intra-mart WebPlatform/AppFramework Ver.7.0

im-JavaEE Framework チュートリアル

2011/06/30 第 3 版

＜＜　変更履歴　＞＞

変更年月日	変更内容
2008/07/07	初版
2010/11/30	第 2 版 「2.7.1 入力エラーチェック」誤植を修正 「2.8.1.3 イベントフレームワークの処理の実装」誤植を修正
2011/06/30	第 3 版 表紙のレイアウトを修正

<< 目次 >>

1 はじめに 1
1.1 本書の目的 1
1.2 対象読者または前提条件 1
1.3 準備 2
1.4 ディレクトリ構成 3
2 簡易掲示板の作成 4
2.1 サンプルアプリケーションの仕様 4
2.2 「はりぼて」をつくる 4
2.3 メニューの登録 9
2.4 掲示板情報モデルクラスの作成 14
2.5 登録画面の作成 18
2.5.1 サービスコントローラを呼び出すJSPプログラムを作ってみる 18
2.5.2 サービスコントローラを作ってみる 24
2.6 参照画面の作成 32
2.6.1 ヘルパーBeanを呼び出すJSPを作ってみる 32
2.6.2 ヘルパーBeanを作ってみる 37
2.6.3 ヘルパーBeanから情報を受け取る 42
2.7 掲示板で入力チェックをしてみよう 48
2.7.1 入力エラーチェック 48
2.7.2 入力エラーチェック画面を作成してみる 54
2.8 掲示情報をデータベースに格納する 60
2.8.1 登録画面のイベントフレームワークを実装する 62
2.8.2 登録画面のデータフレームワークを実装する 78
2.9 掲示情報をデータベースから取り出して表示する 85
2.9.1 参照画面にイベントフレームワークを実装する 86
2.10 処理概要 104
3 Strutsとの連携 111
3.1 Struts連携の概要 111
3.2 web.xmlの編集 112
3.3 メニューへの登録 113
3.4 struts-tutorial_plus.xmlの編集 114
3.5 Struts用のJSPファイルの編集 116
3.6 ActionFormクラスの編集 118
3.7 Actionクラスの編集 121
4 おわりに 125

1 はじめに

1.1 本書の目的

本書は、掲示板サンプルアプリケーションの開発を通して、JavaEE フレームワークによる開発のエッセンスを読者に伝えることを目的とします。本書を通読することで、簡単な JavaEE フレームワークアプリケーション開発の流れを理解することができ、同規模のアプリケーションを作成することができるようになるでしょう。

また、本書は JavaEE フレームワークの仕組みについて、概説的な説明をおこないます。網羅的な説明ではありませんが、大枠の概念については理解できるでしょう。

一方で、本書に含まれない内容もあります。それは以下のとおりです。

- Java 言語の説明
- JSP および Servlet などの ServerSideJava プログラミングに関する説明
- 上記 2 点で使用される API に関する説明
- モデリング言語(主に UML)に関する説明
- JavaEE 技術に関する説明

対象読者の節でも触れますが、これらの知識は本書を読み進めるための前提条件にもなります。

また、本書で作成するアプリケーションはあくまでも、JavaEE フレームワークのアプリケーション作成の流れを理解することに主眼をおいていますので、必ずしもベストなコーディング方法とはいえない方法もあえて取っている個所があります。あくまでも、サンプルとしての位置付けでとらえるようにしてください。

1.2 対象読者または前提条件

- JavaEE フレームワークは Java 言語で記述しますので、Java 言語に関する理解は必須です。とくに、基本文法とオブジェクト指向、クラス、メソッドのオーバーライド、継承、実装などの概念は必須の前提知識になります。これらに関しては、市販でよい本がたくさんありますので、そちらを参照してください。

- 本書は、JSP、Servlet 等の ServerSideJava プログラミング全般の解説を目的とするものではありません。したがって本書を読み進めるために、基本的な JSP 及び Servlet に関する理解が必要になります。それほど高度な知識が必要になるわけではありませんが、市販されている JSP 及び Servlet の解説書を一度通読することができる読者を対象としています(また、それを強くお勧めします)。

- java.*、javax.*の API に関する説明は基本的に省きますので、Sun MicroSystems 社の提供する JavaSE および JavaEE の API リストを用意して、それらの内容を参照して理解しながら読み進めることができる読者を対象とします。

- 一部のクラスの仕様説明に、UML(モデリング言語)のクラス図を使用しています。非常に簡単なものですが、その図を見て感覚的に理解できることが前提条件となります。

1.3 準備

まず、最初に掲示板サンプルアプリケーションを実行するための準備をしましょう。

本書で作成するアプリケーションは im-JavaEE フレームワークを用いたアプリケーションなので、intra-mart WebPlatform か intra-mart AppFramework が必要です。バージョンは Ver7.x を前提としています。開発用の環境に関しましては、それぞれの製品に付属するインストールマニュアルをもとに、開発用の環境を作成してください。

ここからは、ApplicationRuntime サービスのドキュメントルート上での作業になります。この節のフォルダ名に関しては、ApplicationRuntime サービスのドキュメントルート上以下の～フォルダという形で置き換えて読みすすめてください。以下では、ApplicationRuntime サービスのドキュメントルートを C:/imart/doc/imart として説明を進めていきます。

ま CD-ROM に同梱されている im_javaee_framework_tutorial_src.zip をずローカルに解凍します。
次に C:/imart/doc/imart フォルダの下に、解凍した src/1-3/内の全てのフォルダをそのまま上書きコピーします。Windows の場合、コピーするときに上書きしますか？というダイアログが出るかもしれませんが、そのまま OK で上書きコピーしてしまってもかまいません。(以下同様です)

notice というフォルダが作成され、その中に JSP プログラムがコピーされます。これらのファイルは、これから作成するアプリケーションで画面を表示するために必要になります。

WEB-INF/classes/notice/conf ディレクトリ内に、

service-config-notice.xml

という xml ファイルがコピーされています。この xml ファイルはサンプルアプリケーションにおいて、次の遷移先のページを決定したり、サービスフレームワークにおいて、どのクラスがよばれるかということを決定するためのファイルです。あとで詳しく見ていきますが、これらの xml ファイルを修正するだけで、画面遷移や処理の流れを変更または修正したりできることが、JavaEE フレームワークの強みになります。

さらに、classes フォルダの下に notice というフォルダができて、その下にさらにフォルダがきられていくつかの java ファイルがコピーされます。これらは、JavaEE フレームワークにおいて実際に動作するクラスで、xml ファイルの指定に従い、JavaEE フレームワークから呼び出されて処理を実行するためのクラス群になります。実際の JavaEE フレームワークの開発においては、これらのクラスを作成していくことが主な作業になります。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

1.4 ディレクトリ構成

このチュートリアルで利用する WebPlatform と、使用するテンプレートの主なディレクトリ構成について以下にまとめます。

【intra-mart WebPlatform Ver.7.x】

```
C:¥imart                                  ・・・WebPlatform Ver7.x インストールディレクトリ(スタンドアロン)
├ doc
  ├ imart                                 ・・・ApplicationServer ルート
    ├ notice                              ・・・サンプル用 JSP プログラム保存
    ├ WEB-INF
        ├ classes                         ・・・クラスファイル保存
            ├ notice                      ・・・サンプル用 Java プログラム・クラスファイル保存
                ├ conf                    ・・・JavaEE フレームワーク設定ファイル保存
        ├ lib                             ・・・クラスライブラリ保存
        ├ web.xml                         ・・・Web アプリケーション設定ファイル
```

【チュートリアル用テンプレートプログラム】

```
├ src
  ├ 1-3
                                          ・・・1.3 章「準備」関連ファイル
    ├ classes
      ├ notice                            ・・・関連 Java プログラム
        ├ conf                            ・・・JavaEE フレームワーク設定ファイル
    ├ notice                              ・・・関連 JSP プログラム
  ├ 2-X                                   ・・・章毎のプログラム・設定ファイル
  ├ complete                              ・・・完成プログラム
├ notice_ddl.sql                          ・・・サンプルテーブル作成 SQL ファイル
├im_javaee_framework_tutorial_v70.pdf     ・・・チュートリアルテキスト
```

＊各ステップの Java プログラムを ApplicationRuntime サービスに配備する場合、既にコンパイル済みの class ファイルは削除してください。

2 簡易掲示板の作成

2.1 サンプルアプリケーションの仕様

ここでは、サンプルアプリケーションとして「掲示板」を作成していくことで、JavaEE フレームワークを使った開発に慣れていきます。

掲示板といえば、通常、掲示内容を登録する「登録画面」と、掲示された内容を表示する「参照画面」があります。掲示に対してコメントをつけることのできる掲示板システムもありますが、ここでは、サンプルアプリケーションということで、単純に投稿してそれを参照できるアプリケーションとしてシステムを定義します。コメントをつけられると掲示板としては機能的になりますが、今回はあくまでもアプリケーション作成の概要を学ぶということで、今回は触れないこととします。

2.2 「はりぼて」をつくる

さて、まずはシステムの概要をつかむために、単純な JSP で「はりぼて」を作ってみましょう。実際の画面を作って眺めてみたほうが、システムの概要をつかみやすいと思います。
JSP のプログラムとはいっても、基本は HTML タグを使って画面レイアウトを作って行くことには変わりありません。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

出来上がった画面イメージとは以下のようになります。

【登録画面】

図 2.2-a

【参照画面】

図 2.2-b

登録画面の JSP ソースは以下のとおりです。

Source 2-2

<C:/imart/doc/imart/notice/notice_regist.jsp>

```
<%@ page contentType="text/html; charset=Windows-31J" pageEncoding="Windows-31J" %>

<HTML>

<HEAD>
    <TITLE>個人掲示板登録画面</TITLE>
</HEAD>

<BODY>

<!-- タイトル-->
<TABLE bgcolor='#99cc66' width='100%'>
    <TR><TD>
        <FONT color='white' size="+1"><b>  個人掲示板 -登録-  </b></FONT>
    </TD></TR>
</TABLE>

<BR>
<BR>

<CENTER>

<FORM>

<TABLE border="1">
    <TR>
        <TH bgcolor="#99cc66" align="center">
            タイトル
        </TH>
        <TD>
            <INPUT name="title" type="text" size="50">
        </TD>
    </TR>
    <TR>
        <TH bgcolor="#99cc66" align="center">
            内容
        </TH>
        <TD>
            <TEXTAREA name="content" rows="9" cols="50"></TEXTAREA>
        </TD>
    </TR>
</TABLE>

<BR>

<INPUT type="submit" value=" 登 録 ">

</CENTER>

</FORM>

</BODY>
</HTML>
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

参照画面のソースは以下のとおりです。

Source 2-2

<C:/imart/doc/imart/notice/notice_view.jsp>

```
<%@ page contentType="text/html; charset=Windows-31J" pageEncoding="Windows-31J" %>


<HTML>

<HEAD>
    <TITLE>個人掲示板参照画面</TITLE>
</HEAD>

<BODY>

<!-- タイトル-->
<TABLE bgcolor='#99cc66' width='100%'>
    <TR><TD>
        <FONT color='white' size="+1"><b>  個人掲示板 -参照-  </b></FONT>
    </TD></TR>
</TABLE>

<BR>
<BR>

<CENTER>

<FORM>

<TABLE border="1" width="80%">
    <TR>
        <TH bgcolor="#99cc66">
            タイトル
        </TH>
        <TD align="left" colspan="3">
            ここにタイトルが入ります
        </TD>
    </TR>
    <TR>
        <TH bgcolor="#99cc66">
            作成者
        </TH>
        <TD>
            管理者
        </TD>
        <TH bgcolor="#99cc66">
            登録日
        </TH>
        <TD>
            2002/10/01 16:00:00
        </TD>
    </TR>
    <TR>
        <TH bgcolor="#99cc66">
            内容
        </TH>
        <TD align="left" colspan="3">
            ここに内容が入ります。<BR>
            ちゃんと見えてますか？
        </TD>
    </TR>
</TABLE>

</CENTER>

</FORM>
```

```
</BODY>
</HTML>
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

2.3 メニューの登録

JSP プログラムの作成が完了したら、intra-mart のメニュー情報への登録を行います。
ここでは JavaEE フレームワークで作成したアプリケーションを intra-mart のメニューに登録する方法を学びます。

intra-mart の起動が完了したらブラウザを起動し、管理者ユーザで intra-mart にログインします。
URL：http：//サーバ IP アドレス：ポート番号/imart/default. manager
＊default：ログイングループ ID

メニューフレームより
[ログイングループ管理]-[メニュー設定]
画面をクリックしてください。

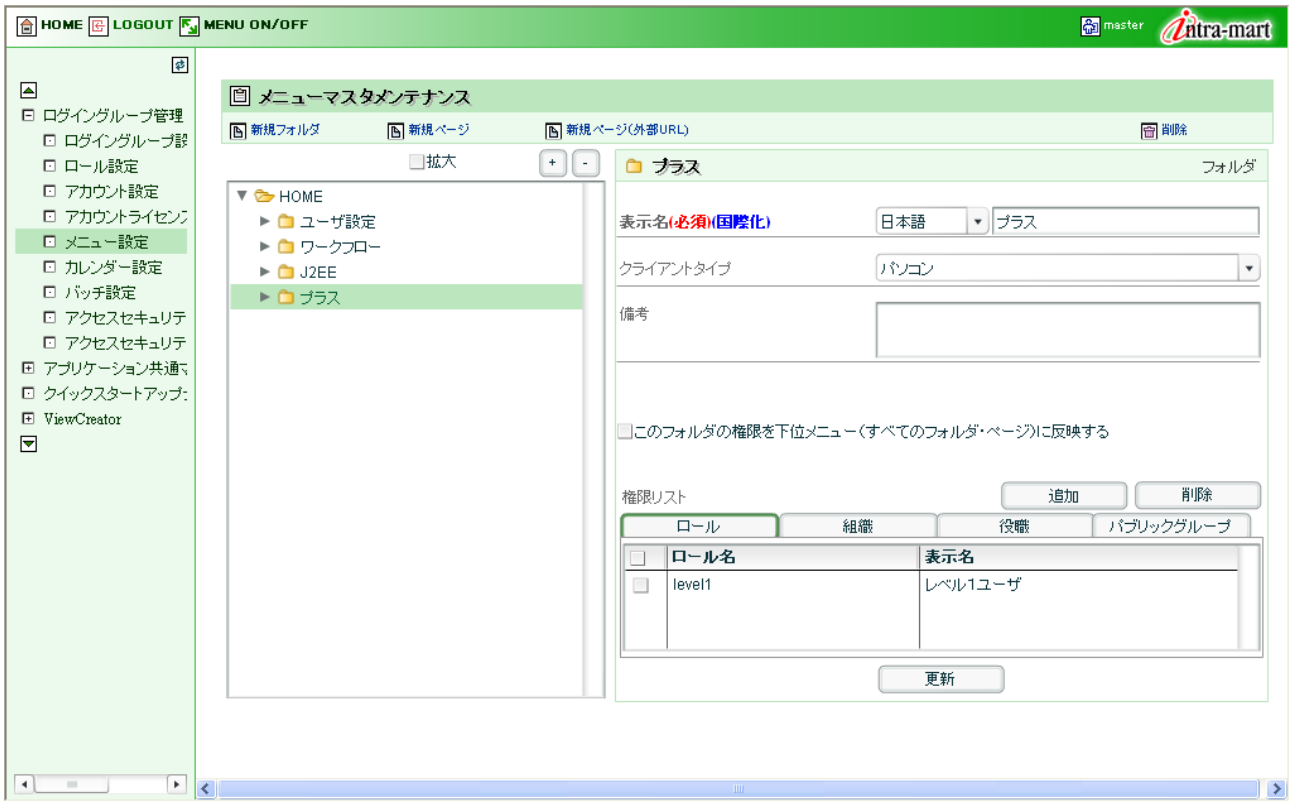

図 2.3-a

今回登録する情報は、「登録画面」と「参照画面」の2つです。
まずは、フォルダを登録してみます。
メニュー画面にある「新規フォルダ」をクリックし、下記内容を登録してください。

表示名（日本語）	プラス
表示名（英語）	tutorial_plus
クライアントタイプ	パソコン
備考	(任意)
権限リスト（ロール）	（このフォルダにアクセスするユーザが保持するロール）

次に掲示板登録画面をページ登録します。

メニュー画面にあるメニュー情報表示画面(ツリー形式)の「最新情報」ボタンをクリックし、先ほど登録したフォルダ(例. プラス)をクリックしてください。
該当フォルダを選択した状態で「新規ページ」をクリックし、以下のページ情報を登録してください。

