intra-mart WebPlatform/AppFramework Ver.7.2

ポートレット　プログラミングガイド

2010/05/31 第 2 版

＜＜　変更履歴　＞＞

変更年月日	変更内容
2010/04/01	初版
2010/05/31	第 2 版 ポートレットの最大化に関する注意書きを追記しました。(7.1)

＜＜　目次　＞＞

1 はじめに

1.1 目的

このドキュメントは、intra-mart 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。

intra-mart WebPlatform/AppFramework には、ポートレットのサンプルが付属しています。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下さい。

2 概要

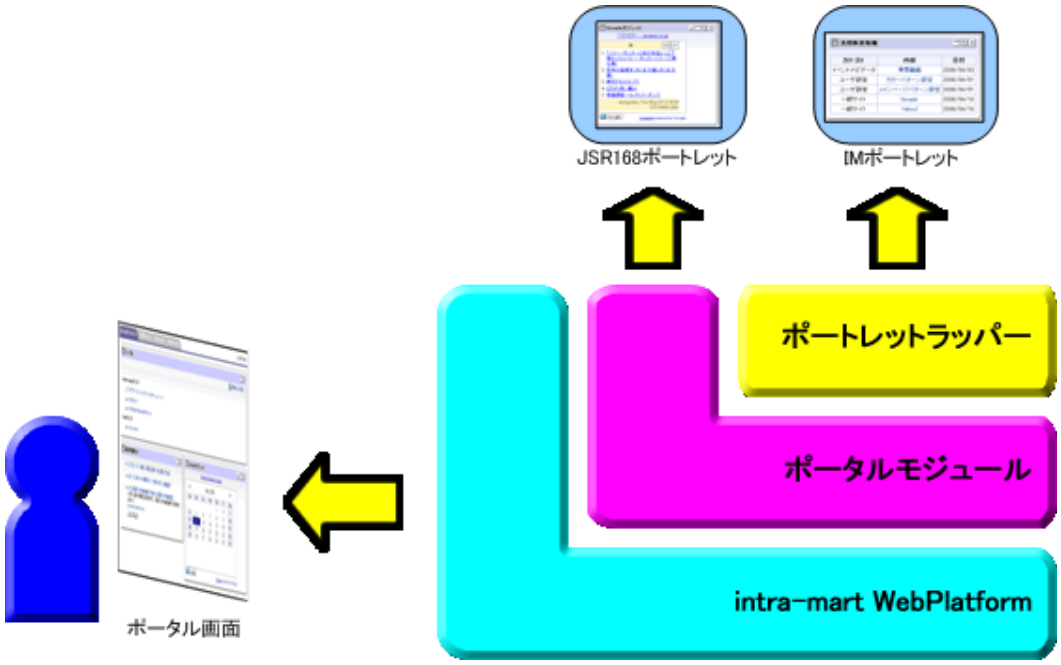
2.1 intra-martポータルモジュールについて

ポータルモジュールは、JSR168 および JSR286 で規定されるポートレットコンテナとして動作します。
このため、JSR168 および JSR286 に準拠したポートレットであれば、そのまま利用することができます。

- JSR168
  Java 標準化団体（Java Community Process）にて策定された、ポートレットに関する標準仕様 ver.1.0。
  ポートレットおよびポートレットコンテナとして必要な機能の定義と、ポートレットを作成するための API（Portlet API 1.0）が含まれます。
- JSR286
  Java 標準化団体（Java Community Process）にて策定された、ポートレットに関する標準仕様 ver.2.0。
  JSR168 に対する追加機能の仕様と API（Portlet API 2.0）を定義します。JSR168 に準拠するポートレットとの互換性を維持する形で定義されています。。

また、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。

本ドキュメントでは、JCP により定義されたポートレットと intra-mart アプリケーションで作成されたポートレットを区別するために、前者を Java ポートレットと呼称します。



＜intra-mart ポータルモジュール構成＞

2.2 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大きくわけて2つに分類されます。

1. intra-mart アプリケーションとして作成されたポートレット
2. Java ポートレット

また、前者の場合開発モデルによってさらに3つに分類されます。

＜ポートレットの種類＞

	ポートレットの種類	開発モデル
1	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
2		JavaEE 開発モデル
3		JSP/サーブレット
4	Java ポートレット	JavaEE Web アプリケーション

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。

2.2.1 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。

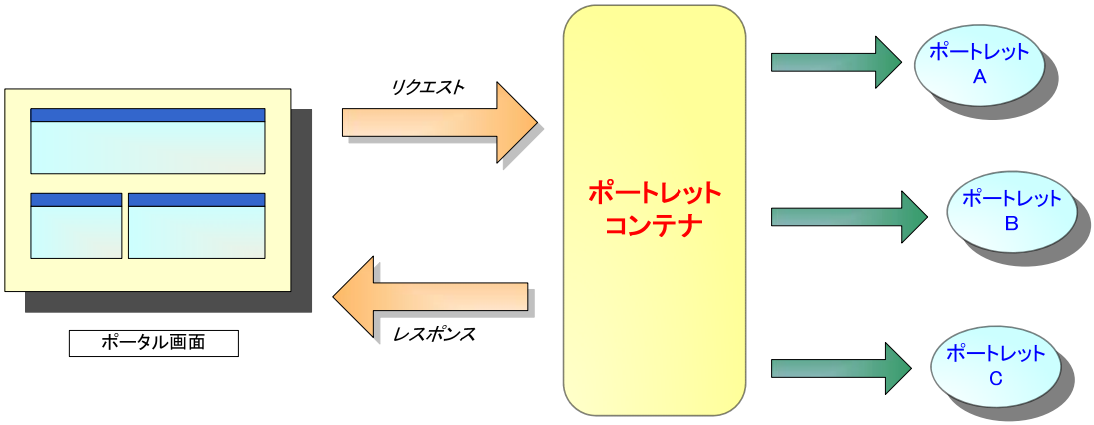
- ポートレットはポータル画面からINCLUDEして呼び出されるため、ポートレット処理中にFORWARD処理することができません。
  Struts では、特に処理を記述しない限り自動的に FORWARD されるため、必ず Action クラスの処理内で RequestDispatcher#include()を呼び出すようにしてください。
- その他、struts-config.xml 内での FORWARD 処理も利用できませんので注意してください。
- また、ServletFilter のターゲットは FORWARD のみですので、もし Filter を利用する必要があるばあい、INCLUDE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

