目次

改訂情報

変更年月日	変更内容
2013-04-01	初版
2013-07-01	第2版 下記より、サポートしていない種類のポートレット(JSR168/286)に関する情報を削除しました。 ■ 「*概要*」 ■ 「*ポートレットの開発（スクリプト開発編）*」 ■ 「*ポートレットの開発（JavaEE 開発編）*」
2013-10-01	第3版 下記を追加・変更しました。 ■ 「*ポートレットの開発（非同期ポートレット編）*」に *画面開発（TERASOLUNA Server Framework for Java (5.x) 開発)* を追記しました。 ■ 「*ポートレットのアクセス制御*」を追記しました。 ■ 「*ポートレットの開発（Spring 編）*」を追記しました。
2014-01-01	第4版 下記を追加・変更しました。 ■ 「*ポートレットの開発（スクリプト開発編）*」にポートレットモードの設定について追記しました。ポートレットモードの初期値について修正しました。 ■ 「*ポートレットの開発（JavaEE 開発編）*」にポートレットモードの設定について追記しました。ポートレットモードの初期値について修正しました。 ■ 「*ポートレットの開発（非同期ポートレット編）*」にポートレットモードとアクセス制御について追記しました。 ■ 「*ポートレットの開発（Spring 編）*」にポートレットモードの設定について追記しました。Ajaxを使う場合について追記しました。
2014-04-01	第5版 下記を変更しました。 ■ 「*「重要なお知らせ」ポートレットについて*」（Javaee開発）の内容を「重要なお知らせ」ポートレットに統一しました。 ■ 「*「重要なお知らせ」ポートレットについて*」（スクリプト開発）の内容を「重要なお知らせ」ポートレットに統一しました。 ■ 「*画面を作成する上での注意事項*」に注意点を追記しました。 ■ パブリックストレージの記述が変わりました。(%PUBLIC_STORAGE_PATH%)
2014-08-01	第6版 下記を変更しました。 ■ ポータルのUI改善にともない、イメージを変更しました。 ■ ポートレット定義の変更にともない、説明を変更しました。

変更年月日	変更内容
2015-04-01	第7版 下記を変更しました。 ▪ 「*画面開発（TERASOLUNA Server Framework for Java (5.x) 開発)*」のTERASOLUNA Global FrameworkをTERASOLUNA Server Framework for Java (5.x)に変更しました。 ▪ 「*概要*」のSpring Frameworkのリンクを 4.1.4.RELEASE に変更しました。 ▪ 「*Ajax （serveResourceサイクル）*」のJsonModelAndViewResolverにMappingJackson2JsonViewを利用することを記述しました。 ▪ 「*Client Side Java Script API*」を追加しました。
2015-08-01	第8版 下記を変更しました。 ▪ 「*はじめに*」にサンプルプログラムの配置場所を追加しました。 ▪ 誤植を修正しました。
2016-12-01	第9版 下記を変更しました。 ▪ 「*ポートレットの開発（Spring 編）*」のSpring Frameworkのリンクを 4.2.7.RELEASE に変更しました。
2018-08-01	第10版 下記を変更しました。 ▪ 「*ポートレットの開発（Spring 編）*」のSpring Frameworkのリンクを 4.3.5.RELEASE に変更しました。 ▪ 「*java.util.Date型プロパティのJSONでのフォーマットについて*」にMappingJackson2JsonView を利用する場合の CVE-2018-11040 対応方法を記述しました。
2019-08-01	第11版 下記を変更しました。 ▪ 「*ポートレットの開発（Spring 編）*」に 2019 Summer(Waltz) 以降では利用できなくなる注意書きを記述しました。

はじめに

目次

目的

このドキュメントは、intra-mart Accel Platform のポータル画面に表示して利用することが可能なポートレットモジュールを作成する方法について説明します。

本書では、ポートレットプログラミングの概要と、ポートレットを作成する上での注意事項を主に取り扱います。 そのため、本書で記述してあるサンプルコードは一部を抜粋したコードとなります。

実際に動作可能なポートレットアプリケーションを作成するためには、それぞれの開発モデルの言語仕様とポートレットの特性を理解して実装を行う必要があります。

サンプルプログラムについて

intra-mart Accel Platform には、ポートレットのサンプルプログラムが付属しています。(2015 Summerより提供)

システム管理者で「サンプルデータセットアップ」を行うことにより各サンプルポートレットを含むポータルがセットアップされます。

サンプルプログラムソースは こちら からダウンロードできます。

内容	配置場所
スクリプト開発ポートレットサンプル	WEB-INF/jssp/src/sample/portal/event
JavaEE Framewok開発ポートレットサンプル	sample/portal/event
JavaEE Framewok開発ポートレット用設定ファイルサンプル	WEB-INF/classes/sample/portal/service-config-portal_sample.xml
アクションハンドラクラス	WEB-INF/classes/sample/portal/event/handler

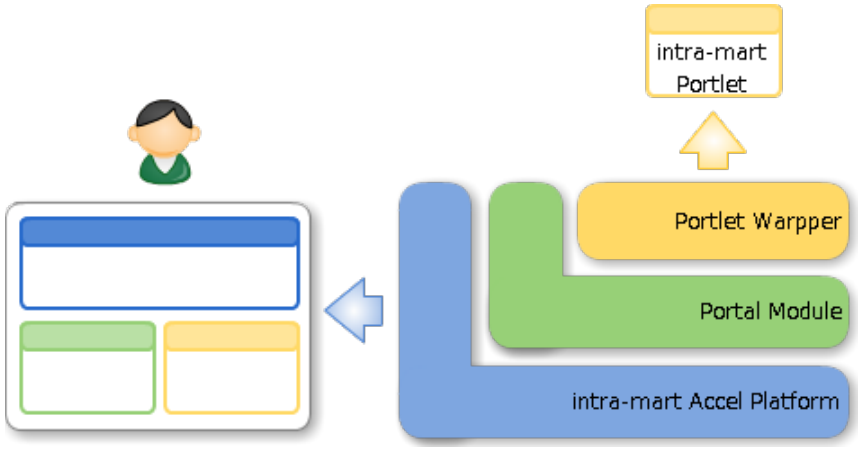
ポートレットアプリケーションを作成する場合は、本ドキュメントと合わせて参照するようにしてください。

概要

目次

ポータルモジュールについて

ポータルモジュールは、ポートレットコンテナ上で動作します。
intra-mart アプリケーションをポートレットとして利用するためのラッパーが標準で準備されているため、旧バージョンまでで利用していたポートレットもそのまま利用することが可能です。



＜intra-mart ポータルモジュール構成＞

ポートレットの種類

ポータルモジュールで利用可能なポートレットは、開発モデルに５つに分類されます。

ポートレットの種類	開発モデル
intra-mart アプリケーションとして作成されたポートレット	スクリプト開発モデル
	JavaEE 開発モデル
	JSP/サーブレット
	SA Struts
	Spring

次章では、これらのポートレット種別ごとにポートレットを開発する方法を説明します。

JSP/Servletを利用したポートレット開発

上記以外に、intra-mart のフレームワークを利用せず、JSP や Servlet で開発した Web ページをポートレットとして登録することも可能です。
これらを作成する場合は、標準的な Web アプリケーションの開発方法で作成することができます。

ただし、Struts のような Web アプリケーションフレームワークは、通常の Web ページを作成するためのフレームワークであるため、ポートレットを作成するためには適しておりません。
ほとんどの場合、そのままポートレットとして利用することはできませんので、ポートレット用に修正する必要があります。
具体的には以下の点に注意して修正してください。

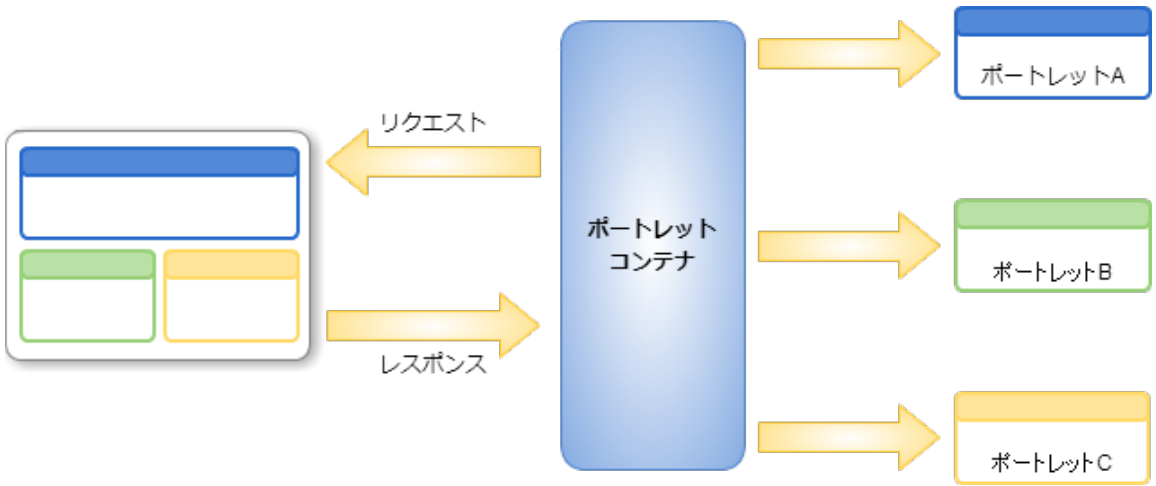
- ポートレットはポータル画面からINCLUDEして呼び出されるため、ポートレット処理中にFORWARD処理することができません。
  Struts では、特に処理を記述しない限り自動的に FORWARD されるため、必ず Action クラスの処理内で RequestDispatcher#include()を呼び出すようにしてください。
- その他、struts-config.xml 内での FORWARD 処理も利用できませんので注意してください。
- また、ServletFilter のターゲットは FORWARD のみですので、もし Filter を利用する必要がある場合、INCLUDE をターゲットに追加するようにしてください。

ポートレットのライフサイクル

ポートレットは、ポータル画面および他のポートレットと協調動作するために、以下のライフサイクルメソッドが定義されています。

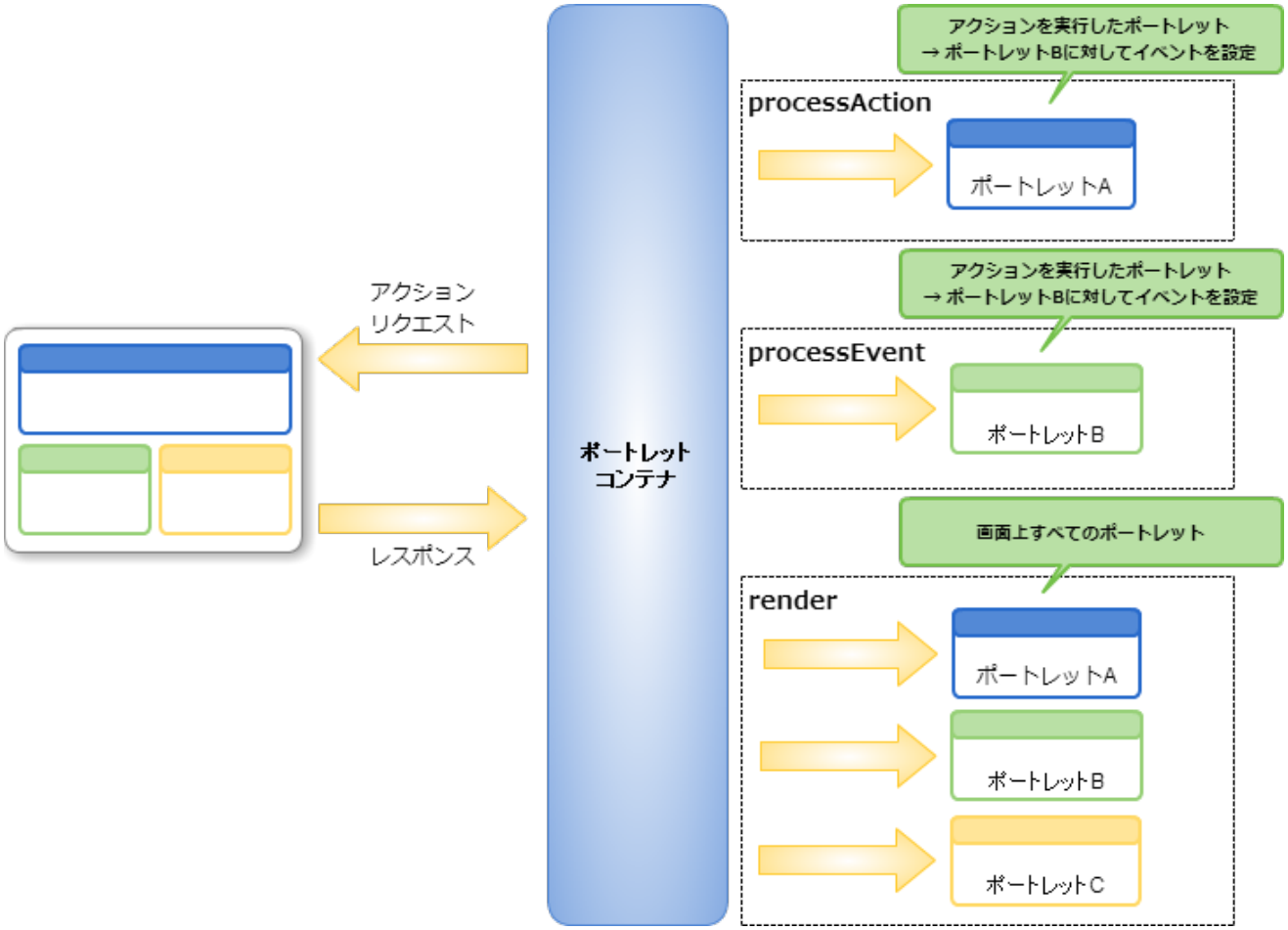
processAction	ポートレットに対する処理の実行メソッド
processEvent	processAction から派生したイベントの受信処理メソッド
render	ポートレット画面の描画処理

通常、ポータル画面に対するリクエストはポートレットコンテナによって受け付けられて、ポートレットコンテナは各ポートレットの render メソッドを呼び出します。



＜ポータル render 処理フロー＞

ポートレットでは、任意のボタンやコントロールにより、アクション URL へフォームデータをサブミットすることで、processAction を呼び出すことができます。
また、processAction内で Event を発生させることで、該当のポートレットの processEvent を呼び出すことができます。



＜ポータル processAction / processEvent 処理フロー＞

processEvent を呼び出すためには、あらかじめポートレットコンテナに対して設定が必要です。
設定については、それぞれの開発モデルの章を参照してください。