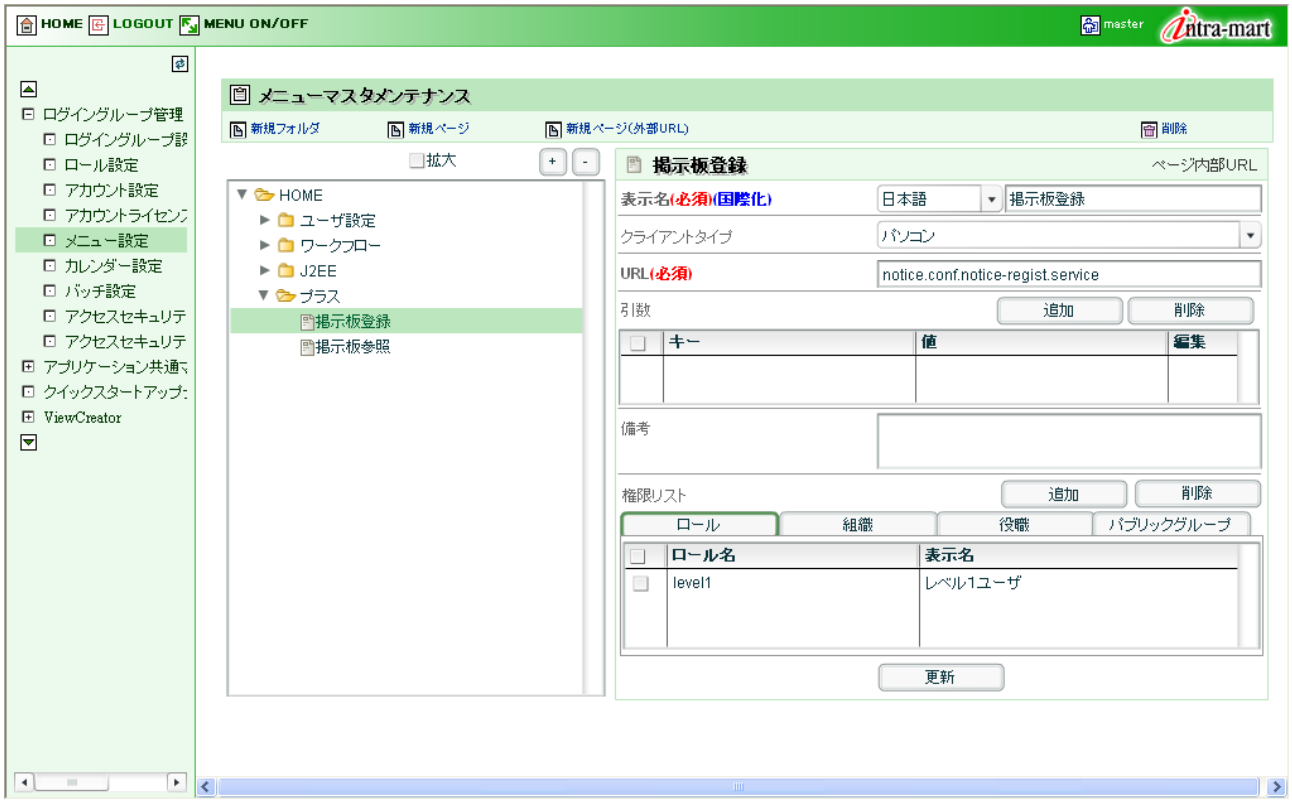

図 2.3-b

表示名(日本語)	掲示板登録
表示名(英語)	regist
クライアントタイプ	パソコン
URL	notice.conf.notice-regist.service
引数	(任意)
備考	(任意)
権限リスト(ロール)	(このフォルダにアクセスするユーザが保持するロール)

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

次に掲示板参照画面をページ登録します。

表示名（日本語）	掲示板参照
表示名（英語）	view
クライアントタイプ	パソコン
URL	notice.conf.notice-view.service
引数	(任意)
備考	(任意)
権限リスト（ロール）	（このフォルダにアクセスするユーザが保持するロール）

登録処理が完了したら、1 度ログアウトして再ログインしてください。
再ログインすると、メニューフレームに登録した情報が表示されます。各項目をクリックし、メインフレームに作成したアプリケーションが表示されることを確認してください。

ここで、前章でコピーした service-config-notice.xml の設定内容を確認してみましょう。

Source 1-3

<C:/imart/doc/imart/WEB-INF/classes/notice/conf/service-config-notice.xml>

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

    <service>
        <service-id>regist</service-id>
        <next-page>
            <page-path>/notice/notice_regist.jsp</page-path>
        </next-page>
    </service>

    <service>
        <service-id>view</service-id>
        <next-page>
            <page-path>/notice/notice_view.jsp</page-path>
        </next-page>
    </service>

</service-config>
```

1 つ目の設定内容を例に解説してみます。
これは、「アプリケーション ID＝" notice.conf.notice"、サービス ID＝"regist"の要求が来た場合、"/notice/notice_regist.jsp "に遷移する」ということを意味します。

アプリケーション ID は以下のようになります。
コンテキストパス（ApplicationRuntime）/WEB-INF/classes からのパッケージパス・・・"notice. conf. "
+ service-config-キーワード・・・notice
＊intra-mart Ver5.x からは従来のバージョンとは異なり、コンテキストパス（ApplicationRuntime）/WEB-INF/classes 以下の任意のディレクトリ内にプロパティファイル（xml）を定義することが可能となりました。

サービス ID は以下のように指定します。
（ファイル「service-config-notice.xml」中）

```
<service-id>サービス ID</service-id>
<next-page>
  <page-path>表示ページのパス</page-path>
</next-page>
```

上記は遷移ページの設定です。
ここで「表示ページのパス」とは、Web アプリケーションのコンテキストパス（Web アプリケーションのルートパス）以下のファイル名を意味します。

本チュートリアルでは「<context path>」がコンテキストパス(C:/imart/doc/imart)となるので、
その下の「/notice/notice_regist.jsp」を指定します。このとき、ファイル名は必ず"/"から開始してください。

「メニューの登録」では intra-mart のメニューから作成した JSP へ遷移するためのメニュー登録を行いました。
このような設定を行うと、intra-mart 内部では JavaEE Framework 内部の ServiceServlet というサーブレットに遷移しています。
このサーブレットは、<context path>/WEB-INF/web.xml の中で以下のように定義されています。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

```
<servlet>
  <servlet-name>ServiceServlet</servlet-name>
  <servlet-class>jp.co.intra_mart.framework.base.service.ServiceServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
・・・
<servlet-mapping>
  <servlet-name>ServiceServlet</servlet-name>
  <url-pattern>*.service</url-pattern>
</servlet-mapping>
```

ここで定義されている ServiceServlet は、JavaEE Frameork が標準で提供するコンポーネントです。
メニュー登録画面で URL に「*.service」というキーワードが設定されると、ServiceServlet は URL に設定されたキーワード（アプリケーション ID・サービス ID）を基に、プロパティ（xml）に設定されている開発者が作成したコンポーネントを実行します。

ServiceServlet の役割をまとめると以下のようになります。

1　アプリケーション ID（notice.conf.notice）に該当する設定（xml）ファイル（service-config-notice.xml）を選択します。

2　設定（xml）ファイルからサービス ID（regist）に該当するプロパティ（遷移ページ）の情報を取得します。

3　取得したページ（/notice/notice_regist.jsp）に遷移します。

2.4 掲示板情報モデルクラスの作成

ここまでの話は、普通の HTML プログラムの話です。
この状態は、直接 JSP ファイルの中に掲示内容やタイトルを書いていますので、当然のことながら、登録画面で掲示を登録しても、表示画面で表示されません。

最終的には、「掲示板機能を実現する」掲示板を作成することが目標です。
「掲示板機能を実現する」掲示板を作成する第一歩として、この HTML プログラムを JSP 化していきましょう。

最初に、掲示情報をあらわすためのモデルを作ってみましょう。掲示情報は、その実体として次の要素を含んでいます。

タイトル
登録日付
登録者
内容

これらの情報をもったモデルクラスを作成します。JavaEE フレームワークではクラスがどうしても増えてしまうので、パッケージ名をつけておいたほうがいいでしょう。ここでは、パッケージ名を「notice.model.object」としておきます。
まず簡単なクラス図を書いてみます。

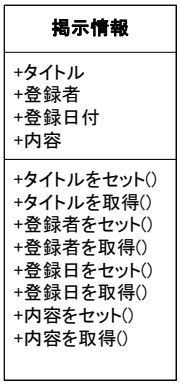

図 2.4-a

クラス間の関係という意味では、他のクラスとの関係も何もありませんので、理解しやすいと思います。基本的には掲示情報の入れ物という理解で十分だと思います。
次に、掲示板情報クラスを作成しましょう。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 2-4

<C:/imart/doc/imart/WEB-INF/classes/notice/model/object/NoticeInf.java>

```
package notice.model.object;

import java.util.Date;

/**
 * @author nttdata intra-mart
 *
 * 掲示板情報を表現するためのクラス
 */
public class NoticeInf {
   private String title;
   private String author;
   private Date registDate;
   private String content;

   public NoticeInf(){
         this.title = null;
         this.author = null;
         this.registDate = null;
         this.content = null;
   }

   /**
    * 登録者を返却します
    * @return String 作成者
    */
   public String getAuthor() {
         return author;
   }

   /**
    * 内容を返却します
    * @return String 内容
    */
   public String getContent() {
         return content;
   }

   /**
    * 登録日付を返却します
    * @return Date 登録日付
    */
   public Date getRegistDate() {
         return registDate;
   }

   /**
    * タイトルを返却します
    * @return String タイトル
    */
   public String getTitle() {
         return title;
   }

   /**
    * 登録者をセットします
    * @param セットする登録者
    */
   public void setAuthor(String author) {
         this.author = author;
   }

   /**
```

```
     * 内容をセットします
     * @param セットする内容
     */
    public void setContent(String content) {
        this.content = content;
    }

    /**
     * 作成日付をセットします
     * @param セットする作成日付
     */
    public void setRegistDate(Date registDate) {
        this.registDate = registDate;
    }

    /**
     * タイトルをセットします
     * @param セットするタイトル
     */
    public void setTitle(String title) {
        this.title = title;
    }
}
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

メンバ変数として、タイトルと作成日付と登録者と内容を持っていて、それらの変数に対する setter メソッドと getter メソッドがあるだけのクラスです。Eclipse など、Java 統合開発環境には自動的に setter メソッドと getter メソッドを生成してくれる開発環境もありますので、そのような開発環境を使えばこのようなクラスの作成もさらに楽になると思います。

この作成した java ファイルは、
%ApplicationRuntime サービスのドキュメントルート%/WEB-INF/classes/notice/model/object
＊当資料においては、ApplicationRuntime サービスのドキュメントルート = C:/imart/doc/imart となります。
の下にコピーしてください。

メモ

スケルトンを利用すると、開発効率を向上させることが可能です。
ModelObject のスケルトンは、intra-mart ドキュメントメディア内の下記ファイルです。
skeleton/domain/category/model/object/XXXModelObject.java

2.5 登録画面の作成

さて、それでは画面の作成のほうに戻りましょう。まずは登録画面を作成します。

2.5.1 サービスコントローラを呼び出すJSPプログラムを作ってみる

先ほど作った登録画面の HTML プログラムをサービスコントローラを呼び出す JSP ファイルに拡張してみます。先ほどの notice_regist.jsp を開いてください。

この画面では、初期表示時には何も処理をおこないません。入力用のテキストフィールドおよびテキストエリアを表示して、登録ボタンを配置しているだけです。登録ボタンが押されたときにはじめて、掲示情報登録処理という処理が実行されます。そこで、この登録処理を JavaEE フレームワークのサービスフレームワークで実装することを考えます。

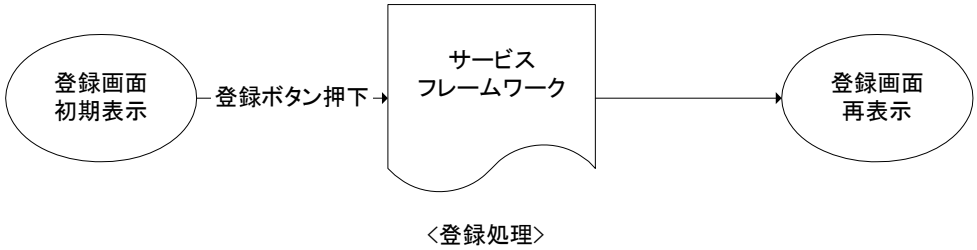

図 2.5-a

このときに考えなければならないのは、掲示情報をどこに保存するかということです。保存場所としては、いろいろ考えられますが、ここではまずイントラマートのセッション情報に情報を保存することを考えてみましょう。もちろんセッション情報なので、イントラマートからログアウトしてしまうと情報は消えてしまいますが、最初は簡単に作成できる仕様のほうから作ってくことにしましょう。

では、さっそく実際の実装作業をすすめていきます。

まず、JavaEE フレームワークに対して、どのアプリケーションの中で呼び出されるどのサービスかを決定するために、名前をつけましょう。JavaEE フレームワークでは、アプリケーション ID とサービス ID をつけることでサービスを区別します。他のアプリケーションやサービスと重複しなければなんでもいいですが、ここではアプリケーション名を notice.conf.notice、掲示情報登録処理のサービス名を notice_regist としましょう。

アプリケーションID	サービスID	概要
notice.conf.notice	notice_regist	掲示情報を登録します。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

JSP ファイルからサービスフレームワークのクラスを呼び出せるようにするために、xml ファイルを用意しましょう。このとき、ここで決定したアプリケーションIDとサービスIDを使って、xml ファイルのパス&ファイル名とその中に記述するサービスの名称が決定します。つまり、

パス：notice.conf

プロパティファイル名：service-config-notice.xml

というファイルに、たとえば次遷移先のページを記述したいときには

```
<service>
    <service-id>サービス ID</service-id>
    <next-page>
        <page-path>遷移ページパス</page-path>
    </next-page>
</service>
```

というように記述します。

では xml ファイルを実際に編集してみましょう。ファイル名は、service-config-notice.xml です。内容は、以下のようになります。

Source 2-5

<C:/imart/doc/imart/WEB-INF/classes/notice/conf/service-config-notice.xml >

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

    <service>
        <service-id>regist</service-id>
        <next-page>
            <page-path>/notice/notice_regist.jsp</page-path>
        </next-page>
    </service>

    <service>
        <service-id>view</service-id>
        <next-page>
            <page-path>/notice/notice_view.jsp</page-path>
        </next-page>
    </service>

  <service>
      <service-id>notice_regist</service-id>
          <controller-class>
              notice.controller.service.NoticeRegistServiceController
          </controller-class>
          <next-page>
              <page-path>/notice/notice_regist.jsp</page-path>
          </next-page>
  </service>

</service-config>
```

網掛け部分が今回追加で設定する内容です。
編集が終わったら、%アプリケーションサーバのドキュメントルート%/WEB-INF/classes/notice/conf ディレクトリにファイルを保存してください。

このプロパティを設定することで、掲示板登録画面で登録ボタンが押下された後に遷移するページを指定することができます。つまり、登録ボタンの遷移先としてアプリケーションID「notice.conf.notice」、サービスID「notice_regist」が指定したときに実行されるサービスと、次のページ遷移先をこれで設定できたわけです。

次にこの設定にしたがって掲示板登録画面からサービスを呼び出すことができるように、さきほどのnotice_regist.jsp を編集します。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 2-5

<C:/imart/doc/imart/notice/notice_regist.jsp>

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

<HTML>

<HEAD>
    <TITLE>個人掲示板登録画面</TITLE>
</HEAD>

<BODY>

<!-- タイトル-->
<TABLE bgcolor='#99cc66' width='100%'>
    <TR><TD>
        <FONT color='white' size="+1"><b>  個人掲示板 -登録-  </b></FONT>
    </TD></TR>
</TABLE>

<BR>
<BR>

<CENTER>

<imartj2ee:Form application="notice.conf.notice" service="notice_regist">

<TABLE border="1">
    <TR>
        <TH bgcolor="#99cc66" align="center">
            タイトル
        </TH>
        <TD>
            <INPUT name="title" type="text" size="50">
        </TD>
    </TR>
    <TR>
        <TH bgcolor="#99cc66" align="center">
            内容
        </TH>
        <TD>
            <TEXTAREA name="content" rows="9" cols="50"></TEXTAREA>
        </TD>
    </TR>
</TABLE>

<BR>

<INPUT type="submit" value=" 登 録 ">

</CENTER>

</imartj2ee:Form>

</BODY>
</HTML>
```

網掛けの部分が今回修正を加える部分です。

<%@ page contentType="text/html; charset=Windows-31J" pageEncoding="Windows-31J" %>

という部分は、この JSP ページが生成するレスポンスの MIME 属性を指定します。詳しくは、JSP の解説書などで「page ディレクティブ」の「contentType 属性」という項を参考にしてください。

```
<%@ taglib prefix="imartj2ee" uri=" http://www.intra-mart.co.jp/taglib/core/framework" %>
```

は、taglib ディレクティブとよばれるもので、JavaEE フレームワークのタグライブラリを使用できるように、「imartj2ee」という名前で宣言しています。

上の 2 行は、JavaEE フレームワークのプログラムを作成する上で、大部分のページで指定することになると思いますので、おまじないのようなものと思っていただいてかまわないと思います。もちろん、意味をわかって使用するにこしたことはありませんが。

```
<imartj2ee:Form application="notice.conf.notice" service="notice_regist">
……
</imartj2ee:Form>
```

は、JavaEE フレームワークのタグライブラリに含まれる Form というタグで(taglib ディレクティブにおいて、imartj2ee という prefix をつけていることに注意してください)、HTML の FORM タグに相当します。HTML の FORM タグに含まれるすべての属性を使用することができ、さらに application という属性で JavaEE フレームワークのアプリケーション名を、service という属性でサービス名を指定することができます。この application 属性と service 属性を指定することで、この Form がサブミットされたときに JavaEE フレームワークのサービスフレームワークが呼び出され、そのときに実行されるサービスの種類を指定することができます。

つまりここでは、「FORM がサブミットされたときに、アプリケーション名が ”notice.conf.notice”、サービス名が ”notice_regist” というサービスフレームワークを呼び出しなさい」という指定をしていることになります。

ここまではよろしいでしょうか？

それでは、ここで一度画面に戻って、これまでに修正したソースがどのように動くかを見てみましょう。

メニューから掲示板登録画面を表示して、 登録ボタンを押してみましょう。押してすぐに、掲示板登録画面が再表示されたと思います。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