2.3 Javaポートレットのライフサイクル

Java 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。

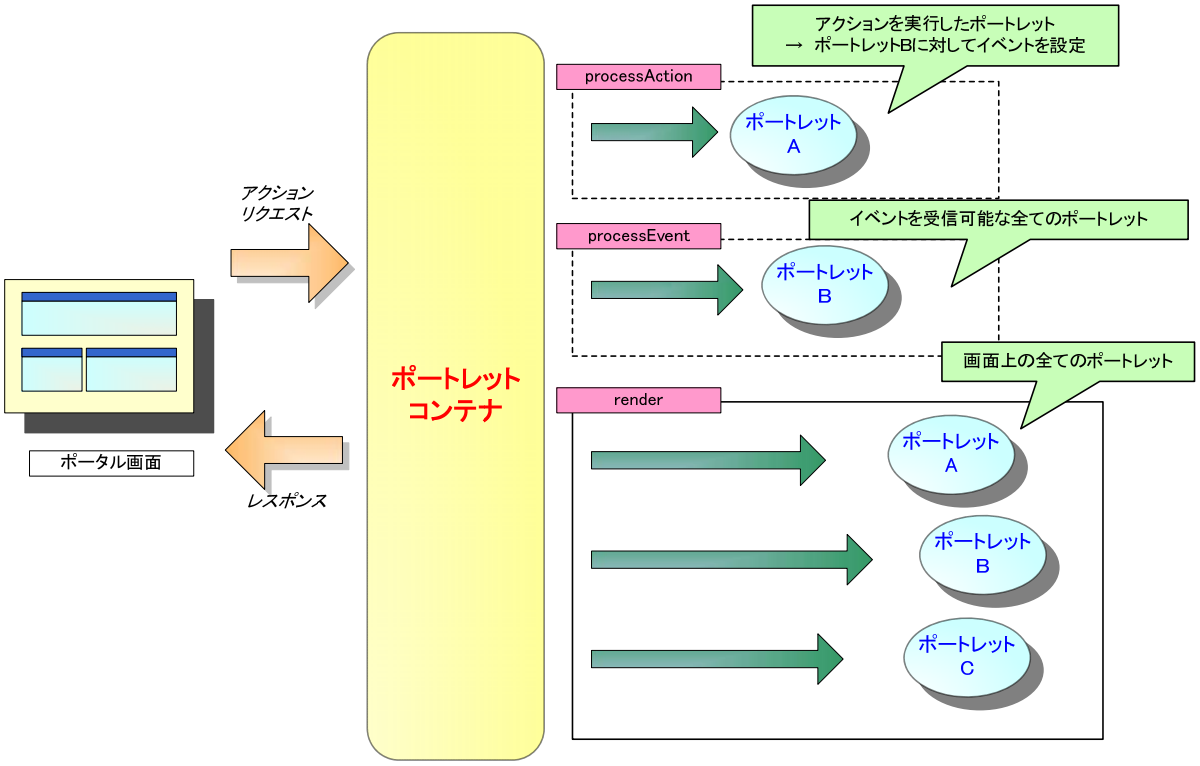
- processAction ・・・ ポートレットに対する処理の実行メソッド
- processEvent ・・・ processAction から派生したイベントの受信処理メソッド
- render ・・・ ポートレット画面の描画処理

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。

2.4 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
- HELP 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。

