Copyright © 2012 NTT DATA INTRAMART CORPORATION

目次

- 改訂情報
- はじめに
  - 本書の目的
  - 対象読者
  - 本書の構成
- 基本（ intra-mart Accel Platform での初めてのプログラミング）
  - 前提条件
  - intra-mart e Builder for Accel Platform で Hello World を作ろう
  - 登録と画面表示
- 応用（ intra-mart Accel Platform の機能を使いこなす）
  - 設定ファイル
  - ルーティング
  - 認可
  - アクセスコンテキスト
  - 国際化
  - データベース
  - ログ
  - UI（デザインガイドライン）
  - UI（スマートフォン開発ガイドライン）
  - エラー処理
  - Storage
  - 非同期処理
  - ジョブスケジューラ
  - Lockサービス
  - Cacheサービス
  - ショートカットアクセス機能
  - サーバサイド単体テスト
- 共通ライブラリの作成
  - 拡張APIの作成
  - グローバル関数の作成
  - 拡張<IMART>タグの作成
- JavaClassとの連携
  - 標準Javaクラスとの連携方法
  - 自作Javaクラスとの連携方法
- 旧バージョンで作成したプログラムの実行
  - 前提
  - 移行手順
- 付録
  - スクリプト開発モデルの実行処理シーケンスについて

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 ▪ 「*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 ▪ 「*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 ▪ 「*バリデーションルール* 」に「date」「time」「datetime」を追加 ▪ 「*バリデーションルール リファレンス* 」に「*date* 」「*time* 」「*datetime* 」を追加 ▪ 「*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。
2014-06-09	第8版　下記を追加・変更しました ▪ 「*共通ライブラリの作成* 」を追加 ▪ 「*JavaClassとの連携* 」を追加 ▪ 「*付録* 」を追加
2014-08-01	第9版　下記を追加・変更しました ▪ 「*アクセスコンテキスト*」の仕様の記述を「 アクセスコンテキスト仕様書 」へ移動
2014-12-01	第10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 ▪ 「*設定ファイル*」を追加 ▪ 「*JSSP Validator*」の「カスタムバリデータ」にコラムを追加

変更年月日	変更内容
2015-08-01	第11版　下記を追加・変更しました ▪ 「*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 ▪ 「*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
2016-12-01	第13版　下記を追加・変更しました ▪ DB2に関する記述を削除
2017-08-01	第14版　下記を追加・変更しました ▪ 「*旧バージョンで作成したプログラムの実行*」に、ファンクションコンテナ(JSファイル)上で実行するJavaのAPIの実行結果に関する記述を追加
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

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
  ただし、 スクリプト開発モデル の基礎知識（概念）を理解している必要があります。
- 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。

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 でスクリプト開発を利用した Hello World を作成することによって intra-mart Accel Platform でのスクリプト開発の流れを体験します。

チュートリアルの流れ
- 前提条件
- intra-mart e Builder for Accel Platform で Hello World を作ろう
  - ステップ1：プロジェクトの作成と設定
  - ステップ2：入力画面の作成
  - ステップ3：出力画面の作成
  - ステップ4：入力画面処理の作成
  - ステップ5：出力画面処理の作成
  - ステップ6：ルーティング設定ファイルの作成
- 登録と画面表示
  - ステップ1：メニューの登録
  - ステップ2：Hello Worldの動作

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スクリプト開発フレームワークを含めて環境を作成してください。
  開発環境の Resin サーバ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：入力画面の作成

入力画面（input.html）を作成します。
プロジェクトの src/main/jssp/src/ の配下に helloworld/js/ 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「 input.html 」ファイルを作成し、 以下のように実装をします。

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

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

<!-- 入力画面 -->
<div class="imui-form-container-narrow">
 <p>
  <label>Please input the name. </label>
 </p>
 <!-- Formの設定 actionに出力結果のパスを入力 -->
 <form action="helloworld/output" name=form id="hello_info" method="post">
  <div>
   <!-- テキストボックスの設定 -->
   <imart type="imuiTextbox" id="name" name="name" size="10" ></imart>
  </div>
  <!-- submitボタンの設定 -->
  <imart type ="imuiButton" value="Hello!!" name="button" class="mt-10" id="button" inputType="submit"></imart>
 </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3：出力画面の作成

出力画面（output.html）を作成します。
プロジェクトの src/main/jssp/src/helloworld/js/ の配下に「 output.html 」という名前で作成し、以下のように実装をします。

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

<!-- 画面上に表示されるタイトル -->
<div class="imui-title">
 <h1>Hello, World (スクリプト開発)</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」ボタンのアイコンを作成 JavaScriptの関数を呼び出す -->
    <a href="javascript:back();" 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> <imart type="imuiTextbox" id="result"
   name="result" value=name class="imui-text-readonly" readonly />
 </label>
</div>

<!-- 「戻る」ボタンのフォーム情報 actionに入力画面のパスを設定する。-->
<form id="back-form" name="backForm" action="helloworld/input"
 method="POST"></form>
```

> **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 <imart type=”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。

> **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：入力画面処理の作成

入力画面の初期化処理を行う JS ファイル（input.js）を実装します。
プロジェクトの src/main/jssp/src/helloworld/js/ の配下に「 input.js 」という名前で作成し、以下のように実装をします。

```
/**
 * Hello Worldでは特別に処理を行わないため未処理
 * 初期化等が必要な場合はinit()にその処理を記述していく。
 */
function init(request) {

}
```

> **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：出力画面処理の作成

入力情報に対する結果表示処理を行う JS ファイル（output.js）を実装する。
プロジェクトの src/main/jssp/src/helloworld/js/ の配下に「 output.js 」という名前で作成し、以下のように実装をします。

```
/**
 * 出力結果のバインド変数
 */
var name;

/**
 * 初期表示
 * 名前を渡す
 */
function init(request) {
  name = 'Hello, ' + request['name'];    // 出力結果を作成し、バインド変数へ格納
}
```

> **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：ルーティング設定ファイルの作成

ルーティング用の xml (helloworld.xml)を作成します。
プロジェクトの src/main/conf/routing-jssp-config/ の配下に helloworld.xml のファイルを作成し、以下のようにファイルのマッピング情報を記述します。
認可の設定に当たる authz-default の mapper 属性には “welcome-all” を設定します。
URLの設定に当たる file-mapping には、 path 属性に2つの画面のURLを、 page 属性に2つの画面のファイルパスをそれぞれ設定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config
  xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-config routing-jssp-config.xsd ">

  <authz-default mapper="welcome-all" />

  <file-mapping path="/helloworld/input" page="/helloworld/js/input" />
  <file-mapping path="/helloworld/output" page="/helloworld/js/output" />
</routing-jssp-config>
```

> **コラム**
>
> ルーティングの設定の詳細に関しては、 *ルーティング* を参照してください。

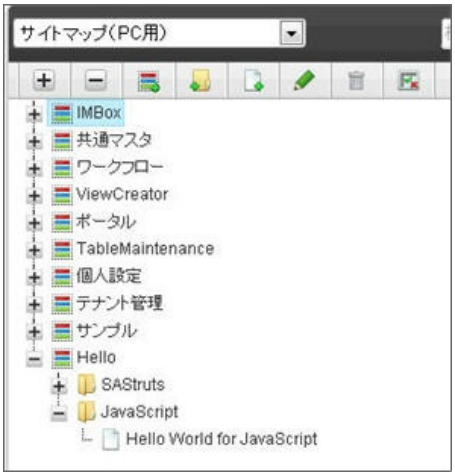
> **注意**
>
> 文字コードを UTF-8 にして保存してください。

ここまでスクリプト開発のサンプルの作成が完了となります。
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* 」となります。
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 JavaScript
URL	helloworld/input

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。

- スクリプト開発を用いて、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。

設定ファイル

項目
- source-config.xml

source-config.xml

source-config.xmlは、スクリプト開発モデルのプログラムリソースの読み込みや実行に関する制御を行う設定ファイルです。
設定を変更することで、プログラムリソースの変更をサーバを再起動させることなく反映することが可能です。
source-config.xmlはデフォルトで以下のディレクトリパスに格納されています。

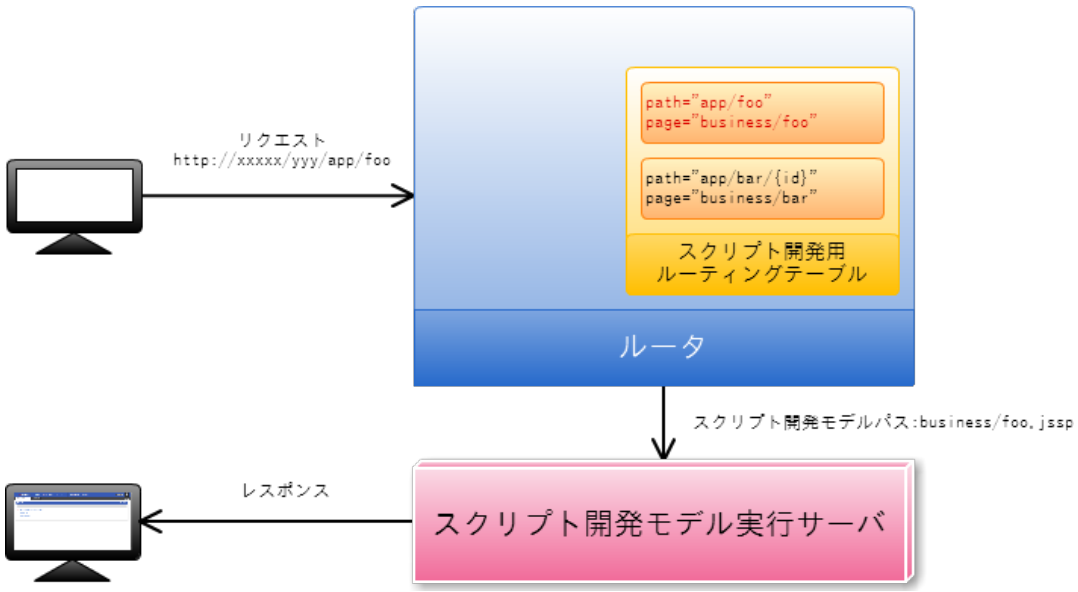
- %CONTEXT_PATH%/WEB-INF/jssp/compatible/src/
- %CONTEXT_PATH%/WEB-INF/jssp/platform/src/
- %CONTEXT_PATH%/WEB-INF/jssp/product/src/
- %CONTEXT_PATH%/WEB-INF/jssp/src/

source-config.xmlの詳しい説明は、設定ファイルリファレンスの source-config.xml を参照してください。

ルーティング

項目
- 概要
- スクリプト開発モデル用ルーティングテーブル
  - URLへのスクリプト開発モデルのプログラムの割り当て
  - PathVariables
  - 認可
  - クライアントタイプ

概要



ルーティングとはURLに対して処理の登録、振り分けを行う機能です。
処理の登録が行われていないリクエストに対しては、アプリケーションサーバへのリクエスト処理が続行します。
スクリプト開発モデル では「スクリプト開発モデル用ルーティングテーブル」を利用してURLに対して スクリプト開発モデル のプログラムの割り当てを行います。

スクリプト開発モデル用ルーティングテーブル

設定ファイルの例を用いてスクリプト開発モデル用ルーティングテーブルが行う役割について説明します。

URLへのスクリプト開発モデルのプログラムの割り当て

- *http://<HOST>:<PORT>/<CONTEXT_PATH>/app/foo* を スクリプト開発モデル のパス business/foo に割り当てるルーティングテーブルを作成します。
  - プレゼンテーション・ページ
    %CONTEXT_PATH%/WEB-INF/jssp/src/business/foo.html にファイルを作成します。
    ファイルを以下の内容にします。

    ```
    This is foo.
    ```

  - ファンクション・コンテナ
    処理が存在しないため、ファンクション・コンテナは作成しません。
  - ルーティングテーブル
    %CONTEXT_PATH%/WEB-INF/conf/routing-jssp-config/routing-programming-guide-foo.xml にファイルを作成します。
    ファイルを以下の内容にします。

    ```
    <?xml version="1.0" encoding="UTF-8"?>
    <routing-jssp-config
      xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-config routing-jssp-config.xsd">

      <authz-default mapper="welcome-all" />

      <file-mapping path="/app/foo" page="business/foo" />

    </routing-jssp-config>
    ```

    file-mapping要素でURLと スクリプト開発モデル のプログラムのマッピングを行います。
    path属性にURLのパス /app/foo 、page属性に スクリプト開発モデル のパス business/foo を指定します。
- 設定ファイルを反映させるために、アプリケーションサーバを再起動します。
- *http://<HOST>:<PORT>/<CONTEXT_PATH>/app/foo* へアクセスします
  ページに以下が表示されます

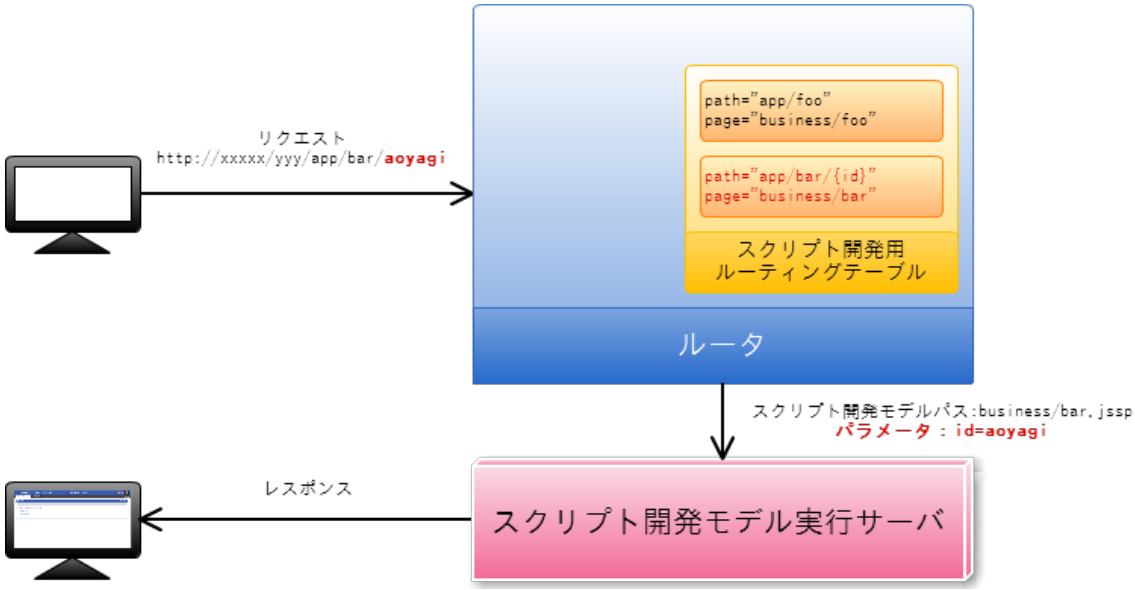
  ```
  This is foo.
  ```

  コラム

  この項目では、下記のポイントを確認しました。

  - URLへの スクリプト開発モデル のプログラムの割り当てはfile-mapping要素を用いて行う。
  - file-mapping要素のpath属性にURLのパス、page属性に スクリプト開発モデル のパスを記述する。

PathVariables



- *http://<HOST>:<PORT>/<CONTEXT_PATH>/app/bar/{id}* を スクリプト開発モデル のパス business/bar に割り当てるルーティングテーブルを作成します。
  - プレゼンテーション・ページ
    %CONTEXT_PATH%/WEB-INF/jssp/src/business/bar.html にファイルを作成します。
    ファイルを以下の内容にします。

```
"id" is <imart type="string" value=id />.
```

  - ファンクション・コンテナ
    %CONTEXT_PATH%/WEB-INF/jssp/src/business/bar.js にファイルを作成します。
    ファイルを以下の内容にします。

```
var id;

function init(request) {
   id = request.id;
}
```

  - ルーティングテーブル
    %CONTEXT_PATH%/WEB-INF/conf/routing-jssp-config/routing-programming-guide-bar.xml にファイルを作成します。
    ファイルを以下の内容にします。

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config
   xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-config routing-jssp-config.xsd">

   <authz-default mapper="welcome-all" />

   <file-mapping path="/app/bar/{id}" page="business/bar" />

</routing-jssp-config>
```

    path属性の中に{[識別子]}と記述することでURLの途中の値がパラメータとして引き渡されます。
- 設定ファイルを反映させるために、アプリケーションサーバを再起動します。
- *http://<HOST>:<PORT>/<CONTEXT_PATH>/app/bar/aoyagi* へアクセスします
  ページに以下が表示されます

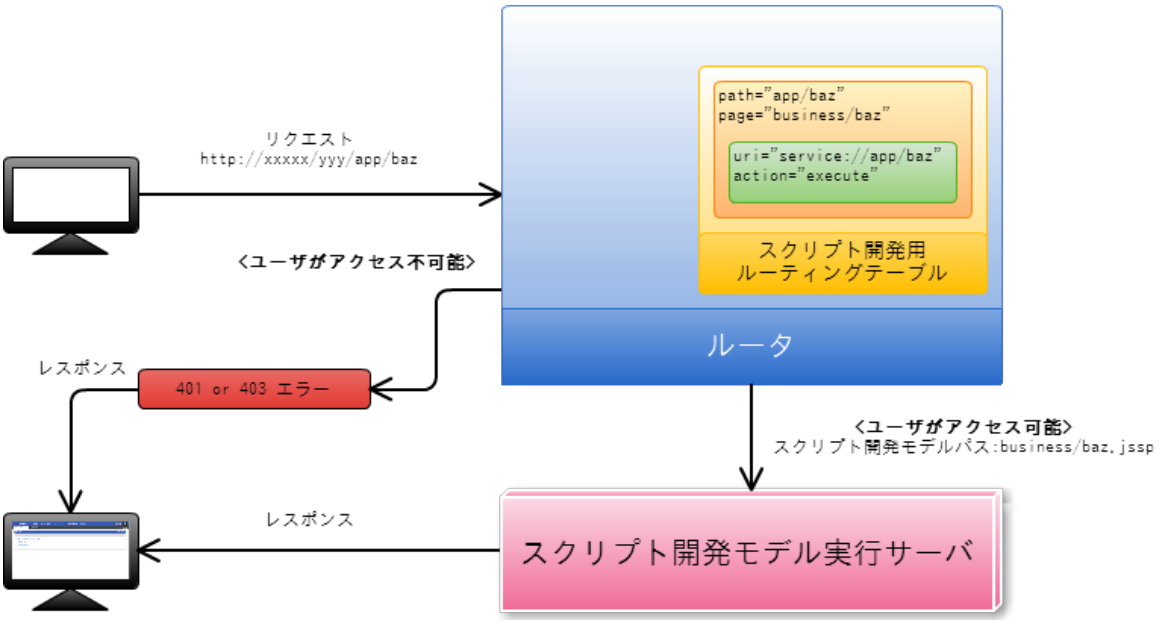
```
"id" is aoyagi.
```

- *http://<HOST>:<PORT>/<CONTEXT_PATH>/app/bar/ueda* へアクセスします
  ページに以下が表示されます

```
"id" is ueda.
```

> **コラム**
>
> この項目では、下記のポイントを確認しました。
>
> - file-mapping要素のpath属性に{[識別子]}と記述することでURLの途中の値がリクエスト・パラメータとして扱える。

認可

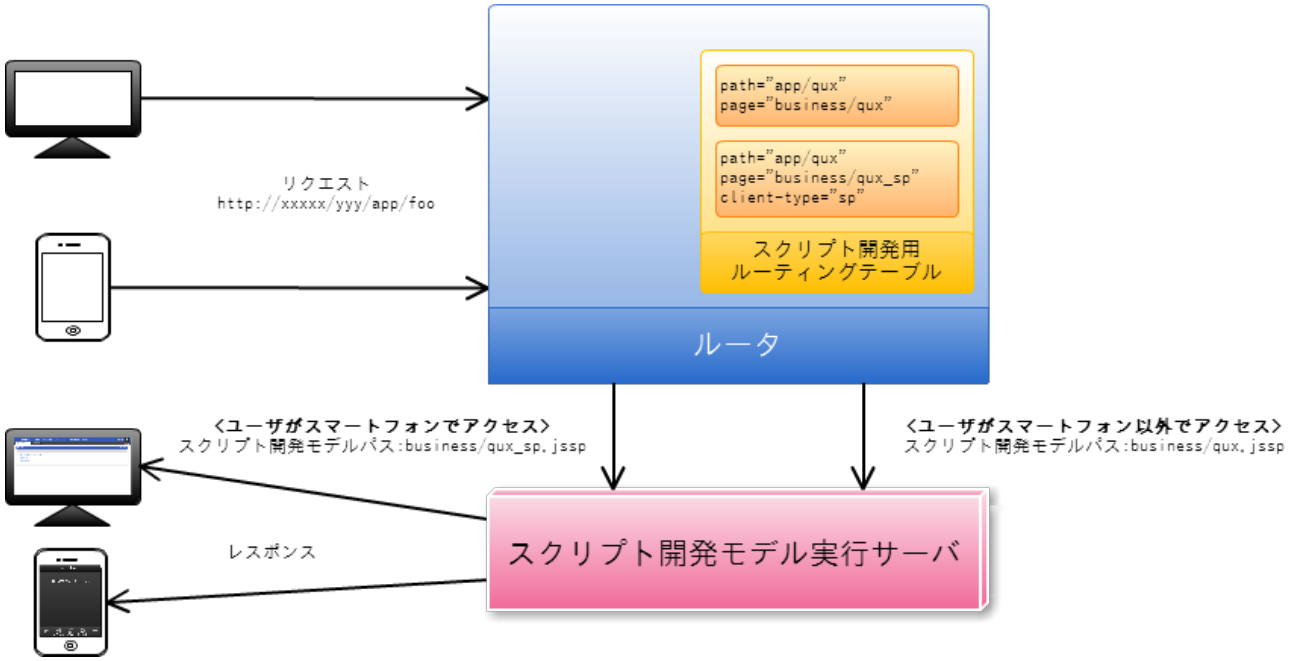



```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config
   xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-config routing-jssp-config.xsd">

   <authz-default mapper="welcome-all" />

   <file-mapping path="/app/foo" page="business/foo" />

   <file-mapping path="/app/baz" page="business/baz">
      <authz uri="service://app/baz" action="execute" />
   </file-mapping>

</routing-jssp-config>
```

authz-default要素にはwelcome-allリソースマッパーが指定されています。この認可リソースマッパーは開発中に使用するものであり、アプリケーションリリース時には適したuriとactionまたはmapperを指定することを強く推奨します。
/app/foo へのアクセスの場合、file-mapping要素内にauthz要素が存在しないため、デフォルト認可設定であるwelcome-allマッパーが認可設定として有効になります。
/app/baz へのアクセスの場合、file-mapping要素内にauthz要素が存在するため、uriがservice://app/baz、actionがexecuteが認可設定として有効になります。
authz-default要素、authz要素共に省略可能ですが、file-mappingに対して有効な認可設定が指定されていない場合は設定エラーとなります。
メニュー項目はアクセス先のURLに対して権限がある物のみ表示されます。
詳細については *認可* 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。
>
> - authz-default要素、authz要素を用いて、認可設定を行える。
> - URLに対して、認可設定が行われていない場合は設定エラーとなる。

クライアントタイプ



- *http://<HOST>:<PORT>/<CONTEXT_PATH>/app/qux* スマートフォンからアクセスされた場合に、 スクリプト開発モデル のパス business/qux_sp に割り当てその他の端末でアクセスされた場合に business/qux に割り当てるルーティングテーブルを作成します。
  - スマートフォン用プレゼンテーション・ページ
    %CONTEXT_PATH%/WEB-INF/jssp/src/business/qux_sp.html にファイルを作成します。
    ファイルを以下の内容にします。

    ```
    Smartphone.
    ```

  - スマートフォン以外からアクセス時のプレゼンテーション・ページの作成
    %CONTEXT_PATH%/WEB-INF/jssp/src/business/qux.html にファイルを作成します。
    ファイルを以下の内容にします。

    ```
    Not smartphone.
    ```

  - ファンクション・コンテナ

    処理が存在しないため、ファンクション・コンテナは作成しません。

  - ルーティングテーブル
    %CONTEXT_PATH%/WEB-INF/conf/routing-jssp-config/routing-programming-guide-qux.xml にファイルを作成します。
    ファイルを以下の内容にします。

    ```
    <?xml version="1.0" encoding="UTF-8"?>
    <routing-jssp-config
      xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-config routing-jssp-config.xsd">

      <authz-default mapper="welcome-all" />

      <file-mapping path="/app/qux" page="business/qux" />

      <file-mapping path="/app/qux" page="business/qux_sp" client-type="sp" />

    </routing-jssp-config>
    ```

    file-mapping要素でclient-type属性を指定することで、指定のクライアントタイプでのマッピング設定が行えます。
- 設定ファイルを反映させるために、アプリケーションサーバを再起動します。
- PCのブラウザで *http://<HOST>:<PORT>/<CONTEXT_PATH>/app/qux* へアクセスします
  ページに以下が表示されます

  ```
  Not smartphone.
  ```

- スマートフォンのブラウザで *http://<HOST>:<PORT>/<CONTEXT_PATH>/app/qux* へアクセスします

ページに以下が表示されます

```
Smartphone.
```

- サーバへアクセスできるスマートフォン端末がない場合は *http://<HOST>:<PORT>/<CONTEXT_PATH>/menu/sitemap* アクセス後の右上のユーティリティメニューの「スマートフォン版へ」を選択後に、URLにアクセスしてください。

> **コラム**
>
> この項目では、下記のポイントを確認しました。
>
> - file-mapping要素にclient-type属性を指定することで、特定のクライアントタイプでの スクリプト開発モデル のプログラムを指定することができる。

認可

項目
- 画面にアクセス権限を設定するために
- 認可とは
  - 認可の概要
  - サブジェクト
  - リソースとリソースグループ
  - リソースタイプとアクション
  - ポリシー
- 画面へアクセスするための認可設定方法
  - 権限設定の流れ
  - ステップ1：ルーティングテーブルに認可を紐づける
  - ステップ2：認可のリソースグループ、リソースを登録する
  - ステップ2-1：認可設定画面を利用して認可のリソースグループ、リソースを登録する
  - ステップ2-2：ジョブを利用して認可のリソースグループ、リソースを登録する
  - ステップ3：リソースに対して権限を設定する

画面にアクセス権限を設定するために

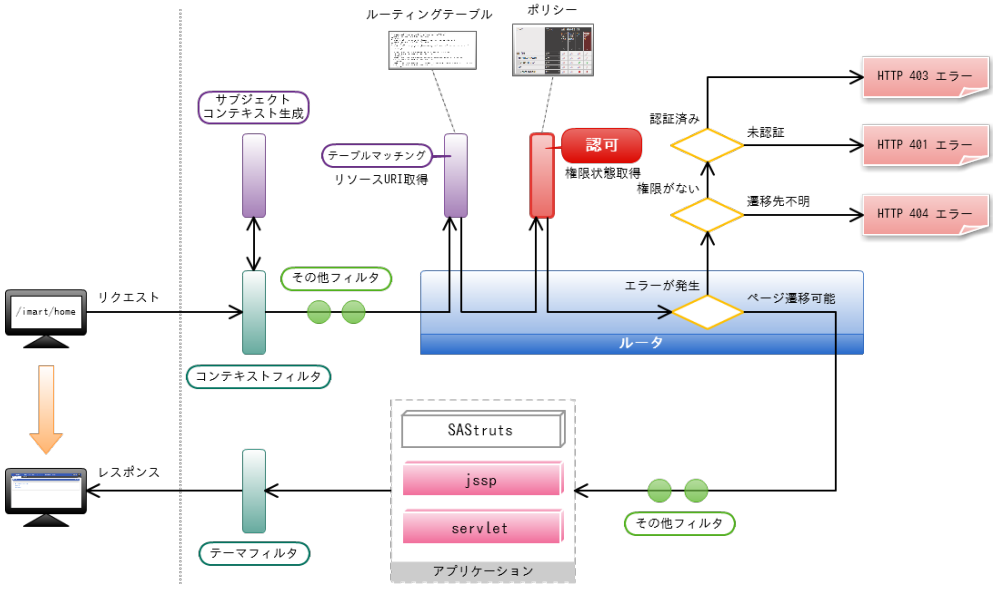
ルータの設定により画面へのアクセスが可能になりました。
しかし、ルーティングテーブルで welcome-all マッパーを使用すると認可処理が省略されるため、誰でも画面を表示することができます。

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。

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

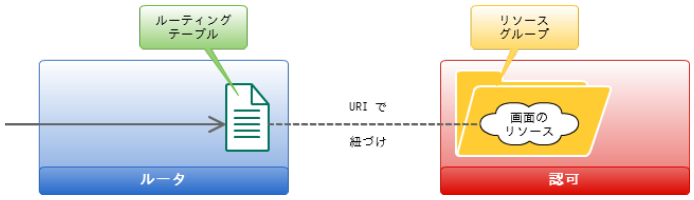
認可での 「ポリシー」 は、「許可」「禁止」 の部分を示します。
各サブジェクト・リソース・アクションの組み合わせ分、ポリシーを設定することができます。

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

ルータではルーティングテーブル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. ルーティングテーブルに認可を紐づける
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：ルーティングテーブルに認可を紐づける

認可に紐づけるため、ルーティングテーブルに authz タグを記述します。

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config
 xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-config routing-jssp-config.xsd">

 <file-mapping path="/app/foo" page="business/foo">
  <authz uri="service://sample/foo" action="execute" />
 </file-mapping>

</routing-jssp-config>
```

> **注意**
>
> 文字コードを UTF-8 にして保存してください。
> <authz-default mapper=”welcome-all” /> は削除してください。

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

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

ステップ2：認可のリソースグループ、リソースを登録する

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

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

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

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

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

「サイトマップ」→「テナント管理」→「認可」の順にクリックします。

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

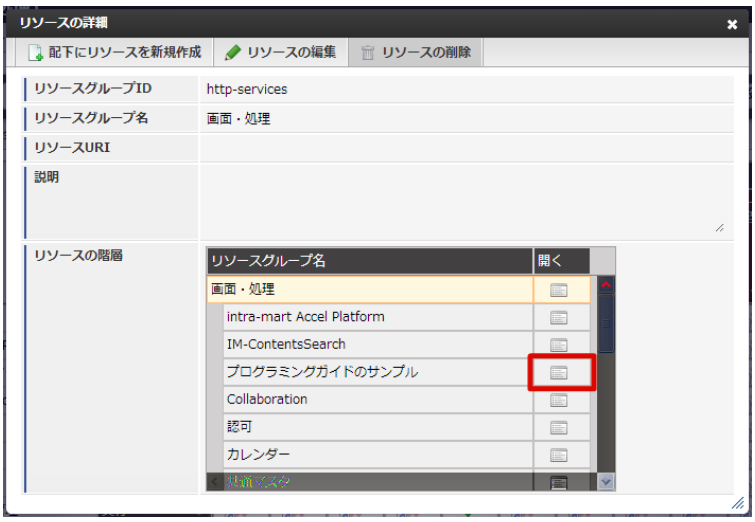
「リソースグループID」 のテキストボックスに 「guide-sample-service」 を入力します。
「リソースグループ名」 の最も上のテキストボックスに 「プログラミングガイドのサンプル」 を入力します。
「リソースURI」 と 「説明」 は未入力のままでかまいません。
「OK」 ボタンをクリックします。 これでリソースグループが登録されました。

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。
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。

