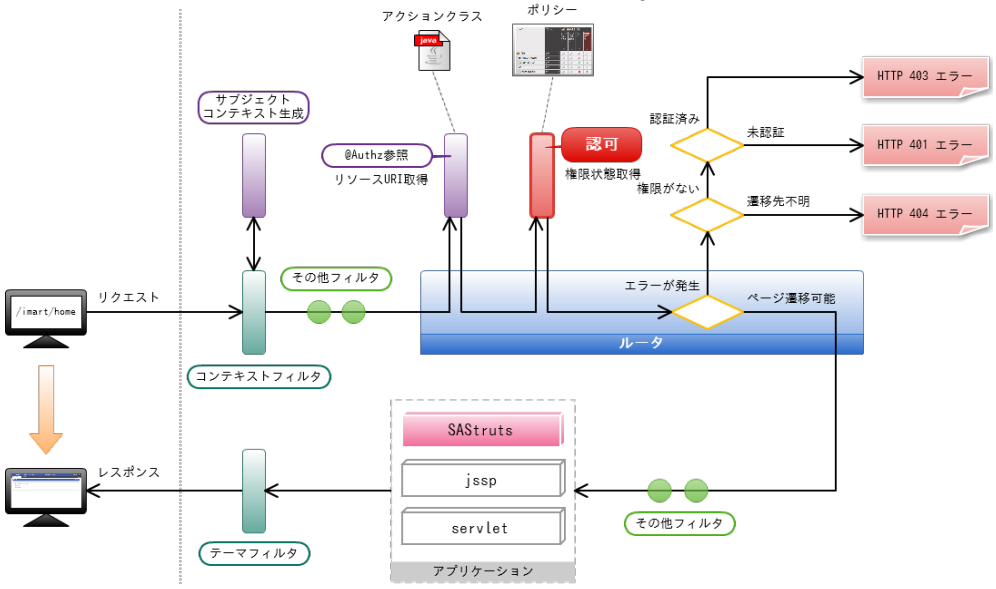
実際にシステムを運用する際は、アクセス権限を設定して特定のユーザのみに画面を表示させ、アクセスの制限をかける場合がほとんどです。
この章では、 intra-mart Accel Platform で用意されている認可機能を利用して、用意した画面を特定のユーザにのみ表示させる手順を説明します。

実行メソッドの詳細については http://sastruts.seasar.org/featureReference.html#ExecuteMethod 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することができます。

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

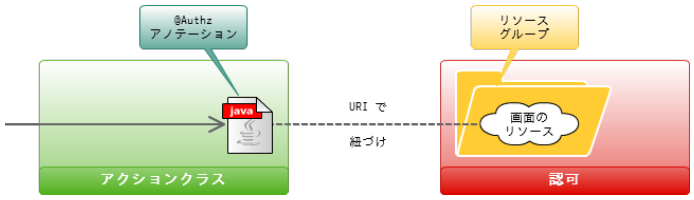
認可での 「ポリシー」 は、「許可」「禁止」 の部分を示します。
各サブジェクト・リソース・アクションの組み合わせ分、ポリシーを設定することができます。

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」 扱いとなります。

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

ルータでは実行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. 実行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：実行メソッドに認可を紐づける

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

```
@Execute
@Authz(uri = "service://sample/foo", action = "execute")
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。

> **コラム**
>
> 認可リソースマッパーを利用したい場合は mapperClass プロパティを設定してください。
> ただし、 uri プロパティと mapperClass プロパティの両方が指定されている場合は uri プロパティが有効になります。
>
> ```
> @Execute
> @Authz(mapperClass = SampleMapper.class)
>  public String index() {
>    return "index.jsp";
>  }
> ```
>
> 認可リソースマッパーに対するパラメータを指定したい場合は mapperParams プロパティに @AuthzMapperParam アノテーションを設定します。
>
> ```
> @Execute
> @Authz(mapperClass = SampleMapper.class, mapperParams={@AuthzMapperParam(key="paramKey",
> value="paramValue")})
>  public String index() {
>    return "index.jsp";
>  }
> ```

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。（クリック後、下図の赤枠のようになります）

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。

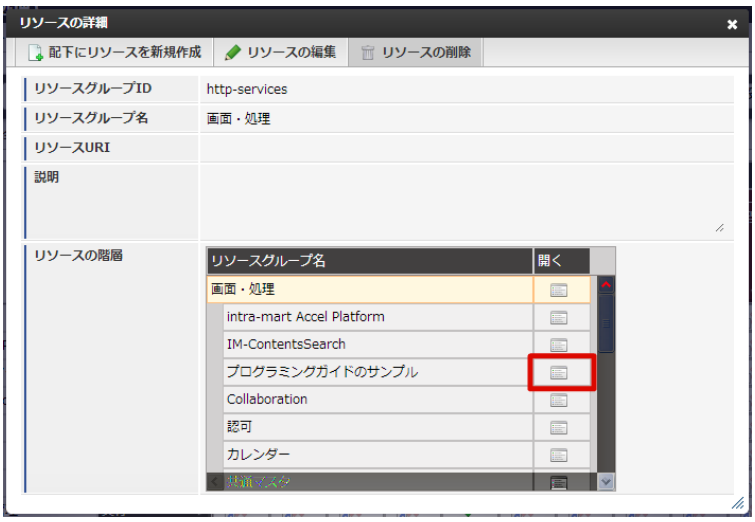
「リソースグループID」 のテキストボックスに 「guide-sample-service」 を入力します。
「リソースグループ名」 の最も上のテキストボックスに 「プログラミングガイドのサンプル」 を入力します。
「リソースURI」 と 「説明」 は未入力のままでかまいません。
「OK」 ボタンをクリックします。 これでリソースグループが登録されました。

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することができます。

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」 が選択された状態になります。



> **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することができます。

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 属性には実行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、パブリックストレージのルート直下に配置します。

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

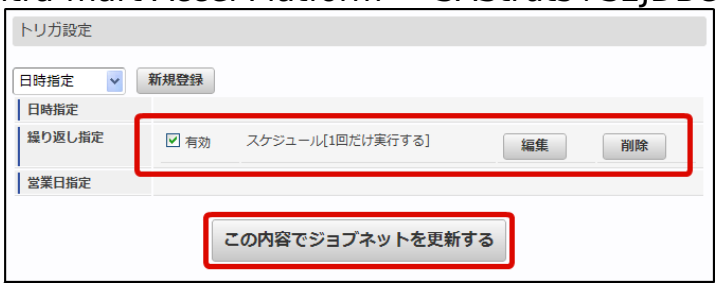
```
http://<HOST>:<PORT>/<CONTEXT_PATH>/login
```

「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。

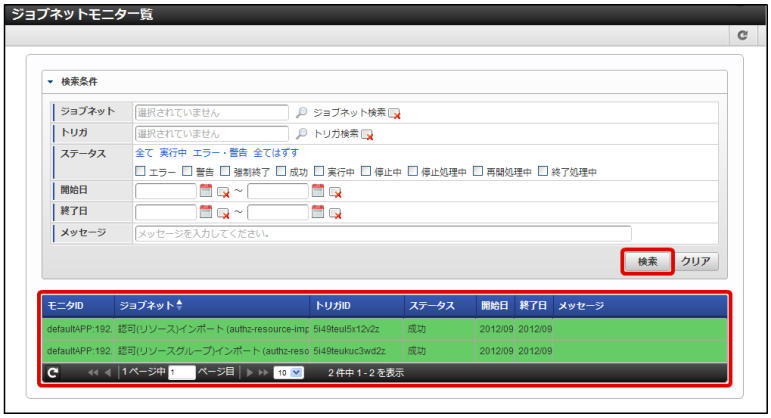
「トリガ設定」 のプルダウンから 「繰り返し指定」 を選択して 「新規登録」 ボタンをクリックします。
「1回だけ実行する」 を選択状態にして 「決定」 ボタンをクリックします。

「有効」 のチェックボックスをチェックして、 「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。（クリック後、下図の赤枠のようになります）
「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/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することができます。

> **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。

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することができます。

- 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することができます。

- 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();
```

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するようになっています。
上記の例は、以下のコードと同じ結果になります。

```
// 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、ジョブ実行環境で特定のユーザで処理を実行したり、Web実行環境でログインユーザとは別のユーザに切り替えて、処理を実行することができます。
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することが可能です。

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

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を変更したり、認証サーバと連携したりすることが可能です。

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

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日付や日時が入力・表示できることが必要です。

国際化機能の仕様

多言語については以下を参照してください。

- *多言語*

タイムゾーンと日付と時刻の形式については以下を参照してください。

- intra-mart Accel Platform タイムゾーン仕様書

国際化プログラミングのサンプル

国際化に対応した画面や処理を作成するには、上述の仕様に従って実装する必要があります。
そのための多言語対応に関するサンプルプログラムを以下で紹介します。

- *多言語化されたメッセージを取得する*

ユーザのタイムゾーンや日付と時刻の形式に対応するためのサンプルプログラムを以下で紹介します。

- *ユーザのタイムゾーンや日付と時刻の形式に対応するためのサンプルプログラム*

データベース

プログラミング方法

ここでは、S2JDBCを使ったデータベースプログラミングの方法を説明します。
S2JDBCは、Seasar2のO/R Mapperです。
S2JDBCの詳細については http://s2container.seasar.org/2.4/ja/s2jdbc_abstract.html を参照してください。

エンティティの作成

エンティティのソースコードは、S2JDBC-Gen を使ってデータベース上のテーブル定義から自動生成します。
S2JDBC-Genの使用方法については、 http://s2container.seasar.org/2.4/ja/s2jdbc_gen/setup.html

Antタスクの実行を参照してください。

サービスの作成

エンティティに対する操作を格納するクラスをサービスと呼びます。 ビジネスロジックは、エンティティかサービスに定義します。
サービスを作成する場合は、親クラスに org.seasar.extension.jdbc.service.S2AbstractService を指定します。

例えば、エンティティ「Sample」に対するサービスは、次のようになります。

```
public SampleService extends S2AbstractService<Sample> {
    ...
}
```

サービスクラスを利用したレコードの取得

アクションクラスでサービスを利用して Sample テーブルのレコードを全て取得する場合は、以下のように実装します。

```
public class SimpleAction {
    /**
     * サービス
     */
    @Resource
    protected SampleService sampleService;

    /**
     * Sampleテーブルのデータ配列
     */
    public List<Sample> records;

    @Execute(validator = false)
    public String index() {
        records = sampleService.findAll();
        return "index.jsp";
    }
}
```

SampleServiceは、@Resource アノテーションを付与することで自動バインディングされます。
そのためサービスクラスのインスタンス生成は行う必要がありません。

SampleService#findAll()メソッドを利用すると、Sampleテーブルの全データが取得できます。

指定した条件に一致するレコードの取得

指定した条件に一致したレコードを取得したい場合は、JdbcManagerを使用してデータベースにアクセスします。
例えば、Sample テーブルからidが「sample01」のレコードを取得する場合は以下のように実装します。

```
public SampleService extends S2AbstractService<Sample> {
    /**
     * Sampleテーブルからidが「sample01」のデータを取得します。
     */
    public Sample getSample01() {
        return jdbcManager.from(Sample.class).where("id = ?", "sample01").getSingleResult();
    }
}
```

シェアードデータベースの利用

シェアードデータベースに接続するには、以下の設定を行う必要があります。

1. 各 Web Application Server 毎にシェアードデータベースとして接続するデータソースの接続設定を行います。
2. 任意の接続IDを指定して intra-mart Accel Platform のシェアードデータベース設定を行います。
3. diconファイルにシェアードデータベースの設定を行います。
4. シェアードデータベース接続用のサービスを作成します。

> **コラム**
>
> ここではS2JDBCの設定方法のみ説明します。
> データソースの設定方法およびシェアードデータベース設定方法については、セットアップガイドを参照してください。

例えば、接続ID「sample」として設定されているシェアードデータベースに接続する場合、以下のように設定・実装します。

1. 「jdbc.dicon」をコピーして、「tenant-jdbc.dicon」を作ります。
2. 「jdbc.dicon」をコピーして、「shared-jdbc.dicon」を作り、以下のように編集することでシェアードデータベースを設定します。

   編集前

   ```
   ...
   <!-- from intra-mart -->
   <component  name="dataSource"
      class="jp.co.intra_mart.framework.extension.seasar.util.AutoDetectedDataSource">
   </component>
   ...
   ```

   編集後

   ```
   ...
   <!-- from intra-mart -->
   <component  name="sharedDataSource" class="jp.co.intra_mart.framework.extension.seasar.util.SharedDataSource">
      <arg>"sample"</arg><!-- 環境に応じた接続IDを指定 -->
   </component>
   ...
   ```

   argには、接続先シェアードデータベースの接続IDを指定します。環境に応じて変更してください。

3. 「jdbc.dicon」を以下のように編集し、「tenant-jdbc.dicon」と「shared-jdbc.dicon」をincludeするだけにします。

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE components PUBLIC "-//SEASAR//DTD S2Container 2.4//EN"
      "https://www.seasar.org/dtd/components24.dtd">
   <components namespace="jdbc">
      <include path="tenant-jdbc.dicon"/>
      <include path="shared-jdbc.dicon"/>
   </components>
   ```

4. 「s2jdbc.dicon」をコピーして、「s2tenant-jdbc.dicon」を作り、以下のように編集します。

   編集前

   ```
   ...
   <include path="jdbc.dicon"/>
   <include path="s2jdbc-internal.dicon"/>
   <component name="jdbcManager" class="org.seasar.extension.jdbc.manager.JdbcManagerImpl">
      <property name="maxRows">0</property>
   ...
   ```

   編集後

   ```
   ...
   <include path="tenant-jdbc.dicon"/><!-- この行を編集-->
   <include path="s2jdbc-internal.dicon"/>
   <component name="jdbcManager" class="org.seasar.extension.jdbc.manager.JdbcManagerImpl">
      <property name="maxRows">0</property>
   ...
   ```

5. 「s2jdbc.dicon」をコピーして、「s2shared-jdbc.dicon」を作り、以下のように編集することでシェアードデータベース用のJdbcManagerを設定します。

   編集前

```
...
<include path="jdbc.dicon"/>
<include path="s2jdbc-internal.dicon"/>
<component name="jdbcManager" class="org.seasar.extension.jdbc.manager.JdbcManagerImpl">
  <property name="maxRows">0</property>
...
```