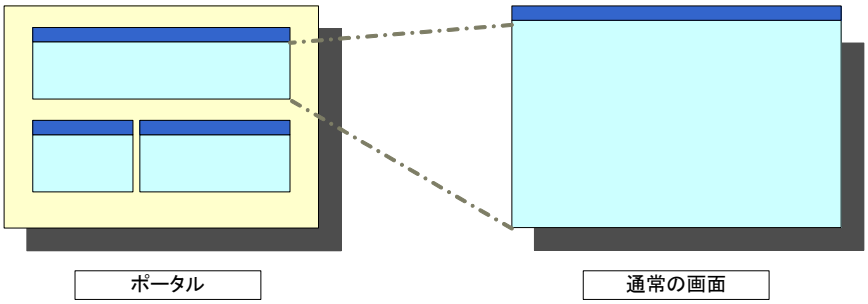
2.5 ウィンドウステータス

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。

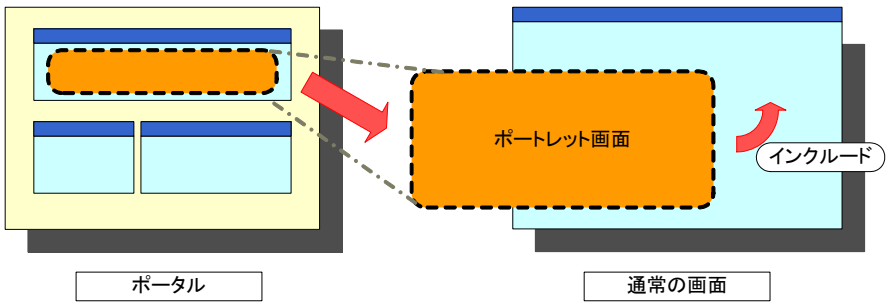
2.6 ポートレット用画面

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「7.1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。項「3.5.3」(スクリプト開発モデルの場合)または項「4.5.3」(JavaEE開発モデルの場合)を参照してください。

3 ポートレットの開発（スクリプト開発編）

3.1 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。スクリプト開発モデルでは、いくつかの制限がありますが、JSR168、JSR286 で定義された機能を利用することが可能です。
また、ポートレットを作成するためには、「2.3 Java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。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し、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。

3.2 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。

3.2.1 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。

3.3 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は、ページ種別が「Presentation Page」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、「Presentation Page」ポートレットの初期状態では、表示モード以外は利用できないようになっています。

システムの設定を変更することによって、編集モード、ヘルプモードの利用が可能となります。
ただし設定を行うことによって、ページ種別が「Presentation Page」のポートレットでは常に同じ設定となり、設定した全てのモードが利用可能となりますので、注意してください。
編集モードが実装されていないポートレットでは、表示モードと同じ動作となりますので、問題は発生しませんが、できる限り、ポートレット編集画面でユーザ利用可否フラグのチェックをはずしておくことをお勧めします。

3.3.1 ポートレットモードの設定

1. portlet.xml 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。Application Runtime を停止して設定後、再起動を行います。

<設定ファイル>

```
<% Application Runtime のルート %>/doc/imart/WEB-INF/portlet.xml
```

<設定箇所>

```
    :
<portlet>
    <description>プレゼンテーションページポートレット</description>
    <portlet-name>PresentationPagePortlet</portlet-name>
    <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> <!-- ← 編集モード追加 -->
        <portlet-mode>HELP</portlet-mode> <!-- ← ヘルプモード追加 -->
    </supports>
    :
<portlet>
    :
```

2. IM ポートレットの初期化
   - 1. の設定後、システム管理者でログインし、「ポートレット管理」 - 「標準ポートレット初期化」を実行します。

3.3.2 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();
```

3.4 ウィンドウステータス

3.4.1 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。

- Ex) normal, maximized, minimized

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。

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

3.5 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ページ引数も、同様に init 関数の request オブジェクトより取得することができます。

3.5.1 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することができます。

- **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(requet) {
  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。

3.5.1.1 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
  - global ・・・ グローバルポータル

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。

3.5.2 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 を利用して取得することができます。

◆ **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 -->
<!-- ウインドウを target 属性で指定 -->
<form name="sampleActionForm" action="<IMART type="string" value=actionURL></IMART>"
	target="IM_MAIN" 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-portal_display.service」という JavaEE フレームワークのサービス ID 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 にサブミットする意外にありません。
  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。

3.5.3 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>
```

3.6 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、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。

3.6.1 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し、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. 「3.5 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>

3.6.2 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することができます。
詳細は、「ポータル API リスト」を参照してください。

3.6.3 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;
```

3.6.4 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);
```

3.6.5 利用可能なActionハンドラ

intra-mart WebPlatform/AppFramework 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、スクリプト開発で利用することが可能です。

3.6.5.1 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。

3.6.5.2 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。

3.7 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、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。

3.7.1 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し、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. 「3.5 画面開発（renderサイクル）」で作成した画面と「3.6 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>

3.7.2 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することができます。
詳細は、「ポータル API リスト」を参照してください。

3.7.3 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

3.7.4 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。

- JSR286 の仕様では、RenderParameter は自動的に引き継がれません。これは、intra-mart ポートレットコンテナ独自の仕様ですので、注意してください。

(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.getParamter(eventId);
```

3.7.5 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);
```

3.8 汎用新着ポートレットについて

intra-mart ポータルモジュールには、汎用的な新着情報を表示するポートレットが用意されています。 この汎用新着ポートレットでは、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)

汎用新着ポートレットは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ーフェースが用意されています。
汎用新着ポートレットに独自の新着情報を表示させるためには、下記の手順でポートレットを登録します。

1. 新着情報を取得するプロバイダを実装する。

汎用新着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	JavaScript の型	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	JavaScript の型	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

<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、Storage Service 配下の任意の場所に設定ファイルを XML 形式で作成して定義します。 設定ファイルには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することが可能です。

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。

<sample_portlet.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。
設定ファイルについての詳細な定義については、「intra-mart WebPlatform/AppFramework ポータル設定ガイド」を参照して下さい。

3. ポートレットを登録する。

グループ管理者の[ポータル] - [ポートレット]編集画面で、新着情報を表示するためのポートレットを登録します。
汎用新着ポートレットを設定する場合は、「ページ種別」、「画面パスあるいは URL 種類」、「ページ引数」の設定が必要になります。たとえば、サンプルでは、以下のように設定されます。

