intra-mart WebPlatform/AppFramework Ver.7.0

JavaEE 開発モデル プログラミングガイド

2010/11/30 第 4 版

<< 変更履歴 >>

変更年月日	変更内容
2008/07/07	初版
2009/02/27	第 2 版 「2.10 バッチ管理モジュール」誤植を修正 「4.3 JMS(Java Messaging Sservice)」を追加
2009/10/30	第 3 版 「2.8 ショートカットアクセス機能」 有効期限の日付指定サンプルを追加
2010/11/30	第 4 版 「2.2 メール連携モジュール」画像を修正

<< 目次 >>

1 イントロダクション 1
1.1 intra-martのアプリケーション開発概要 1
1.1.1 JavaEE開発モデルによるアプリケーション開発 1
1.2 JavaEE開発モデルで開発するにあたって 4
1.2.1 JavaEEフレームワークプログラミング 4
1.2.2 各ファイルの保存場所 5
2 Javaコンポーネント群（im-BizAPI）の利用 6
2.1 画面共通モジュール 6
2.1.1 タグライブラリ 6
2.1.2 画面デザイン共通モジュール 7
2.1.3 ファイル・アップロード 8
2.1.4 ファイルリストの表示 11
2.1.5 ファイル・ダウンロード 13
2.1.6 ファイルの削除 16
2.2 メール連携モジュール 19
2.2.1 メール送信 19
2.2.2 添付ファイル付きメールの送信 23
2.3 外部プロセスの呼び出し 27
2.4 XML形式のデータを扱う 28
2.4.1 XMLパーサーとデータの取得 28
2.5 スクリプト開発モデルとの連携 28
2.5.1 メニューを介さないでim-JavaEE Frameworkの画面を表示する方法 28
2.5.2 スクリプト開発モデルの画面からフレームワークの画面へ遷移する方法 28
2.6 グラフ描画モジュール 30
2.7 アクセスコントローラモジュール 32
2.8 ショートカットアクセス機能 33
2.9 外部ソフトウェア接続モジュール 35
2.10 バッチ管理モジュール 39
2.11 カレンダー表示モジュール 40
2.11.1 呼び出し方法 40
2.11.2 カレンダーデータの受け取り方法 40
2.11.3 カレンダーマスタメンテナンス 41
2.12 ツリー表示モジュール 42
2.13 アプリケーション・ロック機能 44
2.14 一意情報の取得機能 45
2.15 製品のカスタマイズ 46
2.15.1 規定 46
2.15.2 環境移行の手順 46
2.15.3 注意事項 46
2.16 アクセスセキュリティモジュールを利用しないで画面を構築する方法 47
2.16.1 概要 47
2.16.2 準備(インストール) 47
2.16.3 プログラムの作成 47
2.16.4 注意事項 48
3 サンプルプログラムの実行 49
3.1 サンプルのインストール 49

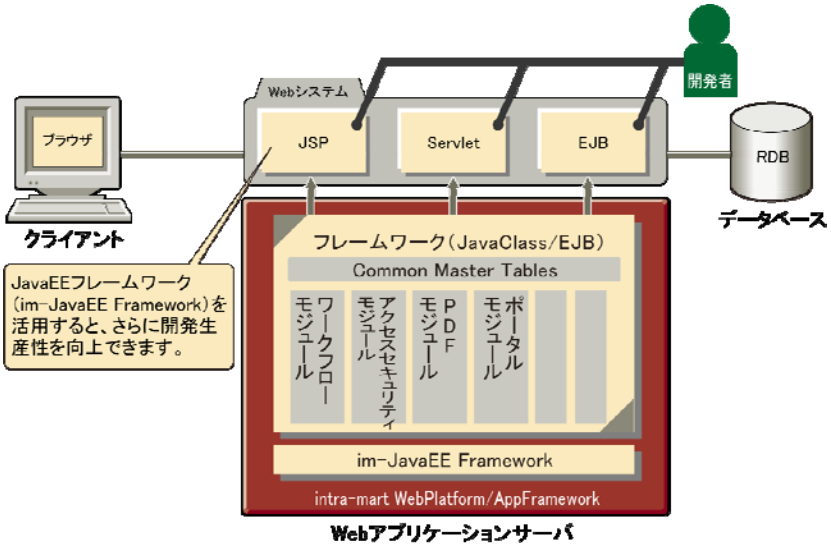
3.2 メニューのサンプル実行 50
3.3 APIリスト 51
4 Appendix 52
4.1 メッセージ設定 52
4.2 制限事項 52
4.2.1 ファイル名称 52
4.2.2 ID、コード 52
4.2.3 JS関数 52
4.3 JMS(Java Messaging Sservice) 53

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

1 イントロダクション

1.1 intra-martのアプリケーション開発概要

JavaEE 開発モデルでは、JSP ファイルと Servlet、EJB コンポーネントで開発します。この際にも、JavaEE フレームワークを利用することで、生産性を大きく向上することができます。

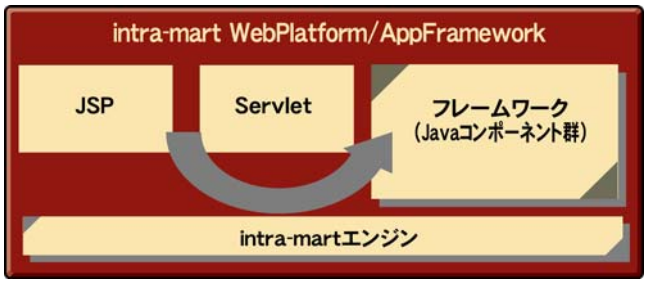


1.1.1 JavaEE開発モデルによるアプリケーション開発

OS や Web アプリケーションサーバに依存しない共有プラットフォームとして、JavaEE による Web システム開発が普及してきました。しかし、JavaEE による開発は Java をベースにしているため、オブジェクト指向などの高度な知識と経験が要求される点や、JavaEE での前提知識が必要になる点など、敷居の高さが問題になってきています。さらに、JavaEE 開発の規約にさえ準拠させれば、あとはいかようにでも組める自由さが、初心者にとってはかえって負担となり、SE によってバラバラな開発スタイルとなってしまう原因ともなります。intra-mart WebPlatform や intra-mart AppFramework では、これらの問題を JavaEE フレームワーク(im-JavaEE Framework)の利用により解決し、JavaEE 開発モデルの生産性を大幅に向上させています。

1.1.1.1 数多くのJavaコンポーネント群(im-BizAPI)

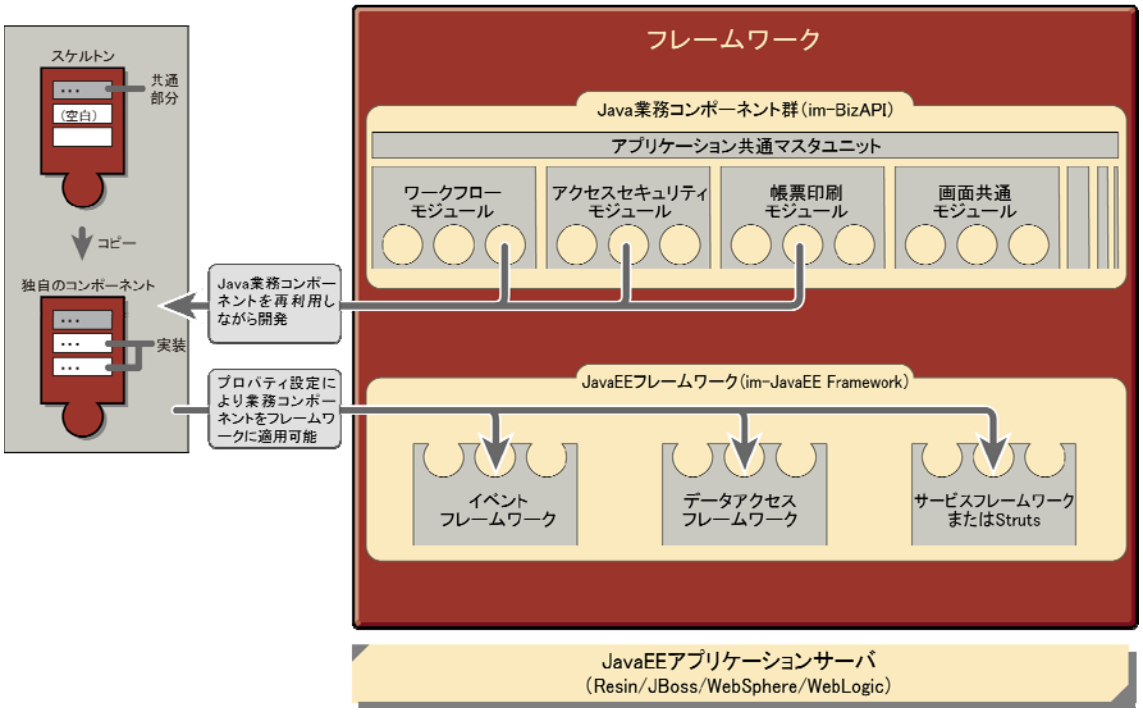
intra-mart WebPlatform や intra-mart AppFramework では、Web システム構築でよく利用される機能を「Java コンポーネント」として数多く提供しており、JavaClass または EJB コンポーネントとして再利用が可能です(アクセスセキュリティモジュールやワークフローモジュール、ポータルモジュールなど)。これにより従来ゼロから開発しなければならなかった複雑な機能を、フレームワークで用意されている Java コンポーネントを利用することで大規模な Web システムを短期間かつ高品質に構築できます。



1.1.1.2 JavaEE開発のフレームワーク

JavaEE での Web システム開発には、構造的に共通な部分が多く、その事実を利用すると、開発生産性をさらに大きく向上させることができます。

intra-mart では、JavaEE 開発で必要になる共通的な処理は、すべてフレームワークとして用意し、開発者に委ねられる箇所はコンポーネントを作成してもらう形態となっています。

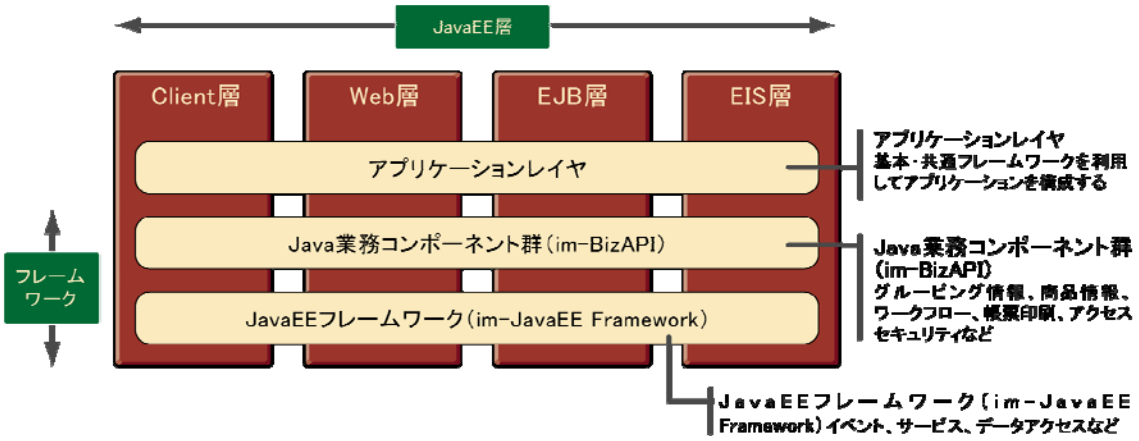


上図のフレームワーク部分にあるように、大きくは「Java 業務コンポーネント群(im-BizAPI)」と「JavaEE フレームワーク(im-JavaEE Framework)」に分かれています。intra-mart WebPlatform の Java コンポーネントモジュール群(ワークフローモジュールやアクセスセキュリティモジュールなど)はすべて「Java 業務コンポーネント群(im-BizAPI)」に配置されます。そして JavaEE 開発で必要になる共通的な処理はすべて「JavaEE フレームワーク(im-JavaEE Framework)」に配置されます。

特に、このフレームワークを利用するコンポーネントについては、雛型となる「スケルトン」を用意しておくことで、さらに生産性を高めることが可能となります(共通部分はあらかじめ実装されているので、開発者は独自のコンポーネントを作成する場合、スケルトンをコピーし、実装されていない箇所を埋めてコンポーネントを作成することになります)。作成されたコンポーネントはプロパティ設定によりフレームワークにあてはめていくことで動作します。

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

フレームワークも JavaEE で定義されている4層（Client 層、Web 層、EJB 層、EIS 層）を使用する共通的な機構となります（Sun の BluePrint に準拠）が、開発者はそのことを一切意識する必要はありません（システム規模に合わせてあとから構成変更することも、設定ファイルの変更のみで可能です）。



1.1.1.3 JavaEE開発モデルにおいてフレームワークを活用した際のメリット

JavaEE 開発時に JavaEE フレームワーク（im-JavaEE Framework）を利用することで、以下のメリットがでてきます。

JavaEE 開発モデルの開発基盤	高度な知識が必要となる部分は隠蔽し、開発者はアプリケーションロジックをスケルトンの中に埋め込みます。前提知識がなくても完成したシステムは JavaEE モデルの推奨型となり、MVC モデルの実現が容易となります（プログラム構造が統一できメンテナンス性向上）。
サーバ構成を意識しなくてもよい	サーバ構成が変更されても、設定ファイルなどにより、外部から調整可能となります。
生産性の向上	共通的なものはすべてフレームワーク中に用意されているので、コンポーネントの再利用性が高まり、生産性および保守性が向上します。
カスタマイズの容易さ	コンポーネントの新規追加時にも元のアプリケーションにはまったく変更が入らない仕組みとなります。また、機能変更時も該当コンポーネントのみの修正となり、他の箇所への影響がありません。
テスト工程期間の短縮	テスト工程でフレームワーク部分の確認が不要となります。これにより、問題発生時の切り分けが可能となります。