ポートレットモード

ポートレットには以下のモードが存在し、ユーザが切り替えて表示内容を制御することが可能です。

VIEW モード	
EDIT モード	
CONFIG モード	設定ポータル用のモードで通常のポータル画面では表示されません。

ただし、モードをサポートするためには、あらかじめポートレットコンテナに利用するモードを設定しておく必要があります。
設定の仕方については、それぞれの開発モデルの章を参照してください。

ウィンドウステータス

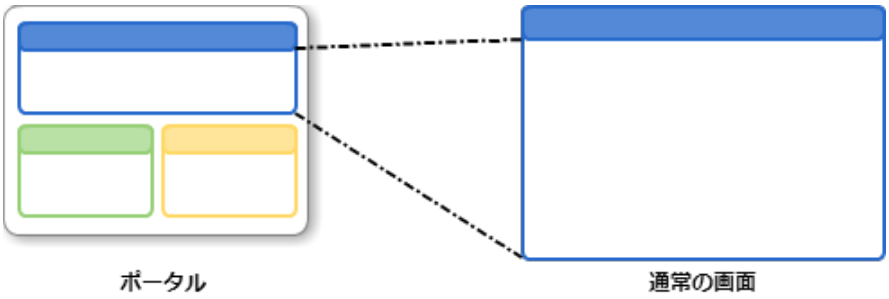
ポートレットには以下のウィンドウステータスが存在し、ユーザが切り替えて表示内容を制御することが可能です。

NORMAL
MAXIMIZE

MINIMIZE	コンテンツが表示されないため、ポートレットは呼び出されません。

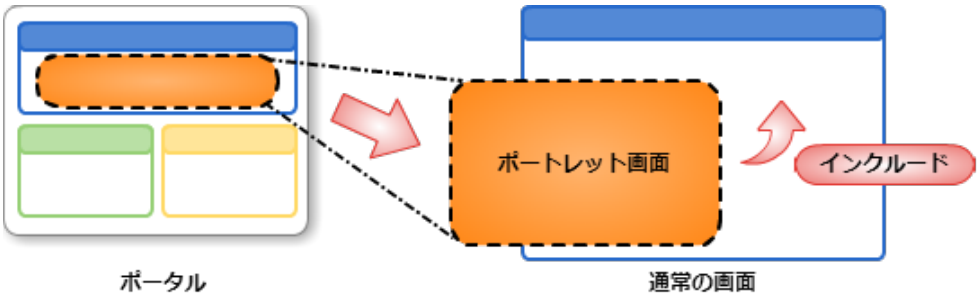
ポートレット用画面

ポートレットの画面はポータル画面の一部として表示されるため、通常の画面と同様に作成することはできません。
詳細は 「*画面を作成する上での注意事項*」を参照してください。



＜ポータル画面と通常画面との比較＞

そのため、通常の画面とポートレット画面で共通の画面を利用したい場合でも、別々に画面を作成する必要があるます。
ただし、画面表示や機能は共通にできるため、以下のようにポートレット画面をインクルードするような画面を作成することで、通常の画面用に画面を作り直す必要はなくなります。



＜通常の画面とポートレット画面を共用する＞

実装方法については、それぞれの開発モデルごとに例を示します。
「*ポートレットの開発（JavaEE 開発編）*」または 「*ポートレットの開発（JSP/Servlet 編）*」を参照してください。

ポートレットの開発（スクリプト開発編）

目次

概要

この章では、スクリプト開発モデルを利用してポートレットの作成を行うための手順や注意事項を説明します。
ポートレットを作成するためには、「*ポートレットのライフサイクル*」の内容を理解し、どのライフサイクルを利用するのかを決定する必要があります。

1. **画面開発 （render サイクル）**

   ポートレットに表示する画面を作成します。モードやウィンドウステータスを判定して切り替えることも可能です。
   表示モード用の画面は必ず必要です。

2. **アクション処理 （processAction サイクル）**

   ポートレットからサブミットされたデータの処理を作成します。作成されたファンクションコンテナは、Actionハンドラと呼ばれ、ポートレット編集画面で登録することによって利用できます。
   また、Event を設定することによって、別のポートレットと連携することも可能です。

3. **イベント処理 （processEvent サイクル）**

他のポートレットから発生したイベントの受信処理を作成します。作成されたファンクションコンテナは、Event ハンドラと呼ばれ、ポートレット編集画面で登録することによって利用できます。
Action ハンドラで Event を設定された場合のみ実行されます。

ポートレットのライフサイクルごとに、以下のように作成するファイルが異なります。

render	必須	スクリプト開発モデルの画面。プレゼンテーションページとファンクションコンテナ。
processAction	任意	handleAction ファンクションを実装したファンクションコンテナ。
processEvent	任意	handleEvent ファンクションを実装したファンクションコンテナ。

全てのライフサイクル用プログラムを作成し、組み合わせることで 1 つのポートレットとして作成することも可能です。
ライフサイクルごとに別々に開発を行って、複数のポートレット間で組み合わせて利用できるように作成することも可能です。
次項より、ポートレットで利用可能な API と、各ライフサイクルでの実装方法について簡単に説明していきます。

ポートレットAPI

単純な画面表示のみのポートレットでは、ポートレットAPIを利用しなくても作成可能ですが、ポートレットの機能を最大限に利用するためには、ポートレット API を利用する必要があります。
特に Action 処理や Event 処理を行うためには、ポートレット API を利用しなくては実行できません。
このドキュメントでは、基本的な関数のみに絞って説明します。全ての API については「ポータル API リスト」を参照してください。

PortalManager

ポートレットの機能を利用するための、さまざまな関数を提供します。
それぞれの関数については、次項以降で実際に利用する際に説明します。

ポートレットモード

スクリプト開発モデルで作成されたポートレットのポートレットタイプは、「スクリプト開発モデルポートレット」です。
ポートレットモードの利用可否はポートレットタイプごとに設定され、「スクリプト開発モデルポートレット」の初期状態では、表示モードと編集モードが利用できます。

ポートレットモードの設定

ポートレット定義を変更し、ポートレットを初期化することでポートレットモードの設定を変更できます。

- ポートレット定義の変更
  ポートレットモードの設定は以下のファイルで管理されています。intra-mart Accel Platform を停止して設定後、再起動を行います。
  設定ファイル

  %CONTEXT_PATH%/WEB-INF/plugin/jp.co.intra_mart.portal.portlets.jssp_8.0.0/plugin.xml

  設定箇所

```
<?xml version="1.0" encoding="utf-8"?>
<plugin>
  <extension point="jp.co.intra_mart.portal.portlets">
    <portlet
      id="jp.co.intra_mart.portal.portlets.jssp"
      name="PresentationPagePortlet"
      display-name="%display-name"
      description="%description"
      version="8.0.0"
      rank="1">
      <portlet-
class>jp.co.intra_mart.foundation.portal.portlets.model.PresentationPagePortlet</portlet-class>
      <expiration-cache>0</expiration-cache>
      <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>VIEW</portlet-mode>
        <portlet-mode>EDIT</portlet-mode>
        <!-- ここにモードを追加 -->
      </supports>
      ...
    </portlet>
  </extension>
</plugin>
```

ポートレットモードの取得

API を利用して、以下のように現在のポートレットモードを取得できます。
ポートレットモードは小文字で取得されますので、注意してください。

(ex.) view, edit, help

```
var renderRequest = PortalManager. getRenderRequest();
var portletMode = renderRequest. getPortletMode();
```

ウィンドウステータス

ウィンドウステータスの取得

API を利用して、以下のように現在のポートレットモードを取得できます。
ウィンドウステータスは小文字で取得されますので、注意してください。

また、最小化時にはポートレットが呼び出されませんので、「minimized」が取得されることはありません。

(ex.) normal, maximized, minimized

```
var renderRequest = PortalManager. getRenderRequest();
var windowState = renderRequest. getWindowState();
```

画面開発 （renderサイクル）

render サイクルでは、通常の Web ページと同様に画面表示が行われます。
スクリプト開発モデルの画面は、おおむね通常の intra-mart の画面開発と変わりません。
ポートレット新規登録／編集画面（「ポータル グループ管理者操作ガイド」参照）で設定されたページ引数は、init 関

数の request オブジェクトより取得できます。
また、次項アクション処理、イベント処理によって設定されたページ引数も、同様にinit関数のrequestオブジェクトより取得できます。

RenderRequest

ポートレットで利用する request オブジェクトは、通常の Web ページのリクエストは異なり、ポートレット専用の request オブジェクトを利用します。
このリクエストを利用することで、ポートレットの情報を取得できます。

- RenderRequest の取得

```
var renderRequest = PortalManager.getRenderRequest();
```

ただし、通常の Webページと共用できるように、ページ引数は、init 関数の request オブジェクトを利用して取得することも可能です。

- ページ引数の取得

```
function init(request) {
 var renderRequest = PortalManager.getRenderRequest();
 // value1 と value2 は同じ値
 var value1 = request.param1;
 var value2 = renderRequest.param1;
}
```

RenderRequest から、現在のポートレットモード、ウィンドウステータスなどが取得できます。

RenderRequestのスコープ

RenderRequest は通常の Web アプリケーションのリクエストとスコープが異なります。
全てのポートレットで独立したリクエストを保持しており、ポータル画面を表示する際のリクエストパラメータを引き継ぎません。
ただし以下のパラメータは、ポートレットコンテナにより、自動的に設定されます。

portal_cd	ポートレットが配置されているポータル画面のキー。
portalKind	ポートレットが配置されているポータル画面のポータル種別。以下の値が取得される。 user　ユーザポータル group　グループポータル

ポートレットにリクエストパラメータを設定するためには、アクション処理またはイベント処理を実行する必要があります。
また、一度設定したリクエストパラメータは、セッションが持続する間、再度アクション処理またはイベント処理が実行されるまで有効です。

ActionURL, RenderURL

スクリプト開発モデルで作成されたポートレットからサブミット処理を行い、 処理終了後に再びポータル画面を表示する機能を作成する場合には、ポートレットコンテナと通信するため、以下の URL に対してサブミットする必要があります。

ActionURL	Action 機能を呼び出す URL
RenderURL	ポータル画面を再表示するための URL

これらは、PortalManager を利用して取得できます。

- ActionURL, RenderURL の取得

```
var actionURL = PortalManager.createActionURL();
var renderURL = PortalManager.createRenderURL();
```

以下に、スクリプト開発モデルで開発されたポートレットでサブミットされた後に、アクション処理を呼び出すサンプルを示します。

1. **ファンクションコンテナ**

```
// ActionURL
var actionURL = "";
function init(request) {
 // ActionURL を取得する
 actionURL = PortalManager.createActionURL();
}
```

2. **プレゼンテーションページ**

```
<!-- ActionURL を指定 -->
<form name="sampleActionForm" action="<IMART type="string" value=actionURL></IMART>"
method="POST">
 <!-- アクション処理で使用する値 -->
 <INPUT type="text" name="param1" value="">
 <INPUT type="submit" value="実行">
</form>
```

- ポータル画面の表示には、「portal/desktop」というパスが割り振られています。
  そのため、このサービスID-URLにサブミットすることでもポータル画面の再表示が可能ですが、その場合ポータル情報を引き継ぐことができませんので、上記の方法を利用するようにしてください。
- アクション処理を呼び出すためには、ActionURL にサブミットする以外にありません。
  ポートレット管理画面で Action ハンドラを登録し、プレゼンテーションページで ActionURL にサブミットするようにしてください。
  Action ハンドラが未定義の場合、単純にポータル画面を再表示します。
- ActionURL はポータル情報を含むため、データ量が多くなります。メソッドは POST を指定するようにしてください。

通常の画面とポートレット画面を共用するには

共用するコードと通常の画面のみ必要なコードを別々のプレゼンテーションページとして作成し、通常の画面用のプレゼンテーションページでは、include タグを利用して共用ページを読み込みます。
ただし、include タグではリクエストパラメータは引き継がれませんので、別途変数を定義して個別に引き継ぐ必要があります。
また、このような利用を行う場合はポートレットの各 API は利用できませんので注意してください。

【ポートレット画面と共用する通常画面のプレゼンテーションページ】

```
<html>
<head>

 <link src="sample.css" type="text/css">

</head>
<body>
 <div>
     ヘッダ表示情報
 </div>
 <!-- 共用コードをインクルードする -->
 <!-- param1, param2 引き継ぐリクエストパラメータ -->
 <IMART type="include" page="sample/sample-portlet"
param1=value1
param2=value2
/>
 <div>
     フッタ表示情報
 </div>
</body>
</html>
```

アクション処理 （processActionサイクル）

processAction サイクルでは画面表示がありませんので、ファンクションコンテナのみ作成します。
さらにポートレット管理画面で作成した Action ハンドラを登録し、ActionURL へサブミットすることでアクション処理が実行されます。
アクション処理では、RenderParameter の設定、PortletPreference の設定、イベントの設定などの更新処理を行います。

Actionハンドラの作成

以下にイベントを設定する Action ハンドラの例を示します。

1. **ファンクションコンテナを作成し、以下の関数を定義します。**

   （このファンクションコンテナを Action ハンドラと呼びます。）

   (ex.) sample/portal/sample_action.js

```
// 「handleAction」という名前で関数を定義する。
function handleAction(request, response) {
 // リクエスト引数を取得
 var value = request.param1;
 // イベントを設定 （ キー： "event1"、値： value ）
 response.setEvent("im_event1", value);
}
```