- ページ種別 : Servlet/JSP
- 画面パスあるいは URL 種類 : /portal/portlets/newly_arrived_view.jsp
- ページ引数 : page_param=/portal/sample_portlet.xml
- Action ハンドラ（オプション） :
  jp.co.intra_mart.foundation.portal.general_purpose_portlet.handler.NewArrivedPortletHandler

「画面パスあるいは URL 種類」で指定された JSP には、プロバイダが取得した新着情報を表示するための JSP などを指定します。サンプルで指定している JSP は、ポータルモジュールで用意されたデフォルトの実装です。
「ページ引数」には、手順2で作成した、プロバイダの情報を定義した設定ファイルのパスを page_param= ... の形式で指定します。page_param の値には、Storage Service のルートディレクトリ配下のパスを指定します。

3.8.1 編集モードの利用について

汎用新着ポートレットは、ページ種別が「JSP/Servlet」のポートレットとなっていて、標準で編集モードが有効となっています。
ただし、実際に編集モードを利用するためには、ポートレット登録時に前項で指定した Action ハンドラを設定する必要があります。

編集モードに切り替えると、以下のような画面が表示されます。
表示モードのソート項目と昇順／降順を切り替えるだけの機能で、通常このような機能は表示モードだけでも実現可能ですが、この画面は編集モードのサンプルとしての意味合いも含めて組み込まれています。
実装は JSP/Java を利用しており Java に関する知識が必要ですが、スクリプト開発モデルを利用しても、同等の処理を行うことが可能です。

<汎用新着情報ポートレットの編集モード>

この機能は主に以下のようなモジュールで構成されています。

- モード切替モジュール ・・・ 登録できるページパスは1つだけですので、モードを判定し、切り替える処理が必要です。
- 表示モード用表示モジュール ・・・ 表示モードでの画面表示を行います。
- 編集モード用表示モジュール ・・・ 編集モードでの画面表示を行います。
- 設定用 Action ハンドラ ・・・ 設定ボタンを押下した場合の処理を行います。データの登録処理の他、登録処理後のモードの切り替えも行います。

3.8.2 「重要なお知らせ」ポートレットについて

「重要なお知らせ」ポートレットは、独自に作成された JSR286 対応のポートレットアプリケーションですが、画面表示処理は汎用新着ポートレットの仕組みを使って実装されています。
現在はJavaEE開発モデルで作成された以下のプロバイダクラスが使用され、システム管理者による連絡事項とパスワード履歴管理の情報が表示されます。

- jp.co.intra_mart.foundation.security.password.PasswordHistoryNewArrivedProvider
- jp.co.intra_mart.foundation.portal.portlets.system_notice.SystemNoticeProvider

重要なお知らせに表示する情報の定義は、以下のファイルで行われています。

- <Storage Service のルートディレクトリ>/portal/system_notice.xml

JavaEE 開発モデルとスクリプト開発モデルのプロバイダも混在可能ですので、スクリプト開発モデルでプロバイダクラスを実装して設定ファイルへ定義を追加すれば、上記以外の情報も「重要なお知らせ」として扱うことができるようになります。

<重要なお知らせポートレット定義ファイル：system_notice.xml>

```
<new-arrived-portlet>
  <sort>true</sort>
  <provider>
    <provider-class>
      jp.co.intra_mart.foundation.security.password.PasswordHistoryNewArrivedProvider
    </provider-class>
  </provider>
  <provider>
    <provider-class>
      jp.co.intra_mart.foundation.portal.portlets.system_notice.SystemNoticeProvider
    </provider-class>
  </provider>
</new-arrived-portlet>
```

4 ポートレットの開発（JavaEE 開発編）

4.1 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。
JavaEE 開発モデルでは、いくつかの制限がありますが、JSR168、JSR286 で定義された機能を利用することが可能です。
また、ポートレットを作成するためには、「2.3 Java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。PortletActionHandler インターフェースを実装した Java クラス。
- processEvent ・・・ 任意。PortletEventHandler インター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し、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。

4.2 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

4.2.1 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。

4.3 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、ページ種別が「im-JavaEE Service Framework」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、「im-JavaEE Service Framework」ポートレットの初期状態では、表示モード以外は利用できないようになっています。

システムの設定を変更することによって、編集モード、ヘルプモードの利用が可能となります。
ただし設定を行うことによって、ページ種別が「im-JavaEE Service Framework」のポートレットでは常に同じ設定となり、設定した全てのモードが利用可能となりますので、注意してください。
編集モードが実装されていないポートレットでは、表示モードと同じ動作となりますので、問題は発生しませんが、できる限り、ポートレット編集画面でユーザ利用可否フラグのチェックをはずしておくことをお勧めします。

4.3.1 ポートレットモードの設定

1. portlet.xml 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。Application Runtime を停止して設定後、再起動を行います。

<設定ファイル>

```
<% Application Runtime のルート %>/doc/imart/WEB-INF/portlet.xml
```

<設定箇所>

```
    :
<portlet>
    <description> JavaEE 開発モデル用ポートレット</description>
    <portlet-name> JavaeeFwPortlet </portlet-name>
    <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> <!-- ← 編集モード追加 -->
        <portlet-mode>HELP</portlet-mode> <!-- ← ヘルプモード追加 -->
    </supports>
    :
<portlet>
    :
```