編集後

```
...
<include path="shared-jdbc.dicon"/><!-- この行を編集-->
<include path="s2jdbc-internal.dicon"/>
<component name="sharedJdbcManager" class="org.seasar.extension.jdbc.manager.JdbcManagerImpl"><!-- この行を編集 --
>
  <property name="maxRows">0</property>
...
```

6. 「s2jdbc.dicon」を以下のように編集し、「s2tenant-jdbc.dicon」と「s2shared-jdbc.dicon」をincludeするだけにします。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC "-//SEASAR//DTD S2Container 2.4//EN"
  "https://www.seasar.org/dtd/components24.dtd">
<components>
  <include path="s2tenant-jdbc.dicon"/>
  <include path="s2shared-jdbc.dicon"/>
</components>
```

7. 「app.dicon」に、「s2jdbc.dicon」がincludeされていることを確認します。includeされていなければincludeします。

8. シェアードデータベース接続用のサービスクラスを作成し、@ResourceでJdbcManagerを指定します。

```
public abstract class AbstractSharedDatabaseService<T> extends S2AbstractService<T> {
  @Resource(name="sharedJdbcManager")
  public void setJdbcManager(JdbcManager jdbcManager) {
    this.jdbcManager = jdbcManager;
  }
}
```

> **コラム**
>
> 複数のシェアードデータベースを利用するには、「jdbc.dicon」と「s2jdbc.dicon」をシェアードデータベース毎にコピーして設定する必要があります。

トランザクション

トランザクション制御の設定は customizer.dicon で行っています。 intra-mart Accel Platform 標準のトランザクション制御の設定では、アクション、ロジック、サービスのメソッドに TxAttributeCustomizer が設定されております。
この設定により、メソッドの開始前にトランザクションが開始されていなければ開始され、実行メソッドが正常に終了した場合はコミットされ、例外がスローされた場合にはロールバックされます。

UserTransaction#setRollbackOnly()を利用したロールバック

トランザクションをロールバックしたいが例外をスローしたくないというケースがよくがあります。
このような場合は、UserTransaction#setRollbackOnly()を利用することで、例外をスローせずトランザクションをロールバックさせることができます。

```
public SampleAction {
   @Resource
   @ActionForm
   public SampleForm sampleForm;

   @Resource
   protected SampleService sampleService;

   @Resource
   protected UserTransaction userTransaction;

   /**
    * フォームの入力内容を登録します。
    */
   @Execute(validator = true)
   public String add() {
      Sample entity = new Sample();
      // フォームの入力内容をエンティティにコピーします。
      BeanUtil.copyProperties(sampleForm, entity);
      try {
         // 登録処理
         sampleService.insert(entity);
      } catch (Exception e) {
         // トランザクションをロールバック
         try {
            userTransaction.setRollbackOnly();
         } catch (Exception e) {
            throw  new  SystemRuntimeException(e);
         }
      }
      return "index.jsp";
   }

}
```

UserTransactionオブジェクトを使用したトランザクションの完全手動制御

トランザクションの開始、コミット、ロールバックは、UserTransaction オブジェクトを使用して手動で制御することができます。
トランザクションを手動で制御する場合は、TransactionAttributeType.NOT_SUPPORTEDを利用して実行メソッドをトランザクション対象外にした上で、UserTransaction オブジェクトを使用したトランザクション制御を行います。

```
public SampleAction {
    @Resource
    @ActionForm
    public SampleForm sampleForm;

    @Resource
    protected SampleService sampleService;

    @Resource
    protected UserTransaction userTransaction;

    /**
     * フォームの入力内容を登録します。
     */
    @Execute(validator = true)
    @TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
    public String add() {
        Sample entity = new Sample();
        // フォームの入力内容をエンティティにコピーします。
        BeanUtil.copyProperties(sampleForm, entity);

        // トランザクションの開始
        userTransaction.begin();
        try {
            // 登録処理
            sampleService.insert(entity);
        } catch (Exception e) {
            // finally 句でトランザクションをロールバックさせるため、ロールバックを予約しておく
            userTransaction.setRollbackOnly();
        } finally {
            if (userTransaction.getStatus() == Status.STATUS_ACTIVE) {
                // コミット
                userTransaction.commit();
                return "index.jsp";
            } else {
                // ロールバック
                userTransaction.rollback();
                return "error.jsp";
            }
        }
    }
}
```

ログ

項目

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SAStrutsフレームワーク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.action.AddAction
    // ロガー名：tutorial.action.AddAction
    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。引数に渡す文字列をロガーの名前として扱います。
引数に何も渡さない場合は、呼び出したクラス名（FQDN）をロガーの名前として扱います。

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.action.AddAction
    // ロガー名：tutorial.action.AddAction
    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)を実行した時のログ出力

```
[DEBUG] t.a.AddAction - arguments=[1, 2]
```

ログレベルtraceでadd(1, 2)を実行した時のログ出力