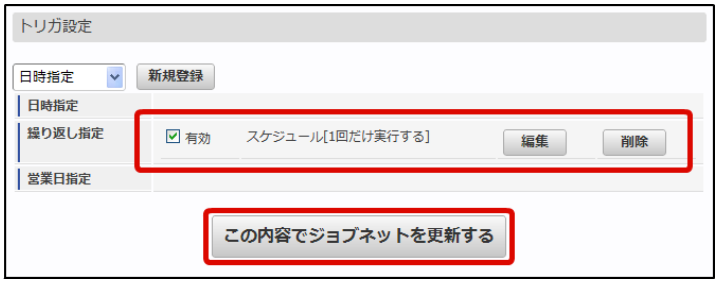
```
http://<HOST>:<PORT>/<CONTEXT_PATH>/login
```

「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>/app/foo
```

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>/app/foo
```

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
  - アカウントコンテキストの状態アクセスユーティリティ
  - 認証状況を問い合わせる
  - ユーザ種別を問い合わせる
- 定義済みアクセスコンテキストの利用方法
  - アカウントコンテキストを利用する
  - ユーザコンテキストを利用する
  - ユーザコンテキストの所属組織を切り替える

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＜短縮名＞()
```

アクセスコンテキストの情報はプロパティとして登録されますので、以下の方法で取得してください。

```
context.＜プロパティ名＞
```

以下にアクセス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。
var accountContext = Contexts.getAccountContext();

// ユーザコードを取得する。
var userCd = accountContext.userCd;

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

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
var 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。

```
var accountContext = Contexts.getAccountContext();
var authenticated = accountContext.authenticated;
```

- ユーティリティを利用して認証状況を問い合わせる。

```
var 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。

```
var accountContext = Contexts.getAccountContext();
var userType= accountContext.userType;

// システム管理者かどうか
var isAdministrator = "administrator" == userType;

// 一般ユーザかどうか
var isUser = "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。

```
var 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。

```
var accountContext = Contexts.getAccountContext();
var locale = accountContext.locale;

// メッセージを取得する。
var message = MessageManager.getLocaleMessage(locale, "I.IWP.CERTIFICATION.SECURITYLOG.00200")
Debug.print('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。
var message = MessageManager.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。

```
var userContext = Contexts.getUserContext();

// ユーザ名を取得します。
var userName = userContext.userProfile.userName;
```

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。

```
var userContext = Contexts.getUserContext();

var department = userContext.currentDepartment;
if (!isBlank(department)) {
  var departmentName = department.departmentFullName;
  Debug.print('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。

```
var resource = {
   currentCompanyCd : companyCd,
   currentDepartmentSetCd : departmentSetCd,
   currentDepartmentCd : departmentCd
};
Lifecycle.switchTo('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。

- Lifecycle.switchTo() メソッドの第1引数「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（resource）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。

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。

国際化

項目

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

概要

intra-mart Accel Platform が国際化に対応する理由

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

国際化対応の要素と理由

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

- 多言語

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

- タイムゾーン

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

- 日付と時刻の形式

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

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

国際化機能の仕様

多言語

項目

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

概要

多言語対応とは？

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

intra-mart Accel Platform の多言語対応

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

- 言語の定義

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

- メッセージの定義

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

- 言語解決

  ユーザは、使用したい言語を intra-mart Accel Platform に登録することができます。
  登録しない場合、自動的に解決された言語が適用されます。
  言語の解決順序については、*定義済みアクセスコンテキスト* を参照してください。

- メッセージの取得

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

言語の定義

初期状態の言語の定義

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

言語	言語 ID
英語	en
日本語	ja
中国語（簡体字）	zh_CN

- 言語マスタファイル

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

  - 設定ファイル

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

メッセージの定義

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

- メッセージコード

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

- 言語 ID

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

- 翻訳されたメッセージ

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

メッセージプロパティファイル

メッセージが記述されているファイルです。

- ファイルの命名規約

```
{任意の名前} _ {言語ID} .properties
```

  - 例

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

> **注意**
>
> {任意の名前} 部分にはアンダースコア(_)を入れないようにしてください。
>
> （例） test_message_ja.properties

- ファイルの内容

```
{メッセージコード} = {翻訳されたメッセージ}
```

  - 例

example_en.properties

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

example_ja.properties

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

タイムゾーン

項目

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

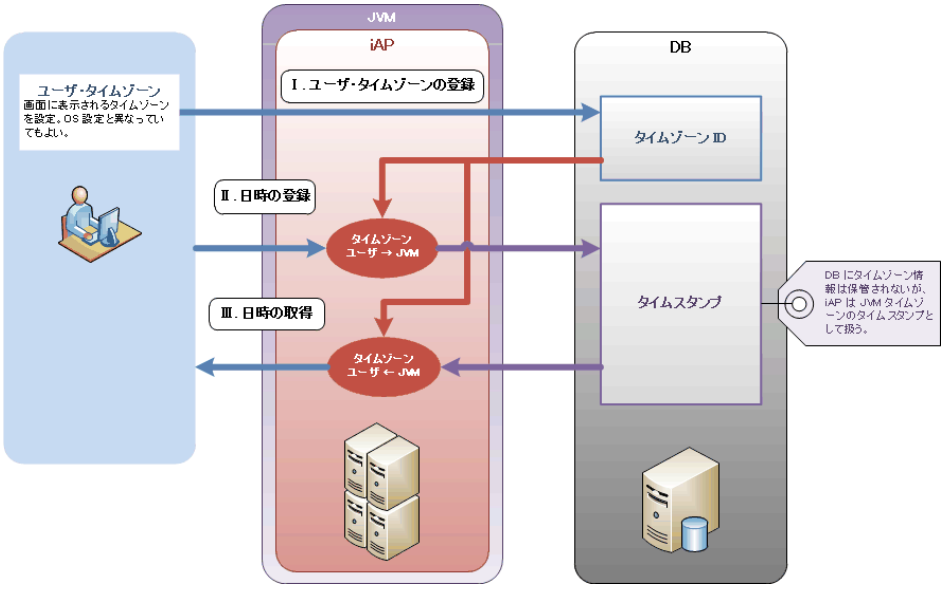
概要

タイムゾーン対応とは？

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

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

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



intra-mart Accel Platform のタイムゾーン対応

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

- タイムゾーンの定義

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

- 日時データを統一のタイムゾーンで変換して保存する

  日時データは、統一されたタイムゾーンで変換して DB に保存します。

- タイムゾーン解決

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

- 日時データをユーザのタイムゾーンで変換する

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

- 夏時間対応

タイムゾーンの定義

初期状態のタイムゾーンの定義

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

初期状態のタイムゾーンの定義

項目
- 初期状態のタイムゾーン一覧

初期状態のタイムゾーン一覧

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

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

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

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

- タイムゾーンマスタファイル

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

  - 設定ファイル

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

日時データを統一のタイムゾーンで変換して保存する

システム要件

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

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

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

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

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

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

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

日時データをシステム・デフォルトのタイムゾーンで変換する

ユーザのタイムゾーン、日付と時刻の形式を利用する を参照してください。

夏時間

夏時間とは？

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

intra-mart Accel Platform の夏時間対応

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

日付と時刻の形式

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

概要

日付と時刻の形式対応とは？

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

- （例）地域による標準的な日付表示形式の違い

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

intra-mart Accel Platform の日付と時刻の形式対応

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

- 日付と時刻の形式の定義

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

- 日付と時刻の形式の解決

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

- 日付と時刻の形式を使って、日時データを整形、解析する

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

日付と時刻の形式の定義

初期状態の日付と時刻の形式の定義

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

初期状態の日付と時刻の形式の定義

項目
- 概要
- 英語形式
- 日本語形式
- 中国語（簡体字）形式

概要

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

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

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

- （例）日本語形式（一部のフォーマットパターンのみを表示しています。）

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

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

英語形式

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

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

日本語形式

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

中国語（簡体字）形式

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

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

日付と時刻の形式マスタファイル

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

- 設定ファイル

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

解決された日付と時刻の形式で日時データを整形、解析する

画面から入力された日時データは、解決された「日付と時刻の形式」で解析されます。
画面には、ユーザの「日付と時刻の形式」で整形された日時データを表示します。
日時データを整形、解析する方法については、「タイムゾーン、日付と時刻の形式に関するサンプルプログラム」で説明します。

「言語」との違い

日付と時刻の形式は、言語とは別に登録することができます。

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

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

国際化プログラミングのサンプル

多言語化されたメッセージを取得する

項目

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

概要

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

MessageManager によるメッセージの取得

メッセージコードを指定してメッセージを取得する

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

```
var exampleMsg = MessageManager.getMessage('I18N.MESSAGE.EXAMPLE');
```

メッセージは、次のように自動解決されます。

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

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

メッセージコードと言語を指定してメッセージを取得する

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

```
var exampleMsg = MessageManager.getLocaleMessage('en', 'I18N.MESSAGE.EXAMPLE');
```

メッセージは、次のように自動解決されます。

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

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

タグによるメッセージの取得

MessageManager に対応するタグが提供されています。

動作は MessageManager と同じです。

メッセージコードを指定してメッセージを取得する

```
<IMART type="message" id="I18N.MESSAGE.EXAMPLE"></IMART>
```

メッセージコードと言語を指定してメッセージを取得する

```
<IMART type="message" id="I18N.MESSAGE.EXAMPLE" locale="en"></IMART>
```

ユーザのタイムゾーン、日付と時刻の形式を利用する

項目

- 前提
- 画面から送信された日時データを DB に保存する
  - 1. 画面から送信された日時文字列を解析する
  - 2. DB へ日時データを保存する
- 画面から送信された日付データを DB に保存する
  - 1. 画面から送信された日付文字列を解析する
  - 2. DB へ日付を保存する
- DB に保存されている日時データをユーザの画面に表示する
  - 1. DB から日時データを取得する
  - 2. 日時文字列に整形する
- DB に保存されている日付をユーザの画面に表示する
  - 1. DB から日付を取得する
  - 2. 日付文字列に整形する
- クライアント側で、ユーザ・タイムゾーンの今日から３日間の日付を生成し、最後の日付をサーバ側へ送信する
  - 1. ユーザ・タイムゾーンの今日を取得する
  - 2. 今日から３日間の日付を生成する
  - 3. 最後の日付をサーバ側へ送信する
  - 4. クライアント側から送信された日付文字列から Date を生成する

前提

- 各種設定値

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

- テーブル定義

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

画面から送信された日時データを DB に保存する

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

ユーザが以下の日時データを入力し、サーバに送信したとします。

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

1. 画面から送信された日時文字列を解析する

```
function init(request) {
    var inputDateStr = request.inputDate; //画面から送信された日時文字列
    var formatIds = [
        AccountDateTimeFormatter.IM_DATETIME_FORMAT_DATE_INPUT,
        AccountDateTimeFormatter.IM_DATETIME_FORMAT_TIME_INPUT
    ];
    var inputDate = AccountDateTimeFormatter.parseToDate(inputDateStr, formatIds);
}
```

日時文字列がユーザの日付と時刻の入力形式に沿っている場合、AccountDateTimeFormatter を使用します。
AccountDateTimeFormatter は、ログインユーザのタイムゾーンを使って解析を行います。

2. DB へ日時データを保存する

```
var db = new TenantDatabase();
var userCd = Contexts.getAccountContext().userCd;
Transaction.begin(function() {
    db.execute('INSERT INTO example_table values(?, ?)', [DbParameter.string(userCd), DbParameter.timestamp(inputDate)]);
});
```

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

画面から送信された日付データを DB に保存する

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

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

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

```
2012/09/19
```

1. 画面から送信された日付文字列を解析する

```
function init(request) {
    var inputDateStr = request.inputDate; //画面から送信された日付文字列
    var inputDate = AccountDateTimeFormatter.parseToDate(inputDate,
AccountDateTimeFormatter.IM_DATETIME_FORMAT_DATE_INPUT);
}
```

日付文字列の場合、時刻部分は「00:00:00」として解析されます。

2. DB へ日付を保存する

「DB へ日時データを保存する」と同様です。

DB に保存されている日時データをユーザの画面に表示する

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

1. DB から日時データを取得する

```
var db = new TenantDatabase();
var userCd = Contexts.getAccountContext().userCd;
var result = db.select('SELECT update_date FROM example_table WHERE user_cd=?', [DbParameter.string(userCd)]);
if (result.error) {
   return;
}
var outputDate = result.data[0];
```

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

2. 日時文字列に整形する

```
var formatIds = [
   AccountDateTimeFormatter.IM_DATETIME_FORMAT_DATE_STANDARD,
   AccountDateTimeFormatter.IM_DATETIME_FORMAT_TIME_STANDARD
];
var outputDateStr = AccountDateTimeFormatter.format(outputDate, formatIds);
```

ユーザの日付と時刻の表示形式で整形する場合、AccountDateTimeFormatter を使用します。
AccountDateTimeFormatter は、ログインユーザのタイムゾーンで時刻を計算します。
以下の結果が得られます。

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

DB に保存されている日付をユーザの画面に表示する

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

1. DB から日付を取得する

「DB から日時データを取得する」と同様です。

2. 日付文字列に整形する

```
var outputDateStr = AccountDateTimeFormatter.format(outputDate,
AccountDateTimeFormatter.IM_DATETIME_FORMAT_DATE_STANDARD);
```

「日時文字列に整形する」と同様です。
以下の結果が得られます。

```
Sep 19, 2012
```

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

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

1. ユーザ・タイムゾーンの今日を取得する

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

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

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

2. 今日から３日間の日付を生成する

Date に標準で用意されているメソッドを使用して構いません。

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

3. 最後の日付をサーバ側へ送信する

年月日の値から日付文字列を作ります。

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

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

> **注意**
>
> サーバ側でユーザ・タイムゾーンの日時データを生成する場合は、Date ではなく、DateTime を使用してください。
> Date はシステム・デフォルトのタイムゾーンで計算するため、ユーザ・タイムゾーンとの時差を考慮して扱う必要があります。
> DateTime はタイムゾーンを指定しない限り、ユーザ・タイムゾーンで計算します。
>
> ```
> //ユーザ・タイムゾーンにおける1996年9月19日3時47分の日時データ
> var dateTime = new DateTime(1996, DateTime.SEPTEMBER, 19, 3, 47, 0);
> ```

4. クライアント側から送信された日付文字列から Date を生成する

```
function init(request) {
   var inputDateStr = request.inputDate; //画面から送信された日付文字列
   var inputDate = DateTimeFormatter.parseToDate('yyyy-MM-dd', inputDateStr);
}
```

日時文字列がユーザの日付と時刻の入力形式に沿っていない場合、DateTimeFormatter を使用します。
DateTimeFormatter は、直接フォーマットパターンを指定することができます。
DateTimeFormatter は、ユーザのタイムゾーンを使って解析を行います。

データベース

項目
- データベースの種類
- プログラミング方法
  - トランザクション
  - クエリの実行
  - 外だしSQLの利用
    - 読み込まれるファイル
    - SQLファイルの記述例

データベースの種類

intra-mart Accel Platform では以下の3種類のデータベースを利用します。

- システムデータベース

  システムのデータを保存するデータベースです。
  システムデータベースはシステム内部で利用されるため、使用しないでください。

- テナントデータベース

  テナント内で利用するデータを保存するデータベースです。

- シェアードデータベース

  intra-mart Accel Platform システム外のデータを保存するデータベースです。
  外部システムと連携したい場合等に利用してください。

プログラミング方法

トランザクション

ここではトランザクション処理の実装方法を解説します。

トランザクションはTransactionオブジェクトのbeginメソッドに実行関数を渡すことで、処理終了時に自動的にコミットされます。

```
// トランザクション処理
Transaction.begin(function() {
  // クエリ実行
  var result = new TenantDatabase().execute(sql, param);
  if(result.error) {
    // エラー時はロールバックします。
    Transaction.rollback();
  }
});
```

上記のように実行関数を渡すことで、処理が正常に終了した場合はコミット処理、例外が発生した場合はロールバック処理が自動的に実行される為、トランザクションのコミット/ロールバックの実行漏れを防ぐことができます。

クエリの実行

クエリの実行方法は、接続先データベースによって異なります。

- テナントデータベース

```
// テナントデータベースへクエリを実行します。
var database = new TenantDatabase();
var result = database.execute("select * from b_m_account_b");
```

- シェアードデータベース

```
// 接続ID「sample」で設定されているシェアードデータベースへクエリを実行します。
var database = new SharedDatabase("sample");
var result = database.execute("select * from b_m_account_b");
```

prepared statementを利用する場合はDbParameterオブジェクトを使用します。

```
// prepared statementを使用してクエリを実行します。
var sql = "select * from b_m_account_b where user_cd = ? and login_failure_count = ?";
var result = new TenantDatabase().execute(sql, [
    DbParameter.string("aoyagi"),
    DbParameter.number(0)
]);
```

外だしSQLの利用

外だしSQLは、SQLをプログラムの外に出して、そのSQLをプログラムから実行する機能です。

外だしSQLを利用する場合は、「.sql」という拡張子のテキストファイルにSQLを記述し、「%CONTEXT_PATH%/WEB-INF/jssp/src」配下の任意のディレクトリに置いてください。

SQLファイル「%CONTEXT_PATH%/WEB-INF/jssp/src/sample/sql/get_sample_data.sql」を使用する場合は以下のように実装します。

```
// 外だしSQLを利用してクエリを実行します。
var database = new TenantDatabase();
var result = database.executeByTemplate('sample/sql/get_sample_data', params);
```

読み込まれるファイル

外だしSQLは接続先データベースのデータベース種別ごとに利用するSQLファイルをわけることができます。

データベース種別ごとにSQLファイルをわける場合は、ファイル名のサフィックスのデータベース種別を付けます。

- 例：データベース種別がOracleでSQLファイル名が「get_sample_data.sql」の場合
  - SQLファイル名を「get_sample_data_oracle.sql」とします。

上記のように、サフィックスをつけたSQLファイルを用意することでデータベース種別がOracleの場合、「get_sample_data_oracle.sql」利用され、それ以外のデータベースの場合「get_sample_data.sql」が利用されるようになります。

SQLファイルにつける各データベースのサフィックスは以下のとおりです。

データベース	サフィックス
Oracle	_oracle
SQLServer	_sqlserver
PostgreSQL	_postgre

SQLファイルの記述例

- **例 1 2Way-SQLの利用**

  2Way-SQLは、コメントでプログラムとのマッピングを書くことで、SQL*PlusなどのSQLのツールでもそのまま実行が可能となっているSQL文です。

  - SQLファイル「sample.sql」

    ```
    SELECT *
    FROM b_m_account_b
    WHERE user_cd = /*userCd*/'aoyagi'
    ```

  - プログラム「sample.js」

    ```
    new TenantDatabase().executeByTemplate('sample', {
        userCd : DbParameter.string("harada")
    });
    ```

  プログラムを実行するとuser_cd='harada'のデータを取得します。
  「sample.sql」をツール等でそのまま実行するとuser_cd='aoyagi'のデータを取得します。

  プログラム実行時は、WHERE句のコメント部分をprepared statementに置き換えて実行します。

SQLファイルのコメント部分（/*パラメータ名*/）に「:データ型」をつけるとDbParameterを使用せずにprepared statementを利用することができます。

- SQLファイル「sample.sql」

```
SELECT *
FROM b_m_account_b
WHERE user_cd = /*userCd:string*/'aoyagi'
```

- プログラム「sample.js」

```
new TenantDatabase().executeByTemplate('sample', {
    userCd : "harada"
});
```

- **例2 prepared statementの利用**

prepared statement (?を含むSQL) をSQLファイルに記述した場合、その出現順にプロパティ名を$1, $2, ..., $Nとしたオブジェクトを引数に指定します。

- SQLファイル「sample.sql」

```
SELECT *
FROM b_m_account_b
WHERE user_cd = ? AND login_failure_count = ?
```

- プログラム「sample.js」

```
new TenantDatabase().executeByTemplate('sample', {
    $1 : DbParameter.string('aoyagi'),
    $2 : DbParameter.number(0)
});
```

- **例3 文字列の置換**

SQLファイルに記述されたクエリ文字列を動的に置換してSQLを実行したい場合はコメント内に“$”をつけてパラメータ名を指定します。

- SQLファイル「sample.sql」

```
SELECT *
FROM b_m_account_b
ORDER BY /*$order*/'user_cd'
```

- プログラム「sample.js」

```
new TenantDatabase().executeByTemplate('sample', {
    order : "user_cd"
});
```

ORDER BY句のようにprepared statementを利用できない部分の文字列を置換したい場合等に利用します。

- **例4 条件分岐**

特定の条件によって実行するSQLを動的に変更したい場合は/*IF*/ /*END*/を利用します。

- SQLファイル「sample.sql」

```
SELECT *
FROM b_m_account_b
WHERE
    /*IF userCd != null*/
    user_cd LIKE /*userCd:string*/'aoyagi'
    /*END*/
    /*IF predicate()*/
    AND login_failure_count = /*failureCount:number*/0
    /*END*/
```

- プログラム「sample.js」

```
new TenantDatabase().executeByTemplate('sample', {
    userCd : "harada",
    failureCount : 0,
    predicate : function() {
        return true;
    }
});
```

条件分岐によってWHERE句が空白になる等、不正なSQLが実行される可能性がある場合は、WHERE句を/*BEGIN*/ /*END*/で囲みます。/*BEGIN*/ /*END*/を使うことで、以下の処理が行われ、不正なSQLが実行されるのを防ぐことができます。

1. すべての条件が不成立の場合、WHEREそのものをSQLから削除します。
2. はじめに成立した条件のクエリ文字列がANDで始まっている場合、ANDを削除します。

- SQLファイル「sample.sql」

```
SELECT *
FROM b_m_account_b
/*BEGIN*/
WHERE
    /*IF userCd != null*/
    user_cd LIKE /*userCd:string*/'aoyagi'
    /*END*/
    /*IF predicate()*/
    AND login_failure_count = /*failureCount:number*/0
    /*END*/
/*END*/
```

ログ

項目

- 概要
- Logger APIを利用する
  - ログレベル
  - Loggerオブジェクトを取得する
  - ログを出力する
- LoggerMDC APIを利用する
  - MDC
  - MDCを利用したログを出力する

概要

内部統制、セキュリティ確保や保守などの目的からログを出力します。
この項目では スクリプト開発モデル で行うログの実装について記述します。

Logger APIを利用する

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

```
function add(value1, value2) {
    // ロガーオブジェクトを取得
    // ロガー名を指定していないため、ファイル名がロガー名となる
    // ファイルの配置場所：%CONTEXT_PATH%/WEB-INF/jssp/src/foo/bar/logger_sample.js
    // ロガー名：'foo.bar.logger_sample'
    var logger = Logger.getLogger();
}
```

Logger.getLoggerメソッド()を利用してLoggerオブジェクトを取得します。引数に渡す文字列がロガーの名前となります。
引数に何も渡さない場合は、このメソッドを呼び出したJSファイルのソースパスを元に作成されます。具体的には、JSファイルのソースパスのファイル区切りを「.」に置き換えた文字列をロガー名とします。なお、JSファイルのソースパスとは、ソース検索ディレクトリからの相対パス（拡張子なし）を意味します。

ログを出力する

```
function add(value1, value2) {
    // ロガーオブジェクトを取得
    // ロガー名を指定していないため、ファイル名がロガー名となる
    // ファイルの配置場所：%CONTEXT_PATH%/WEB-INF/jssp/src/foo/bar/logger_sample.js
    // ロガー名：'foo.bar.logger_sample'
    var logger = Logger.getLogger();

    logger.debug('arguments=[{}, {}]', value1, value2);
    var result = value1 + value2;
    logger.trace('result={}', result);

    return result;
}
```

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

ログレベルdebugでadd(1, 2)を実行した場合は以下のような出力になります。

```
[DEBUG] f.b.logger_sample - arguments=[1, 2]
```

ログレベルtraceでadd(1, 2)を実行した場合は以下のような出力になります。

```
[DEBUG] f.b.logger_sample - arguments=[1, 2]
[TRACE] f.b.logger_sample - result=3
```

LoggerMDC APIを利用する

MDC

Mapped Diagnostic Context（マップ化された診断コンテキスト）を利用することにより、ログ設定ファイルのレイアウト設定で独自に定義したkeyで保存した情報をログに出力することが可能となります。
LoggerMDC APIを利用することにより、独自に定義したkeyへの情報の書き込みが可能となります。

MDCを利用したログを出力する

```
// MDCのキーを定義
var MDC_FUNC_KEY = 'application.func';
function add(value1, value2) {
    // ロガーオブジェクトを取得
    // ロガー名を指定していないため、ファイル名がロガー名となる
    // ファイルの配置場所：%CONTEXT_PATH%/WEB-INF/jssp/src/foo/bar/logger_sample.js
    // ロガー名：'foo.bar.logger_sample'
    var logger = Logger.getLogger();

    // MDCに値を設定
    LoggerMDC.put(MDC_FUNC_KEY, 'add');

    logger.debug('arguments=[{}, {}]', value1, value2);
    var result = value1 + value2;
    logger.trace('result={}', result);

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

    return result;
}
```

実行中の関数名をMDCに設定しています。
LoggerMDC.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LoggerMDC.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] f.b.logger_sample - add arguments=[1, 2]
[TRACE] f.b.logger_sample - 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

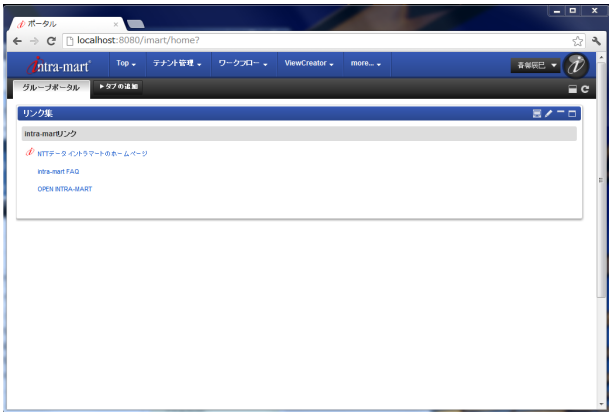
項目

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

クライアントタイプとスマートフォン版テーマ

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

- PCブラウザからHOME画面にアクセスした例



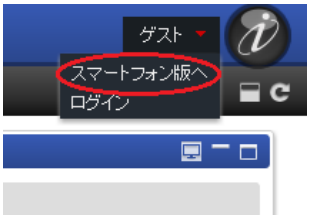
- スマートフォンブラウザからHOME画面にアクセスした例。スマートフォン版テーマが適用されている

クライアントタイプの変更

クライアントタイプはユーザが明示的に変更することもできます。

PCブラウザからスマートフォン版画面を表示する

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



スマートフォンからPC版画面を表示する

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

スマートフォン版テーマ

項目

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

スマートフォン版テーマとは

スマートフォン版テーマとは、スマートフォン用に最適化されたテーマのことを指します。
<HTML>タグや<HEAD>タグなど冗長的な記述を省き、予め必要なライブラリ群をロードした環境下で開発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<imart type=”head”>タグで定義し、その他はBODYタグの内部のみ記述します。

```
<imart type="head">
 <title>My Page</title>
</imart type="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%/WEB-INF/jssp/src/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* 配下の画面へアクセスすると、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。

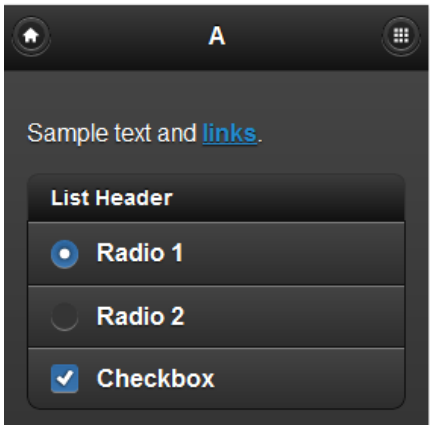
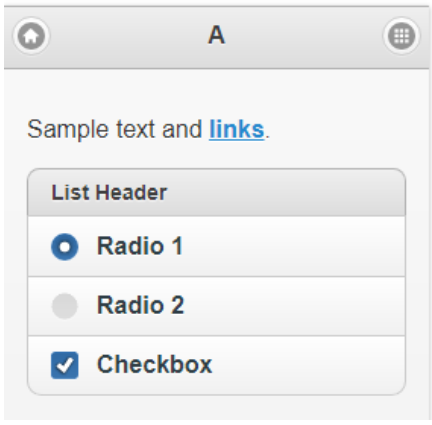
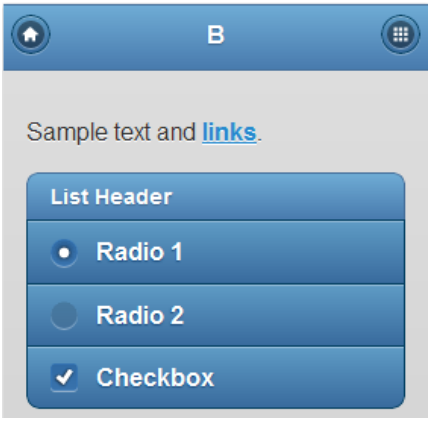
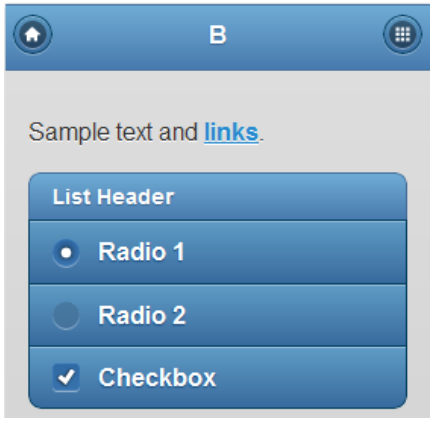
- デフォルトテーマが c から a 変更になったため、data-theme を指定していない画面は a（黒色）のテーマが適用されるようになった
- フラットデザインになったため、ボーダーやグラデーションがほとんどなくなった

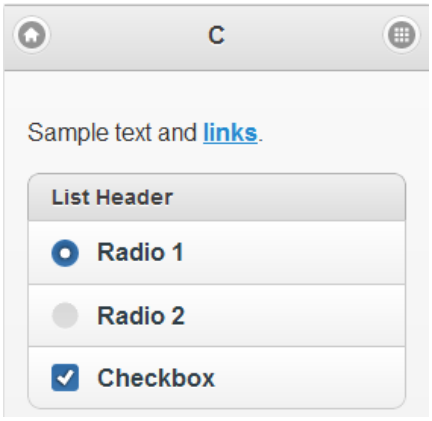
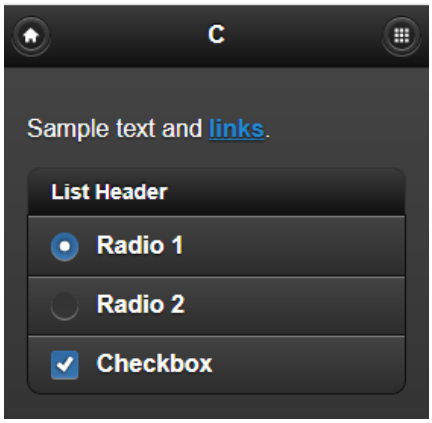
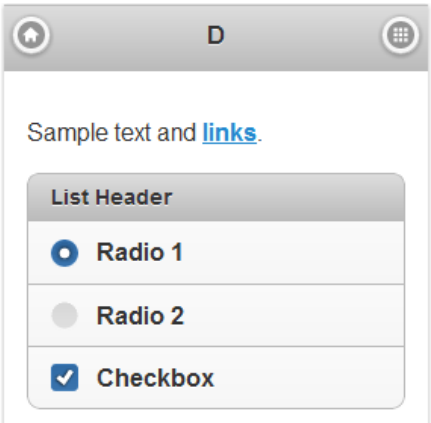
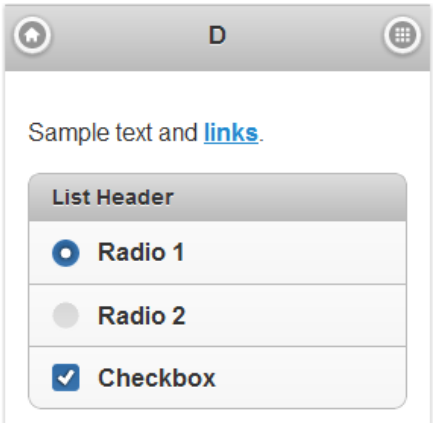
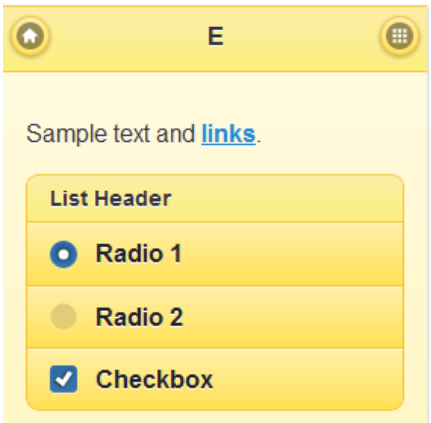
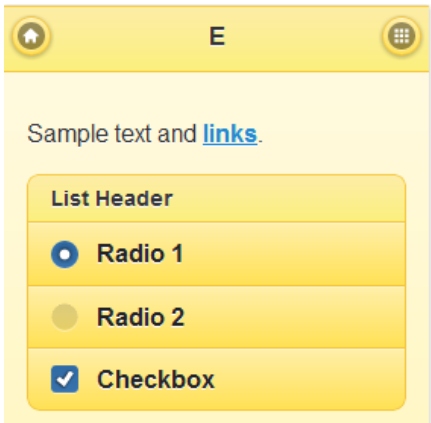
> **コラム**
> その他の変更点や変更点の詳細は http://jquerymobile.com/upgrade-guide/1.4 を参照してください。