2. IM ポートレットの初期化
   - 1. の設定後、システム管理者でログインし、「ポートレット管理」 - 「標準ポートレット初期化」を実行します。

4.3.2 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();
```

4.4 ウィンドウステータス

4.4.1 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。

- Ex) WindowState.NORMAL, WindowState.MAXIMIZED, WindowState.MINIMIZED

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。

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

4.5 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。

4.5.1 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。

4.5.1.1 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
  - global ・・・ グローバルポータル

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。

4.5.2 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。

① 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 -->
<!-- ウインドウを target 属性で指定 -->
<form action="<%= actionURL.toString() %>" target="IM_MAIN" 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-portal_display.service」という JavaEE フレームワークのサービス ID 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 にサブミットする意外にありません。
  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。

4.5.3 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>
```

4.6 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 インター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。

4.6.1 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 インター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」インター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. 「4.5 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>

4.6.2 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。

4.6.3 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");
```

4.6.4 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 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();
```

4.6.5 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);
```

4.6.6 利用可能なActionハンドラ

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

4.6.6.1 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。

4.6.6.2 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。

4.7 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 インター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。

4.7.1 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 インター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」インター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. 「4.5 画面開発（renderサイクル）」で作成した画面と「4.6 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>

4.7.2 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

4.7.3 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。

4.7.4 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。

- JSR286 の仕様では、RenderParameter は自動的に引き継がれません。これは、intra-mart ポートレットコンテナ独自の仕様ですので、注意してください。

(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);
```

4.7.5 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 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();
```

4.7.6 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);
```

4.8 汎用新着ポートレットについて

intra-mart ポータルモジュールには、汎用的な新着情報を表示するポートレットが用意されています。 この汎用新着ポートレットでは、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)

汎用新着ポートレットに独自の新着情報を表示させるためには、下記の手順でポートレットを登録します。

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、以下のインターフェースを実装したプロバイダより取得されます。

- 汎用新着情報のプロバイダーのインターフェース
  jp.co.intra_mart.foundation.portal.general_purpose_portlet.provider.NewArrivedProvider

ポータルモジュールでは、汎用新着ポートレットを利用したポートレットとして、重要なお知らせポートレットが用意されており、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汎用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ー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」を参照して下さい。

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、Storage Service 配下の任意の場所に設定ファイルを作成して定義します。 設定ファイルには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。

<sample_portlet.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。
設定ファイルについての詳細な定義については、「intra-mart WebPlatform/AppFramework ポータル設定ガイド」を参照して下さい。

3. ポートレットを登録する。

グループ管理者の[ポータル] - [ポートレット]編集画面で、新着情報を表示するためのポートレットを登録します。
汎用新着ポートレットを設定する場合は、「ページ種別」、「画面パスあるいは URL 種類」、「ページ引数」の設定が必要になります。たとえば、サンプルでは、以下のように設定されます。

- ページ種別 : Servlet/JSP
- 画面パスあるいは URL 種類 : /portal/portlets/newly_arrived_view.jsp
- ページ引数 : page_param=/portal/sample_portet.xml
- Action ハンドラ（オプション） :
  jp.co.intra_mart.foundation.portal.general_purpose_portlet.handler.NewArrivedPortletHandler

「画面パスあるいは URL 種類」で指定された JSP には、プロバイダが取得した新着情報を表示するための JSP などを指定します。サンプルで指定している JSP は、ポータルモジュールで用意されたデフォルトの実装です。
「ページ引数」には、手順2で作成した、プロバイダの情報を定義した設定ファイルのパスを page_param= ... の形式で指定します。para_param の値には、Storage Service のルートディレクトリ配下のパスを指定します。

4.8.1 編集モードの利用について

汎用新着ポートレットは、ページ種別が「JSP/Servlet」のポートレットとなっていて、標準で編集モードが有効となっています。
ただし、実際に編集モードを利用するためには、ポートレット登録時に前項で指定した Action ハンドラを設定する必要があります。

編集モードに切り替えると、以下のような画面が表示されます。
表示モードのソート項目と昇順／降順を切り替えるだけの機能で、通常このような機能は表示モードだけでも実現可能ですが、この画面は編集モードのサンプルとしての意味合いも含めて組み込まれています。

<汎用新着情報ポートレットの編集モード>

この機能は主に以下のようなモジュールで構成されています。

- モード切替モジュール ・・・ 登録できるページパスは1つだけですので、モードを判定し、切り替える処理が必要です。
  - 「portal/portlets/ newly_arrived_view.jsp」
- 表示モード用表示モジュール ・・・ 表示モードでの画面表示を行います。
  - 「portal/portlets/ newly_arrived /newly_arrived_view.jsp」
- 編集モード用表示モジュール ・・・編集モードでの画面表示を行います。
  - 「portal/portlets/ newly_arrived /newly_arrived_edit.jsp」
- 設定用 Action ハンドラ ・・・ 設定ボタンを押下した場合の処理を行います。データの登録処理の他、登録処理後のモードの切り替えも行います。
  - 「jp.co.intra_mart.foundation.portal.general_purpose_portlet.handler.NewArrivedPortletHandler」

これらのソースをポートレット作成時の参考にしてください。

4.8.2 「重要なお知らせ」ポートレットについて

「重要なお知らせ」ポートレットは、独自に作成された JSR286 対応のポートレットアプリケーションですが、画面表示処理は汎用新着ポートレットの仕組みを使って実装されています。
現在はJavaEE開発モデルで作成された以下のプロバイダクラスが使用され、システム管理者による連絡事項とパスワード履歴管理の情報が表示されます。

- jp.co.intra_mart.foundation.security.password.PasswordHistoryNewArrivedProvider
- jp.co.intra_mart.foundation.portal.portlets.system_notice.SystemNoticeProvider