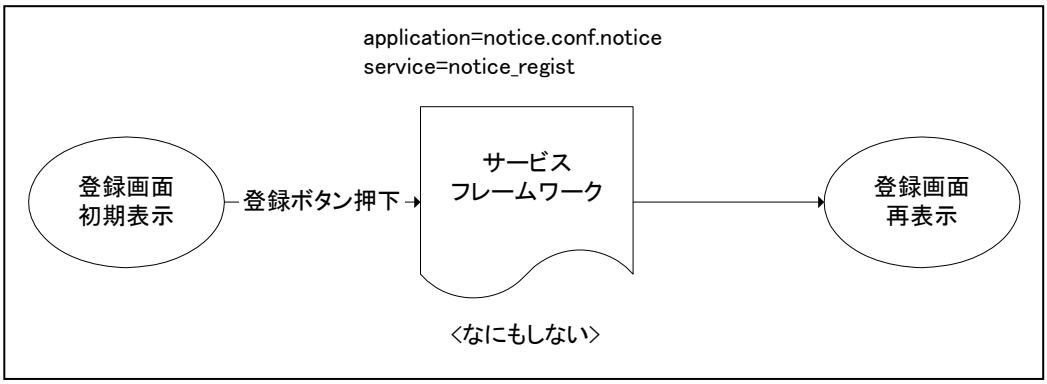

図 2.5-b

今の状態は、上記の図のような状態です。掲示板登録画面の登録ボタンが押されたときに、application 名が”notice.conf.notice”、サービス名が”notice_regist”というサービスが呼び出されています。notice_regist サービスでは、何も処理をせずに(まだ実際の処理はコーディングしていませんから)登録画面を再表示しています。

以上で、JSP プログラムからサービスフレームワークを呼び出すことができました。
次の節では、サービスフレームワークの中に実際に掲示情報の登録処理を書いていくことにしましょう。

2.5.2 サービスコントローラを作ってみる

さて、いよいよ実際の処理の部分を記述していきます。サービスフレームワークの中で、処理を記述するためにはJavaEE フレームワークの jp.co.intra_mart.framework.base.service.ServiceControllerAdapter というクラスを継承したクラスを作成します。皆さんが作成するクラスのパッケージはもちろん任意なのですが、ここでは notice.controller.service というパッケージ名にしましょう。

さらに、サービスフレームワークで実行したい処理プログラムを記述するには ServiceControllerAdapter クラスの service()メソッドをオーバーライドします。

ServiceControllerAdapter クラスの service()メソッドは、ApplicationException と SystemException という例外をスローします。

まずは、これらの外枠だけを記述したクラスを作成します。
本チュートリアルでは、1-3 章 準備 で既にテンプレートとして配備されています。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 1-3

<C:/imart/doc/imart/WEB-INF/classes

/notice/controller/service/NoticeRegistServiceController.java>

```
package notice.controller.service;

import jp.co.intra_mart.framework.base.service.ServiceControllerAdapter;
import jp.co.intra_mart.framework.base.service.ServiceResult;
import jp.co.intra_mart.framework.system.exception.ApplicationException;
import jp.co.intra_mart.framework.system.exception.SystemException;

/**
 * @author NTTDATA intra-mart
 *
 * 掲示情報を登録するサービスコントローラです。
 */
public class NoticeRegistServiceController extends ServiceControllerAdapter {

   /**
    * デフォルトコンストラクタ
    */
   public NoticeRegistServiceController() {
         super();
   }

  /**
   * 入力に対する処理を実行します。
   *
   * @return 処理結果
   * @throws SystemException 処理実行時にシステム例外が発生
   * @throws ApplicationException 処理実行時にアプリケーション例外が発生
   */
   public ServiceResult service() throws SystemException, ApplicationException {
         return null;
   }

}
```

上で説明したものを、そのままクラスにしたソースです。編集が終わったら、
%アプリケーションサーバのドキュメントルート%/WEB-INF/classes/notice/controller/service/
の下に保存してください。

次に、service()メソッドの中に、実際の掲示情報登録処理を記述していきましょう。下の図を見てください。

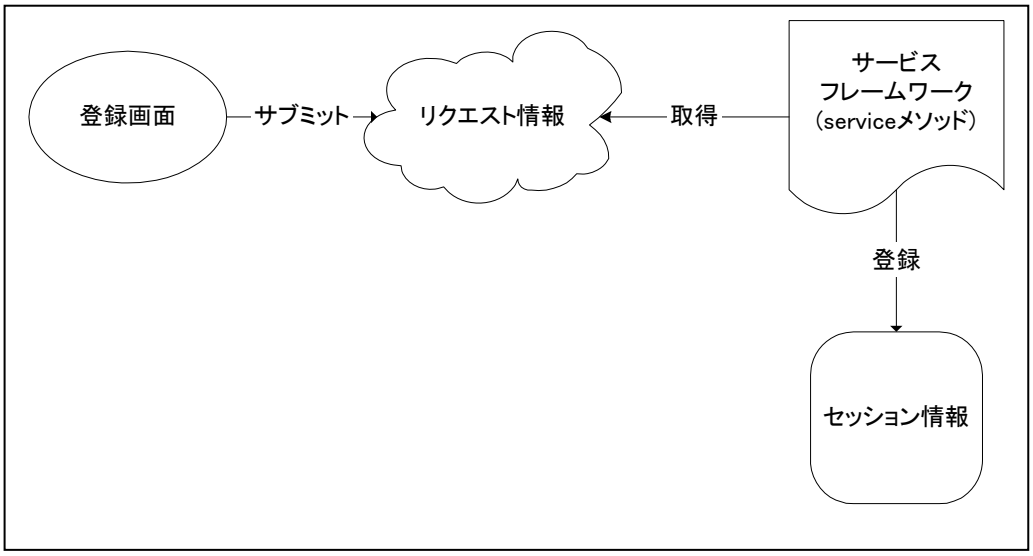

図 2.5-c

この図は、これから service()メソッドの中で記述する処理の流れを概念的に表したものです。まず掲示板登録画面で登録ボタンが押されたとします。すると、入力されたタイトル、内容などの情報がリクエスト情報という形で、サービスフレームワークで取得することができます。service()メソッドではこの受け取ったリクエスト情報をもとに掲示情報を整形して、イントラマートのセッション情報に情報を保存します。保存した後は、処理を終了して、次の画面に遷移します。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 2-5

<C:/imart/doc/imart/WEB-INF/classes

/notice/controller/service/NoticeRegistServiceController.java>

```
package notice.controller.service;

import java.util.Date;
import java.util.HashMap;
import java.util.Vector;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import jp.co.intra_mart.framework.base.service.RequestException;
import jp.co.intra_mart.framework.base.service.ServiceControllerAdapter;
import jp.co.intra_mart.framework.base.service.ServiceResult;
import jp.co.intra_mart.framework.base.session.SessionObject;
import jp.co.intra_mart.framework.system.exception.ApplicationException;
import jp.co.intra_mart.framework.system.exception.SystemException;

/**
 * @author NTTDATA intra-mart
 *
 * 掲示情報を登録するサービスコントローラーです。
 */
public class NoticeRegistServiceController extends ServiceControllerAdapter {

  /**
   * デフォルトコンストラクタ
   */
  public NoticeRegistServiceController() {
        super();
  }

     /**
      * 入力に対する処理を実行します。
      *
      * @return 処理結果
      * @throws SystemException 処理実行時にシステム例外が発生
      * @throws ApplicationException 処理実行時にアプリケーション例外が発生
      */
  public ServiceResult service() throws SystemException, ApplicationException {
        HttpServletRequest request = getRequest();
        String title;
        String content;
        String author;
        Date date;

        // セッションオブジェクトの取得
        HttpSession session = request.getSession(false);

        // null チェック
        if(session == null){
              return null;
  }

        // データの初期化
        Vector notice = new Vector();
        HashMap data = new HashMap();

        // セッションオブジェクトに保存されている掲示板情報の取得
        notice = (Vector)session.getAttribute("tutorial_notice_info");

        // null が帰ってきたときは、掲示板情報登録用のオブジェクトを
        // 初期化する
        if(notice == null){
```

```
            notice = new Vector();
        }

        // リクエスト情報から登録する掲示板情報を取得
        title = request.getParameter("title");
        content = request.getParameter("content");
        author = getUserInfo().getUserID();
        date = new Date();

        // データオブジェクトに掲示板情報をセットする
        data.put("title",title);
        data.put("content",content);
        data.put("author",author);
        data.put("date",date);

        notice.add(data);

        // セッションオブジェクトに掲示板情報を再登録
        session.setAttribute("tutorial_notice_info",notice);

        return null;
    }

}
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

網掛け部分が今回の追加分です。順に説明していきます。

まず、

```
HttpServletRequest request = getRequest();
```

この行では、前のページから渡されたリクエスト情報を取得しています。getRequest()というメソッドは、スーパークラスである ServiceControllerAdapter クラスで実装されているメソッドで HttpServletRequest オブジェクトを取得することができます。

次に、

```
HttpSession session = request.getSession(false);
```

という行では、HTTP セッションオブジェクトを取得しています。セッションオブジェクトは HttpServletRequest に対して getSession()メソッドを使用して取得しています。ここで引数に false を与えている理由は true を与えるとセッションが切れていたとき(タイムアウトになっているとき)に、新しい空のセッションが作成されてしまうからです。JavaEE フレームワークでは実行するタイミングでセッションが維持された状態にあるかどうかをチェックしているのですが、たまたまそのチェックを通過した後にセッション切れになった時、このメソッドで true を引数に指定していると、不具合が生じてしまう可能性があります。ここでは、HttpServletRequest#getSession(boolean)メソッドでは、必ず false を指定すると覚えておきましょう。(詳しくは JAVA の API リストを参照してください)

HttpServletRequest#getSession(boolean)メソッドに対して false を指定すると有効なセッションが存在しないときに null が返却されますので、null チェックを入れておきましょう。

```
if(session == null){
                return null;
}
```

次にこのセッションオブジェクトに対して、

```
notice = (Vector)session.getAttribute("notice.conf.notice_info");
```

とすることで、現在セッションオブジェクト上に保存されている掲示板情報(“notice.conf.notice_info”という名前で保存されています)を取得することができます。ここでは変数 notice という Vector のインスタンスにその情報を格納しています。

あとは、リクエスト情報から受け取った登録情報を整形しながらデータ格納用のオブジェクトに登録していく

```
data.put("title",title);
data.put("content",content);
data.put("author",author);
data.put("date",date);

notice.add(data);
```

というような処理が続き、最後に登録された情報を Vector 変数 notice に追加した後、

```
session.setAttribute("tutorial_notice_info",notice);
```

という行で、セッションオブジェクト上に保存されている掲示板情報に登録しなおして処理を終了します。

メモ

スケルトンを利用すると、開発効率を向上させることが可能です。
ServiceController のスケルトンは、intra-mart ドキュメントメディア内の下記ファイルです。
skeleton/domain/category/controller/service/XXXServiceController.java

これら一連の処理の流れを図解したものが、以下の図になります。

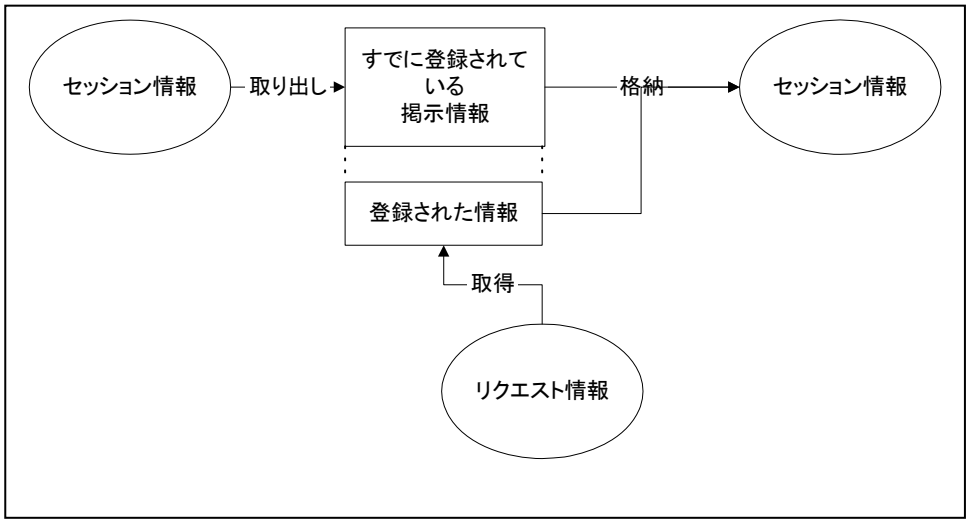

図 2.5-d

さて、これで掲示情報を登録できるようになりました。それでは実際に動かしてみましょう。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

【登録画面】

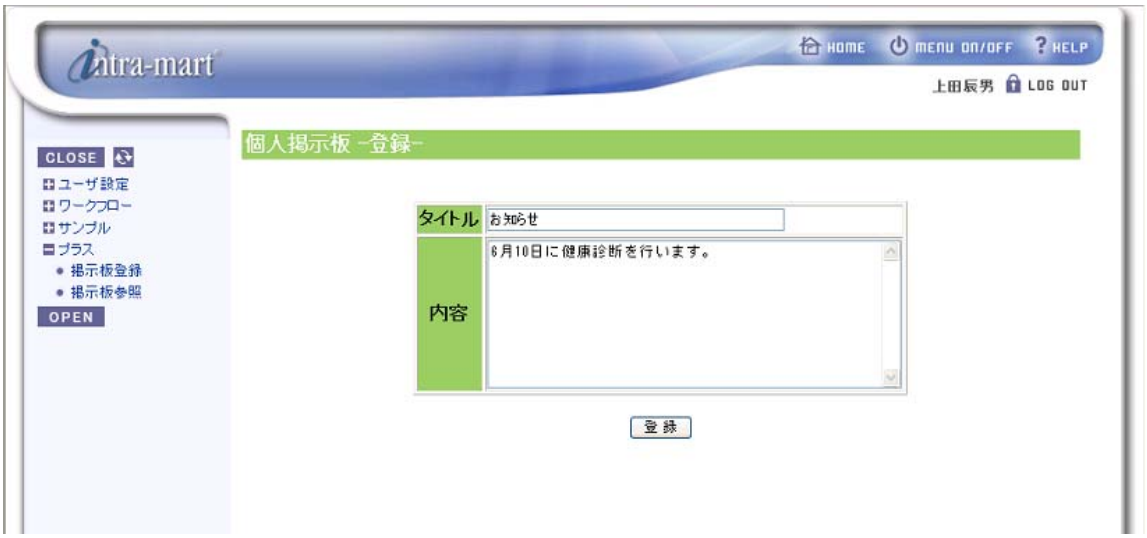

図 2.5-e

画面上は、いまいち面白くない画面遷移ですが、これで掲示板に登録した情報がセッション上に格納されるようになりました。

次は、登録された情報を表示する画面を作っていきます。

2.6 参照画面の作成

2.6.1 ヘルパーBeanを呼び出すJSPを作ってみる

さて、次は参照画面を作成していくわけですが、まずはさきほど作ったはりぼての参照画面を眺めてみましょう。

【参照画面】

図 2.6-a

あたりまえですが、この画面は登録された掲示板情報を参照するためのものです。現在、掲示板情報はセッションオブジェクト上にありますので、この画面は初期表示時(たとえばメニューから選択されたとき)に、セッションオブジェクトから掲示情報を取得して表示する必要があります。
登録画面で行ったようにサービスフレームワークを利用して情報を取ってくることもできますが、サービスフレームワークは前画面のアクションに反応して入力チェックや遷移先の動的な決定を行いながら処理するものです。

また、もちろん JSP ファイルに処理を記述して情報を取り出すこともできます。しかし、JSP ファイルにはブラウザ上に表示するためのタグも多数埋め込まれるので、表示用のタグと表示処理用のスクリプトレット(Java で記述する部分)が混在すると非常によみにくいソースになりますし、メンテナンス性も低下します。

そこで、JSP ファイルと対になる Java クラス(ヘルパーBean クラス)を用意して、表示用の記述とデータ処理用の記述をわけるために用意されている仕組みがヘルパーBean です。

この画面のように、参照画面があってそこで表示するための情報をとってくるような処理を記述するときは、ヘルパーBean を使用します。そのときの画面と処理プログラムの関係を以下に図で示します。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

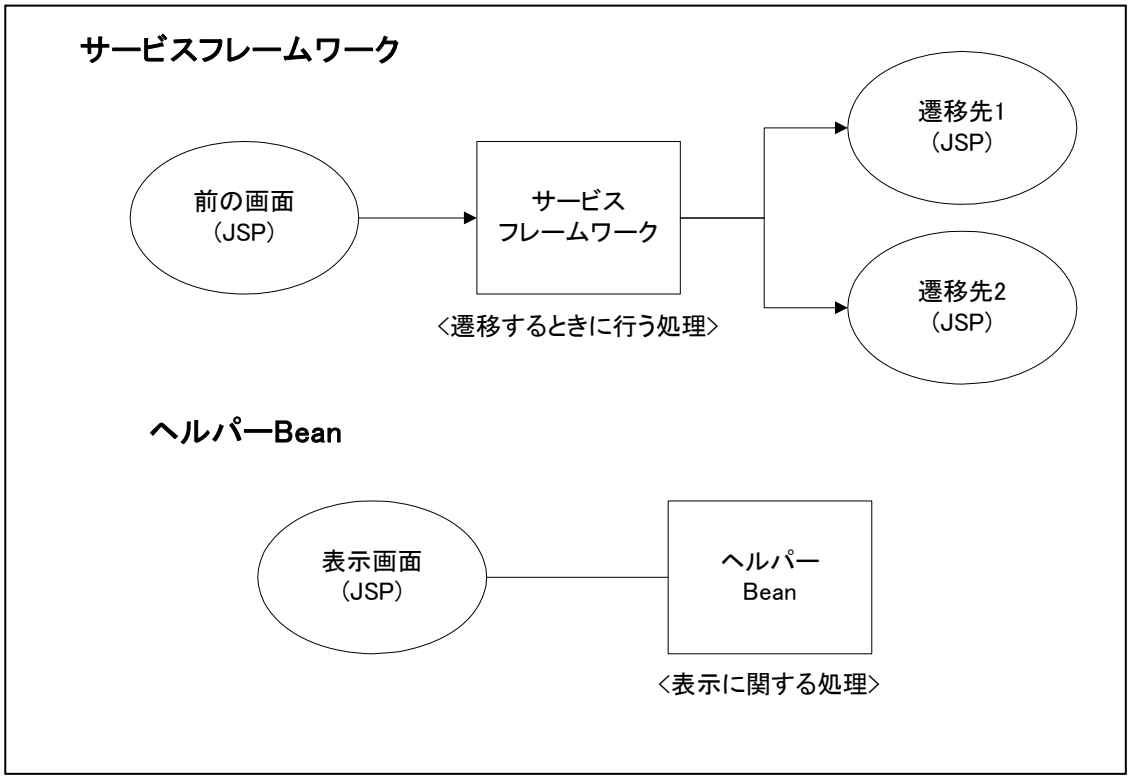

図 2.6-b

画面と画面への遷移をともなわずに、特定の画面を「表示するための」処理を記述する場合には、この図にあるようにヘルパーBean を使用します。

さて、それでは参照画面用の JSP プログラムからヘルパーBean を呼び出してみましょう。

Source 2-6

<C:/imart/doc/imart/notice/notice_view.jsp>

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

<imartj2ee:HelperBean id="notice_data" class="notice.view.bean.NoticeViewBean"/>

<HTML>

<HEAD>
    <TITLE>個人掲示板参照画面</TITLE>
</HEAD>

<BODY>
<!-- タイトル-->
<TABLE bgcolor='#99cc66' width='100%'>
    <TR><TD>
        <FONT color='white' size="+1"><b>  個人掲示板 -参照-  </b></FONT>
    </TD></TR>
</TABLE>

<BR>
<BR>

<CENTER>

<FORM>

<TABLE border="1" width="80%">
    <TR>
        <TH bgcolor="#99cc66">
            タイトル
        </TH>
        <TD align="left" colspan="3">
            ここにタイトルが入ります
        </TD>
    </TR>
    <TR>
        <TH bgcolor="#99cc66">
            作成者
        </TH>
        <TD>
            管理者
        </TD>
        <TH bgcolor="#99cc66">
            登録日
        </TH>
        <TD>
            2002/10/01 16:00:00
        </TD>
    </TR>
    <TR>
        <TH bgcolor="#99cc66">
            内容
        </TH>
        <TD align="left" colspan="3">
            ここに内容が入ります。<BR>
            ちゃんと見えてますか？
        </TD>
    </TR>
</TABLE>

</CENTER>

</FORM>
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