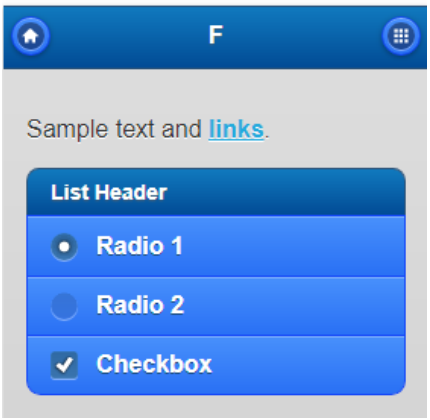
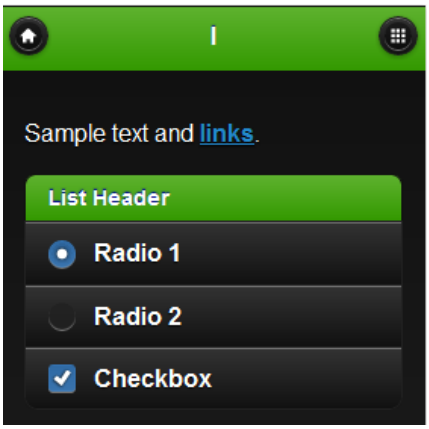
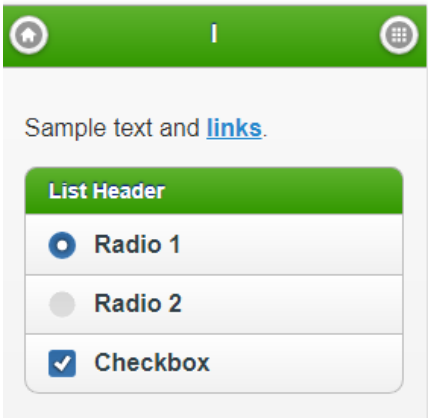
> **コラム**
> 2018 Summer(Tiffany) より標準テーマ白が追加されました。既存の標準テーマ黒を利用している場合とスウォッチの色が異なります。
> 各テーマのスウォッチについては下記を参照してください。

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			

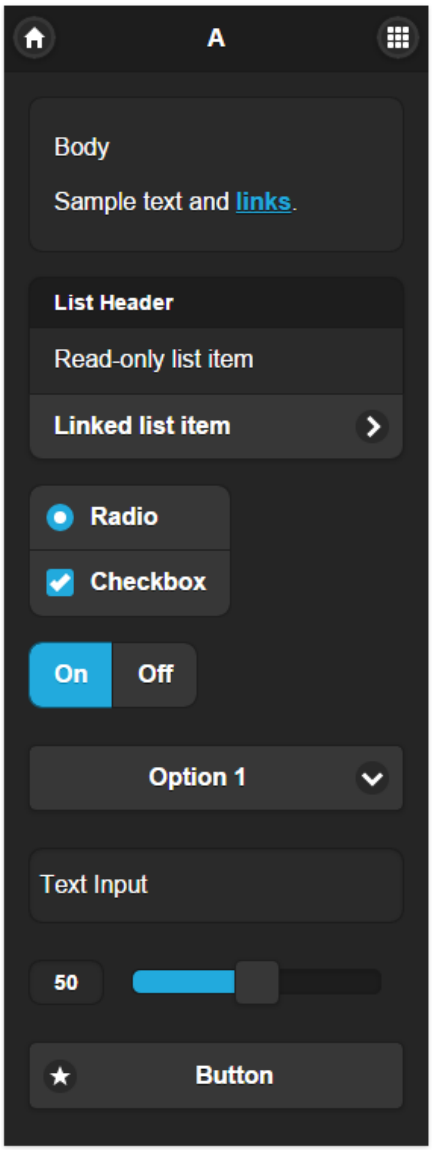
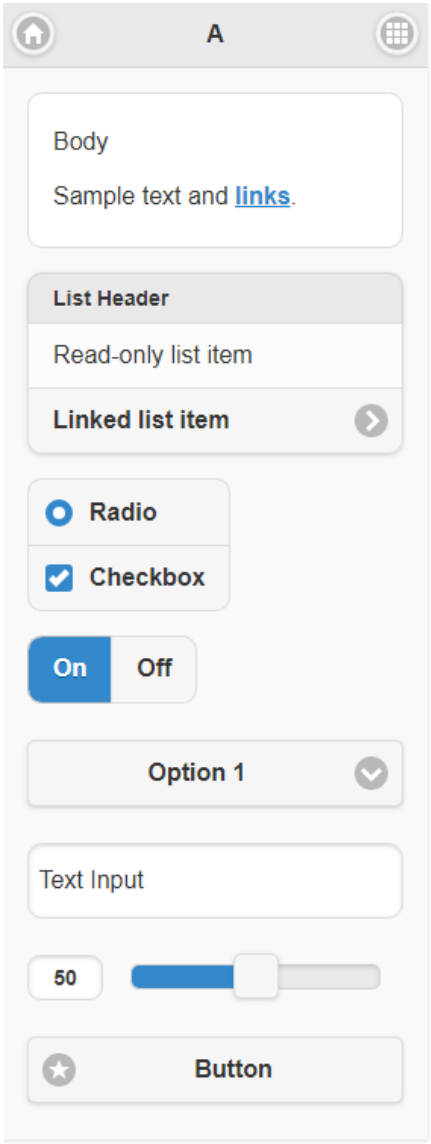
設定	説明	黒テーマイメージ	白テーマイメージ
C	jQueryMobile標準色		
D			
E			

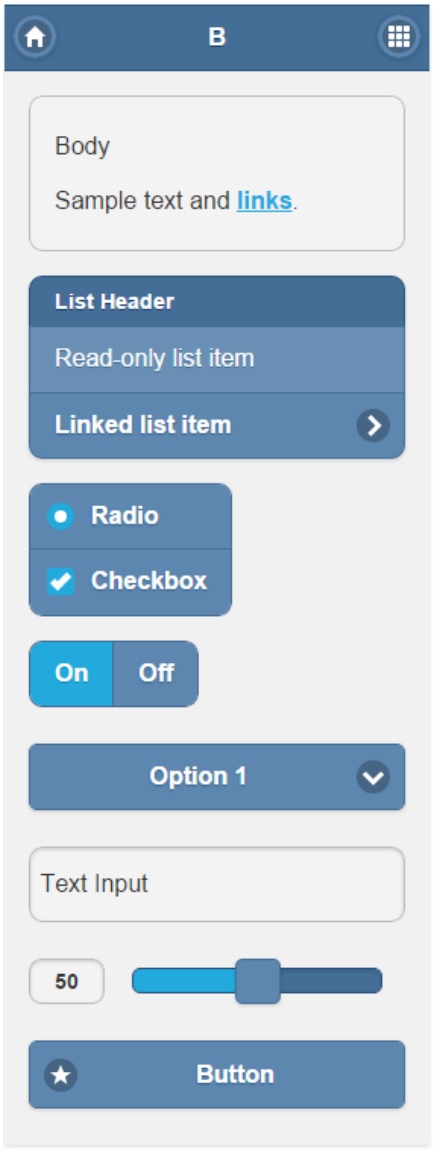
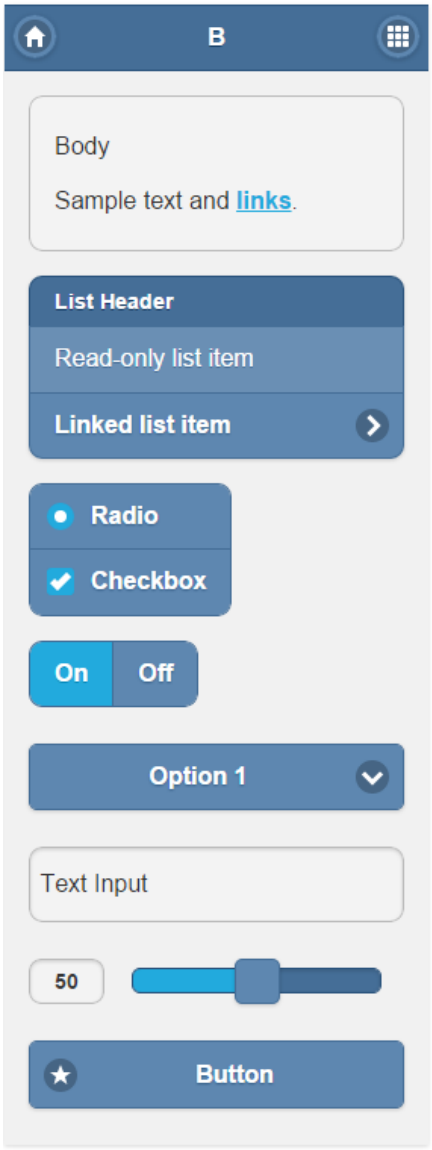
設定	説明	黒テーマイメージ	白テーマイメージ
F	追加スウォッチ		
I	追加スウォッチ		

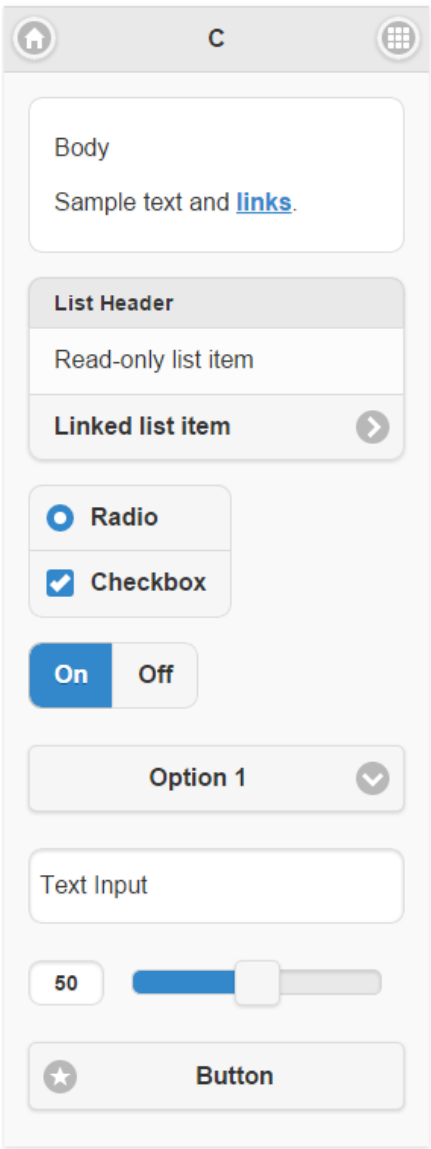
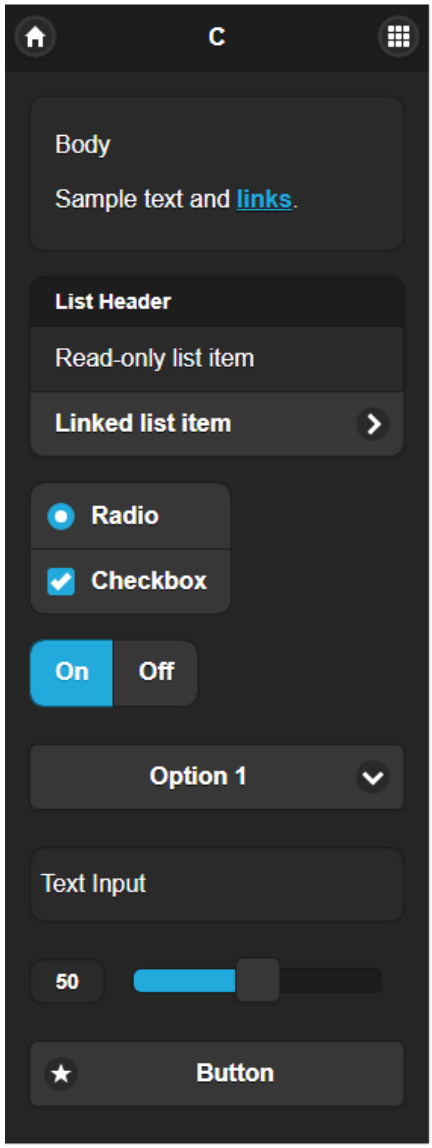
jQuery Mobile 1.4.5

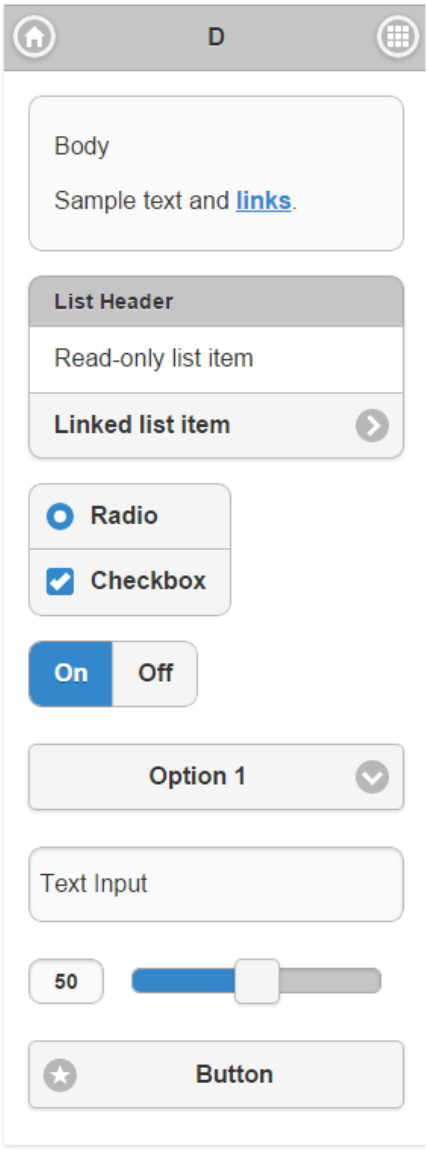
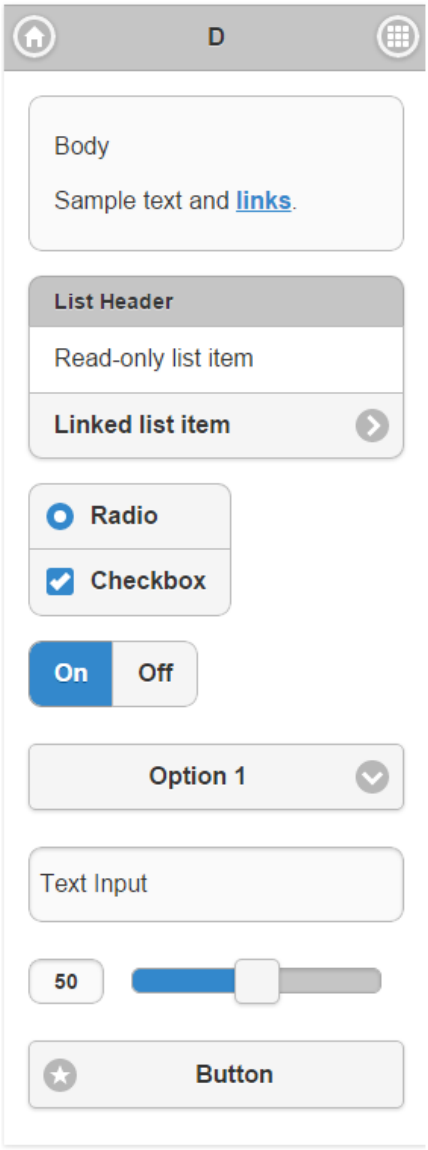
intra-mart Accel Platform では、jQuery Mobileデフォルトスウォッチの「A」～「E」のほかに、拡張スウォッチ「F」「I」を使用することができます。
「A」が jQueryMobile標準色へ変更になりました。

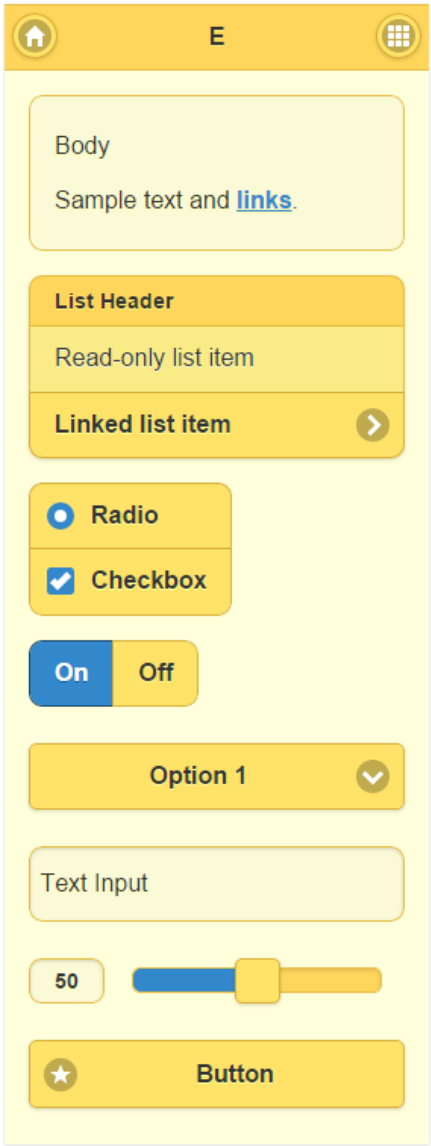
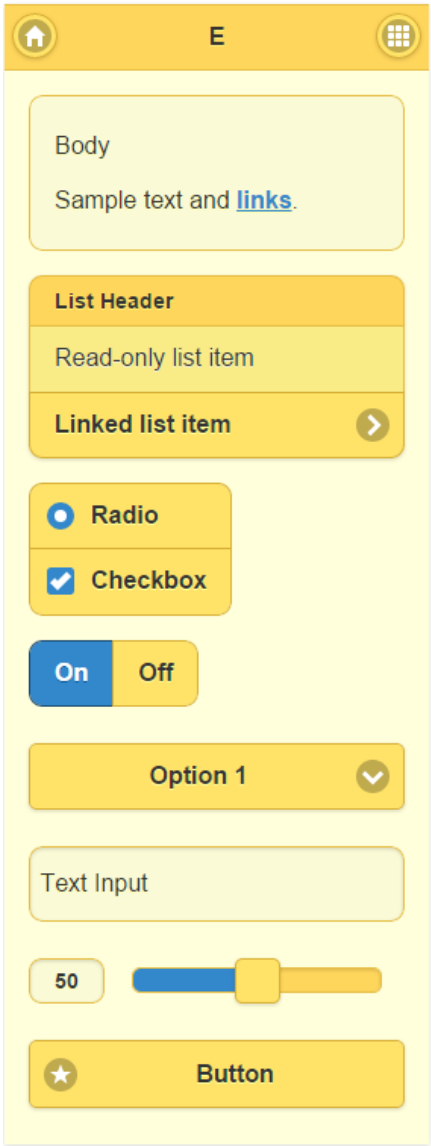
設定	説明	黒テーマイメージ	白テーマイメージ

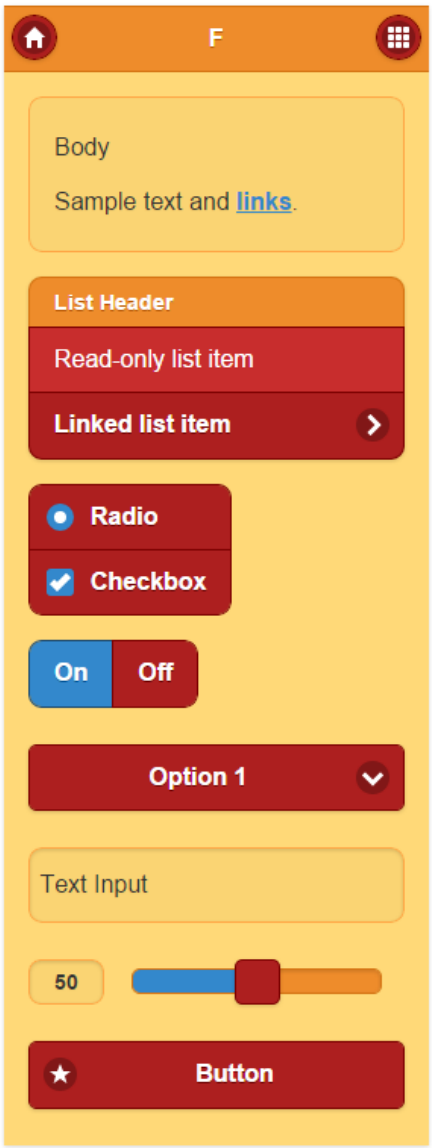
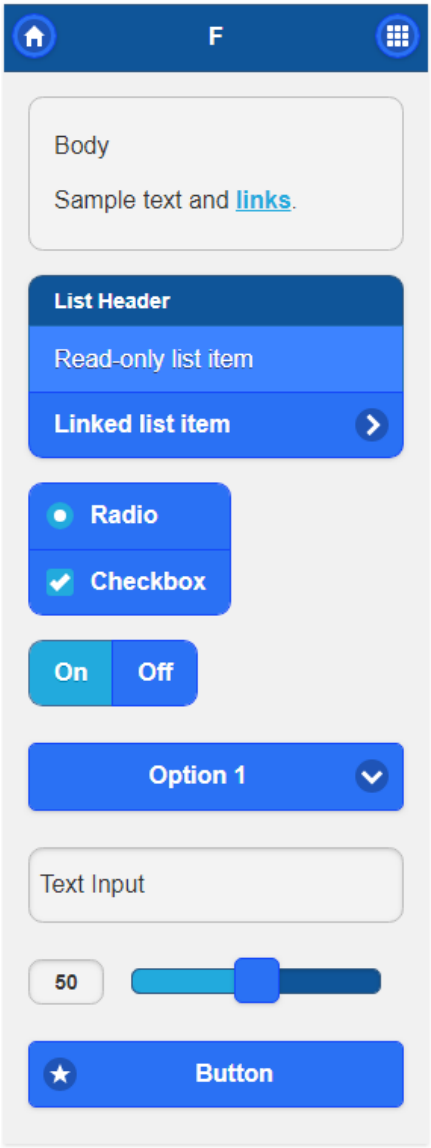
設定	説明	黒テーマイメージ	白テーマイメージ
A	jQueryMobile、intra-mart Accel Platform 標準色		

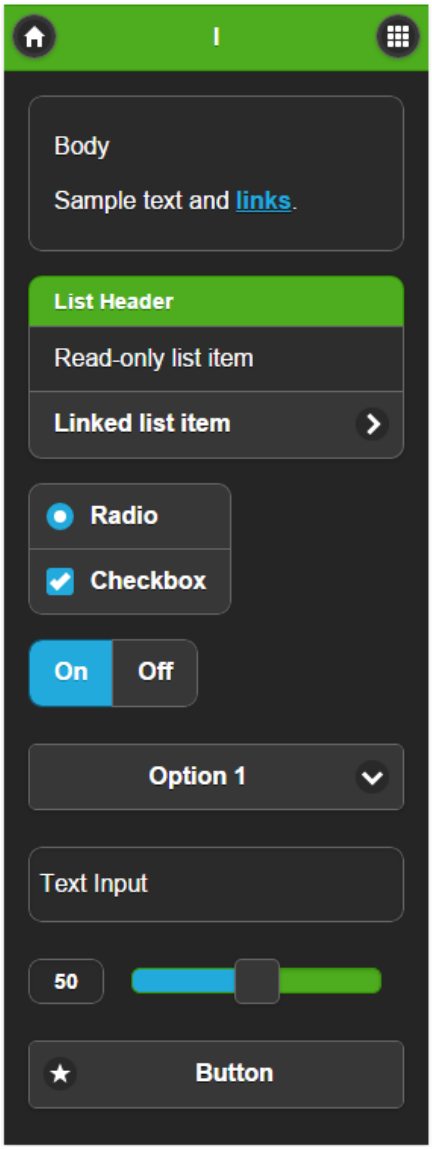
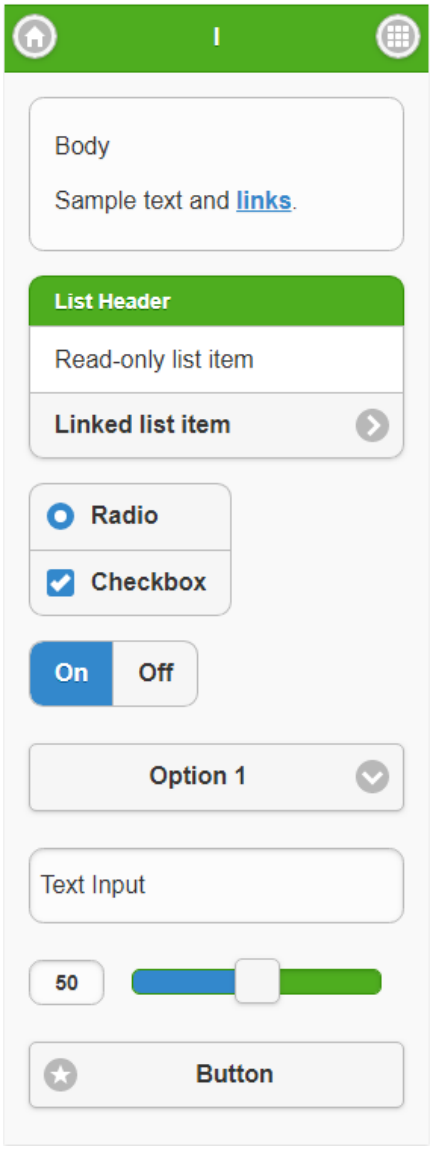
設定	説明	黒テーマイメージ	白テーマイメージ
B			

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
  - HTMLファイルを用意する
  - ライブラリ群の設定用ファイルを用意する
  - ルーティングXMLファイルを用意する
  - マークアップの説明
- スウォッチを指定する

Hello IM-Mobile Framework!を作る

HTML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/jssp/srcフォルダに保存します。

- hello_mfw.html

```
<div data-role="page">
 <div data-role="header">
  <h3>Header</h3>
 </div>
 <div role="main" class="ui-content">
  <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">/mobile_fw/.*</path>
</theme-full-theme-path-config>
```

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

ルーティングXMLファイルを用意する

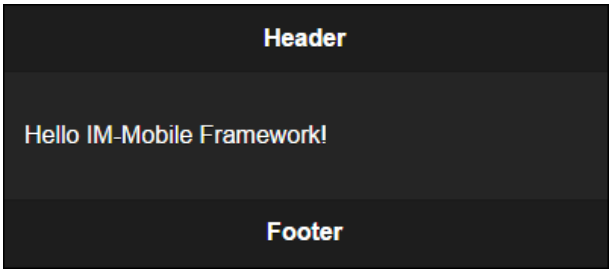
以下のファイルを作成し、%CONTEXT_PATH%/webapps/imart/WEB-INF/conf/routing-jssp-configフォルダに保存します。

- im_mobile_fw_test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-config routing-jssp-config.xsd ">
 <authz-default mapper="welcome-all" />
 <file-mapping path="/mobile_fw/hello" page="hello_mfw" />
</routing-jssp-config>
```

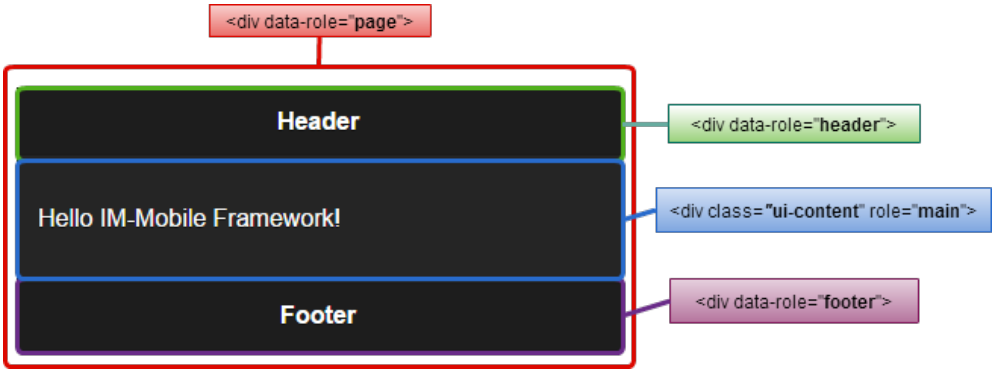
> **注意**
>
> サンプルで使用している `<authz-default mapper="welcome-all" />` の記述は認可の設定がない場合、デフォルトで全てのユーザがアクセス可能にする設定です。
> サンプルでは説明省略のため使用していますが、本番環境などで使用することは控え、認可を必ず設定するようにしてください。
> 認可の設定については *認可* を参照してください。

- サーバを再起動し、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>/mobile_fw/hello`



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

マークアップの説明



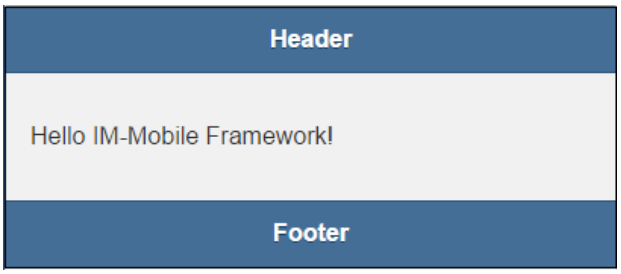
-
  1ページ当たりのブロック要素であることを定義します。
  最も上位のブロック要素であり、以下３つの要素は必ずこの要素内に定義する必要があります。
-
  ヘッダのブロック要素であることを定義します。
  任意の要素です。
-
  コンテンツのブロック要素であることを定義します。
  必須の要素です。
  でも動作しますが、推奨されていません。
-
  フッタのブロック要素であることを定義します。
  任意の要素です。

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
  - 下準備 テーブル作成
- 画面を表示できるようにする
  - ソースの準備と配置
  - jqueyMobile1.4.5を読み込むようにする
  - メニューから遷移できるようにする
  - メニューの作成
  - 認可の設定
  - 画面を表示する
- 画面に要素を配置する
- 登録処理を実装する
- 入力チェック処理を実装する（サーバサイド）
- 入力チェック処理を実装する（クライアントサイド）
- 非同期で登録処理を実行する
- 最終結果

前提条件

- intra-mart Accel Platform をインストールし、初期設定までが完了していること。
- ベースモジュールにスクリプト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。

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_store.js

```
function init(request) {
}
```

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_store.html