重要なお知らせに表示する情報の定義は、以下のファイルで行われています。

- <Storage Service のルートディレクトリ>/portal/system_notice.xml

プロバイダクラスを実装して設定ファイルへ定義を追加すれば、上記以外の情報も「重要なお知らせ」として扱うことができるようになります。

<定義ファイル：system_notice.xml>

```
<new-arrived-portlet>
  <sort>true</sort>
  <provider>
```

```
    <provider-class>
      jp.co.intra_mart.foundation.security.password.PasswordHistoryNewArrivedProvider
    </provider-class>
  </provider>
  <provider>
    <provider-class>
      jp.co.intra_mart.foundation.portal.portlets.system_notice.SystemNoticeProvider
    </provider-class>
  </provider>
</new-arrived-portlet>
```

4.8.3 スクリプト開発モデルによるプロバイダの実装

ポータルモジュールでは、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以下のプロバイダの実装クラスがあらかじめ用意されています。

- jp.co.intra_mart.foundation.portal.general_purpose_portlet.provider.PageBaseCallProvider

JavaEE 開発モデルとスクリプト開発モデルのプロバイダも混在可能ですので、スクリプト開発モデルでプロバイダクラスを実装して設定ファイルへ定義を追加すれば、ポートレットに表示する汎用新着情報を追加することが可能となります。

<スクリプト開発モデルによるプロバイダ定義の例>

```
<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>
      <!-- スクリプト開発モデルで作成した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>
```

5 ポートレットの開発（JSP/Servlet 編）

5.1 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「4.6 アクション処理 （processActionサイクル）」および「4.7 イベント処理 （processEventサイクル）」を参照してください。

6 ポートレットの開発（Java ポートレット編）

6.1 画面開発

Java ポートレットは、サーブレットに類似したモジュールとして定義されており、サーブレットを利用する Web アプリケーションと同様にして作成され、WAR ファイルとしてポータルサーバに配備されて利用されます。
ただし、ポートレットの定義情報は WEB-INF 配下の portlet.xml に記述されます。
それ以外は通常の Web アプリケーションを作成する手順で開発を行いますが、サーブレットではないため、サーブレットを利用する Jakarta Struts のようなフレームワークは利用できません。

以下に、Java ポートレットの作成方法を簡単に記述します。
Java ポートレット（JSR168, JSR286）に関する詳細な情報は専門のドキュメントおよびAPIドキュメント（Portlet API 2.0）を参照してください。

- http://jcp.org/en/jsr/detail?id=168（JSR168 仕様のページ）
- http://jcp.org/en/jsr/detail?id=286（JSR286 仕様のページ）

1. Portlet クラスの作成

Java ポートレットは、javax.portlet.Portlet インターフェースを実装する必要があります。
ただし PortletAPI では、ポートレットを簡単に作成するために基本的な処理を実装した GenericPortlet クラスが用意されています。

```
package sample.portlet;
import javax.portlet.GeneralPortlet;

// GenericPortlet を継承する。
public class SamplePortlet extends GenericPortlet {

  // GenericPortlet ではモードごとに表示処理メソッドが呼ばれる。
  // VIEW モードでの処理を記述する。
  protected void doView(RenderRequest request, RenderResponse response)

      // VIEW モードでの処理を記述する。

    // JSP を呼び出す。
    // ServletContext の代わりに、PortletContext を利用する。
    PortletContext context = getPortletContext();
    // RequestDispatcher の代わりに、PortletRequestDispatcher を利用する。
    PortletRequestDispatcher rd = context.getRequestDispatcher("/WEB-INF/jsp/sample_viewjsp");
    // ポートレットでは、forward メソッドは利用できない。Include のみ利用可能。
    rd.include(request, response);
  }

  // EDIT モードでの処理を記述する。
  protected void doEdit(RenderRequest request, RenderResponse response)

      // EDIT モードでの処理を記述する。
  }

  // HELP モードでの処理を記述する。
  protected void doHelp(RenderRequest request, RenderResponse response)
```

```
        // HELP モードでの処理を記述する。
    }

    // サブミット時の処理を記述する。
    public void processAction(ActionRequest request, ActionResponse response)

        // サブミット時の処理を記述する。
        // この後、表示メソッドが自動的に呼ばれるため、JSP の呼び出しは不要。
    }

}
```

2. JSP の作成

```
// portlet タグライブラリが利用できます。
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>

<%
    // ポートレットでは、HttpServletRequest、HttpServletResponse の代わりに、
    // RenderRequest、RenderResponse を利用します。
    // 使い方はほぼ同じです。
    RenderRequest rRequest = (RenderRequest)request.getAttribute("javax.portlet.request");
    RenderRespnse rResponse = (RenderResponse)request.getAttribute("javax.portlet.response");

    //リクエストからパラメータを取得する。
    String param = rRequest.getParameter("param");


    // ポータル画面を再表示するための URL を取得します。
    // ポータル画面を再表示するための URL（Render URL）の取得。
    PortletURL renderURL = rResponse.createRenderURL();

    // サブミット処理を行ってから、ポータル画面を再表示するための URL（Action URL）の取得。
    PortletURL actionURL = rResponse.createActionURL();
%>

<!-- 何らかの登録処理を行うサービスを呼び出す -->
<!-- Action URL を設定 -->
<!-- ウインドウを target 属性で指定 -->
<form action="<%= actionURL.toString() %>" name="sample_portlet_form" target="IM_MAIN" method="POST">

    <!-- 登録処理で使用する値 -->
    <input type="text" name="param" value="">

    <input type="submit" value="実行">
</form>
```