```
[DEBUG] t.a.AddAction - arguments=[1, 2]
[TRACE] t.a.AddAction - 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）を利用すると、ログ設定ファイルのレイアウト設定において、独自に定義したkeyで保存した情報をログに出力できます。
MDC APIを利用すると、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.action.AddAction
    // ロガー名：tutorial.action.AddAction
    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>の内容を以下の通りに変更し、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.a.AddAction - add arguments=[1, 2]
[TRACE] t.a.AddAction - add result=3
```

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 サイトを構築することができます。

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画面を作ることが可能になります。
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/

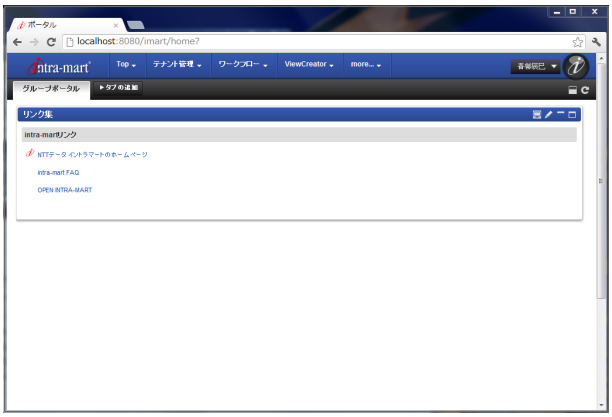
クライアントタイプとテーマ

項目

クライアントタイプとスマートフォン版テーマ

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

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することができるため、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 role="main" class="ui-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 role="main" class="ui-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することができます。

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。

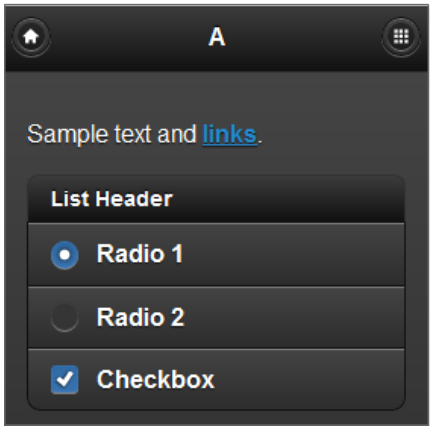
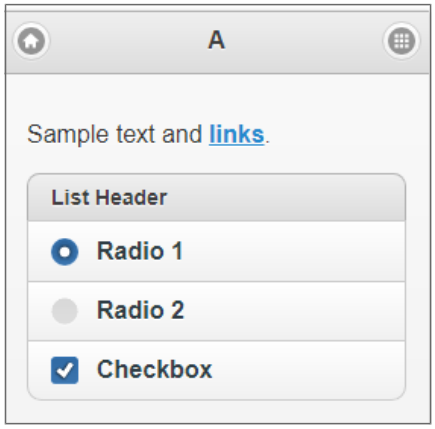
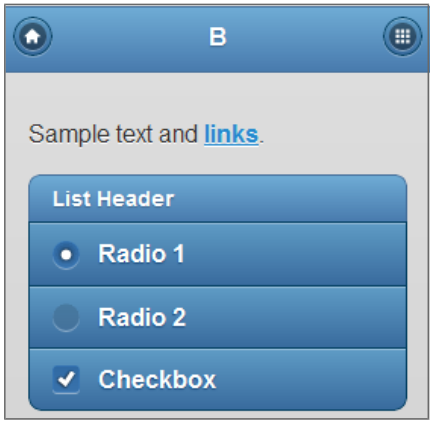
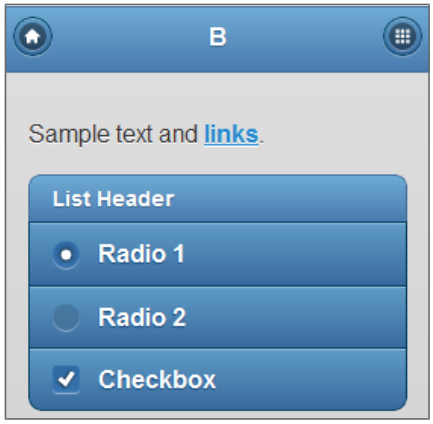
- デフォルトテーマが c から a 変更になったため、data-theme を指定していない画面は a（黒色）のテーマが適用されるようになった
- フラットデザインになったため、ボーダーやグラデーションがほとんどなくなった

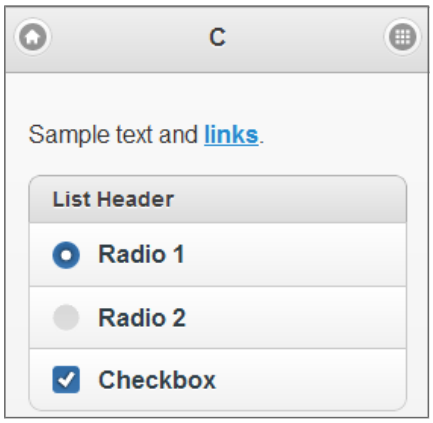
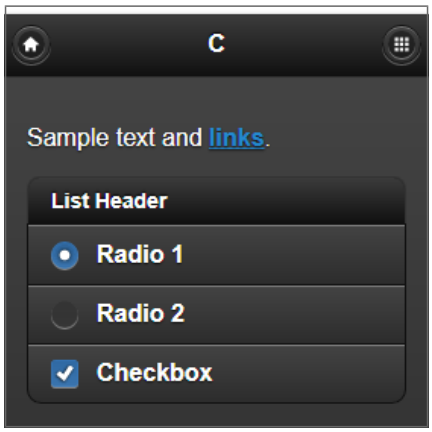
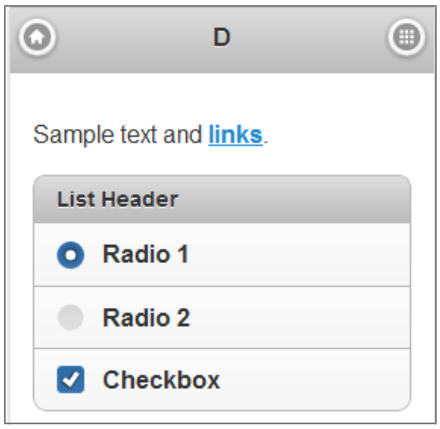
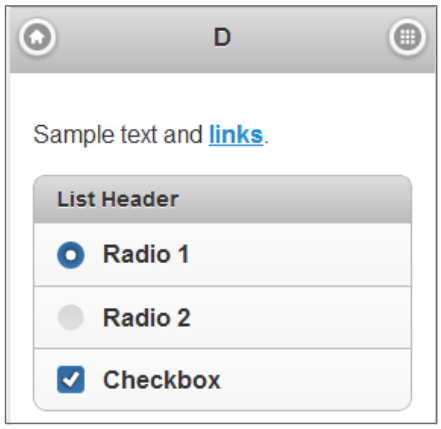
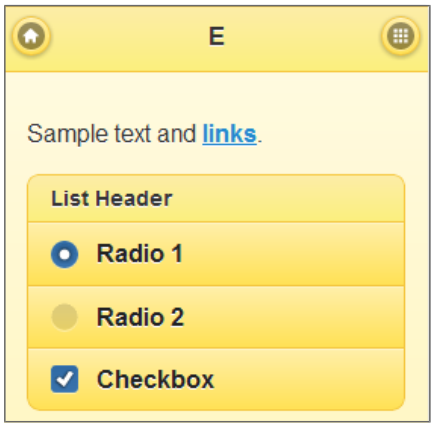
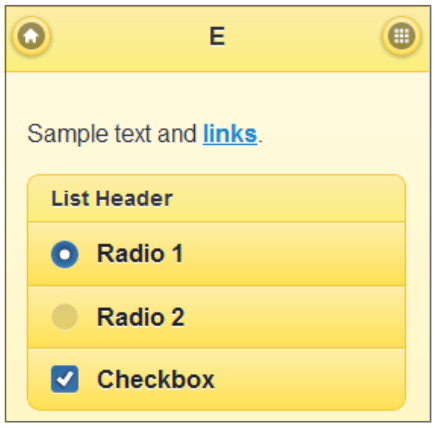
> **コラム**
> その他の変更点や変更点の詳細は http://jquerymobile.com/upgrade-guide/1.4 を参照してください。

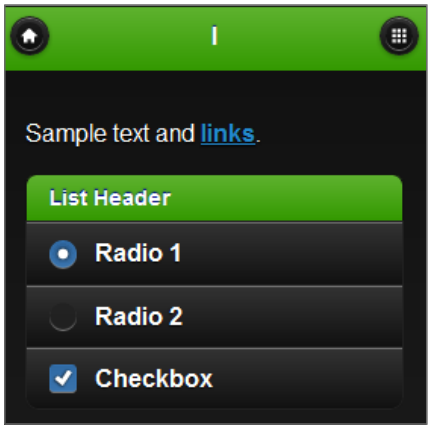
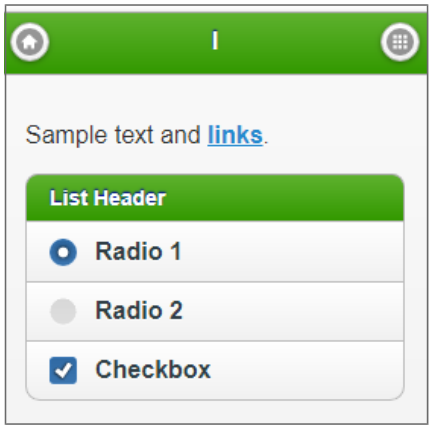
> **コラム**
> 2018 Summer(Tiffany) より標準テーマ白が追加されました。既存の標準テーマ黒を利用している場合とスウォッチの色が異なります。
> 各テーマのスウォッチについては下記を参照してください。

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」を使用することができます。

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		
B			

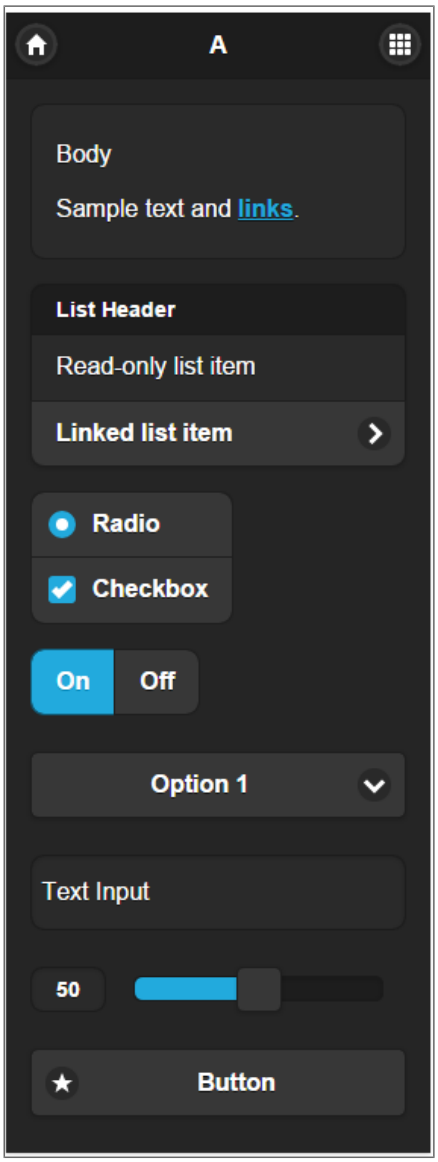
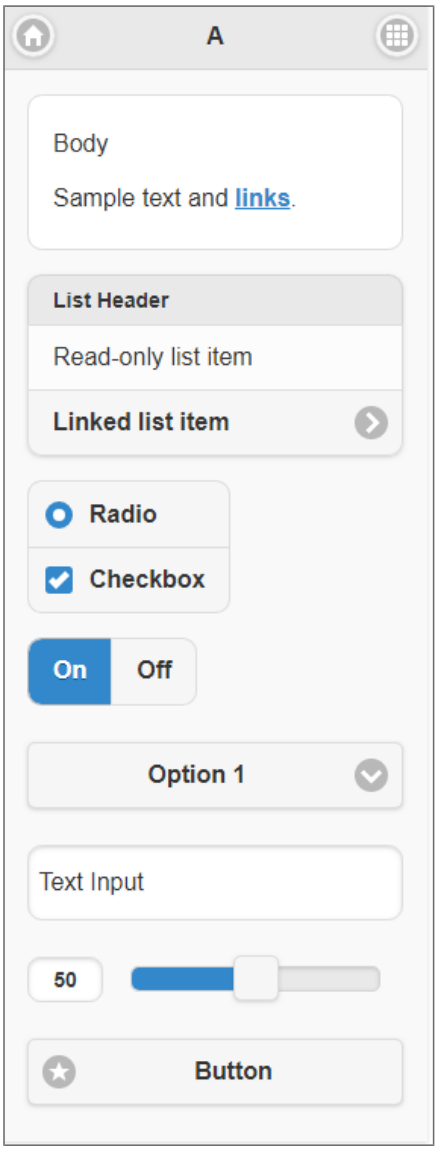
設定	説明	黒テーマイメージ	白テーマイメージ
C	jQueryMobile標準色		
D			
E			

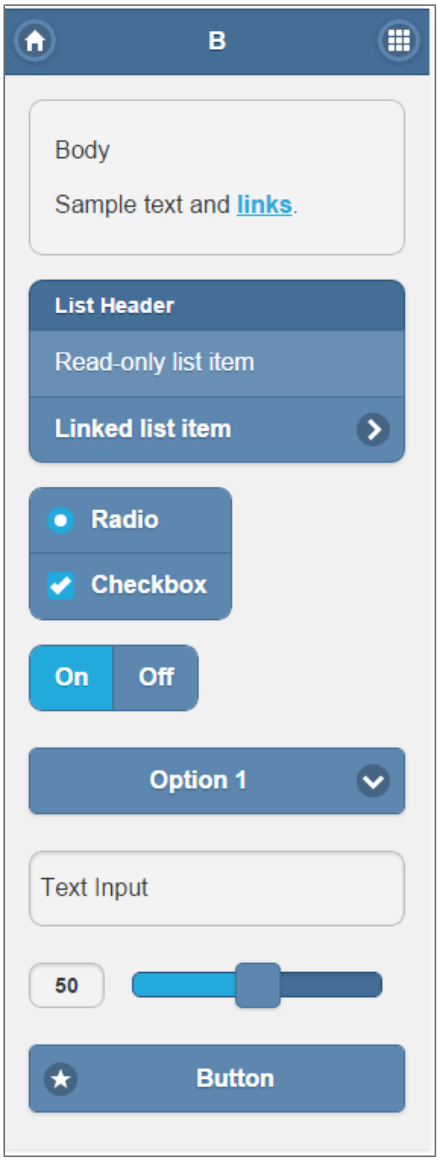
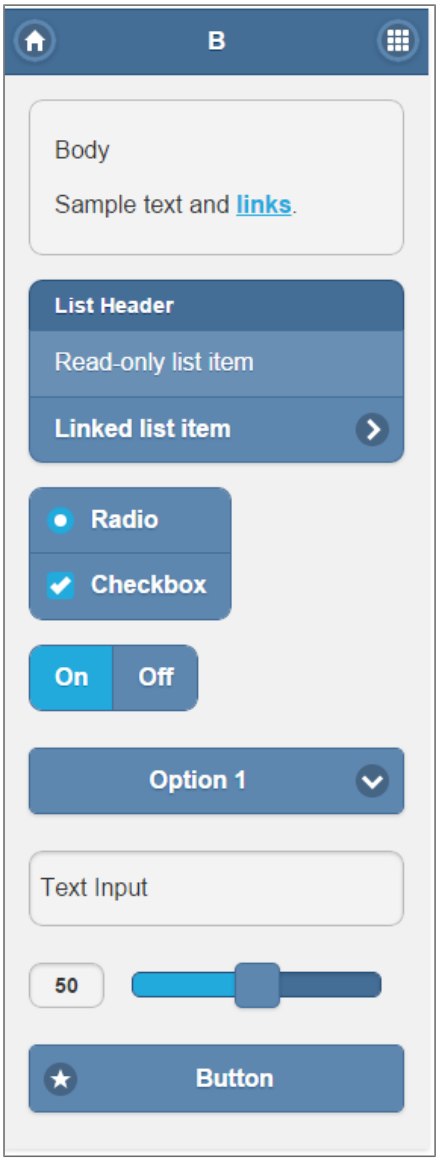
設定	説明	黒テーマイメージ	白テーマイメージ
F	追加スウォッチ		
I	追加スウォッチ		

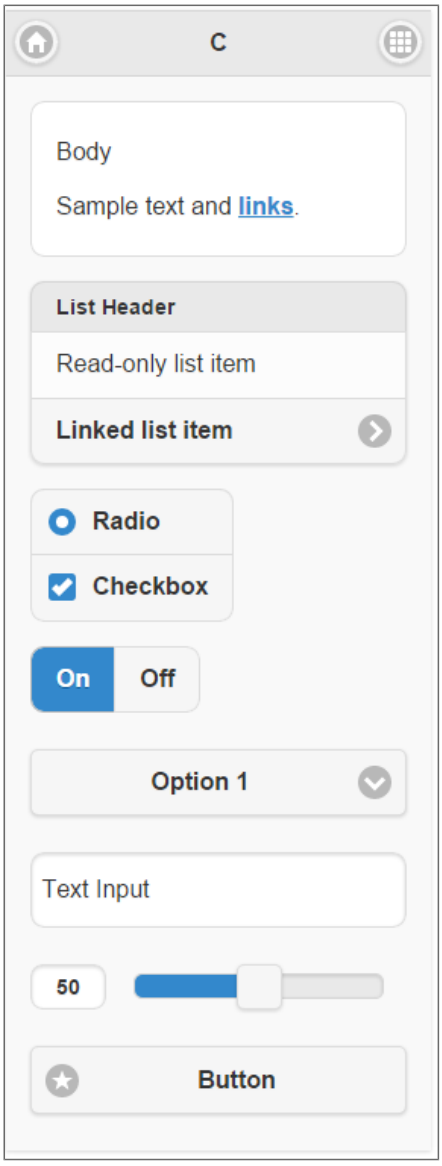
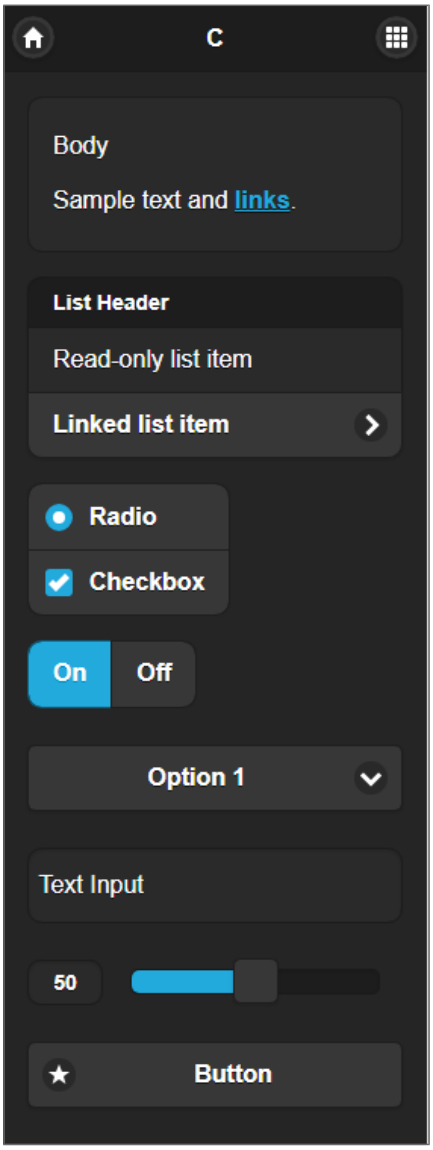
jQuery Mobile 1.4.5

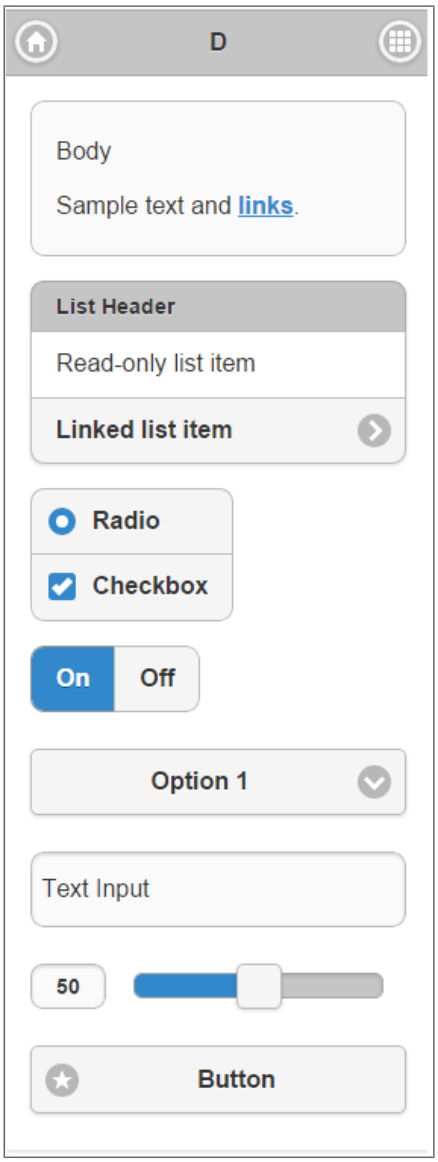
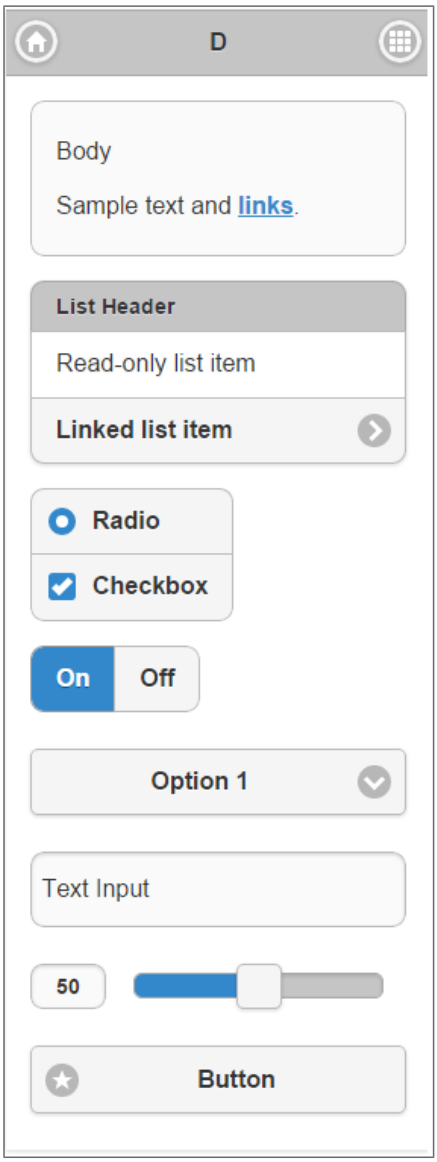
intra-mart Accel Platform では、jQuery Mobileデフォルトスウォッチの「A」～「E」のほかに、拡張スウォッチ「F」「I」を使用することができます。
「A」が jQueryMobile標準色へ変更になりました。

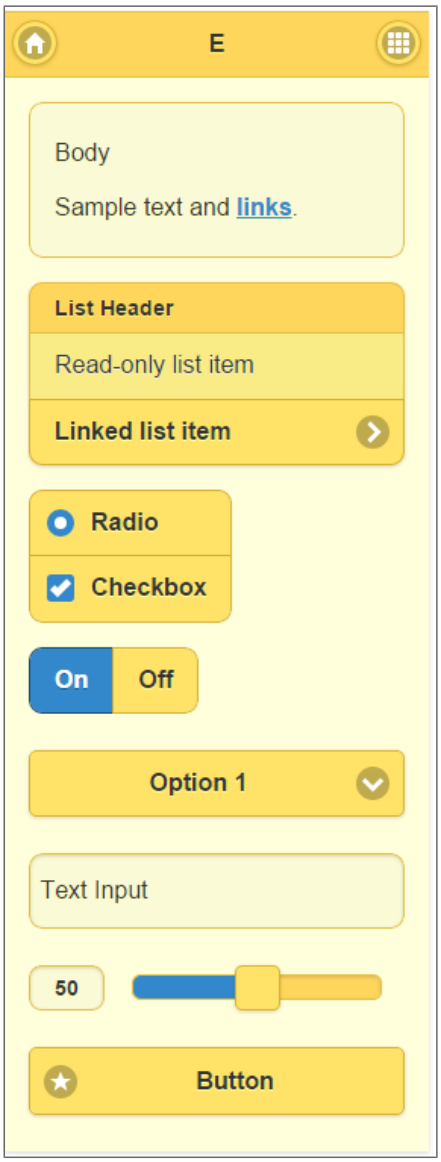
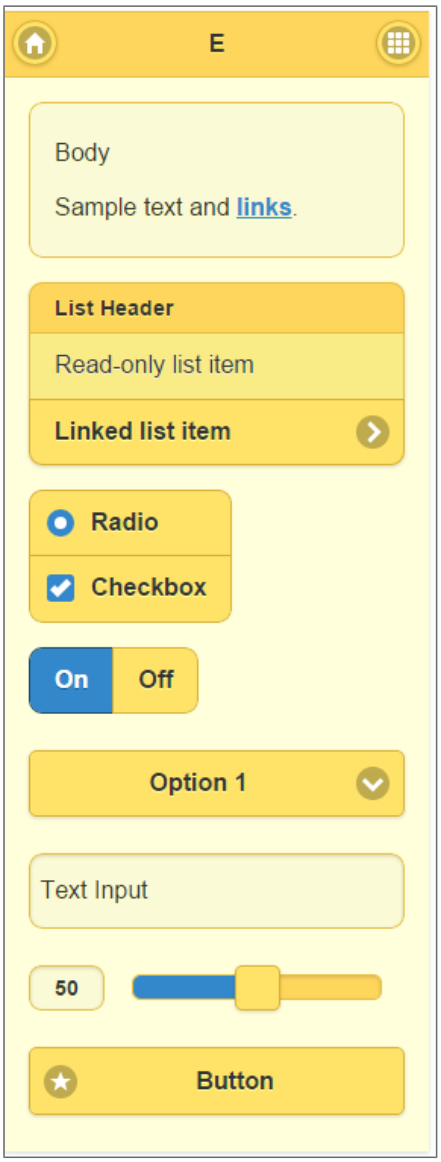
設定	説明	黒テーマイメージ	白テーマイメージ

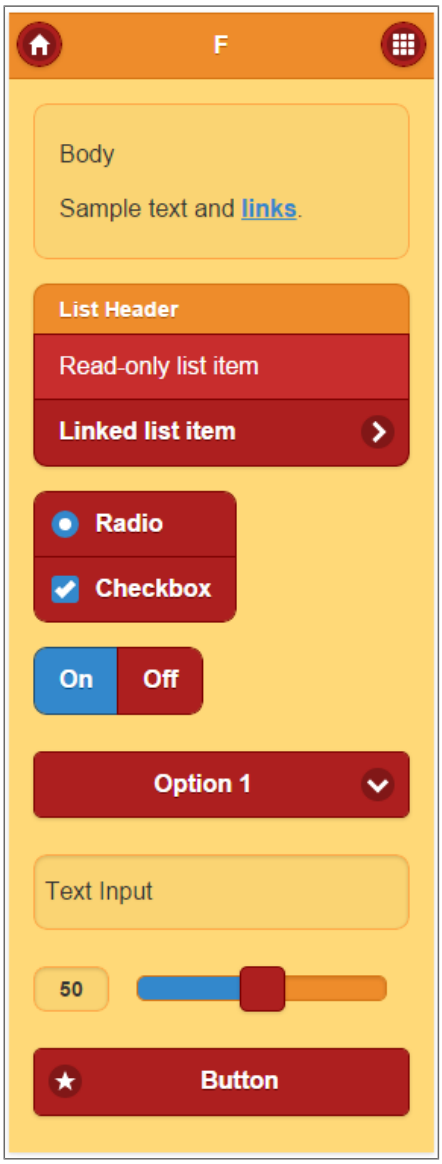
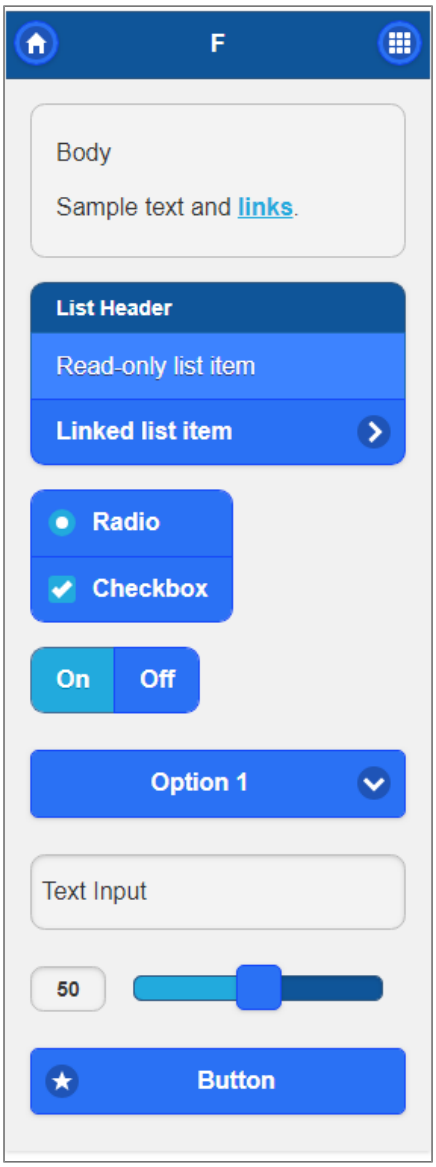
設定	説明	黒テーマイメージ	白テーマイメージ
A	jQueryMobile、intra-mart Accel Platform 標準色		

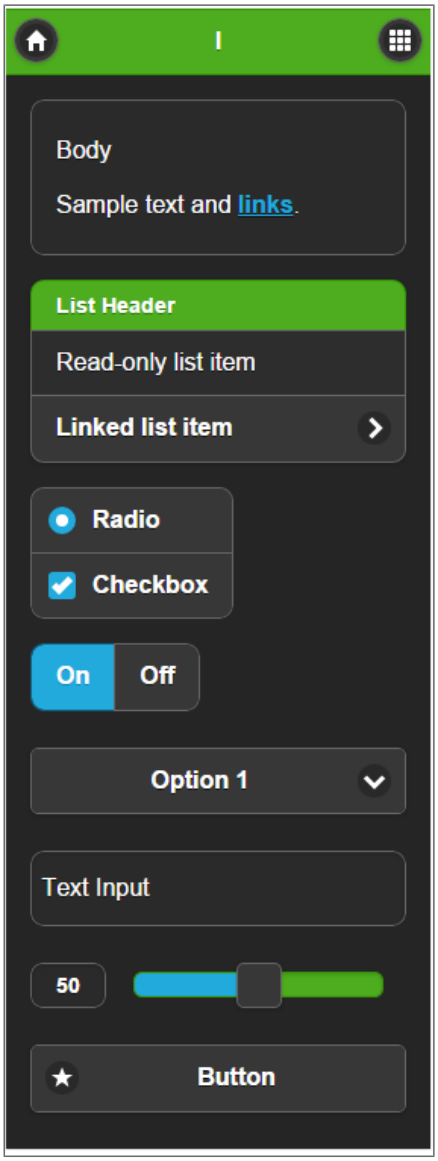
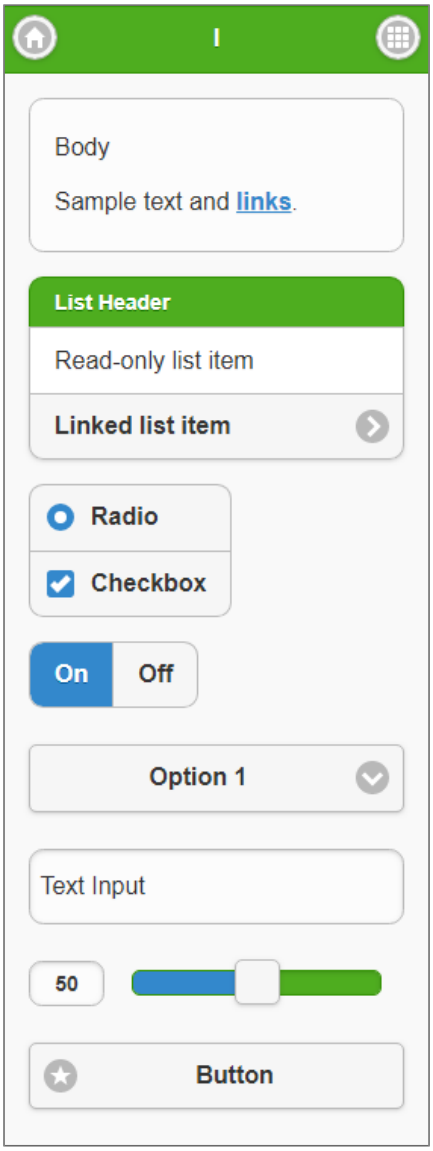
設定	説明	黒テーマイメージ	白テーマイメージ
B			

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

- Actionクラスを作成します。 パッケージ名はsample.sastruts.action.samplespとし、クラス名はHelloActionとします。