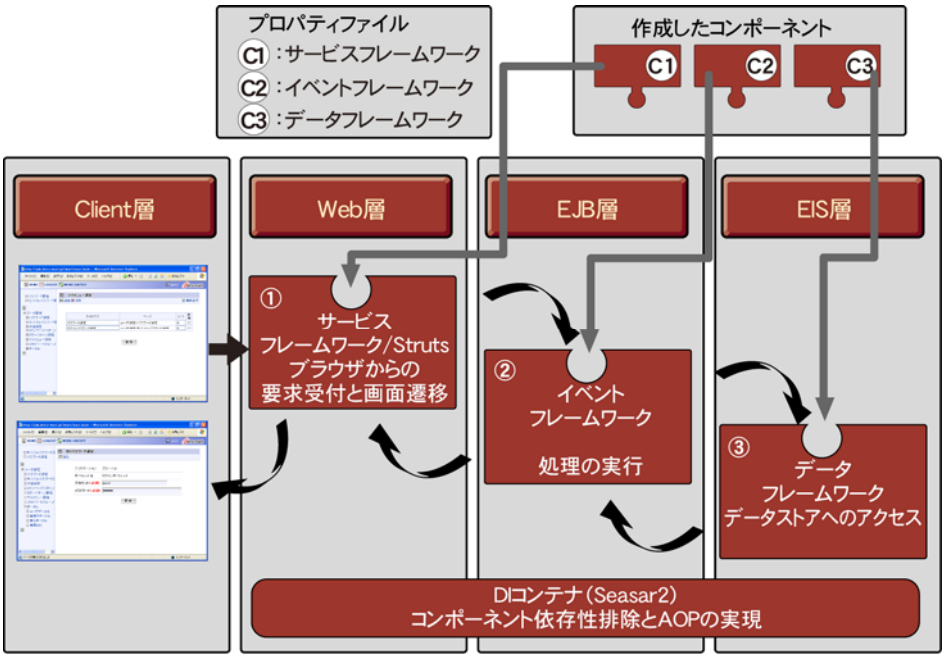
1.2 JavaEE開発モデルで開発するにあたって

JavaEE 開発モデルで開発するにあたっては、本書第 2 章以降で、次のような解説をしています。

1.2.1 JavaEEフレームワークプログラミング

JavaEE での Web システム開発には、構造的に共通な部分が多く、その事実を利用すると、開発生産性をさらに大きく向上させることができます。intra-mart WebPlatform では、JavaEE 開発で必要になる共通的な処理は、すべてフレームワークとして用意し、開発者に委ねられる箇所はコンポーネントを作成してもらう形態となっています。

1. JavaEE 開発モデルによるアプリケーション開発
2. フレームワーク
3. JavaEE フレームワーク（im-JavaEE Framework）
4. アプリケーションレイヤ
5. チュートリアル
6. サンプルプログラム



※ JavaEE フレームワークを初めて利用する方に、よりやさしく解説した「im_JavaEE Framework チュートリアル」が用意されています。第 3 章の前にお読みになると、理解が深まります（インストールした Windows メニューから呼び出せます。本書は PDF 形式のファイルですので、必要に応じて印刷してお使いください）。

※ その他、「im-JavaEE Framework 仕様書」もドキュメント CD に用意されています。合わせてご活用ください。※ Web 層のサービスフレームワークの代わりに、オープンソースのフレームワークである、「struts」が利用できるようになりました。「struts」から EJB 層のイベントフレームワークや EIS 層のデータフレームワークを連携して呼び出すことができます。連携方法については、「intra-mart im-JavaEE Framework Struts 連携ガイド」をご覧ください。

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

1.2.2 各ファイルの保存場所

intra-mart WebPlatform/AppFramework の各ファイルの保存場所を示します。

1.2.2.1 intra-mart WebPlatformの場合

- 静的コンテンツ（HTML ファイルや画像ファイルなど）
  Web サーバコネクタのインストールディレクトリ以下
  （スタンドアロン型の場合はサーバモジュールのインストールディレクトリ直下 doc/ディレクトリ以下）

- JavaEE 開発モデルのプログラム（JSP）（JSP ファイル（.jsp, .xtp））
  %Application Runtime%（スタンドアロンの場合はサーバモジュール）/doc/ディレクトリ以下

- JavaEE 開発モデルのプログラム（Servlet）（JAVA クラスファイル（.class））
  %Application Runtime%（スタンドアロンの場合はサーバモジュール）/doc/ディレクトリ以下の該当ディレクトリ内 WEB-INF/classes/以下（または、クラスパスに設定されているディレクトリ内）

- Storage Service により一元管理されるファイル群
  %Storage Service%/storage/ディレクトリ内

1.2.2.2 intra-mart AppFrameworkの場合

- 静的コンテンツ（HTML ファイルや画像ファイルなど）
  フレームワークサーバのインストールディレクトリ直下 doc/ディレクトリ以下
  （スタンドアロン型の場合はサーバモジュールのインストールディレクトリ直下 doc/ディレクトリ以下）

- JavaEE 開発モデルのプログラム（JSP）（JSP ファイル（.jsp））
  %Application Runtime%（スタンドアロンの場合はサーバモジュール）/doc/ディレクトリ以下

- JavaEE 開発モデルのプログラム（Servlet）（JAVA クラスファイル（.class））
  %Application Runtime%（スタンドアロンの場合はサーバモジュール）/doc/ディレクトリ以下の該当ディレクトリ内 WEB-INF/classes/以下（または、クラスパスに設定されているディレクトリ内）

- Storage Service により一元管理されるファイル群
  %Storage Service%/storage/ディレクトリ内

2 Javaコンポーネント群（im-BizAPI）の利用

2.1 画面共通モジュール

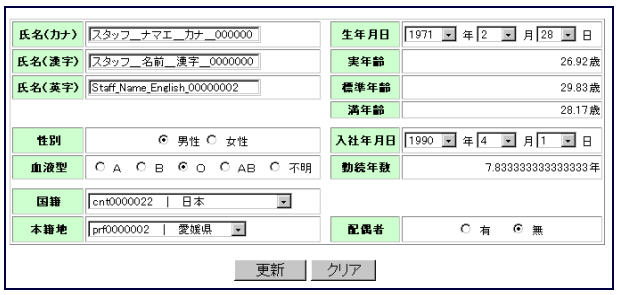
Web ベースでの GUI 開発でよく利用される画面部品のモジュールです。それぞれのモジュールに適当なプロパティを設定して呼び出すだけで、データベースと連動したユーザインタフェースを簡単に作成できます。また、JavaEE 開発モデルには、JSP で利用できる intra-mart タグライブラリを使用しています。タグライブラリの詳細は、次ページの解説「タグライブラリ」を参照してください。

■ 提供される画面共通モジュールの例

一般的な入力コントロール系	ユーザインタフェース構築に必要となる一般的な入力コントロール（テキストフィールド、パスワードボックス、ラジオボタン、チェックボックス、テキストエリア など）を用意しています。これらのコントロール群は、サーバサイドのスクリプトやデータと連動が可能なコントロールとなります。
レイアウト制御モジュール	さまざまな条件により表示すべき値を変化させたり、表示する内容を選択したりと通常はHTMLでは表現できないプログラム的な要素をプレゼンテーションページ内に定義することができます。

■ 構築された Web ユーザインターフェースの例

前述のオブジェクト／関数群を利用して HTML 上で編集していくことで、細かなレベルのユーザインターフェースの構築が可能になり、従来の VisualBasic などによるユーザインタフェースと遜色がないスタイルの Web システムの構築が可能です。画面の作成例は以下のようになっています。



<画面の作成例>

2.1.1 タグライブラリ

画面共通モジュールをJSPで呼び出すには、intra-martタグライブラリを使用します。intra-mart タグライブラリを使用するには、タグライブラリを使用する JSP のページで、以下のような taglib ディレクティブを指定する必要があります。

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

uri 属性には "http://www.intra-mart.co.jp/taglib/core/standard" を指定してください。prefix 属性は、そのページで使用される全てのタグライブラリの前に付加 される XML 名前空間識別子を指定するもので、任意の名前を付けても構いま せん。(以下、この API リストでは、 prefix 属性を "imtag" として説明します)

＜使用例＞

```
<imtag:imartDateFormat value="<%=new Date()%>" format="yyyy/MM/dd"/>
```


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

指定された日付データを文字列として挿入する。

2008/04/24

＜実行画面＞

■ タグライブラリリスト

- ◆ calendar カレンダモジュールの画面 HTML ソースを作成する
- ◆ condition タグに挟まれた部分の実行の条件分岐（真偽）
- ◆ dicision タグに挟まれた部分の実行の条件分岐（値指定）
- ◆ imartDateFormat タグの指定された場所に、指定された日付データを文字列として挿入
- ◆ imartEvent ブラウザ上で動作するイベント処理ロジックを設定
- ◆ imartNumberFormat タグの指定された場所に、指定された数値データを文字列として挿入
- ◆ loop 処理の繰り返し
- ◆ repeat タグ内のネストされた部分の繰り返し処理
- ◆ select コンボボックス（リストボックス）の作成
- ◆ drawer 画像ファイルを作成し HTML に表示する

2.1.2 画面デザイン共通モジュール

タグライブラリには、画面共通モジュールのほかに、画面デザイン共通モジュールがあり、主に表示系に特化したタグ群が用意されています。

intra-mart タグライブラリを使用するには、タグライブラリを使用する JSP のページで、以下のような taglib ディレクティブを指定する必要があります。

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

uri 属性には "http://www.intra-mart.co.jp/taglib/foundation/imarttag" を指定してください。 prefix 属性は、そのページで使用される全てのタグライブラリの前に付加される XML 名前空間識別子を指定するもので、 任意の名前を付けても構いません。(以下、この API リストでは、 prefix 属性を "imarttag" として説明します)

画面デザイン共通モジュールを使用する際は、必ず各画面を構成している HTML の<HEAD>タグ内に、 デザインスタイルシートの宣言（＝<imarttag:imartDesignCss>）を行ってください。

＜使用例＞

```
<imarttag:imartTitleBar  title="新規登録" icon="/images/standard/title.gif" />
```

指定されたタイトルとアイコンのタイトルバーを表示します。

＜実行画面＞

■ Storage Service の利用方法

intra-mart では Storage Service を利用することにより、分散システム構築時においても、容易にファイルの共有が可能となります。

ここでは、ブラウザからアップロードされたファイルを Storage Service に保存したり、また Storage Service に保存されているファイルをダウンロードしたりする画面を作ってみます。

2.1.3 ファイル・アップロード

まずは、ファイルをアップロードして Storage Service に保存するためのフォームを作成します。

＜filebox.jsp＞

```
<%@ page contentType="text/html; charset=windows-31j" pageEncoding="windows-31j" %>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/foundation/imarttag" %>

<%@ page import="jp.co.intra_mart.foundation.service.client.file.*" %>

<%
  NetworkFile file = new NetworkFile("filebox/");
  if(!file.isDirectory()){
    file.makeDirectories();
  }
%>

 <HTML>
  <HEAD>
    <imarttag:imartDesignCss />
    <SCRIPT>

    function checkInputValue(){
      //送信先
      if( document.formUpload.local_file.value == "" ) {
        alert("[ File: ]を指定してください。");
        document.formUpload.local_file.focus();
        return false;
      }

      return true;
    }

    function onUpload(){
      if (checkInputValue() == true ) {
          document.formUpload.submit();
      }
    }
    </SCRIPT>
  </HEAD>
  <BODY>
    <!-- タイトルバー表示 -->
    <imarttag:imartTitleBar title="ファイル・アップ" />
    <!-- ツールバー表示-->
    <imarttag:imartToolbarFrame>
      <imarttag:imartToolbarLeft>
        <imarttag:imartIcon
          name="追加"
          icon="/images/standard/next.gif"
          href="JavaScript:onUpload()" />
      </imarttag:imartToolbarLeft>
    </imarttag:imartToolbarFrame>

    <Form name="formUpload"
          method="POST"
          action="upload.jsp"
          enctype="multipart/form-data"
          onSubmit="return checkInputValue();" >
```


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

```
    <TABLE>
      <TR>
        <TD align="center"><br>
          <TABLE class="edit">
            <TR class="bottom">
              <imarttag:imartItemNameTd name = "File：" require = "true" />
              <imarttag:imartInputTd
                  type = "file"
                  name = "local_file"
                  value = ""
                  size = "47" />
            </TR>
            </TR>
            <TR><TD><BR></TD></TR><!-- スペース用 -->
          </TABLE>
        </TD>
      </TR>
    </TABLE>
  </FORM>
 </BODY>
</HTML>

<!-- End of File -->
```

受信したファイルデータを Storage Service に保存するための JSP を記述します。

※ ここでのサンプルは、理解しやすくするために、JSP ファイル中に呼び出しのコーディングが記述されています。このコーディングは、JavaEE フレームワーク利用時では、サービスコントローラやイベントリスナーの中に記述するなど適切な位置に盛り込んでください。

＜ファイルを受信するための JSP ファイル（upload.jsp）＞

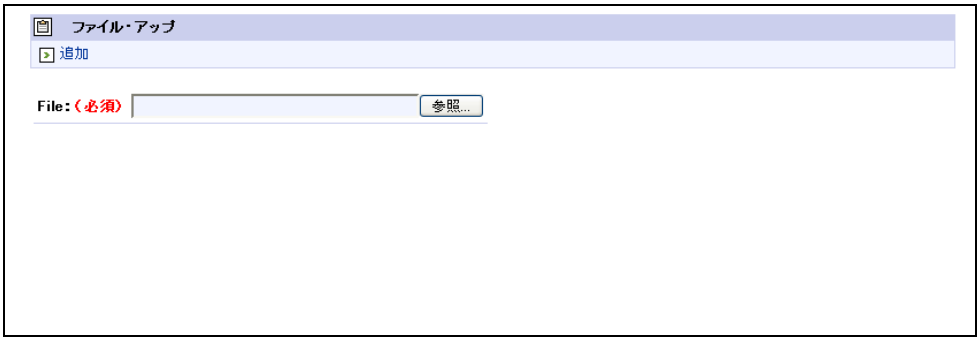
```
<%@ page contentType="text/html; charset=windows-31j" pageEncoding="windows-31j" %>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/foundation/imarttag" %>

<%@ page import="jp.co.intra_mart.common.aid.javaee.http.MultipartFormData" %>
<%@ page import="jp.co.intra_mart.foundation.service.client.file.*" %>

<%
  String msg = new String("ファイルをアップロードしました。");
  try{
    // リクエストから必要な情報を取得
    MultipartFormData data = new MultipartFormData(request);
    MultipartFormData.Entity entity = data.getEntity("local_file");
    String header = entity.getHeader("Content-Disposition");
    String file_name = entity.getFileName();

    // Strage Service にファイルをアップロード
    NetworkFile file = new NetworkFile("filebox/" + file_name);
    boolean result = file.save(entity.getBytes());
    if(result == false){
        msg = "ファイルのアップロードが失敗しました。";
    }
  }catch(Exception e){
    msg = "ファイルのアップロードが失敗しました。";
  }
%>

<!-- 処理後に表示する画面 -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 //EN">
<HTML>
  <HEAD>
    <imarttag:imartDesignCss />
  </HEAD>
  <BODY>
    <!-- タイトルバー表示 -->
```

```
  <imarttag:imartTitleBar title="ファイル・アップ 結果" />
  <!-- ツールバー表示-->
  <imarttag:imartToolbarFrame>
    <imarttag:imartToolbarRight>
      <imarttag:imartIcon
        name="戻る"
        icon="/images/standard/arrow_left.gif"
        href="filebox.jsp" />
    </imarttag:imartToolbarRight>
  </imarttag:imartToolbarFrame>
  <H2 align="center"><%= msg %><H2>
 </BODY>
</HTML>
```