```
<imart type="head">
  <title>TODO登録</title>
</imart>
<div data-role="page" id="main">
  <imart type="spHeaderWithLink" headerText="TODO登録" />
  <div class="ui-content" role="main">
  </div>
  <imart type="spCommonFooter" dataPosition="fixed" />
</div>
```

- spHeaderWithLink - 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。
- 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 /sample/sp 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">/sample/sp/.*</path>
</theme-full-theme-path-config>
```

メニューから遷移できるようにする

次にルーティングの設定を行います。

- %CONTEXT_PATH%/WEB-INF/conf/routing-jssp-config/im_mobile_sample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-config routing-jssp-config.xsd ">
  <authz-default mapper="welcome-all" />
  <file-mapping path="/sample/sp/store" client-type="sp" page="/sample/mobile_fw/sp_store" />
</routing-jssp-config>
```

コラム

- ルーティングの詳細については *ルーティング* を参照してください。
- file-mapping要素にclient-type=”sp”属性を付加するとクライアントタイプがスマートフォンの場合のみ使用可能なルーティングを設定できます。

メニューの作成

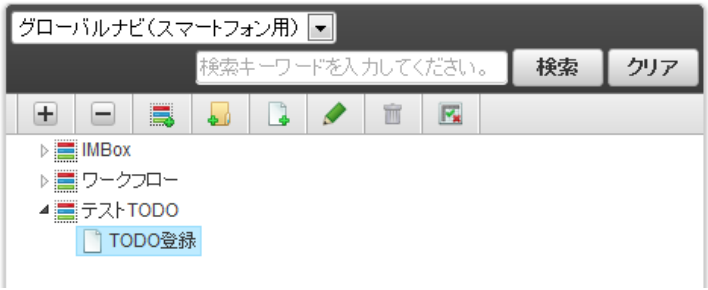
再起動してメニューの設定を行います。

- PCブラウザからテナント管理者でログインし、「メニュー設定」画面を表示します。

- グローバルナビ（スマートフォン用）を選択し、新規メニューグループ「テストTODO」を作成します。

新規メニューアイテムを作成します。

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



認可の設定

認可を設定します。

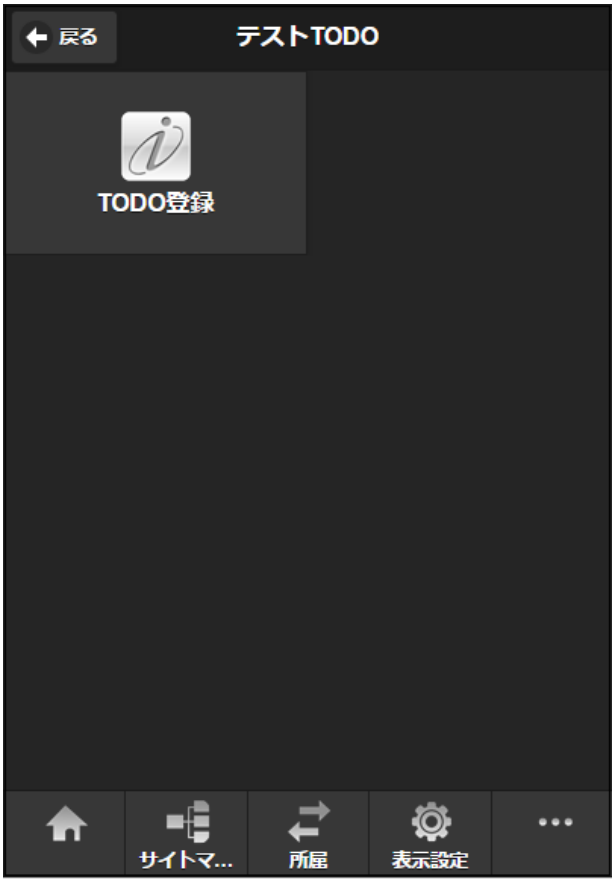
- 「権限設定」ボタンをクリックし権限設定（グローバルナビ（スマートフォン用））を表示します。
- 「権限設定を開始する」ボタンをクリックします。
- テストTODOの権限の「参照」権限を認証済みユーザに付与します。

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

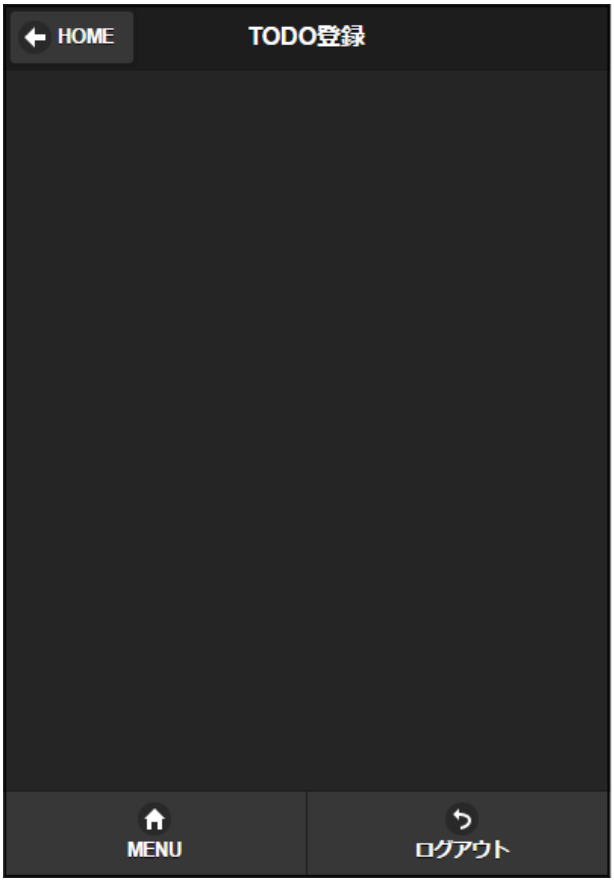
画面を表示する

作成したプレゼンテーション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。

- ファイルマッピングにclient-type=”sp”属性を付加することでスマートフォン用のルーティングが設定できる
- 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<imart type=”spFieldContain”>タグを使用します。

- ファンクションコンテナ（HTML）

```
<imart type="spFieldContain" label="TODO名" required="true">
  <imart type="input" style="text" name="title" value=title />
</imart>
```

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

同様に他の要素も配置していきます。

- spDatePicker–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。

```
<imart type="spFieldContain" label="期限" required="true">
  <imart type="spDatePicker" name="limit_date" value=limit_date />
</imart>
```

- textarea–テキストエリアを提供します（PC版共通）。

```
<imart type="spFieldContain" label="コメント">
  <imart type="textarea" name="comment" value=comment />
</imart>
```

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

```
<imart type="spControlGroup" label="重要度">
  <imart type="spRadioButton" name="priority" id="radio1" value="0" label="低"></imart>
  <imart type="spRadioButton" name="priority" id="radio2" value="1" label="中"></imart>
  <imart type="spRadioButton" name="priority" id="radio3" value="2" label="高"></imart>
</imart>
```

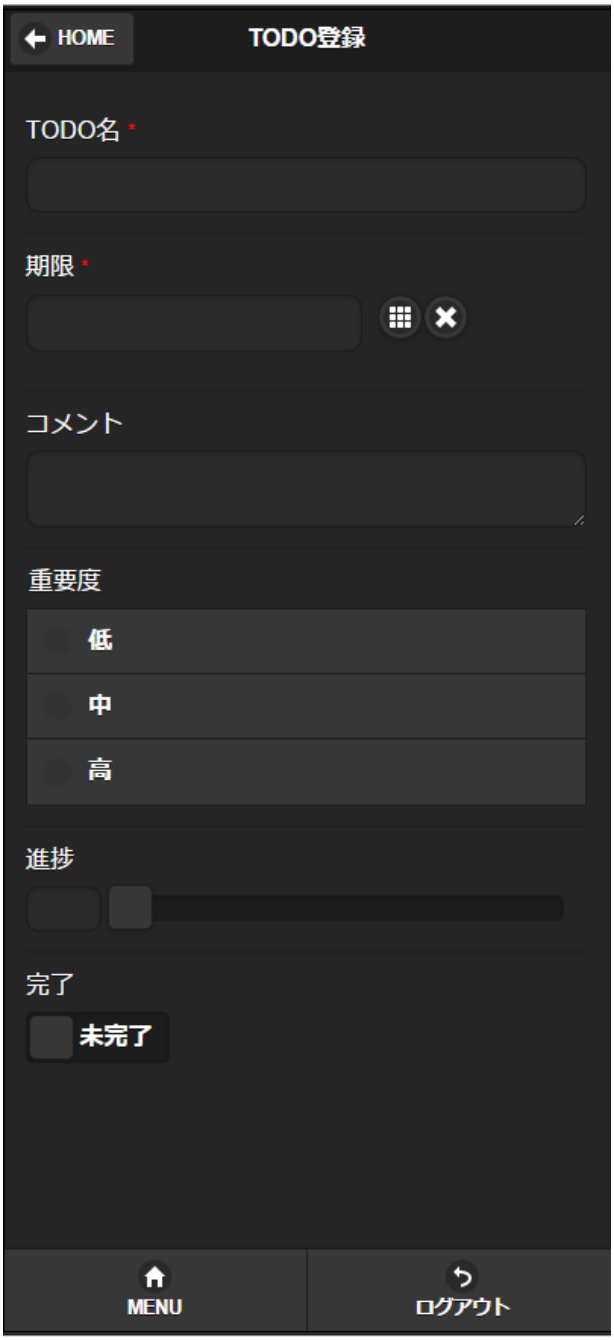
- spSlider–スライダーを提供します。

```
<imart type="spFieldContain" label="進捗">
  <imart type="spSlider" name="progress" value=progress min="0" max="100" />
</imart>
```

- spToggleSwitch–トグルスイッチを提供します。

```
<imart type="spFieldContain" label="完了">
  <imart type="spToggleSwitch" name="complete" selected=complete onLabel="完了" offLabel="未完了" onValue="1"
offValue="0" />
</imart>
```

- 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<imart type=”spFieldContain”>タグを使用する

登録処理を実装する

登録処理用のファンクションコンテナを新規作成します。

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_store_do.js

```
function init(request) {
  //ユーザプロファイル情報を取得します。
  var userProfile    = Contexts.getUserContext().userProfile;

  //登録データを作成します。
  var insertObject = {
    id:Identifier.get(),              //レコードのユニークID
    user_cd:userProfile.userCd,       //登録ユーザCD
    user_nm:userProfile.userName,     //登録ユーザ名
    title:request.title,              //TODOのタイトル
    limit_date:request.limit_date,    //TODOの期日
    comment:request.comment,          //コメント
    progress:parseInt(request.progress),//進捗度
    complete:request.complete,        //完了or未完了
    priority:request.priority,        //重要度
    timestmp:DateTimeFormatter.format("yyyy/MM/dd HH:mm", new Date()) //タイムスタンプ(yyyy/MM/dd HH:mm)
  };

  //トランザクションを開始します。
  Transaction.begin(function() {
    // データの保存を行います
    var result = new TenantDatabase().insert("mfw_sample", insertObject);
    if(result.error){
      //エラー時、ロールバックしエラー画面へ遷移します。
      Transaction.rollback();
      Transfer.toErrorPage({
        title: 'エラー',
        message: 'データ登録時にエラーが発生しました。',
        detail: result.errorMessage
      });
    }
  });

  //画面を再表示します。
  forward("sample/mobile_fw/sp_store", request);
}
```

> **コラム**
>
> - データベースを使用したプログラミングの詳細については *データベース* を参照してください。

- im_mobile_sample.xmlに登録処理用のルーティングを追加します。

```
<file-mapping path="/sample/sp/store/insert" client-type="sp" page="/sample/mobile_fw/sp_store_do" />
```

- 画面にFormタグと「登録」ボタンを配置します。 formタグのaction属性は先ほど設定したルーティングのパスと同様にします。

```
<div class="ui-content" role="main">
  <form id="storeForm" name="storeForm" method="POST" action="sample/sp/store/insert" data-ajax="false">
    <imart type="spFieldContain" label="TODO名" required="true">
      <imart type="input" style="text" name="title" value=title />
    </imart>
    ...
    <imart type="spFieldContain" label="完了">
      <imart type="spToggleSwitch" name="complete" selected=complete onLabel="完了" offLabel="未完了" onValue="1"
offValue="0" />
    </imart>
    <a class="ui-btn ui-btn-b ui-corner-all" id="storeButton">登録</a>
  </form>
</div>
```

> **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。
>
> - フォームを送信するにはformタグのaction属性にルーティングのパスを与える

入力チェック処理を実装する（サーバサイド）

例としてタイトルと日付の入力チェックを実装します。

> **コラム**
>
> 以下のメッセージがある前提として説明します。
> `CAP.Z.IWP.MFW.SAMPLE.TITLE=タイトル`
> `CAP.Z.IWP.MFW.SAMPLE.LIMIT_DATE=期限`
>
> メッセージプロパティは各環境に合わせて設定してください。
> メッセージプロパティの詳細については *多言語* を参照してください。

バリデーションルールを定義するため、以下のファイルを作成します。

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/validator.js

```
var validateRule = {
  "title": {
   caption:"CAP.Z.IWP.MFW.SAMPLE.TITLE",
   required:true,
   maxlength:20
  },
  "limit_date": {
   caption:"CAP.Z.IWP.MFW.SAMPLE.LIMIT_DATE",
   required:true,
   date:true
  }
};
```

- sp_store_do.jsのinit関数にアノテーションを追加します。
  `@validate` には利用するバリデーションルールのパスを``@onerror`` には入力チェックエラー時に実行する関数名を指定します。

```
/**
 * @validate sample/mobile_fw/validator#validateRule
 * @onerror handleErrors
 */
function init(request) {
```

- 入力チェックエラー時に実行する関数「handleErrors」をsp_store_do.jsに追加します。

```
function handleErrors(request, validationErrors) {
 Transfer.toErrorPage({
     title: 'エラー',
     message: '入力チェックエラーが発生しました。',
     detail: validationErrors.getMessages()
  });
}
```

- マークアップ結果。タイトルと日付未入力の状態で「登録」ボタンをクリックすると、
  登録処理が呼出されず入力チェックエラー処理が呼出され、エラー画面へ遷移します。

> **コラム**
> JSSP Validatorの詳細については *JSSP Validator* 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。
>
> - 入力チェックのルールはバリデーションルールで定義する
> - 入力チェックを実装するにはチェック対象の関数にアノテーションを定義する

入力チェック処理を実装する（クライアントサイド）

sp_store.htmlにimuiValidationRuleタグを追加します。
rule属性に *入力チェック処理を実装する（サーバサイド）* で作成したvalidate.jsのパスを、
rulesName属性とmessagesName属性にそれぞれクライアントJS内で使用する
オブジェクト名を定義します。

- プレゼンテーションページ（HTML）

```
<div data-role="page" id="main">
  <imart type="imuiValidationRule" rule="sample/mobile_fw/validator#validateRule" rulesName="rules"
messagesName="messages" />
  <script>
    (function($){
    ...
```

「登録」ボタン押下時のスクリプト処理を修正します。

- プレゼンテーションページ（HTML）

```
<script>
 (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」ボタンをクリックすると、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、 *エラー処理* 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 <imart type=”imuiValidationRule”>タグを定義する
> - 入力チェックを実行するにはimspValidate関数を使う

非同期で登録処理を実行する

sp_store_do.jsのinit関数の処理を一部修正します。

- ファンクションコンテナ（サーバサイドJavaScript）

```
var resultObject;

Transaction.begin(function() {
    // データの保存を行います
    var result = new TenantDatabase().insert("mfw_sample", insertObject);
    if(result.error){
        Transaction.rollback();
        //Transfer.toErrorPage({
        //   title: 'エラー',
        //   message: 'データ登録時にエラーが発生しました。',
        //   detail: result.errorMessage
        // });

        resultObject = {
          error:true,
          errorMessage:"データ登録時にエラーが発生しました。",
          detailMessages:["管理者にお問い合わせください。"]
        };
    } else {
        resultObject = {
           error:false,
           errorMessage:"",
           successMessage:"登録が完了しました。"
        };
    }
});

var response = Web.getHTTPResponse();
response.setContentType('application/json; charset=utf-8');
response.sendMessageBodyString(ImJson.toJSONString(resultObject));

//画面を再表示します。
//forward("sample/mobile_fw/sp_store", request);
```

> **コラム**
>
> - 非同期時はresponse.sendMessageBodyString関数で返却データを画面に返します。
> - 返却するデータはImJson.toJSONString関数でJSON文字列化する必要があります。

「登録」ボタン押下時のスクリプト処理を修正します。

- プレゼンテーションページ（HTML）

```
<script>
 (function($){
   $('#main').bind("pagecreate create", 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');
      }
      return false;
    });
   });
 })(jQuery);
</script>
```

- 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
- サーバから非同期で返信するにはresponse.sendMessageBodyString関数を使う

最終結果

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_store.js

```
function init(request) {
}
```

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_store.html

```
<imart type="head">
 <title>TODO登録</title>
</imart>
<div data-role="page" id="main">
 <imart type="imuiValidationRule" rule="sample/mobile_fw/validator#validateRule" rulesName="rules"
messagesName="messages" />
 <script>
  (function($){
   $('#main').on("pagecreate", function() {
    // Formの2度押し防止
    $('#storeForm').imspDisableOnSubmit();
    //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)) {
      //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>
 <imart type="spHeaderWithLink" path="home" headerText="TODO登録" />
 <div class="ui-content" role="main">
  <form id="storeForm" name="storeForm" method="POST" action="sample/sp/store/insert" data-ajax="false">
   <imart type="spFieldContain" label="TODO名" required="true">
    <imart type="input" style="text" name="title" value=title />
   </imart>

   <imart type="spFieldContain" label="期限" required="true">
    <imart type="spDatePicker" name="limit_date" value=limit_date />
   </imart>

   <imart type="spFieldContain" label="コメント">
    <imart type="textarea" name="comment" value=comment />
   </imart>

   <imart type="spControlGroup" label="重要度">
    <imart type="spRadioButton" name="priority" id="radio1" value="0" label="低"></imart>
    <imart type="spRadioButton" name="priority" id="radio2" value="1" label="中"></imart>
    <imart type="spRadioButton" name="priority" id="radio3" value="2" label="高"></imart>
   </imart>

   <imart type="spFieldContain" label="進捗">
    <imart type="spSlider" name="progress" value=progress min="0" max="100" />
   </imart>

   <imart type="spFieldContain" label="完了">
    <imart type="spToggleSwitch" name="complete" selected=complete onLabel="完了" offLabel="未完了" onValue="1"
offValue="0" />
   </imart>
   <a class="ui-btn ui-btn-b ui-corner-all" id="storeButton">登録</a>
  </form>
 </div>
 <imart type="spCommonFooter" dataPosition="fixed"/>
</div>
```

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_store_do.js

```
/**
 * @validate sample/mobile_fw/validator#validateRule
 * @onerror handleErrors
 */
function init(request) {
   //ユーザプロファイル情報を取得します。
   var userProfile    = Contexts.getUserContext().userProfile;

   //登録データを作成します。
   var insertObject = {
      id:Identifier.get(),              //レコードのユニークID
      user_cd:userProfile.userCd,       //登録ユーザCD
      user_nm:userProfile.userName,     //登録ユーザ名
      title:request.title,              //TODOのタイトル
      limit_date:request.limit_date,    //TODOの期日
      comment:request.comment,          //コメント
      progress:parseInt(request.progress),//進捗度
      complete:request.complete,        //完了or未完了
      priority:request.priority,        //重要度
      timestmp:DateTimeFormatter.format("yyyy/MM/dd HH:mm", new Date()) //タイムスタンプ(yyyy/MM/dd HH:mm)
   };

   var resultObject;

   Transaction.begin(function() {
      // データの保存を行います
      var result = new TenantDatabase().insert("mfw_sample", insertObject);
      if(result.error){
         Transaction.rollback();
         resultObject = {
            error:true,
            errorMessage:"データ登録時にエラーが発生しました。",
            detailMessages:["管理者にお問い合わせください。"]
         };
      } else {
         resultObject = {
            error:false,
            errorMessage:"",
            successMessage:"登録が完了しました。"
         };
      }
   });
   var response = Web.getHTTPResponse();
   response.setContentType('application/json; charset=utf-8');
   response.sendMessageBodyString(ImJson.toJSONString(resultObject));
}

function handleErrors(request, validationErrors) {
 Transfer.toErrorPage({
     title: 'エラー',
     message: '入力チェックエラーが発生しました。',
     detail: validationErrors.getMessages()
  });
}
```

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/validator.js

```
var validateRule = {
   "title": {
      caption:"CAP.Z.IWP.MFW.SAMPLE.TITLE",
      required:true,
      maxlength:20
   },
   "limit_date": {
      caption:"CAP.Z.IWP.MFW.SAMPLE.LIMIT_DATE",
      required:true,
      date:true
   }
};
```

- %CONTEXT_PATH%/WEB-INF/conf/routing-jssp-config/im_mobile_sample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-
config routing-jssp-config.xsd ">
  <authz-default mapper="welcome-all" />
  <file-mapping path="/sample/sp/store" client-type="sp" page="/sample/mobile_fw/sp_store" />
  <file-mapping path="/sample/sp/store/insert" client-type="sp" page="/sample/mobile_fw/sp_store_do" />
</routing-jssp-config>
```

- %CONTEXT_PATH%/WEB-INF/conf/message/im_mobile_sample.properties

  native2ascii コマンドを利用して、プロパティファイルを変換した結果を示します。

```
# CAP.Z.IWP.MFW.SAMPLE.TITLE=タイトル
CAP.Z.IWP.MFW.SAMPLE.TITLE=\u30bf\u30a4\u30c8\u30eb
# CAP.Z.IWP.MFW.SAMPLE.LIMIT_DATE=期限
CAP.Z.IWP.MFW.SAMPLE.LIMIT_DATE=\u671f\u9650
```

実装例：一覧画面を作る

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

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

前提条件

- intra-mart Accel Platform をインストールし、初期設定までが完了していること。
- ベースモジュールにスクリプト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。

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_list.js

```
function init(request) {
}
```

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_list.html

```
<imart type="head">
 <title>TODO一覧</title>
</imart>
<div data-role="page">
 <imart type="spHeaderWithLink" headerText="TODO一覧" path="home" />
 <div class="ui-content" role="main">
 </div>
 <imart type="spCommonFooter" dataPosition="fixed"/>
</div>
```

ルーティングの設定

次に、一覧画面へのURLのルーティングを追加します。

- %CONTEXT_PATH%/WEB-INF/conf/routing-jssp-config/im_mobile_sample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-
config routing-jssp-config.xsd ">
  <authz-default mapper="welcome-all" />
  <file-mapping path="/sample/sp/store" client-type="sp" page="/sample/mobile_fw/sp_store" />
  <file-mapping path="/sample/sp/store/insert" client-type="sp" page="/sample/mobile_fw/sp_store" action="insertData" />
  <file-mapping path="/sample/sp/list" client-type="sp" page="/sample/mobile_fw/sp_list" />
</routing-jssp-config>
```

メニューの設定

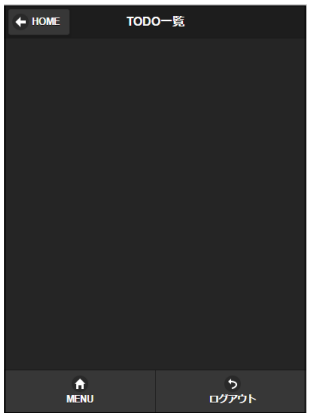
登録画面と同様に一覧画面をメニューに追加します。

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一覧」とし、URLを”sample/sp/list”とします。

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

TODO一覧を選択します。先ほど作ったブランク画面が表示されました。



一覧表示部品を配置する

一覧表示を実現するには、<ul>タグ、および<li>タグを使用します。

一覧のマークアップ例

例として見出し行、および３行の一覧を表示してみます。
sp_list.html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。
> - data-divider-theme属性にテーマを指定することができます。
> - 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

- sp_list.jsに検索処理を追加します。

```
var $list = [];

function init(request) {
 var tenantDB  = new TenantDatabase();
 var selQuery     = "select * from mfw_sample order by timestmp";

 var seltResult = tenantDB.select(selQuery);
 if (seltResult.error) {
    Transfer.toErrorPage({
        title: 'エラー',
        message: 'データ検索時にエラーが発生しました。',
        detail: seltResult.errorMessage
    });
 }
 $list = seltResult.data;
}
```

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。

- sp_list.htmlのulタグの内部を修正します。

```
<ul data-role="listview" data-divider-theme="b">
 <li data-role="list-divider">テストの一覧</li>
 <imart type="repeat" list=$list item="record">
   <li><imart type="string" value=record.title /></li>
 </imart>
</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

- sp_list.jsを以下のように修正します。

```
var $list = [];
var $maxRecord;       //レコード総件数
var $pageLine = 5;    //一度に表示する件数
var $currentPage = 1;  //現在表示ページ
var $contextPath;   //コンテキストパス

function init(request) {
  //表示ページの指定がある場合は$currentPageに設定
  if (request.currentPage !== undefined) {
    $currentPage = request.currentPage;
  }
  //コンテキストパスの取得
  $contextPath = Web.getContextPath();

  var tenantDB  = new TenantDatabase();

  //全レコード件数を取得
  var countQyery   = "select count(*) from mfw_sample ";
  var countResult = tenantDB.select(countQyery);
  if (countResult.error) {
    Transfer.toErrorPage({
       title: 'エラー',
       message: 'データ検索時にエラーが発生しました。',
       detail: countResult.errorMessage
    });
  }
  $maxRecord = countResult.data[0].count;

  //表示するページ分だけTODO情報を取得
  var startLine =  ($currentPage - 1) * $pageLine;
  var selQuery     = "select * from mfw_sample order by timestmp";
  var seltResult = tenantDB.fetch(selQuery, startLine, $pageLine);
  if (seltResult.error) {
    Transfer.toErrorPage({
      title: 'エラー',
      message: 'データ検索時にエラーが発生しました。',
      detail: seltResult.errorMessage
    });
  }
  $list = seltResult.data;
}
```

ページング用タグの配置

- sp_list.htmlのulタグの内部に<imart type=”spPagingButton”>タグを表示します。

```
<ul data-role="listview" data-divider-theme="b">
  <li data-role="list-divider">テストの一覧</li>
  <imart type="repeat" list=$list item="record">
    <li><imart type="string" value=record.title /></li>
  </imart>
  <imart type="spPagingButton" maxRecord=$maxRecord pageLine=$pageLine currentPage=$currentPage />
</ul>
```

- 使用するタグについて

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

ルーティングにパラメータを設定

- im_mobile_sample.xmlにURLを新規追加します。

```
<file-mapping path="/sample/sp/list/{currentPage}" client-type="sp" page="/sample/mobile_fw/sp_list" />
```

> **コラム**
> PathVariablesの詳細については *ルーティング* を参照してください。

ページング処理の実装

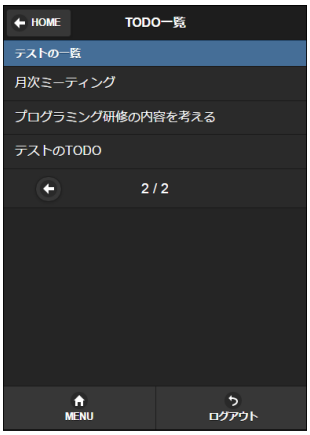
- 最後に、sp_list.htmlにページ遷移ボタン押下時の処理を追加します。 onPageLinkFunc関数で受け取ったページをURLに埋め込みパラメータとして送信します。

```
<div data-role="page">
 <script>
  function onPageLinkFunc(page) {
   document.location.href = "sample/sp/list/" + page;
  }
 </script>
```

> **コラム**
> spPagingButton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。



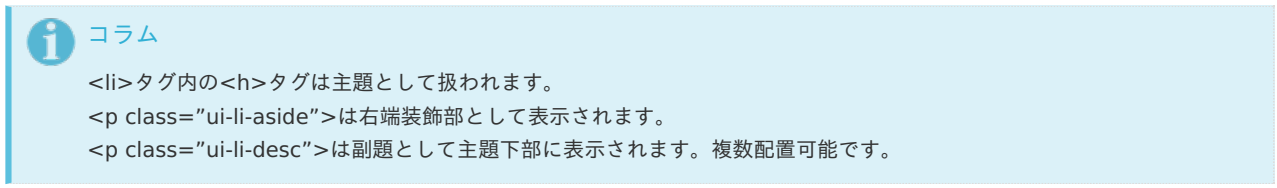
> **コラム**
> この項目では、下記のポイントを確認しました。
>
> - ページング処理を実装するには<imart type=”spPagingButton”>タグを使用する
> - ルーティングのpath valiable機能を利用してページ番号を送信する

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"><imart type="string" value=$currentPage />ページ目</li>
 <imart type="repeat" list=$list item="record">
  <li>
   <p class="ui-li-aside"><imart type="string" value=record.limit_date />まで</p>
   <h3><imart type="string" value=record.title /></h3>
   <p class="ui-li-desc"><imart type="string" value=record.comment /></p>
  </li>
 </imart>
 <imart type="spPagingButton" maxRecord=$maxRecord pageLine=$pageLine currentPage=$currentPage/>
</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

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_list.js

```
var $list = [];
var $maxRecord;       //レコード総件数
var $pageLine = 5;    //一度に表示する件数
var $currentPage = 1; //現在表示ページ
var $contextPath;   //コンテキストパス

function init(request) {
  //表示ページの指定がある場合は$currentPageに設定
  if (request.currentPage !== undefined) {
    $currentPage = request.currentPage;
  }
  //コンテキストパスの取得
  $contextPath = Web.getContextPath();

  var tenantDB  = new TenantDatabase();

  //全レコード件数を取得
  var countQyery   = "select count(*) from mfw_sample ";
  var countResult = tenantDB.select(countQyery);
  if (countResult.error) {
    Transfer.toErrorPage({
      title: 'エラー',
      message: 'データ検索時にエラーが発生しました。',
      detail: countResult.errorMessage
    });
  }
  $maxRecord = countResult.data[0].count;

  //表示するページ分だけTODO情報を取得
  var startLine =  ($currentPage - 1) * $pageLine;
  var selQuery     = "select * from mfw_sample order by timestmp";
  var seltResult = tenantDB.fetch(selQuery, startLine, $pageLine);
  if (seltResult.error) {
    Transfer.toErrorPage({
      title: 'エラー',
      message: 'データ検索時にエラーが発生しました。',
      detail: seltResult.errorMessage
    });
  }
  $list = seltResult.data;
}
```

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_list.html

```
<imart type="head">
 <title>TODO一覧</title>
</imart>
<div data-role="page">
 <script>
  function onPageLinkFunc(page) {
   document.location.href = "sample/sp/list/" + page;
  }
 </script>
 <imart type="spHeaderWithLink" 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"><imart type="string" value=$currentPage />ページ目</li>
   <imart type="repeat" list=$list item="record">
    <li>
     <p class="ui-li-aside"><imart type="string" value=record.limit_date />まで</p>
     <h3><imart type="string" value=record.title /></h3>
     <p class="ui-li-desc"><imart type="string" value=record.comment /></p>
    </li>
   </imart>
   <imart type="spPagingButton" maxRecord=$maxRecord pageLine=$pageLine currentPage=$currentPage />
  </ul>
 </div>
 <imart type="spCommonFooter" dataPosition="fixed"/>
</div>
```

- %CONTEXT_PATH%/WEB-INF/conf/routing-jssp-config/im_mobile_sample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-
config routing-jssp-config.xsd ">
  <authz-default mapper="welcome-all" />
  <file-mapping path="/sample/sp/store" client-type="sp" page="/sample/mobile_fw/sp_store" />
  <file-mapping path="/sample/sp/store/insert" client-type="sp" page="/sample/mobile_fw/sp_store_do" />
  <file-mapping path="/sample/sp/list" client-type="sp" page="/sample/mobile_fw/sp_list" />
  <file-mapping path="/sample/sp/list/{currentPage}" client-type="sp" page="/sample/mobile_fw/sp_list" />
</routing-jssp-config>
```