```
</BODY>
</HTML>
```

網掛けされたところが今回編集する部分です。

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

1 行目の page ディレクティブ(<%@ page で始まる部分)は、JSP ファイルを作成するときのおまじないのようなものです。
2 行目の taglib ディレクティブ(<%@ taglib で始まる部分)は、JavaEE フレームワークで用意しているタグライブラリの使用を宣言しています。Prefix が imartj2ee なので、このページでは<imartj2ee: >というキーワードの記述で、JavaEE フレームワークのタグライブラリを使用することができるようになります。

このタグライブラリを実際に使っている部分が、次の網掛けの部分です。

```
<imartj2ee:HelperBean id="notice_data" class="notice.view.bean.NoticeViewBean"/>
```

ここでは、JavaEE フレームワークの HelperBean タグをつかって、このページのために処理をするヘルパーBean のクラスを指定しています。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

2.6.2 ヘルパーBeanを作ってみる

次に、ヘルパーBean を作成してみましょう。パッケージ名は、
notice.view.bean
クラス名は、
NoticeViewBean

以下にソースを示します。

Source 2-6

<C:/imart/doc/imart/WEB-INF/classes/notice/view/bean/NoticeViewBean.java>

```
package notice.view.bean;

import java.io.Serializable;
import java.util.Date;
import java.util.Vector;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import jp.co.intra_mart.framework.base.web.bean.HelperBean;
import jp.co.intra_mart.framework.base.web.bean.HelperBeanException;

/**
 * @author NTTDATA intra-mart
 *
 * 掲示板情報表示用の Bean です
 */
public class NoticeViewBean extends HelperBean implements Serializable {
	private Vector notices;

	public NoticeViewBean() throws HelperBeanException{
		super();

		this.notices = null;
	}

	public void init() throws HelperBeanException{
		HttpServletRequest request = getRequest();

		int i;

		String title;
		String author;
		Date date;
		String content;

		// 掲示板情報
		notices = new Vector();

		// セッションオブジェクトの取得
		HttpSession session = request.getSession();

		// null チェック
		if(session == null){
			return;
		}

		// セッション情報からの掲示情報の取得
		try {
			notices = (Vector)session.getAttribute("tutorial_notice_info");
		} catch (IllegalStateException e) {
			throw new HelperBeanException();
		}

		if(notices == null){
			// 掲示情報がセットされていなかったとき
			notices = new Vector();
		}
	}

	/**
	 * 掲示情報を返却します
	 * @return Vector 掲示情報
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

```
     */
    public Vector getNotices() {
        return notices;
    }

    /**
     * 掲示情報をセットします
     * @param notices  セットする掲示情報
     */
    public void setNotices(Vector notices) {
        this.notices = notices;
    }
}
```

コンストラクタと、setNotices()、getNotices()は問題ありませんね。コンストラクタでは、スーパークラスの処理を呼び出しています。setNotices ()メソッドと getNotices ()メソッドは private 変数の notices という変数に対する、setter メソッドと getter メソッドです。

このなかで、init()というメソッドに注目してください。このメソッドは HelperBean クラスを継承したクラスの中で特別なメソッドです。このメソッドは、JavaEE フレームワークタグライブラリの HelperBean タグとの関係で次の図のようなタイミングで実行されます。

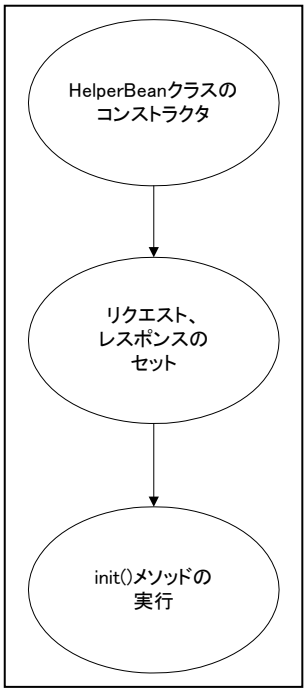

図 2.6-c

つまり、init()メソッドではリクエスト情報、レスポンス情報が HelperBean クラスのインスタンスにセットされた状態で実行することができます。これは、init()メソッドの中で安全に request 情報と response 情報の中に入っている情報を利用することができるということを意味します。

この仕組みを利用して、HelperBean クラスを継承した NoticeViewBean クラスでは、init()関数をオーバーライドしたメソッドの中で、request 情報と response 情報を利用した初期化処理を行うことができます。
ここでは、regist_view.jsp 画面で必要となる掲示情報を取得してきています。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

では、init()メソッドをみていきましょう。

この中では、掲示情報の表示に必要な処理を行っています。まず、

```
HttpSession session = request.getSession();
```

という行で、セッションオブジェクトを取得しています。これは登録画面でも行いましたね。

```
if(session == null){
        return;
}
```

は登録画面でも説明した、null チェックです。

次に、

```
notices = (Vector)session.getAttribute("tutorial_notice_info");
```

という行で、取得したセッションオブジェクトから、" tutorial_notice_info"という名前のセッション情報、つまり登録画面でセットした掲示板の掲示情報を取得しています。このとき、IllegalStateException を catch する必要があります。このとき、登録画面で掲示情報は Vector クラスとして登録しているので、Object 型から、Vector 型にキャストしています。さらに、掲示情報にデータが入っていない場合には、notices には null が入ってしまうので、

```
if(notices == null){
        notices = new Vector();
}
```

という形で、notices が null の時は初期化しておきます。

これで、init()メソッドは終了です。

さて、これでヘルパーBean の中で掲示情報を作ることができました。

スケルトンを利用すると、開発効率を向上させることが可能です。
HelperBean のスケルトンは、intra-mart ドキュメントメディア内の下記ファイルです。
skeleton/domain/category/view/bean/XXXBean.java

2.6.3 ヘルパーBeanから情報を受け取る

次に、先ほど作った JSP プログラムを修正して掲示情報を受け取る部分を作りこんでいきます。

以下に JSP プログラムのソースを示します。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 2-6

<C:/imart/doc/imart/notice/notice_view.jsp>

```
<%@ page contentType="text/html; charset=Windows-31J" pageEncoding="Windows-31J" %>
<%@ taglib prefix="imartj2ee" uri="http://www.intra-mart.co.jp/taglib/core/framework" %>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/core/standard" %>
<%@ page import="java.util.*" %>

<imartj2ee:HelperBean id="notice_data" class="notice.view.bean.NoticeViewBean"/>

<%
    // 掲示情報の取り出し
    List notices = notice_data.getNotices();
%>

<HTML>

<HEAD>
    <TITLE>個人掲示板参照画面</TITLE>
</HEAD>

<BODY>

<!-- タイトル-->
<TABLE bgcolor='#99cc66' width='100%'>
    <TR><TD>
        <FONT color='white' size="+1"><b>  個人掲示板 -参照-  </b></FONT>
    </TD></TR>
</TABLE>

<BR>
<BR>

<CENTER>

<FORM>

<imarttag:repeat list="<%= notices%>" item="record">

<TABLE border="1" width="80%">
    <TR>
        <TH bgcolor="#99cc66">
            タイトル
        </TH>
        <TD align="left" colspan="3">
            <%= ((Map)record).get("title") %>
        </TD>
    </TR>
    <TR>
        <TH bgcolor="#99cc66">
            作成者
        </TH>
        <TD>
            <%= ((Map)record).get("author") %>
        </TD>
        <TH bgcolor="#99cc66">
            日付
        </TH>
        <TD>
            <imarttag:imartDateFormat value="<%=((Date)((Map)record).get("date"))%>"
                format="yyyy 年 MM 月 dd 日 kk 時 mm 分 ss 秒"></imarttag:imartDateFormat>
        </TD>
    </TR>
    <TR>
        <TH bgcolor="#99cc66">
            内容
```

```
        </TH>
        <TD align="left" colspan="3">
            <%= ((Map)record).get("content") %>
        </TD>
    </TR>
</TABLE>
<BR>

</imarttag:repeat>

</CENTER>

</FORM>

</BODY>
</HTML>
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

これが、参照画面の JSP ファイルのソースです。例によって前のソースから変更のあった部分は網掛けにしています。

```
<%@ taglib prefix="imartj2ee" uri="http://www.intra-mart.co.jp/taglib/core/framework" %>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/core/standard" %>
```

この 2 つの行は、イントラマートが持つ 2 つのタグライブラリの使用を宣言しています。まず、登録画面でも使った JavaEE フレームワークのタグライブラリ(uri が" http://www.intra-mart.co.jp/taglib/core/framework"になっている行です)、あとはイントラマート標準のタグライブラリ(uri が" http://www.intra-mart.co.jp/taglib/core/standard"になっている部分)です。

次の、

```
<%@ page import="java.util.*" %>
```

これは、後のほうで Vector クラスと Map クラスを使用するので、そのための import 宣言です。

```
<imartj2ee:HelperBean id="notice_data" class="notice.view.bean.NoticeViewBean"/>
```

これは、JavaEE フレームワークのタグライブラリに含まれる、HelperBean タグです。このタグが呼び込まれたときに、図 2-3 で説明したように、[コンストラクタ]-[request,response 情報のセット]-[init()メソッドの実行]という処理が行われます。この時点で、id 属性で指定している"notice_data"という名前で NoticeViewBean クラスのインスタンス変数が使用できるようになることに注意してください。

さらに、

```
<%
    // 掲示情報の取り出し
    List notices = notice_data.getNotices();
%>
```

という記述で、先ほどの NoticeViewBean クラスの init()メソッドでセットした掲示情報を取り出しています。
JSP プログラムの中で<%～%>で囲まれた部分をスクリプトレットといい、ここに Java のコードを埋め込むことができます。

あとは、ここまでの処理で notices という Vector 型の変数に取り込まれた掲示情報を HTML の画面上に表示するための記述をしていきます。

```
<imarttag:repeat list="<%= notices%>" item="record">
～
</imarttag:repeat>
```

ここでは、intra-mart が標準で提供するタグライブラリ、repeat を使用しています。このタグは、与えられた変数(List 型)をその要素数ぶんだけ繰り返し処理しながらその変数の中に入っているデータを表示するときに使います。
intra-mart スクリプト開発モデルの開発経験がある方には、imart タグの repeat タグといったほうがわかりやすいかもしれません。

変数の notices の中には掲示情報(タイトル・作成者・作成日付・内容を含みます)が入っていますので、これを

repeat タグで繰り返しならが中身を取り出していこうというわけです。

実際に取り出している部分は、

```
<%= ((Map)record).get("title") %>
<%= ((Map)record).get("author") %>
<imarttag:imartDateFormat value="<%=((Date)((Map)record).get("date"))%>"
                        format="yyyy 年 MM 月 dd 日 kk 時 mm 分 ss 秒"></imarttag:imartDateFormat>
<%= ((Map)record).get("content") %>
```

の 4 行です。上から順番に、タイトル、作成者、作成日、内容を取り出しています。登録画面を作成したときに、HashMap の key に設定した文字列を思い出してください。それぞれ、title、author、date、content でしたね。

日付の情報を取得する際に、intra-mart のタグライブラリ
imartDateFormat
を利用しています。
imartDateFormat は、日付データを指定フォーマット文字列に変換して表示する際に用いるタグライブラリです。
今回は format 属性に指定した
"yyyy 年 MM 月 dd 日 kk 時 mm 分 ss 秒"
という文字列に変換して表示しています。

変換を行わずに
<%= ((Map)record).get("date") %>
と記述すると
Thu Feb 06 15:30:33 JST 2003
このように表示されてしまい、ユーザには少しわかりづらい表現となってしまいます。

これで掲示情報を JSP プログラムのコーディングは終了です。
今度はブラウザのほうで実際に掲示板が動くかを確かめてみましょう。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

【登録画面】

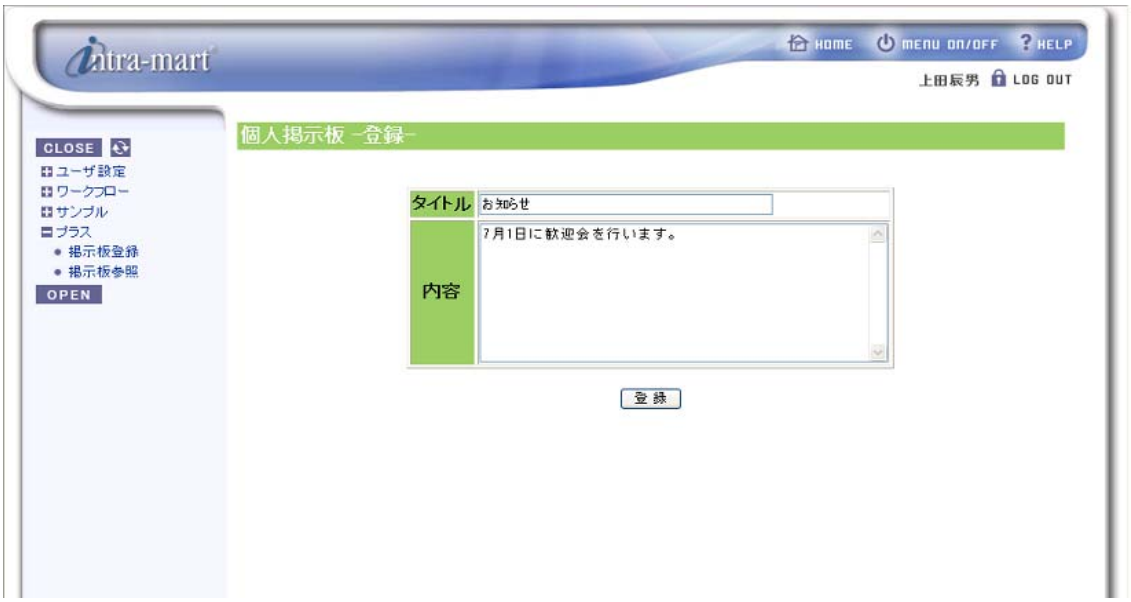

図 2.6-d

【参照画面】

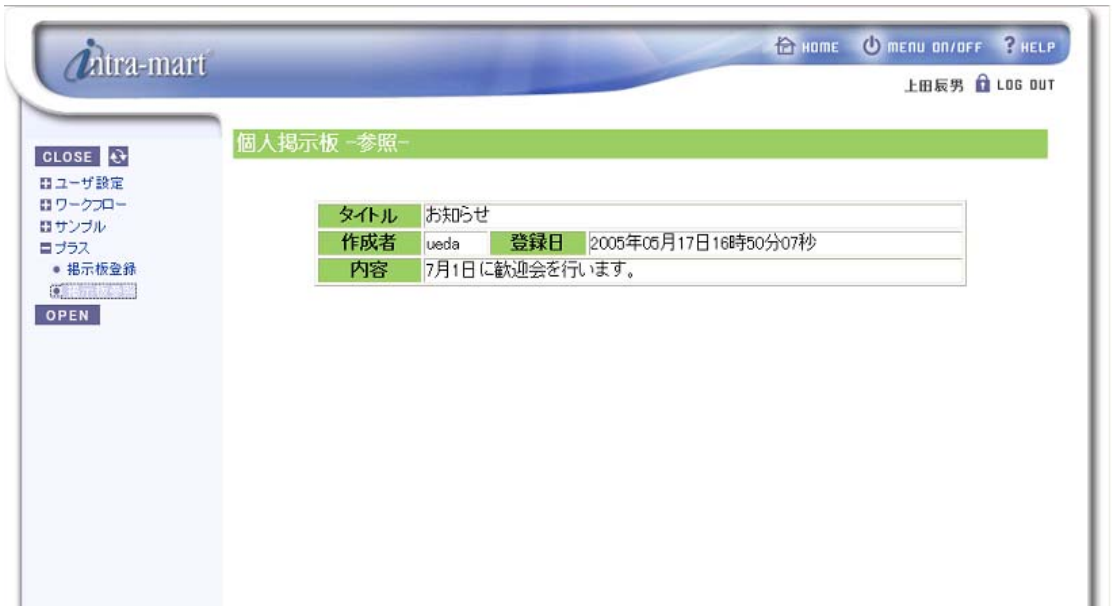

図 2.6-e

どうでしょうか？正常に動きましたか？

もし動かなかったときは、ソースと見比べながら、どこが間違っていたのか確認してみましょう。

2.7 掲示板で入力チェックをしてみよう

2.7.1 入力エラーチェック

前節までで、一通り動く掲示板が完成しました。実際に作ってみるとそれほど難しくはないと思います。
ここからは、すでに作成した掲示板に機能追加をしていきましょう。

まずは、入力チェックです。もう一度掲示板登録画面を眺めてみましょう。

【登録画面】

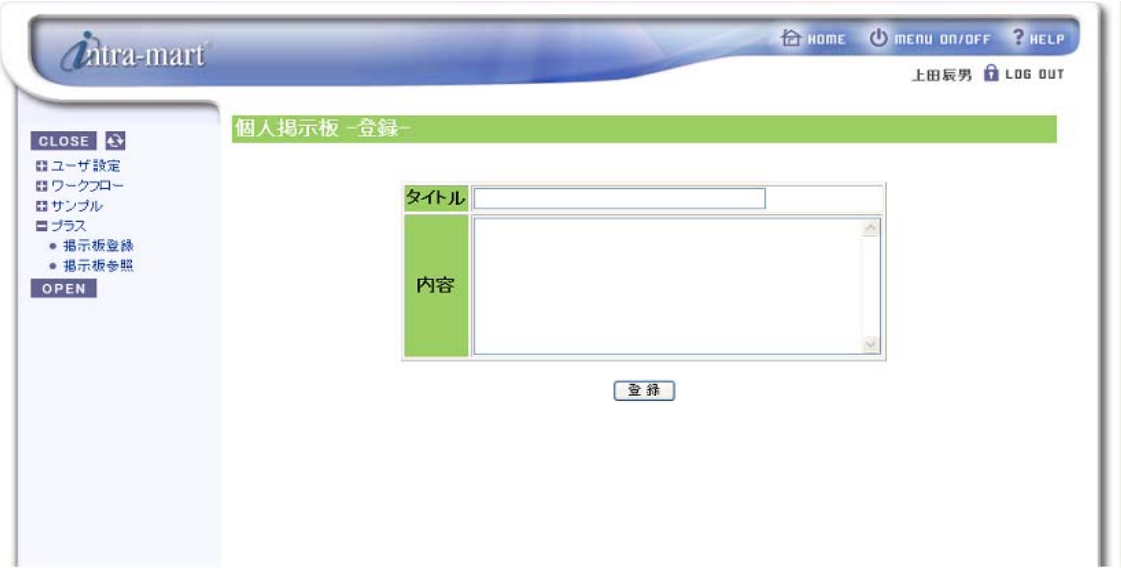

図 2.7-a

仕様の問題でいろいろな掲示板はあると思いますが、この登録画面ではタイトルと内容に何も入力していない状態、つまり、空白のまま登録ボタンが押されたときには、入力エラー画面に遷移するようにしてみましょう。

この画面で入力チェックの処理を実現するためには、いくつか方法があります。
クライアントサイド JavaScript でタイトルと内容のチェックする
サーバサイドのプログラム(この場合は、JavaEE フレームワーク)でチェックする
ちゃんと動きさえすればどちらの方法をとってもかまわないのですが、ここでは②の方法をとることにしましょう。

JavaEE フレームワークの中で入力チェックをするためには、サービスコントローラに check()メソッドを記述します。登録画面の登録ボタンが押された後の処理に対応するサービスコントローラは、NoticeRegistServiceController です。したがって、NoticeRegistServiceController クラスの中に check()メソッドを記述します。まずは、ソースを見てください。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 2-7

<C:/imart/doc/imart/WEB-INF/classes

/notice/controller/service/NoticeRegistServiceController.java>