この画面では、以下のように動作します。

1. ファイルコントロールにより選択されたファイルデータがサーバに送られます。
2. サーバに送られたデータを解析します。
3. 受信したファイルのファイル名を取得します。
4. 元のファイル名のまま受信したファイルデータを Storage Service に出力します。

アップロードされたファイルと同名のファイルをサーバ上に作成することが可能です。



＜実行画面＞

■ 解説

◆ **<INPUT type="file">**

HTML フォーム中で利用している <INPUT type="file"> について、もう少し詳しく説明します。フォームコントロールである <INPUT type="file"> を利用すると、ブラウザからサーバに対してファイルをアップロードすることができます。

この時、フォームは以下のような記述が必要になります。

<FORM method="POST" enctype="multipart/form-data">

これは、ファイルの情報を MIME 形式にエンコードして POST モードでサーバにリクエストをするという指定になります。

◆ **クラス MultipartFormData/インターフェス MultipartFormData.Entity**

マルチパート形式のフォームデータを簡単に扱うことができるAPIです(このクラスを利用するためには、jp.co.intra_mart.foundation.http パッケージをインポートする必要があります)。

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

＜upload.jsp＞

```
MultipartFormData data = new MultipartFormData(request);
MultipartFormData.Entity entity = data.getEntity("local_file");
String header = entity.getHeader("Content-Disposition");
String file_name = header.substring(header.lastIndexOf("¥¥")
+ 1, header.length() - 1);
```

MultipartFormData のコンストラクタに HttpServletRequest を与えることにより各パート毎を取り出し、任意の操作を行なうことができます。
各パートの取り出し方は getEntity メソッドに取得したいパラメータの名前を指定します。
このとき返却される形式は MultipartFormData.Entity インターフェースとなります。
MultipartFormData.Entity インターフェースを使用することにより、パラーメータの取得やヘッダの解析などを簡単に行なうことができます。

◆ **クラス NetworkFile**

intra-mart File Server 上のファイルを操作するＡＰＩです（このクラスを利用するためには、jp.co.intra_mart.foundation.server.file をインポートする必要があります）。
このAPIを利用する事で、各サーバを分散配置している場合においてもファイルを一元的に管理することができます。ここでは、save()メソッドを利用して新しいファイルを作成しています。

＜upload.jsp＞

```
NetworkFile file = new NetworkFile("filebox/" + file_name);
file.save(entity.getBytes());
```

NetworkFile のコンストラクタに指定したファイルまたはディレクトリにたいして様々な操作を行なうことができます。

2.1.4 ファイルリストの表示

ここでは、前節で作成した画面を改良して、アップロードしたファイルの一覧を表示するようにソースを修正します。アップロードされたファイルを一覧表として表示するためには、保存されているファイルをリストとして取得する必要があります。Storage Service に問い合わせてファイルリストを取得するためのコードを追加します。

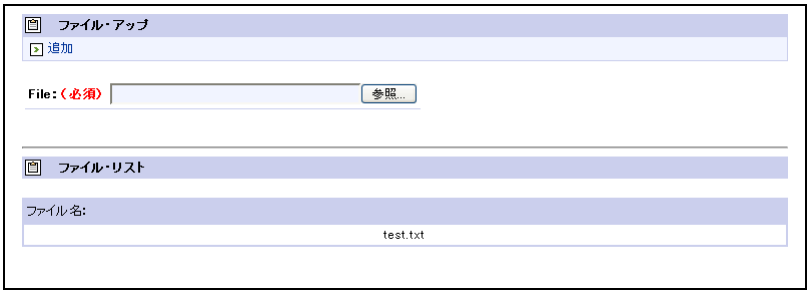
＜一覧表を表示するための JSP（filebox.jsp）＞

```
  <%@ page contentType="text/html; charset=windows-31j" pageEncoding="windows-31j" %>
<%@ taglib prefix="imart" uri="http://www.intra-mart.co.jp/taglib/core/standard" %>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/foundation/imarttag" %>

<%@ page import="jp.co.intra_mart.foundation.http.*" %>
<%@ page import="jp.co.intra_mart.foundation.service.client.file.*" %>
<%@ page import="java.util.*" %>

<%
 // ファイル一覧を取得
 NetworkFile file = new NetworkFile("filebox/");
 if(!file.isDirectory()){
   file.makeDirectories();
 }
  Collection list = file.files();
  List view = new ArrayList();
  if(list != null && list.size() > 0) {
        view = new ArrayList(list);
  }
  String file_name;
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 //EN">
```

```
<HTML>
 <HEAD>
  <imarttag:imartDesignCss />
  <SCRIPT>

  function checkInputValue(){
    //送信先
    if( document.formUpload.local_file.value == "" ) {
      alert("[ File: ]を指定してください。");
      document.formUpload.local_file.focus();
      return false;
    }

    return true;
  }

  function onUpload(){
    if (checkInputValue() == true ) {
      document.formUpload.submit();
    }
  }
  </SCRIPT>
 </HEAD>
 <BODY>
  <!-- タイトルバー表示 -->
  <imarttag:imartTitleBar title="ファイル・アップ" />
  <!-- ツールバー表示-->
  <imarttag:imartToolbarFrame>
    <imarttag:imartToolbarLeft>
      <imarttag:imartIcon
        name="追加"
        icon="/images/standard/next.gif"
        href="JavaScript:onUpload()" />
    </imarttag:imartToolbarLeft>
  </imarttag:imartToolbarFrame>
  <Form name="formUpload"
        method="POST"
        action="upload.jsp"
        enctype="multipart/form-data"
        onSubmit="return checkInputValue();" >
    <TABLE>
      <TR>
        <TD align="center"><br>
          <TABLE class="edit">
            <TR class="bottom">
              <imarttag:imartItemNameTd name = "File:" require = "true" />
              <imarttag:imartInputTd
                  type = "file"
                  name = "local_file"
                  value = ""
                  size = "47" />
            </TR>
            <TR><TD><BR></TD></TR><!-- スペース用 -->
          </TABLE>
        </TD>
      </TR>
    </TABLE>
  </FORM>

  <HR>

  <!-- タイトルバー表示 -->
  <imarttag:imartTitleBar title="ファイル・リスト" /><BR>
  <TABLE class="list_border_bg">
        <TR>
              <imarttag:imartItemNameTd name = "ファイル名:" />
        </TR>
```


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

```
        <%
            for(int i = 0; i < view.size(); i++) {
                file_name = (String)view.get(i);
        %>
        <TR class="bottom">
            <TD class="list_data_bg"><%= file_name %></TD>
        </TR>
        <%
            }
        %>
        </TABLE>
    </BODY>
</HTML>

<!-- End of File -->
```



＜実行画面＞

この実行画面では、すでにファイルがアップロードされていることが確認できます。

■ 解説

◆ **NetworkFile クラス**

NetworkFile クラスは、ファイルのみでなく、ディレクトリ名を指定してインスタンスを作ることができます。これによって、インスタンスは、ディレクトリに対してさまざまな操作を行うことができます。

isDirctory()	存在チェック
makeDirctories()	ディレクトリの作成
files();	ファイルリストの取得

※ 詳細については、API リストを参照してください。

2.1.5 ファイル・ダウンロード

ファイルをアップロードできて、アップロードされたファイルが Storage Service に保存されていることが確認できたので、次は、保存されているファイルをブラウザにダウンロードする機能を追加します。前節で作成した画面に対してファイルをダウンロードできるようにソースを修正します。

ファイルをダウンロードするには、JSP にダウンロードをするためのリンクを追加します。また、そのリンクがクリックされた時にファイルを送信するためのロジックを追加していきます。

＜ファイルダウンロード用リンクを追加した JSP ファイル（filebox.jsp）＞

```
<%@ page contentType="text/html; charset=windows-31j" pageEncoding="windows-31j" %>
<%@ taglib prefix="imart" uri="http://www.intra-mart.co.jp/taglib/core/standard" %>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/foundation/imarttag" %>

<%@ page import="jp.co.intra_mart.foundation.http.*" %>
<%@ page import="jp.co.intra_mart.foundation.service.client.file.*" %>
<%@ page import="java.util.*" %>

<%
 // ファイル一覧を取得
```

```
  NetworkFile file = new NetworkFile("filebox/");
  if(!file.isDirectory()){
    file.makeDirectories();
  }
  Collection list = file.files();
  List view = new ArrayList();
  if(list != null && list.size() > 0) {
    view = new ArrayList(list);
  }
  String file_name;
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 //EN">

<HTML>
  <HEAD>
    <imarttag:imartDesignCss />
    <SCRIPT>

    function checkInputValue(){
      //送信先
      if( document.formUpload.local_file.value == "" ) {
        alert("[ File: ]を指定してください。");
        document.formUpload.local_file.focus();
        return false;
      }

      return true;
    }

    function onUpload(){
      if (checkInputValue() == true ) {
        document.formUpload.submit();
      }
    }
    </SCRIPT>
  </HEAD>
  <BODY>
    <!-- タイトルバー表示 -->
    <imarttag:imartTitleBar title="ファイル・アップ" />
    <!-- ツールバー表示-->
    <imarttag:imartToolbarFrame>
      <imarttag:imartToolbarLeft>
        <imarttag:imartIcon
          name="追加"
          icon="/images/standard/next.gif"
          href="JavaScript:onUpload()" />
      </imarttag:imartToolbarLeft>
    </imarttag:imartToolbarFrame>
    <Form name="formUpload"
          method="POST"
          action="upload.jsp"
          enctype="multipart/form-data"
          onSubmit="return checkInputValue();" >
      <TABLE>
        <TR>
          <TD align="center"><br>
            <TABLE class="edit">
              <TR class="bottom">
                <imarttag:imartItemNameTd name = "File:" require = "true" />
                <imarttag:imartInputTd
                    type = "file"
                    name = "local_file"
                    value = ""
                    size = "47" />
              </TR>
              <TR><TD><BR></TD></TR><!-- スペース用 -->
            </TABLE>
          </TD>
```


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

```
      </TR>
    </TABLE>
  </FORM>

  <HR>

  <!-- タイトルバー表示 -->
  <imarttag:imartTitleBar title="ファイル・リスト" />><BR>
  <TABLE class="list_border_bg" >
    <TR>
      <imarttag:imartItemNameTd name="ファイル名" />
      <TD><imarttag:imartIcon name="表示" icon="/images/standard/list.gif" />></TD>
    </TR>
    <%
      for(int i = 0; i < view.size(); i++) {
        file_name = (String)view.get(i);
    %>
    <TR class="bottom">
        <TD class="list_data_bg"><%= file_name %></TD>
        <TD class="list_data_bg"><A href="download.jsp?file_name=<%= file_name %> ">download</A></TD>
    </TR>
    <%
      }
    %>
  </TABLE>
 </BODY>
</HTML>

<!-- End of File -->
```

2.1.5.1 Downloadと表示するためのリンクを作成

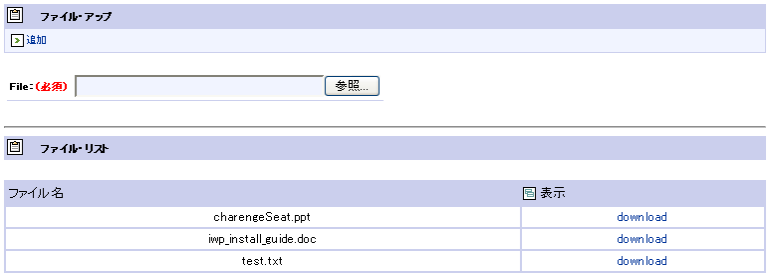
このリンクには、クリックされた時にダウンロード処理をするための JSP のページ download.jsp が指定されていますので、download.jsp を作成します。

＜ダウンロード用 JSP ファイル（download.jsp）＞

```
<%@ page contentType="text/html; charset=Windows-31J" %>
<%@ page
import="jp.co.intra_mart.foundation.service.client.file.NetworkFile" %><%@ page
import="java.io.OutputStream" %>
<%

    String fileName = request.getParameter("file_name");
    NetworkFile file = new NetworkFile("filebox/" + fileName);

    response.reset();

    OutputStream os = response.getOutputStream();
    response.setContentType("application/octet-stream");
    response.setHeader("Content-Disposition", "attachment; filename=¥"" + fileName + "¥"");
    os.write(file.load());
    os.close();
%>
```

※ ここでは説明を簡単にするため JSP でのダウンロードサンプルを紹介しましたが、一般的には Servlet を用いてダウンロードします。ダウンロード用 Servlet ファイルのサンプルは以下に用意されています。

※ im_sample.zip の im_sample/src/main/java/jp/co/intra_mart/sample/servlet/Download.java



＜実行画面＞

■ 解説

◆ **setContentType()**

ファイルをダウンロードする場合、ファイルの種類に合わせてコンテントタイプを指定する必要があります。このサンプルでは、不明なデータを意味する application/octet-stream を指定してブラウザの判断にまかせていますが、実際にはダウンロードするファイルの内容に合わせて適切な MIME タイプを指定するようにしてください。

2.1.6 ファイルの削除

前節までで、ファイルをアップロードすることと、アップロードしたファイルをダウンロードすることができました。しかし、このままではアップロードされたファイルが Storage Service に溜まっていってしまうので、ここでは、アップロードされて Storage Service 上に保存されているファイルを削除する機能を追加します。