実装例：参照画面を作る

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

項目

- 前提条件
- 画面の用意
- 画面遷移処理を実装する
- 参照項目を表示する
- レイアウトをカスタマイズする
- 最終結果

前提条件

- intra-mart Accel Platform をインストールし、初期設定までが完了していること。
- ベースモジュールにスクリプト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２点のファイルを作成します。

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_ref.js

```
var $id;
function init(request) {
   $id = request.id;
}
```

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_ref.html

```
<imart type="head">
  <title>TODO参照</title>
</imart>
<div data-role="page">
  <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">
    <imart type="string" value=$id />
  </div>
  <imart type="spCommonFooter" 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。

次に、参照画面へのURLのルーティングを追加します。

- %CONTEXT_PATH%/WEB-INF/conf/routing-jssp-config/im_mobile_sample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-
config routing-jssp-config.xsd ">
  <authz-default mapper="welcome-all" />
  <file-mapping path="/sample/sp/store" client-type="sp" page="/sample/mobile_fw/sp_store" />
  <file-mapping path="/sample/sp/store/insert" client-type="sp" page="/sample/mobile_fw/sp_store" action="insertData" />
  <file-mapping path="/sample/sp/list" client-type="sp" page="/sample/mobile_fw/sp_list" />
  <file-mapping path="/sample/sp/list/{currentPage}" client-type="sp" page="/sample/mobile_fw/sp_list" />
  <file-mapping path="/sample/sp/ref/{id}" client-type="sp" page="/sample/mobile_fw/sp_ref" />
</routing-jssp-config>
```

> **コラム**
> PathVariablesの詳細については *ルーティング* を参照してください。

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。

- sp_list.htmlの<li>タグにaタグを追加します。

```
<imart type="repeat" list=$list item="record">
  <li>
    <a href='<imart type="string" value=$contextPath/>/sample/sp/ref/<imart type="string" value=record.id />'>
      <p class="ui-li-aside"><imart type="string" value=record.limit_date />まで</p>
      <h3><imart type="string" value=record.title /></h3>
      <p class="ui-li-desc"><imart type="string" value=record.comment /></p>
    </a>
  </li>
</imart>
```

- 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。

> **コラム**
>
> この項目では、下記のポイントを確認しました。
>
> - 画面遷移を実装する場合、ルーティングを設定し<a>タグのhref属性に指定する

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。

- sp_ref.jsを以下のように修正します。

```
var $record;
var $id;

function init(request) {
 $id = request.id;
 var tenantDB  = new TenantDatabase();
 var query   = "select * from mfw_sample where id = ?";

 var result = tenantDB.select(query, [new DbParameter($id, DbParameter.TYPE_STRING)]);
 if (result.error) {
     Transfer.toErrorPage({
       title: 'エラー',
       message: '削除処理時にエラーが発生しました。',
       detail: result.errorMessage
   });
 }

 $record = result.data[0];
}
```

- sp_ref.htmlの 以下を以下のように修正します。

```
<div class="ui-content" role="main">
 <imart type="spFieldContain" label="登録者">
  <imart type="string" value=$record.user_nm />
 </imart>
 <imart type="spFieldContain" label="登録日">
  <imart type="string" value=$record.timestmp />
 </imart>
 <imart type="spFieldContain" label="TODO名">
  <imart type="string" value=$record.title />
 </imart>
 <imart type="spFieldContain" label="期限">
  <imart type="string" value=$record.limit_date />
 </imart>
 <imart type="spFieldContain" label="コメント">
  <imart type="string" value=$record.comment />
 </imart>
 <imart type="spFieldContain" label="進捗">
  <imart type="string" value=$record.progress />%
 </imart>
 <imart type="spFieldContain" label="完了">
  <imart type="decision" value=$record.complete case="0">
   未完了
  </imart>
  <imart type="decision" value=$record.complete case="0">
   完了
  </imart>
 </imart>
</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

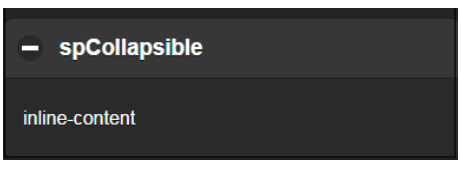
より参照画面を見やすくするために、画面をカスタマイズします。

- sp_ref.htmlの登録者・登録日項目を以下の様に修正します。

```
...
<div data-role="content" role="main">
  <imart type="spCollapsible" title="登録者情報" dataTheme="b" collapse="false" dataInset="false">
    <imart type="spFieldContain" label="登録者">
      <h3><imart type="string" value=$record.user_nm /></h3>
    </imart>
    <imart type="spFieldContain" label="登録日">
      <h3><imart type="string" value=$record.timestmp /></h3>
    </imart>
  </imart>
  <imart type="spFieldContain" 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

jssp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">
  <imart type="spCollapsible" title="TODO内容" dataTheme="b" collapse="false">
    <imart type="spFieldContain" label="TODO名">
      <imart type="string" value=$record.title />
    </imart>
      ...
    <imart type="spFieldContain" label="コメント">
      ...
    </imart>
  </imart>
  <imart type="spCollapsible" title="進捗状況" dataTheme="b">
    <imart type="spFieldContain" label="進捗">
      <imart type="string" value=$record.progress />%
    </imart>
    <imart type="spFieldContain" label="完了">
      ...
    </imart>
  </imart>
</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。

> **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

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_list.html

```
<div data-role="page" id="main" data-theme="a">
  <script>
    function onPageLinkFunc(page) {
      document.location.href = "sample/sp/list/" + page;
    }
  </script>
  <imart type="spHeaderWithLink" path="home" label="HOME" headerText="TODO一覧" />
  <div class="ui-content" role="main">
    <ul data-role="listview" data-divider-theme="b">
      <li data-role="list-divider"><imart type="string" value=$currentPage />ページ目</li>
      <imart type="repeat" list=$list item="record">
        <li>
          <a href="<imart type="string" value=$contextPath/>/sample/sp/ref/<imart type="string" value=record.id />">
            <p class="ui-li-aside"><imart type="string" value=record.limit_date />まで</p>
            <h3><imart type="string" value=record.title /></h3>
            <p class="ui-li-desc"><imart type="string" value=record.comment /></p>
          </a>
        </li>
      </imart>
      <imart type="spPagingButton" maxRecord=$maxRecord pageLine=$pageLine currentPage=$currentPage/>
    </ul>
  </div>
  <imart type="spCommonFooter" dataPosition="fixed"/>
</div>
```

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_ref.js

```
var $record;
var $id;

function init(request) {
    $id = request.id;
    var tenantDB  = new TenantDatabase();
    var query   = "select * from mfw_sample where id = ?";

    var result = tenantDB.select(query, [new DbParameter($id, DbParameter.TYPE_STRING)]);
    if (result.error) {
        Transfer.toErrorPage({
            title: 'エラー',
            message: '削除処理時にエラーが発生しました。',
            detail: result.errorMessage
        });
    }

    $record = result.data[0];
}
```

- %CONTEXT_PATH%/WEB-INF/jssp/src/sample/mobile_fw/sp_ref.html

```
<imart type="head">
 <title>TODO参照</title>
</imart>

<div data-role="page">
 <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">
  <imart type="spCollapsible" title="登録者情報" dataTheme="b" collapse="false" dataInset="false">
   <imart type="spFieldContain" label="登録者">
    <imart type="string" value=$record.user_nm />
   </imart>
   <imart type="spFieldContain" label="登録日">
    <imart type="string" value=$record.timestmp />
   </imart>
  </imart>
  <div data-role="collapsible-set">
   <imart type="spCollapsible" title="TODO内容" dataTheme="b" collapse="false">
    <imart type="spFieldContain" label="TODO名">
     <imart type="string" value=$record.title />
    </imart>
    <imart type="spFieldContain" label="期限">
     <imart type="string" value=$record.limit_date />
    </imart>
    <imart type="spFieldContain" label="コメント">
     <imart type="string" value=$record.comment />
    </imart>
   </imart>
   <imart type="spCollapsible" title="進捗状況" dataTheme="b">
    <imart type="spFieldContain" label="進捗">
     <imart type="string" value=$record.progress />%
    </imart>
    <imart type="spFieldContain" label="完了 / 未完了">
     <imart type="decision" value=$record.complete case="0">
                未完了
     </imart>
     <imart type="decision" value=$record.complete case="1">
                完了
     </imart>
    </imart>
   </imart>
  </div>
 </div>
 <imart type="spCommonFooter" dataPosition="fixed"/>
</div>
```

- %CONTEXT_PATH%/WEB-INF/conf/routing-jssp-config/im_mobile_sample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<routing-jssp-config xmlns="http://www.intra-mart.jp/router/routing-jssp-config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.intra-mart.jp/router/routing-jssp-
config routing-jssp-config.xsd ">
 <authz-default mapper="welcome-all" />
 <file-mapping path="/sample/sp/store" client-type="sp" page="/sample/mobile_fw/sp_store" />
 <file-mapping path="/sample/sp/store/insert" client-type="sp" page="/sample/mobile_fw/sp_store_do" />
 <file-mapping path="/sample/sp/list" client-type="sp" page="/sample/mobile_fw/sp_list" />
 <file-mapping path="/sample/sp/list/{currentPage}" client-type="sp" page="/sample/mobile_fw/sp_list" />
 <file-mapping path="/sample/sp/ref/{id}" client-type="sp" page="/sample/mobile_fw/sp_ref" />
</routing-jssp-config>
```

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
- ページ
- ヘッダ
- フッタ
- ボタン
- フォーム要素
- クライアントJavaScript
- イベント

スマートフォン版テーマ

積極的に利用してください。
使用することで intra-mart Accel Applications 基盤画面部品、およびPlatform上の各種アプリケーションと同じ画面デザインを利用できます。

ページ

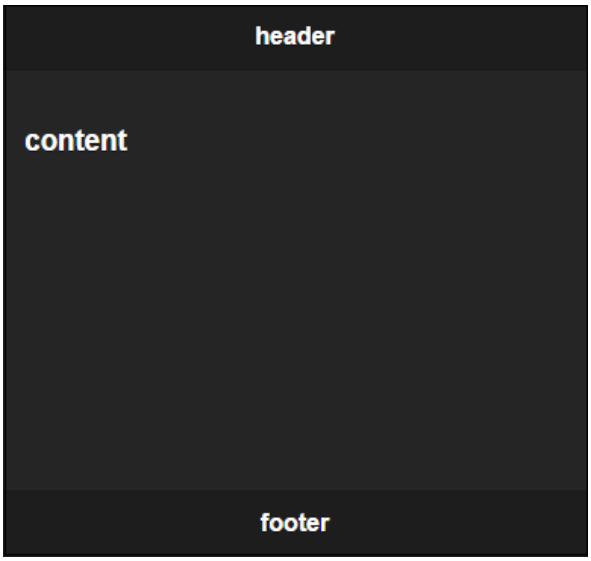
スウォッチ

明暗所の視認性を考慮し、基本的に”A”（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、<imart type=”spHeaderWithLink”>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。

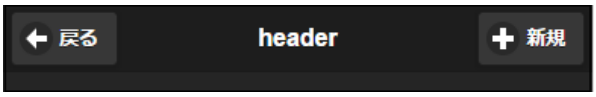
- 基本的に、入力操作系ボタン（新規ボタン、編集ボタン、投稿ボタン）やトランザクション処理ボタン（登録ボタン、更新ボタン）を配置します。
- 入力操作系ボタンは、文字列とします。

マークアップ例

- html

```
<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 <imart type=”spCommonFooter”>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。

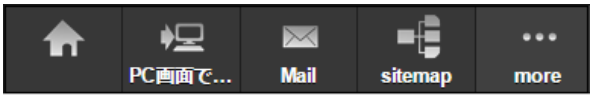
- data-icon=”custom”
- class=”im-smart-icon-common-32-more-navbar”
- data-iconpos=”top”

マークアップ例

- html

```
<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



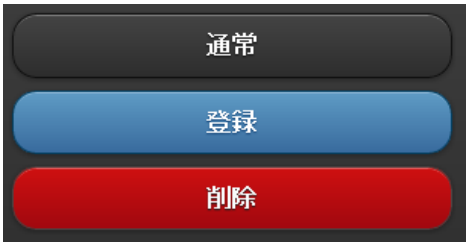
ボタン

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

<imart type=”spFieldContain”>タグまたは 、<imart type=”spControlGroup”>タグまたは タグを使用してください。

必須項目の強調

spFieldContain、またはspControlGroup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。

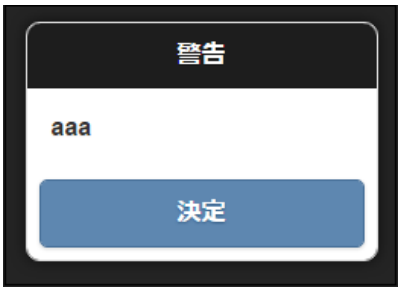
window.alert() について

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() {
  ...
 });
 ...
});
```

旧バージョンで使用していたプログラムの移行

項目
- 共通
- スマートフォン版テーマを利用して移行する（推奨）
- スマートフォン版テーマを利用しないで移行する

共通

intra-mart Accel Platform ではルータ機能によりURLの取り扱い方法が変わりました。
そのため、旧バージョンで使用していたIM-Mobile Frameworkの画面を移行するためには以下の修正作業が必要です。
サーバサイド処理の移行はPC版と共通です。別途 *旧バージョンで作成したプログラムの実行* を参考に修正作業を行ってください。

各画面・アクションに対するルーティングを設定する

ルーティング を参考に、各画面のルーティングを設定してください。

spHeaderWithBackタグをspHeaderWithLinkタグに変更する

spHeaderWithBackタグは非推奨になりました。 代替タグとしてspHeaderWithLinkタグを使用してください。

imart type=”link”タグをaタグに変更する

代替として<A>タグを使用し、href属性にルーティングで設定したパスを指定してください。
属性値を要する場合、ルーティング を参考にのpath valiable機能を使用してください。

imart type=”form”タグをformタグに変更する

代替として<form>タグを使用し、action属性にルーティングで設定したパスを指定してください。

スマートフォン版テーマを利用して移行する（推奨）

プレゼンテーションページ

<HTML><HEAD><BODY>タグの記述は不要ですので削除してください。
画面固有のスタイル、およびタイトルなどは別途<imart type=”head”>タグを配置し、内部に記述してください。
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。

※別途スクリプトやスタイルを定義している場合は記述を残してください。

スマートフォン版テーマを利用しないで移行する

プレゼンテーションページ

共通に挙げた事項を除きプレゼンテーションページの変更は不要です。

テーマを使用しない設定を加える

特定のパスに対して明示的にテーマ使用しないための設定を加えます。
例として、%CONTEXT_PATH%/WEB-INF/jssp/src/mobile_fw/nothemeフォルダ配下のプレゼンテーションページに対してこの設定を適用するため、新規XMLファイルを作成します。

- %CONTEXT_PATH%/WEB-INF/conf/theme-no-theme-path-config/im_mobile.xml

```
<?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>/mobile_fw/notheme/.*</path>
</theme-no-theme-path-config>
```

エラー処理

項目
- クライアントサイドのエラー処理
- サーバサイドのエラー処理

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。

サーバサイドのエラー処理

サーバサイドのエラー処理は以下を参照してください。

JSSP Validator

項目

- JSSP Validatorとは
- 仕様
  - 前提条件
  - バリデーションの仕組み
  - バリデーションルール
- プログラミング方法
  - JSSP Validatorアノテーション
  - バリデーション設定ファイル
  - エラーメッセージ
  - エラーオブジェクト
  - エラーハンドラ関数
- カスタムバリデータ
  - Java Scriptで実装
  - Javaで実装
  - jssp-validation-configの設定

JSSP Validatorとは

JSSP Validatorとはスクリプト開発モデルにおいて、リクエストパラメータの値が指定した条件に合致しているかを検証する機能です。 検証の結果、合致していなければエラーメッセージ生成し指定した関数を実行します。

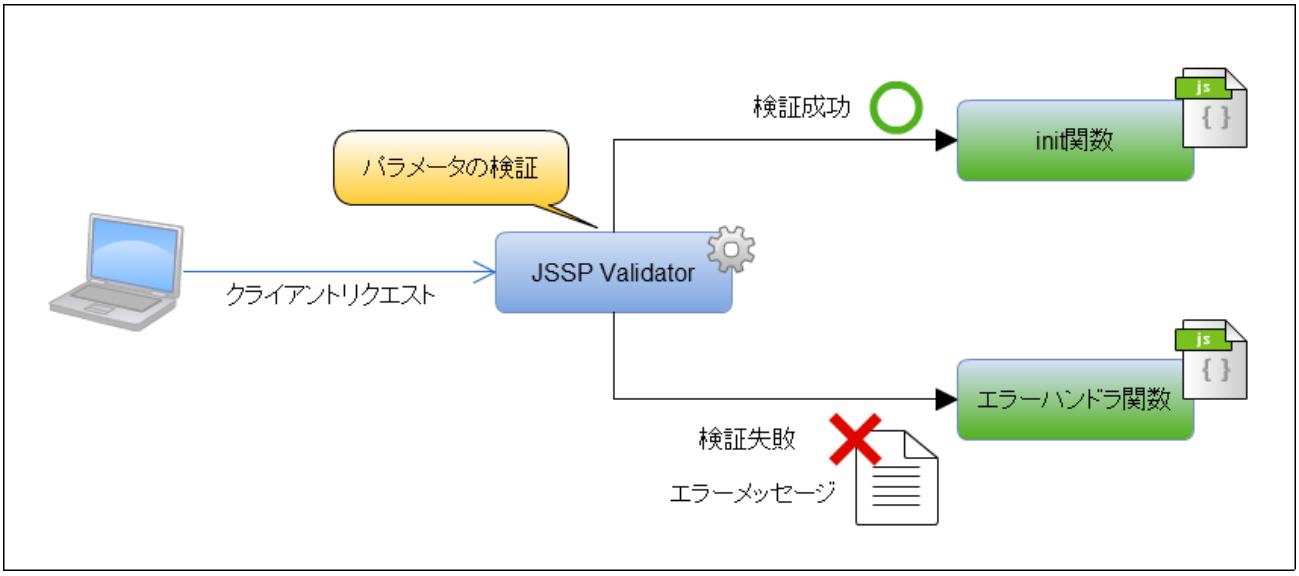
仕様

前提条件

JSSP Validatorは下記の関数に使用することが可能です。

- js初期化時に実行されるinit関数
- imartタグのaction属性に指定されたアクション関数
  - <imart type=”form” />
  - <imart type=”submit” />
  - <imart type=”link” />

バリデーションの仕組み



JSSP Validatorは下記の流れで処理を実行します。

1. クライアントからリクエストが送信されます。
2. JSSP Validatorがバリデーション設定ファイルに記述されたルールに基づきリクエストパラメータを検証します。
3. 検証に失敗した場合、エラーオブジェクトにエラーメッセージを追加します。

4. エラーオブジェクトにエラーメッセージが有る場合はエラーハンドラ関数を実行します。エラーメッセージが無い場合は init関数またはformなどのaction属性で指定されたアクション関数を実行します。

バリデーションルール

下記はバリデーションルールの一覧です。各ルールの詳細は *バリデーションルール リファレンス* を参照してください。

バリデーションルールの一覧

キー	説明
required	必須である
alpha	値がアルファベットである
alphanumeric	値がアルファベットと数字である
numeric	値が数字である
digits	整数部、小数部の桁数
lowercase	アルファベットの小文字である
uppercase	アルファベットの大文字である
integer	整数型の数値である
decimal	実数型の数値である
minlength	文字列の最小長
maxlength	文字列の最大長
min	数字の最小値
max	数字の最大値
range	数値の範囲
email	メールアドレスである
url	URLである
equals	等しい
contains	含む
isIn	含まれる
regex	正規表現に一致する
file	ファイルアップロードである
mimeType	ファイルのMIME Typeが一致する
id	ID、コード系に使用可能な文字である
userCd	ユーザコードに使用可能な文字である
date	アカウントコンテキストの日時表示形式に含まれている「日付（入力）」と同じフォーマットである
time	アカウントコンテキストの日時表示形式に含まれている「時刻（入力）」と同じフォーマットである
datetime	アカウントコンテキストの日時表示形式に含まれている「日付（入力）」＋「時刻（入力）」と同じフォーマットである

プログラミング方法

JSSP Validatorを使用する手順は大まかに下記のようになります。

1. バリデーションを行う関数にJSSP Validatorアノテーションの記述する。
2. バリデーション設定ファイルを作成し、バリデーションルールを記述する。
3. エラーメッセージを記述する。
4. エラーハンドラ関数を記述しエラー処理を行う。

JSSP Validatorアノテーション

バリデーションを行う関数にアノテーションを記述します。 関数のコメントに **@validate** と **@onerror** アノテーションを記述してください。

```
/**
 * 初期化処理
 *
 * . . .
 * @validate foo/bar_validation#init
 * @onerror handleErrors
 */
function init(request){
 ...
}
```

JSSP Validatorアノテーション

アノテーション	説明
@validate	バリデーション設定ファイルのパスと変数名を「#」で区切り記述します。ファイルの拡張子は省略してください。 たとえばバリデーション設定ファイルのパスが「**foo/bar_validation.js** 」で変数名が「**init** 」の場合、「**foo/bar_validation#init** 」となります。 バリデーション設定ファイルについては *バリデーション設定ファイル* を参照してください。
@onerror	エラーハンドラ関数名を記述します。@validateアノテーションを記述している場合は必ず@onerrorアノテーションも記述してください。 エラーハンドラ関数については *エラーハンドラ関数* を参照してください。

バリデーション設定ファイル

スクリプトパスの任意のディレクトリにjsファイルを作成してください。
この例では「%CONTEXT_PATH%/WEB-INF/jssp/src/foo/bar_validation.js」とします。

変数の宣言を記述し、その属性には検証を行うパラメータ名を記述します。 パラメータ名の属性には検証をするバリデーションルールのキーを記述してください。

caption 属性はエラーメッセージを生成するために必要な属性です。必ず記述してください。 エラーメッセージについては *エラーメッセージ* を参照してください。

下記の例ではリクエストパラメータ「xxx_name」に必須項目かつ50文字以下であるかを検証しています。

```
var init = {
 'xxx_name': { // リクエストパラメータ名
     caption: "CAP.Z.EXAMPLE.NAME", // エラーメッセージのキャプション
     required: true,
     maxlength: 50
  }
}
```

バリデーション設定ファイルには複数の設定を記述することが可能です。変数名を設定ファイル内で固有のものとしてください。

```
var init = {
 'xxx_name': {
    . . .
  }
}

var updateXXX = {
 'xxx_name': {
    . . .
  }
}

var deleteXXX = {
 'xxx_name': {
    . . .
  }
}
```

エラーメッセージ

リクエスト検証に失敗した場合エラーメッセージを生成します。

バリデーション設定ファイルに記述された **caption** 属性の値が「CAP.Z.EXAMPLE.NAME」であり、 バリデーションルールが「required」と「maxlength」となっている場合を例に説明します。

```
var init = {
 'xxx_name': { // リクエストパラメータ名
     caption: "CAP.Z.EXAMPLE.NAME", // エラーメッセージのキャプション
     required: true,
     maxlength: 50
  }
}
```

「%CONTEXT_PATH%/WEB-INF/conf/message/」ディレクトリ内の任意のディレクトリにプロパティファイルを作成します。
この例では「%CONTEXT_PATH%/WEB-INF/conf/message/foo/bar-message_ja.properties」とします。

プロパティファイルに「CAP.Z.EXAMPLE.NAME」をキーとしたエラーメッセージのキャプションを記述してください。
日本語などのマルチバイト文字の場合はUnicodeに変換して記述する必要があります。

この例では「名前」をキャプションとします。

```
CAP.Z.EXAMPLE.NAME=\u540d\u524d
```

この例では、バリデーションルールに「required」と「maxlength」が設定されているため、 リクエスト検証に失敗した場合のエラーメッセージは下記のものとなります。

```
名前は必須です。
名前は50文字以内でなければなりません。
```

コラム

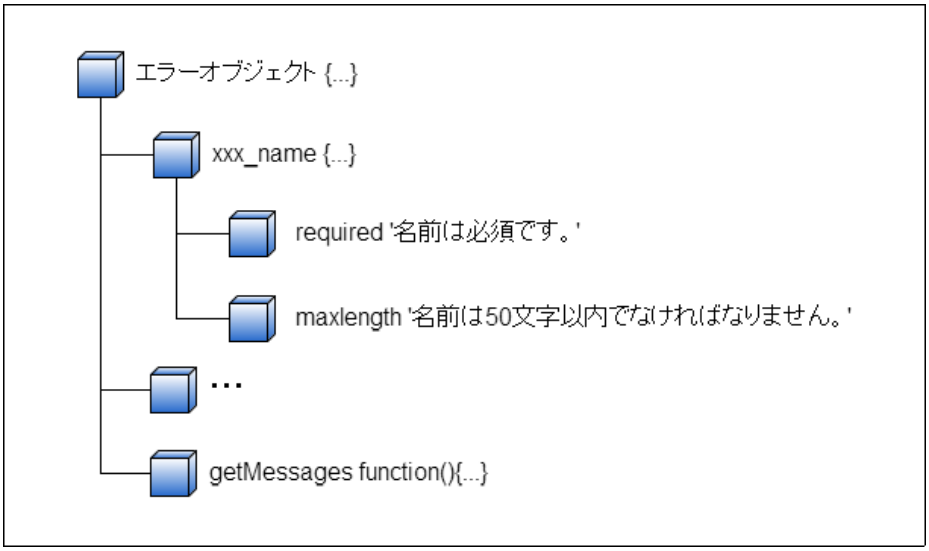
エラーメッセージの多言語対応を行う場合は、必要に応じて言語ID毎のメッセージファイルを作成してください。

- 英語 - bar-message_en.properties
- 日本語 - bar-message_ja.properties
- 中国語（簡体字） - bar-message_zh_CN.properties
- 言語IDの付いていないメッセージファイル - bar-message.properties

詳しい仕様については *多言語* を参照してください。

エラーオブジェクト

エラーオブジェクトには検証に失敗したリクエストパラメータのエラーメッセージが格納されています。下記の図はその例です。



エラーオブジェクトの属性にはリクエストパラメータ名の配列が格納されており、 その属性には検証に失敗したルールのキーがありその値にはエラーメッセージが格納されています。 検証に成功したルールのキーは存在しません。 検証の失敗がひとつもないパラメータは属性がないオブジェクトが格納されています。

getMessagesメソッドはすべてのリクエストパラメータの検証に失敗したルールのエラーメッセージを配列で返します。

エラーハンドラ関数

全てのリクエストパラメータを検証した結果、ひとつでも検証に失敗しているパラメータがある場合は、 エラーハンドラ関数を実行します。**@onerror** アノテーションに記述した名前の関数を記述してください。

エラーハンドラ関数の第1引数にはリクエストオブジェクト、第2引数にはエラーオブジェクトを渡されます。 エラーオブジェクトに格納されているエラーメッセージを取得しエラー処理を行ってください。

下記の例ではリクエストパラメータの値とエラーメッセージを取得しエラーページにフォーワードしています。

エラーオブジェクトについては *エラーオブジェクト* を参照してください。

```
/**
 * 初期化処理
 *
 * . . .
 * @onerror handleErrors
 */
function init(request){
 ...
}

/**
 * エラーハンドラ関数
 *
 * . . .
 */
function handleErrors(request, validationErrors) {
  var param = {
    foo: request.foo,
    errors: validationErrors.getMessages()
  };
  forward("foo/bar_error", param);
}
```

カスタムバリデータ

開発者が独自にバリデーションルールを作成することが可能です。

> **コラム**
>
> カスタムバリデータはサーバサイド側のバリデーションルールの追加でありクライアント側のルールの追加ではありません。
> クライアントサイドで同じバリデーションを行いたい場合は、クライアントサイドでも同様のルールを定義してください。
> クライアントサイドのカスタムバリデーションについては クライアントサイド JavaScript の imuiAddValidationRule を参照してください。

Java Scriptで実装

スクリプトパスの任意のディレクトリにjsファイルを作成し **validate** 関数を記述してください。
この例では「%CONTEXT_PATH%/WEB-INF/jssp/src/foo/custom_validator.js」とします。

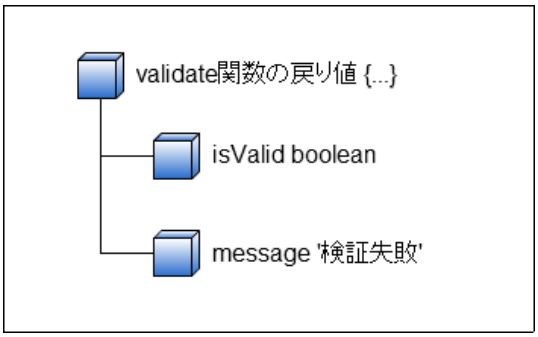
```
/**
 * カスタムバリデータ
 *
 * . . .
 */
function validate(request, config, paramName, caption) {
 var value = request.getParameterValue(paramName);
 if (条件) {
   return {isValid: true};
 } else {
   return {isValid: false, message: "検証失敗"};
 }
}
```

validate関数の引数には下記のオブジェクトが渡されます。

validate関数の引数

引数	型	説明
request	Requestオブジェクト	リクエストオブジェクト
config	オブジェクト（設定値の型による）	バリデーション設定ファイルに記述された設定値
paramName	文字列型	検証を行うパラメータ名
caption	文字列型	バリデーション設定ファイルに記述された設定値caption属性の設定値

validate関数の戻り値には下記の構造のオブジェクトを返してください。 **isValid** 属性は検証成功時にはtrue、失敗時にはfalseを設定してください。 検証失敗時には **message** プロパティにエラーメッセージを設定してください。



Javaで実装

jp.co.intra_mart.foundation.jssp.validation.Validatorインタフェース の実装クラスを作成してください。

```
package foo;

import jp.co.intra_mart.foundation.jssp.validation.InitParam;
import jp.co.intra_mart.foundation.jssp.validation.RequestInfo;
import jp.co.intra_mart.foundation.jssp.validation.ValidationResult;
import jp.co.intra_mart.foundation.jssp.validation.Validator;
import jp.co.intra_mart.system.javascript.Context;
import jp.co.intra_mart.system.javascript.Scriptable;

public class CustomValidator implements Validator {

   @Override
   public void destroy() {
   }

   @Override
   public ValidationResult validate(RequestInfo info) {
      final String value = info.getRequest().getParameter(info.getParameterName());
      if (条件) {
         return ValidationResult.ok();
      } else {
         return ValidationResult.ng("検証失敗");
      }
   }

   @Override
   public void init(InitParam params, Context cx, Scriptable scope) {
   }
}
```

引数、戻り値の説明についてはAPIリストを参照してください。

jssp-validation-configの設定

jssp-validation-configを記述してください。「%CONTEXT_PATH%/WEB-INF/conf/jssp-validation-config/」ディレクトリ内にファイルを作成してください。この例では「jssp-validation-example-config.xml」とします。

下記は記述例です。 **<validator-name>** はシステムで固有のバリデーションルールのキーを記述してください。Java Scriptで実装した場合は **<validator-script>** に作成したjsファイルのパスを記述します。拡張子は省略してください。
Javaで実装した場合は **<validator-class>** に作成したクラス名をパッケージパスを含めて記述します。

jssp-validation-configの文字コードはUTF-8で保存してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<jssp-validation-config
   xmlns="http://intra-mart.co.jp/system/jssp-validation"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://intra-mart.co.jp/system/jssp-validation ../../schema/jssp-validation-config.xsd">