```
package sample.sastruts.action.samplesp;
import org.seasar.struts.annotation.Execute;

public class HelloAction {
    /**
     * 入力ページのパスを返却します。
     * @return 入力ページのパス
     */
    @Execute(validator = false)
    public String index() {
        return "/sample/imsp/hello/hello_mfw.jsp";
    }
}
```

JSPファイルを用意する

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

- hello_mfw.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 class="ui-content" role="main">
    <p>Hello IM-Mobile Framework!</p>
  </div>
  <div data-role="footer">
    <h3>Footer</h3>
  </div>
</div>
```

ライブラリ群の設定用ファイルを用意する

以下のファイルを作成し、%CONTEXT_PATH%/webapps/imart/WEB-INF/conf/theme-full-theme-path-configフォルダに保存します。

- im_mobile_fw_test.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 client-type="sp" libraries-version="iap-8.0.11">/samplesp/hello</path>
</theme-full-theme-path-config>
```

> **注意**
>
> この設定を行わない場合は、jQuery Mobile 1.3.0 が読み込まれます。

app.diconを編集する

- %CONTEXT_PATH%/webapps/imart/WEB-INF/classes/app.dicon を編集します。
  s2jdbc.dicon のコメントアウトをはずして、include を有効にします。

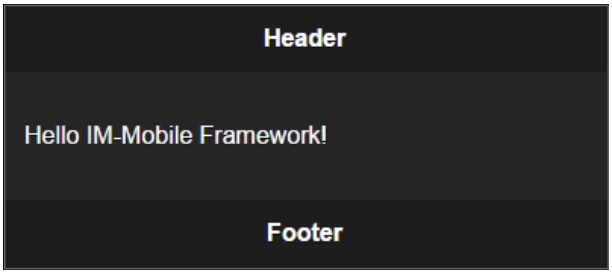
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC "-//SEASAR//DTD S2Container 2.4//EN"
  "https://www.seasar.org/dtd/components24.dtd">
<components>
  <include path="convention.dicon"/>
  <include path="aop.dicon"/>
  <include path="j2ee.dicon"/>
  <include path="s2jdbc.dicon"/>
  <component name="actionMessagesThrowsInterceptor"
class="org.seasar.struts.interceptor.ActionMessagesThrowsInterceptor"/>
</components>
```

convention.diconを編集する

- %CONTEXT_PATH%/webapps/imart/WEB-INF/classes/convention.dicon を編集します。

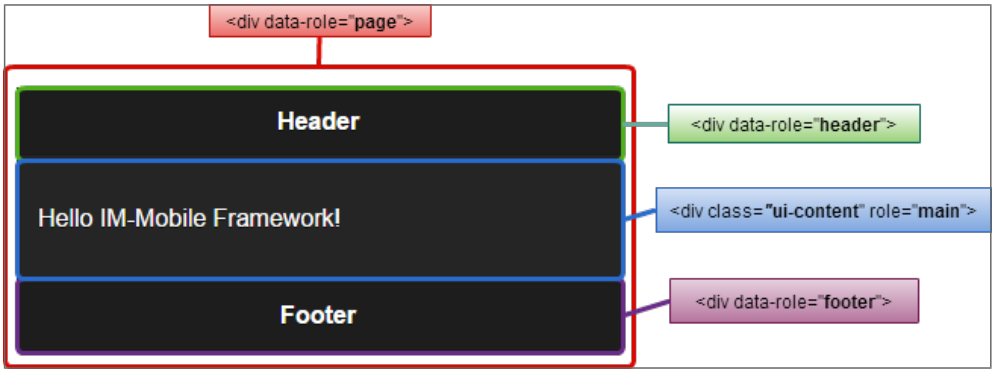
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC "-//SEASAR//DTD S2Container 2.4//EN"
   "https://www.seasar.org/dtd/components24.dtd">
<components>
   <component class="jp.co.intra_mart.framework.extension.seasar.convention.IMNamingConventionImpl">
      <initMethod name="addRootPackageName">
         <arg>"org.seasar.framework.container.warmdeploy"</arg>
      </initMethod>
      <initMethod name="addRootPackageName">
         <arg>"sample.sastruts"</arg>
      </initMethod>
   </component>
   <component class="org.seasar.framework.convention.impl.PersistenceConventionImpl"/>
</components>
```

- 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>/samplesp/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*
> - ルートパッケージの設定はconversion.diconで行います。
>   詳しくは *基本（intra-mart Accel Platform での初めてのプログラミング）* を参照してください。

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。

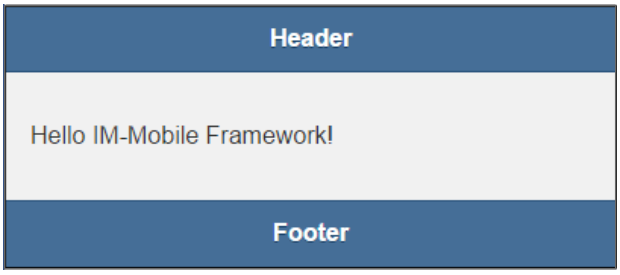
-
  ヘッダのブロック要素であることを定義します。
  任意の要素です。
-
  コンテンツのブロック要素であることを定義します。
  必須の要素です。
  でも動作しますが、推奨されていません。
-
  フッタのブロック要素であることを定義します。
  任意の要素です。

スウォッチを指定する

jQuery Mobileでは各要素に *スウォッチ* を指定することができます。
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">
    <h3>Header</h3>
  </div>
  <div class="ui-content" role="main">
    <p>Hello IM-Mobile Framework!</p>
  </div>
  <div data-role="footer">
    <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

- 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 SAStruts 開発フレームワーク、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
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。

- sample.sastruts.action.samplesp.StoreAction