＜filebox.jsp＞

```
<%@ page contentType="text/html; charset=windows-31j" pageEncoding="windows-31j" %>
<%@ taglib prefix="imart" uri="http://www.intra-mart.co.jp/taglib/core/standard" %>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/foundation/imarttag" %>

<%@ page import="jp.co.intra_mart.foundation.http.*" %>
<%@ page import="jp.co.intra_mart.foundation.service.client.file.*" %>
<%@ page import="java.util.*" %>

<%
  // ファイル一覧を取得
  NetworkFile file = new NetworkFile("filebox/");
  if(!file.isDirectory()){
    file.makeDirectories();
  }
  Collection list = file.files();
  List view = new ArrayList();
  if(list != null && list.size() > 0) {
    view = new ArrayList(list);
  }
  String file_name;
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 //EN">

<HTML>
  <HEAD>
    <imarttag:imartDesignCss />
    <SCRIPT>
    function checkInputValue(){
      //送信先
      if( document.formUpload.local_file.value == "" ) {
        alert("[ File: ]を指定してください。");
        document.formUpload.local_file.focus();
        return false;
      }
```


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

```
    return true;
  }

  function onUpload(){
    if (checkInputValue() == true ) {
      document.formUpload.submit();
    }
  }
  </SCRIPT>
</HEAD>
<BODY>
  <!-- タイトルバー表示 -->
  <imarttag:imartTitleBar title="ファイル・アップ" />
  <!-- ツールバー表示-->
  <imarttag:imartToolbarFrame>
    <imarttag:imartToolbarLeft>
      <imarttag:imartIcon
        name="追加"
        icon="/images/standard/next.gif"
        href="JavaScript:onUpload()" />
    </imarttag:imartToolbarLeft>
  </imarttag:imartToolbarFrame>
  <Form name="formUpload"
        method="POST"
        action="upload.jsp"
        enctype="multipart/form-data"
        onSubmit="return checkInputValue();" >
    <TABLE>
      <TR>
        <TD align="center"><br>
          <TABLE class="edit">
            <TR class="bottom">
              <imarttag:imartItemNameTd name = "File：" require = "true" />
              <imarttag:imartInputTd
                  type = "file"
                  name = "local_file"
                  value = ""
                  size = "47" />
            </TR>
            <TR><TD><BR></TD></TR><!-- スペース用 -->
          </TABLE>
        </TD>
      </TR>
    </TABLE>
  </FORM>

  <HR>

  <!-- タイトルバー表示 -->
  <imarttag:imartTitleBar title="ファイル・リスト" /><BR>
  <TABLE class="list_border_bg" >
    <TR>
      <imarttag:imartItemNameTd name="ファイル名" />
      <TD><imarttag:imartIcon name="表示" icon="/images/standard/list.gif" /></TD>
      <TD><imarttag:imartIcon name="削除" icon="/images/standard/delete.gif" /></TD>
    </TR>
    <%
      for(int i = 0; i < view.size(); i++) {
        file_name = (String)view.get(i);
    %>
    <TR class="bottom">
      <TD class="list_data_bg"><%= file_name %></TD>
      <TD class="list_data_bg"><A href="download.jsp?file_name=<%= file_name %>">
            download</A>
      </TD>
      <TD class="list_data_bg"><A href="remove.jsp?file_name=<%= file_name %> ">
            remove</A>
```

```
        </TD>
      </TR>
      <%
        }
      %>
    </TABLE>
  </BODY>
</HTML>

<!-- End of File -->
```

2.1.6.1 removeと表示するためのリンクを作成

＜ファイルを削除する JSP ファイル（remove.jsp）＞

```
<%@ page language="java" contentType="text/html; charset=windows-31j"
      pageEncoding="windows-31j"%>
<%@ page import="jp.co.intra_mart.foundation.service.client.file.NetworkFile" %>
<%@ page import="java.util.*"%>

<%
   String msg = new String("Remove succeeded.");
   String file_name = request.getParameter("file_name");
   NetworkFile file = new NetworkFile("filebox/" + file_name);
   boolean result = file.remove();
   if(result == false){
         msg = "Remove failed.";
   }
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 //EN">
<HTML>
<HEAD>
 <TITLE>File Remove</TITLE>
</HEAD>
<BODY bgcolor="WhiteSmoke">
 <H2><%= msg %><H2>
 <A href="filebox.jsp">return</A>
</BODY>
</HTML>
```

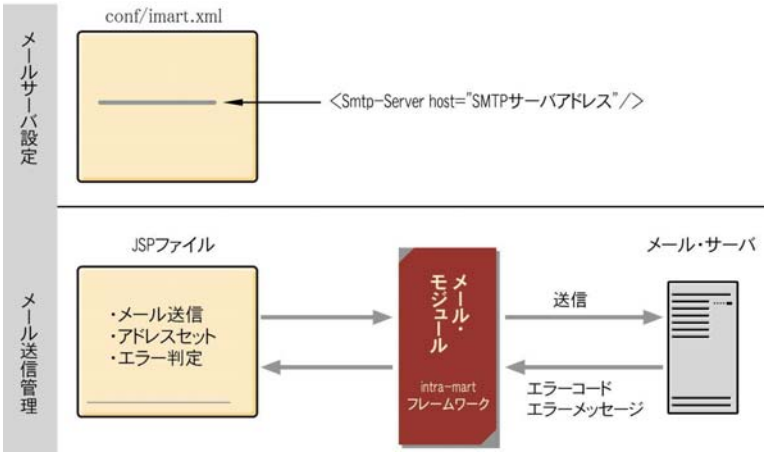
＜実行画面（結果）＞

削除リンクをクリックすると、該当するファイルを削除することができます。

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

2.2 メール連携モジュール

本モジュールを利用することで、SMTP/POP3 互換のメールサーバに対するメールの送信処理を行うことができます。



2.2.1 メール送信

ここでは、メールを送信するための画面を作成します。メールを送信するためには、conf/imart.xml の SMTP サーバの設定を正しく行ってください。

■ メールサーバの設定

メールサーバの設定は、conf/imart.xml ファイルで行います。記述例は以下の通りです。

```
<smtp-server host="localhost" port="25" mailbox-check="false" />
```

2.2.1.1 メール送信フォームの作成

メールを送信するためのフォームを作成します。フォーム内には、メール送信に必要な送信先アドレス、送信者アドレス、件名、本文を登録するコントロールを用意します。また、受け取った情報を元にして Mail API を利用してメール送信を行うための関数を定義します。

＜sender.jsp＞

```
<%@ page contentType="text/html; charset=windows-31j" pageEncoding="windows-31j" %>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/foundation/imarttag" %>

<HTML>
 <HEAD>
  <imarttag:imartDesignCss />

  <SCRIPT>
  function checkInputValue(){

    //送信先
    if( document.formSendMail.mail_to.value == "" ) {
      alert("[ To: ]を入力してください。");
      document.formSendMail.mail_to.focus();
      return false;
    }
    /*
    // 簡単メールフォーマットチェック
    if( !document.formSendMail.mail_to.value.match(/.+@.+¥..+/)){
      alert("[ To: ]に入力したメールアドレスが不正です。");
      document.formSendMail.mail_to.focus();
```

```
      return false;
    }
    */

    //送信元
    if( document.formSendMail.mail_from.value == "" ) {
      alert("[ From: ]を入力してください。");
      document.formSendMail.mail_from.focus();
      return false;
    }
    /*
    // 簡単メールフォーマットチェック
    if( !document.formSendMail.mail_from.value.match(/.+@.+¥..+/)){
      alert("[ From: ]に入力したメールアドレスが不正です。");
      document.formSendMail.mail_from.focus();
      return false;
    }
    */

    return true;
  }
  function onSenderMail(){
    if (checkInputValue() == true ) {
      document.formSendMail.submit();
    }
  }

  </SCRIPT>
</HEAD>
<BODY>
  <!-- タイトルバー表示 -->
  <imarttag:imartTitleBar title="メール送信" />
  <!-- ツールバー表示-->
  <imarttag:imartToolbarFrame>
    <imarttag:imartToolbarLeft>
      <imarttag:imartIcon
        name="送信"
        icon="/images/standard/next.gif"
        href="JavaScript:onSenderMail()" />
    </imarttag:imartToolbarLeft>
  </imarttag:imartToolbarFrame>

  <Form   name="formSendMail"
          method="POST"
          action="mail.jsp"
          onSubmit="return checkInputValue();" >
    <TABLE class="list_border_bg">
      <TR>
        <TD class="list_title_bg" align="center"><br>
          <TABLE class="edit">
            <TR class="bottom">
              <imarttag:imartItemNameTd name = "To:" require = "true" />
              <imarttag:imartInputTd
                  type = "text"
                  name = "mail_to"
                  value = ""
                  size = "57" />
            </TR>
            <TR class="bottom">
              <imarttag:imartItemNameTd name = "From:" require = "true" />
              <imarttag:imartInputTd
                  type = "text"
                  name = "mail_from"
                  value = ""
                  size = "57" />
            </TR>
            <TR class="bottom">
              <imarttag:imartItemNameTd name = "Subject:" />
```


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

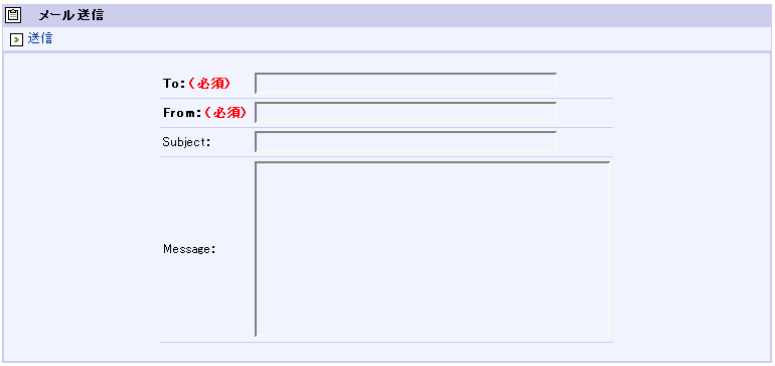
```
            <imarttag:imartInputTd
                type = "text"
                name = "mail_subject"
                size = "57" />
          </TR>
          <TR class="bottom">
            <imarttag:imartItemNameTd name = "Message:" />
            <imarttag:imartInputTd
                type="textarea"
                name="mail_body"
                cols="40" rows="8" />
          </TR>
          <TR><TD><BR></TD></TR><!-- スペース用 -->
        </TABLE>
      </TD>
    </TR>
  </TABLE>
 <Form>
</BODY>
</HTML>
```

＜メール送信ロジックを記述した JSP ファイル（mail.jsp）＞

```
<%@ page import="jp.co.intra_mart.foundation.mail.MailSender" %>
<%@ page import="jp.co.intra_mart.foundation.mail.javamail.*" %>
<%@ page import="jp.co.intra_mart.foundation.http.*" %>
<%@ page import="jp.co.intra_mart.foundation.utility.io.*" %>
<%@ page import="java.io.*" %>

<%
  String msg = new String("Succeeded.");

  // 送信するメールを作成
  StandardMail mail = new StandardMail();
  mail.addTo(request.getParameter("mail_to"));
  mail.setFrom(request.getParameter("mail_from"));
  mail.setSubject(request.getParameter("mail_subject"));
  mail.setText(request.getParameter("mail_body"));

  // 実際にメールを送信
  MailSender mailSender = new JavaMailSender(mail);
  mailSender.send();
%>

<!-- 送信後に表示する画面 -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 //EN">
<HTML>
  <HEAD>
    <imarttag:imartDesignCss />
  </HEAD>
  <BODY>
    <!-- タイトルバー表示 -->
    <imarttag:imartTitleBar title="メール送信結果" />
    <!-- ツールバー表示-->
    <imarttag:imartToolbarFrame>
      <imarttag:imartToolbarRight>
        <imarttag:imartIcon
          name="戻る"
          icon="/images/standard/arrow_left.gif"
          href="sender.jsp" />
      </imarttag:imartToolbarRight>
    </imarttag:imartToolbarFrame>
    <H2 align="center"><%= msg %><H2>
  </BODY>
</HTML>
```

※ ここでのサンプルは、理解しやすくするために、JSP ファイル中に呼び出しのコーディングが記述されています。

このコーディングは、JavaEE フレームワーク利用時では、イベントリスナーや DAO 中に記述するなど適切な位置に盛り込んでください。



<実行画面>

フォーム中の必要事項をすべて入力した後に[send]ボタンをクリックするとメールを送信することができます。

■ 解説

◆ **Mail API**

MailSender API

メールを送信を行なうクラスです。

＜mail.jsp＞

```
StandardMail mail = new StandardMail();
mail.addTo(request.getParameter("mail_to"));
mail.setFrom(request.getParameter("mail_from"));
mail.setSubject(request.getParameter("mail_subject"));
mail.setText(request.getParameter("mail_body"));
MailSender sender = new JavaMailSender(mail);
sender.send();
```

boolean result = mail.send();

サンプルプラグラムのようにコンストラクタに引数を与えない場合はイントラマートの初期設定ファイル（conf/jp.co.intra_mart.foundation.conf ）に設定されているSMTPサーバを利用します。コンストラクタにSMTPサーバのアドレスとポートを指定することにより任意のSMTPサーバを利用することができます。実際にメールが送信されるのは、send()メソッドを実行したときになります。

◆ **TO および From 設定時の注意点**

設定データ中にメールアドレス以外の不正な文字が含まれている場合、メール送信エラーが発生する場合があります。

◆ **メール送信とサーバ処理速度**

メールを送信する場合、intra-mart Application Server と SMTP サーバが連携する必要があります。送信するメールの情報量はもちろんのこと、ネットワーク環境やネットワークトラフィックなどによりメール送信処理時間がかかる場合があります。