 <!-- Java Script の設定例 -->
 <validator>
   <validator-name>js_example</validator-name>
   <validator-script>foo/custom_validator</validator-script>
 </validator>

 <!-- Java の設定例 -->
 <validator>
   <validator-name>java_example</validator-name>
   <validator-class>foo.CustomValidator</validator-class>
 </validator>
</jssp-validation-config>
```

バリデーションルール リファレンス

項目

- required
- alpha
- alphanumeric
- numeric
- digits
- lowercase
- uppercase
- integer
- decimal
- minlength
- maxlength
- min
- max
- range
- email
- url
- equals
- contains
- isIn
- regex
- file
- mimeType
- id
- userCd
- date
- time
- datetime

required

値が入力されているかを検証します。パラメータが存在しないまたは空文字の場合はエラーとなります。
設定値がfalseの場合は検証を行いません。

設定値

trueまたはfalse

設定例

```
required: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	{0} is required.
ja	{0}は必須です。
zh_CN	{0}□必□□目。

alpha

値がアルファベットであるかを検証します。アルファベット以外の文字が含まれている場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

設定値

trueまたはfalse

設定例

```
alpha: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter alphabets for {0}.
ja	{0}は英字を入力してください。
zh_CN	□在{0}中□入英文字母。

alphanumeric

値がアルファベットまたは数字であるかを検証します。アルファベットと数字以外の文字が含まれている場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

設定値

trueまたはfalse

設定例

```
alphanumeric: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter alphanumeric for {0}.
ja	{0}は英数字を入力してください。
zh_CN	□在{0}中□入英文数字。

numeric

値が数字であるかを検証します。数字以外の文字が含まれている場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

設定値

trueまたはfalse

設定例

```
numeric: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter numeric for {0}.
ja	{0}は数字を入力してください。
zh_CN	□在{0}中□入数字。

digits

数値の整数部、小数部の桁数を検証します。指定した桁数より大きい場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。

設定値

[数値1, 数値2]

- 数値1 - 整数部の最大桁数
- 数値2 - 小数部の最大桁数

設定例

```
digits: [3, 2]
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter integers of {1} digit and decimals of {2} digit for {0}.
ja	{0}は{1}桁の整数、および{2}桁の小数で入力してください。
zh_CN	□以{1}位的整数以及{2}位的小数来□入{0}。

lowercase

値がアルファベット小文字であるかを検証します。アルファベット小文字以外の文字が含まれている場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

設定値

trueまたはfalse

設定例

```
lowercase: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter alphabets and lower case characters for {0}.
ja	{0}は英字かつ小文字で入力してください。
zh_CN	□在{0}中□入小写英文字母。

uppercase

値がアルファベット大文字であるかを検証します。アルファベット大文字以外の文字が含まれている場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

設定値

trueまたはfalse

設定例

```
uppercase: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter alphanumeric and upper case characters for {0}.
ja	{0}は英字かつ大文字で入力してください。
zh_CN	□在{0}中□入大写英文字母。

integer

値が整数型の数字であるかを検証します。整数型の数字ではない場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

設定値

trueまたはfalse

設定例

```
integer: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter integers for {0}.
ja	{0}は整数を入力してください。
zh_CN	□在{0}中□入整数。

decimal

値が実数型の数字であるかを検証します。実数型の数字ではない場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

設定値

trueまたはfalse

設定例

```
decimal: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter numeric for {0}.
ja	{0}は数値を入力してください。
zh_CN	□在{0}中□入数□。

minlength

値の最小文字数を検証します。指定した長さより文字数が小さい場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。

設定値

数値

設定例

```
minlength: 6
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter more than {1} character for {0}.
ja	{0}は{1}文字以上で入力してください。

zh_CN	□在{0}中□入{1}位以上的字符。

maxlength

値の最大文字数を検証します。指定した長さより文字数が大きい場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。

設定値

数値

設定例

```
maxlength: 50
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter within {1} character for {0}.
ja	{0}は{1}文字以内で入力してください。
zh_CN	□□入{0}在{1}字符以内。

min

数字の最小値を検証します。設定値より値が小さい場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。

設定値

数値

設定例

```
min: 6
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter numeric greater than {1} for {0}.
ja	{0}は{1}以上の数値で入力してください。
zh_CN	□□入{0}比{1}大的数□。

max

数字の最大値を検証します。設定値より値が大きい場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。

設定値

数値

設定例

```
max: 6
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter numeric less than {1} for {0}.
ja	{0}は{1}以下の数値を入力してください。

zh_CN	□□入小于{1}的数□来□入{0}。

range

数字の範囲を検証します。設定値より値が大きい場合、または小さい場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。

設定値

[数値1, 数値2]

- 数値1 - 最小値
- 数値2 - 最大値

設定例

```
range: [0, 12]
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter between the range of {1} and {2} for {0}.
ja	{0}は{1}から{2}までの範囲で入力してください。
zh_CN	□□入{0}在{1}到{2}的范□之□。

email

値がe-mailアドレスのフォーマットであるかを検証します。e-mailアドレスのフォーマットではない場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

以下のような値をe-mailアドレスとして認識します。

```
foo@example.com
foo.bar@baz.org
aaa@bbb.ccc
```

ドットが連続していたり、@の直前にドットが存在する場合はe-mailアドレスとして認識されません。

```
foo..bar@baz.org
foo.@bar.org
```

設定値

trueまたはfalse

設定例

```
email: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter mail address format for {0}.
ja	{0}はメールアドレス形式で入力してください。
zh_CN	□以□件地址的形式□入{0}。

url

値がURLのフォーマットであるかを検証します。URLのフォーマットではない場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。

設定値がfalseの場合は検証を行いません。

以下のような値をURLとして認識します。

```
http://example.com
https://foo.com
http://aaa.org:8080/bbb/ccc/
ftp://example.com
```

設定値

trueまたはfalse

設定例

```
url: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter URL format for {0}.
ja	{0}はURL形式で入力してください。
zh_CN	□以URL的形式□入{0}。

equals

値が設定値と同じ文字列であるかを検証します。設定値と同じ文字列ではない場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。

設定値

文字列

設定例

```
equals: 'foo'
```

エラーメッセージ

ロケール	エラーメッセージ
en	{0} should be equal to {1}.
ja	{0}は{1}と等しくなければなりません。
zh_CN	{0}必□与{1}相等。

contains

値に設定値の文字列が含まれているかを検証します。設定値の文字列が含まれていない場合はエラーとなります。
パラメータが存在しないまたは値が空の場合は検証を行いません。

設定値

文字列

設定例

```
contains: 'foo'
```

エラーメッセージ

ロケール	エラーメッセージ
en	{0} must contain {1}.

ja	{0}には{1}が含まれていなければなりません。
zh_CN	{0}必□包含{1}。

isIn

値が設定値の文字列に一致するものが有るかを検証します。設定値に一致するものが無い場合はエラーとなります。
パラメータが存在しない場合は検証を行いません。

設定値

[文字列1, 文字列2, ...]

設定例

```
isIn: ['foo', 'bar', 'baz']
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter any of [{1}] for {0}.
ja	{0}には[{1}]のうちいずれかを入力してください。
zh_CN	□□□“{1}”的范□内的其中一个来□入{0}。

regex

値が正規表現に一致するかを検証します。正規表現に一致しない場合はエラーとなります。
パラメータが存在しない場合は検証を行いません。

設定値

正規表現オブジェクト

設定例

```
regex: /^[a-zA-Z0-9]+$/
```

エラーメッセージ

ロケール	エラーメッセージ
en	Format of {0} is not correct.
ja	{0}の形式が正しくありません。
zh_CN	{0}的格式不正确。

file

値がファイルアップロードであるかを検証します。ファイルアップロードではない場合はエラーとなります。
パラメータが存在しない場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

このバリデーションルールを使用する場合は、送信されるフォームに必ず **enctype=”multipart/form-data”** を設定してください。
設定されていないフォームにこのバリデーションルールを使用するとサーバエラー（HTTPステータス 500）の画面が表示されます。

設定値

trueまたはfalse

設定例

```
file: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	{0} is not a file.
ja	{0}がファイルではありません。
zh_CN	{0}不是文件。

mimeType

ファイルの内容からMIME Type判別し、設定値に記述された条件に合致しているかを検証します。条件に合致しない場合はエラーとなります。
パラメータが存在しない場合は検証を行いません。

このバリデーションルールを使用する場合は、送信されるフォームに必ず **enctype=”multipart/form-data”** を設定してください。
設定されていないフォームにこのバリデーションルールを使用するとサーバエラー（HTTPステータス 500）の画面が表示されます。

設定値

```
{
    include: [文字列, 文字列, ...],
    exclude: [文字列, 文字列, ...]
}
```

- include - 許可するMIME Type
- exclude - 除外するMIME Type

設定例

```
mimeType: {
  include: ['text/html', 'application/msword'],
  exclude: ['application/octet-stream']
}
```

エラーメッセージ

ロケール	エラーメッセージ
en	{0} has file format not allowed.
ja	{0}は許可されていないファイル形式です。
zh_CN	{0}□未□□可的文件格式。

id

値がID、コード系であるかを検証します。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

ID、コード系は下記の文字で構成されている必要があります。これらの文字以外が含まれている場合エラーとなります。

- 半角英字（a-z）（A-Z）
- 半角数字（0-9）
- アンダースコア（_）
- ダッシュ（-）
- アットマーク（@）
- ドット（.）
- プラス（+）
- エクスクラメーション（!）

設定値

trueまたはfalse

設定例

```
id: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter ID format for {0}.
ja	{0}はID形式で入力してください。
zh_CN	□以ID的形式来□入{0}。

userCd

値がユーザコードであるかを検証します。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

ユーザコードは下記の文字で構成されている必要があります。これらの文字以外が含まれている場合エラーとなります。

- 半角英字（a-z）（A-Z）
- 半角数字（0-9）
- アンダースコア（_）
- ダッシュ（-）
- アットマーク（@）
- ドット（.）
- プラス（+）
- エクスクラメーション（!）

設定値

trueまたはfalse

設定例

```
userCd: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	Enter user code format for {0}.
ja	{0}はユーザコード形式で入力してください。
zh_CN	□以用□代□的形式□入{0}。

date

値がアカウントコンテキストの日時表示形式に含まれている「日付（入力）」と同じフォーマットであるかを検証します。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

日付と時刻の形式マスタの詳細については「設定ファイルリファレンス 」を参照してください。
「日付（入力）」を示すフォーマット名は `IM_DATETIME_FORMAT_DATE_INPUT` です。

実際に使用される日付と時刻の形式は、アカウントコンテキストのプロパティから取得します。
アカウントコンテキストに設定されているプロパティの解決順序については、「アクセスコンテキスト仕様書」-「アカウントコンテキスト」を参照してください。

設定値

trueまたはfalse

設定例

```
date: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	{0}is not a valid date format. ({1})
ja	{0}は有効な日付形式({1})ではありません。
zh_CN	{0}是不是一个有效的日期格式({1})。

time

値がアカウントコンテキストの日時表示形式に含まれている「時刻（入力）」と同じフォーマットであるかを検証します。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

日付と時刻の形式マスタの詳細については「設定ファイルリファレンス 」を参照してください。
「時刻（入力）」を示すフォーマット名は `IM_DATETIME_FORMAT_TIME_INPUT` です。

実際に使用される日付と時刻の形式は、アカウントコンテキストのプロパティから取得します。
アカウントコンテキストに設定されているプロパティの解決順序については、「アクセスコンテキスト仕様書」-「アカウントコンテキスト」を参照してください。

設定値

trueまたはfalse

設定例

```
time: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	{0}is not a valid time format. ({1})
ja	{0}は有効な時刻形式({1})ではありません。
zh_CN	{0}是不是一个有效的□□格式({1})。

datetime

値がアカウントコンテキストの日時表示形式に含まれている「日付（入力）」＋半角スペース＋「時刻（入力）」と同じフォーマットであるかを検証します。
パラメータが存在しないまたは値が空の場合は検証を行いません。
設定値がfalseの場合は検証を行いません。

日付と時刻の形式マスタの詳細については「設定ファイルリファレンス 」を参照してください。
「日付（入力）」を示すフォーマット名は `IM_DATETIME_FORMAT_DATE_INPUT`、「時刻（入力）」を示すフォーマット名は `IM_DATETIME_FORMAT_TIME_INPUT` です。

実際に使用される日付と時刻の形式は、アカウントコンテキストのプロパティから取得します。
アカウントコンテキストに設定されているプロパティの解決順序については、「アクセスコンテキスト仕様書」-「アカウントコンテキスト」を参照してください。

設定値

trueまたはfalse

設定例

```
datetime: true
```

エラーメッセージ

ロケール	エラーメッセージ
en	{0}is not a valid date-time format. ({1})
ja	{0}は有効な日時形式({1})ではありません。

zh_CN	{0}是不是一个有效的日期和□□格式({1})。

Storage

項目

- Storageとは
- Storageの種類
- ストリーミング
- プログラミング方法
  - ファイルアップロード
  - ファイルダウンロード

Storageとは

Storageは分散システムで intra-mart Accel Platform を利用しているときに、アップロードされたファイルやシステムで
共有したいファイル（主にデータファイル）を一元管理します。

> **コラム**
>
> 分散システムを構築する場合、各 Web Application Server で共有するディレクトリを設定しておく必要があります。
> 詳細はセットアップガイドを参照してください。

Storageの種類

- SystemStorage

  システムで利用するファイルを保存する領域です。
  主に、 intra-mart Accel Platform の基盤APIやアプリケーション内の処理で利用されます。

- PublicStorage

  アップロードされたファイルや利用者間で共有したいファイルを保存する領域です。
  Storageにファイルを保存する場合は、基本的にPublicStorageに保存します。

- SessionScopeStorage

  一時的にファイルを保存する領域です。
  処理の途中でアップロードされたファイルや保存されたデータを一時的に保存したい場合に利用します。
  SessionScopeStorageに保存されたファイルは、セッションの有効期間が切れたタイミングで自動的に削除されます。

ストリーミング

intra-mart Accel Platform ではStorageの保存されているデータをストリームで扱うことができるようになりました。

ストリームを利用することで、従来のintra-mart WebPaltformのように一度 Web Application Server のメモリ上に
ファイルデータを持つ必要がなくなり、サイズの大きいファイルのアップロードやダウンロードが行えるようになります。

> **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。

```
/**
 * ファイルをpublicStorage「sample」ディレクトリへアップロードします。
 * @param request リクエストオブジェクト
 */
function init(request) {
  // アップロードされたファイルを取得します。
  var upfile = request.getParameter("local_file");

  // アップロードされたファイルのバイナリストリームを取得します。
  upfile.openValueAsBinary(function(reader) {

    // ファイル保存先を指定したPublicStorageオブジェクトを生成します。
    var storage = new PublicStorage("sample", upfile.getFileName());

    // 保存先のバイナリストリームを生成します。
    storage.createAsBinary(function(writer, error) {
      if(error) {
        // 保存先のバイナリストリームが生成できなかった場合は例外処理を行ないます。
      } else {
        // アップロードされたファイルをPublicStorageに書き込みます。
        reader.transferTo(writer);
      }
    });
  });
}
```

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。

```
/**
 * sample.txtをクライアントへ送信します。
 * @param request リクエストオブジェクト
 */
function init(request) {
  var storage = new PublicStorage("sample.txt");
  if(!storage.isFile()) {
    // ファイルが存在しない場合は例外処理を行います。
  } else {
    // ファイルをクライアントへ送信します。
    Module.download.send(storage, storage.getName(), "text/plain");
  }
}
```

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
- 仕様
- サンプルプログラム
  - サンプル内容
  - プログラムソース
- ジョブの実行方法
  - ジョブ・ジョブネットを登録する
  - ジョブを実行する

ジョブスケジューラとは

ジョブスケジューラとは、ジョブと呼ばれる処理単位に事前にいつ実行するかというスケジュールを定義しておくことで
自動的に実行する機能です。１つの業務を定期的に複数回実行する場合や、大量データを扱うため時間かかる業務処理を
夜間に実行する場合などに利用します。

intra-mart Accel Platform のジョブスケジューラは、このような要求を実現するためサーバ上の Java や
サーバサイドJavaScript で構成された任意の業務処理をスケジュールで定義されたタイミングで
自動的に実行する機能や、実行状況の監視や実行結果の管理を行うための機能を提供します。

仕様

ジョブスケジューラの仕様については、ジョブスケジューラ仕様書を参照してください。

サンプルプログラム

サンプル内容

固定文字列”Hello.”に、”message”というキーに設定されたパラメータ値を連結して出力します。

プログラムソース

```
/**
 * @parameter message world!
 */
function execute() {
  let accountContext = Contexts.getAccountContext();
  let jobSchedulerContext = Contexts.getJobSchedulerContext();
  let message = jobSchedulerContext.getParameter('message');
  if (null == message) {
    return {
      status : 'error' ,
      message : 'パラメータにメッセージが存在しません。'
    };
  } else if ('' == message) {
    return {
      status : 'warning' ,
      message : 'メッセージが空です。'
    };
  }
  Debug.console('Hello. ' + message);
  return {
    status : 'success' ,
    message : 'ジョブが正常に実行されました。'
  };
}
```

サーバサイドJavaScript のジョブプログラムは、任意のJSファイルにexecute関数を記述します。
ジョブ開発者は、このexecute関数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。

```
//ジョブスケジューラマネージャの作成
var manager = new JobSchedulerManager();

var jobNames = {};

jobNames['ja'] = {
  name : 'サンプルジョブ',
  description : 'ジョブのサンプルです。'
};

var job = {
  categoryId : null, // ジョブカテゴリID。
  id : 'sample-job', // ジョブID。
  jobPath : '/app/job/sample_job', // 実行プログラムのパス。
  jobType : 'SCRIPT', // ジョブ実行言語。
  localizes : jobNames, // 国際化情報オブジェクト。
  parameters : {} // ジョブパラメータ。
}
//ジョブの登録
var result = manager.insertJob(job);
if (result.error) {
  // エラー処理
}
```

ここでは、ジョブネットを登録する例を示します。

```
//ジョブスケジューラマネージャの作成
var manager = new JobSchedulerManager();

var jobnetNames = {};

jobnetNames['ja'] = {
  name : 'サンプルジョブネット',
  description : 'ジョブネットのサンプルです。'
};

var useJobIds = [ 'sample-job' ];

var jobnet = {
  categoryId : null, // ジョブネットカテゴリID。
  disallowConcurrent : false, // 並列実行不可。
  id : 'sample-jobnet', // ジョブネットID。
  jobIds : useJobIds, // 実行ジョブ配列。(ジョブの実行順にジョブIDを設定した配列です。)
  localizes : jobnetNames, // 国際化情報オブジェクト。
  parameters : { message : 'world!' }, // ジョブネットパラメータ。
  useJobIds : useJobIds // このジョブネットで利用するジョブの配列。
}
//ジョブネットの登録
var result = manager.insertJobnet(jobnet);
if (result.error) {
  // エラー処理
}
```

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。

```
//ジョブスケジューラマネージャの作成
var manager = new JobSchedulerManager();

var triggerId = 'sample-trigger';

var result = manager.findTrigger(triggerId);
if (result.error) {
  var repeatTrigger = {
    id : triggerId, // トリガID。
    type : 'RepeatTrigger', // トリガ種別。
    enable : true, // 有効。
    jobnetId : 'sample-jobnet', // ジョブネットID。
    count : 1, // 実行回数。
    interval : null // 実行間隔(秒)。
  }
  // トリガーが存在しない場合は新規登録
  result = manager.insertTrigger(repeatTrigger);
} else {
  // 登録済みのトリガーを再登録して実行
  result = manager.updateTrigger(result.data);
}
if(result.error){
  //エラー処理
}
```

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
var manager = new JobSchedulerManager();

// 指定したジョブネットの即時実行
var result = manager.execute('sample-jobnet');
if (result.error) {
  // エラー処理
}
```

Lockサービス

項目
- Lockとは
- サンプルプログラム
- リクエストに紐付けたロック

Lockとは

Lockサービスはintra-martシステム全体で一意のロックを行う機能です。
特定の機能を使用不可能にしたい場合や処理の直列化を行いたい場合等に利用します。

> **コラム**
>
> ロックの利用状況は[システム管理者] - [アプリケーションロック一覧]画面より確認できます。
> 詳細は、システム管理者 操作ガイドを参照してください。

サンプルプログラム

ロックを開始した後、処理ロジックを実行して、最後にロックを開放する場合は以下のように実装します。
（このサンプルでは５秒間ロックが開始できなかった場合は例外をスローします。）

```
// ロックの開始
var lockId = "lock_key";
if (!NewLock.tryLock(lockId, 5)) {
   // ロックの開始に失敗
   return false;
}

//処理ロジック


//ロックの解放
if (!NewLock.unlock(lockId)) {
   // ロックの開放に失敗
   return false;
}
```

> **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.tryLockRequestScope(lockId, 5);
```

この関数を利用して開始されたロックは、レスポンスを返却する際に自動的に開放されます。
また、unlock()関数を使用して任意のタイミングで開放することもできます。
ロックの解放漏れを防ぎたい場合などは、この関数を利用してロックを開始してください。

> **注意**
>
> この関数はリクエストにロックを紐付けて自動開放を行うものなので、非同期タスクや
> ジョブスケジューラで実行されるジョブの処理内で利用することはできません。

Cacheサービス

項目

- Cacheとは
- 仕様
- プログラミング方法

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となります。

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となります。
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は無効となります。
max-btyes-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。

プログラミング方法

```
function getUsers() {
  // cacheインスタンスを生成します。引数'myCache'はim-ehcache-configに定義されているCache名を指定する必要があります。
  var cache = new Cache('myCache');
  // キャッシュから情報の取得を試みます。取得できた場合はその値を返却します。
  var users = cache.get('key');
  if(users != null){
    return users;
  }
  // キャッシュに情報が存在しなかったため、データベースから情報の取得を行います。
  var database = new TenantDatabase();
  var result = database.select('SELECT user_cd FROM b_m_account_b');
  // データベースから情報の取得に失敗した場合は空の配列を返却します。
  // 通常はログ出力、エラー情報の返却が必要となります。
  目的に合わせて適切に実装を行ってください。
  if(result.error){
    return [];
  }
  // 取得した情報をキャッシュに格納した後、返却します。
  users = [];
  for(var i = 0, length = result.data.length; i < length; i++){
    var record = result.data[i];
    users.push(record.user_cd);
  }
  // キャッシュに格納し、返却します。
  cache.put('key', users);
  return users;
}
```

データベースからデータを取得する際の例です

コラム

キャッシュの生存期間等はすべて設定ファイルに記述された内容に従います。
明示的にキャッシュの削除が必要となる場合はCache#removeまたはCache#removeAllを呼び出すことにより削除可能です。

コラム

「max-bytes-memory」及び、「max-bytes-disk」属性が設定されている場合、Cacheにオブジェクトを登録する際に、そのオブジェクトのサイズの計算処理が行われます。
この際、登録するオブジェクトが、別のオブジェクトの参照を大量に持つ場合、計算処理に時間がかかりパフォーマンスの低下の原因となる可能性があります。
登録するオブジェクトが1000以上の参照を持つ場合、下記のようなメッセージがログに出力されます。

```
The configured limit of 1,000 object references was reached while attempting to calculate the size of the object graph.
Severe performance degradation could occur if the sizing operation continues.
This can be avoided by setting the CacheManger or Cache <sizeOfPolicy> elements maxDepthExceededBehavior to
"abort" or adding stop points with @IgnoreSizeOf annotations.
If performance degradation is NOT an issue at the configured limit, raise the limit value using the CacheManager or
Cache <sizeOfPolicy> elements maxDepth attribute.
For more information, see the Ehcache configuration documentation.
```

このログが出力される場合は、キャッシュに格納するオブジェクトの構成を見直すか、「max-bytes-memory」または、「max-bytes-disk」の代わりに、「max-elements-on-memory」または「max-elements-on-disk」の利用を検討して下さい。

ショートカットアクセス機能

項目

- ショートカットアクセス機能とは
- ショートカットアクセス URL
- ショートカット ID の作成
- ショートカット拡張検証機能
  - 標準の検証コードについて

ショートカットアクセス機能とは

ショートカットアクセス機能は、初期アクセス URL にショートカットアクセス用の URL を指定することによって、ログイン後の画面を任意の画面に取り替えることができる機能です。
ショートカットアクセス機能を用いると、ログイン後の画面で任意のページを表示することが可能になります。

ショートカットアクセス URL

ショートカットアクセス用の URL は、ショートカットIDを含む以下の URL になります。

```
http:// <server> / <context-path> /user/shortcut/ <ショートカットID>
```

<記述例>

```
http://localhost/imart/user/shortcut/5i4deh98wou5uuc
```

ショートカット ID の作成

ショートカット ID は、表示するページの情報およびセキュリティの情報に紐づく ID となります。
ショートカット IDは、表示するページの情報およびセキュリティの情報を指定してAPI を用いて作成します。
ログイン後に表示する画面に、/sample/shortcut を指定する場合のショートカット IDの作成手順を説明します。

```
// ショートカットマネージャの作成
var manager = new ShortCutManager();

// ショートカット情報の作成
var shortCutInfo = {
   url : '/sample/shortcut', // 表示するURL
   urlParams : { // 表示する URL に渡すパラメータの設定(任意指定)
      arg1 : 'value1',
      arg2 : 'value2'
   },
   allowUsers : ['guest', 'ueda'], // 表示許可を行うユーザ
   isAuth : true, // ログイン認証が必要かどうか？
   validEndDate : manager.addValidEndDate(10) // この情報の有効期限(作成時から１０日間有効)
};

// ショートカットID 作成
var shortCutId = manager.createShortCut(shortCutInfo);
```

ショートカット拡張検証機能

拡張検証機能では、ショートカット情報の URL の表示を許可ユーザ以外で、ログインしたユーザに対して検証プログラム利用して、ページの表示許可を与えるかどうかの判定を追加で行う機能です。
拡張検証コードは、検証プログラムのコード名を表します。
拡張検証パラメータは、検証プログラムに渡されるパラメータとなります。

```
// ショートカットマネージャの作成
var manager = new ShortCutManager();

// ショートカット情報の作成
var shortCutInfo = {
   url : '/sample/shortcut', // 表示するURL
   urlParams : { // 表示する URL に渡すパラメータの設定(任意指定)
      arg1 : 'value1',
      arg2 : 'value2'
   },
   allowUsers : ['guest', 'ueda'], // 必ず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。）
   isAuth : true, // 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。）
   validEndDate : manager.addValidEndDate(10), // この情報の有効期限(作成時から１０日間有効)

   validationCode : 'RoleUser', // 追加でユーザが許可されるかどうかの検証コード
   validationParam : '' // 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
var shortCutId = manager.createShortCut(shortCutInfo);
```

> **コラム**
>
> 検証プログラムはjavaでのみ作成できます。
> 検証プログラムとコード名の紐付けは、ショートカットアクセス設定の定義ファイルで行います。
> 作成方法および紐付けについては、「SAStruts+S2JDBC プログラミングガイド」を参照してください。

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

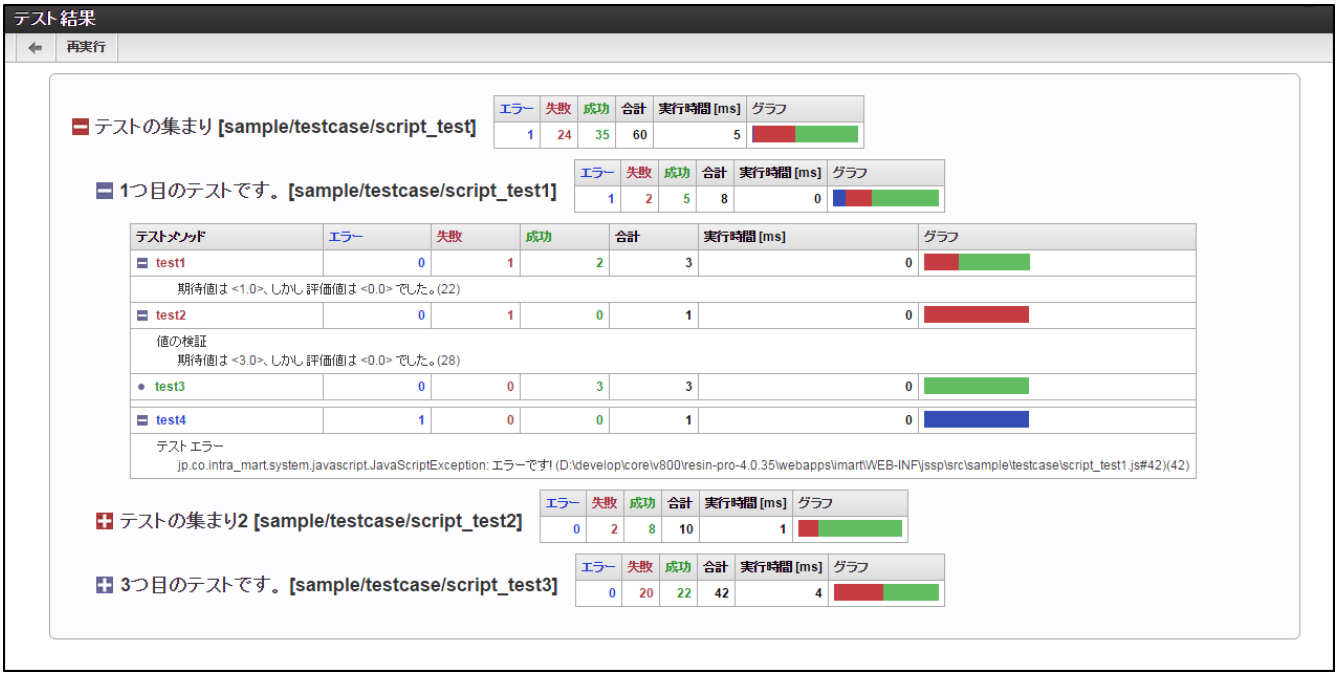
項目

- サーバサイド単体テストとは
- アプリケーションサーバの構築
- テストケースについて
  - テストケースでの関数の種類と実行順序
  - テストケースの作成
  - テストスィートの作成
- テストケースの配置と実行
  - テストケースの配置
  - テストケースの実行

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スクリプト開発モデルで作成したファンクションコンテナ(js)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。

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。

テストケースについて

テストケースは、スクリプト開発モデルで作成します。
テストケース作成には、いくつかのルールが存在します。

テストケースでの関数の種類と実行順序

特殊関数の種類

特殊関数は、単体テストを行う上で特別な意味を持つ関数です。

特殊関数	説明
testXXXXXX()	test から始まる関数を検索して、随時実行します。 この関数内に評価関数を用いて、テスト内容を記述します。 各関数の実行する順序に決まりはありません。
setUp()	各 testXXXXXX（）が実行される前に実行される関数です。 存在しない場合は実行されません。
tearDown()	各 testXXXXXX（）が実行された後に実行される関数です。 存在しない場合は実行されません。
oneTimeSetUp()	テストケースファイルがロードされた直後に一度だけ実行されます。 存在しない場合は実行されません。
oneTimeTearDown()	テストケース内のすべての testXXXXXX（）の実行が終わった後の一番最後に一度だけ実行されます。 存在しない場合は実行されません。
defineTestSuite()	この関数が定義されていた場合、このファイルをテストスィートとして扱います。 「テストスィート」とは、複数のテストケースをまとめて実行するための機能です。 その他の関数はすべて無視されます。 テストスィートのファイルを作成する場合は、この関数だけを定義します。

評価関数

評価関数は、テストの結果を評価するために利用する関数です。
この関数を用いることで、テスト結果として情報が収集されます。

評価関数	説明
JsUnit.assert([comment], actual)	評価値(actual)が true であることをチェックします。 コメント(comment)は、評価に失敗した場合、結果に表示されます。省略可能です。
JsUnit.assertTrue([comment], actual)	評価値(actual)が true であることをチェックします。 コメント(comment)は、評価に失敗した場合、結果に表示されます。省略可能です。
JsUnit.assertEquals([comment], expect,actual)	評価値(actual)と期待値(expect)が同じであることをチェックします。 コメント(comment)は、評価に失敗した場合、結果に表示されます。省略可能です。
JsUnit.assertNull([comment], actual)	評価値(actual)が null であることをチェックします。 コメント(comment)は、評価に失敗した場合、結果に表示されます。省略可能です。
JsUnit.assertUndefined([comment], expect, actual)	評価値(actual)が undefined であることをチェックします。 コメント(comment)は、評価に失敗した場合、結果に表示されます。省略可能です。
JsUnit.assertNaN([comment], actual)	評価値(actual)が NaN であることをチェックします。 コメント(comment)は、評価に失敗した場合、結果に表示されます。省略可能です。
JsUnit.assertFalse([comment], actual)	評価値(actual)が false であることをチェックします。 コメント(comment)は、評価に失敗した場合、結果に表示されます。省略可能です。
JsUnit.assertNotEquals([comment], expect,actual)	評価値(actual)と期待値(expect)が同じでないことをチェックします。 コメント(comment)は、評価に失敗した場合、結果に表示されます。省略可能です。
JsUnit.assertNotNaN([comment], actual)	評価値(actual)が NaN でないことをチェックします。 コメント(comment)は、評価に失敗した場合、結果に表示されます。省略可能です。

評価関数	説明
JsUnit.assertNotNull([comment], actual)	評価値(actual)が null でないことをチェックします。 コメント(comment)は、評価に失敗した場合、結果に表示されます。省略可能です。
JsUnit.assertNotUndefined([comment], actual)	評価値(actual)が undefined でないことをチェックします。 コメント(comment)は、評価に失敗した場合、結果に表示されます。省略可能です。
JsUnit.fail(message)	評価に失敗させます。 メッセージ(message)は結果に表示されます。

実行順序

テストケース内の関数の実行順序のイメージは以下の通りです。
テストケースファイルにテスト関数として testSample1() と testSample2() が記述されていた場合を例にします。
なお、testSample1() と testSample2()は入れ替わる場合があります。（順位不同）

1. oneTimeSetUp()
2. setUp()
3. testSample1()
4. tearDown()
5. setUp()
6. testSample2()
7. tearDown()
8. oneTimeTearDown()

テストケースの作成

テストケースは、実際の単体内容を記述します。
特殊関数（defineTestSuite()を除く）および評価関数を用いて、テスト内容を記述します。

テスト対象ファイル(user/test/source.js)