- 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。

3. portlet.xml の作成

portlet.xml には、ポートレットの定義情報を記述します。
詳しい記述方法については、以下より JSR168 または JSR286 のドキュメントを取得して参照してください。

- http://jcp.org/en/jsr/detail?id=168
- http://jcp.org/en/jsr/detail?id=286

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app
    xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
    http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
    version="2.0">

    <portlet>
        <portlet-name>SamplePortlet</portlet-name>
        <portlet-class>sample.SamplePortlet</portlet-class>
        <expiration-cache>300</expiration-cache>
        <supports>
            <mime-type>text/html</mime-type>
        </supports>
        <supported-locale>ja</supported-locale>
        <portlet-info>
            <title>Sample Portlet</title>
        </portlet-info>
    </portlet>

</portlet-app>
```

<JSR286 の portlet.xml>

4. web.xml の作成

web.xml の記述は必要ありませんが、Web アプリケーションとして AP サーバにデプロイするため、空のファイルを格納します。
追加情報を記述することも可能ですが、その場合、サーブレット API によって web.xml の記述が異なります。
バージョン情報を取得できるようにバージョンの定義情報は省略しないで下さい。

- Servlet2.3　：DOCTYPE による指定
- Servlet2.4～　：XMLSchema による指定

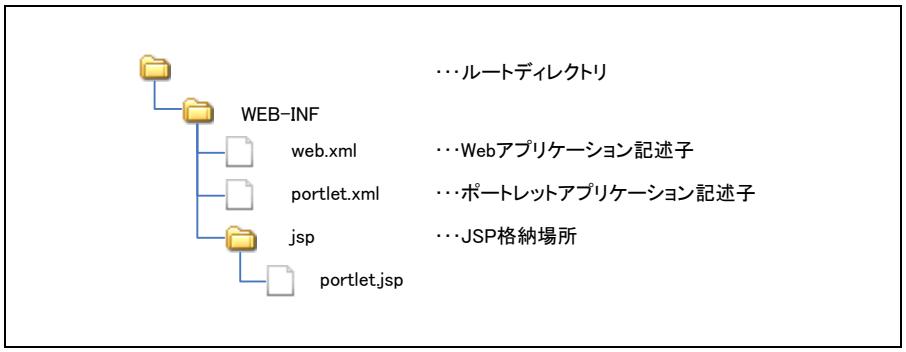
```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Servlet 2.4 の場合 -->
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
        version="2.4">

</web-app>
```

5. WAR ファイルの作成

作成したファイルをまとめて、WAR ファイルを作成してポートレットアプリケーションが完成します。
WEB-INF の中には、web.xml、portlet.xml の2つのデプロイメント記述子が格納されます。
また、Java ポートレットでは、ポートレットを直接呼び出してはいけないと定義されていますが、AP サーバに配置した場合は JSP に直接アクセスできてしまいます。
そこで、ポートレットアプリケーションでは通常 WEB-INF 配下に JSP を格納します。



＜WARファイルの構造＞

6.2 アクション処理（processActionサイクル）

前項では、画面表示処理に絞ってポートレットの作成方法を説明しました。
ここでは、processAction サイクルでできることについていくつか説明していきます。
以下に processAction のサンプルを記載します。

(ex.) processAction のサンプル

```
// 「PortletActionHandler」インターフェースを実装する。
public class SamplePortlet extends GenericPortlet {

 …

  public void processAction(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
  // portlet.xml で指定した QName をキーとして利用する。
  response.setEvent(new QName("sampleNamespace", "SampleEvent"), value);

 }
}
```

6.2.1 ActionRequest, ActionResponse

processAction の引数は、このメソッド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。

6.2.2 イベントの設定

上で示したサンプルコードでは、ActionResponse にイベントを設定していますが、イベントを設定できるかどうか、またどのようなイベントを設定できるかは、portlet.xml に記述する必要があります。

(ex.) 送信イベント定義のサンプル

```
<portlet-app
    xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
    http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
    version="2.0">

    <portlet>
        <portlet-name>SamplePortlet</portlet-name>
        <portlet-class>sample.SamplePortlet</portlet-class>
        <expiration-cache>300</expiration-cache>
        <supports>
            <mime-type>text/html</mime-type>
        </supports>
        <supported-locale>ja</supported-locale>
        <portlet-info>
            <title>Sample Portlet</title>
        </portlet-info>
        <supported-publishing-event>
            <!-- event-definition で定義したイベントのうち、このポートレットで送信可能なイベントの定義 -->
            <qname xmlns:im_portal="urn: sampel:event">
              sample:SampleEvent
            </qname>
        </supported-publishing-event>
    </portlet>

  <event-definition>
     <!-- 利用可能なイベントの定義 -->
     <qname xmlns:sample='urn:sampel:event'>sample:SampleEvent</qname>
     <!--イベントオブジェクトの型指定 -->
     <value-type>java.lang.String</value-type>
  </event-definition>

</portlet-app>
```

6.3 イベント処理（processEventサイクル）

この項では、processEvent サイクルの処理を説明します。
以下に processEvent のサンプルを記載します。

(ex.) processEvent のサンプル

```
// 「PortletEventHandler」インターフェースを実装する。
public class SamplePortlet extends GenericPortlet {