```
package sample.sastruts.action.samplesp;

import org.seasar.struts.annotation.Execute;

public class StoreAction {
    /**
     * 登録ページのパスを返却します。
     * @return 入力ページのパス
     */
    @Execute(validator = false)
    public String index() {
        return "/sample/imsp/store/store.jsp";
    }
}
```

- %CONTEXT_PATH%/WEB-INF/view/sample/imsp/store/store.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>
<div data-role="page" id="main">
  <imsp:headerWithLink headerText="TODO登録" />
  <div class="ui-content" role="main">
  </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。

jqueyMobile1.4.5を読み込むようにする

ライブラリ群の設定を行います。本サンプルでは /samplesp 以下すべてに jQueryMobile1.4.5 が読み込まれるように設定します。

- %CONTEXT_PATH%/WEB-INF/conf/theme-full-theme-path-config/im_mobile_sample.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 client-type="sp" libraries-version="iap-8.0.11">/samplesp/.*</path>
</theme-full-theme-path-config>
```

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”samplesp/store”とします。

認可の設定

認可を設定します。

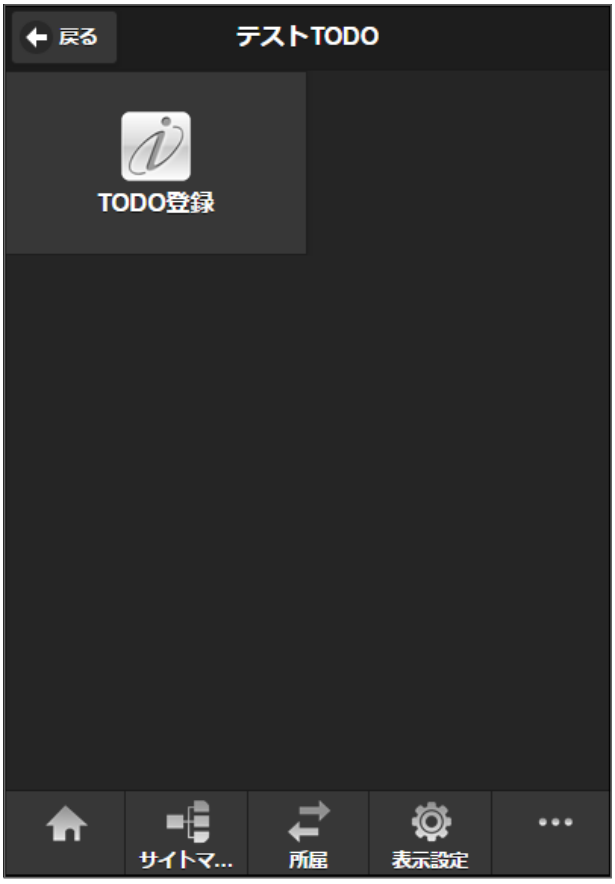
- 「権限設定」ボタンを押下し権限設定（グローバルナビ（スマートフォン用））を表示します。
- 「権限設定を開始する」ボタンを押下します。
- テストTODOの権限の「参照」権限を認証済みユーザに付与します。

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

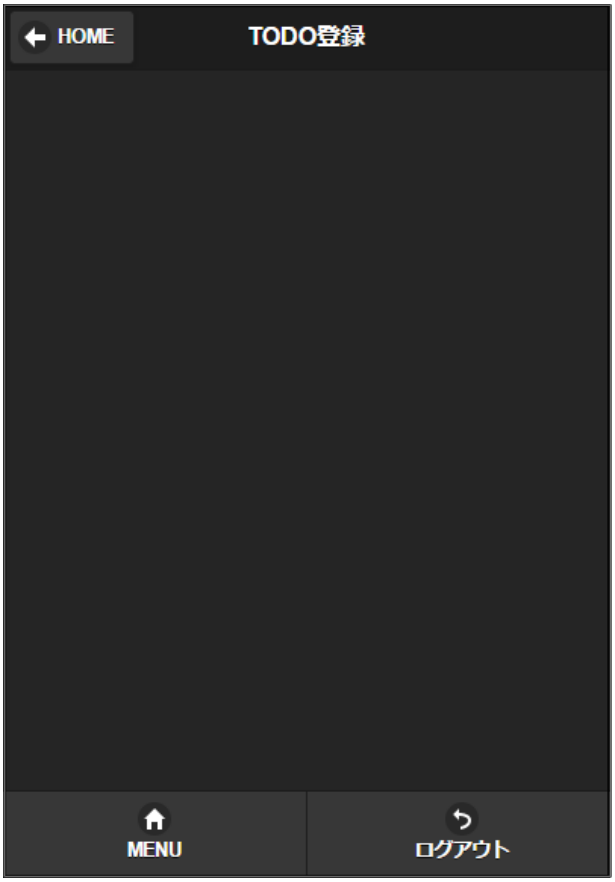
画面を表示する

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。目的の画面を表示することができました。



> **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="limit_date" />
</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>
```

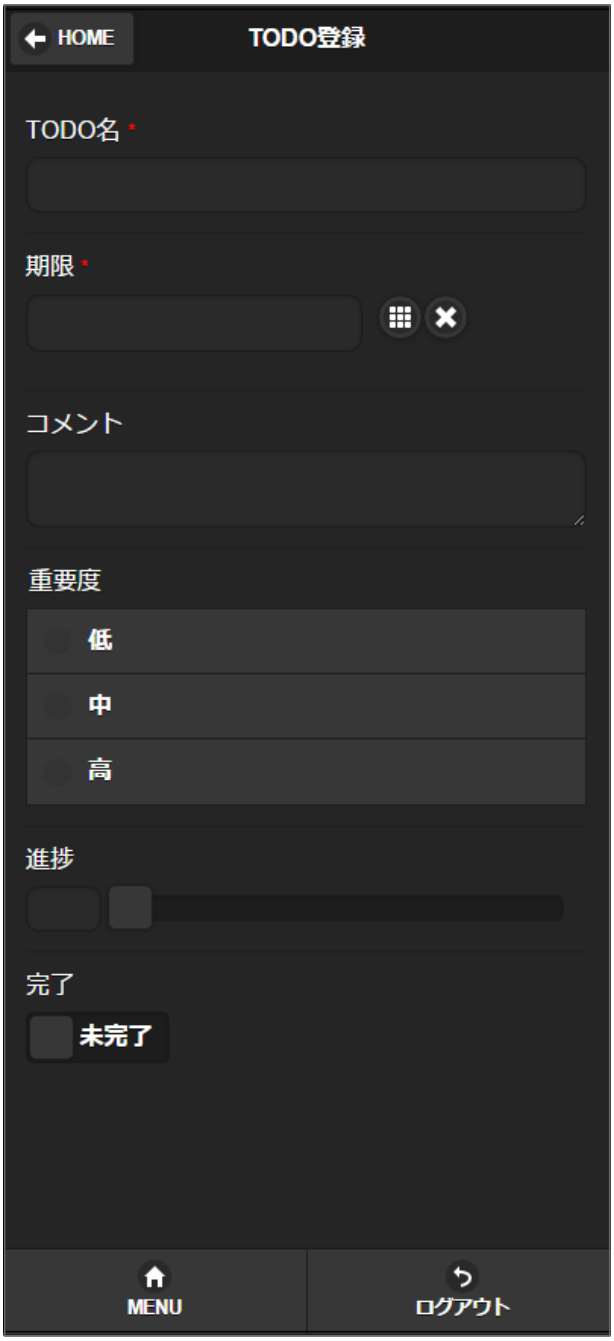
- imsp:slider–スライダーを提供します。

```
<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。

- sample.sastruts.form.samplesp.StoreForm

```
package sample.sastruts.form.samplesp;

public class StoreForm {
    public String title;
    public String limit_date;
    public String comment;
    public String priority;
    public String progress;
    public String complete;
}
```

次に登録処理用のエンティティクラスを作成します。

- sample.sastruts.entity.MfwSample

```
package sample.sastruts.entity;

import java.math.BigDecimal;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name="mfw_sample")
public class MfwSample {

  @Column(name = "id")
  public String todoId;

  @Column(name = "user_cd")
  public String userCd;

  @Column(name = "user_nm")
  public String userName;

  public String title;

  @Column(name = "limit_date")
  public String limitDate;

  public String comment;

  public String priority;

  public BigDecimal progress;

  public String complete;

  public String timestmp;
}
```

> コラム
> エンティティクラスの詳細については S2JDBC エンティティ を参照してください。

登録処理用のサービスクラスを定義します。
S2AbstractServiceクラスを継承し、型変数の宣言は上記で作成したエンティティを指定します。

- sample.sastruts.service.samplesp.MfwSampleService

```
package sample.sastruts.service.samplesp;

import org.seasar.extension.jdbc.service.S2AbstractService;
import sample.sastruts.entity.MfwSample;

public class MfwSampleService extends S2AbstractService<MfwSample> {
}
```

> コラム
> サービスクラスの詳細については S2JDBC サービスの作り方 を参照してください。

- StoreActionクラスにフォームとサービスクラスのフィールド宣言を追加します。

```
import javax.annotation.Resource;
import org.seasar.struts.annotation.ActionForm;
import sample.sastruts.form.samplesp.StoreForm;
import sample.sastruts.service.samplesp.MfwSampleService;

public class StoreAction {

    @Resource
    @ActionForm
    public StoreForm storeForm;

    @Resource
    public MfwSampleService mfwSampleService;
```

- StoreActionクラスに登録処理メソッドを追加します。

```
import java.io.IOException;
import java.math.BigDecimal;

import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.database.exception.DatabaseException;
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 sample.sastruts.entity.MfwSample;
...
@Execute(validator = false)
public String doStore() throws IOException, DatabaseException {
    //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();
    insertData.todoId = new Identifier().get();
    insertData.userCd = userProfile.getUserProfile().getUserCd();
    insertData.userName = userProfile.getUserProfile().getUserName();
    insertData.title = storeForm.title;
    insertData.limitDate = storeForm.limit_date;
    insertData.comment = storeForm.comment;
    insertData.priority =  storeForm.priority;
    insertData.complete =  storeForm.complete;
    if (storeForm.progress != null && !storeForm.progress.equals("")) {
        insertData.progress = new BigDecimal(storeForm.progress);
    }
    insertData.timestmp =  formatter.format(DateTime.now(accountContext.getTimeZone()));

    //登録処理実行
    int result = mfwSampleService.insert(insertData);

    return "/sample/imsp/store/store.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。
> - 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属性は先ほど追加したdoStoreメソッドとします。

```
  <form name="storeForm" name="storeForm" id="storeForm" method="POST" action="samplesp/store/doStore" 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 class="ui-btn ui-btn-b ui-corner-all" id="storeButton">登録</a>
  </form>
```

> **コラム**
>
> - リンクにui-btnクラスを指定するとリンクをボタン表示します。
> - テーマを指定するにはクラス属性にui-btn-bのように「ui-btn-」の後にテーマを指定します。
> - ボタンを角丸にするにはui-corner-allクラスを指定します。
> - 「data-role=”button”」を指定してもボタンになりますが、jQuery Mobile1.4以降では非推奨となっています。
> - 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することができます。

- 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">
<script>
 (function($){
   $('#main').on("pagecreate", function() {
     $("#storeButton").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"
   }
 };
```

登録ボタン押下時のスクリプト処理を修正します。

- JSP

```
 (function($){
  $('#main').on("pagecreate", function() {
   //登録ボタン押下時のイベント
   $("#storeButton").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');
     $("#storeForm").submit();
    } 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

```
(function($){
  $('#main').on("pagecreate", function() {
    // Formの2度押し防止
    $('#storeForm').imspDisableOnSubmit();
    $("#storeButton").tap(function() {
      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。

- sample.sastruts.dto.samplesp.MyAjaxResponse

```
package sample.sastruts.dto.samplesp;

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

- doStoreメソッドを修正します。
  メソッドの返却値はnullとし、ResponseUtil.writeメソッドを使用し
  レスポンスオブジェクトのJSON形式に変換した文字列を返却します。

```
import sample.sastruts.dto.samplesp.MyAjaxResponse;
import net.arnx.jsonic.JSON;
import org.seasar.struts.util.ResponseUtil;
...
@Execute(validator = false)
public String doStore() throws IOException, DatabaseException {
    //ユーザコンテキストを取得します。
    UserContext userProfile = Contexts.get(UserContext.class);
    ...
    insertData.timestmp =  formatter.format(DateTime.now(accountContext.getTimeZone()));

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

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

    //レスポンスオブジェクトをJSON文字列で返却
    ResponseUtil.write(JSON.encode(responseObject));

    return null;
}
```

- 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

- sample.sastruts.action.samplesp.StoreAction.java

```
package sample.sastruts.action.samplesp;

import java.io.IOException;
import java.math.BigDecimal;

import javax.annotation.Resource;

import jp.co.intra_mart.foundation.context.Contexts;
import jp.co.intra_mart.foundation.context.model.AccountContext;
import jp.co.intra_mart.foundation.database.exception.DatabaseException;
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 net.arnx.jsonic.JSON;

import org.seasar.struts.annotation.ActionForm;
import org.seasar.struts.annotation.Execute;
import org.seasar.struts.util.ResponseUtil;

import sample.sastruts.dto.samplesp.MyAjaxResponse;
import sample.sastruts.entity.MfwSample;
import sample.sastruts.form.samplesp.StoreForm;
import sample.sastruts.service.samplesp.MfwSampleService;

public class StoreAction {
    @Resource
    @ActionForm
    public StoreForm storeForm;