```
// ２つの値を加算します。
function calcPlus(x, y) {
 return x + y;
}
```

テストケース

```
// テスト対象 user/test/source.js をオブジェクトとしてロードします。
let module = JsUnit.loadScriptModule("user/test/source");

function testSample1() {
 // テスト対象の関数を呼び出します。
 let result = module.calcPlus(1,2);
 // 関数の結果が正しいかテストします。
 JsUnit.assertEquals(3,result);
 // 関数の結果が正しいかテストします。(コメント付)
 JsUnit.assertEquals("足し算のテスト（1 + 2）",3,result);
}
```

注意

テストケー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。
  画面遷移が発生する API （Debug.browse(), redirect（）, forward（）, Module.alert.* など）を記述した場合、指定した画面に遷移するため正常に動作できません。
  別の関数に分けるなどして、テストを行ってください。

テストスィートの作成

テストスィートは、複数のテストケースをまとめて実行することができます。

テストスィートの作成には、defineTestSuite 関数を定義します。
defineTestSuite 関数内でJsTestSuiteオブジェクトを作成し、グループ化したいテストケースおよびテストスィートファイルを追加します。

```
function defineTestSuite() {

  // JsTestSuiteオブジェクトの作成
  // JsTestSuiteオブジェクトの第一引数は名称です。
  let suite = new JsTestSuite("テストの集まり");

  // テストケースまたはテストスィートファイルを追加します。
  // addTest関数の第一引数は名称です。
  // addTest関数の第二引数はテストケースまたはテストスィートファイルのパス（フルパス）です。
  suite.addTest("１つ目のテストです。","sample/testcase/test2");
  suite.addTest("２つ目のテストです。","sample/testcase/test3");
  suite.addTest("３つ目のテストです。","sample/testcase/test_suite2");

  // テストスィートオブジェクトを返却します。
  return suite;
}
```

テストケースの配置と実行

テストケースの配置

作成したテスト対象ファイル、テストケースファイルおよびテストスウィート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/jssp/src

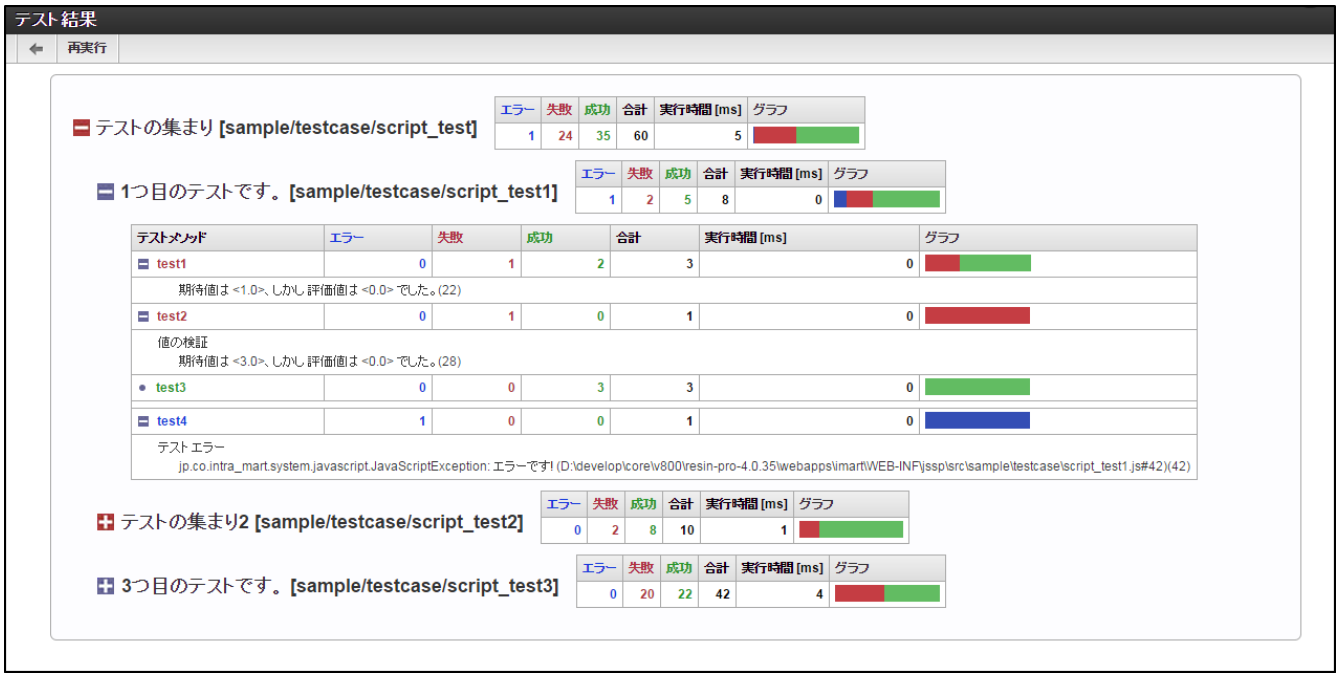
テストケースの実行

1. 任意にユーザでログインします。
2. 「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。

3. テストケースパスに、実行したいテストケースまたはテストスウィートファイルのパスを入力します。
4. 実行ボタンをクリックします。

5. 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。
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。
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。
合計	テスト中に評価関数で評価した数およびエラーの数の合計です。
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。

共通ライブラリの作成

intra-mart Accel Platform では、ユーザが<IMART>タグや、グローバル関数、API などを自由に定義し、利用することができます。
本項では、これらの設定方法や利用方法などを紹介します。

拡張APIの作成

intra-mart Accel Platform では、API に関してもユーザが定義することが可能です。
ユーザ定義の拡張API はユーザ独自のオブジェクトを実装した js ファイルと initializer-XXX.xml によって登録することができます。
オブジェクトの実装の仕方によって、静的な関数を定義した API、および new 演算子を用いてインスタンスを生成する形式の API を実装することができます。
本項では、jsAPIの登録から利用方法までを記載しています。

jsAPI の登録の流れ
- 前提条件
- intra-mart e Builder for Accel Platform で静的な関数の登録
  - ステップ1：initializerの作成
  - ステップ2：initializerに設定したjsファイルの作成
  - ステップ3：モジュールのデプロイ
  - ステップ4：登録したAPIの呼び出し
- intra-mart e Builder for Accel Platform でインスタンス オブジェクトの登録
  - ステップ1：initializerの作成
  - ステップ2：initializerに設定したjsファイルの作成
  - ステップ3：モジュールのデプロイ
  - ステップ4：登録したAPIの呼び出し

前提条件

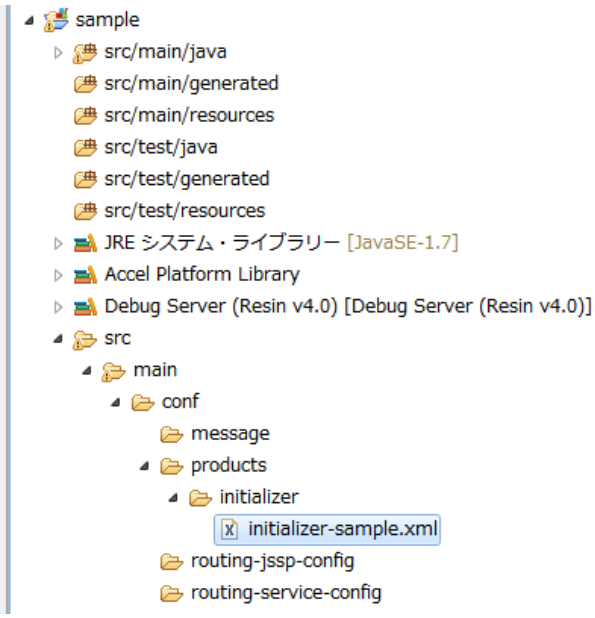
- intra-mart e Builder for Accel Platform をインストール済みであること。
- モジュール・プロジェクトの作成が完了していること。

intra-mart e Builder for Accel Platform で静的な関数の登録

以下の手順で静的な関数を持つオブジェクトを登録します。

ステップ1：initializerの作成

src/main/conf/products/initializer/initializer-[プロジェクト名].xmlを作成します。



> **コラム**
>
> initializer-[プロジェクト名].xml は、厳密にはプロジェクト名ではなくモジュールのIDから決定されます。
> モジュールIDの”.”で分割してその末尾になります。
> 例として、モジュールIDが「org.example.foo」場合は「initializer-foo.xml」をという名前で initializer を定義してください。
> intra-mart e Builder for Accel Platform で開発する場合はプロジェクト名と考えて問題ありません。

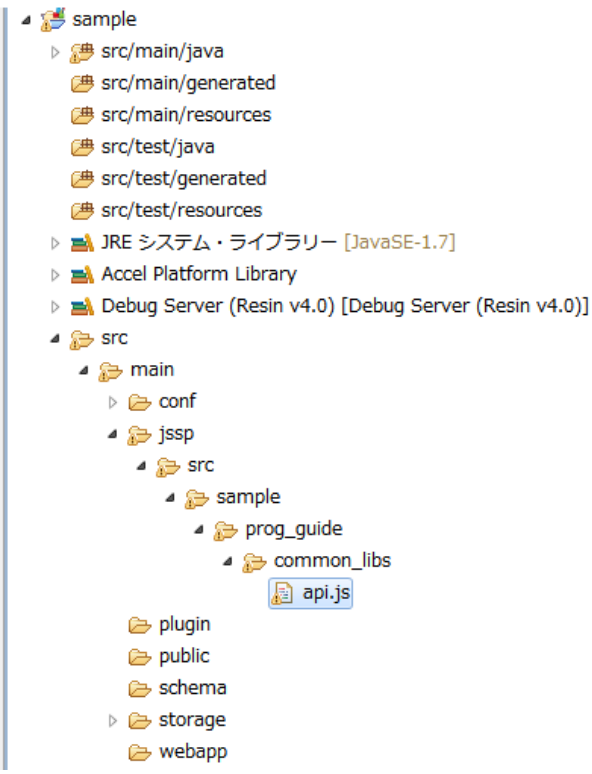
> **注意**
>
> intra-mart e Builder for Accel Platform でプロジェクトを作成する際は 「im」 で始まる名前は利用できません。

ファイルに内容を加えます。記述方法は、<java-script-api><api-script>オブジェクトを定義した js ファイルパス#オブジェクト</api-script></java-script-api>です。

```
<?xml version="1.0" encoding="UTF-8"?>
<initializer-config
   xmlns="http://intra-mart.co.jp/system/secure/product/initializer/config/initializer-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://intra-mart.co.jp/system/secure/product/initializer/config/initializer-config ../../../schema/initializer-config.xsd ">

 <java-script-api>
  <api-script>sample/prog_guide/common_libs/api#StaticApi</api-script>
 </java-script-api>

</initializer-config>
```

ステップ2：initializerに設定したjsファイルの作成

上記のxmlにて設定したjsファイルを作成します。
この例では、プロジェクトの src/main/jssp/src/sample/prog_guide/common_libs 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「 api.js 」ファイルを作成し、 以下のように実装をします。



```
/**
* staticオブジェクトの作成
 */
function StaticApi(){}

/**
 * hogeという名前の関数を登録します。
 */
StaticApi.hoge = function(){
   //ここに処理を書きます
   Debug.console("この関数は静的関数です。");
}
```

ステップ3：モジュールのデプロイ

エクスポートしたユーザモジュールを含んだwarファイルを作成します。
そのwarファイルをデプロイして intra-mart Accel Platform を再起動します。

> **注意**
>
> initializer は warファイルに含まれるモジュール構成を元に読み込まれるので、一旦ユーザモジュールにしてwarファイルに組み込んだ形でないと動作しません。
> ユーザモジュールに関しては、『 intra-mart e Builder for Accel Platform アプリケーション開発ガイド / immファイルのエクスポート 』を参照してください。

ステップ4：登録したAPIの呼び出し

登録したAPIは、任意のjsファイルから呼び出すことができます。

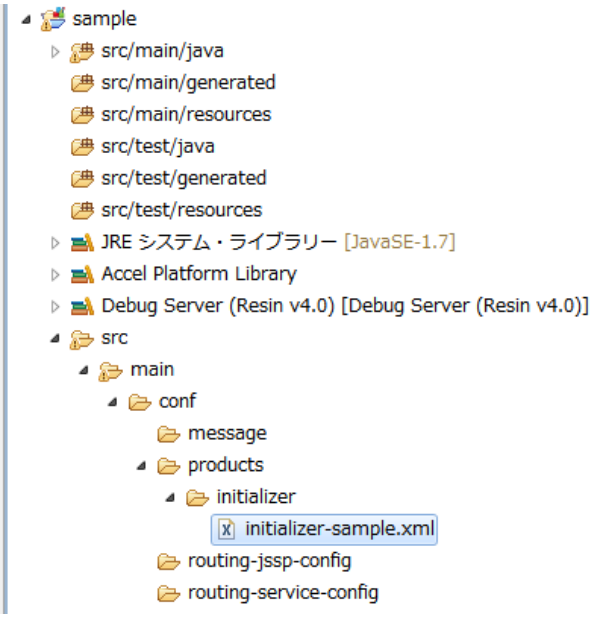
```
function init(request){
 StaticApi.hoge();
}
```

intra-mart e Builder for Accel Platform でインスタンス オブジェクトの登録

以下の手順でインスタンスな関数を持つオブジェクトを登録します。

ステップ1：initializerの作成

src/main/conf/products/initializer/initializer-[プロジェクト名].xmlを作成します。



> **コラム**
>
> initializer-[プロジェクト名].xml は厳密には、プロジェクト名ではなくモジュールのIDから決定されます。
> モジュールIDの”.”で分割してその末尾になります。
> 例として、モジュールIDが「org.example.foo」場合は「initializer-foo.xml」をという名前で initializer を定義してください。
> intra-mart e Builder for Accel Platform で開発する場合はプロジェクト名と考えて問題ありません。

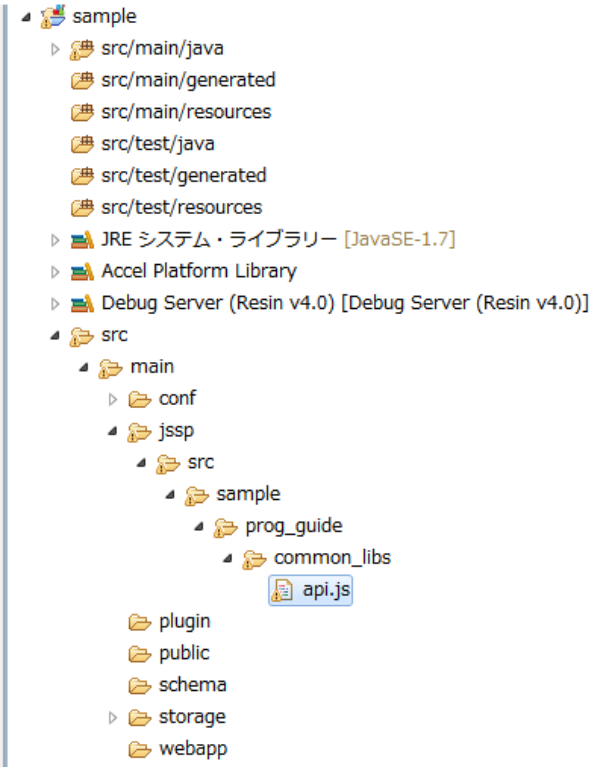
> **注意**
>
> intra-mart e Builder for Accel Platform でプロジェクトを作成する際は 「im」 で始まる名前は利用できません。

ファイルに内容を加えます。記述方法は、<java-script-api><api-script>オブジェクトを定義した js ファイルパス#オブジェクト</api-script></java-script-api>です。

```
<?xml version="1.0" encoding="UTF-8"?>
<initializer-config
   xmlns="http://intra-mart.co.jp/system/secure/product/initializer/config/initializer-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://intra-mart.co.jp/system/secure/product/initializer/config/initializer-config ../../../schema/initializer-config.xsd ">

 <java-script-api>
  <api-script>sample/prog_guide/common_libs/api#InstanceApi</api-script>
 </java-script-api>

</initializer-config>
```

ステップ2：initializerに設定したjsファイルの作成

上記のxmlにて設定したjsファイルを作成します。
この例では、プロジェクトの src/main/jssp/src/sample/prog_guide/common_libs 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「 api.js 」ファイルを作成し、 以下のように実装をします。



```
/**
* インスタンスオブジェクトの作成
 */
function InstanceApi(){}

InstanceApi.prototype.hoge = function(){
        //ここに処理を書きます
        Debug.console("この関数はインスタンスオブジェクトの関数です。");
};
```

ステップ3：モジュールのデプロイ

エクスポートしたユーザモジュールを含んだwarファイルを作成します。
そのwarファイルをデプロイして intra-mart Accel Platform を再起動します。

> **注意**
>
> initializer は warファイルに含まれるモジュール構成を元に読み込まれるので、一旦ユーザモジュールにしてwarファイルに組み込んだ形でないと動作しません。
> ユーザモジュールに関しては、『 intra-mart e Builder for Accel Platform アプリケーション開発ガイド / immファイルのエクスポート 』を参照してください。

ステップ4：登録したAPIの呼び出し

登録したAPIは、任意のjsファイルから呼び出すことができます。

```
function init(request){
 var api = new InstanceApi();
 api.hoge();
}
```

グローバル関数の作成

intra-mart Accel Platform には JavaScript で記述したユーザ定義関数を「グローバル関数」として登録する方法があります。
グローバル関数には、Procedure.define 関数と initializer-XXX.xml設定ファイルの2通りの定義方法があります。

グローバル関数の登録の流れ
- 前提条件
- 設定ファイルでグローバル関数を登録
  - ステップ1：initializerの作成
  - ステップ2：initializerに設定したjsファイルの作成
  - ステップ3：モジュールのデプロイ
  - ステップ4：登録したグローバル関数の呼び出し
- Procedure.define 関数でグローバル関数を登録
  - ステップ1：任意の js ファイルにユーザ定義関数を格納する
  - ステップ2：起動時にメモリ上に格納される設定を行う
  - ステップ3：モジュールのデプロイ
  - ステップ4：登録したグローバル関数の呼び出し

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。
- モジュール・プロジェクトの作成が完了していること。

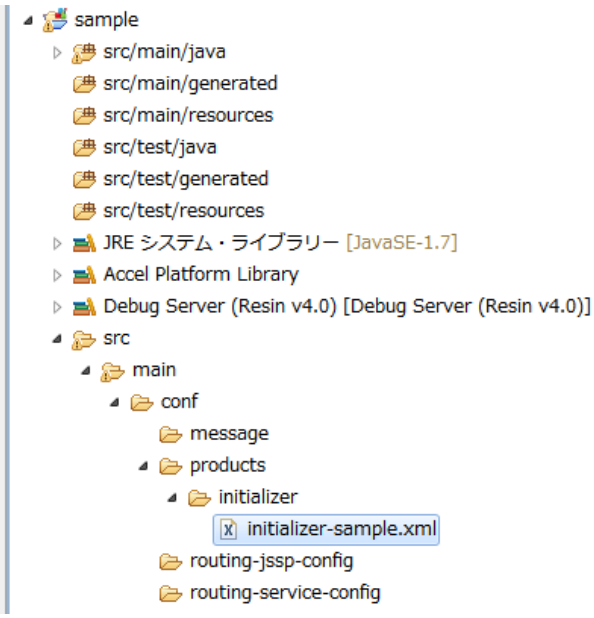
設定ファイルでグローバル関数を登録

以下の手順でグローバル関数を登録します。

> **注意**
>
> 関数名が被らないように注意してください。
> グローバルとして関数を定義する場合、特定のプログラムにおいて同名の関数が存在した場合等に影響を与える可能性があります。

ステップ1：initializerの作成

src/main/conf/products/initializer/initializer-[プロジェクト名].xmlを作成します。



> **コラム**
>
> initializer-[プロジェクト名].xml は、厳密にはプロジェクト名ではなくモジュールのIDから決定されます。
> モジュールIDの”.”で分割してその末尾になります。
> 例として、モジュールIDが「org.example.foo」場合は「initializer-foo.xml」をという名前で initializer を定義してください。
> intra-mart e Builder for Accel Platform で開発する場合はプロジェクト名と考えて問題ありません。

> **注意**
>
> intra-mart e Builder for Accel Platform でプロジェクトを作成する際は 「im」 で始まる名前は利用できません。

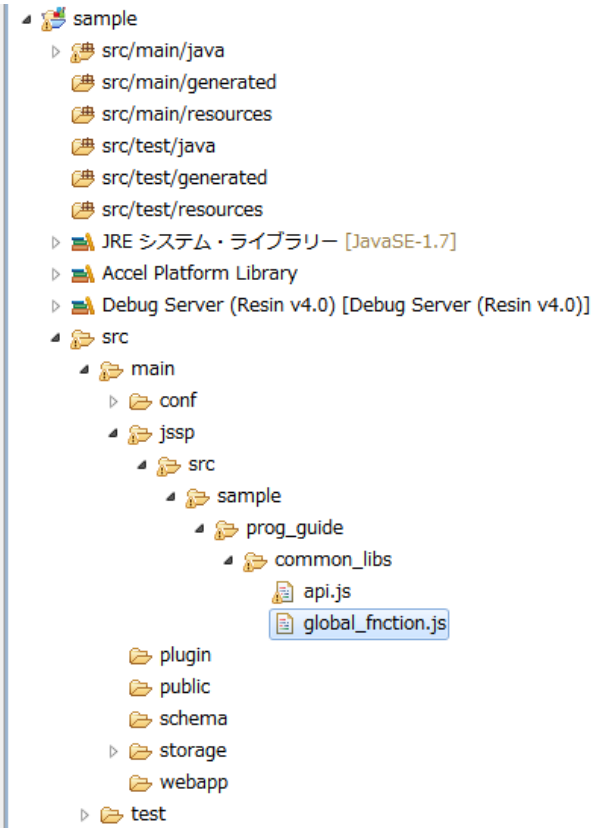
ファイルに内容を加えます。記述方法は、<java-script-api><global-function-script>グローバル関数を定義する js ファイルのパス#実行関数</global-function-script></java-script-api>です。

```
<?xml version="1.0" encoding="UTF-8"?>
<initializer-config
   xmlns="http://intra-mart.co.jp/system/secure/product/initializer/config/initializer-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://intra-mart.co.jp/system/secure/product/initializer/config/initializer-config ../../../schema/initializer-config.xsd ">

 <java-script-api>
  <global-function-script>sample/prog_guide/common_libs/global_fnction#global_fnction1</global-function-script>
 </java-script-api>

</initializer-config>
```

> **注意**
>
> 拡張APIの登録を行っている場合は、<global-function-script>タグは<api-script>タグよりも先に記述してください。

ステップ2：initializerに設定したjsファイルの作成

上記のxmlにて設定したjsファイルを作成します。
この例では、プロジェクトの src/main/jssp/src/sample/prog_guide/common_libs 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「 global_fnction.js 」ファイルを作成し、 以下のように実装をします。




```
/**
 * 共通関数作成
 */
function global_fnction1(valueA, valueB){
//ここに処理を書きます
 return valueA - valueB;
}
```

ステップ3：モジュールのデプロイ

エクスポートしたユーザモジュールを含んだwarファイルを作成します。
そのwarファイルをデプロイして intra-mart Accel Platform を再起動します。

> **注意**
>
> initializer は warファイルに含まれるモジュール構成を元に読み込まれるので、一旦ユーザモジュールにしてwarファイルに組み込んだ形でないと動作しません。
> ユーザモジュールに関しては、『 intra-mart e Builder for Accel Platform アプリケーション開発ガイド / immファイルのエクスポート 』を参照してください。

ステップ4：登録したグローバル関数の呼び出し

登録したグローバル関数を、任意のjsファイルから呼び出すことができます。

```
function init(request){
 Debug.console(global_fnction1(5,3));
}
```

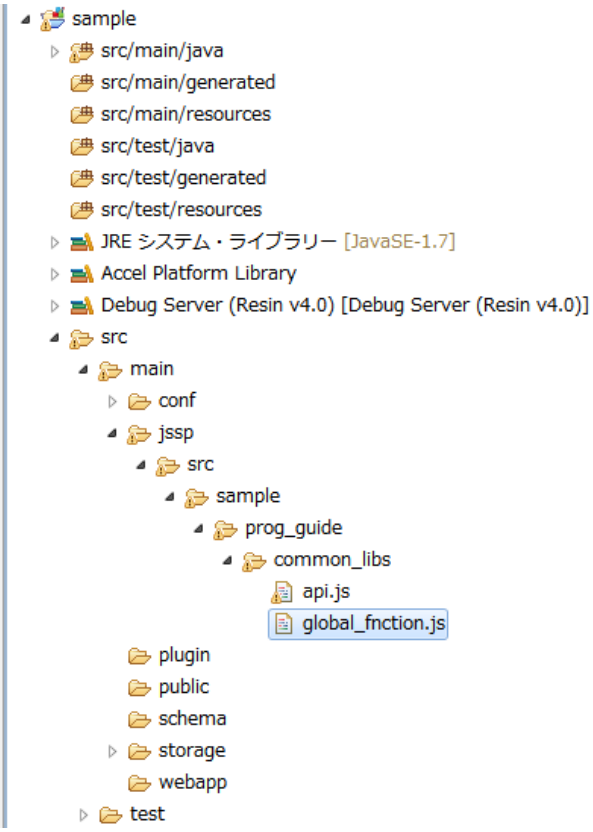
Procedure.define 関数でグローバル関数を登録

以下の手順でグローバル関数を登録します。

> **注意**
>
> 関数名が被らないように注意してください。
> グローバルとして関数を定義する場合、特定のプログラムにおいて同名の関数が存在した場合等に影響を与える可能性があります。

ステップ1：任意の js ファイルにユーザ定義関数を格納する

この例では、プロジェクトの src/main/jssp/src/sample/prog_guide/common_libs 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「 global_fnction.js 」ファイルを作成し、 以下のように実装をします。




```
//global_fnction2 という名前で共通関数を登録する
Procedure.define("global_fnction2", global_fnction2);

/**
 * 共通関数作成
 */
function global_fnction2( valueA, valueB ) {
   //ここに処理を書きます
   return valueA + valueB;
}
```

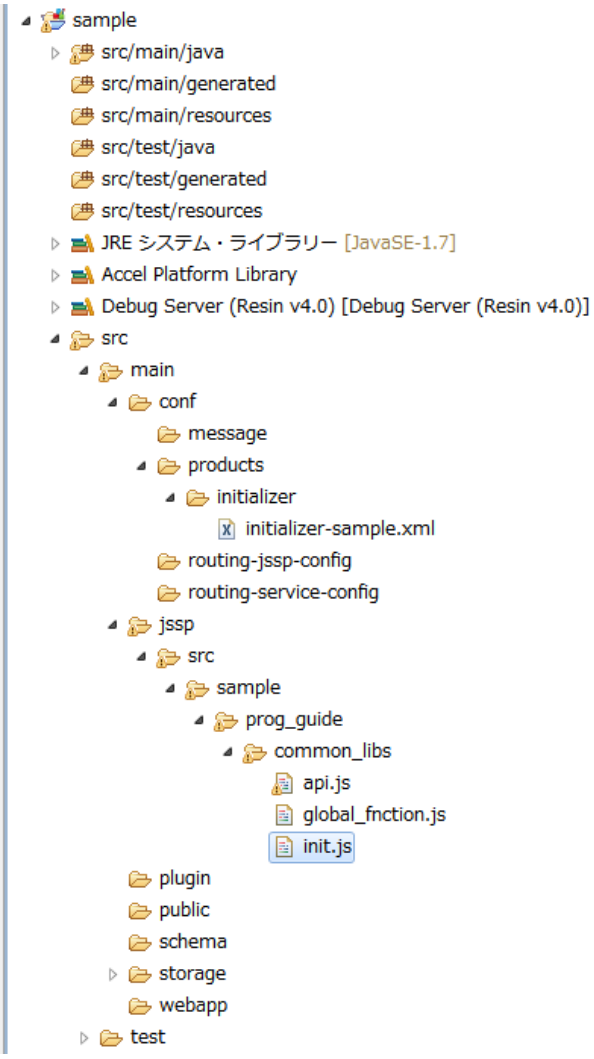
ステップ2：起動時にメモリ上に格納される設定を行う

ユーザ定義関数を格納した js ファイルについて、intra-mart起動後にメモリ上に格納されるよう js ファイルを取り込む記述をします。
intra-mart起動時に読み込まれる js ファイルは　src/init.js ですが、initializer-XXX.xml設定ファイルに設定を書くことで任意の js ファイルを読み込むことができます。

- src/init.js に記載する場合は以下のようになります。

```
//共通関数格納ファイルの取り込み
include("sample/prog_guide/common_libs/global_fnction");

/**
 * Initialize function for user-application.
 * @param nothing
 * @return void
 */
function init(){
 return;
}


/* END OF FILE */
```

- initializer-XXX.xml設定ファイルを使う場合は以下のようになります。
  - initializer と js ファイルを作成
    initializer-sample.xml と init.js を作成します。
    この例では、プロジェクトの src/main/jssp/src/sample/prog_guide/common_libs 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「 init.js 」ファイルを作成します。



  - initializerに起動時に呼ばれるファイルを編集
    記述方法は、<initializer><script-name>起動時に読み込まれる js ファイルのパス</script-name></initializer>です。