 ～

  public void processEvent(EventRequest request, EventResponse response) {

   // イベントオブジェクトを取得
   // イベントオブジェクトの型は portlet.xml で指定した型になります。
   String event = (String) request.getEvent()

   // 画面表示に利用するリクエストパラメータを設定
   response.setRenderParameter("param1", event);
 }
}
```

6.3.1 EventRequest, EventResponse

processEvent の引数は、このメソッド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。

6.3.2 イベントの設定

上で示したサンプルコードでは、EventResponse からイベントを取得していますが、どのようなイベントを受信できるかは、portlet.xml に記述する必要があります。

(ex.) 受信イベント定義のサンプル

```
<portlet-app
    xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
    http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
    version="2.0">

    <portlet>
        <portlet-name>SamplePortlet</portlet-name>
        <portlet-class>sample.SamplePortlet</portlet-class>
        <expiration-cache>300</expiration-cache>
        <supports>
            <mime-type>text/html</mime-type>
        </supports>
        <supported-locale>ja</supported-locale>
        <portlet-info>
            <title>Sample Portlet</title>
        </portlet-info>
        <supported-processing-event>
            <!-- event-definition で定義したイベントのうち、このポートレットで受信可能なイベントの定義 -->
            <qname xmlns:im_portal="urn:sample:event">
                sample:SampleEvent
            </qname>
        </supported-processing-event>
    </portlet>

    <event-definition>
        <!-- 利用可能なイベントの定義 -->
        <qname xmlns:sample='urn:sampel:event'>sample:SampleEvent</qname>
        <!--イベントオブジェクトの型指定 -->
        <value-type>java.lang.String</value-type>
    </event-definition>

</portlet-app>
```

7 注意事項

7.1 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下さい。
   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下さい。
   - ＜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。
   - <IMART type="imDesignCss"> / < imarttag:imartDesignCss>タグを利用した場合、動的にカラーパターンが変更されますが、ポートレットのキャッシュ機能を有効にした場合、カラーパターンの変更が反映されなくなります。
     これらのタグはポータル画面で指定しており、再指定しなくても利用可能ですので、ポートレットには記述しないようにしてください。

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下さい。
   - 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下さい。
   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。

8 サンプル

8.1 サンプルについて

intra-mart WebPlatform/AppFramework には、ポートレットに特有の機能である、アクション処理、イベント処理の使い方を説明するサンプルを添付しています。

このサンプルでは、Action ハンドラと Event ハンドラの処理が確認できます。
「イベント送信」ポートレットで送信先を指定し、メッセージが発生することを確認してください。
また、ポートレット編集画面で、Action ハンドラ／Event ハンドラの設定なども確認してみてください。

8.2 サンプルの設定

サンプルを有効にしてインストールを行うと、サンプルプログラムおよびデータがインストールされます。
ポータルのサンプルを利用するためには、さらにポータルデータのインポートが必要です。
ログイングループ管理者でログインし、管理メニューより「ポータル」－「インポート・エクスポート」を選択し、以下のサンプルデータのインポートを行います。

- sample/portal/portal_event_sample.zip（サーバの文字コードが Windows-31J の場合）
- sample/portal/portal_event_sample_u8.zip（サーバの文字コードが UTF-8 の場合）

サンプルデータのインポートにより、グループポータルに「サンプル」タブが作成されます。
「サンプル」タブには、以下のポートレットが配置されています。

- **im-JavaEE Service Framework ポートレット**
  - イベント送信(JavaEE) ・・・ Action を実行し、イベントを発生させます。
  - イベント受信1(JavaEE) ・・・ 該当のイベントを受信し、メッセージを表示します。
  - イベント受信2(JavaEE) ・・・ 該当のイベントを受信し、メッセージを表示します。

- **Presentation Page ポートレット**
  - イベント送信(JSSP) ・・・ Action を実行し、イベントを発生させます。
  - イベント受信1(JSSP) ・・・ 該当のイベントを受信し、メッセージを表示します。
  - イベント受信2(JSSP) ・・・ 該当のイベントを受信し、メッセージを表示します。

※ このサンプルデータは、ログイングループに依存しているため、ログイングループ ID が「default」以外では、「サンプル」タブは表示されません。もし「サンプル」タブが表示されない場合、以下の手順でサンプルポータルを作成してください。

1. ポータル画面で、「タブの追加」をクリックし、「サンプル」という名称でタブを新規に作成する。
2. コンテキストメニューから「ポートレットの追加」を選択する。
3. 「アプリケーション」で「イベントサンプル」を選択して、検索された全てのポートレットをポータル画面に追加する。

8.3 サンプルのファイル一覧

ファイルパス	説明
<% IM_ROOT %>/doc/imart/sample/portal/event	JavaEE 開発モデル用サンプル JSP ファイル
<% IM_ROOT %>/doc/imart/WEB-INF/classes/sample/portal/event/handler	JavaEE 開発モデル用サンプルハンドラクラス
<% IM_ROOT %>/pages/src/sample/portal/event	スクリプト開発モデル用サンプル JSSP ファイル
<% IM_ROOT %>/storage/sample/portal	「サンプル」ポータル画面用インポートデータ

intra-mart WebPlatform/AppFramework Ver.7.2
ポートレット　プログラミングガイド

2010/05/31 第2版

TEL: 03-5549-2821
FAX: 03-5549-2816
E-MAIL: info@intra-mart.jp
URL: http://www.intra-mart.jp/