```
package notice.controller.service;

import java.util.Date;
import java.util.HashMap;
import java.util.Vector;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import jp.co.intra_mart.framework.base.service.RequestException;
import jp.co.intra_mart.framework.base.service.ServiceControllerAdapter;
import jp.co.intra_mart.framework.base.service.ServiceResult;
import jp.co.intra_mart.framework.system.exception.ApplicationException;
import jp.co.intra_mart.framework.system.exception.SystemException;

/**
 * @author NTTDATA intra-mart
 *
 * 掲示情報を登録するサービスコントローラーです。
 */
public class NoticeRegistServiceController extends ServiceControllerAdapter {

    /**
     * デフォルトコンストラクタ
     */
    public NoticeRegistServiceController() {
        super();
    }

    /**
     * 入力内容に対するチェックを行います
     *
     * @throws RequestException リクエスト内容に誤りがあるときに発生
     * @throws SystemException 処理実行時にシステム例外が発生
     */
  public void check() throws RequestException, SystemException{
        // リクエスト情報の取得
        HttpServletRequest request = getRequest();

        String title;
        String content;

        // タイトルと内容の取得
        title = request.getParameter("title");
        content = request.getParameter("content");

        if(title.equals("")){
            // タイトルが空文字の時は、RequestException を
            // スローする
            throw new RequestException("タイトルに何か入力してください");
        }

        if(content.equals("")){
            // 内容が空文字の時は、RequestException を
            // スローする
            throw new RequestException("内容に何か入力してください");
        }
  }


    /**
     * 入力に対する処理を実行します。
```

```
     *
     * @return 処理結果
     * @throws SystemException 処理実行時にシステム例外が発生
     * @throws ApplicationException 処理実行時にアプリケーション例外が発生
     */
    public ServiceResult service() throws SystemException, ApplicationException {
        HttpServletRequest request = getRequest();
        String title;
        String content;
        String author;
        Date date;

        // セッションオブジェクトの取得
        HttpSession session = request.getSession(false);

        // null チェック
        if(session == null){
            return null;
        }

        // データの初期化
        Vector notice = new Vector();
        HashMap data = new HashMap();

        // セッションオブジェクトに保存されている掲示板情報の取得
        notice = (Vector)session.getAttribute("tutorial_notice_info");

        // null が帰ってきたときは、掲示板情報登録用のオブジェクトを
        // 初期化する
        if(notice == null){
            notice = new Vector();
        }

        // リクエスト情報から登録する掲示板情報を取得
        title = request.getParameter("title");
        content = request.getParameter("content");
        author = getUserInfo().getUserID();
        date = new Date();

        // データオブジェクトに掲示板情報をセットする
        data.put("title",title);
        data.put("content",content);
        data.put("author",author);
        data.put("date",date);

        notice.add(data);

        // セッションオブジェクトに掲示板情報を再登録
        session.setAttribute("tutorial_notice_info",notice);

        return null;
    }

}
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

このソースでも、変更点は網掛けで示してあります。ここでは check()メソッドが追加されました。それでは、ソースをみていきましょう。

まずこれは service()メソッドと同じなのですが、

```
// リクエスト情報の取得
HttpServletRequest request = getRequest();
```

で、登録画面で入力された情報を含むリクエスト情報を取得しています。登録画面の作成の節で説明したように、getRequest()というメソッドは、スーパークラスである ServiceControllerAdapter クラスで実装されているものです。

次に、

```
// タイトルと内容の取得
title = request.getParameter("title");
content = request.getParameter("content");
```

という部分で、リクエスト情報からタイトルと内容の情報を取り出しています。check()の目的はこれらのタイトルと内容が空白かどうかを判定して、空白であれば入力エラーページに遷移することを目的とするので、次にそのチェック用の処理を記述します。まず、先にタイトルの空白チェックを行います。

```
if(title.equals("")){
    // タイトルが空文字の時は、RequestException を
    // スローする
    throw new RequestException("タイトルに何か入力してください");
}
```

title.equals("")というのは、タイトルという文字列変数が空文字かどうかを真偽値で返す文です。タイトルが""(空文字)に等しい場合には true が返ります。そしてここで、true が返ってきたときには、RequestException という例外を throw しています(例外の throw の仕方は大丈夫ですよね？わからない場合は、市販の Java の解説本を見てください)。RequestException というのは、JavaEE フレームワークで用意されている例外で JavaEE フレームワークがこの例外を catch すると、設定されている入力エラーページに遷移します。ここで、
%intra-mart のドキュメントルート%/WEB-INF/classes/ServiceConfig.properties
というファイルをエディタなどで開いてみてください。

```
input.error.page.path=/j2ee/document/error/input_error.jsp
```

という記述が確認できると思います。ServiceConfig.properties は、JavaEE フレームワークが動いている環境全体のデフォルトの設定を定義しているファイルなので、(他で入力エラーページを設定していない場合)入力エラーが発生した場合にはここで設定されているページに遷移します。とりあえず、

このあと、check()メソッドは内容(contents)に対しても同じチェックを行い処理を終了します。

ここで、画面を実際に動かしてみましょう。
ソースの修正が終わり保存したら、掲示板登録画面のほうに戻ってタイトルに何も入力せずに登録ボタンを押してみてください。
以下のような画面が現れると思います。

【入力エラー画面】

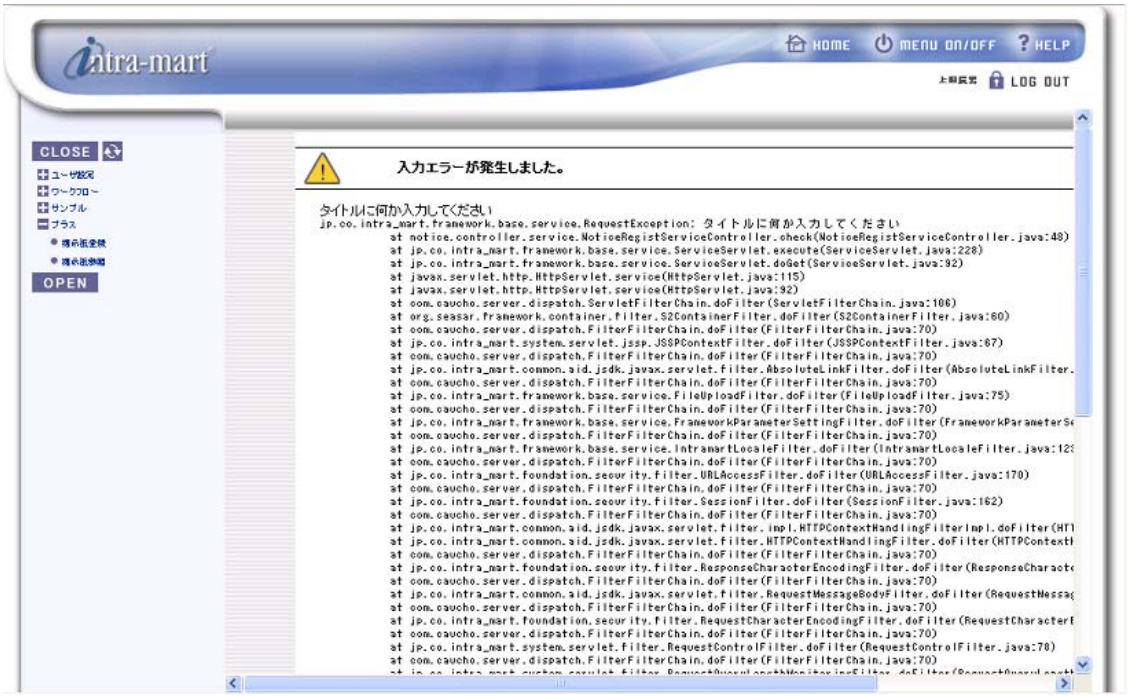

図 2.7-b

「入力エラーが発生しました」という少しフォントが大きめの文字の下に「タイトルに何か入力してください」という文字列が確認できると思います。

ここで、NoticeRegistServiceController.java に戻って入力エラー例外をスローしているところに戻ってください。

```
throw new RequestException("タイトルに何か入力してください");
```

入力エラー画面で「入力エラーが発生しました」という文字列の下に表示されている文字列は、例外をスローするときに与えた引数であることが確認できると思います。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

次にこの画面を眺めていて、「タイトルに何か入力してください」という文字列の下に、書かれている部分を見てください。これは StackTrace の内容ということがわかると思います。JavaEE フレームワークの標準の(service-config.xml で定義されている)入力エラー画面は、StackTrace の内容を表示するような仕組みになっています。

ここで少し考えてみて下さい。掲示板を使うユーザにとって、StackTrace の内容は必要でしょうか？プログラムを作成しているときには(つまりバグを取ろうとしているときには)必要な情報でも、ユーザにとってはあまり有用な情報ではないことに気づきます。

そこで次にこの入力エラー画面を、掲示板登録時用の入力エラー画面に変更することを考えてみましょう。

2.7.2 入力エラーチェック画面を作成してみる

2.7.2.1 入力エラー画面の作成

入力エラーを出力するための画面(JSP プログラム)を作成してみましょう。
今回エラー画面に表示するのは「入力エラーが発生しました。」というメインメッセージと、補助メッセージです。

Source 2-7

<C:/imart/doc/imart/notice/notice_input_error.jsp>

```
<%@ page contentType="text/html; charset=Windows-31J" pageEncoding="Windows-31J" %>
<%@ taglib prefix="imartj2ee" uri="http://www.intra-mart.co.jp/taglib/core/framework" %>
<%@ page errorPage="/j2ee/document/error/error.jsp" %>
<%@ page import="jp.co.intra_mart.framework.base.service.RequestException" %>
<%@ page import="java.io.PrintWriter" %>

<imartj2ee:HelperBean id="bean" class="jp.co.intra_mart.framework.base.web.bean.ErrorHelperBean" />

<%
    RequestException exception = (RequestException)bean.getException();
%>

<HTML>
<HEAD>
    <TITLE>入力エラーページ</TITLE>
        <STYLE type="text/css">
        <!--
        .contntClass {
            font-size: 12px;
        }
        -->
        </STYLE>
</HEAD>

<BODY bgcolor="#FFFFFF" leftmargin="0" topmargin="0" marginwidth="0" marginheight="0">
<TABLE width="100%" height="100%" border="0" cellpadding="0" cellspacing="0">
    <TR>
        <TD width="1%" height="1%"><IMG src="images/j2ee/exp/explanation_01.gif" width="64" height="22"></TD>
        <TD width="99%" height="1%" background="images/j2ee/exp/explanation_02.gif" > </TD>
    </TR>
    <TR>
        <TD background="images/j2ee/exp/explanation_04.gif" height="99%" width="1%"> </TD>
        <TD valign="top"><br>
            <TABLE width="100%" border="0" cellspacing="0" cellpadding="0">
                <TR>
                    <TD bgcolor="#000000" height="1" width="100%"></TD>
                </TR>
            </TABLE>
            <TABLE width="100%" height="37" border="0" cellpadding="0" cellspacing="0">
                <TR>
                    <TD width="5%"><IMG src="images/j2ee/mark.gif" width="48" height="37"></TD>
                    <TD width="5%"> </TD>
                    <TD width="95%"><STRONG>
                        入力エラーが発生しました。
                    </STRONG></TD>
                </TR>
            </TABLE>
            <TABLE width="100%" border="0" cellspacing="0" cellpadding="0">
                <TR>
                    <TD bgcolor="#000000" height="1" width="100%"></TD>
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

```
                </TR>
            </TABLE><br>
            <TABLE width="800" border="0" cellpadding="0" cellspacing="0">
                <TR>
                    <TD width="23"><IMG src="images/j2ee/spacer.gif" width="23" height="8"></TD>
                    <TD align="left"><span class="contntClass">
                        <%= exception.getMessage() %>
                        </span>
                    </TD>
                </TR>
                <TR>
                    <TD>
                         
                    </TD>
                    <TD>
                        <PRE><%-- exception.printStackTrace(new PrintWriter(out)); --%></PRE>
                        <BR>
                        <BR>
                    </TD>
                </TR>
                <TR>
                    <TD width="23"><IMG src="images/j2ee/spacer.gif" width="23" height="8"></TD>
                    <TD align="left">
                        <FORM name="login">
                            <INPUT type="button" value=" B A C K " onClick="history.back()">
                        </FORM>
                    </TD>
                </TR>
                <TR>
                    <TD width="23"><IMG src="images/j2ee/spacer.gif" width="23" height="8"></TD>
                    <TD align="left">
                        <span class="contntClass">
                        </span>
                    </TD>
                </TR>
            </TABLE>
            <BR>
            <BR>
            <BR>
            <BR>
            <BR>
            <BR>
            <BR>
            <br>

<TABLE width="100%" border="0" cellspacing="0" cellpadding="0">
    <TR>
        <TD bgcolor="#CCCCCC" height="1" width="100%"></TD>
    </TR>
</TABLE>

            <span style="font-size:11px"><BR>
               Copyright(C)NTT DATA INTRAMART CO.,LTD 2000-2005 All Rights Reserved.<BR>
            </span></TD>
    </TR>
</TABLE>


</HTML>
```

今回作成した入力エラーチェック画面では、前章で作成したサービスコントローラ(NoticeRegistServiceController.java)がスローした例外発生時のメッセージ内容を取得して、表示します。

実装内容をみていきましょう。
まずポイントとなるのは、JavaEE フレームワークが提供するタグライブラリ HelperBean を使用している部分です。