    @Resource
    public MfwSampleService mfwSampleService;

    @Execute(validator = false)
    public String doStore() throws IOException, DatabaseException {
        // ユーザコンテキストを取得します。
        final 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。
        final AccountContext accountContext = Contexts.get(AccountContext.class);
        final DateTimeFormatter formatter = DateTimeFormatter.withPattern("yyyy/MM/dd HH:mm");

        // 登録値の設定
        final MfwSample insertData = new MfwSample();
        insertData.todoId = new Identifier().get();
        insertData.userCd = userProfile.getUserProfile().getUserCd();
        insertData.userName = userProfile.getUserProfile().getUserName();
        insertData.title = storeForm.title;
        insertData.limitDate = storeForm.limit_date;
        insertData.comment = storeForm.comment;
        insertData.priority = storeForm.priority;
        insertData.complete = storeForm.complete;
        if (storeForm.progress != null && !storeForm.progress.equals("")) {
            insertData.progress = new BigDecimal(storeForm.progress);
        }
        insertData.timestmp = formatter.format(DateTime.now(accountContext.getTimeZone()));

        final MyAjaxResponse responseObject = new MyAjaxResponse();
        try {
            // 登録処理実行
            final int result = mfwSampleService.insert(insertData);

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

        // レスポンスオブジェクトをJSON文字列で返却
        ResponseUtil.write(JSON.encode(responseObject));

        return null;
    }

    /**
     * 登録ページのパスを返却します。
     * @return 入力ページのパス
     */
    @Execute(validator = false)
    public String index() {
```

```
        return "/sample/imsp/store/store.jsp";
    }
}
```

- %CONTEXT_PATH%/WEB-INF/view/sample/imsp/store/store.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>
<div data-role="page" id="main">
  <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').on("pagecreate", function() {
        // Formの2度押し防止
        $('#storeForm').imspDisableOnSubmit();
        $("#storeButton").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 class="ui-content" role="main">
    <form name="storeForm" id="storeForm" method="POST" action="samplesp/store/doStore" 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="limit_date" />
      </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>
```

- sample.sastruts.form.samplesp.StoreForm.java

```
package sample.sastruts.form.samplesp;

public class StoreForm {
    public String title;
    public String limit_date;
    public String comment;
    public String priority;
    public String progress;
    public String complete;
}
```

- sample.sastruts.entity.MfwSample.java

```
package sample.sastruts.entity;

import java.math.BigDecimal;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name = "mfw_sample")
public class MfwSample {

    @Column(name = "id")
    public String todoId;

    @Column(name = "user_cd")
    public String userCd;

    @Column(name = "user_nm")
    public String userName;

    public String title;

    @Column(name = "limit_date")
    public String limitDate;

    public String comment;

    public String priority;

    public BigDecimal progress;

    public String complete;

    public String timestmp;
}
```

- sample.sastruts.service.samplesp.MfwSampleService.java

```
package sample.sastruts.service.samplesp;

import org.seasar.extension.jdbc.service.S2AbstractService;
import sample.sastruts.entity.MfwSample;

public class MfwSampleService extends S2AbstractService<MfwSample> {
}
```

- sample.sastruts.dto.samplesp.MyAjaxResponse.java

```
package sample.sastruts.dto.samplesp;

public class MyAjaxResponse {
    public String result;

    public boolean error;

    public String errorMessage;

    public String successMessage;

    public String[] detailMessages;
}
```

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

- 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 SAStruts 開発フレームワーク、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。

- sample.sastruts.action.samplesp.ListAction.java

```
package sample.sastruts.action.samplesp;

import org.seasar.struts.annotation.Execute;

public class ListAction {

    @Execute(validator = false)
    public String index() {
        return "/sample/imsp/list/list.jsp";
    }
}
```

- %CONTEXT_PATH%/WEB-INF/view/sample/imsp/list/list.jsp

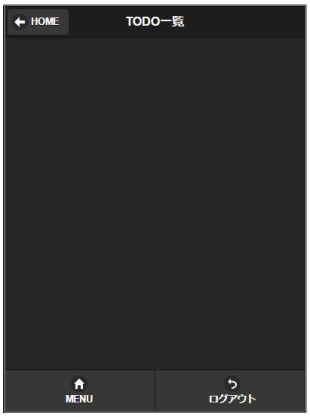
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="imsp" uri="http://www.intra-mart.co.jp/taglib/imsp" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<imui:head>
  <title>TODO一覧</title>
</imui:head>
<div data-role="page">
  <imsp:headerWithLink headerText="TODO一覧" />
  <div class="ui-content" role="main">
  </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”samplesp/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 class="ui-content" role="main">
  <ul data-role="listview" data-divider-theme="b">
    <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

- ListActionに検索処理を追加します。

```
package sample.sastruts.action.samplesp;

import java.util.List;

import javax.annotation.Resource;

import org.seasar.struts.annotation.Execute;
import org.seasar.extension.jdbc.JdbcManager;

import sample.sastruts.entity.MfwSample;

public class ListAction {
    @Resource
    public JdbcManager jdbcManager;

    public List<MfwSample> list;

    @Execute(validator = false)
    public String index() {
        list = jdbcManager.from(MfwSample.class).getResultList();
        return "/sample/imsp/list/list.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。
> - JdbcManagerの詳細については S2JDBC JdbcManager 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

- ListActionクラスを以下のように修正します。

```
import org.seasar.extension.jdbc.JdbcManager;
import org.seasar.extension.jdbc.operation.Operations;

import sample.sastruts.entity.MfwSample;

public class ListAction {

    @Resource
    protected JdbcManager jdbcManager;

    public List<MfwSample> list;

    public long maxRecord;

    public int currentPage = 1;

    public int pageLine = 5;

    @Execute(validator = false)
    public String index() {

        maxRecord = jdbcManager.from(MfwSample.class).getCount();

        list = jdbcManager.from(MfwSample.class)
                    .limit(pageLine)
                    .offset(pageLine * (currentPage - 1))
                    .orderBy(Operations.asc("limitDate"))
                    .getResultList();
        return "/sample/imsp/list/list.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。	

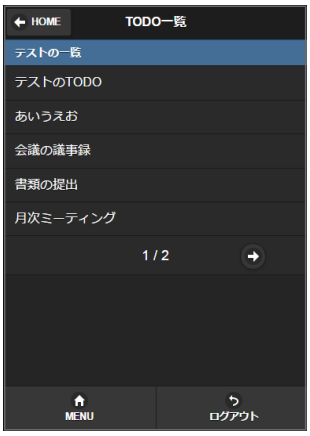
ページング処理の実装

- 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。

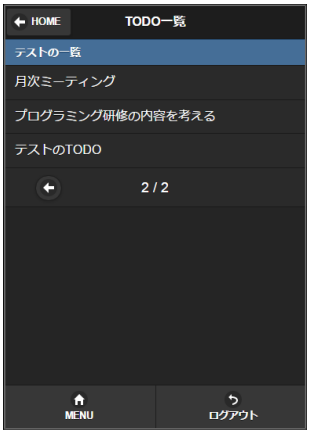
```
<div data-role="page">
  <script>
    function onPageLinkFunc(page) {
      $("input[name=currentPage]").val(page);
      $("#pageForm").submit();
    }
  </script>
```

> **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。



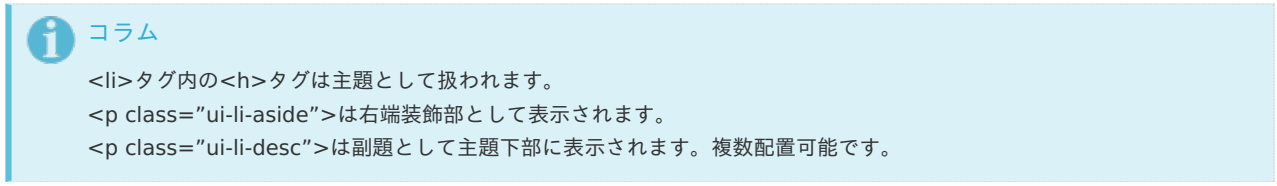
> **コラム**
>
> この項目では、下記のポイントを確認しました。
>
> - ページング処理を実装するには<imsp:spPagingButton>タグを使用する

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" data-divider-theme="b">
  <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

- sample.sastruts.action.samplesp.ListAction.java

```
package sample.sastruts.action.samplesp;

import java.util.List;

import javax.annotation.Resource;

import org.seasar.extension.jdbc.JdbcManager;
import org.seasar.extension.jdbc.operation.Operations;
import org.seasar.struts.annotation.Execute;

import sample.sastruts.entity.MfwSample;

public class ListAction {

    @Resource
    protected JdbcManager jdbcManager;

    public List<MfwSample> list;

    public long maxRecord;

    public int currentPage = 1;

    public int pageLine = 5;

    @Execute(validator = false)
    public String index() {

        maxRecord = jdbcManager.from(MfwSample.class).getCount();

        list = jdbcManager.from(MfwSample.class).limit(pageLine).offset(pageLine * (currentPage -
1)).orderBy(Operations.asc("limitDate")).getResultList();
        return "/sample/imsp/list/list.jsp";
    }
}
```

- %CONTEXT_PATH%/WEB-INF/view/sample/imsp/list/list.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">
  <script>
    function onPageLinkFunc(page) {
      $("input[name=currentPage]").val(page);
      $("#pageForm").submit();
    }
  </script>
  <imsp:headerWithLink headerText="TODO一覧" path="home"/>
  <div class="ui-content" role="main">
    <ul data-role="listview" data-divider-theme="b">
      <li data-role="list-divider"><c:out value="${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>
```

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

- 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 SAStruts 開発フレームワーク、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。

- sample.sastruts.action.samplesp.RefAction.java

```
package sample.sastruts.action.samplesp;

import org.seasar.struts.annotation.Execute;
public class RefAction {

    public String todoId;

    @Execute(validator = false)
    public String index() {
        return "/sample/imsp/ref/ref.jsp";
    }
}
```

- %CONTEXT_PATH%/WEB-INF/view/sample/imsp/ref/ref.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">
  <div data-role="header">
    <h3>TODO参照</h3>
    <a data-rel="back" data-icon="arrow-l">戻る</a>
  </div>
  <div class="ui-content" role="main">
    <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<li>タグにリンクを追加します。

```
 <c:forEach var="record" items="${list}">
 <li>
   <a list-todo-id='<c:out value="${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>
```

- さらに、参照画面へ遷移するフォームを設置します。

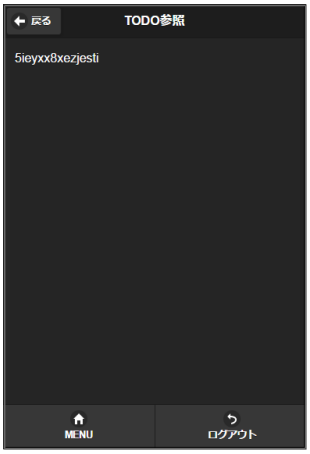
```
<form id="refForm" method="POST" action="<c:url value="ref" />">
  <input type="hidden" name="todoId" />
</form>
```

- 一覧タップ時のイベントを実装します。

```
<script>
 ...
  (function($){
    $(document).on("pagecreate", function() {
      $("li a[list-todo-id]").on("tap", function() {
        $("input[name=todoId]").val($(this).attr("list-todo-id"));
        $("#refForm").submit();
      });
    });
  })(jQuery);
 </script>
```

- 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。

- RefAction.javaを以下のように修正します。

```
import javax.annotation.Resource;

import org.seasar.struts.annotation.Execute;
import org.seasar.extension.jdbc.JdbcManager;

import sample.sastruts.entity.MfwSample;

public class RefAction {
    @Resource
    JdbcManager jdbcManager;

    public String todoId;

    public MfwSample record;

    @Execute(validator = false)
    public String index() {
        record = jdbcManager.from(MfwSample.class).where("id = ?", todoId).getSingleResult();
        return "/sample/imsp/ref/ref.jsp";
    }
}
```

- ref.jspの を以下のように修正します。

```
<div class="ui-content" role="main">
  <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="${record.comment}" />
  </imsp:fieldContain>
  <imsp:fieldContain label="進捗">
    <c:out value="${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

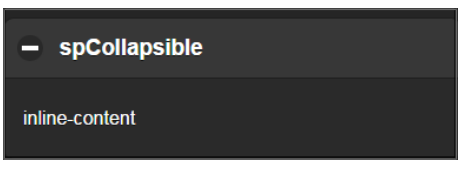
より参照画面を見やすくするために、画面をカスタマイズします。

- ref.jspの登録者・登録日項目を以下の様に修正します。

```
...
<div class="ui-content" role="main">
 <imsp:collapsible title="登録者情報" dataTheme="b" collapse="false" dataInset="false">
  <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:collapsible>
 <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
spCollapsible	開閉可能なブロックを提供します。	

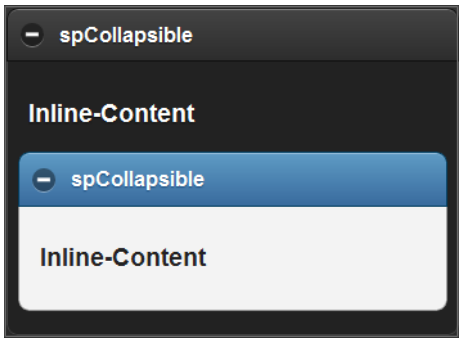
> **コラム**
>
> spCollapsibleタグのdataInset属性にfalseを指定すると、通常のリスト形式表示になります。

- 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="${record.title}" />
   </imsp:fieldContain>
      ...
   <imsp:fieldContain label="コメント">
     ...
   </imsp:fieldContain>
 </imsp:collapsible>
 <imsp:collapsible title="進捗状況" dataTheme="b">
   <imsp:fieldContain label="進捗">
     <imart type="string" value=$record.progress />
   </imsp:fieldContain>
   <imsp:fieldContain label="完了">
     ...
   </imsp:fieldContain>
 </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

- sample.sastruts.action.samplesp.RefAction.java

```
package sample.sastruts.action.samplesp;

import javax.annotation.Resource;

import org.seasar.extension.jdbc.JdbcManager;
import org.seasar.struts.annotation.Execute;

import sample.sastruts.entity.MfwSample;

public class RefAction {
    @Resource
    JdbcManager jdbcManager;

    public String todoId;

    public MfwSample record;