```
<?xml version="1.0" encoding="UTF-8"?>
<initializer-config
   xmlns="http://intra-mart.co.jp/system/secure/product/initializer/config/initializer-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://intra-mart.co.jp/system/secure/product/initializer/config/initializer-config
../../../schema/initializer-config.xsd ">

   <initializer>
      <script-name>sample/prog_guide/common_libs/init</script-name>
   </initializer>

</initializer-config>
```

- 上記で設定したファイルに、ファイル読み込み処理を追加
  include関数は 第一引数で指定された js ファイル内の init() 関数を実行し、その結果を返却します。
  第一引数は実行対象 js ファイルパスを拡張子なしで指定します。

```
//共通関数格納ファイルの取り込み
include("sample/prog_guide/common_libs/global_fnction");
function init(){
}
```

> **注意**
>
> initializer は起動中に処理が走るため全てのAPIが利用可能な状態ではありません。
> 例）DBに接続に行うなど
> Procedure#defineなど起動時に必要な処理を定義してください。

ステップ3：モジュールのデプロイ

エクスポートしたユーザモジュールを含んだwarファイルを作成します。
そのwarファイルをデプロイして intra-mart Accel Platform を再起動します。
ステップ2にて init.jsでファイルを取り込む記述をした場合は、intra-mart Accel Platform の再起動のみで問題ありません。

> **注意**
>
> initializer は warファイルに含まれるモジュール構成を元に読み込まれるので、一旦ユーザモジュールにしてwarファイルに組み込んだ形でないと動作しません。
> ユーザモジュールに関しては、『 intra-mart e Builder for Accel Platform アプリケーション開発ガイド / immファイルのエクスポート 』を参照してください。

ステップ4：登録したグローバル関数の呼び出し

Procedure.define(“関数名”,function) で登録した関数は Procedure.関数名() で呼び出すことができます。

```
function init(request){
 Debug.browse(Procedure.global_fnction2(5,3));
}
```

拡張<IMART>タグの作成

intra-mart Accel Platform で既に用意されている<IMART>タグ以外に、ユーザが定義し、独自の機能を持たせた任意の<IMART>タグを拡張<IMART>タグと言います。
具体的には、<IMART>タグに対して、新しい type 属性を定義し、その type 属性が要求された時に、実際に処理を実行するタグ関数を登録することで、html 側で API を拡張できる機能です。
拡張<IMART>タグには、Imart.defineType 関数と initializer-XXX.xml設定ファイルの2通りの定義方法があります。

拡張<IMART>タグの登録の流れ
- 前提条件
- 設定ファイルで拡張<imart>タグを登録
  - ステップ1：initializerの作成
  - ステップ2：initializerに設定したjsファイルの作成
  - ステップ3：モジュールのデプロイ
  - ステップ4：登録した拡張<IMART>タグの呼び出し
- Imart.defineType 関数で拡張<imart>タグを登録
  - ステップ1：任意の js ファイルにユーザ定義関数を格納する
  - ステップ2：intra-mart Accel Platform の再起動
  - ステップ3：登録した拡張<IMART>タグの呼び出し

前提条件

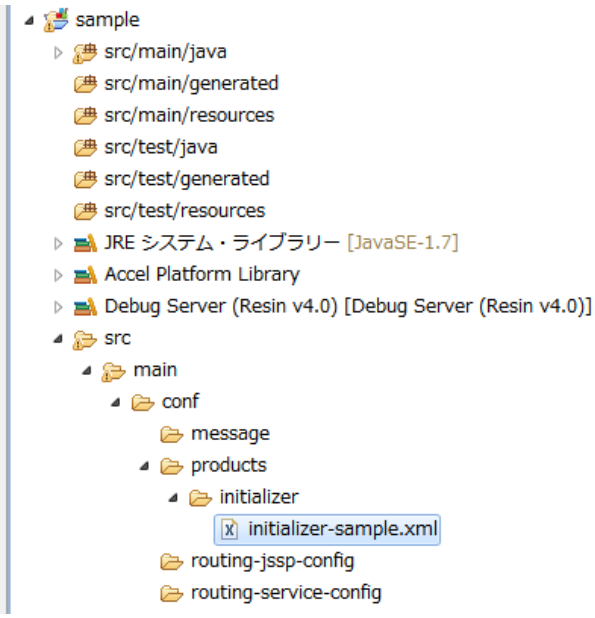
- intra-mart e Builder for Accel Platform をインストール済みであること。
- モジュール・プロジェクトの作成が完了していること。

設定ファイルで拡張<imart>タグを登録

以下の手順で拡張<IMART>タグを登録します。

ステップ1：initializerの作成

src/main/conf/products/initializer/initializer-[プロジェクト名].xmlを作成します。



コラム

initializer-[プロジェクト名].xml は、厳密にはプロジェクト名ではなくモジュールのIDから決定されます。
モジュールIDの”.”で分割してその末尾になります。
例として、モジュールIDが「org.example.foo」場合は「initializer-foo.xml」をという名前で initializer を定義してください。
intra-mart e Builder for Accel Platform で開発する場合はプロジェクト名と考えて問題ありません。

注意

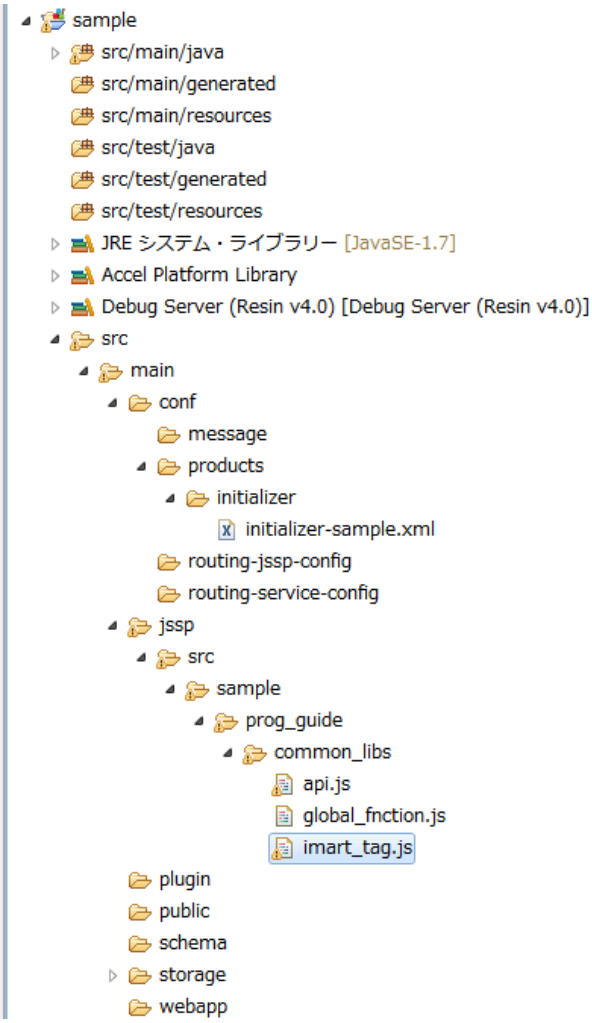
intra-mart e Builder for Accel Platform でプロジェクトを作成する際は 「im」 で始まる名前は利用できません。

ファイルに内容を加えます。記述方法は、<jssp-tag><tag-script>タグ定義js ファイルのパス#実行関数</tag-script></jssp-tag>です。

```
<?xml version="1.0" encoding="UTF-8"?>
<initializer-config
   xmlns="http://intra-mart.co.jp/system/secure/product/initializer/config/initializer-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://intra-mart.co.jp/system/secure/product/initializer/config/initializer-config ../../../schema/initializer-config.xsd ">

 <jssp-tag>
  <tag-script>sample/prog_guide/common_libs/imart_tag#imSample1</tag-script>
 </jssp-tag>

</initializer-config>
```

ステップ2：initializerに設定したjsファイルの作成

上記のxmlにて設定したjsファイルを作成します。
この例では、プロジェクトの src/main/jssp/src/sample/prog_guide/common_libs 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「 imart.js 」ファイルを作成し、 以下のように実装をします。



```
/**
 * 第 1 引数 attributes は、<IMART> タグにより指定された引数群になります。<br>
 * 第 2 引数 innerContent は、<IMART> と </IMART> に挟まれた内部ソース（実行オブジェクト形式）になります。<br>
 * innerContent は execute() メソッドを呼び出さないとオブジェクトの中身を参照できません。
 */
function imSample1(attributes,innerContent){
   var width = "500px";
   var height = "500px";
   // imart タグ内のオブジェクトの実行を行います。
   var inner = innerContent.execute();
   if(attributes.width){
      width = attributes.width;
   }
   if(attributes.height){
      height = attributes.height;
   }
   if(inner.length < 1){
      inner =  "拡張imartタグ";
   }
   return '<div class="imui-title" style="width:' + width + ';height:' + height + ';"><h1>' + inner + '</h1></div>';
}
```

ステップ3：モジュールのデプロイ

エクスポートしたユーザモジュールを含んだwarファイルを作成します。
そのwarファイルをデプロイして intra-mart Accel Platform を再起動します。

> **注意**
>
> initializer は warファイルに含まれるモジュール構成を元に読み込まれるので、一旦ユーザモジュールにしてwarファイルに組み込んだ形でないと動作しません。
> ユーザモジュールに関しては、『 intra-mart e Builder for Accel Platform アプリケーション開発ガイド / immファイルのエクスポート 』を参照してください。

ステップ4：登録した拡張<IMART>タグの呼び出し

登録した拡張<IMART>タグを使用するには、htmlでタグの呼び出しを記述します。

```
<imart type="imSample1" width="100px" height="50px"></imart>
```

出力されるHTMLは以下のようになります。

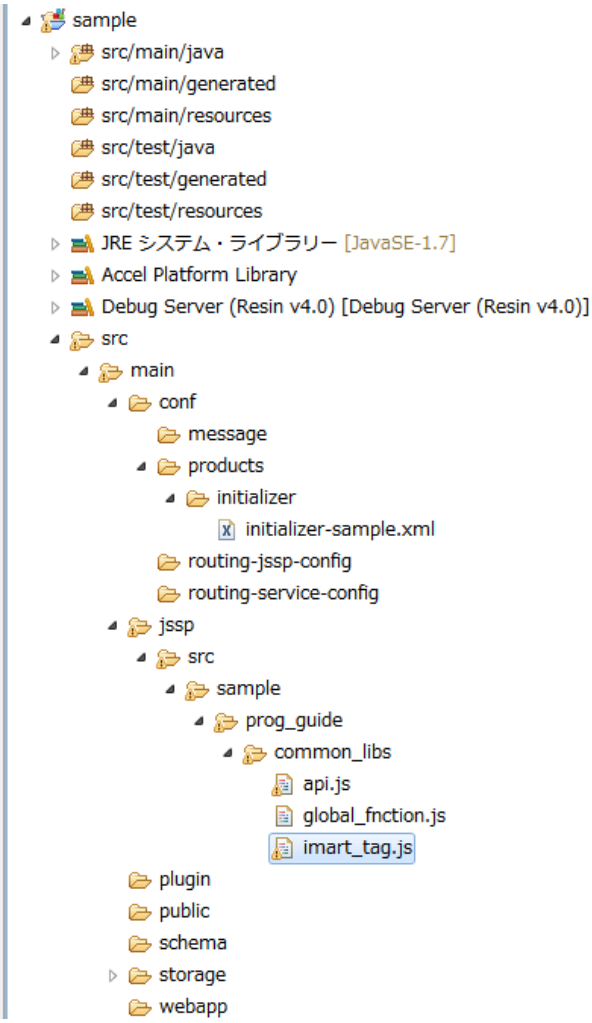
```
<div class="imui-title" style="width:100px;height:50px;"><h1>拡張imartタグ</h1></div>
```

Imart.defineType 関数で拡張<imart>タグを登録

以下の手順で拡張<IMART>タグを登録します。

ステップ1：任意の js ファイルにユーザ定義関数を格納する

この例では、プロジェクトの src/main/jssp/src/sample/prog_guide/common_libs 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「 imart_tag.js 」ファイルを作成し、 以下のように実装をします。




```
/**
 * <IMART> タグの実行関数を定義します。
 * 第 1 引数で指定した定義名称が <IMART> タグの type 属性での指定名称になります。
 * 第 2 引数には、<IMART> タグの type 属性により定義名称でコールされた際に実行する関数を指定します。
 * 関数内では、実行時に 2 つの引数を取ります。
 */
lmart.defineType("imSample2",imSample2);

function init(request){
}


/**
 * 第 1 引数 attributes は、<IMART> タグにより指定された引数群になります。<br>
 * 第 2 引数 innerContent は、<IMART> と </IMART> に挟まれた内部ソース（実行オブジェクト形式）になります。<br>
 * innerContent は execute() メソッドを呼び出さないとオブジェクトの中身を参照できません。
 */
function imSample2(attributes,innerContent){
   var width = "500px";
   var height = "500px";
   // imart タグ内のオブジェクトの実行を行います。
   var inner = innerContent.execute();
   if(attributes.width){
      width = attributes.width;
   }
   if(attributes.height){
      height = attributes.height;
   }
   if(inner.length < 1){
      inner =  "拡張imartタグ";
   }
   return '<div class="imui-title" style="width:' + width + ';height:' + height + ';"><h1>' + inner + '</h1></div>';
}
```

ステップ2：intra-mart Accel Platform の再起動

intra-mart Accel Platform を再起動します。

ステップ3：登録した拡張<IMART>タグの呼び出し

登録した拡張<IMART>タグを使用するには、htmlでタグの呼び出しを記述します。

```
<imart type="imSample2" width="100px" height="50px"></imart>
```

出力されるHTMLは以下のようになります。

```
<div class="imui-title" style="width:100px;height:50px;"><h1>拡張imartタグ</h1></div>
```

JavaClassとの連携

intra-mart Accel Platform で使用している サーバサイドJavaScript には、さまざまな優れた機能が実装されています。
しかし、スクリプト言語としての制限から通信機能の実現や、特殊なファイルアクセス等、システム構築上問題となる場合があります。
このようなシステム構築において問題となる部分を拡張する機能として、 JavaScript とJavaClassの連携機能を説明します。
本項では、これらの設定方法や利用方法などを紹介します。

標準Javaクラスとの連携方法

intra-mart Accel Platform は簡単に Java の標準 Class と連携を行うことができます。
intra-mart Accel Platform からは、決められた宣言方法を用いてクラスを定義することにより、 intra-mart Accel Platform のオブジェクトと同等に Javaクラスメソッドにアクセスすることができます。
本項では、標準Javaクラスとの連携方法を記載しています。

> **コラム**
>
> ここで説明する標準のJavaクラスとは、パッケージ名に java.、または、javax.で始まるクラスを指します。
> これらの標準クラスは、 Java 上から直接呼び出すことが可能です。

項目

- Java からの呼び出し方
- 例外の捕捉

Java からの呼び出し方

任意のファイルからプログラム内において、通常の Java のプログラミングの要領で目的のクラスを呼び出して利用してください。
インスタンスを生成する必要のあるクラスについては、new 演算子を用いる事でコンストラクタを呼び出す事ができます。
また、生成したインスタンスを JavaScript で宣言した変数に格納して利用する事もできます。

```
function init(request){
 // static method なのでそのまま使えます。
 java.lang.Thread.sleep(5000);
 // インスタンス生成する必要がある場合は new 演算子を用いる事でコンストラクタを呼び出します。
 var list = new java.util.ArrayList();
 // 生成したインスタンスを JavaScript で宣言した変数に格納して利用する事もできます。
 Debug.browse(list.isEmpty());
}
```

> **注意**
>
> JavaScript では import 句を利用する事ができません。
> Java のクラスを呼び出す場合は、必ずパッケージ名を省略せずに完全な形で目的のクラスを指定するようにしてください。

例外の捕捉

Javaクラスを呼び出した場合に発生する例外は、try catchにより捕捉することが可能です。
ただし、Errorクラスのサブクラスの捕捉は行えませんので注意してください。
捕捉可能な例外クラスは、Exceptionのサブクラスである必要があります。
捕捉した例外はJavaScriptの型に変換されています。
実際に発生したJavaの例外を操作したい場合には、捕捉した例外の__javaException__プロパティに格納されています。

```
    try {
      java.foo.bar.Baz.qux();
    } catch(e) {
      e.__javaException__.getLocalizedMessage();
      e.__javaException__.printStackTrace();
    }
```

自作Javaクラスとの連携方法

自作Javaクラス との連携を行う場合、Javaクラス ファイルの配置と自作Javaクラス側、サーバサイドJavaScript 側の双方に特別な記述方法が必要となります。
本項では、これらの設定方法や利用方法などを紹介します。

自作 Javaクラス との連携方法
- 前提条件
- 自作 Javaクラス との連携手順
  - ステップ1：Javaクラス ファイルの配置
  - ステップ2：サーバサイドJavaScript 側の記述方法
  - ステップ3：intra-mart Accel Platform の再起動
  - Packagesの利用に関しての注意事項

前提条件

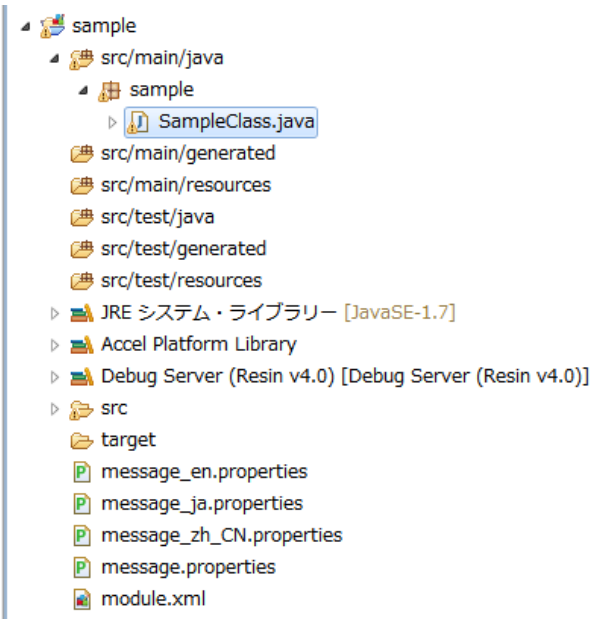
- intra-mart e Builder for Accel Platform をインストール済みであること。
- モジュール・プロジェクトの作成が完了していること。

自作 Javaクラス との連携手順

以下の手順で自作の Javaクラス との連携を行います。

ステップ1：Javaクラス ファイルの配置

自作Java Class ファイル、または jarファイルを、既定のフォルダに配置することで、 intra-mart Accel Platform 上で動作させることができます。
この例では、プロジェクトの src/main/java/ に 「sample」パッケージを作成し 「SampleClass.java」を作成します。



ファイルに内容を加えます。

```
//パッケージ定義
package sample;

//クラス定義
public class SampleClass {

   // 処理を追加します。

   public static String staticMethod() {
      return "staticMethod";
   }

   public boolean getTrue() {
      return true;
   }
}
```

ステップ2：サーバサイドJavaScript 側の記述方法

サーバサイドJavaScript を作成する場合は、自作Javaクラス を JavaScript オブジェクトとして生成する場合、Java の package名の前に必ず「Packages」句を記述します。

```
function init(request){
// インスタンス生成する必要がある場合は new 演算子を用いる事でコンストラクタを呼び出します。
 var instance = new Packages.sample.SampleClass();
// static method なのでそのまま使えます。
 var static = Packages.sample.SampleClass().staticMethod();
 Debug.browse(instance.getTrue(),static);
}
```

ステップ3：intra-mart Accel Platform の再起動

intra-mart Accel Platform を再起動します。

Packagesの利用に関しての注意事項

> **注意**
>
> Packages を利用することで処理速度が出ない場合があります。

Packages.foo.bar.baz等の形式で記述した場合は以下のフローで処理が行われます。

1. Packageの中にfooが存在するかを確認 (synchronized)
2. fooの中にbarが存在するのかを確認
3. barの中にbazが存在するのかを確認

これらの確認を行う際に、Javaクラスの呼び出し（Packagesを利用した呼び出し方法）では、全て確認処理部分が同期化されます。
従って、大量のリクエストがある場合、全てのリクエスト処理においてこの存在するかを確認する部分で同期化が走り処理速度が出ない場合があります。
以下のサンプルを参考にご利用の際には同期化を減らす等を検討してみてください。

- 事前に変数に代入する

  ```
  // 処理の度に Pacakges を宣言せずに事前に変数に代入しておく
  var baz = Pacakges.foo.bar.baz;
  baz.XXX();
  ```

- グローバル関数の初期化時に変数に代入する

```
// init.jsの中で確保しておく，グローバル関数の初期化時に事前に変数に代入しておく
var baz = Packages.foo.bar.baz;
function global_baz(){
    return baz();
}

/**
 * Initialize function for user-application.
 * @param nothing
 * @return void
 */
function init(){
 return;
}


/* END OF FILE */
```

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
- 移行手順
  - intra-mart Accel Platformの画面仕様に合わせる
    - 画面（HTMLファイル）の対応
    - diconファイルの設定
    - ロジック（JSファイル）の対応
  - iframe内で表示する
    - iframe設定
    - no-theme設定
    - baseタグ設定
    - diconファイルの設定
    - ロジック（JSファイル）の対応

前提

ここでは、 *ルーティング* & *認可* については基本的に触れません。
移行の場合、メニュー移行でどちらも設定されます。
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。
> - サポートするファイルのエンコードは、**UTF-8** のみです。移行前に既存のファイルエンコードを変更してください。
> - ファイルの格納先は、**pages/** から **WEB-INF/jssp** に変更になりました。
> - これまでのとおり、WEB-INF/jssp/src等に直接ファイルを配置すること（source-config.xmlの設定により変わる）で、サーバ再起動なしで、ソースの追加や変更して動作しますが、WARファイルに格納されていないファイルは、アンデプロイ時に削除されますので、 intra-mart e Builder for Accel Platform でのモジュールプロジェクトで運用を強く推奨致します。

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();
});
```

- <imart type=”imDesignCss”/>を削除
- headタグを置き換える

<head>タグを<imart type=”head”>タグに置き換えます。

- タイトルバータグを置き換える

imTitleBar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（JSファイル）の対応

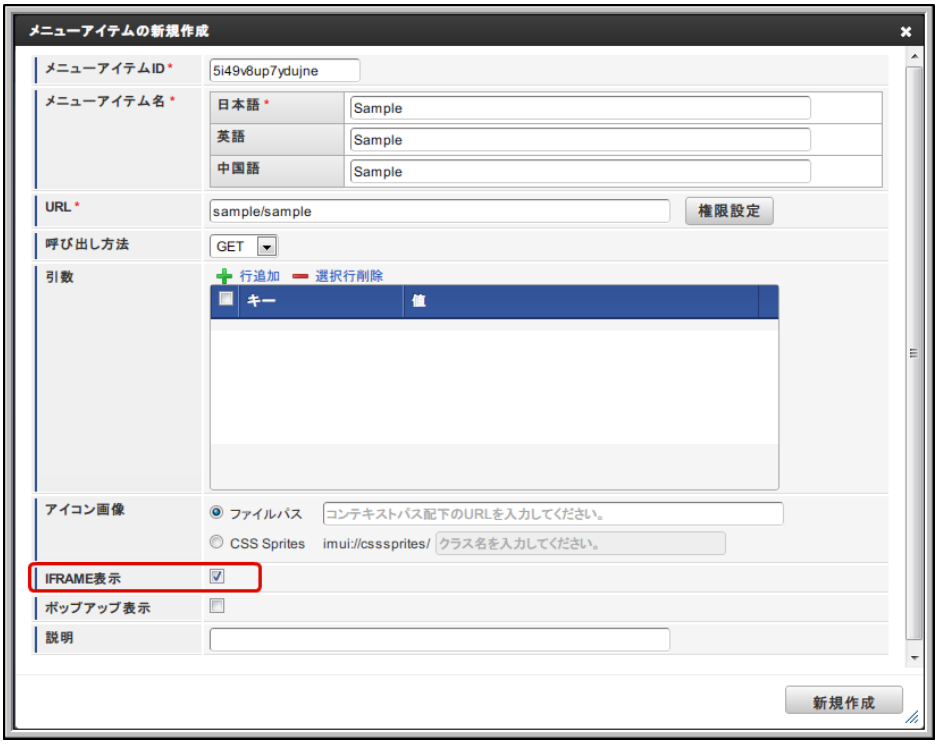
API対応を 互換対応表 を参照の上、対応してください。
intra-mart Accel Platformからファンクションコンテナ(JSファイル)上で実行するJavaのAPIについては、実行結果が以下の場合は、JavaScriptのオブジェクトに変換されるようになるため修正が必要です。

- java.lang.Boolean は Boolean オブジェクトに変換されます。
- java.lang.String は String オブジェクトに変換されます。
- java.lang.Number のサブクラスは Number オブジェクトに変換されます。

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。

baseタグ設定

URLの階層化により、これまでのような相対パスでは画像やCSS, CSJSが指定できません。
しかし、baseタグを設定すれば、既存の配置のままでパスの問題を解消できます。
テーマを適用している場合は、テーマ側で出力されますが、上記のno-themeの設定を行う場合は、個別に対応が必要です。

- 対応例
  - jsファイル

```
var base = "";
function init(request) {
   base = "<base href=\"" + Web.base() + "/\" target=\"_self\">";
}
```

  - htmlファイル

```
<imart type="string" value=base />
```

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（JSファイル）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。
intra-mart Accel Platformからファンクションコンテナ(JSファイル)上で実行するJavaのAPIについては、実行結果が以下の場合は、JavaScriptのオブジェクトに変換されるようになるため修正が必要です。

- java.lang.Boolean は Boolean オブジェクトに変換されます。
- java.lang.String は String オブジェクトに変換されます。
- java.lang.Number のサブクラスは Number オブジェクトに変換されます。

> **コラム**
> 詳細は、移行ガイドを参照してください。

項目

- スクリプト開発モデルの実行処理シーケンスについて
  - ブラウザからリクエストがあった場合の処理が流れ
    - action 属性
    - page 属性
    - エラーが発生した場合
    - session.js
  - サーバサイドJavaScript の種類
    - 初期起動用
    - html 連携用
    - 特殊プログラム

スクリプト開発モデルの実行処理シーケンスについて

ブラウザから intra-mart Accel Platform に対してアクセスした場合の動作仕様を説明します。
ブラウザから HTTP リクエストにより スクリプト開発モデルのプログラムが実行される場合、ページを作成するために必要な サーバサイド JavaScript と html を実行します。
ページプログラムは、サーバサイドJavaScript と html の２ファイルで１対となります。
２つのファイルは、ファイルラベル名で関連付けられます。それぞれのファイル(html および js)は、必ず同じファイルラベル名で作成してください。
拡張子は、htmlの場合は .html で、サーバサイドJavaScript の場合は .js となります。
（どちらの拡張子も小文字のみとなります。大文字で記述した場合、正しく動作しなくなります。）
js が必要ない場合には、jsの省略は可能ですが、html の省略はできません。

ブラウザからリクエストがあった場合の処理が流れ

ブラウザからリクエストがあった場合、以下のように処理が流れます。

1. ブラウザからのリクエスト受付
2. session.js 内 init() 関数実行
3. action 属性関数の実行
4. page 属性 js内 init() 関数の実行
5. page 属性 html の実行
6. session.js 内 close() 関数実行
7. ページ返却

action 属性

action 属性関数とは、intra-mart 連携用のリンク(<IMART type=”link”>)またはフォーム(<IMART type=”form”>)またはサブミット（<IMART type=”submit”>）からのリクエストであり、action 属性関数の実行指定がされていた場合にのみ処理されます（action 属性関数の実行指定がなかった場合には、この処理フェーズはスキップされます）。
action 属性関数には引数として、URL引数情報を持つオブジェクト(request)が渡されます。

コラム

action 属性関数指定が複数同時に行われていた場合、以下の順位付けにしたがって実行対象となる関数を決定します。

1. submit
2. form
3. link

action 属性関数の引数 request の詳細に関しては、[APIドキュメント - スクリプト開発モデルim-BizAPI - platform - Request]を参照してください。

> **注意**
>
> action, page属性や、imart type=”form”の利用は互換目的で残されており、利用の推奨はしておりません。
> 代わりの手段として、[スクリプト開発モデル プログラミングガイド - 応用（ intra-mart Accel Platform の機能を使いこなす） - ルーティング]を参照してください。

page 属性

page 属性とは、intra-mart 連携用のリンク(<IMART type=”link”>)またはフォーム(<IMART type=”form”>)またはサブミット(<IMART type=”submit”>)からのリクエストであり、page 属性関数の実行指定がされていた場合にのみ処理されます。
page 属性指定がない場合は、リクエストをしてきたページを再実行します。

> **コラム**
>
> page 属性 サーバサイドJavaScript 内 init() 関数には引数として、URL引数情報を持つオブジェクト(request)が渡されます。
> page 属性関数の引数 request の詳細に関しては、[APIドキュメント - スクリプト開発モデルim-BizAPI - platform - Request]を参照してください。

> **注意**
>
> action, page属性や、imart type=”form”の利用は互換目的で残されており、利用の推奨はしておりません。
> 代わりの手段として、[スクリプト開発モデル プログラミングガイド - 応用（ intra-mart Accel Platform の機能を使いこなす） - ルーティング]を参照してください。

エラーが発生した場合

各スクリプトの実行フェーズにおいて、何らかの要因でエラーが発生した場合、即座にスクリプト実行を中止して、ブラウザに対してはエラー画面を送信します。
エラーが発生した場合は、エラーの発生した実行フェーズおよび『session.js 内 close() 関数実行』および『ページ返却』を除く残りの実行フェーズはスキップされます。

session.js

session.js 内 close() 関数実行フェーズは、ページ処理に関する終了処理を行うことを目的とします。

> **注意**
>
> session.js 内 close() 関数実行フェーズにおいて、Debug.browse() や forward() などの
> スクリプト実行を強制的に中断して処理を遷移させるAPIを実行した場合、正しく処理されない場合があります。

サーバサイドJavaScript の種類

サーバサイドJavaScript は、用途により3種に大別されます。

初期起動用

初期起動用 サーバサイドJavaScript は、サーバによりプログラムファイルがロードされると、ファイル内に記述された init() 関数が実行されます。
共通関数登録などの初期化処理は、init() 関数内に記述するようにしてください。
サーバの初期起動時には、ソースのルートディレクトリ（標準では、jssp/src ）にある init.js 内に定義された init() 関数が実行されます。

- 実行シーケンス
    1. init.js のロード
    2. init.js 内の init() 関数の実行

html 連携用

html と連動する サーバサイドJavaScript は、引数を伴って呼び出されます。
HTTP リクエストにより実行された場合、URL引数を持つ request オブジェクトを引数として init() 関数が実行さ れます。
また、リンクやフォームの action 属性により任意の関数実行が指定されている場合、URL引数を持つ requestオブジェクトを引数として指定

の関数が実行されます。

HTTP リクエスト以外の方法（API forward() 等）により実行された場合、サーバサイドJavaScript 内の init() 関数 は、その方法によって定義された引数を伴って実行されます。（詳細に関しては、各々のAPI仕様を参照してください）

html の実行が完了したあと、サーバサイドJavaScript 内に close() 関数が定義されていた場合、URL引数を持つ request オブジェクトを引数として close() 関数が実行されます。

- 実行シーケンス

    1. action 属性指定関数の実行
    2. init() 関数の実行
    3. html の実行
    4. close() 関数の実行

特殊プログラム

session.js は、HTTP リクエストを受け付けた時に実行されるプログラムです。
HTTP リクエストを受け付けると、session.js 内に定義された init() 関数が実行されます。
また、 html の作成を完了した後、HTTP レスポンスを返す前に close() 関数が実行されます。
session.js ファイルを変更した場合は、intra-mart Accel Platform を再起動するまでシステムには反映されません。