```
<imartj2ee:HelperBean id="bean" class="jp.co.intra_mart.framework.base.web.bean.ErrorHelperBean" />
```

タグライブラリの使用方法に関しては、前章で学習したとおりです。
では、ここで JSP プログラムから呼び出されている ErrorHelperBean とは、どのようなプログラムなのでしょうか？
ErrorHelperBean は、JavaEE フレームワークが提供する例外情報を取得するコンテンツです。
詳細は API リスト(jp.co.intra_mart.framework.base.web.bean.ErrorHelperBean)を参照してください。

入力エラーチェック画面では、このあと

```
RequestException exception = (RequestException)bean.getException();
```

上記のような方法で、ErrorHelperBean が提供する getException()メソッドを呼び出してサービスコントローラ(NoticeRegistServiceController.java)がスローした例外情報を取得しています。

メインメッセージとしては、「入力エラーが発生しました。」という固定文字列を設定しています。
次にエラーが発生した詳細情報を表示するための補助メッセージとして、

```
<%= exception.getMessage() %>
```

このような記述を行い、サービスコントローラ(NoticeRegistServiceController.java)がスローしたエラーメッセージの内容を出力しています。
具体的には、「タイトルに何か入力してください」、あるいは、「内容に何か入力してください」というメッセージが表示されます。

前章で表示されていた Stack Trace の内容が開発者にとって必要な情報であるのに対し、今回表示した補助メッセージはユーザがエラーの原因を知るために有益となる情報であると言えます。

例外発生時に表示されるエラープログラムの作成は完了しましたが、ここまでの作業では作成したエラー画面へ遷移させることはできません。
では次に、当アプリケーションにおいて想定される例外(入力漏れ)が発生した際、今回作成したエラー画面へ遷移するよう、プロパティファイルの設定を行ってみましょう。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

2.7.2.2 xmlファイルの編集

エラーページへの画面遷移においても、プロパティファイル(service-config-notice.xml)で設定を行います。
service-config-notice.xml をエディタで開き、以下の内容を追記してください。

Source 2-7

<C:/imart/doc/imart/WEB-INF/classes/notice/conf/service-config-notice.xml>

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

    <service>
        <service-id>regist</service-id>
        <next-page>
            <page-path>/notice/notice_regist.jsp</page-path>
        </next-page>
    </service>

    <service>
        <service-id>view</service-id>
        <next-page>
            <page-path>/notice/notice_view.jsp</page-path>
        </next-page>
    </service>

    <service>
        <service-id>notice_regist</service-id>
        <controller-class>
            notice.controller.service.NoticeRegistServiceController
        </controller-class>
        <input-error>
            <page-path>/notice/notice_input_error.jsp</page-path>
        </input-error>
        <next-page>
            <page-path>/notice/notice_regist.jsp</page-path>
        </next-page>
    </service>

</service-config>
```

追記するのは、網掛け部分です。(紙面の関係上一部改行しています)

処理の流れを簡単にまとめてみます。

エラー! 編集中のフィールド コードからは、オブジェクトを作成できません。

図 2.7-c

今回追記したのは、

```
<input-error>
    <page-path>エラーページパス</page-path>
</input-error>
```

上記のような内容です。

登録処理を行うサービスコントローラ(NoticeRegistServiceController.java)の check メソッドにおいて例外(RequestException)が発生した場合に、プロパティファイルに設定したエラーページ(notice_input_error.jsp)へ画面遷移します。

これまでにサービスプロパティファイル(xml)に設定するパラメータがいくつか出てきましたが、その他にサービスプロパティファイルで設定可能となるパラメータの詳細に関しては、API リストの jp.co.intra_mart.framework.base.service.XmlServicePropertyHandler クラスの解説内容を確認してください。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

以上で作業は完了です。
掲示板登録画面で入力欄が未入力の状態で登録ボタンをクリックし、動作を確認してください。

【入力エラー画面】

図 2.7-d

2.8 掲示情報をデータベースに格納する

ここまでの掲示板アプリケーションでは、掲示情報をセッション上に保存していました。当然、ユーザがログアウトしてしまうと掲示内容は消えてしまいますし、他のユーザがその情報を参照することもできません。通常の掲示板のように、ログアウトしても情報は消えず、他のユーザからも掲示情報を参照できるようにするためには、なにか掲示情報を永続的に保存する方法を考えなければなりません。

よくある方法として、
情報を StorageService 上のファイルに保存する
データベースに保存する
という方法が考えられますが、ここではデータベースを使用して情報を永続的に保持することを考えていきましょう。

利用するテーブルは、tutorial_plus_notice です。
構造は以下のとおりです。

列名	内容	属性	データ型	サイズ(bytes)
notice_cd	掲示板コード	主キー	テキスト型	20
title	タイトル		テキスト型	100
author	投稿者 ユーザーコード		テキスト型	50
content	内容		テキスト型	255
record_date	更新日付		テキスト型	19

intra-mart で、データベースアクセスを行うためには「データベースの接続設定」が必要となります（詳細はマニュアル等を参照してください）。

データベースの接続設定が完了したら、コピーしたテンプレートディレクトリ直下にある「notice_ddl.sql」をデータベースに対して実行し、tutorial_plus_notice テーブルを作成してください。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

データベースの準備ができたら、実装にうつりましょう。

JavaEE フレームワークにおいてデータベースを使用する場合には、イベントフレームワークとデータフレームワークを用いて作成された部分を追加していきます。

JavaEE フレームワークでは、イベントフレームワークの StandardEventListner が業務処理の実行やトランザクション管理を制御しています。サービスコントローラからイベントフレームワークを使用すれば、フレームワークがトランザクションの管理を行ってくれるので便利です。
ここでは、サービスコントローラからイベントフレームワークを呼び出して、データベースにアクセスする方法を学習します。

このとき、全体の流れとしては以下の図のようなものになります。

エラー! 編集中のフィールド コードからは、オブジェクトを作成できません。

図 2.8-a

サービスコントローラは画面から掲示情報を受け取ることができます。ビジネスロジック(つまり、実際に行いたい処理)を記述するイベントフレームワークではその掲示情報を受け取り、データフレームワークを呼び出して登録処理を依頼します。データ処理を受け持つデータフレームワークは、受け取った情報をもとに、DB 登録処理を行います。登録処理の結果ですが、今回のアプリケーションでは返却値として返すのではなく、例外を投げることでたとえば DB エラーの発生を知らせるという仕組みにしたいと思います。ですので、帰り道である[データフレームワーク]→[イベントフレームワーク]→[サービスフレームワーク]では、特に返却値を返さないこととします。

以上の図を念頭において、イベントフレームワークとデータフレームワークを作成していきましょう。

2.8.1 登録画面のイベントフレームワークを実装する

イベントフレームワークは JavaEE フレームワークの中で主にビジネスロジックを記述するフレームワークです。JavaEE フレームワークでは、表示用の処理の部分とデータアクセス用の処理の部分と切り離されています。これが、MVC モデルを採用している JavaEE フレームワークの強みとなっている部分で、後々のシステム変更への対応をしやすくしている部分です。
それでは、これから登録画面の掲示情報登録時のイベントフレームワークを実装していきましょう。

イベント部分を実装するためには次のクラスを用意する必要があります。
イベント
イベントリスナー

ここで①のイベントはサービスフレームワークから処理に必要なデータの受け渡しをするための器になるクラスです。そして、②のイベントリスナーにイベントの中で行いたい処理を記述します。

エラー! 編集中のフィールド コードからは、オブジェクトを作成できません。

図 2.8-b

図 2.8-bを見て下さい。ここでは、掲示情報をデータベースに格納しようとしているわけですから、イベントという器に格納する情報は、登録ボタンが押されたときに登録しようとしている掲示情報です。次に、イベントリスナーではイベントから受け取った掲示情報をデータベースに登録するためにデータフレームワークに登録処理を依頼する処理を行います。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

それでは、イベントフレームワークを以下の図ような流れで作成していきましょう。

エラー！ 編集中のフィールド コードからは、オブジェクトを作成できません。

図 2.8-c

2.8.1.1 xmlファイルの編集

まずは、掲示板登録用のイベントと DAO の名前を決めてしまいましょう(サービスフレームワークを使用するときに決めた名前と同じようなものです)。ここでは掲示板登録用のイベントの名前を notice_regist、dao の名前を notice_inf とします。

アプリケーション ID	キー	説明
notice.conf.notice	notice_regist	掲示板登録用のイベントです。
notice.conf.notice	notice_inf	掲示板情報用の DAO です。

次にイベントとイベントリスナーと DAO のクラスの名前を決ましょう。これも単なるキーワードなので、ここでは下のようにすることにします。

種別	名称	説明
イベント	NoticeRegistEvent	掲示板登録用のイベントです。
イベントリスナー	NoticeRegistEventListener	掲示板登録用のイベントリスナーです。
DAO	NoticeIntramartDBDAO	掲示板情報用の DAO です。

JavaEE フレームワークで、これらのクラスを使用するためにはまず xml ファイルに NoticeRegistEvent クラスと NoticeRegistEventListener クラスの場所を指定する必要があります。サービスフレームワークの xml ファイルと同じように event-config-notice.xml というファイルの中に、イベント名をキーとしてイベント用のクラスを設定します。

テキストエディタで下のような記述をして、
%intra-mart のドキュメントルート%/WEB-INF/classes/notice/conf/event-config-notice.xml
という名前でファイルを保存してください。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 2-8

<C:/imart/doc/imart/WEB-INF/classes/notice/conf/event-config-notice.xml>

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



<event-config>
  <event-group>
      <event-key>notice_regist</event-key>
      <event-class>notice.model.event.NoticeRegistEvent</event-class>
      <event-factory>
          <factory-class>
              jp.co.intra_mart.framework.base.event.StandardEventListenerFactory
          </factory-class>
          <init-param>
              <param-name>listener</param-name>
              <param-value>notice.model.event.NoticeRegistEventListener</param-value>
          </init-param>
  </event-factory>
</event-group>
</event-config>
```

<event-key>notice_regist</event-key>
この部分は、イベントキーを指定しています。

<event-class>notice.model.event.NoticeRegistEvent</event-class>
この部分は、掲示板登録で使用する Event のクラスを指定しています。

<factory-class>
jp.co.intra_mart.framework.base.event.StandardEventListenerFactory
</factory-class>
この部分は EventListenerFactory を指定していますが、ここでは JavaEE フレームワークのデフォルトの StandardEventListenerFactory を指定しています。EventListenerFactory は、JavaEE フレームワークの中でイベントの実行を制御しています。処理の制御を自分で記述することもできますが、ほとんどの場合、標準の StandardEventListenerFactory を使用します。

<init-param>
<param-name>listener</param-name>
<param-value>notice.model.event.NoticeRegistEventListener</param-value>
</init-param>
この部分は、掲示板情報登録処理で使用する EventListener を指定しています。

次にデータフレームワークに関する xml ファイルの編集です。サービスフレームワークやイベントフレームワークと同様に、data-config-notice.xml ファイルの中に、キー名をキーとして記述します。
また、ここでは、intra-mart WebPlatform で設定されているデータベースを使用することを前提とします。

%intra-mart のドキュメントルート%/WEB-INF/classes/notice/conf
の下に data-config-notice.xml というファイルを作成してください。

その中に、以下のような記述をして保存します。

Source 2-8

<C:/imart/doc/imart/WEB-INF/classes/notice/conf/data-config-notice.xml>

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

<data-config>
  <dao-group>
      <dao-key>notice_inf</dao-key>
      <dao>
          <dao-class>notice.model.data.NoticeIntramartDBDAO</dao-class>
          <connector-name>logingroup_db</connector-name>
      </dao>
  </dao-group>
</data-config>
```

<dao-key>notice_inf</dao-key>
この部分は、DAO キーを指定しています。

<dao-class>notice.model.data.NoticeIntramartDBDAO</dao-class>
この部分は、データストア（掲示板テーブル）へのアクセス処理を定義する NoticeIntramartDBDAO クラスのパス情報を設定しています。

<connector-name>logingroup_db</connector-name>
この部分は、コネクタとして intra-mart 標準の DB に接続する
jp.co.intra_mart.framework.base.data.LoginGroupDBConnector
を使用することを宣言しています。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

2.8.1.2 サービスフレームワークの修正

以上で、JavaEE フレームワークでイベントフレームワークを作成する準備が整いました。

ここからはしばらく画面で動かして確認をしながら作業をすることができなくなるので(途中で動かしたら、エラーになります)、少しつらいところですが皆さん辛抱してついてきてください。

最初に、さきほど作成した NoticeRegistServiceController を編集して、イベントフレームワークに掲示情報を渡すようにプログラムの修正を行います。

以下のソースを見てください。

Source 2-8

<C:/imart/doc/imart/WEB-INF/classes/notice

/controller/service/NoticeRegistServiceController.java >

```
package notice.controller.service;

import java.util.Date;
import javax.servlet.http.HttpServletRequest;

import notice.model.event.NoticeRegistEvent;
import notice.model.object.NoticeInf;

import jp.co.intra_mart.framework.base.service.RequestException;
import jp.co.intra_mart.framework.base.service.ServiceControllerAdapter;
import jp.co.intra_mart.framework.base.service.ServiceResult;
import jp.co.intra_mart.framework.base.util.UserInfo;
import jp.co.intra_mart.framework.system.exception.ApplicationException;
import jp.co.intra_mart.framework.system.exception.SystemException;

/**
 * @author NTTDATA intra-mart
 *
 * 掲示情報を登録するサービスコントローラーです。
 */
public class NoticeRegistServiceController extends ServiceControllerAdapter {

    /**
     * デフォルトコンストラクタ
     */
    public NoticeRegistServiceController() {
        super();
    }

    /**
     * 入力内容に対するチェックを行います
     *
     * @throws RequestException リクエスト内容に誤りがあるときに発生
     * @throws SystemException 処理実行時にシステム例外が発生
     */
    public void check() throws RequestException, SystemException{
        // リクエスト情報の取得
        HttpServletRequest request = getRequest();

        String title;
        String content;

        // タイトルと内容の取得
        title = request.getParameter("title");
        content = request.getParameter("content");

        if(title.equals("")){
            // タイトルが空文字の時は、RequestException を
            // スローする
            throw new RequestException("タイトルに何か入力してください");
        }

        if(content.equals("")){
            // 内容が空文字の時は、RequestException を
            // スローする
            throw new RequestException("内容に何か入力してください");
        }
    }


    /**
     * 入力に対する処理を実行します。
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

```
     *
     * @return 処理結果
     * @throws SystemException 処理実行時にシステム例外が発生
     * @throws ApplicationException 処理実行時にアプリケーション例外が発生
     */
    public ServiceResult service() throws SystemException, ApplicationException {
        HttpServletRequest request = getRequest();
        String title;
        String content;
        UserInfo author;
        Date date;

        // 掲示板情報登録用のインスタンスの初期化
        NoticeInf notice = new NoticeInf();

        // リクエスト情報から登録する掲示板情報を取得
        title = request.getParameter("title");
        author = getUserInfo();
        date = new Date();
        content = request.getParameter("content");

        // 掲示板情報を作成
        notice.setTitle(title);
        notice.setAuthor(author.getUserID());
        notice.setRegistDate(date);
        notice.setContent(content);

        // イベントの呼び出し
        NoticeRegistEvent event
            = (NoticeRegistEvent)createEvent("notice.conf.notice", "notice_regist");
        // イベントへの掲示情報の登録
        event.setNotice(notice);
        // イベントの実行
        dispatchEvent(event);

            return null;
    }

}
```

例によって今回の修正で変更のあった部分は灰色で示しています。主に service()メソッドの修正になります。

まず、今回のサービスコントローラでは0節で作成した掲示板情報クラスを利用しています。

```
// 掲示板情報登録用のインスタンスの初期化
NoticeInf notice = new NoticeInf();
```

ここでは、掲示板情報登録用のインスタンスを初期化しています。次に、

```
// リクエスト情報から登録する掲示板情報を取得
title = request.getParameter("title");
author = getUserInfo();
date = new Date();
content = request.getParameter("content");
```

この部分でリクエスト情報を受け取っています。タイトル(title)は、リクエスト情報の中に"title"という名前で入っています。作成者は、現在ログインしているユーザなので getUserInfo().getUserID()というメソッドを使っています。これは、NoticeRegistServiceController クラスのスーパークラスである ServiceControllerAdapter クラスで定義されているメソッドで、ログインユーザコードを取得するメソッドです。作成日付(date)は、現在の日付なので new Date()として取得しています。最後に、内容(content)はリクエスト情報から"content"という名前で取得できます。

これで、掲示情報を登録するためのデータはそろいましたので、先ほど初期化した掲示板情報登録用のインスタンス(notice)にデータを格納していきます。これが、

```
// 掲示板情報を作成
notice.setTitle(title);
notice.setAuthor(author.getUserID());
notice.setRegistDate(date);
notice.setContent(content);
```

この部分です。これらは、掲示板情報モデルクラスのそれぞれの属性の set メソッドをもちいて登録していきます。

次にいよいよイベントの呼び出しです。まずは、先ほど定義した掲示板登録イベントのイベント(アプリケーション ID が"notice.conf.notice"、イベント ID が"notice_regist")をもちいてイベントを呼び出します。

```
NoticeRegistEvent event
= (NoticeRegistEvent)createEvent("notice.conf.notice", "notice_regist");
```

NoticeRegistEvent というクラスは、先ほど掲示板登録用のイベントのクラス名です。createEvent()というメソッドに、アプリケーション ID とイベント ID を引数として渡すと、掲示板登録用のイベントがサービスコントローラ内で取得できます。この createEvent() というメソッドも、NoticeRegistServiceControlle r のスーパークラスである ServiceControllerAdapter クラスで定義されているメソッドです。

これでイベントを取得することができたので、次はこのイベントに必要な情報を登録します。

```
// イベントへの掲示情報の登録
event.setNotice(notice);
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

setNotice()というメソッドは後で作成しますが、掲示情報モデルクラス(NoticeInf クラス)を引数としてもらい、イベントクラスに掲示情報を登録するためのメソッドです。