    @Execute(validator = false)
    public String index() {
        record = jdbcManager.from(MfwSample.class).where("id = ?", todoId).getSingleResult();
        return "/sample/imsp/ref/ref.jsp";
    }
}
```

- %CONTEXT_PATH%/WEB-INF/view/sample/imsp/ref/ref.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">
 <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="${record.comment}" />
        </imsp:fieldContain>
      </imsp:collapsible>
      <imsp:collapsible title="進捗状況" dataTheme="b">
        <imsp:fieldContain label="進捗">
          <c:out value="${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")}'>
```

```
            <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/view/sample/imsp/list/list.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);
      $("#pageForm").submit();
    }
    (function($){
      $(document).live("pageinit", function() {
        $("li a[list-todo-id]").unbind().bind("tap", function() {
          $("input[name=todoId]").val($(this).attr("list-todo-id"));
          $("#refForm").submit();
        });
      });
    })(jQuery);
  </script>
  <imsp:headerWithLink headerText="TODO一覧" />
  <div data-role="content">
    <ul data-role="listview">
      <li data-role="list-divider"><c:out value="${currentPage}" />ページ目</li>
      <c:forEach var="record" items="${list}">
      <li>
      <a list-todo-id="<c:out value="${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>
    <form id="refForm" method="POST" action="<c:url value="samplesp/ref" />">
      <input type="hidden" name="todoId" />
    </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することができます。

項目

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

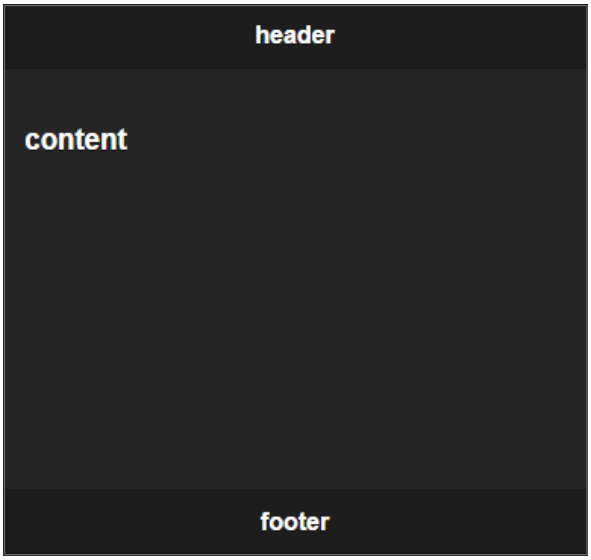
スウォッチ

明暗所の視認性を考慮し、基本的に”A”（デフォルト）としてください。

- マークアップ例。

```
<div data-role="page">
  <div data-role="header" data-position="fixed"> <h3>header</h3></div>
  <div class="ui-content" role="main"><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。

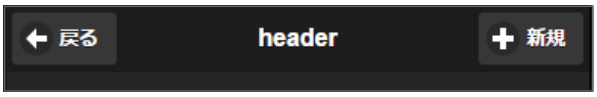
- 基本的に、入力操作系ボタン（新規ボタン、編集ボタン、投稿ボタン）やトランザクション処理ボタン（登録ボタン、更新ボタン）を配置します。
- 入力操作系ボタンは、文字列とします。

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	個別	画面個別機能	各画面で個別に設定します。
3	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。

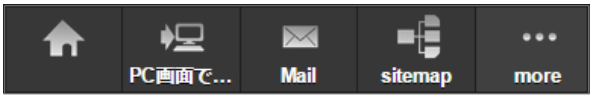
- data-icon=”custom”
- class=”im-smart-icon-common-32-more-navbar”
- data-iconpos=”top”

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



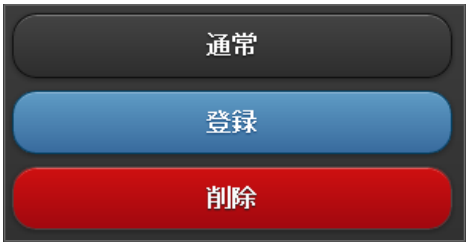
ボタン

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”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).on(“pagecreate”, 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).on("pagecreate",function() {
  //ボタンのタップイベントをバインド
  $("#someButton").on("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).on("pagecreate",function() {
 //ボタンのタップイベントをバインド
 $("input[name=someButton]").on("tap", function() {
  ...
 });
 ...
});
```

エラー処理

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別ドキュメント 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。

Storage

項目

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 WebPaltformのようにⅠ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るようになります。

> **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。

- Actionクラス

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import org.apache.struts.upload.FormFile;
import org.seasar.framework.container.annotation.tiger.Binding;
import org.seasar.framework.container.annotation.tiger.BindingType;
import org.seasar.framework.exception.IORuntimeException;
import org.seasar.struts.annotation.Execute;

import jp.co.intra_mart.common.aid.jdk.java.io.IOUtil;
import jp.co.intra_mart.foundation.service.client.file.PublicStorage;

public class UploadAction {
    @Binding(bindingType = BindingType.NONE)
    public FormFile formFile;

    /**
     * クライアントよりアップロードされたファイルをPublicStorageに保存します。
     */
    @Execute(validator = false)
    public String upload() {
        // ファイル保存先を指定したPublicStorageのインスタンスを生成します。
        final PublicStorage storage = new PublicStorage("sample", formFile.getFileName());

        try {
            final InputStream istr = formFile.getInputStream();

            try {
                final OutputStream ostr = storage.create();

                try {
                    // アップロードされたファイルをPublicStorageに書き込みます。
                    IOUtil.transfer(istr, ostr);
                } finally {
                    IOUtil.closeCloseableQuietly(ostr);
                }
            } finally {
                IOUtil.closeCloseableQuietly(istr);
            }
        } catch (final IOException e) {
            throw new IORuntimeException(e);
        }

        return "index.jsp";
    }
}
```

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。

```
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;

import org.seasar.framework.exception.IORuntimeException;
import org.seasar.struts.annotation.Execute;

import jp.co.intra_mart.common.aid.jdk.java.io.IOUtil;
import jp.co.intra_mart.foundation.service.client.file.PublicStorage;
import jp.co.intra_mart.framework.extension.seasar.struts.util.IMResponseUtil;

public class DownloadAction {
  /**
   * PublicStorageに保存されているファイル「sample.txt」をクライアントへ送信します。
   */
  @Execute(validator = false)
  public String download() {
    try {
      // 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");

      if (!storage.isFile()) {
        // ファイルが存在しない場合は例外をスローします。
        throw new FileNotFoundException();
      }

      final InputStream istr = storage.open();

      try {
        // クライアントへsample.txtを送信します。
        IMResponseUtil.download(storage.getName(), istr, storage.length());
      } finally {
        IOUtil.closeCloseableQuietly(istr);
      }

      return null;
    } catch (final IOException e) {
      throw new IORuntimeException(e);
    }
  }
}
```

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となります。

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

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。

プログラムソース

```java
package jp.co.intra_mart.example.job;

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;

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.err.println("Hello. " + message);
      // 処理結果：正常
      return JobResult.success("ジョブが正常に実行されました。");
    } catch (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);
    }
  }

}
```

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。

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することができます。

```
// 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

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();
}
```

> **コラム**
>
> バーチャルテナントによる複数テナント 環境でテナント毎にロックを行いたい場合は「lockId」にテナントIDを含める等、テナント毎に一意のIDでロックを行うようにしてください。

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

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です。

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 です。

```
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 です。
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して扱います。

```
// 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、許可されたユーザであると判定します。

```
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 Platform 2013 Autumn(Eden) 以降が必要です。
> intra-mart Accel Platform 2013 Autumn(Eden) 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)

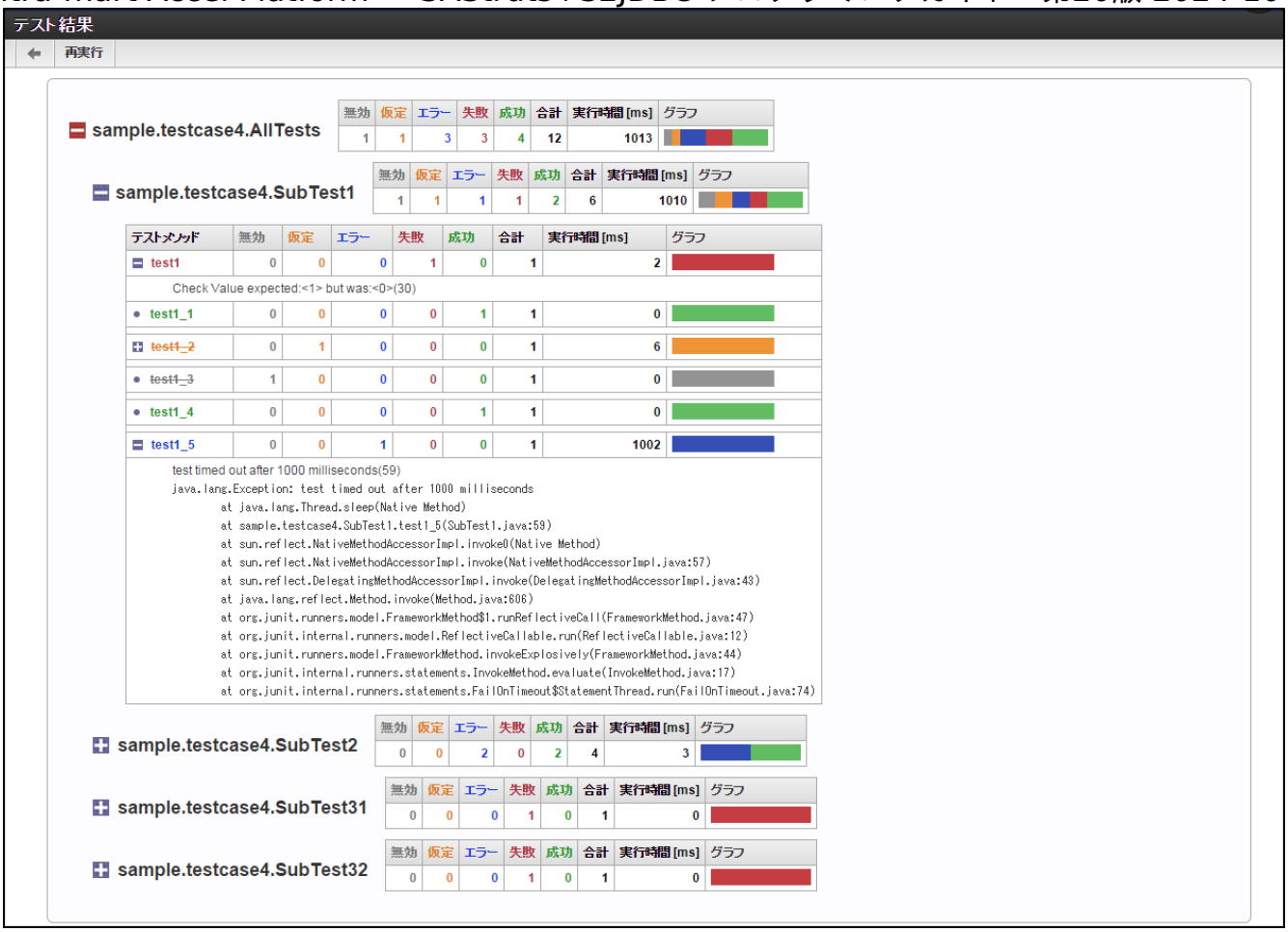
サーバサイド単体テスト

項目

サーバサイド単体テストとは

サーバサイド単体テストはJUnitで作成したテストクラスファイルの単体テストを稼働中のサーバで実施します。
実際に動作しているアプリケーションサーバ内で実行されますので、実際の動作環境で単体テストを実施できます。
JUnit3.x および JUnit4.x 系のテストが実行できます。

以下の図は、サーバで単体テストを実行した結果画面サンプルです。
テスト結果状況およびエラー状況が色覚的に分かりやすい表現になっています。



> **注意**
>
> この機能は、単体テストを稼働中のサーバ単体で実施するものであり、eBuilderと連携するものではありません。

アプリケーションサーバの構築

サーバサイド単体テストを実行できる環境を構築します。

IM-Juggling においてIM-UnitTestを含めたWARファイルでアプリケーションサーバを構築してください。
IM-UnitTestはモジュール構成内の「開発フレームワーク」カテゴリに存在します。

テストケースクラスについて

JUnitの仕様に基づいてテストケースクラスを作成します。
作成するテストケースクラスはJUnit3.x と JUnit4.x のどちらでも構いません。

> **注意**
>
> テストケースクラスを作成するにあたって、以下の点に注意してください。
>
> - 画面遷移が発生する API を使用してはいけません。
>   画面遷移に関しての試験はできません。
>   別の関数に分けるなどして、テストを行ってください。

テストケースクラスの配置と実行

テストケースクラスの配置

作成したテスト対象ファイル、テストケースファイルを以下の場所に配置してください。
なお、テスト対象ファイルが既に配置されている場合は、配置の必要はありません。

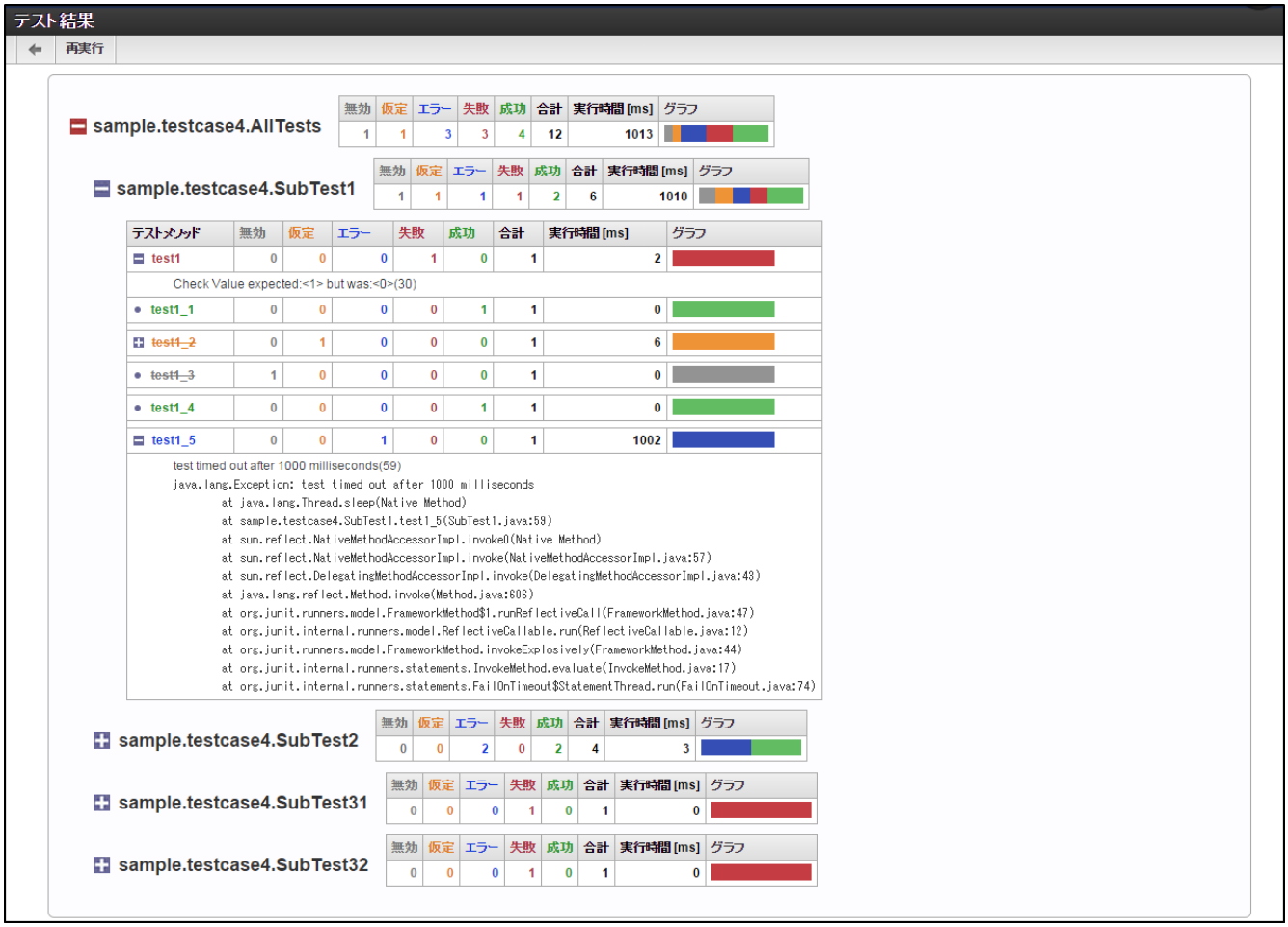
アプリケーションサーバ内のWARファイル展開フォルダ/WEB-INF/classes

テストケースクラスの実行

1. **クラスファイルを配置したら必ずサーバを再起動してください。**
2. 任意にユーザでログインします。
3. 「グローバルナビ」-「サイトマップ」-「テストツール」-「Unit Test Launchar」をクリックします。

> **コラム**
>
> 履歴には、過去に実行されたテストのパスがパス名の昇順で表示されます。
> テストのパスのリンクをクリックすることで、再実行が可能です。
> また、履歴の「削除」アイコンをクリックすることで、履歴を削除できます。
> 履歴をすべて削除する場合は、「全削除」ボタンをクリックします。

4. テストケースパスに、実行したいテストケースクラスをフルパッケージ名で入力します。
5. 実行ボタンをクリックします。
6. テストが実行され、結果が表示されます。



項目	説明
テストメソッド	実行したテストメソッド名です。 テストメソッドで事前検証に失敗、評価に失敗、またはエラーが発生した場合は、テストメソッドがリンクとなります。 リンクをクリックすることで、詳細な評価内容を確認できます。
無効	無効となっているテストメソッドの数です。 テストメソッドに対する値は、テストメソッドが無効となっている場合に1となります。 JUnit4.xの仕様に基づいて作成されたテストケースの場合のみ表示されます。
仮定	前提条件の評価に失敗した数です。 テストメソッドに対する値は、テストメソッドで、事前検証が失敗した場合に1となります。 JUnit4.xの仕様に基づいて作成されたテストケースの場合のみ表示されます。
エラー	テスト中にエラーが発生した数です。
失敗	テスト中に評価関数において、評価に失敗した数です。 テストメソッドに対する値は、テストメソッドで評価関数が1つでも失敗した場合に1となります。
成功	テスト中に評価関数において、評価に成功した数です。 テストメソッドに対する値は、テストメソッド内ですべての評価関数が成功した場合に1となります。
合計	テストメソッドの合計です。 テストメソッドに対する値は、必ず1になります。
実行時間	テストの実行時間です。単位はミリ秒です。
グラフ	テスト結果を色覚的に棒グラフで表現しています。

CSRF対策

項目

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トークンチェック処理をインターセプターを使って実行する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」をインターセプターを使って実行する方法を紹介します。

インターセプター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

- 入力フォームのjsp
- 登録処理のActionクラス
- app.diconファイル
- customizer.diconファイル

注意
ここでは、 intra-mart Accel Platform 2014 Winter(Iceberg) で導入されたSecureTokenValidationInterceptor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

入力フォームにimSecureTokenタグを追加します。