MailSender クラスの詳細についてはAPIリストをご覧ください。

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

2.2.2 添付ファイル付きメールの送信

前節で作成したメール送信フォームを完了して添付ファイルをメール本文と共に送信できるようにします。

まずは、HTML ページでは添付ファイルとするためのファイルをアップロードするためのフォーム・コントロールを追加し、フォームの属性を変更します。

JSP ファイルは、フォームの修正に合わせて、メール送信関数を添付ファイルに対応できるように修正します。

＜sender.jsp＞

```
<%@ page contentType="text/html; charset=windows-31j" pageEncoding="windows-31j" %>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/foundation/imarttag" %>
<%@ taglib prefix="imartj2ee" uri="http://www.intra-mart.co.jp/taglib/core/framework" %>

<HTML>
 <HEAD>
  <imarttag:imartDesignCss />

  <SCRIPT>
  function checkInputValue(){

    //送信先
    if( document.formSendMail.mail_to.value == "" ) {
      alert("[ To: ]を入力してください。");
      document.formSendMail.mail_to.focus();
      return false;
    }
    /*
    // 簡単メールフォーマットチェック
    if( !document.formSendMail.mail_to.value.match(/.+@.+¥..+/)){
      alert("[ To: ]に入力したメールアドレスが不正です。");
      document.formSendMail.mail_to.focus();
      return false;
    }
    */

    //送信元
    if( document.formSendMail.mail_from.value == "" ) {
      alert("[ From: ]を入力してください。");
      document.formSendMail.mail_from.focus();
      return false;
    }
    /*
    // 簡単メールフォーマットチェック
    if( !document.formSendMail.mail_from.value.match(/.+@.+¥..+/)){
      alert("[ From: ]に入力したメールアドレスが不正です。");
      document.formSendMail.mail_from.focus();
      return false;
    }
    */

    return true;
  }

  function onSenderMail(){
    if (checkInputValue() == true ) {
      document.formSendMail.submit();
    }
  }

  </SCRIPT>
 </HEAD>
 <BODY>
  <!-- タイトルバー表示 -->
  <imarttag:imartTitleBar title="メール送信" />
```

```
  <!-- ツールバー表示-->
  <imarttag:imartToolbarFrame>
    <imarttag:imartToolbarLeft>
      <imarttag:imartIcon
        name="送信"
        icon="/images/standard/next.gif"
        href="JavaScript:onSenderMail()" />
    </imarttag:imartToolbarLeft>
  </imarttag:imartToolbarFrame>

  <Form   name="formSendMail"
          method="POST"
          action="mail.jsp"
          onSubmit="return checkInputValue();"
          enctype="multipart/form-data" >
    <TABLE class="list_border_bg">
      <TR>
        <TD class="list_title_bg" align="center"><br>
          <TABLE class="edit">
            <TR class="bottom">
              <imarttag:imartItemNameTd name = "To:" require = "true" />
              <imarttag:imartInputTd
                  type = "text"
                  name = "mail_to"
                  value = ""
                  size = "57" />
            </TR>
            <TR class="bottom">
              <imarttag:imartItemNameTd name = "From:" require = "true" />
              <imarttag:imartInputTd
                  type = "text"
                  name = "mail_from"
                  value = ""
                  size = "57" />
            </TR>
            <TR class="bottom">
              <imarttag:imartItemNameTd name = "Subject:" />
              <imarttag:imartInputTd
                  type = "text"
                  name = "mail_subject"
                  size = "57" />
            </TR>
            <TR class="bottom">
              <imarttag:imartItemNameTd name = "Message:" />
              <imarttag:imartInputTd
                  type="textarea"
                  name="mail_body"
                  cols="40" rows="8" />
            </TR>
            <TR class="bottom">
              <imarttag:imartItemNameTd name = "Attachment:" />
              <imarttag:imartInputTd
                    type="file"
                    name="mail_file"
                    size="47" />
            </TR>
            <TR><TD><BR></TD></TR><!-- スペース用 -->
          </TABLE>
        </TD>
      </TR>
    </TABLE>
  <Form>
 </BODY>
</HTML>
```


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

■ 解説

◆ ファイルをアップロードするためのフォーム

ファイルをアップロードするためには以下のようなフォームの記述が必要になります。

<FORM method="POST" enctype="multipart/form-data">

こうすることで、サーバー上では、フォームコントロール <INPUT type="file"> によりローカルのファイルを受け取ることができます。

◆ 受信情報と文字コード

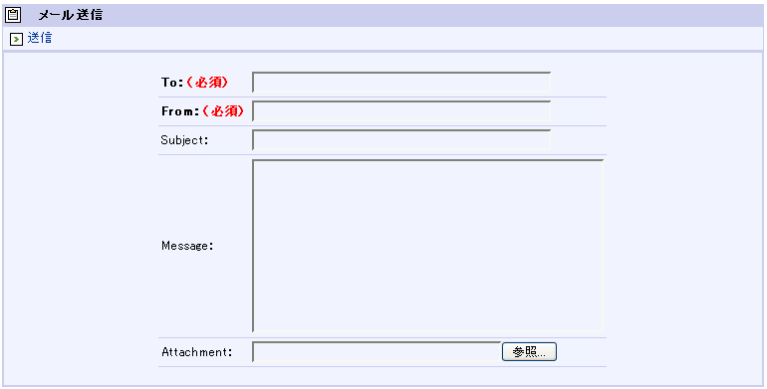
ファイルアップロード時のフォームから受け取った情報は、すべてバイナリ状態になっています。

当然、ファンクションコンテナでは request オブジェクトを通して受け取る値がバイナリになっていますので、その情報を扱う場合には、十分な注意が必要となります。

<添付ファイル送信に対応した JSP ファイル（mail.jsp）>

```
<%@ page contentType="text/html; charset=windows-31j" pageEncoding="windows-31j" %>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/foundation/imarttag" %>

<%@ page import="jp.co.intra_mart.foundation.mail.MailSender" %>
<%@ page import="jp.co.intra_mart.foundation.mail.javamail.*" %>
<%@ page import="jp.co.intra_mart.foundation.http.*" %>
<%@ page import="jp.co.intra_mart.foundation.utility.io.*" %>
<%@ page import="jp.co.intra_mart.foundation.security.*" %>
<%@ page import="java.io.*" %>

<%
  String msg = new String("メール送信しました。");

  // セッション情報を取得
  SessionInfo info = AccessSecurityManager.getInstance().getSessionInfo(request, response);

  // リクエストから必要な情報を取得
  MultipartFormData data = new MultipartFormData(request);

 // 送信するメールを作成
 StandardMail mail = new StandardMail();
 mail.addTo(data.getEntity("mail_to").getContent());
 mail.setFrom(data.getEntity("mail_from").getContent());
 mail.setSubject(data.getEntity("mail_subject").getContent(info.getEncoding()));
 mail.setText(data.getEntity("mail_body").getContent(info.getEncoding()));

  // メールに添付ファイルを追加
  MultipartFormData.Entity entity = data.getEntity("mail_file");
  if(entity.getContentLength() > 0){

        // ヘッダを取得し、添付ファイル名を抽出
        String header = entity.getHeader("Content-Disposition");
        String file_name = header.substring(header.lastIndexOf("¥¥") + 1, header.length() - 1);

        // 添付ファイルを追加
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        baos.write(entity.getBytes());
        mail.addAttachment(file_name, baos.toString("8859_1"));
              // [注意]エンコードはバイナリ(8859_1)を指定すること
        baos.close();
  }

 // 実際にメールを送信
 MailSender mailSender = new JavaMailSender(mail);
 mailSender.send();
%>

<!-- 送信後に表示する画面 -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 //EN">
<HTML>
```

```
<HEAD>
  <imarttag:imartDesignCss />
</HEAD>
<BODY>
  <!-- タイトルバー表示 -->
  <imarttag:imartTitleBar title="メール送信結果" />
  <!-- ツールバー表示-->
  <imarttag:imartToolbarFrame>
    <imarttag:imartToolbarRight>
      <imarttag:imartIcon
        name="戻る"
        icon="/images/standard/arrow_left.gif"
        href="sender.jsp" />
    </imarttag:imartToolbarRight>
  </imarttag:imartToolbarFrame>
  <H2 align="center"><%= msg %><H2>
</BODY>
</HTML>
```



＜実行画面(結果)＞

■ 解説

◆ 添付ファイル

ファイルを添付してメール送信する場合、RFC の規約によりファイルのデータそのものをエンコードしたのちにメール本本も含めたメール情報全体をエンコードしてから SMTP サーバに対して送信する必要があります。

◆ 添付ファイルと処理速度

添付ファイルを送信する場合、ブラウザが Web サーバに対してファイルデータを送信し、その情報を受信したアプリケーションサーバがメール送信処理を行います。

1 つのメール送信に対して複数のネットワークを介しますので、サイズの大きなファイルを添付してメール送信する場合には、メール送信処理に時間がかかってしまう場合があります。

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

2.3 外部プロセスの呼び出し

実行中の intra-mart アプリケーションから他のプログラムを実行するには、java.lang.Runtime クラスの exec() メソッドを利用します。この関数は、指定の文字列コマンドを新しいプロセスとして実行します。

＜サンプル例＞

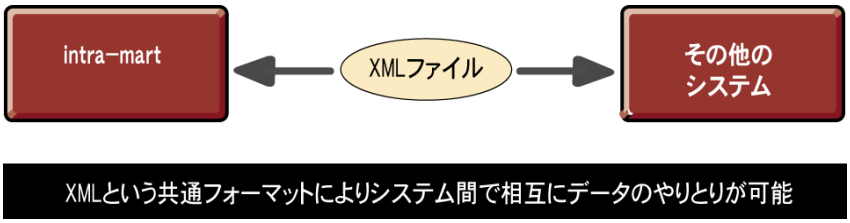
```
java.lang.Runtime runtime = java.lang.Runtime.getRuntime();
java.lang.Process process = runtime.exec(command)
process.waitFor();
```

サンプルの command には String 型のコマンドを指定してください。(例 String command = “notepad”)
command に指定されたコマンドをOSに渡してプロセスを実行します。java プログラム側では、java.lang.Process のインスタンス process の waitFor() メソッドにより、新しく実行したプロセスの終了を待ちます。

※ command によるプロセスは、Application Runtime の動作している環境で実行されます。他のサーバの環境では実行できません。
※ プロセスを実行するためのコマンドはOSにより異なります。Application Runtime の実行環境に合わせて command を指定するようにしてください。
※ 詳細は、JAVA の API 仕様を参照してください。

2.4 XML形式のデータを扱う

XML パーサを利用することにより、XML 形式のデータを解析して、目的のデータを取り出すことができます。
XML（Extensible Markup Language）は環境にとらわれない非常に柔軟性の高い汎用的な規約となっています。
これにより、他のアプリケーションと XML ファイルを通してデータのやりとりをスムーズに行うことができます。



2.4.1 XMLパーサーとデータの取得

intra-mart の API として提供されている XML パーサーを利用すると、XML 形式のデータを解析して DOM（Document Object Model）ツリー形式に変換します。XML の各タグやその中に記述されているデータを DOM ツリーオブジェクトから取得する事ができます。

※ XML および DOM ツリーに関しては、W3C が規約を定めています。最新の情報に関しては W3C のホームページ等を参照してください。
※ XML パーサーに関しては、W3C および SAX のホームページ上で最新の技術情報が公開されています。
※ 詳細は、API リストの「アプリケーション共通モジュール」の「java.xml.parsers パッケージ」を参照してください。

2.5 スクリプト開発モデルとの連携

ここでは、スクリプト開発モデルの画面からフレームワークの画面へ遷移する方法について説明します。下記の方法で、スクリプト開発モデルの画面から im-JavaEE Framework の画面へ遷移することができます。この方法を応用してログイン画面の遷移をカスタマイズすることにより、標準提供のメニュー表示のないシステムを構築することもできます。
なお、intra-mart にログイン済であることが前提となりますのでご注意ください。

2.5.1 メニューを介さないでim-JavaEE Frameworkの画面を表示する方法

標準でインストールし、web.xml で/ServiceServlet の URL マッピングに関連するところを修正していない場合、以下のようなリクエストを出せば表示することが可能です。

```
http://<サーバ IP アドレス>[:<サーバポート>]/<コンテキストパス>/[アプリケーション ID]-[サービス ID].service。
```

2.5.2 スクリプト開発モデルの画面からフレームワークの画面へ遷移する方法

上記と同じリクエストを行います。

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

ページベースの画面のフォームのサブミットで im-JavaEE Framework に遷移する例を以下に示します。

＜sample.html＞

```
・・・
<FORM name="frmInfo" method="POST" action="<アプリケーション ID>-<サービス ID>.service">

</FORM>
・・・
```

2.6 グラフ描画モジュール

グラフの画像ファイルをサーバサイド作成して、ブラウザ画面上にグラフを表示します。
以下の 5 種類のグラフが利用できます。

- 折れ線グラフ
- 棒グラフ
- 円グラフ
- レーダーチャート
- ポートフォリオ

■ グラフ描画の設定

intra-mart タグライブラリの Drawer タグを利用します。
Drawer タグの属性 data に jp.co.intra_mart.foundation.drawer.AbstractDrawer をスーパークラスとしたインスタンスを渡すことによりグラフの画像ファイルを HTML に表示します。

AbstractDrawer をスーパークラスとした、グラフ描画クラスは以下の5種類です。

クラス	グラフの種類
jp.co.intra_mart.foundation.graph.BarGraphDrawer	折れ線グラフ
jp.co.intra_mart.foundation.graph.CircleGraphDrawer	棒グラフ
jp.co.intra_mart.foundation.graph.LineGraphDrawer	円グラフ
jp.co.intra_mart.foundation.graph.PortFolioDrawer	レーダーチャート
jp.co.intra_mart.foundation.graph.RadarChartDrawer	ポートフォリオ

棒グラフ描画の記述例は以下のようになります。

```
<%@ page contentType="text/html; charset=Windows-31J" pageEncoding="Windows-31J" %>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/core/standard" %>
<%@ page import="jp.co.intra_mart.foundation.graph.*" %>
<%@ page import="java.awt.Color" %>
<%
  // 棒グラフの作成
  BarGraphDrawer drawerBar = new BarGraphDrawer(0, 60, 10);
  drawerBar.setWidth(300);
  drawerBar.setHeight(300);
  drawerBar.setCaption(new String[]{"１月", "２月", "３月"});

  BarGraphObject objBar = new BarGraphObject(Color.black);
  objBar.addData(50);
  objBar.addData(10);
  objBar.addData(30);
  drawerBar.addData(objBar);
%>
<HTML>
  <BODY>
    <imarttag:Drawer data="<%= drawerBar %>" />
  </BODY>
</HTML>
```