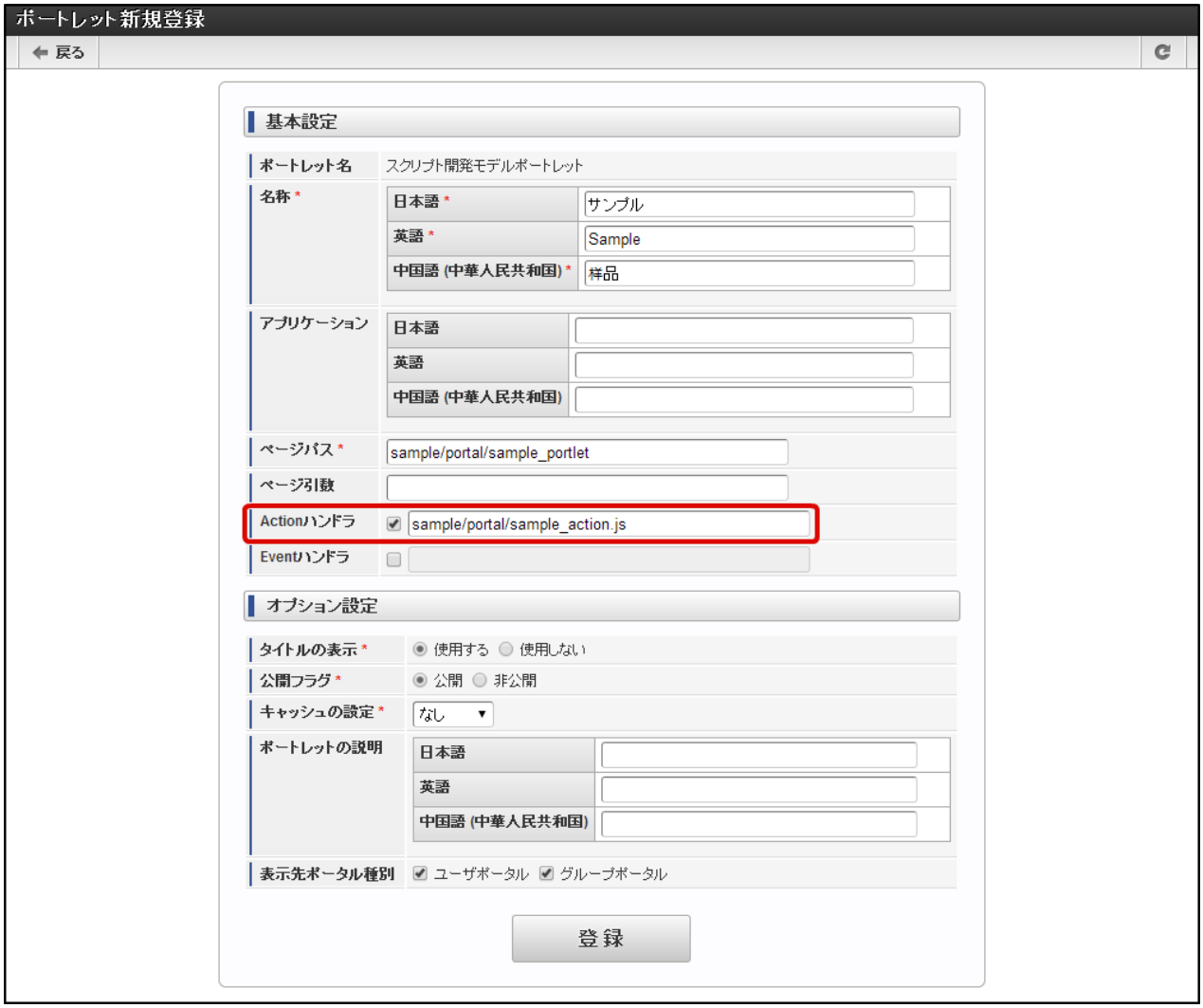
   - Action ハンドラは、render で利用するファンクションコンテナに記述しても構いません。
     また、Java で作成された Action ハンドラクラスを利用することもできます。
   - 以下のような値を return することでもイベントを発生させることができます。

Object	プロパティのキーと値をペアとするイベント
その他	文字列として判断し、キーと値が同じイベント

ただし、IM 用ポートレットに対するイベントを発生させるためには、**'im_'** で始まるキーを指定する必要があります。

- response.setEvent()で設定する値については、「ポータルAPI リスト」の ActionResponse を参照してください。

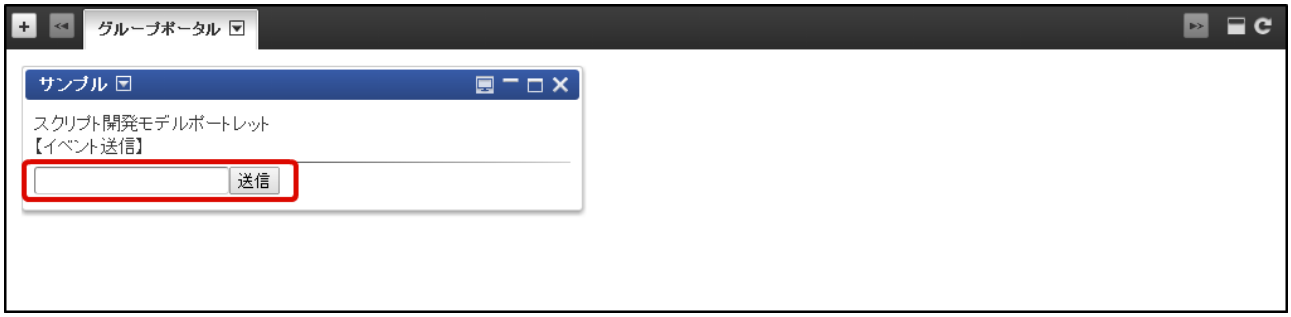
2. **Action ハンドラをポートレットに登録します。**



<ポートレット新規登録画面>

3. **「*画面開発 （renderサイクル）*」で作成した画面をポータル画面に配置し、ポートレットからアクション処理を呼び出します。**

Actionハンドラ実行後に、画面を再表示します。



<ポータル画面 - アクション処理>

ActionRequest, ActionResponse

Action ハンドラの引数、request, response は、それぞれ ActionRequest オブジェクト、ActionResponse オブジェクトです。
これらのオブジェクトを利用することにより、リクエスト引数を取得したり、イベントを設定したりできます。
詳細は、「ポータル API リスト」を参照してください。

RenderParameterの設定

画面表示時に取得可能なリクエストパラメータ(RenderParameter)は、アクション処理またはイベント処理の中で設定を行います。
アクション処理呼び出し時のリクエストパラメータは、ActionRequestに格納されていますが、 **renderサイクルには引き継がれないため** 、必要な場合は明示的に設定する必要があります。
また、アクション処理の呼び出し時には過去に設定された RenderParameter は全てクリアされています。

(ex.) processAction サイクルでの RenderParameter の設定

```
// RenderParameter を設定する場合。
response.setRenderParameter("param1", "value1");
// RenderParameter を削除する場合。
response.setRenderParameter("param2", null);
```

(ex.) render サイクルでの RenderParameter の利用

```
String value = request.param1;
```

イベントの設定

イベントを設定することにより、アクション処理を受け付けたポートレット以外の複数のポートレットにイベントの発生を通知して処理を引き渡すことが可能です。

(ex.) processAction サイクルでのイベントの設定

```
response.setEvent(eventId, eventValue);
```

利用可能なActionハンドラ

intra-mart Accel Platform では、以下の Action ハンドラが標準で用意されています。
これらは、Java 言語で実装されたクラスですが、スクリプト開発でも利用できます。

jp.co.intra_mart.foundation.portal.common.handler.DefaultPortletHandler

ポートレット新規登録／編集画面で、Action ハンドラチェックボックスにチェックを行うと、初期値としてこのActionハンドラが設定されます。
この Action ハンドラは、リクエストパラメータから値を取得して、そのキーと値を用いてイベントを発生させます。
単純に画面からイベントを発生させたいだけの場合、Action ハンドラを作成せずに、このハンドラを利用できます。
イベントを発生させるためには、以下のリクエストパラメータを設定します。

- “portlet.event.” + 任意の文字列
  これにより、以下のイベントが作成されます。

プロパティ名	内容
id	リクエストパラメータキーの任意の文字列

プロパティ名	内容
value	リクエストパラメータの値

リクエストパラメータに複数設定することで、複数のイベントを発生させることも可能です。

jp.co.intra_mart.foundation.portal.common.handler.SetRenderParameterHandler

ポータル画面表示時のリクエストパラメータは、ポートレットとスコープが異なるため、ポートレット内の処理からは取得できません。
この Action ハンドラを設定することで、ポータル画面表示時のリクエストパラメータをポートレットで利用できます。

イベント処理 （processEventサイクル）

processEvent サイクルでは画面表示がありませんので、ファンクションコンテナのみ作成します。
さらにポートレット管理画面で作成した Event ハンドラを登録し、任意のポートレットのアクション処理でイベントが設定されることにより、イベント処理が実行されます。
イベント処理では、RenderParameter の設定、PortletPreference の設定、イベントの設定などの更新処理を行います。

Eventハンドラの作成

以下に RenderParameter を設定する Event ハンドラの例を示します。

1. **ファンクションコンテナを作成し、以下の関数を定義します。**

   （このファンクションコンテナを Event ハンドラと呼びます。）

   (ex.) sample/portal/sample_event.js

   ```
   // 「handleEvent」という名前で関数を定義する。
   function handleEvent(event, request, response) {
    // イベント情報を取得
    var eventId = event.id;
    var value = event.value;
    // 画面表示に利用するリクエストパラメータを設定
    response.setRenderParameter(eventId, value);
   }
   ```

   - Event ハンドラは、render で利用するファンクションコンテナに記述しても構いません。
     また、Java で作成された Event ハンドラクラスを利用することもできます。

2. Event ハンドラをポートレットに登録します。

<ポートレット新規登録画面>

3. **「*画面開発（renderサイクル）*」で作成した画面と「*アクション処理（processActionサイクル）*」で作成したイベントを呼び出すことができる**
   ポートレットをポータル画面に配置し、イベントを設定するポートレットからアクション処理を呼び出します。

   アクション処理からEventハンドラが呼ばれ、その後に画面を再表示します。

<ポータル画面 - イベント処理>

EventRequest, EventResponse

Event ハンドラの引数、request, response は、それぞれ EventRequest オブジェクト、EventResponse オブジェクトです。
これらのオブジェクトを利用することにより、リクエスト引数を取得したり、さらにイベントを設定したりできます。

詳細は、「ポータル API リスト」を参照してください。

Eventオブジェクト

Event ハンドラの第 1 引数は、アクション処理で設定されたイベント情報オブジェクトです。以下の情報を格納しています。

プロパティ名	型	値
id	String	アクション処理で設定されたイベント ID 省略された場合、文字列「ImEvent」が設定される。
value	String	アクション処理で設定されたイベントの値 オブジェクトを設定した場合も文字列として取得される。
source	String	Event を設定したポートレットのポートレットコード

RenderParameterの設定

画面表示時に取得可能なリクエストパラメータ(RenderParameter)は、アクション処理またはイベント処理の中で設定を行います。
イベント処理呼び出し時に設定済みであった RenderParameter は、EventRequest に格納されており、特に処理を行わない限り、 **render サイクルに自動的に引き継がれます。** 引継ぎが不要な場合は明示的に削除する必要があります。

(ex.) processEvent サイクルでの RenderParameter の設定

```
// RenderParameter を設定する場合。
response.setRenderParameter("param1", "value1");
// RenderParameter を削除する場合。
response.setRenderParameter("param2", null);
```

(ex.) render サイクルでの RenderParameter の利用

```
request.getParameter(eventId);
```

イベントの設定

イベントを設定することにより、イベント処理を受け付けたポートレット以外の複数のポートレットにイベントの発生を通知して処理を引き渡すことが可能です。
ただし、イベント処理でさらにイベント処理を行うとパフォーマンスの低下をまねき、またイベントループの発生する可能性もありますので、十分注意して実装してください。

(ex.) processEvent サイクルでのイベントの設定

```
response.setEvent(eventId, eventValue);
```

「重要なお知らせ」ポートレットについて

ポータルモジュールには、汎用的な新着情報を表示する「重要なお知らせ」ポートレットが用意されています。
この「重要なお知らせ」ポートレットでは、新着情報として以下の項目を表示します。

- 新着情報の種類を示すためのカテゴリ

- 新着情報の内容（概要）
- 配信日付
- 新着情報に紐付く遷移先の情報（リンク）

「重要なお知らせ」ポートレットはJavaEE開発モデルを利用して作成されていますが、スクリプト開発モデルで利用できるようにインタフェースが用意されています。
「重要なお知らせ」ポートレットに独自の新着情報を表示させるためには、下記の手順で新着情報を追加します。

1. **新着情報を取得するプロバイダを実装する。**

   「重要なお知らせ」ポートレットに表示される情報の配列を取得する以下のファンクションを実装したファンクションコンテナを作成します。

   - ファンクション名 ： getNewArrivedList
   - パラメータ

パラメータ名	型	内容
user	String	ユーザ ID
group	String	ログイングループ ID
properties	Object	設定ファイルに定義された、初期パラメータ

   このファンクションは、以下の新着情報オブジェクトの配列を返却する必要があります。

   - 新着情報オブジェクト

プロパティ	型	内容
category	String	新着情報の種類を示すためのカテゴリ
contents	String	新着情報の内容（概要）
url	String	新着情報の詳細情報へ遷移する URL（リンク）
popup	Boolean	詳細情報へ遷移する際にポップアップするかどうかのフラグ
date	String	新着情報の配信日付（システム日時）

   <sample_provider.js>

```
// getNewArrivedList 関数を実装します。
function getNewArrivedList(user, group, properties) {
 // 新着情報の配列を作成します。
 var values = new Array();
 for (var i = 0; i < result.countRow; i++) {
   values[i] = new Object();
   values[i].category = result.data[i].category;       // 新着情報の種類を示すためのカテゴリ
   values[i].contents = result.data[i].contents;       // 新着情報の内容（概要）
   values[i].date  =  result.data[i].arrived_date;     // 配信日付
   values[i].url  =  result.data[i].url;               // 新着情報に紐付く遷移先の情報（リンク）
   values[i].popup = ("1" == result.data[i].popup);    // リンクへ参照時にポップアップを呼び出すかどうか
 }
 // 新着情報の配列を返却します。
 return values;
}
```

2. **プロバイダの実装を定義する。**

実装したプロバイダクラスについての情報
は、**%PUBLIC_STORAGE_PATH%/portal/system_notice.xml** に 以下の情報を定義します。

<プロバイダ設定ファイル>

要素	説明
new-arrived-portlet/sort	取得した新着情報をカテゴリでソートするかどうかを定義します。
new-arrived-portlet/provider/provider-class	プロバイダの実装クラス。固定です。
new-arrived-portlet/provider/init-param	新着情報を取得するファンクションコンテナを「pagePath」パラメータに定義します。 それ以外の任意のパラメータを設定して、ファンクションコンテナで利用できます。

たとえば、上記のスクリプト開発モデルプログラムより新着情報を取得するプロバイダを使って新着情報を取得する場合は、以下のように定義を行います。

<system_notice.xml>

```
<new-arrived-portlet>
 <sort>true</sort>
  :
  :
 <provider>
    <!--  スクリプト開発モデル用のプロバイダクラス。固定とする  -->
    <provider-class>
      jp.co.intra_mart.foundation.portal.general_purpose_portlet.provider.PageBaseCallProvider
    </provider-class>
    <init-param>
      <!--  プロバイダを定義する  -->
      <param-name>pagePath</param-name>
      <param-value>/hoge/sample_provider.js</param-value>
    </init-param>
 </provider>
</new-arrived-portlet>
```