これで、イベントへの掲示情報の登録が終わったので、イベントを実行します。

```
// イベントの実行
dispatchEvent(event);
```

dispatchEvent()メソッドも、ServiceControllerAdapter クラスで定義されているメソッドで、引数として渡されたイベントを実行しています。

これで、ServiceController の修正は終了です。以下に service メソッドの中で行った処理の流れを記述しておきます。参考にしてください。

エラー! 編集中のフィールド コードからは、オブジェクトを作成できません。

図 2.8-d

2.8.1.3 イベントフレームワークの処理の実装

次に比較的作成の簡単な NoticeRegistEvent クラスを作成しましょう。先に述べたように、イベントクラスはイベントフレームワークの中でイベントリスナーに情報を受け渡す器のような働きをします。現在作成しているイベントフレームワークの処理は掲示板の登録処理なので、このときに渡される情報とは、掲示板の情報ということになります。サービスフレームワークを作成したときに渡している掲示板情報とは、掲示板情報モデルクラスでしたので、ここでのイベントクラスは掲示板情報モデルクラスを受け渡すことができるはずです。

それでは、さっそく NoticeRegistEvent を作成してみましょう。パッケージ名は notice.model.event とします。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 2-8

<C:/imart/doc/imart/WEB-INF/classes/notice

/model/event/NoticeRegistEvent.java>

```
package notice.model.event;

import notice.model.object.NoticeInf;

import jp.co.intra_mart.framework.base.event.Event;

/**
 * 掲示板情報登録のイベントに対する処理です
 *
 * @author nttdata intra-mart
 *
 */
public class NoticeRegistEvent extends Event {
   private NoticeInf notice;

   /**
    * デフォルトコンストラクタ
    */
   public NoticeRegistEvent() {
         super();
   }

   /**
    * 掲示板情報を返却する
    * @return NoticeInf 掲示板情報
    */
   public NoticeInf getNotice() {
         return notice;
   }

   /**
    * 掲示板情報をセットする
    * @param notice セットする掲示板情報
    */
   public void setNotice(NoticeInf notice) {
         this.notice = notice;
   }

}
```

ぱっと見た感じそれほど難しくありませんね。

まず、イベントクラスを作成するときは JavaEE フレームワークの Event クラスを継承して作成します。

public class NoticeRegistEvent extends Event {

さらに、このクラスは0節で作成した、掲示板情報モデルクラスをプライベートな変数として持ちます。そして、この掲示板情報を保持する変数(notice)に対して値をセットするメソッドであるsetNotice()メソッドと、値を取得するメソッドであるgetNotice()メソッドが定義されています。

こちらは、全体的に難しくないと思います。

次に、このイベントで渡される値を受け取って実際の処理を行うイベントリスナーを作成してみましょう。ここでいう実際の処理とは、データフレームワークを呼び出して掲示板情報をデータベースに格納する処理のことをさします。このことを念頭において、次のソースを眺めていってください。

メモ

スケルトンを利用すると、開発効率を向上させることが可能です。
Event のスケルトンは、intra-mart ドキュメントメディア内の下記ファイルです。
skeleton/domain/category/model/event/XXXEvent.java

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 2-8

<C:/imart/doc/imart/WEB-INF/classes/notice

/model/event/NoticeRegistEventListener.java>

```
package notice.model.event;

import notice.model.data.NoticeDAOIF;
import notice.model.object.NoticeInf;

import jp.co.intra_mart.framework.base.event.Event;
import jp.co.intra_mart.framework.base.event.EventResult;
import jp.co.intra_mart.framework.base.event.StandardEventListener;
import jp.co.intra_mart.framework.system.exception.ApplicationException;
import jp.co.intra_mart.framework.system.exception.SystemException;

/**
 * 掲示板情報を登録するイベントリスナー
 *
 * @author NTTDATA intra-mart
 */
public class NoticeRegistEventListener extends StandardEventListener {

    /**
     * @param event  イベント
     * @return EventResult  処理結果
     * @see jp.co.intra_mart.framework.base.event.StandardEventListener#fire(Event)
     */
  protected EventResult fire(Event event)
                  throws SystemException, ApplicationException {
      NoticeRegistEvent noticeRegistEvent;
      NoticeInf notice;
      NoticeDAOIF dao;

      //  イベントのキャスト
      noticeRegistEvent = (NoticeRegistEvent)event;
      //  登録する掲示情報の取得
      notice = noticeRegistEvent.getNotice();

      // dao の取得
      dao = (NoticeDAOIF)getDAO("notice.conf.notice","notice_inf"
                                ,noticeRegistEvent.getUserInfo().getLoginGroupID());
      dao.insert(notice);

  return null;
  }

}
```

このクラスの中で定義されているメソッドは fire()メソッドだけです。fire()メソッドは、NoticeRegistEventListener クラスのスーパークラスである、StandardEventListener で定義されているメソッドでこの NoticeRegistEventListener クラスでは fire()メソッドをオーバーライドしていることになります。

イベントリスナーにおいて、イベントが実行されたときに行いたい処理は fire()メソッドに記述していきます。fire()メソッドは呼び出し元から event クラスのインスタンスを引数という形で受け取ることができます。イベントクラスの中には、掲示板の情報が保持されていてそれを取り出すことができますので、ここでは取り出した情報をデータベースに格納することを考えます。

まず、受け取る引数は一般的な(クラスの視点ではより上位の階層にある)Event クラスの形で受け取るので、まずこれを掲示板登録用に利用できるようにキャストしてやる必要があります。

```
// イベントのキャスト
noticeRegistEvent = (NoticeRegistEvent)event;
```

ここでは、Event クラスから NoticeRegistEvent クラスにキャストしています。このようにキャストすることで掲示板情報を取得することができるようになります。

```
// 登録する掲示情報の取得
notice = noticeRegistEvent.getNotice();
```

ここで、NoticeRegistEvent クラスの getNotice()メソッドに戻ってみてみると、返り値は掲示板情報モデルクラス(NoticeInf クラス)で返ってくるので、ここでも NoticeInf クラスの型を持つ変数、notice に情報を格納することにしましょう。

次に、DAO のクラスを取得しています。

```
// dao の取得
dao = (NoticeDAOIF)getDAO("notice.conf.notice","notice_inf"
,noticeRegistEvent.getUserInfo().getLoginGroupID());
```

DAO とはデータフレームワークの特定の処理対象に対応するクラスをあらわしていて、このクラスにメソッドを発行することでデータフレームワークがあらわす様々なデータストレージ(ここではデータベース)に対して処理を行うことができます。

ここで、getDAO()というメソッドに引数として、アプリケーション ID と(データフレームワーク用の)キーID、それに現在ログインしているユーザの DB 参照名を与えています。getDAO()メソッドは、NoticeRegistEventListener のスーパークラスである StandardEventListener で実装しているメソッドで、与える引数で指定される dao を取得できます。

次に、この取得した DAO に対して、掲示板情報のデータベースへのデータ挿入処理を行っています。

```
dao.insert(notice);
```

この文で、引数として与えた掲示板情報モデルクラス(notice)の内容の挿入処理を、DAO に処理させているということになります。

以上で、イベントリスナーは完成です。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

これで、図 2.8-bで登場した、イベントとイベントリスナーの実装が終わったので、掲示板登録に関するイベントフレームワークの処理の実装は終了ということになります。いかがでしょうか？作成するクラスは多くなってきていますが、それぞれのクラスで記述すべき処理はそれほど多くないことが理解していただけたと思います。

それでは次に、イベントリスナーの中で少し触れたデータフレームワークの処理を実装していきます。

メモ

スケルトンを利用すると、開発効率を向上させることが可能です。
EventListener のスケルトンは、intra-mart インストールメディア内の下記ファイルです。
skeleton/domain/category/model/event/XXXEventListener.java

2.8.2 登録画面のデータフレームワークを実装する

2.8.2.1 データフレームワークの処理の実装

データフレームワークでは、DAO(Data Access Object)を作成することが目的になります。DAO とは、背後に存在するデータアクセスを抽象化したものです。データアクセスの対象となるものは、データベース、LDAP サーバ、EJB 様々なものが考えられます。

DAO は背後に存在するデータソースに合わせて作成されるものです。DAO でデータソースへのアクセスを抽象化しておけば、背後にあるシステムが後々のシステム変更などで別のデータソースに切り替わっても業務処理のロジック(イベントフレームワーク)に変更を加える必要はなくなります。

このために、DAO は DAO インターフェースという形でそのインターフェースを統一しておきます。つまり、データフレームワークの開発では DAO インターフェースと DAO のクラスを作成する必要があります。

インターフェースと実際に処理を行うクラスを分けるのは、データ処理の部分はその対象のデータソースが変更されることが多いためです。

この掲示板アプリケーションでは、データベース、特に intra-mart で設定しているデータベースに永続データを格納することを想定しているので、データベースアクセスをすることを前提に話を進めていきましょう。

まずは、DAO インターフェースを作成します。以下のソースを見てください。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 2-8

<C:/imart/doc/imart/WEB-INF/classes/notice

/model/data/NoticeDAOIF.java>

```
package notice.model.data;

import java.util.Vector;
import notice.model.object.NoticeInf;
import jp.co.intra_mart.framework.base.data.DAOException;

/**
 * 掲示板情報にアクセスするための DAOIF です
 *
 * @author NTTDATA intra-mart
 */
public interface NoticeDAOIF {
public void insert(NoticeInf notice) throws DAOException;
}
```

ここでは、掲示板の登録処理を作成しているので、insert メソッドのみが定義されています。インターフェースとはメソッドのメソッド宣言のみを記述した抽象クラスの一種なので、ここでは insert メソッドのメソッド宣言のみを記述しています。

public void insert(NoticeInf notice) throws DAOException;

掲示板情報モデルクラス(NoticeInf)を引数にもらい、返却値を持たないメソッドの宣言です。また、SQL エラーなどのシステムエラーを検知するために、DAOException という例外を throw します。

この insert()メソッドの具体的な処理は、この NoticeDAOIF インターフェースを実装した DAO クラスで記述していきます。ここでは、intara-mart の DB アクセス用の DAO を作成するので名前を NoticeIntramartDBDAO として作成していきましょう。

以下に NoticeIntramartDBDAO のソースを示します。

メモ

スケルトンを利用すると、開発効率を向上させることが可能です。
DAOIF のスケルトンは、intra-mart ドキュメントメディア内の下記ファイルです。
skeleton/domain/category/model/data/XXXDAOIF.java

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 2-8

<C:/imart/doc/imart/WEB-INF/classes/notice

/model/data/NoticeIntramartDBDAO.java>

```
package notice.model.data;

import java.sql.SQLException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Vector;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import notice.model.object.NoticeInf;

import jp.co.intra_mart.foundation.service.client.information.Identifier;
import jp.co.intra_mart.framework.base.data.DAOException;
import jp.co.intra_mart.framework.base.data.LoginGroupDBDAO;


/**
 * 掲示板情報にアクセスするための intra-martDBDAO
 *
 * @author NTTDATA intra-mart
 */
public class NoticeIntramartDBDAO extends LoginGroupDBDAO implements NoticeDAOIF {

    /**
     * デフォルトコンストラクタ
     */
    public NoticeIntramartDBDAO() {
        super();
    }

    //SQL 定義
    private final String SQL_INSERT = "INSERT INTO tutorial_plus_notice " +
        "(notice_cd,title,author,record_date,content) VALUES (?,?,?,?,?)";

    /**
     * 掲示板情報を登録します
     * @see notice.model.data.NoticeDAOIF#insert(NoticeInf)
     * @param notice 掲示板情報
     */
  public void insert(NoticeInf notice) throws DAOException {
        Connection con = null;
        PreparedStatement stmt = null;
        String noticeCd;
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd|HH:mm:ss");
        String registDate = sdf.format(notice.getRegistDate());


        if(notice == null){
            return;
        }

        //Connection の取得
        try{
            con = getConnection();
        } catch(Exception e){
            throw new DAOException(e.getMessage(), e);
        }

        // 掲示板コードの取得
        noticeCd = Identifier.make();
```

```
        //insert 処理
        try{
            stmt = con.prepareStatement(SQL_INSERT);
            stmt.setString(1,noticeCd);
            stmt.setString(2,notice.getTitle());
            stmt.setString(3,notice.getAuthor());
            stmt.setString(4,registDate);
            stmt.setString(5,notice.getContent());
            stmt.executeUpdate();
        }catch(SQLException e){
            throw new DAOException(e.getMessage(), e);
        } finally {
            try{
                if(stmt != null){
                    stmt.close();
                }
            } catch (SQLException e){
            }
        }
    }

}
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

NoticeIntramartDBDAO クラスの insert()メソッドが先ほど NoticeDAOIF クラスで定義した insert()メソッドの具体的な処理を記述している部分です。まず、クラス宣言の部分を見てください。

```
public class NoticeIntramartDBDAO extends LoginGroupDBDAO implements NoticeDAOIF {
```

まず先ほど作成した、NoticeDAOIF インターフェースを実装しています。また LoginGroupDBDAO クラスを継承していることに注目してください。LoginGroupDBDAO クラスは JavaEE フレームワークで用意されている intra-mart の DB アクセス用のクラスで、intra-mart で接続設定しているデータベースを利用するにはこのクラスを継承します。

まず冒頭で SQL 文の定義を行っています。

```
private final String SQL_INSERT = "INSERT INTO tutorial_plus_notice " +
                "(notice_cd,title,author,record_date,content) VALUES (?,?,?,?,?)";
```

insert()関数の中身をみていきます。
まず、Connection の取得です。

```
//Connection の取得
try{
    con = getConnection();
} catch(Exception e){
    throw new DAOException(e.getMessage(), e);
}
```

ここでは、jp.co.intra_mart.framework.base.data.LoginGroupDBDAO クラスの getConnection()メソッドを利用してコネクションを取得しています。このメソッドを利用することによりログイングループに関連づけられた DB の領域に対するコネクションを取得することが可能となります。

ここで、さきほどイベントフレームワークの節で作成したイベントリスナー(NoticeRegistEventListener)で getDAO()と記述している部分を思い出してください。

```
dao = (NoticeDAOIF)getDAO("notice.conf.notice",
"notice_inf",
noticeRegistEvent.getUserInfo().getLoginGroupID());
```

第 3 引数で noticeRegistEvent.getUserInfo().getLoginGroupID()とすることで、getDAO メソッドに現在ログインしているユーザのグループ ID を指定しています。
intra-mart では予め登録しておいた DB の領域をグループ ID に関連づけて設定します。
LoginGroupDBDAO クラスではこのグループ ID 情報を受け取って、getConnection()メソッドで返却されるコネクションを取得し、アクセスを行っています。

さて、これで DB 操作をするためのクラスを取得することができましたので、実際のデータを挿入するためのデータを作成していきます。

挿入する必要のあるデータは、
掲示板コード
タイトル
作成者
作成日付
内容
です。このうち、掲示板コードと作成日付(現在の時間です)はこのメソッドの中で作成しなければなりません。

掲示板コードは、主キーカラムなのでユニークでなければなりません。ここでは inntra-mart の標準 API の Identifier.make()を使うことにより、複数のユーザが同時に同じ内容のリクエストを行った際にも確実にユニークな ID を生成し、掲示板テーブルに登録しています。

```
// 掲示板コードの取得
noticeCd = Identifier.make();
```

前述の PreparedStatement を生成しています。

```
stmt = con.prepareStatement(SQL_INSERT);
```

掲示板コード以外の登録データは、引数で受け渡される掲示板情報モデルクラス（NoticeInf）から情報を参照し設定しています。

```
stmt.setString(1,noticeCd);
stmt.setString(2,notice.getTitle());
stmt.setString(3,notice.getAuthor());
stmt.setString(4,registDate);
stmt.setString(5,notice.getContent());
```

データの設定が完了したところで、SQL 文を DB に対して発行します。