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

■ 新しいグラフの作成

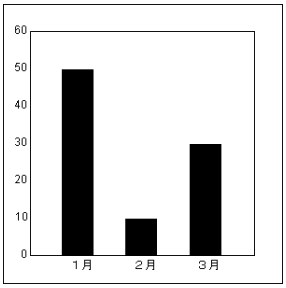
グラフを新規に作成する場合、AbstractDrawer を利用します。

AbstractDrawer を継承したクラスを作成し、createImage メソッドをオーバーライドしてください。

```
public class MyGraphDrawer extends AbstractDrawer {

  public MyGraphDrawer(
    super();
  }

  protected void createImage(Graphics g) {

    g.setColor(Color.red);
    g.drawLine(0, 0, getWidth() - 1, getHeight() - 1);

    g.setColor(Color.blue);
    g.drawRect(0, 0, getWidth() - 1, getHeight() - 1);

    .
    .
    .
    .
  }
}
```

createImage メソッドには画像描画の実行部分を記述します。引数のグラフィックコンテキストに対して、グラフイメージを描画してください。

グラフのプログラムソースはインストールディレクトリ/source に格納されています。具体的な記述方法はプログラムソースを参照してください。



2.7 アクセスコントローラモジュール

アクセスコントローラタグで挟まれている内容の、表示・非表示を制御することが可能です。

- ◆ アクセスコントローラタグで囲まれた領域がアクセスコントローラの制御範囲となります。
- ◆ アクセス権はロール、組織、役職、パブリックグループにより制御できます。
- ◆ アクセス権が存在しないユーザに対しては、アクセスコントローラ制御範囲の内容を非表示にします。
- ◆ アクセスコントローラタグ

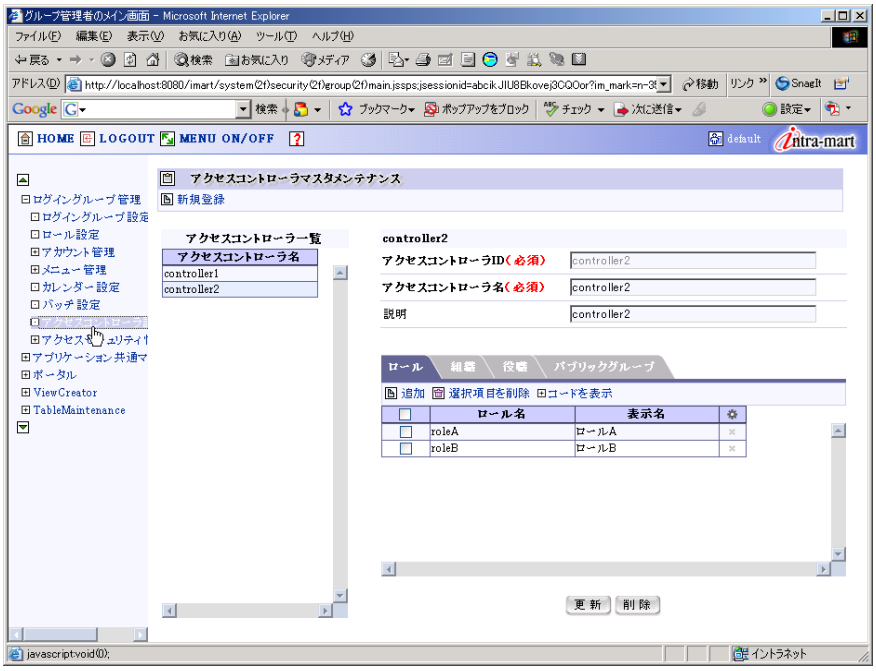
  JSP ページにアクセスコントローラタグを記述することで、表示・非表示の制御を行います。

```
// アクセスコントローラ ID(controller1)に設定されたアクセス権で表示を制御します。
// controller1 のアクセス権情報が存在しない場合は、内容を表示しません。
<imartTag:accessCtrl controller="controller1">
    アクセスコントローラタグ:controller1 で囲まれた内容です。
</imartTag:accessCtrl>

// アクセスコントローラ ID(controller2)に設定されたアクセス権で表示を制御します。
// controller2 のアクセス権情報が存在しない場合は、内容を表示します。
<imartTag:accessCtrl controller="controller2" defaultShow="true">
   アクセスコントローラタグ:controller2 で囲まれた内容です。
</imartTag:accessCtrl>

// アクセスコントローラ ID(controller3)に設定されたアクセス権で表示を制御します。
// controller3 のアクセス権情報が存在しない場合は、内容を表示しません。
<imartTag:accessCtrl controller="controller3" defaultShow="false">
    アクセスコントローラタグ:controller3 で囲まれた内容です。
</imartTag:accessCtrl>
```

- ◆ アクセスコントローラのアクセス権設定

  アクセスコントローラのアクセス権設定は、ログイングループ管理者の「アクセスコントローラ設定」画面行います。

  各アクセスコントローラ(アクセスコントローラ ID)に対して、表示させるための権限(ロール、組織、役職、パブリックグループ)を設定します。



Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

2.8 ショートカットアクセス機能

ショートカットアクセス機能は、初期アクセス URL にショートカットアクセス用の URL パラメータを指定することによって、ログイン後の画面を任意の画面に取り替えることができる機能です。
ショートカットアクセス機能を用いると、ログイン後の画面でトップバーおよびメニュー画面が存在し、そのメインエリアに任意のページを表示することが可能になります。
また、表示に関するセキュリティも設定することが可能です。
詳しくは、「アクセスセキュリティ仕様書」を参照してください。

ショートカットアクセス機能を用いた場合に利用できるセキュリティ機能は以下の通りです。

- ◆ 指定ページを表示できるユーザの指定。（複数設定可能）
- ◆ 指定ページを表示できる有効期間の指定。
- ◆ ログインの制御に関して以下の機能を選択できます。
  - ● ログイン画面からユーザ ID、パスワードを入力後、直接指定ページにアクセス。
  - ● ユーザ ID、パスワードを指定せずに、直接ページにアクセス可能。
    （表示許可ユーザを一名だけ指定した場合のみ利用できる機能です。）
- ◆ ショートカットアクセス URL
  ショートカットアクセス用の URL は、通常の初期アクセス URL にショートカット ID をパラメータとして追加した URL です。

```
http://<server>/<context-path>/<login-group>.portal?im_shortcut=<ショートカット ID>
```

＜例＞

```
http://localhost/imart/default.portal?im_shortcut=xazh03nbe43wd
```

- ◆ ショートカット ID の作成
  ショートカット ID は、表示するページの情報およびセキュリティの情報に紐づく ID となります。
  ショートカット ID は、表示するページの情報およびセキュリティの情報を指定して API を用いて作成します。
  メインエリアに、pages/src/sample/shortcut.js および shortcut.html を指定する場合のショートカット ID の作成手順を説明します。

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

//  ショートカット情報の作成
ShortCutInfo shortCutInfo = new ShortCutInfo();

//  表示する URL
shortCutInfo.setUrl("sample/shortcut.jssp");

//  表示する URL に渡すパラメータの設定(任意指定)
shortCutInfo.setUrlParam("arg1","value1");
shortCutInfo.setUrlParam("arg2","value2");

//  表示許可を行うユーザ
shortCutInfo.setAllowsUsers(new String[]{"guest","ueda"});

// ログイン認証が必要かどうか？（認証必要）
shortCutInfo.setIsAuth(true);

// 以下の 2 通りの方法からどちらかを選択します。
// この情報の有効期限（作成時から１０日間有効）
```

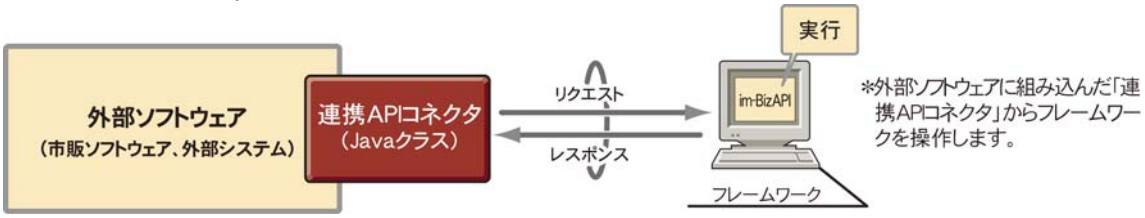
```
shortCutInfo.setValidEndDate(manager.addValidEndDate(10));
// この情報の有効期限(日付指定)
Calendar calendar = Calendar.getInstance();
calendar.set(9999, 11, 31,0, 0, 0);
calendar.set(Calendar.MILLISECOND,0);
shortCutInfo.setValidEndDate(calendar.getTime());

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


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

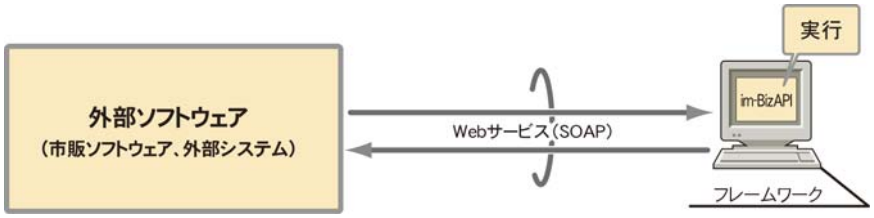
2.9 外部ソフトウェア接続モジュール

市販のアプリケーションパッケージから im-BizAPI の各種 API を呼び出して直接利用するなど、intra-mart と外部ソフトウェアを簡単に連携・接続できるモジュールです。連携・接続する方法には、次の 2 通りの方法が用意されています。

ひとつは、この「連携 API コネクタ」がフレームワークの中の im-BizAPI と連携するための Java ベース API として提供されているので、外部ソフトウェアが Java 実行環境であれば任意のプロセスと im-BizAPI を連携させる方法です。例えば、市販のポータルサーバ製品と組み合わせて、ポータル画面中に intra-mart の画面を表示したり、他のアプリケーションと連携してバッチ動作する独自の Java プロセスからユーザアカウント情報を操作したりすることが可能です。



2 つ目の方法としては、外部ソフトウェアから Web サービスにより im-BizAPI の各種 API を呼び出すことも可能です。



■ 概要

外部ソフトウェア接続モジュールのインターフェースは、Java のクラスです。したがって、連携するアプリケーションは、Java インターフェースを利用できることが前提となります。また、Java インターフェースを利用することから、このモジュールが動作する環境には、Java-VM が必要になります。

連携用のコネクタクラスは、ネットワークを介して Application Runtime と連動します。したがって、ネットワークが利用可能な環境であることも前提条件となります。ただし、API は URL によってネットワーク通信を制御するため、開発者がネットワークを直接意識することはありません。

コネクタは Application Runtime に対して、HTTP 接続を行います。ネットワークは、API に指定された URL により解決されますが、この URL は intra-mart システムの運用形態により以下のような注意が必要です。

■ intra-mart システムがスタンドアロンで運用されている場合は、そのスタンドアロンサーバへの URL を指定します。

■ 分散サーバ型のシステム形態で intra-mart を運用している場合には、Web-Server Connector への接続 URL を指定します(URL の接続先として Application Runtime のアドレスおよびサーバポートを指定しても連携させることはできません)。

連携用に指定する URL には、専用のサーブレットパス(/imart/HTTPActionEventListener(標準))を指定します。通常のサーブレットパスである imart/intramart(標準)を指定しても連携させることはできません。

外部ソフトウェア接続モジュールは、大きく2つの種類に大別できます。1つはサーバ側ロジックで、もう1つはクライアント側ロジックです。

サーバ側ロジックは、サーバの Web-Application 領域(通常は WEB-INF 内)にプログラムを配置します。実行は、サーバ環境内です。主にビジネスロジックを記述する事になります。

クライアント側ロジックは、Application Runtime を利用する Java 実行環境内で実行します。プログラムの

配置は、呼び出し側環境になります。

■ クラスの構成と配置

外部ソフトウェア接続モジュールを利用したクラスのコンパイルおよび動作には、以下のクラスアーカイブが必要になります。

◆ imaca_client.jar (接続するクライアント用)

◆ imaca_provider.jar (サーバ側ロジック用)

サーバ側ロジックはサーバで動作するため、上記アーカイブと共にサーバ側に配置します。配置先ディレクトリは、doc/imart/WEB-INF/ です（上記クラスアーカイブは、doc/imart/WEB-INF/lib にあります）。

クライアント側ロジックは、連携するアプリケーションの動作環境に上記クラスアーカイブと共に配置します。

■ 動作に必要な環境構築

下記の手順で、連携に必要な環境を作成してください。

1. .コネクタの含まれるクラスアーカイブ・ファイル imaca_client.jar を連携させるアプリケーションが動作する環境にコピーしてください(jar ファイルは、CD-ROM 内にあります)。
2. .コピーしたアーカイブファイルの imaca_client.jar に対してクラスパスを設定してください。

これで準備は完了です。アプリケーションを動作させて intra-mart と連動させてみてください。うまく連動できない場合は、クラスパスの設定をもう一度確認してください。また、バージョンの異なる jar ファイルの場合、ファイル名が同じでも連携動作させることができませんので、パッチ等を適用した場合やリビジョンアップをした場合などは、十分注意をして ください。なお、intra-mart サーバに対してパッチを適用したときなどは、パッチに含まれる imaca_client.jar を上書きコピーしてご利用ください。

■ サンプル

intra-mart システム外の Java 実行環境から任意のページをリクエストするURLを取得するサンプルソースについて見て行きます。このクラスでは、スクリプト開発モデルで作成された画面 sample/graph/graph_drawer

(Resource Service の pages/src/ ディレクトリ内にソースがあります) をリクエストするためのURLを作成します。

画面作成に関しては、guest というアカウントIDによるログイン環境にて実行されますので、guest というユーザが存在するようにしてください。

なお、このプログラムでは、ログイングループIDを固定値 default として解決しています。

このソースをコンパイルするときは、以下をクラスパスに設定してコンパイルしてください。

doc/imart/WEB-INF/lib/imaca_client.jar

実行時は、このソースをコンパイルしてできたクラスファイルを、実行する Java プロセス環境から利用できる場所に保存してください。

＜ソース例＞

```
package jp.co.intra_mart.sample.service.client.application;

import java.io.IOException;

import jp.co.intra_mart.foundation.service.client.application.AccountSecurityHTTPActionEventFilterHandler;
import jp.co.intra_mart.foundation.service.client.application.HTTPActionEventHandler;
import jp.co.intra_mart.foundation.service.client.application.HTTPActionEventHandlerException;
import jp.co.intra_mart.foundation.service.client.application.HTTPActionEventURL;
import jp.co.intra_mart.foundation.service.client.application.WebApplicationHTTPActionEventHandler;
import jp.co.intra_mart.foundation.service.client.application.content.AccessibleLinkHTTPActionEventFilterHandler;
import jp.co.intra_mart.foundation.service.client.application.content.PresentationPageHTTPActionEventHandler;

public class JSSPConnectURLCreator{
  /**
   * コマンドプロンプトからメインクラスとして指定された場合に
   * 実行されるメソッド。
```