プロバイダを複数登録して、複数の新着情報を 1 つのポートレットに表示することも可能です。

ポートレットの開発（JavaEE 開発編）

目次

概要

この章では、JavaEE 開発モデルを利用してポートレットの作成を行うための手順や注意事項を説明します。
ポートレットを作成するためには、「*ポートレットのライフサイクル*」の内容を理解し、どのライフサイクルを利用するのかを決定する必要があります。

1. **画面開発 （render サイクル）**

   ポートレットに表示する画面を作成します。モードやウィンドウステータスを判定して切り替えることも可能です。
   表示モード用の画面は必ず必要となります。

2. **アクション処理 （processAction サイクル）**

   ポートレットからサブミットされたデータの処理を作成します。作成されたファンクションコンテナは、Action ハンドラと呼ばれ、ポートレット編集画面で登録することによって利用できます。
   また、Event を設定することによって、別のポートレットと連携することも可能です。

3. **イベント処理 （processEvent サイクル）**

   他のポートレットから発生したイベントの受信処理を作成します。作成されたファンクションコンテナは、
   Event ハンドラと呼ばれ、ポートレット編集画面で登録することによって利用できます。
   Action ハンドラで Event を設定された場合のみ実行されます。

ポートレットのライフサイクルごとに、以下のように作成するファイルが異なります。

render	必須	JavaEE 開発モデルの画面。主 JSP と HelperBean。
processAction	任意	handleAction PortletActionHandler インタフェースを実装した Java クラス。
processEvent	任意	PortletEventHandler インタフェースを実装した Java クラス。

全てのライフサイクル用プログラムを作成し、組み合わせることで 1 つのポートレットとして作成することも可能です。
ライフサイクルごとに別々に開発を行って、複数のポートレット間で組み合わせて利用できるように作成することも可能です。
次項より、ポートレットで利用可能な API と、各ライフサイクルでの実装方法について簡単に説明していきます。

ポートレットAPI

単純な画面表示のみのポートレットでは、ポートレットAPIを利用しなくても作成可能ですが、ポートレットの機能を最大限に利用するためには、ポートレット API を利用する必要があります。
特に Action 処理や Event 処理を行うためには、ポートレット API を利用しなくては実行できません。
このドキュメントでは、基本的な関数のみに絞って説明します。
全ての API については「ポータル API リスト」および「Portlet API2.0 のドキュメント」を参照してください。

ポートレット API2.0 のドキュメントは以下より取得してください。

- http://jcp.org/en/jsr/detail?id=286

PortalManager

JavaEE 開発モデルでは、PortletAPI2.0 をそのまま利用可能ですが、これらの API を簡単に利用するためのアクセッサとして、PortalManager クラスが用意されています。

```
jp.co.intra_mart.foundation.portal.common.PortalManager
```

PortalManager のそれぞれの関数については、次項以降で実際に利用する際に説明します。

ポートレットモード

JavaEE 開発モデルで作成されたポートレットは、ポートレットタイプが「JavaEE開発モデルポートレット」となります。
ポートレットモードの利用可否はページ種別ごとに設定され、「JavaEE開発モデルポートレット」の初期状態では、表示モードと編集モードが利用できます。

ポートレットモードの設定

ポートレット定義を変更し、ポートレットを初期化することでポートレットモードの設定を変更することができます。

- ポートレット定義の変更

ポートレットモードの設定は以下のファイルで管理されています。intra-mart Accel Platform を停止して設定後、再起動を行います。
設定ファイル

%CONTEXT_PATH%/WEB-INF/plugin/jp.co.intra_mart.portal.portlets.javaee_8.0.0/plugin.xml

設定箇所

```
<?xml version="1.0" encoding="utf-8"?>
<plugin>
  <extension point="jp.co.intra_mart.portal.portlets">
    <portlet
      id="jp.co.intra_mart.portal.portlets.javaee"
      name="JavaeeFwPortlet"
      display-name="%display-name"
      description="%description"
      version="8.0.0"
      rank="1">
      <portlet-class>jp.co.intra_mart.foundation.portal.portlets.model.JavaeeFwPortlet</portlet-class>
      <expiration-cache>0</expiration-cache>
      <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>VIEW</portlet-mode>
        <portlet-mode>EDIT</portlet-mode>
        <!-- ここにモードを追加 -->
      </supports>
      ...
    </portlet>
  </extension>
</plugin>
```

ポートレットモードの取得

API を利用して、以下のように現在のポートレットモードを取得できます。
ポートレットモードは、以下のような「javax.portlet.PortletMode」クラスの定数が取得されます。

(ex.) PortletMode.VIEW, PortletMode.EDIT, PortletMode.HELP

```
javax.portlet.RenderRequest renderRequest = PortalManager.getRenderRequest();
javax.portletPortletMode portletMode = renderRequest.getPortletMode();
```

ウィンドウステータス

ウィンドウステータスの取得

API を利用して、以下のように現在のポートレットモードを取得できます。
ウィンドウステータスは、以下のような「javax.portlet.WindowState」クラスの定数が取得されます。

また、最小化時にはポートレットが呼び出されませんので、「WindowState.MINIMIZED」が取得されることはありません。

(ex.) WindowState.NORMAL, WindowState.MAXIMIZED, WindowState.MINIMIZED

```
javax.portlet.RenderRequest renderRequest = PortalManager. getRenderRequest();
javax.portlet.WindowState windowState = renderRequest. getWindowState();
```

画面開発 （renderサイクル）

render サイクルでは、通常の Web ページと同様に画面表示が行われます。
JavaEE 開発モデルの画面は、おおむね通常の intra-mart の画面開発と変わりません。
ポートレット新規登録／編集画面 （「ポータル グループ管理者操作ガイド」参照）で設定されたページ引数は、HttpServletRequest オブジェクトの getParameter メソッドより取得することができます。
また、次項アクション処理、イベント処理によって設定されたリクエストパラメータも、HttpServletRequest オブジェクトより取得することができます。

RenderRequest, RenderResponse

ポートレットで利用する requestオブジェクトおよびresponse オブジェクトは、通常のWebページのリクエストは異なり、ポートレット表示時の専用のオブジェクトである、RenderRequestオブジェクトおよびRenderResponseオブジェクトを利用します。
これらのオブジェクトを利用することで、ポートレットの情報を取得することができます。

- RenderRequest, RenderResponse の取得

  ```
  RenderRequest renderRequest = PortalManager.getRenderRequest();
  RenderResponse renderResponse = PortalManager.getRenderResponse();
  ```

  ただし、通常の Web ページと共用できるように、ページ引数は、HttpServletRequest オブジェクト（JSP で「request」キーワードによりアクセスされる）を利用して取得することも可能です。

- JSP でのページ引数の取得

  ```
  <%
  RenderRequest renderRequest = PortalManager.getRenderRequest();
   // value1 と value2 は同じ値
   String value1 = request.getParameter("param1");
   String value2 = renderRequest.getParameter("param1");
  %>
  ```

  RenderRequest から、現在のポートレットモード、ウィンドウステータスなどが取得できます。

RenderRequestのスコープ

RenderRequest は通常の Web アプリケーションのリクエストとスコープが異なります。
全てのポートレットで独立したリクエストを保持しており、ポータル画面を表示する際のリクエストパラメータを引き継ぎません。
ただし以下のパラメータは、ポートレットコンテナにより、自動的に設定されます。

portal_cd	ポートレットが配置されているポータル画面のキー。
portalKind	ポートレットが配置されているポータル画面のポータル種別。以下の値が取得される。 user　ユーザポータル group　グループポータル

ポートレットにリクエストパラメータを設定するためには、アクション処理またはイベント処理を実行する必要があります。
また、一度設定したリクエストパラメータは、セッションが持続する間、再度アクション処理またはイベント処理が実行されるまで有効となります。

ActionURL, RenderURL

JavaEE 開発モデルで作成されたポートレットからサブミット処理を行い、 処理終了後に再びポータル画面を表示する機能を作成する場合には、ポートレットコンテナと通信するため、以下の URL に対してサブミットする必要があります。

ActionURL	Action 機能を呼び出す URL
RenderURL	ポータル画面を再表示するための URL

これらは、PortalManager を利用して取得することができます。

- ActionURL, RenderURL の取得

```
PortletURL actionURL = PortalManager.createActionURL();
PortletURL renderURL = PortalManager.createRenderURL();
```

以下に、JavaEE 開発モデルで開発されたポートレットでサブミットされた後に、アクション処理を呼び出すサンプルを示します。

- **JSP**

```
<%
 PortletURL actionURL = PortalManager.createActionURL();
%>
<!-- 何らかの登録処理を行うサービスを呼び出す -->
<form action="<%= actionURL.toString() %>" method="POST">
<!-- 登録処理で使用する値 -->
 <input type="text" name="param1" value="">
 <input type="submit" value="実行">
</form>
```

  - ポータル画面の表示には、「portal/desktop」というパスが割り振られています。
    そのため、このサービスID-URLにサブミットすることでもポータル画面の再表示が可能ですが、その場合ポータル情報を引き継ぐことができませんので、上記の方法を利用するようにしてください。
  - アクション処理を呼び出すためには、ActionURL にサブミットする以外にありません。
    ポートレット管理画面で Action ハンドラを登録し、プレゼンテーションページで ActionURL にサブミットするようにしてください。
    Action ハンドラが未定義の場合、単純にポータル画面を再表示します。
  - ActionURL はポータル情報を含むため、データ量が多くなります。メソッドは POST を指定するようにしてください。

通常の画面とポートレット画面を共用するには

共用するコードと通常の画面のみ必要なコードを別々の JSP ページとして作成し、通常の画面用の JSP ページでは、include タグを利用して共用ページを読み込みます。
ただし、このような利用を行う場合はポートレットの各 API は利用できませんので注意してください。

<ポートレット画面と共用する通常画面のプレゼンテーションページ>

```
<html>
<head>
 <link src="sample.css" type="text/css">

</head>
<body>
   <div>
      ヘッダ表示情報
 </div>
 <!-- 共用コードをインクルードする -->
 <jsp:include page="sample/sample-portlet.jsp" flush="true"/>
   <div>
      フッタ表示情報
 </div>
</body>
</html>
```

アクション処理 （processActionサイクル）

processAction サイクルでは画面表示がありませんので、PortletActionHandler インタフェースを実装した Action ハンドラクラスのみ作成します。
さらにポートレット管理画面で作成した Action ハンドラを登録し、ActionURL へサブミットすることでアクション処理が実行されます。
アクション処理では、RenderParamter の設定、PortletPreference の設定、イベントの設定などの更新処理を行います。

Actionハンドラの作成

以下にイベントを設定する Action ハンドラの例を示します。

1. **PortletActionHandler インタフェースを実装した Java クラスを作成し、以下の関数を定義します。**
   （この Java クラスを Action ハンドラと呼びます。）

   (ex.) sample.portal.SampleActionHandler.java

```
// 「PortletActionHandler」インタフェースを実装する。
public class SampleActionHandler implements PortletActionHandler {
  public Serializable handleAction(String portletCd, ActionRequest request, ActionResponse response) {
    // リクエスト引数を取得
    String value = request.getParameter("param1");
    // イベントを設定
    // intramart 用ポートレットにイベントを送信する場合、ImEvent オブジェクトを利用する。
    // QName は ImEvent.IM_QNAME を利用する。
    ImEvent event = new ImEvent();
    event.setEvent(value);
    response.setEvent(ImEvent.IM_QNAME, event);
    return null;
  }
}
```

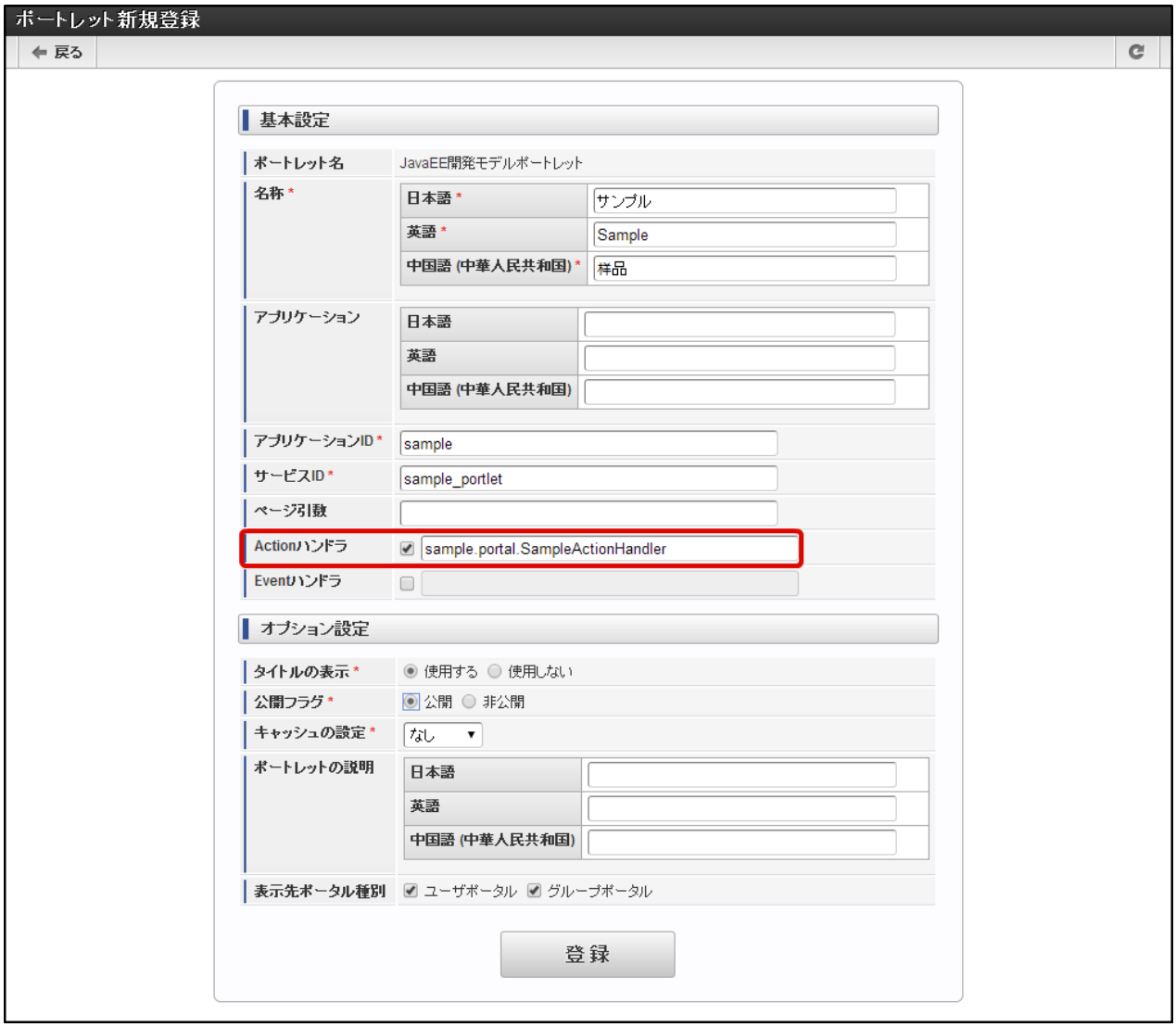
   - Action ハンドラは、render で利用するスクリプト開発モデルで作成されたファンクションコンテナを利用することもできます。
   - 以下のような値を return することでもイベントを発生させることができます。
     - jp.co.intra_mart.foundation.portal.common.handler .ImEvent のインスタンス