```jsp
<%@page pageEncoding="UTF-8"%>
<%@ 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>登録</title>
</imui:head>

<div class="imui-title">
 <h1>登録</h1>
</div>

<div class="imui-form-container-narrow">
 <form name="doRegisterForm" action="todo/register/doRegister" method="POST">
   <!-- 入力フォームにimSecureTokenタグを追加 -->
   <imst:imSecureToken />

   ...

 </form>
</div>
```

登録処理のActionクラス

登録処理を行うメソッド名をdoRegisterにします。このメソッド名は後述のcustomizer.diconの設定と一致させます。

ここではインターセプトするメソッド名をdoRegisterとしています。適宜変更してください。

```java
  /**
   * 登録処理
   *
   * @return index
   */
  @Execute(validator = true, input = "index")
  public String doRegister() {

    // 登録処理

    return index();
  }
```

app.diconファイル

secureTokenValidationInterceptorコンポーネント定義を追加します。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE components PUBLIC "-//SEASAR//DTD S2Container 2.4//EN"
   "https://www.seasar.org/dtd/components24.dtd">
<components>
   <include path="convention.dicon"/>
   <include path="aop.dicon"/>
   <include path="j2ee.dicon"/>
   <!--
   <include path="s2jdbc.dicon"/>
   -->
   <component name="actionMessagesThrowsInterceptor"
class="org.seasar.struts.interceptor.ActionMessagesThrowsInterceptor"/>

   <component name="secureTokenValidationInterceptor"
         class="jp.co.intra_mart.framework.extension.seasar.struts.interceptor.SecureTokenValidationInterceptor"
         instance="prototype">
      <aspect pointcut="invoke">
         <component class="org.seasar.framework.aop.interceptors.InterceptorLifecycleAdapter" />
      </aspect>
   </component>

</components>
```

customizer.diconファイル

actionCustomizerにsecureTokenValidationInterceptorを追加します。

ここでは、対象となるActionのメソッド名としてdoRegisterを設定しています。適宜変更してください。

```
   <component name="actionCustomizer"
      class="org.seasar.framework.container.customizer.CustomizerChain">
      <initMethod name="addAspectCustomizer">
         <arg>"aop.traceInterceptor"</arg>
      </initMethod>
      <initMethod name="addAspectCustomizer">
         <arg>"actionMessagesThrowsInterceptor"</arg>
      </initMethod>
      <initMethod name="addAspectCustomizer">
         <arg>"secureTokenValidationInterceptor"</arg>
         <arg>"doRegister"</arg>
      </initMethod>
      <initMethod name="addCustomizer">
         <arg>
            <component
               class="org.seasar.framework.container.customizer.TxAttributeCustomizer"/>
         </arg>
      </initMethod>
      <initMethod name="addCustomizer">
         <arg>
            <component
               class="org.seasar.struts.customizer.ActionCustomizer"/>
         </arg>
      </initMethod>
   </component>
```

> **コラム**
> 複数のメソッドを設定する場合は、”doRegister, doUpdate, doRemove”のようにカンマ区切りで記述します。

設定ファイル

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

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
- 作成したクラスに関する package-info 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();
    }

}
```

- 作成したクラスに関する package-info 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。

package-info クラスのサンプル（jp.co.intra_mart.jaxb.sample.package-info）

```
@javax.xml.bind.annotation.XmlSchema(namespace = "http://intra_mart.co.jp/jaxb/sample/jaxb-config",
elementFormDefault = javax.xml.bind.annotation.XmlNsForm.QUALIFIED)
package jp.co.intra_mart.jaxb.sample;
```

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

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;
        }
    }
}
```

旧バージョンで作成したプログラムの実行

version 7.2 で動作していたプログラムをintra-mart Accel Platform上で動作するための方法を説明します。
この章では、最低限必要な対応について説明します。

> **コラム**
> 詳細は、移行ガイドを参照してください。

項目

前提

ここでは、*認可* については基本的に触れません。
移行の場合、メニュー移行により認可のリソースが設定されます。
新規/追加開発時は、マニュアルに従い、設定してください。

> **注意**
> 前提として、以下の注意点、変更点がありますので、注意してください。
>
> - v7.2でのAPIを利用する場合は、IM-Jugllingで互換機能モジュールとモジュール開発支援ライブラリを選択してください。
> - 互換機能モジュールの動作の前提として、v7.2からの移行が必要です。
> - JSPファイルの格納先は、SAStruts標準のWEB-INF/viewディレクトリに変更になりました。

移行手順

旧バージョンで動作していたプログラムをintra-mart Accel Platform上で動作させる方法としては、 以下の2つの方法があります。

- intra-mart Accel Platformの画面仕様に合わせる
- iframe内で表示する

動作させたいアプリケーションにより以下の対応方法を参照の上、対応してください。

intra-mart Accel Platformの画面仕様に合わせる

画面（HTMLファイル）の対応

- frameset, frameタグを削除

  frame間でデータのやりとりを行っている場合、Ajaxを使った実装に置き換えることをお勧めします。

- html, bodyタグを削除

  bodyタグのonload属性にJavaScriptを記述していた場合、jQueryの機能を利用して実行するようにしてください。

```
jQuery(document).ready(function() {
doSomething();
});
```

- <imarttag:imartDesignCss> を削除
- headタグを置き換える

  <head>タグを<imui:head>タグに置き換えます。

- タイトルバータグを置き換える

  <imarttag:imartTitleBar> をheaderタグに置き換えます。

- form の target を変更

  target=”IM_MAIN” を target=”_top” に変更します。

diconファイルの設定

必要に応じて、

- s2jdbc.dicon
- convention.dicon
- app.dicon

を編集してください。

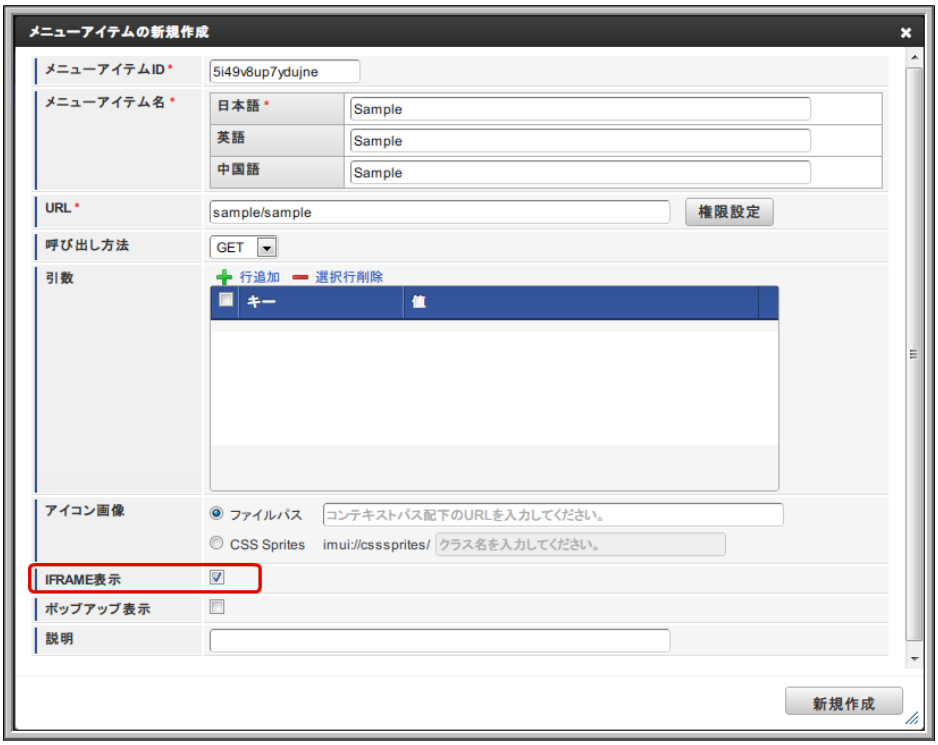
ロジックの対応

API対応を 互換対応表 を参照の上、対応してください。

iframe内で表示する

iframe設定

テナント管理者で、メニュー登録を行い、iframeの設定を行います。



注意

iframe利用時は以下の制限事項がありますので、注意してください。

- IFRAMEを使用したページをマイメニューに登録して開くとページを表示できない場合があります。
- エラーページをカスタマイズするとIFrameリダイレクタのiframe内にエラーページが表示されます。
- 認証確認対象の画面には、iframe 内に表示する前提の画面のURLは設定できません。

no-theme設定

iframeを使うと、テーマが2重に表示されることがあるため、<im_path>/WEB-INF/conf/theme-no-theme-path-config/に以下のようなxmlファイルにて、no-theme設定を行う必要があります。

```
<?xml version="1.0" encoding="UTF-8"?>
 <theme-no-theme-path-config xmlns="http://www.intra-mart.jp/theme/theme-no-theme-path-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.intra-mart.jp/theme/theme-no-theme-path-config theme-no-theme-path-config.xsd ">

  <path>/bpw\-.+\.service.*</path>
  <path>bpw/.*</path>
```

<path>タグ内に、テーマを表示したくないファイルパスを記述してください。
フルパスでも、上記のように正規表現でも記載できます。

diconファイルの設定

必要に応じて、

- s2jdbc.dicon
- convention.dicon
- app.dicon

を編集してください。

ロジックの対応

API対応を 互換対応表 を参照の上、対応してください。

コラム

詳細は、移行ガイドを参照してください。