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

```
 * @param args コマンドライン引数
 */
public static void main(String[] args){
    try{
        // スクリプト開発モデルの画面をリクエストするためのURLを取得
        String jsspURL = createJSSPURL();
        // 結果の表示
        System.out.println("URL: " + jsspURL);
    }
    catch(Exception e){
        e.printStackTrace();
    }
}

/**
 * 指定のアカウントIDによるセキュリティセッション環境で、
 * メニュー画面を作成してソースを返します。
 */
public static String createJSSPURL() throws IOException,
    ClassNotFoundException, HTTPActionEventHandlerException{

    // イベント実行ハンドラの作成
    String path = "sample/graph/graph_drawer";
    HTTPActionEventHandler handler = new PresentationPageHTTPActionEventHandler(path);

    // 絶対パスでリンクするための定義
    handler = new AccessibleLinkHTTPActionEventFilterHandler(handler);

    // ログイン・セキュリティ環境の構築
    String groupId = "default";
    String accountId = "guest";
    handler = new AccountSecurityHTTPActionEventFilterHandler(handler, groupId, accountId);

    // URL の取得
    String url = "http://localhost:8080/imart/HTTPActionEventListener";
    HTTPActionEventURL result = WebApplicationHTTPActionEventHandler.getURL(handler, url);
    return result.getURL();
}

/**
 * コンストラクタ
 */
private JSSPConnectURLCreator(){
    super();
}
}


/* End of File */
```

※ 尚、このサンプルや、その他のサンプルのソースファイルは、
インストール CD の iwp_afw/src/im_sample-src.zip に格納されています。
この機能を利用したサンプルプログラムのパッケージは jp.co.intra_mart.sample.service.client.application
となっています。この中のソースを開き、ソース中に記載されているコメントを良くご覧ください。
サンプルは、以下に列挙するものが用意されています。

- アカウント登録プログラム
- スクリプト開発モデルプログラムへリンクするURL作成プログラム (*)
- サービスフレームワークへリンクするURL作成プログラム (*)
  (* = ポータルサーバ製品との連携等に利用可能)

※ 尚、コンパイルおよび実行には、
doc/imart/WEB-INF/lib/imaca_client.jar , doc/imart/WEB-INF/lib/imaca_provider.jar および
JSDK(Java Servlet Development Kit)が必要です。
コンパイルや実行の際には、これらをクラスパスに定義してください。

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

2.10 バッチ管理モジュール

intra-mart は、バッチサーバによるプログラム実行のスケジューリング機能を提供しています。バッチ実行したいロジックの記述されたバッチプログラムを作成し、バッチ設定画面にて起動日時を設定してください。
※ バッチプログラムは view（画面）を持たないため、JavaEE Framework を利用して作成することはできません。また、画面や Web の仕組みに関わる機能（ファイルのダウンロードなど）を行うこともできません。
※ バッチ管理モジュールの詳細と設定に関しては、グループ管理者ガイドの第 1 章「10 バッチ管理の操作」を参照してください。

＜ソース例＞

```
public class MyBatch implements ProcedureComponent{

    // バッチサーバにインスタンス化されるためのコンストラクタ

    public MyBatch(){

        super();

    }


    public void execute(Properties arg) throws java.lang.RuntimeException, java.lang.Error{

        // バッチのロジック

        System.out.println("Hello. my batch program !");
    }

}
```

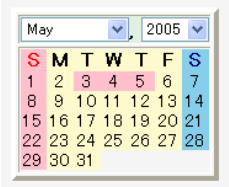
■ 解説

◆ **Java で記述したバッチプログラム**

バッチ実行するロジックを Java で記述するためには intra-mart で提供されているインタフェースクラス jp.co.intra_mart.foundation.server.batch.ProcedureComponent を実装する必要があります。
ProcedureComponent の詳しい仕様は API リストを参照してください。

2.11 カレンダー表示モジュール

カレンダー表示モジュールを組み込みと、カレンダーマスメンテナンス画面で設定したデータと連携したカレンダー画面を表示することができます。カレンダー画面を利用すると、会社の休日や営業日を考慮した日付の入力が行えます。

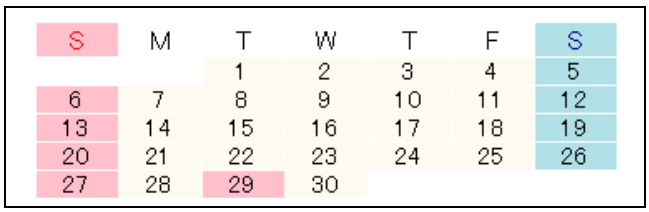


2.11.1 呼び出し方法

intra-mart が標準で提供している拡張タグライブラリの calendar タグを利用することによりカレンダーを画面に貼り付けることができます。

```
<imtag:calendar year="2008" month="4" group="default"/>
```

上記の例では画面上にカレンダーを表示するだけですが、様々なオプションを駆使することで、カレンダーの表示形状を変更したり、ユーザのクリックアクションに対する応答を定義することも可能です。
タグライブラリの仕様に関しては、製品に付属の API リストを参照してください。



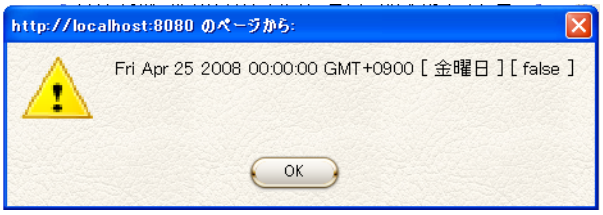
<実行画面>

2.11.2 カレンダーデータの受け取り方法

タグの click_action オプションに対して、クライアントサイドの JavaScript で記述された関数名を指定することで、ユーザが選択(クリック)した年月日情報を取得することができます。

```
<SCRIPT language="JavaScript">
  function click_date(nDate, sName, bHoliday) {
        var dClick = new Date(parseFloat(nDate));
        var sStr = dClick.toString();
        sStr += " [ " + sName + " ]";
        sStr += " [ " + bHoliday + " ]";
        window.alert(sStr);
}
</SCRIPT>
<imtag: calendar year="2008" month="4" group="default" click_action="click_date"/>
```


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.



<実行画面>

2.11.3 カレンダーマスタメンテナンス

カレンダーマスタメンテナンスでは、曜日、会社独自の休日や営業日等を管理する機能を持っています。

※ カレンダーマスタメンテナンスに関しては、アドミニストレータガイドを参照してください。

※ カレンダー表示モジュールは、以下のフォルダに用意されています。

インストールディレクトリ/pages/unit/almanac

なお、カレンダーunit はスクリプト開発モデルにより作成されています。

2.12 ツリー表示モジュール

ツリー表示モジュールを組み込むと階層化されたデータをツリー表示することができ、階層構造の把握やメニューの選択が用意になります。[ページ]メニューの[ページ設定]画面の組織ツリー表示に利用されます。ツリー表示ユニットの詳細については、API リストの「ユニット」-「ツリー表示」を参照してください。

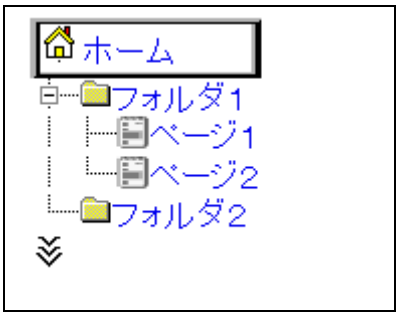
＜ツリー表示モジュールの例＞

＜ソース例＞

```
<%@ page contentType="text/html;charset=Shift_JIS" %>
<%@ page import="jp.co.intra_mart.foundation.page.*" %>
<%@ taglib prefix="imart"
uri="http://www.intra-mart.co.jp/taglib/core/standard" %>
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
  function onClickHomeFunction(){
        window.alert("click home !");
  }
  function onClickFolderFunction(arg){
        window.alert("click folder: " + arg);
  }
  function onClickPageFunction(arg){
        window.alert("click page: " + arg);
  }
</SCRIPT>
<TITLE>Sample JSP</TITLE>
</HEAD>
<BODY>

<%
//「ホーム」を作成する
  TreeRoot home = new TreeRoot();

//「フォルダ」を作成する
  TreeBranch branch1 = new TreeBranch();
//「フォルダ」の表示名を設定する
  branch1.setName("フォルダ１");
//「フォルダ」の説明を設定する
  branch1.setNote("フォルダ１です");
//「フォルダ」の JavaScript に渡される引数を設定する
  branch1.setArgument("フォルダ１");

  TreeBranch branch2 = new TreeBranch();
  branch2.setName("フォルダ２");
  branch2.setNote("フォルダ２です");
  branch2.setArgument("フォルダ２");

//「ページ」を作成する
  TreeLeaf leaf1 = new TreeLeaf();
```


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

```
//「ページ」の表示名を設定する
   leaf1.setName("ページ1");
//「ページ」の説明を設定する
   leaf1.setNote("ページ1です");
//「ページ」の JavaScript に渡される引数を設定する
   leaf1.setArgument("ページ1");

   TreeLeaf leaf2 = new TreeLeaf();
   leaf2.setName("ページ2");
   leaf2.setNote("ページ2です");
   leaf2.setArgument("ページ2");

//「ホーム」に「フォルダ」を登録する
   home.addChildNode(branch1);
   home.addChildNode(branch2);
//「フォルダ」に「ページ」を登録する
   branch1.addChildNode(leaf1);
   branch1.addChildNode(leaf2);
%>
<!--list 属性に home を渡してツリーを表示させる -->
<imart:tree clickHome="onClickHomeFunction"
         clickFolder="onClickFolderFunction"
         clickItem="onClickPageFunction"
         list="<%=home%>"
         type="@DYNAMIC_TREE_OPEN_VIEW"
         rootName="ホーム"
         rootNote="ホームです"/>

</BODY>
</HTML>
```

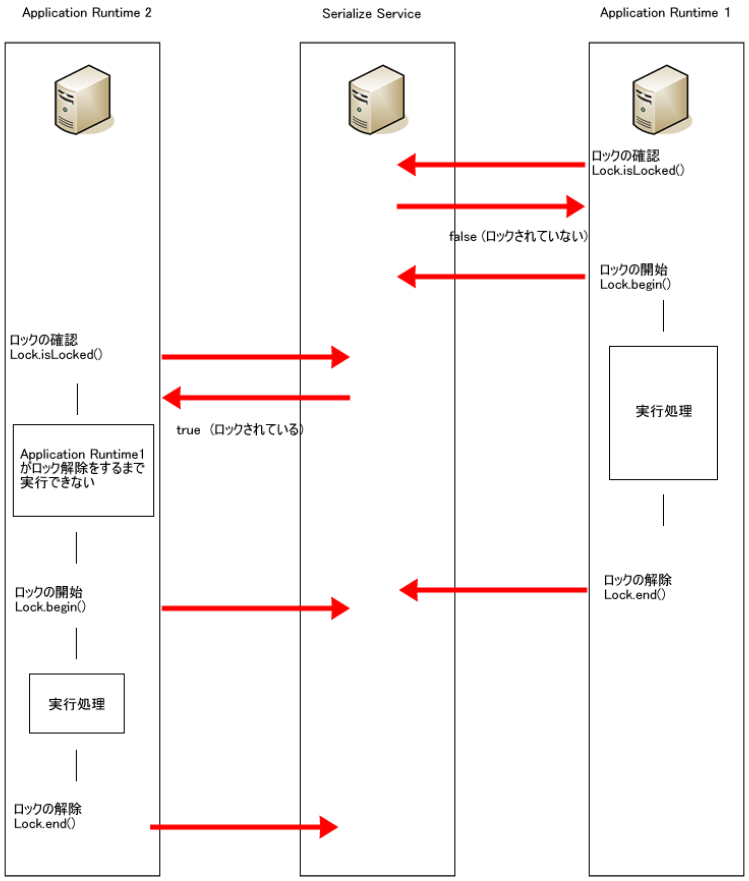


<実行結果>

2.13 アプリケーション・ロック機能

アプリケーション・ロック機能（処理のトランザクション）を実現します。Lock というAPIを利用することで、プログラムの直列処理を行うことができます。また、このAPIは、アプリケーションサーバが分散している場合においても、すべてのサーバで共通的にロックを掛けることができます。詳細は「API リスト」を参照してください。アプリケーション・ロック機能

アプリケーション・ロック機能（処理のトランザクション）を実現します。Lock というAPIを利用することで、プログラムの直列処理を行うことができます。また、このAPIは、アプリケーションサーバが分散している場合においても、すべてのサーバで共通的にロックを掛けることができます（この機能は、Serialization Service を利用します）。



＜アプリケーション・ロックの直列処理＞

＜ソース例＞

```
Lock lock = new Lock("sampleApp");
long timeout = 3L;

if(!lock.isLocked()){
   lock.begin(timeout);

   /***実行処理を記述***

         System.out.println("now processing...");

   *** 処理終了 ***/

   lock.end();
}
```


Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

2.14 一意情報の取得機能

この API は、Application Runtime が分散している場合においても、すべての Application Runtime でユニーク（一意）の情報を取得することができる機能です（この機能は、Server Manager の管理情報を元に各 Application Runtime がシステム一意となるように制御します）。詳細は、「API リスト」の以下のページを参照してください。

jp.co.intra_mart.foundation.service.client.information.Identifier

2.15 製品のカスタマイズ

2.15.1 規定

intra-mart では、製品として提供されたプログラムを自由にカスタマイズして利用することができます。
カスタマイズが可能なプログラムは、オープン・ソースとして提供されているプログラムファイル(***.ini, *.html, *.js, *.properties, *.java** および , ***.jsp** など)すべてが対象となります。
製品のソース・コードに対してカスタマイズをした場合、カスタマイズをしたプログラムおよび動作に関連のあるプログラム群に関して、同パッケージの提供元は動作保証をいたしません。 また、カスタマイズをしたことにより発生した不具合に関しては、サポート対象外となります。