- java.util.Map のインスタンス ・・・Map のキーと値をペアとするイベント
- その他 ・・・ 文字列として判断し、キーと値が同じイベント

ただし、IM 用ポートレットに対するイベントを発生させるためには、ImEvent クラスを利用するか、**'im_'** で始まるキーを指定する必要があります。

- 詳細は「ポータル API リスト」の PortletActionHandler を参照してください。

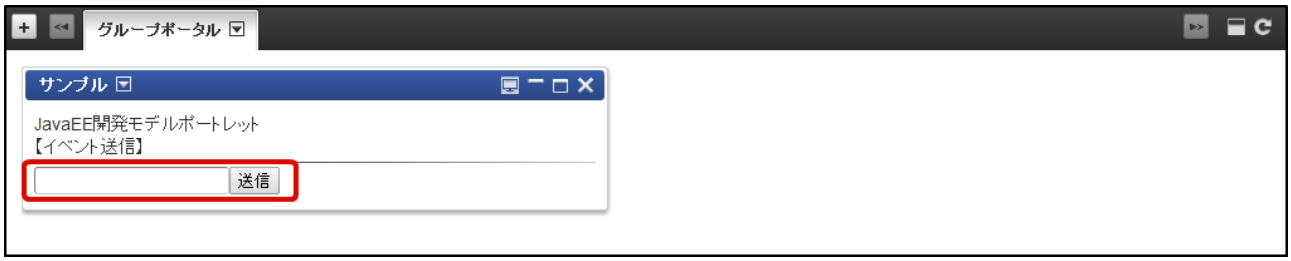
2. **Action ハンドラをポートレットに登録します。**



<ポートレット新規登録画面>

3. **「*画面開発 （renderサイクル）*」で作成した画面をポータル画面に配置し、ポートレットからアクション処理を呼び出します。**

Action処理実行後に、画面を再表示します。



<ポータル画面 - アクション処理>

ActionRequest, ActionResponse

Action ハンドラの引数は、アクション処理専用のオブジェクトである、ActionRequest オブジェクト、ActionResponseオブジェクトです。
これらのオブジェクトを利用することにより、リクエスト引数を取得したり、イベントを設定したりすることができます。
詳細は、「Portlet API2.0 ドキュメント」を参照してください。

RenderParameterの設定

画面表示時に取得可能なリクエストパラメータ(RenderParameter)は、アクション処理またはイベント処理の中で設定を行います。
アクション処理呼び出し時のリクエストパラメータは、ActionRequestに格納されていますが、 **renderサイクルには引き継がれないため** 、必要な場合は明示的に設定する必要があります。
また、アクション処理の呼び出し時には過去に設定された RenderParameter は全てクリアされています。

(ex.) processAction サイクルでの RenderParameter の設定

```
// リクエストパラメータを設定する場合。
response.setRenderParameter("param1", "value1");
// リクエストパラメータを削除する場合。
response.setRenderParameter("param2", null);
```

(ex.) render サイクルでの RenderParameter の利用

```
String value = request.getParamter("param1");
```

PortletPreferencesの設定

PortletPreferences とは、ユーザごとのポートレット情報保存領域です。
intra-mart Accel Platform のポートレットコンテナでは、PortletPreferences に設定された情報は、データベースに保存されます。
ここで設定された情報は、画面表示時にいつでも参照が可能です。

(ex.) javax.portlet.PortletPreferences の利用

```
// PortletPreferences の取得。
PortletPreferences preferences = request.getPreferences();
// PortletPreferences に設定された情報の取得。
String value = preferences.getValue("key1", "Default Value");
// PortletPreferences に情報を設定する。
preferences.setValue("key1", "value1");
// PortletPreferences を確定する。
//  この処理を行わない場合、情報は保存されません。
preferences.store();
```

イベントの設定

イベントを設定することにより、アクション処理を受け付けたポートレット以外の複数のポートレットにイベントの発生を通知して処理を引き渡すことが可能です。

(ex.) processEvent サイクルでのイベントの設定

```
response.setEvent(eventId, eventValue);
```

利用可能なActionハンドラ

intra-mart Accel Platform では、以下の Action ハンドラが標準で用意されています。

jp.co.intra_mart.foundation.portal.common.handler.DefaultPortletHandler

ポートレット新規登録／編集画面で、Action ハンドラチェックボックスにチェックを行うと、初期値としてこの Actionハンドラが設定されます。
この Action ハンドラは、リクエストパラメータから値を取得して、そのキーと値を用いてイベントを発生させます。
単純に画面からイベントを発生させたいだけの場合、Action ハンドラを作成せずに、このハンドラを利用することが可能です。
イベントを発生させるためには、以下のリクエストパラメータを設定します。

- “portlet.event.” + 任意の文字列

  これにより、以下のイベントが作成されます。

プロパティ名	内容
id	リクエストパラメータキーの任意の文字列
value	リクエストパラメータの値

  リクエストパラメータに複数設定することで、複数のイベントを発生させることも可能です。

jp.co.intra_mart.foundation.portal.common.handler.SetRenderParameterHandler

ポータル画面表示時のリクエストパラメータは、ポートレットとスコープが異なるため、ポートレット内の処理からは取得できません。
この Action ハンドラを設定することで、ポータル画面表示時のリクエストパラメータをポートレットで利用することが可能となります。

イベント処理 （processEventサイクル）

processEvent サイクルでは画面表示がありませんので、PortletEventHandler インタフェースを実装した Event ハンドラクラスのみ作成します。
さらにポートレット管理画面で作成した Event ハンドラを登録し、任意のポートレットのアクション処理でイベントが設定されることにより、イベント処理が実行されます。

イベント処理では、RenderParamter の設定、PortletPreferences の設定、イベントの設定などの更新処理を行います。

Eventハンドラの作成

以下に RenderParameter を設定する Event ハンドラの例を示します。

1. **PortletEventHandler インタフェースを実装した Java クラスを作成し、以下の関数を定義します。**

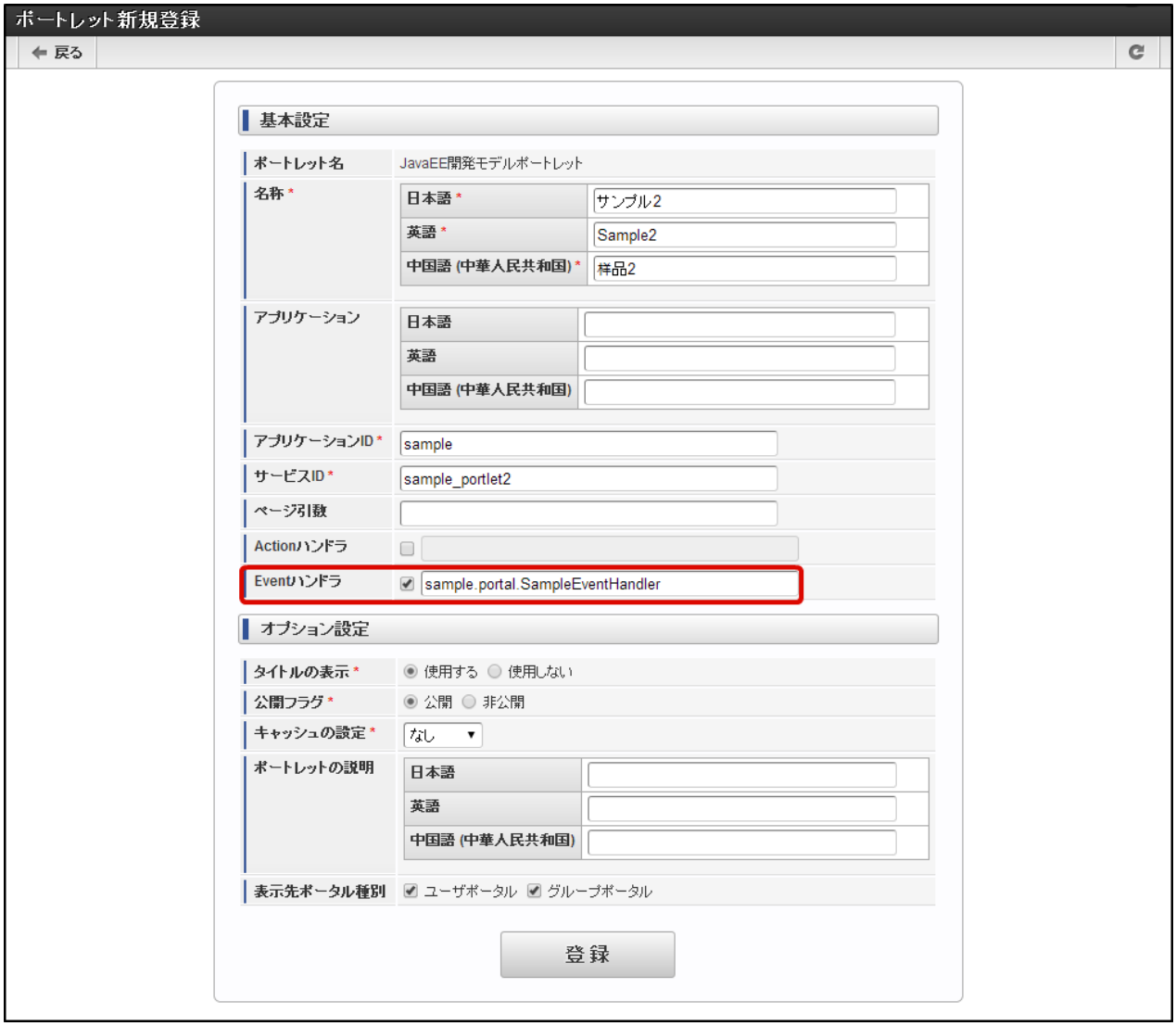
   （この Java クラスを Event ハンドラと呼びます。）

   (ex.) sample.portal.SampleEventHandler.java

```
// 「PortletEventHandler」インタフェースを実装する。
public class SampleEventHandler implements PortletEventHandler {
  public Serializable handleEvent(String portletCd, EventRequest request, EventResponse response) {
    // イベントオブジェクトを取得
    // intramart 用ポートレット間で利用するイベントオブジェクトは、「ImEvent」になります。
    ImEvent event = (ImEvent) request.getEvent().getValue();
    // 画面表示に利用するリクエストパラメータを設定
    response.setRenderParameter(event.getEventId(), event.getEvent());
    return null;
  }
}
```

- Event ハンドラは、render で利用するスクリプト開発モデルで作成されたファンクションコンテナを利用することもできます。
- 戻り値を設定することで、さらにイベントを発生させることができます。
  ※ループしないように注意して設計してください。

2. **Event ハンドラをポートレットに登録します。**



<ポートレット新規登録画面>

3. **「*画面開発 （renderサイクル）*」で作成した画面と「*アクション処理 （processActionサイクル）*」で作成したイベントを呼び出すことができるポートレットをポータル画面に配置し、イベントを設定するポートレットからアクション処理を呼び出します。**

   アクション処理からEventハンドラが実行され、その後、画面を再表示します。

<ポータル画面 - イベント処理>

ImEventオブジェクト

EventRequestから取得できる Eventオブジェクトは、intramartのポートレット間では、ImEventオブジェクトが設定されます。

ImEvent オブジェクトは以下の情報を格納しています。

プロパティ名	型	値
id	String	アクション処理で設定されたイベント ID 省略された場合、文字列「ImEvent」が設定される。
value	String	アクション処理で設定されたイベントの値 オブジェクトを設定した場合も文字列として取得される。
source	String	Event を設定したポートレットのポートレットコード

EventRequest, EventResponse

Event ハンドラの引数は、イベント処理専用のオブジェクトである、EventRequest オブジェクト、EventResponse オブジェクトです。
これらのオブジェクトを利用することにより、リクエスト引数を取得したり、さらにイベントを設定したりすることができます。
詳細は、「Portlet API2.0 ドキュメント」を参照してください。

RenderParameterの設定

画面表示時に取得可能なリクエストパラメータ(RenderParameter)は、アクション処理またはイベント処理の中で設定を行います。
イベント処理呼び出し時に設定済みであった RenderParameter は、EventRequest に格納されており、特に処理を行わない限り、 **render サイクルに自動的に引き継がれます** 。
引継ぎが不要な場合は明示的に削除する必要があります。

(ex.) processEvent サイクルでの RenderParameter の設定

```
// リクエストパラメータを設定する場合。
response.setRenderParameter("param1", "value1");
// リクエストパラメータを削除する場合。
response.setRenderParameter("param2", null);
```

(ex.) render サイクルでの RenderParameter の利用

```
request.getParamter(eventId);
```

PortletPreferencesの設定

PortletPreferences とは、ユーザごとのポートレット情報保存領域です。
intra-mart Accel Platform のポートレットコンテナでは、PortletPreferences に設定された情報は、データベースに保存されます。
ここで設定された情報は、画面表示時にいつでも参照が可能です。

(ex.) java.portlet.PortletPreferences の利用

```
// PortletPreferences の取得。
PortletPreferences preferences = request.getPreferences();
// PortletPreferences に設定された情報の取得。
String value = preferences.getValue("key1", "Default Value");
// PortletPreferences に情報を設定する。
preferences.setValue("key1", "value1");
// PortletPreferences を確定する。
//  この処理を行わない場合、情報は保存されません。
preferences.store();
```

イベントの設定

イベントを設定することにより、イベント処理を受け付けたポートレット以外の複数のポートレットにイベントの発生を通知して処理を引き渡すことが可能です。
ただし、イベント処理でさらにイベント処理を行うとパフォーマンスの低下をまねき、またイベントループの発生する可能性もありますので、十分注意して実装してください。