2.15.2 環境移行の手順

カスタマイズしたプログラムを別の環境へ移行(例えば開発機から本番運用機への移行)する場合、以下の手順で移行を行って下さい。

1. 移行先へ intra-mart をインストールします。
2. 移行先環境の intra-mart に対してライセンス登録を行います(アプリケーションを追加インストールしていてソースを展開している場合には、ソース展開も行います)。
3. 移行先環境の intra-mart を停止します。
4. カスタマイズした各プログラムソース(ini, html および js)を元環境から移行先環境へコピーします。
5. 移行先の intra-mart を起動します。

カスタマイズしたプログラムを移行する場合、**Resource Service** の **pages/** ディレクトリ内にある html および js 以外のファイルと **Permanent Data Service** の **treasure/** ディレクトリを上書きコピーしてしまわないように注意して下さい。万一、元環境から移行先環境に対してバイナリファイルの上書きコピーをしてしまいシステムが正常に動作しなくなった場合には、すべてのファイルを削除して移行先への intra-mart のインストールから再度行うようにして下さい。

2.15.3 注意事項

ソースが公開されているプログラムであっても、そのコード中に非公開のAPIを利用している場合があります。これら、非公開APIは予告なく仕様が変更されることがあるので注意が必要です。
製品のソースを直接カスタマイズした場合、カスタマイズをしたソースに関してはバージョンアップ対象外となります。パッチおよびバージョンアップ版のインストールの際にはソースが自動的に上書きされてしまうことがありますので、ご注意下さい。

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

2.16 アクセスセキュリティモジュールを利用しないで画面を構築する方法

2.16.1 概要

intra-mart では、アクセスセキュリティモジュールにとらわれない独自のアプリケーション作成を可能にするソリューションを標準機能として提供しています。

2.16.2 準備(インストール)

intra-mart をインストールガイドにしたがってインストールしてください。
分散システムを構築する場合には、すべてのサーバをインストールします。
インストールが完了したら、すべての Application Runtime の doc/imart/WEB-INF/web.xml にて 以下のフィルターマッピングの設定(<filter-mapping>)をすべて削除してください。
(intra-mart フレームワークの場合、この設定ファイル編集後に再デプロイを行ってください)

- RequestCharacterEncodingFilter
- ResponseCharacterEncodingFilter
- URLAccessFilter
- SessionFilter

2.16.3 プログラムの作成

アプリケーション・プログラムを作成する場合、Servlet および JSP 等の仕様や im-JavaEE Framework の動作仕様にしたがって、それぞれ目的にあったプログラムを記述していきます。
動作仕様等に関しては、**JavaServerPages(Servlet)**について 、**im-JavaEE Framework APIリスト** および製品同梱の im-JavaEE Framework 仕様書を参照して下さい。
なお、im-JavaEE Framework を利用して作成した画面へは、以下URLにてアクセスできます（下記は標準インストールした場合の例です）。

```
http://<ホスト名>[:<ポート番号>]/imart/<アプリケーション ID>-<サービス ID>.service
```

2.16.4 注意事項

アクセスセキュリティモジュールを利用せずに画面を作成した場合、標準で提供されている一部の機能(API)が利用できません。

利用できないAPI(スクリプト開発モデル)

- アクセスセキュリティに関連した機能
  - AccessSecurityManager.*
  - AccountManager.*
  - LicenseManager.*
  - LoginGroupManager.*
  - RoleManager.*
  - UserManager.*
  - Module.client.*
- アプリケーション共通マスタに関連した機能
  - CategoryManager.*
  - CompanyManager.*
  - PrivateGroupManager.*
  - PublicGroupManager.*
  - Procedure.AppCommonUtil.*
- ワークフロー機能
  - AckApplicant.*
  - AckItem.*
  - Acknowledge.*
  - AckUtil.*
- その他
  - Module.alert.*
  - Module.external.*

利用できないAPI(JavaEE 開発モデル)

- アクセスセキュリティに関連した機能
  - jp.co.intra_mart.foundation.security.*
  - jp.co.intra_mart.foundation.acssecurity.*
- アプリケーション共通マスタに関連した機能
  - jp.co.intra_mart.foundation.datastore.*
  - jp.co.intra_mart.foundation.appcomn.*
- ワークフロー機能
  - jp.co.intra_mart.foundation.ackwkf.*

アクセスセキュリティモジュールを利用せずに画面を作成した場合、[install_directory]/conf/imart.xml 内の設定項目の中で有効に機能しないものがあります。

アクセスセキュリティモジュールを利用せずに画面を作成した場合、intra-mart の他のパッケージおよびアドオン・モジュールを追加インストールすることはできません。

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

3 サンプルプログラムの実行

3.1 サンプルのインストール

intra-mart インストール時に「サンプルをインストール」を選択すると、doc/imart または pages/src/sample 以下のディレクトリにサンプルがインストールされます。このサンプルはログイングループのメニューから実際にアクセスし、実行することができます。

スクリプト開発モデル	
サンプルプログラム・メニュー	ソースファイル・パス
［サンプル］-［JavaEE 開発モデル］-[ショッピングカート]	pages/src/sample/item_sample/standard
［サンプル］-［JavaEE 開発モデル］-[ショッピングカート[mskat]]	doc/imart/sample/item_sample/maskat/script
［サンプル］-［JavaEE 開発モデル］-[グラフ]	pages/src/sample/graph/
［サンプル］-［JavaEE 開発モデル］-[ファイル操作]	pages/src/sample/filer/
［サンプル］-［JavaEE 開発モデル］-[メール送信]	pages/src/sample/mail/

※ ［サンプル］-［JavaEE 開発モデル］- [ショッピングカート[maskat]]はマスカットを使用しているため、UTF-8 の環境のみ実行できます。マスカットを使用する場合は intra-mart をインストールする際の文字コードに UTF-8 を指定してください。

インストールされるサンプルは、プログラミングガイドで解説しているサンプルソースコードに比べ、実践的で上級者向けです。こちらのサンプルも合わせて活用することで、より深く JavaEE 開発モデルのプログラミングを理解することが出来ます。

3.2 メニューのサンプル実行

右のメニューから[サンプル]-[JavaEE 開発モデル]以下のサンプルメニューを選択します。

下図は[サンプル]-[JavaEE 開発モデル]- [ショッピングカート]の例です。
[デモ]の画面では、サンプルを実行することができます。

<ショッピングカートの画面例 1>

[概要]の画面では、サンプルの目的や概要図等を確認することができます。
サンプルに関する詳細が載っていますので、参照してください。

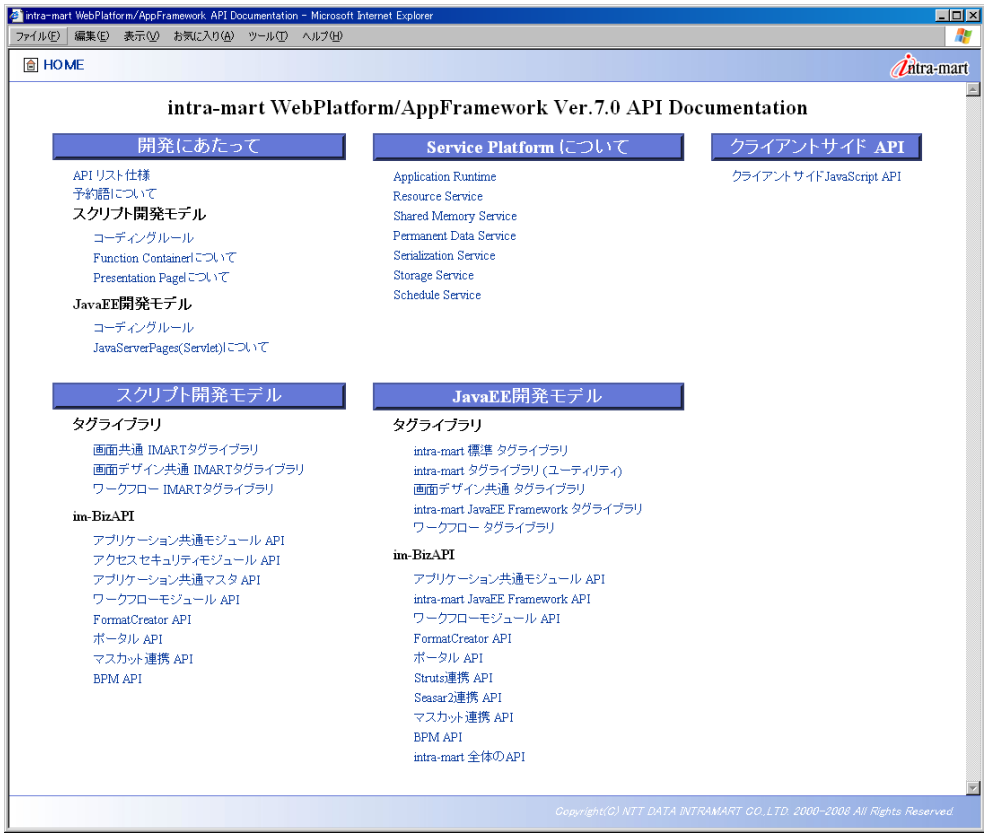
<ショッピングカートの画面例 2>

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

3.3 APIリスト

API リストは、ドキュメント CD の次の場所に格納されていますのでご利用ください。

HTML 形式 ： iwp_iaf/development/iwp_iaf_apilist_v70.zip



<API リスト>

4 Appendix

4.1 メッセージ設定

%Server Manager%/conf/message ディレクトリに、Java のプロパティファイル形式でメッセージを設定します。
ここで設定したメッセージを取得するには、MessageManager クラスを利用します。 MessageManager には、メッセージ ID からメッセージ文字列を取得する getMessage()メソッドが用意されています。このメソッドは、ログインした際のロケールを元にメッセージを取得します。

※ jp.co.intra_mart.foundation.security.message.MessageManager クラスの記述を参照してください。

■ 予約語一覧

以下の用語は、intra-mart の中で予約語として使用されていますので、使用することができません。

- ◆ IMXXX (prefix が “IM(im)”)
- ◆ XXX（prefix が “ （アンダースコア）”）
- ◆ intra-mart API で使用されているクラス、およびグローバル関数名
- ◆ （intra-mart API では大文字を接頭辞として使用しています）
- ◆ JavaScript、Java での予約語

4.2 制限事項

アプリケーション作成時のファイル名および JavaScript 関数名には次のような制限があります。

4.2.1 ファイル名称

ファイル名称に、次の文字は使用できません。

```
¥ / : ; * ? ” < > | & # [ ] ( ) { } (space) (tab)
（全角文字等日本語は使用できません）
```

※ ファイル名とは、プレゼンテーションページ（.html ファイル）とファイルコンテナ（.js ファイル）が対象です。データファイルはこれに含まれません。

4.2.2 ID、コード

intra-mart で提供している機能において、すべての ID、コード（ユーザ ID など）には、以下に示す文字を含めることはできません。

```
¥ / : , ; * ? ’ ″ < > | & # + [ ] ( ) { } (space) (tab)
（全角文字等日本語は使用できません）
```

4.2.3 JS関数

関数名称に、次の文字は使用できません。

```
* < > [ ]
（全角文字等日本語は使用できません）
（その他、JavaScript の仕様に依存します）
```

※ サーバ上で動作する関数に関しての制約です。クライアント上で動作する関数（HTML 内）に関してはこれに含まれません。また、関数だけでなく、その関数の登録名称やメソッド名にもこの制約は適応されます。

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

4.3 JMS(Java Messaging Sservice)

この章では JMS の使用方法について説明します。ここで説明する方法は intra-mart WebPlatform、 アプリケーションサーバは Resin の構成が対象です。

メッセージを送信するクライアントを作成します。ここではサーブレットを使用します。

demo.MySendingServlet.java

```
package demo;

import java.io.IOException;
import java.util.concurrent.BlockingQueue;

import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.webbeans.Named;

public class MySendingServlet extends GenericServlet
{
   @Named("myQueue") private BlockingQueue _queue;

   public void service(ServletRequest req, ServletResponse res)
      throws IOException, ServletException
   {
      String msg = "hello, world";

      _queue.offer(msg);

      System.out.println("Sent: " + msg);
   }
}
```

サーブレットを web.xml に登録します。

/doc/imart/WEB-INF/web.xml

```
・・・
      <servlet>
            <servlet-name>MySendingServlet</servlet-name>
            <servlet-class>demo.MySendingServlet</servlet-class>
      </servlet>

・・・

      <servlet-mapping>
            <servlet-name>MySendingServlet</servlet-name>
            <url-pattern>/test</url-pattern>
      </servlet-mapping>
```

メッセージを表示するリスナを作成します。

demo. MyListener.java

```
package demo;

import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;

public class MyListener implements MessageListener
{
   public void onMessage(Message message)
   {
      ObjectMessage oMsg = (ObjectMessage) message;

      try {
         System.out.println("Received: " + oMsg.getObject());
      } catch (JMSException e) {
         e.printStackTrace();
      }
   }
}
```

http.xml に JMS Queue とリスナを設定します。

/conf/http.xml

```
・・・
   <web-app id="/imart" root-directory="${resin.home}/doc/imart" redeploy-mode="manual">
      <!--
      <session-config>
         <use-persistent-store>true</use-persistent-store>
         <always-save-session>true</always-save-session>
         <save-mode>after-request</save-mode>
      </session-config>
      -->

      <jms-connection-factory uri="resin:"/>
      <jms-queue name="myQueue" uri="memory:"/>

      <ejb-message-bean class="demo.MyListener">
         <destination>#{myQueue}</destination>
      </ejb-message-bean>

   </web-app>
```

作成した class は/doc/imart/WEB-INF/classes 等のクラスパス上に配置してください。

以下の URL にアクセスするとコンソールにメッセージが表示されます。

http://localhost:8080/imart/test

Copyright 2000-2010 株式会社 NTT データ イントラマート All rights Reserved.

intra-mart WebPlatform/AppFramework Ver.7.0
JavaEE 開発モデル プログラミングガイド

2010/11/30 第 4 版

Copyright 2000-2010 株式会社 NTT データ イントラマート
All rights Reserved.

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