(ex.) processEvent サイクルでのイベントの設定

```
response.setEvent(eventId, eventValue);
```

「重要なお知らせ」ポートレットについて

ポータルモジュールには、汎用的な新着情報を表示する「重要なお知らせ」ポートレットが用意されています。
この「重要なお知らせ」ポートレットでは、新着情報として以下の項目を表示します。

- 新着情報の種類を示すためのカテゴリ
- 新着情報の内容（概要）
- 配信日付
- 新着情報に紐付く遷移先の情報（リンク）

「重要なお知らせ」ポートレットに独自の新着情報を表示させるためには、下記の手順で新着情報を追加します。

1. **新着情報を取得するプロバイダを実装する。**

   新着ポートレットに表示される情報は、以下のインタフェースを実装したプロバイダより取得されます。

   - 「重要なお知らせ」新着情報のプロバイダーのインタフェース
     jp.co.intra_mart.foundation.portal.general_purpose_portlet.provider.NewArrivedProvider

   「重要なお知らせ」ポートレットに、以下のプロバイダを利用して、パスワード履歴管理情報を表示していま

す。
独自のプロバイダを実装する場合の参考にしてください。

- パスワード履歴管理「重要なお知らせ」新着情報のプロバイダ
  jp.co.intra_mart.foundation.security.password.PasswordHistoryNewArrivedProvider

プロバイダのインタフェースおよびプロバイダの実装についての詳細は「ポータル API リスト」を参照してください。

2. **プロバイダの実装を定義する。**

実装したプロバイダクラスについての情報
は、**%PUBLIC_STORAGE_PATH%/portal/system_notice.xml** に 以下の情報を定義します。

<プロバイダ設定ファイル>

要素	説明
new-arrived-portlet/sort	取得した新着情報をカテゴリでソートするかどうかを定義します。
new-arrived-portlet/provider/provider-class	プロバイダの実装クラスを定義します。
new-arrived-portlet/provider/init-param	プロバイダクラスに設定する初期化パラメータを定義します。

以下に設定ファイルの記述例を示します。

<system_notice.xml>

```
<new-arrived-portlet>
 <sort>true</sort>
    :
    :
 <provider>
    <provider-class>
       sample.SampleProvider
    </provider-class>
    <init-param>
       <param-name>param1</param-name>
       <param-value>hogehoge</param-value>
    </init-param>
 </provider>
</new-arrived-portlet>
```

プロバイダを複数登録して、複数の新着情報を 1 つのポートレットに表示することも可能です。

ポートレットの開発（JSP/Servlet 編）

目次

概要

JSP/Servlet ポートレットは、画面表示処理を JSP/Servlet を利用して作成します。
通常の画面開発と同様に作成することが可能です。
ポートレットのライフサイクルを実現するために、Action ハンドラ、Event ハンドラを作成することができますが、その作成方法は、JavaEE フレームワークで作成する場合と変わりません。
JavaEE開発編の「*アクション処理 （processActionサイクル）*」および「*イベント処理 （processEventサイクル）*」を参照してください。

ポートレットの開発（SA Struts 編）

目次

概要

SA Struts ポートレットは、画面表示処理を SA Struts を利用して作成します。
通常の SA Struts 画面開発と同様に作成することが可能です。
ポートレットのライフサイクルを実現するために、Action ハンドラ、Event ハンドラを作成することができますが、その作成方法は、JavaEE フレームワークで作成する場合と変わりません。
JavaEE開発編の「*アクション処理 （processActionサイクル）*」および「*イベント処理 （processEventサイクル）*」を参照してください。

ポートレットの開発（Spring 編）

注意

Spring ポートレットは 2019 Summer(Waltz) 以降では利用できなくなりました。
ご利用のバージョンが 2019 Summer(Waltz) 以降の方は、「*ポートレットの開発（非同期ポートレット編）*」をご利用ください。

目次

概要

Spring ポートレットは、画面表示処理を Spring Framework を利用して作成します。
通常の Spring Framework ポートレット開発と同様に作成することが可能です。
ポートレットのライフサイクルを実現するために、@RenderMapping, @ActionMapping, @EventMapping アノテーションを利用します。
Spring Portlet MVC Framework の詳細については Portlet MVC Framework を参照してください。

1. **画面開発 （render サイクル）**

   ポートレットに表示する画面を作成します。モードやウィンドウステータスを判定して切り替えることも可能です。
   表示モード用の画面は必須です。

2. **アクション処理 （processAction サイクル）**

   ポートレットからサブミットされたデータの処理を作成します。 Controller クラスのメソッドに @ActionMapping アノテーションを設定することによって利用できます。
   また、Event を設定することによって、別のポートレットと連携することも可能です。

3. **イベント処理 （processEvent サイクル）**

   他のポートレットから発生したイベントの受信処理を作成します。 Controller クラスのメソッドに @EventMapping アノテーションを設定することによって利用できます。
   アクション処理 で Event を設定された場合のみ実行されます。

ポートレットのライフサイクルごとに、以下のようにメソッドに設定するアノテーションが異なります。

render	必須	@RenderMapping
processAction	任意	@ActionMapping
processEvent	任意	@EventMapping

次項より、ポートレットで利用可能な API と、各ライフサイクルでの実装方法について簡単に説明していきます。

Spring Portlet MVC Frameworkのポートレット開発との違い

主な違いは以下の通りです。

- portlet.xml
  Spring Portlet MVC Frameworkでは個々のポートレットごとにportlet.xmlに<portlet>を追加していましたが、 intra-mart Accel Platform では、個々には登録しません。
  %CONTEXT_PATH%/WEB-INF/plugin/jp.co.intra_mart.portal.portlets.spring_8.0.0/plugin.xmlの<portlet-name>SpringPortlet</portlet-name>の設定を元にポートレットを動作させます。
  portlet-name は、「Springポートレット識別子」に設定した値が設定されます。
- portletのcontextのxmlファイルの命名規則と配置場所
  classpath:META-INF/spring/[Springポートレット識別子]-portlet.xml です。
  詳細は、「*portlet context のxmlファイル* 」を参照してください。
- JSP
  HEADやBODYタグなどが使えないなど、注意事項があります。
  詳細は 「*画面を作成する上での注意事項*」を参照してください。

ポートレットAPI

単純な画面表示のみのポートレットでは、ポートレットAPIを利用しなくても作成可能ですが、ポートレットの機能を

最大限に利用するためには、ポートレット API を利用する必要があります。
特に Action 処理や Event 処理を行うためには、ポートレット API を利用しなくては実行できません。
このドキュメントでは、基本的な関数のみに絞って説明します。
全ての API については「ポータル API リスト」および「Portlet API2.0 のドキュメント」を参照してください。

ポートレット API2.0 のドキュメントは以下より取得してください。

- http://jcp.org/en/jsr/detail?id=286

PortalManager

PortletAPI2.0 をそのまま利用可能ですが、これらの API を簡単に利用するためのアクセッサとして、PortalManager クラスが用意されています。

```
jp.co.intra_mart.foundation.portal.common.PortalManager
```

PortalManager のそれぞれの関数については、次項以降で実際に利用する際に説明します。

ポートレットモード

利用できるポートレットモードは、表示モード(VIEW)と編集モード(EDIT)です。

ポートレットモードの設定

ポートレット定義を変更し、ポートレットを初期化することでポートレットモードの設定を変更できます。

- ポートレット定義の変更
  ポートレットモードの設定は以下のファイルで管理されています。intra-mart Accel Platform を停止して設定後、再起動を行います。
  設定ファイル

  %CONTEXT_PATH%/WEB-INF/plugin/jp.co.intra_mart.portal.portlets.spring_8.0.0/plugin.xml

  設定箇所

```
<?xml version="1.0" encoding="utf-8"?>
<plugin>
  <extension point="jp.co.intra_mart.portal.portlets">
    <portlet
      id="jp.co.intra_mart.portal.portlets.spring"
      name="SpringPortlet"
      display-name="%display-name"
      description="%description"
      version="8.0.0"
      rank="1">
      <portlet-class>jp.co.intra_mart.foundation.portal.portlets.model.SpringPortlet</portlet-class>
      <expiration-cache>0</expiration-cache>
      <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>VIEW</portlet-mode>
        <portlet-mode>EDIT</portlet-mode>
        <!-- ここにモードを追加 -->
      </supports>
      ...
    </portlet>
  </extension>
</plugin>
```

ポートレットモードの取得

API を利用して、以下のように現在のポートレットモードを取得できます。
ポートレットモードは、以下のような「javax.portlet.PortletMode」クラスの定数が取得されます。

(ex.) PortletMode.VIEW, PortletMode.EDIT

```
javax.portlet.RenderRequest renderRequest = PortalManager.getRenderRequest();
javax.portlet.PortletMode portletMode = renderRequest.getPortletMode();
```

ウィンドウステータス

ウィンドウステータスの取得

API を利用して、以下のように現在のポートレットモードを取得できます。
ウィンドウステータスは、以下のような「javax.portlet.WindowState」クラスの定数が取得されます。

また、最小化時にはポートレットが呼び出されませんので、「WindowState.MINIMIZED」が取得されることはありません。

(ex.) WindowState.NORMAL, WindowState.MAXIMIZED, WindowState.MINIMIZED

```
javax.portlet.RenderRequest renderRequest = PortalManager.getRenderRequest();
javax.portlet.WindowState windowState = renderRequest.getWindowState();
```

portlet context のxmlファイル

Spring Portletのcontextのxmlファイルのファイル名の命名規則と配置場所は以下の通りです。

- **ファイル名**
  [Springポートレット識別子]-portlet.xml 形式で作成してください。
  (ex.) Springポートレット識別子に 「welcome」 と設定した場合、 welcome-portlet.xml です。
- **配置場所**
  モジュール・プロジェクトでは、src/main/resources に 「 META-INF/spring」 フォルダを作成し、xmlファイルを配置してください。
  (ex.) src/main/resources/META-INF/spring/welcome-portlet.xml

xmlに、Controller クラスをcomponent scanの対象になるように設定してください。

```
<context:component-scan base-package="Controller クラスのパッケージ" />
```

画面開発 （renderサイクル）

render サイクルでは、通常の Web ページと同様に viewとしてJSPを使った画面表示が行われます。
Springモデルの画面は、おおむね通常の Spring Portlet MVC Framework の画面開発と変わりません。
ポートレット新規登録／編集画面 （「ポータル グループ管理者操作ガイド」参照）で設定するSpringポートレット識別子は、通常のポートレット開発のポートレット名に対応します（通常のポートレット開発の、portlet.xmlのportlet-nameの部分。）

Controller クラス

@Controller アノテーション、 @RequestMapping アノテーション を設定した Controllerクラスを作成し、
@RenderMapping アノテーションを設定したメソッドを作成します。
@RequestMapping にはポートレットモード(VIEW または EDIT)を指定します。
renderを処理するメソッドに @RenderMapping アノテーションを設定します。
パラメータでrenderを処理するメソッドを選択するには、 @RenderMapping(params = “action=list”) のように設定します。

```
@Controller
@RequestMapping("VIEW")
public class ViewController {

   // default render
   @RenderMapping
   public String view() {
      return "welcome/view.jsp";
   }

   // renderURL.setParameter("action", "list") に対応するrender
   @RenderMapping(params = "action=list")
   public String list() {
      return "welcome/list.jsp";
   }
}
```

JSP ファイル

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

<%@ page import="jp.co.intra_mart.foundation.portal.common.PortalManager" %>
<%@ page import="javax.portlet.PortletURL" %>

<%
PortletURL renderURL = PortalManager.createRenderURL();
%>
@RenderMapping に対応する。
<a href="<%=renderURL.toString() %>">default render</a>
<br/>

<%
PortletURL renderURL2 = PortalManager.createRenderURL();
renderURL2.setParameter("action", "list");
%>
@RenderMapping(params = "action=list") に対応する。
<a href="<%=renderURL2.toString() %>">render (action=list)</a>
<br/>
```

RenderRequest, RenderResponse

ポートレットで利用する requestオブジェクトおよびresponse オブジェクトは、通常のWebページのリクエストは異なり、ポートレット表示時の専用のオブジェクトである、RenderRequestオブジェクトおよびRenderResponseオブジェクトを利用します。
これらのオブジェクトを利用することで、ポートレットの情報を取得できます。

- RenderRequest, RenderResponse の取得

  ```
  RenderRequest renderRequest = PortalManager.getRenderRequest();
  RenderResponse renderResponse = PortalManager.getRenderResponse();
  ```

- JSP でのページ引数の取得

  ```
  <%
    RenderRequest renderRequest = PortalManager.getRenderRequest();
    String value = renderRequest.getParameter("param1");
  %>
  ```

  RenderRequest から、現在のポートレットモード、ウィンドウステータスなどが取得できます。

RenderRequestのスコープ

RenderRequest は通常の Web アプリケーションのリクエストとスコープが異なります。
全てのポートレットで独立したリクエストを保持しており、ポータル画面を表示する際のリクエストパラメータを引き継ぎません。
ただし以下のパラメータは、ポートレットコンテナにより、自動的に設定されます。

portal_cd	ポートレットが配置されているポータル画面のキー。
portalKind	ポートレットが配置されているポータル画面のポータル種別。以下の値が取得される。 user　ユーザポータル group　グループポータル

ポートレットにリクエストパラメータを設定するためには、アクション処理またはイベント処理を実行する必要があります。
また、一度設定したリクエストパラメータは、セッションが持続する間、再度アクション処理またはイベント処理が実行されるまで有効です。

ActionURL, RenderURL

ポートレットからサブミット処理を行い、 処理終了後に再びポータル画面を表示する機能を作成する場合には、ポートレットコンテナと通信するため、以下の URL に対してサブミットする必要があります。

ActionURL	Action 機能を呼び出す URL
RenderURL	ポータル画面を再表示するための URL

これらは、PortalManager を利用して取得できます。

- ActionURL, RenderURL の取得

```
PortletURL actionURL = PortalManager.createActionURL();
PortletURL renderURL = PortalManager.createRenderURL();
```

以下に、ポートレットでサブミットされた後に、アクション処理を呼び出すサンプルを示します。

- **JSP**

```
<%
 PortletURL actionURL = PortalManager.createActionURL();
%>
<!-- 何らかの登録処理を行うサービスを呼び出す -->
<form action="<%= actionURL.toString() %>" method="POST">
<!-- 登録処理で使用する値 -->
 <input type="text" name="param1" value="">
 <input type="submit" value="実行">
</form>
```

  - アクション処理を呼び出すためには、ActionURL にサブミットする以外にありません。
  - ActionURL はポータル情報を含むため、データ量が多くなります。メソッドは POST を指定するようにしてください。

  パラメータを指定してAction処理を呼び出す場合

```
<%
 PortletURL actionURL = PortalManager.createActionURL();
 actionURL.setParameter("action", "register");
%>
<!-- 何らかの登録処理を行うサービスを呼び出す -->
<form action="<%= actionURL.toString() %>" method="POST">
<!-- 登録処理で使用する値 -->
 <input type="text" name="param1" value="">
 <input type="submit" value="実行">
</form>
```

  - setParameterを使ってパラメータを設定し、アクション処理を呼び出します。
    この場合、 「action=register」です。

アクション処理 （processActionサイクル）

アクション処理は、 Controllerクラスのメソッドに @ActionMapping アノテーションを設定します。
アクション処理を呼び出すためには、JSPなどで作成した画面から、ActionURLのURL へサブミットすることでアクション処理が実行されます。
アクション処理では、RenderParameter の設定、PortletPreference の設定、イベントの設定などの更新処理を行います。

Controller クラス

@Controller アノテーション、 @RequestMapping アノテーション を設定した Controllerクラスを作成し、@ActionMapping アノテーションを設定したメソッドを作成します。
@RequestMapping にはポートレットモード(VIEW または EDIT)を指定します。

```
import jp.co.intra_mart.foundation.portal.common.handler.ImEvent;

@Controller
@RequestMapping("VIEW")
public class ViewController {

   // default render
   @RenderMapping
   public String view() {
      return "welcome/view.jsp";
   }

   // renderURL.setParameter("action", "list") に対応するrender
   @RenderMapping(params = "action=list")
   public String list() {
      return "welcome/list.jsp";
   }

   // default action
   @ActionMapping
   public void register(@RequestParam("name") String name, ActionResponse response) {
      // 登録処理 ...

      // @RenderMapping(params="action=list") のrenderへ遷移させる。
      response.setRenderParameter("action", "list");
   }

   // actionURL.setParameter("action", "publish") に対応するaction
   // eventを発行するaction
   @ActionMapping(params = "action=publish")
   public void publish(ActionRequest request, ActionResponse response) {
      // イベント発行
      ImEvent event = new ImEvent();
      event.setEvent("hello");
      response.setEvent(ImEvent.IM_QNAME, event);
   }
}
```

JSP ファイル

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

<%@ page import="jp.co.intra_mart.foundation.portal.common.PortalManager" %>
<%@ page import="javax.portlet.PortletURL" %>

<%
PortletURL renderURL = PortalManager.createRenderURL();
%>
@RenderMapping に対応する。
<a href="<%=renderURL.toString() %>">default render</a>
<br/>

<%
PortletURL renderURL2 = PortalManager.createRenderURL();
renderURL2.setParameter("action", "list");
%>
@RenderMapping(params = "action=list") に対応する。
<a href="<%=renderURL2.toString() %>">render (action=list)</a>
<br/>

<%
PortletURL actionURL = PortalManager.createActionURL();
%>
@ActionMapping に対応する。
<form method="post" action="<%=actionURL.toString() %>">
 <input type="text" name="name" value="">
 <button type="submit">登録</button>
</form>
<br/>

<%
PortletURL actionURL2 = PortalManager.createActionURL();
actionURL2.setParameter("action", "publish");
%>
@ActionMapping(params = "action=publish") に対応する。
<a href="<%=actionURL2.toString() %>">イベント発行</a>
<br/>
```

ActionRequest, ActionResponse

アクション処理のメソッドの引数は、アクション処理専用のオブジェクトである、ActionRequest オブジェクト、ActionResponse オブジェクトです。
これらのオブジェクトを利用することで、リクエスト引数の取得や、イベントの設定ができます。
詳細は、「Portlet API2.0 ドキュメント」を参照してください。

RenderParameterの設定

画面表示時に取得可能なリクエストパラメータ(RenderParameter)は、アクション処理またはイベント処理の中で設定を行います。
アクション処理呼び出し時のリクエストパラメータは、ActionRequestに格納されていますが、 **renderサイクルには引き継がれないため** 、必要な場合は明示的に設定する必要があります。
また、アクション処理の呼び出し時には過去に設定された RenderParameter は全てクリアされています。

(ex.) processAction サイクルでの RenderParameter の設定

```
// ActionResponse response
// リクエストパラメータを設定する場合。
response.setRenderParameter("param1", "value1");
// リクエストパラメータを削除する場合。
response.setRenderParameter("param2", null);
```

(ex.) render サイクルでの RenderParameter の利用

```
String value = request.getParameter("param1");
```

PortletPreferencesの設定

PortletPreferences とは、ユーザごとのポートレット情報保存領域です。
intra-mart Accel Platform のポートレットコンテナでは、PortletPreferences に設定された情報は、データベースに保存されます。
ここで設定された情報は、画面表示時にいつでも参照が可能です。

(ex.) javax.portlet.PortletPreferences の利用

```
// PortletPreferences の取得。
PortletPreferences preferences = request.getPreferences();
// PortletPreferences に設定された情報の取得。
String value = preferences.getValue("key1", "Default Value");
// PortletPreferences に情報を設定する。
preferences.setValue("key1", "value1");
// PortletPreferences を確定する。
//  この処理を行わない場合、情報は保存されません。
preferences.store();
```

イベントの設定

イベントを設定することにより、アクション処理を受け付けたポートレット以外の複数のポートレットにイベントの発生を通知して処理を引き渡すことが可能です。
event idには jp.co.intra_mart.foundation.portal.common.handler.ImEvent.IM_QNAME を設定してください。

(ex.) processAction サイクルでのイベントの設定

```
@ActionMapping(params = "action=publish")
public String publish(ActionRequest request, ActionResponse response) {
    ImEvent event = new ImEvent();
    event.setEvent("hello");
    response.setEvent(ImEvent.IM_QNAME, event);
}
```

イベント処理 （processEventサイクル）

processEvent サイクルでは画面表示がありませんので、@EventMapping アノテーションを設定したメソッドのみ作成します。

イベント処理では、RenderParameter の設定、PortletPreferences の設定、イベントの設定などの更新処理を行います。

Controller クラス

@Controller アノテーション、 @RequestMapping アノテーション を設定した Controllerクラスを作成し、
@EventMapping アノテーションを設定したメソッドを作成します。
@RequestMapping にはポートレットモード(VIEW または EDIT)を指定します。
@EventMapping には、 @EventMapping(“ImEvent”) と ImEvent を指定してください。

```
import jp.co.intra_mart.foundation.portal.common.handler.ImEvent;

@Controller
@RequestMapping("VIEW")
public class AnotherController {

   // default render
   @RenderMapping
   public String view() {
      return "receive/view.jsp";
   }

   // event
   @EventMapping("ImEvent")
   public void event(EventRequest request, EventResponse response) {
      Event event = request.getEvent();
      ImEvent imEvent = (ImEvent) event.getValue();
      if (!"hello".equals(imEvent.getEvent())) {
         // "hello"のイベント以外では、処理を実行しない。
         return;
      }
      // 何らかの処理...
   }
}
```

ImEventオブジェクト

EventRequestから取得できる Eventオブジェクトは、intramartのポートレット間では、ImEventオブジェクトが設定されます。

ImEvent オブジェクトは以下の情報を格納しています。

プロパティ名	型	値
id	String	アクション処理で設定されたイベント ID 省略された場合、文字列「ImEvent」が設定される。
value	String	アクション処理で設定されたイベントの値 オブジェクトを設定した場合も文字列として取得される。
source	String	Event を設定したポートレットのポートレットコード

EventRequest, EventResponse

イベントのメソッドの引数は、イベント処理専用のオブジェクトである、EventRequest オブジェクト、
EventResponse オブジェクトを使うことができます。
これらのオブジェクトを利用することで、リクエスト引数の取得や、イベントの設定ができます。
詳細は、「Portlet API2.0 ドキュメント」を参照してください。

RenderParameterの設定

画面表示時に取得可能なリクエストパラメータ(RenderParameter)は、アクション処理またはイベント処理の中で設定を行います。
イベント処理呼び出し時に設定済みであった RenderParameter は、EventRequest に格納されており、特に処理を行わない限り、 **render サイクルに自動的に引き継がれます** 。
引継ぎが不要な場合は明示的に削除する必要があります。

(ex.) processEvent サイクルでの RenderParameter の設定

```
// リクエストパラメータを設定する場合。
response.setRenderParameter("param1", "value1");
// リクエストパラメータを削除する場合。
response.setRenderParameter("param2", null);
```

(ex.) render サイクルでの RenderParameter の利用

```
request.getParameter(eventId);
```

PortletPreferencesの設定

PortletPreferences とは、ユーザごとのポートレット情報保存領域です。
intra-mart Accel Platform のポートレットコンテナでは、PortletPreferences に設定された情報は、データベースに保存されます。
ここで設定された情報は、画面表示時にいつでも参照が可能です。

(ex.) java.portlet.PortletPreferences の利用

```
// PortletPreferences の取得。
PortletPreferences preferences = request.getPreferences();
// PortletPreferences に設定された情報の取得。
String value = preferences.getValue("key1", "Default Value");
// PortletPreferences に情報を設定する。
preferences.setValue("key1", "value1");
// PortletPreferences を確定する。
//  この処理を行わない場合、情報は保存されません。
preferences.store();
```

イベントの設定

イベントを設定することにより、イベント処理を受け付けたポートレット以外の複数のポートレットにイベントの発生を通知して処理を引き渡すことが可能です。
ただし、イベント処理でさらにイベント処理を行うとパフォーマンスの低下をまねき、またイベントループの発生する可能性もありますので、十分注意して実装してください。

(ex.) processEvent サイクルでのイベントの設定

```
response.setEvent(eventId, eventValue);
```

Ajax （serveResourceサイクル）

Ajaxを使ってリソースを取得するために、serveResourceサイクルで動的なリソースを取得します。

ここでは、Ajaxを使ってリソースを取得する方法について説明します。

- Controllerクラスに @ResourceMapping アノテーションを設定したメソッドを作成します。
- portletのbean定義xmlに、Jacksonの設定を追加します。

Controller クラス

@Controller アノテーション、 @RequestMapping アノテーション を設定した Controllerクラスを作成し、
@ResourceMapping アノテーションを設定したメソッドを作成します。
@RequestMapping にはポートレットモード(VIEW または EDIT)を指定します。
Ajaxで呼び出される処理を実行するメソッドに @ResourceMapping アノテーションを設定します。
リソースID “sample” と対応付けるには、 @ResourceMapping(“sample”) のように設定します。
JSON文字列を返すために、 @ResourceMappingアノテーションを設定したメソッドの返り値のタイプに JsonResult を設定します。

```
@Controller
@RequestMapping("VIEW")
public class ViewController {

   // default render
   @RenderMapping
   public String view() {
      return "welcome/view.jsp";
   }

   // resource id = fetch
   @ResourceMapping("fetch")
   public JsonResult fetch() {
      Map<String, String> result;

      // result にデータをセットする処理
      // ...

      // JsonResultにデータをセットして返す。
      return new JsonResult(result);
   }
}
```

JSP ファイル

JsonResultのresultプロパティに設定した情報は、クライアント側では、data.result に設定されています。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

<%@ page import="jp.co.intra_mart.foundation.portal.common.PortalManager" %>
<%@ page import="javax.portlet.ResourceURL" %>

<%
ResourceURL resourceURL = PortalManager.createResourceURL();
resourceURL.setResourceID("fetch");
%>

<imui:head>
<script type="text/javascript">
(function ($) {
  fetchData = function () {
    $.ajax({
      dataType : 'json',
      url : '<%=resourceURL.toString()%>', // resource id = "fetch"のURL
      type :'GET',
      success : function (data, textStatus, jqXHR) {
        var result = data.result; // data.resultとJsonResult().getResult()が対応しています。
        // 取得データを使った何らかの処理
      }
    });
  };
})(jQuery);
</script>
</imui:head>

<a href="javascript:void(0);" onclick="fetchData();">resource id = "fetch"</a>
```

portlet context の xmlファイル

AnnotationMethodHandlerAdapterのcustomModelAndViewResolverプロパティにJsonModelAndViewResolverを設定します。
これにより、コントローラークラスのメソッドの戻り値が JsonResult の場合、viewとしてMappingJackson2JsonView を使い、responseにJSONを返します。

```
    <bean class="org.springframework.web.portlet.mvc.annotation.AnnotationMethodHandlerAdapter">
      <property name="customModelAndViewResolver">
        <bean
class="jp.co.intra_mart.framework.extension.spring.web.portlet.mvc.JsonModelAndViewResolver" />
      </property>
    </bean>
```

java.util.Date型プロパティのJSONでのフォーマットについて

Jacksonのデフォルトの設定では、java.util.Date型のプロパティをJSONで出力すると、エポックミリ秒（1970年1月1日からのミリ秒）で出力されます。
アカウントごとに定義した日付フォーマットを適用して、Date型のプロパティをフォーマットしてJSONに出力するには、以下のようにします。

- モデルクラスのDate型プロパティにアノテーションを設定をする。
- portlet context の xmlファイルにアノテーションを有効にする設定をする。

モデルクラスへの設定

JsonResultにセットするモデルクラスのDate型プロパティに AccountDateFormat、AccountDateTimeFormat、AccountTimeFormat アノテーションを付けます。
それぞれのアノテーションについては、 「APIリスト jp.co.intra_mart.framework.extension.spring.format.annotation パッケージ」 を参照してください。
アノテーションを付けない場合は、エポックミリ秒のままです。

```
import java.util.Date;

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

public class DateForm {

   // アノテーションなし
   private Date date;

   // アノテーションあり
   @AccountDateFormat
   private Date dateDefault;

   // アノテーションあり
   @AccountDateFormat(type = TYPE.SIMPLE)
   private Date dateSimple;

   // setter and getter...
}
```

portlet context の xmlファイルへの設定
JsonModelAndViewResolverのmappingJacksonJsonViewプロパティに、MappingJackson2JsonViewを設定し、このMappingJackson2JsonViewのobjectMapperプロパティにAccountDateObjectMapperを設定します。
objectMapperにAccountDateObjectMapperを設定したMappingJackson2JsonViewを使うことにより、モデルをJsonに変換するときに、AccountDateFormat、AccountDateTimeFormat、AccountTimeFormat アノテーションの設定を有効にできます。
CVE-2018-11040 対応として、 MappingJackson2JsonView を利用する場合は、 jsonpParameterNames プロパティに空のSetを設定します。

```
   <bean class="org.springframework.web.portlet.mvc.annotation.AnnotationMethodHandlerAdapter">
      <property name="customModelAndViewResolver">
         <bean
class="jp.co.intra_mart.framework.extension.spring.web.portlet.mvc.JsonModelAndViewResolver">
            <property name="mappingJacksonJsonView">
               <bean class="org.springframework.web.servlet.view.json.MappingJackson2JsonView">
                  <property name="jsonpParameterNames" value="#{T(java.util.Collections).EMPTY_SET}"
/>
                  <property name="objectMapper">
                     <bean
class="jp.co.intra_mart.framework.extension.spring.http.converter.json.AccountDateObjectMapper" />
                  </property>
               </bean>
            </property>
         </bean>
      </property>
   </bean>
```

アカウントごとに定義した日付フォーマットについては、「一般ユーザ操作ガイド」 - 「日付と時刻の形式を設定する」または APIリスト「DateTimeFormatIds」を参照してください。

ポートレットの開発（非同期ポートレット編）

目次

概要

非同期ポートレットは、intra-mart Accel Platform で作成した画面を 非同期ポートレット内のiFrameに表示します。

通常の画面開発と方法は変わりませんが以下の点に注意する必要があります。

- 非同期ポートレットに表示される内容（開発した画面）は、iFrame内に表示されるため、内容は通常のリクエストにより取得されます。
- Action ハンドラ、Event ハンドラは利用できません。
- 開発した画面は、通常の画面としてアクセスされるため、URLに対するリソース登録および認可の設定が必要となります。 認可の設定が無い場合、直接URLを入力することでポータルの参照権限が無いユーザが画面を表示させることができてしまいます。 ポータルの参照権限が無いユーザからのアクセスを抑止したい場合は、ポートレット用の画面に対してポータルの参照権限と同等のアクセス権限を付与してください。
- 開発した画面には、通常 テーマ（ヘッダー、フッター）が付加されますので、これを付与されないようにする設定が必要です。

ポートレットモード

非同期ポートレットでサポートするポートレットモードは、表示モードのみです。

画面開発（スクリプト開発）

1. **ポートレット用の画面を作成します。**
   表示したい内容は で囲む必要があります。

   (ex.) WEB-INF/jssp/src/sample/async_portlet.html

```
<IMART type="head">

 // <head>タグ内に記述したいコードを記述します。

</IMART>

<div id=”imui-container”>

  // ここに表示したい内容を記述します。

</div>
```

必要であればプレゼンテーションページも作成してください。

2. **ルーティングとリソースの登録を行います。**

   方法については、「認可仕様書」または「スクリプト開発モデル プログラミングガイド」を参照してください。

3. **アクセス制御の設定を行います。**

   方法については、「認可仕様書」または「スクリプト開発モデル プログラミングガイド」を参照してください。

画面開発（JavaEE開発）

1. **ポートレット用の画面を作成します。**
   IM-JavaEE Frameworkの仕様に従って画面を開発します。
   表示したい内容は で囲む必要があります。

   (ex.) 画面用JSP sample/async_portlet.jsp

```
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui"%>

<imui:head>

    // <head>タグ内に記述したいコードを記述します。

</imui:head>

<div id=”imui-container”>

  // ここに表示したい内容を記述します。

</div>
```

2. **ルーティングとリソースの登録を行います。**

   方法については「認可仕様書」を参照してください。

3. **アクセス制御の設定を行います。**

   方法については「認可仕様書」または「 *ポートレットのアクセス制御* 」を参照してください。

画面開発（SA Struts開発）

1. **ポートレット用の画面を作成します。**
   SA Strutsの仕様に従って画面を開発します。

表示したい内容は で囲む必要があります。

(ex.) 画面用JSP sample/async_portlet.jsp

```
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui"%>

<imui:head>

    // <head>タグ内に記述したいコードを記述します。

</imui:head>

<div id=”imui-container”>

  // ここに表示したい内容を記述します。

</div>
```

2. **ルーティングとリソースの登録を行います。**

   方法については「認可仕様書」または「SAStruts+S2JDBC プログラミングガイド」を参照してください。

3. **アクセス制御の設定を行います。**

   方法については「認可仕様書」または「SAStruts+S2JDBC プログラミングガイド」を参照してください。

画面開発（ TERASOLUNA Server Framework for Java (5.x) 開発)

1. **ポートレット用の画面を作成します。**
   TERASOLUNA Server Framework for Java (5.x) for Accel Platform の仕様に従って画面を開発します。
   表示したい内容は で囲む必要があります。

   (ex.) 画面用JSP sample/async_portlet.jsp

```
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui"%>

<imui:head>

    // <head>タグ内に記述したいコードを記述します。

</imui:head>

<div id=”imui-container”>

  // ここに表示したい内容を記述します。

</div>
```

2. **ルーティングとリソースの登録を行います。**

   方法については「認可仕様書」または「 TERASOLUNA Server Framework for Java (5.x) プログラミングガイド」を参照してください。

3. **アクセス制御の設定を行います。**

   方法については「認可仕様書」または「 TERASOLUNA Server Framework for Java (5.x) プログラミングガイド」を参照してください。

テーマの付与を行わないようにするには

(ex.) URL sample/async_portletに対して、テーマ（ヘッダー、フッター）の付与を行わないようにします。

WEB-INF/conf/theme-head-only-path-config フォルダにユニークな名前のファイルを作成します。（拡張子はxml）

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

  <path>/sample/async_portlet</path>

</theme-head-only-path-config>
```

ポートレットのアクセス制御

目次

概要

ポートレットがブラウザ等からURL指定で直接アクセスされた場合、ポートレットの呼び出しはポートレットコンテナを介さないため
ポータルに設定されたアクセス権限による制御は実行されません。
そのため、本来ポートレットを参照できないユーザが直接アクセスを行った場合に、意図せずポートレットが実行されてしまいます。

意図しない実行を抑止するためには直接アクセスを制御（禁止／指定されたユーザにのみ許可）する必要があります。
本稿ではポートレットに対する直接アクセスに関して、認可機構を利用して直接アクセスを制御する方法を説明します。

アクセス制御の方法

ポートレットモジュールのURLを認可リソースとして登録し、登録した認可リソースに対して認可ポリシーの設定を行います。
直接アクセスを制御するために認可設定が必要な開発モデルは以下の通りです。

- JavaEE 開発モデル
- JSP/サーブレット
- SA Struts

認可設定の方法

JavaEE 開発モデルを例に、直接アクセスを制御する方法を説明します。

1. ポートレットモジュール用に認可リソースURIを割り当てます。

   例えば、以下のような認可リソースURIを決定します。

   `service://portal/sample-direct-access`

2. ポートレットモジュールのルーティング設定（routing-service-config）を行います。

   ポートレットモジュール用の認可リソースURIをルーティング設定に割り当てます。

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

        <!-- ポートレットモジュールのURL -->
        <service-mapping path="javaee_sample-test_portlet.service"
            application="portal" service="test_portlet">
            <authz uri="service://portal/sample-direct-access" action="execute" />
        </service-mapping>

</routing-service-config>
```

3. ポートレットモジュール用の認可リソースを登録します。

   認可設定については、「スクリプト開発モデル プログラミングガイド」 - 「認可」 - 「画面へアクセスするための認可設定方法」を参照してください。
   スクリプト開発モデルの認可設定方法が記載されていますが、手順は同じです。

4. ポートレットモジュール用の認可リソースに適切な権限を設定してください。
   - 直接アクセスを禁止したい場合は全ての対象者条件に対して「禁止」または「未設定（禁止）」となるように権限を設定してください。
   - ポートレットモジュールの画面と通常の画面を共有して使用する場合は、通常の画面として利用するユーザのみが直接アクセスできる必要があります。
     直接アクセスさせたいユーザの対象条件に対して「許可」となるように権限を設定してください。

JSP/サーブレット、SA Strutsに関してもそれぞれ同様の手順で設定することで、直接アクセスを制御する事ができます。

Client Side Java Script API

ポートレット開発において利用可能な Client Side Java Script API について解説します。

ポートレットの初期処理を登録する

ポートレットが配置された後に、処理を実行する方法を説明します。
以下のAPIを利用することで、ポータルがすべてのポートレットを配置後に、各ポートレットの初期処理を随時呼び出します。

```
<script type="text/javascript">

 imPortal.ready('ポートレットID',function(){

   // ここにコードを記述

 });

</script>
```

コラム

ポートレットIDは以下のAPIで取得できます。

- スクリプト開発

  PortalManagerのgetPortletId()

- JavaEE開発

  jp.co.intra_mart.foundation.portal.common.PortalManagerのgetPortletId()

ポートレットの高さ変更に追随する

2015 Spring(Juno) 以降では、ユーザがポートレットの高さを変更できます。
ユーザによってポートレットの高さが変更された場合にポートレットのコンテンツのサイズ調整を行う方法を説明します。

```
<script type="text/javascript">

 imPortal.addUpdatedSizeListener('ポートレットID',function(widget,width,height){

   if(height > 0) {

    // ユーザによってポートレットの高さが変更された場合は、heightにはコンテンツエリアの高さが渡されます。
    // heightの値を利用してheight内に収まるようにコンテンツの高さを指定してください。
    // なお、コンテンツの高さがheightの値を超えるような調整を行った場合は、縦スクロールバーが表示されます。

   }
   else {

    // ポートレットの高さが自動調整モードの場合は、heightには0が渡されます。
    // ポートレットのコンテンツのデフォルトの高さに調整してください。

   }
 });

</script>
```

コラム

ポートレットIDは以下のAPIで取得できます。

- スクリプト開発

  PortalManagerのgetPortletId()

- JavaEE開発

  jp.co.intra_mart.foundation.portal.common.PortalManagerのgetPortletId()

動的にポートレットのコンテンツが変更された場合にポートレット表示を調整する

ポートレットのイベント(リンクやボタンのクリック)などでAJAXを利用して動的にコンテンツが書き換わる場合に、ポートレット表示（高さなど）を調整する方法について説明します。

Client Side Java Scriptで動的にコンテンツを更新した場合は、 **onResizePortal()** メソッドを利用して、ポートレット表示を調整します。

```
<script type="text/javascript">

 function update_xxxxxx(){

   // 動的にコンテンツを書き換える処理
   ......

   // ポートレット表示の調整
   onResizePortal();

 });

</script>
```

> 注意
> **imPortal.addUpdatedSizeListener** に設定する関数の内部では、利用しないでください。

注意事項

目次

画面を作成する上での注意事項

ポートレットも Web アプリケーションですので、基本的には通常の開発方法と変わりません。
しかし、結果として表示されるポートレットの画面はポータル画面の一部として表示されるため、画面を作成する上で以下のような制約事項があります。

1. ポートレットは HTML の一部分として表示されるため、HTML 構造を表す以下のようなタグは利用できません。
   <BODY>タグ内に表示されるドキュメントの内容のみ記述するようにしてください。
   ブラウザによっては、以下のタグを記述しても表示されますが、推奨はされません。
   - <HEAD>
   - <BODY>
   - <!DOCTYPE>

2. 画面のスタイルはポータル画面で指定されます。
   前項に記述したように<HEAD>タグが利用できないため、スタイルクラスの定義は記述できません。
   タグに個別にスタイルを設定することもできますが、できるだけスタイルに依存しないシンプルな画面作成をしてください。
   - **＜STYLE>タグや<LINK>タグを用いてスタイルを指定することによって、ポータル画面のレイアウトに影響を与えることがあります。**

3. JavaScript で利用するオブジェクト名や変数名、関数名をユニークな名前で定義する必要があります。
   これは通常の画面開発でも同様ですが、ポートレットの場合、ポータル画面上の全てのポートレットでバッティングしないようにする必要があるため、特に注意が必要です。
   また、公開された共通の JavaScript を利用する場合も、そのバージョンが一致しないと思わぬ動作をする可能性があるため、注意する必要があります。
   ポータル画面では、以下の共通ライブラリを利用しているため、ポートレットで利用する場合は、同じバージョンのライブラリを利用するようにしてください。
   - prototype.js v1.6.0.3
   - script.aculo.us（builder.js, effect.js, dragdrop.js, slider.js) v1.8.2

4. タグの ID 属性やコントロールの NAME 属性をユニークな名前で定義する必要があります。
   全ての ID にアプリケーションに割り当てられるプリフィックスをつけるなどしてください。
   また、以下のプリフィックスはシステムおよび intra-mart が提供するアプリケーションが使用するため、利用できません。
   - ‘IM_****’
   - ‘_’（アンダースコア）

5. ポータル画面には複数のページが表示されるため、処理に時間のかかるページを作成すると、ポータル画面が表

示されるまでにそれだけ時間がかかってしまいます。
できるだけシンプルな画面を作成することをお勧めします。

6. ポータル画面では、ポートレットを最大化した場合コンテンツの高さをポートレットに合わせて最大化する処理を行っています。
   しかし、コンテンツが複数の要素から構成されている場合、どの要素を最大化するかの制御は行えないため、意図しない結果となる場合があります。
   そのような場合、コンテンツの全てを 要素で囲うなどして、要素が 1 つだけとなるように作成してください。

7. ポータルの画面においてprototype.jsを利用しているため、CSJSの変数$をjQueryとして利用できません。
   ポートレットで作成された画面でCSJSを利用する場合、CSJSの変数$は利用しないようにしてください。

   jQueryを利用するためには、以下をコードを参照して記述してください。

```
<script type="text/javascript">

  jQuery('#aaa').xxx; // $を使用しないで jQuery を使う。

  jQuery(document).ready(function($){
    // ここでは、$はjQueryとして使えます。
  });

</script>
```

8. ポータル画面には同一のポートレットを複数配置することが出来ますが、ポートレットでCSJSを利用し、固定的な名前の関数を宣言した場合、正常に動作しなくなります。（関数名の重複）

   ポートレットで関数名を動的に生成するようにしてください。

```
<script type="text/javascript">

  jQuery(document).ready(function($){
    func_{ユニークなID}();          // 画面作成毎に{ユニークなID}を生成する。
  });

  function func_{ユニークなID}() {    // 画面作成毎に{ユニークなID}を生成する。

  }

</script>
```

9. ポータル画面には同一のポートレットを複数配置することが出来ますが、ポートレットでHTMLタグのid属性に固定値を設定した場合、正常に動作しなくなります。（id属性値の重複）

   ポートレットでid属性値を動的に生成するようにしてください。

```
<script type="text/javascript">

  function submit_{ユニークなID}() {         // 画面作成毎に{ユニークなID}を生成する。
    jQuery('#form_{ユニークなID}').submit(); // 画面作成毎に{ユニークなID}を生成する。
  }

</script>
<form id="form_{ユニークなID}" >              // 画面作成毎に{ユニークなID}を生成する。
</form >
```