```
stmt.executeUpdate();
```

以上で、データの挿入処理は完了です。

メモ

スケルトンを利用すると、開発効率を向上させることが可能です。
IntramartDBDAO のスケルトンは、intra-mart ドキュメントメディア内の下記ファイルです。
skeleton/domain/category/model/data/XXXIntramartDBDAO.java
XXXIntramartDBDAO.java は、IntramartDBDAO を extends したクラスを作成する際に利用します。
DBDAO を extends したクラスを作成する場合は、下記スケルトンを利用してください。
skeleton/domain/category/model/data/ XXXDBDAO.java

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

2.9 掲示情報をデータベースから取り出して表示する

前節で掲示情報をデータベースの中に格納することができるようになったので、今度は掲示板参照画面でデータベースに入っている掲示情報を取り出して表示する部分を作成しましょう。

掲示板参照画面では、ヘルパーBean を使ってセッション情報から掲示情報を取得してそれを表示していました。この節では、ヘルパーBean からイベントフレームワークを呼び出し、データベースから掲示情報を取得して表示するように掲示板参照画面を修正していきます。

プログラムの流れとしては、以下のような内容になります。

エラー! 編集中のフィールド コードからは、オブジェクトを作成できません。

図 2.9-a

登録画面とは逆の処理になりますが、まずヘルパーBean は掲示情報取得用のイベントフレームワークを呼び出して、掲示情報参照処理を依頼します。次に、呼び出されたイベントフレームワークでは掲示情報取得用のデータフレームワークを呼び出して、データベースから掲示情報を取得します。イベントフレームワークでは、受け取った掲示情報をヘルパーBean に返して、ヘルパーBean はその情報を表示用に整形して、画面プログラムである JSP プログラムに渡して表示します。

それでは、まずヘルパーBean からイベントフレームワークを呼び出すように修正していきましょう。

2.9.1 参照画面にイベントフレームワークを実装する

2.9.1.1 xmlファイルの修正

それではまず、登録画面の時と同じように、参照画面で使用するイベントフレームワーク名前を決めて、xml ファイルの編集を行いましょう。

イベントフレームワークでは、参照画面から掲示板情報の取得の依頼を受け、データフレームワークから掲示情報を受け取り、ヘルパーBeanのほうに掲示情報を渡します。下の図は、図 2.9-aのうち、ヘルパーBeanとイベントフレームワークの関係に着目して抜き出したものです。

エラー! 編集中のフィールド コードからは、オブジェクトを作成できません。

図 2.9-b

この図から、情報の受け渡しを見ると、

ヘルパーBean からイベントフレームワークには渡されるものはない。
イベントフレームワークからヘルパーBean へは掲示情報が渡される。

という関係になっています。イベントフレームワークの入力に対応するのが Event クラス、イベントフレームワークからの出力に対応するのが EventResult クラスですので、このケースでは EventResult クラスのみを作成して、Event クラスは空のものでよいということになります。それを踏まえて次の表を見てください。

＜イベントフレームワーク＞

種類	アプリケーション ID	キー	クラス	概要
イベント	notice.conf.notice	notice_view	EmptyEvent	空のクラスです。
イベントリスナー	notice.conf.notice	notice_view	NoticeViewEventListener	掲示板参照用のイベントリスナーです。
イベントリザルト			NoticeViewEventResult	掲示板参照用のイベントリスナーです。

今回のイベントフレームワークでは表のような名前をつけることにします。このなかで、注目していただきたいのはイベントのクラスです。ここでは特に情報の受け渡しをする必要がないので、JavaEE フレームワークで用意されている EmptyEvent クラスを使用することにします。ですので、今回作成する必要のあるクラスはイベントリスナーとイベントリザルトということになります。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

それでは、掲示情報参照画面用のイベントフレームワークのクラスを、JavaEE フレームワークに登録することにしましょう。掲示情報登録画面を作成したときに編集した event-config-notice.xml を開いて、以下の記述を追加して下さい。

Source 2-9

<C:/imart/doc/imart/WEB-INF/classes/notice/conf/event-config-notice.xml >

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

    <event-group>
        <event-key>notice_regist</event-key>
        <event-class>notice.model.event.NoticeRegistEvent</event-class>
        <event-factory>
            <factory-class>
                jp.co.intra_mart.framework.base.event.StandardEventListenerFactory
            </factory-class>
            <init-param>
                <param-name>listener</param-name>
                <param-value>notice.model.event.NoticeRegistEventListener</param-value>
            </init-param>
        </event-factory>
    </event-group>

  <event-group>
      <event-key>notice_view</event-key>
      <event-class>jp.co.intra_mart.framework.base.event.EmptyEvent</event-class>
      <event-factory>
          <factory-class>
              jp.co.intra_mart.framework.base.event.StandardEventListenerFactory
          </factory-class>
          <init-param>
              <param-name>listener</param-name>
              <param-value>notice.model.event.NoticeViewEventListener</param-value>
          </init-param>
      </event-factory>
  </event-group>

</event-config>
```

網掛けになっている部分を追加定義します。

<event-key>notice_view</event-key>
この部分はイベントキーの設定です。

<event-class>jp.co.intra_mart.framework.base.event.EmptyEvent</event-class>
この部分は Event 用のクラスを登録する記述です。ここでは、JavaEE フレームワークで用意されている空の Event クラスを使用しています。このパラメータは設定してもしなくてもプログラムの動作には大した影響はありません。

<factory-class>
jp.co.intra_mart.framework.base.event.StandardEventListenerFactory
</factory-class>
この部分は、EventFactory 用のクラスを登録する記述です。こちらも、通常標準のものを使えばよいので、JavaEE フレームワークで用意されている StandardEventListenerFactory クラスを使用します。

```
<init-param>
    <param-name>listener</param-name>
    <param-value>notice.model.event.NoticeViewEventListener</param-value>
</init-param>
```

この部分は、EventListener 用のクラスを登録する記述です。ここでは、掲示板参照用にこれから作成する NoticeViewEventListener を notice.model.event というパッケージ名で登録しています。

以上で、イベントフレームワーク用の xml ファイルの編集は終了です。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

2.9.1.2 ヘルパーBeanクラスの修正

それでは、次に参照画面用のヘルパーBean クラスを修正してイベントフレームワークを呼び出す形式に編集します。

掲示板参照画面のヘルパーBean クラスである、NoticeViewBean.java を編集します。

Source 2-9

<C:/imart/doc/imart/WEB-INF/classes/notice

/view/bean/NoticeViewBean.java>

```
package notice.view.bean;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Vector;

import notice.model.event.NoticeViewEventResult;
import notice.model.object.NoticeInf;

import jp.co.intra_mart.framework.base.event.Event;
import jp.co.intra_mart.framework.base.web.bean.HelperBean;
import jp.co.intra_mart.framework.base.web.bean.HelperBeanException;


/**
 * @author NTTDATA intra-mart
 *
 * 掲示板情報表示用の Bean です
 */
public class NoticeViewBean extends HelperBean implements Serializable {
	private Vector notices;

	public NoticeViewBean() throws HelperBeanException{
		super();

		this.notices = null;
	}

	public void init() throws HelperBeanException{
	  int i;
	  Event event = createEvent("notice.conf.notice", "notice_view");
	  NoticeViewEventResult eventResult;
	  Vector noticeView;

	  try {
		eventResult = (NoticeViewEventResult)dispatchEvent(event);
	  } catch (Exception e) {
		throw new HelperBeanException(e.getMessage(), e);
	  }

	  noticeView = eventResult.getNotices();

	  notices = new Vector();

	  for(i=0;i<noticeView.size();i++){
		NoticeInf notice = (NoticeInf)noticeView.get(i);
		HashMap map = new HashMap();

		map.put("title", notice.getTitle());
		map.put("author", notice.getAuthor());
		map.put("date", notice.getRegistDate());
		map.put("content", notice.getContent());

		notices.add(map);
	  }
	}

	/**
	 * 掲示情報を返却します
	 * @return Vector 掲示情報
	 */
	public Vector getNotices() {
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

```
            return notices;
        }

        /**
         * 掲示情報をセットします
         * @param notices  セットする掲示情報
         */
        public void setNotices(Vector notices) {
            this.notices = notices;
        }
}
```

ここでも、init()メソッドに注目してください。

まず、サービスフレームワークでイベントを呼び出したときと同じようなメソッドで、イベントクラスを取得します。ただし先に説明したとおり、今回は空のイベントである EmptyEvent クラスを使用していますので、サービスフレームワークのイベント呼び出しでやったようにキャストをする必要がありません。変数 event は、あとでイベントを起動する時に必要になります。

```
Event event = createEvent("notice.conf.notice", "notice_view");
```

createEvent()メソッドは、この NoticeViewBean クラスのスーパークラスである HelperBean クラスで定義されているメソッドで、サービスフレームワークの時と同じように、アプリケーション ID とキーID を引数にとって、それに対応するイベントクラスのインスタンスを取得します。xml ファイルに

```
<event-class>jp.co.intra_mart.framework.base.event.EmptyEvent</event-class>
```

と記述しているので、このときに実際に取得できるクラスは EmptyEvent クラスということになります。次に、

```
try {
    eventResult = (NoticeViewEventResult)dispatchEvent(event);
} catch (Exception e) {
    throw new HelperBeanException(e.getMessage(), e);
}
```

では、dispatchEvent()メソッドを使用してイベントを起動して、その結果である NoticeViewEventResult を取得しています。このとき、何らかの例外が発生した場合には、HelperBeanException を投げています。この辺は、サービスフレームワークの時とやりかたはほとんど変わりません。

次に、このイベントリザルトから必要な掲示情報を取得します。

```
noticeView = eventResult.getNotices();
```

これでイベントを実行して必要とする掲示情報を取得することができました。
次に JSP 画面に渡すためにデータを整形する処理を行います。

```
notices = new Vector();

for(i=0;i<noticeView.size();i++){
    NoticeInf notice = (NoticeInf)noticeView.get(i);
    HashMap map = new HashMap();

    Date registDate = notice.getRegistDate();
    String dateStr = registDate.toString();

    map.put("title", notice.getTitle());
    map.put("author", notice.getAuthor());
    map.put("date", dateStr);
    map.put("content", notice.getContent());
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

```
    notices.add(map);
}
```

JSP プログラムでは、掲示情報を「title」、「author」、「date」、「content」というキーで格納して HashMap クラスのインスタンスとして受け取っていますので、ここではその形に合うようにデータを整形しています。

以上で、ヘルパーBean クラスの修正は終了です。

2.9.1.3 イベントフレームワークの作成

それでは、次にイベントフレームワークを作成していきましょう。
2.9.1.1で触れたように、この掲示板情報取得用のイベントフレームワークではイベントリスナーとイベントリザルトを作成する必要があります。

イベントリスナーではデータフレームワークを呼び出して掲示板情報を取得し、イベントリザルトにその結果を格納する処理を、イベントリザルトでは受け取った掲示情報を保持する処理を記述していきます。

それでは、まずイベントリスナーを作成しましょう。

Source 2-9

<C:/imart/doc/imart/WEB-INF/classes/notice

/model/event/NoticeViewEventListener.java>

```
package notice.model.event;

import java.util.Vector;

import notice.model.data.NoticeDAOIF;

import jp.co.intra_mart.framework.base.event.Event;
import jp.co.intra_mart.framework.base.event.EventResult;
import jp.co.intra_mart.framework.base.event.StandardEventListener;
import jp.co.intra_mart.framework.system.exception.ApplicationException;
import jp.co.intra_mart.framework.system.exception.SystemException;

/**
 * @author NTTDATA intra-mart
 *
 * 掲示情報を取得するイベントリスナー
 */
public class NoticeViewEventListener extends StandardEventListener {

    /**
     * デフォルトコンストラクタ
     */
    public NoticeViewEventListener() {
        super();
    }

    /**
     * イベント実行
     */
    protected EventResult fire(Event event)
                                throws SystemException, ApplicationException {
        Vector notices;
        NoticeDAOIF dao;
        NoticeViewEventResult eventResult = new NoticeViewEventResult();

        // DAO の取得
        dao = (NoticeDAOIF)getDAO("notice.conf.notice", "notice_inf"
                                        , event.getUserInfo().getLoginGroupID());

        // 掲示情報の取得
        notices = dao.select();

        // イベントリザルトへの登録
        eventResult.setNotices(notices);

        return eventResult;
    }

}
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

ここで、処理を記述する必要があるのは fire()メソッドです。

まず、掲示板情報用の DAO を取得します。

```
// DAO の取得
dao = (NoticeDAOIF)getDAO("notice.conf.notice", "notice_inf", event.getUserInfo().getLoginGroupID());
```

ここでは、アプリケーション IDを”notice.conf.notice”、キーを”notice_inf”として DAOを取得しています。このキーに相当する DAO が掲示情報用の DAO です。
第三引数は掲示板登録処理と同じく、グループ ID 情報を取得し、関連付けられたデータベースの領域にアクセスしています。

次に、この取得した DAO に対して select()メソッドを実行し、掲示情報を全件取得します。

```
// 掲示情報の取得
notices = dao.select();
```

さらに、この取得した掲示情報を掲示情報取得用のイベントリザルトに登録して、そのイベントリザルトをメソッドの返り値として返却します。

```
// イベントリザルトへの登録
eventResult.setNotices(notices);
return eventResult;
```

以上で、イベントリスナーの記述は終了です。
次に、掲示情報取得用のイベントリザルトを記述しましょう。

Source 2-9

<C:/imart/doc/imart/WEB-INF/classes/notice

/model/event/NoticeViewEventResult.java>

```
package notice.model.event;

import java.util.Vector;

import jp.co.intra_mart.framework.base.event.EventResult;

/**
 * @author NTTDATA      intra-mart
 *
 * 掲示情報を取得するイベントリザルトです
 */
public class NoticeViewEventResult implements EventResult {
   private Vector notices;

   /**
    * デフォルトコンストラクタ
    */
   public NoticeViewEventResult() {
         super();
   }

   /**
    * 掲示情報を返却します
    * @return Vector  掲示情報
    */
   public Vector getNotices() {
         return notices;
   }

   /**
    * 掲示情報をセットします
    * @param notices  セットする掲示情報
    */
   public void setNotices(Vector notices) {
         this.notices = notices;
   }

}
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

NoticeViewEventResult クラスでは、掲示情報モデルオブジェクトを保持して、その情報に対するセットメソッドとゲットメソッドを提供します。

まず、getNotices()メソッドでは、NoticeViewEventResult クラスが private 変数として持っている notices という変数を返却します。

```
public Vector getNotices() {
    return notices;
}
```

次に setNotices()メソッドでは、受け取った掲示情報を notices 変数にセットします。

```
public void setNotices(Vector notices) {
    this.notices = notices;
}
```

以上でイベントリザルトの処理は終了です。

メモ

スケルトンを利用すると、開発効率を向上させることが可能です。
EventResult のスケルトンは、intra-mart ドキュメントメディア内の下記ファイルです。
skeleton/domain/category/model/event/XXXEventResult.java

2.9.1.4 データフレームワークの作成

掲示情報に対するデータアクセス用のデータフレームワークとして、掲示情報を登録する処理を作成していますので、ここではそれらのクラスに情報取得用の処理を加えましょう。

修正する必要のあるクラス(インターフェース)は、NoticeDAOIF インターフェースと NoticeIntramartDBDAO クラスです。

まずは、NoticeDAOIF インターフェースを修正しましょう。
ここでは、掲示情報取得用のメソッドの宣言を追加します。

Source 2-9
<C:/imart/doc/imart/WEB-INF/classes/notice
/model/data/NoticeDAOIF.java>

```
package notice.model.data;

import java.util.Vector;

import notice.model.object.NoticeInf;

import jp.co.intra_mart.framework.base.data.DAOException;

/**
 * 掲示板情報にアクセスするための DAOIF です
 *
 * @author NTTDATA intra-mart
 */
public interface NoticeDAOIF {
   public void insert(NoticeInf notice) throws DAOException;

   public Vector select() throws DAOException;
}
```

今回修正するのは、網掛けされた部分です。

```
public Vector select()
                    throws DAOException;
```

掲示情報取得の select()メソッドを追加しています。返却されるデータは複数の掲示情報ということになりますので、掲示情報モデルオブジェクトを複数保持する Vector クラスという形で返却することを想定しています。

次に実際の掲示情報取得処理を記述する、NoticeIntramartDBDAO クラスを修正します。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 2-9

<C:/imart/doc/imart/WEB-INF/classes/notice

/model/data/NoticeIntramartDBDAO.java>

```
package notice.model.data;

import java.sql.SQLException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Vector;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import notice.model.object.NoticeInf;

import jp.co.intra_mart.foundation.service.client.information.Identifier;
import jp.co.intra_mart.framework.base.data.DAOException;
import jp.co.intra_mart.framework.base.data.LoginGroupDBDAO;


/**
 * 掲示板情報にアクセスするための intra-martDBDAO
 *
 * @author NTTDATA intra-mart
 */
public class NoticeIntramartDBDAO
	extends LoginGroupDBDAO
	implements NoticeDAOIF {

	/**
	 * デフォルトコンストラクタ
	 */
	public NoticeIntramartDBDAO() {
		super();
	}

	//SQL 定義
	private final String SQL_INSERT = "INSERT INTO tutorial_plus_notice " +
		"(notice_cd,title,author,record_date,content) VALUES (?,?,?,?,?)";

	private final String SQL_SELECT = "SELECT notice_cd,title,author,record_date,content" +
		" FROM tutorial_plus_notice ORDER BY record_date ";


	/**
	 * 掲示板情報を登録します
	 * @see notice.model.data.NoticeDAOIF#insert(NoticeInf)
	 * @param notice 掲示板情報
	 */
	public void insert(NoticeInf notice) throws DAOException {
		Connection con = null;
		PreparedStatement stmt = null;
		String noticeCd;
		SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd|HH:mm:ss");
		String registDate = sdf.format(notice.getRegistDate());


		if(notice == null){
			return;
		}

		//DbsConnection の取得
		try{
			con = getConnection();
		} catch(Exception e){
```

```
            throw new DAOException(e.getMessage(), e);
        }

        // 掲示板コードの取得
        noticeCd = Identifier.make();

        //insert 処理
        try{
            stmt = con.prepareStatement(SQL_INSERT);
            stmt.setString(1,noticeCd);
            stmt.setString(2,notice.getTitle());
            stmt.setString(3,notice.getAuthor());
            stmt.setString(4,registDate);
            stmt.setString(5,notice.getContent());
            stmt.executeUpdate();
        }catch(SQLException e){
            throw new DAOException(e.getMessage(), e);
        } finally {
            try{
                if(stmt != null){
                    stmt.close();
                }
            } catch (SQLException e){
            }
        }
    }

    /**
     * 掲示板情報を取得します
     * @see notice.model.data.NoticeDAOIF#insert(NoticeInf)
     * @return 掲示板情報
     */
public Vector select() throws DAOException{
    int i;
    Connection con = null;
    PreparedStatement stmt = null;
    ResultSet result = null;
    Vector notices = new Vector();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd|HH:mm:ss");

    //Connection の取得
    try{
        con = getConnection();
    } catch(Exception e){
        throw new DAOException(e.getMessage(), e);
    }

    //SQL 文発行
    try{
        stmt = con.prepareStatement(SQL_SELECT);
        result = stmt.executeQuery();

        while (result.next()) {
            NoticeInf notice = new NoticeInf();
            Date recordDate = new Date();
            notice.setTitle(result.getString("title"));
            notice.setAuthor(result.getString("author"));
            notice.setRegistDate(sdf.parse(result.getString("record_date")));
            notice.setContent(result.getString("content"));

            notices.add(notice);
        }

    } catch(Exception e){
        throw new DAOException(e.getMessage(), e);
    } finally {
        try{
            if(result != null){
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

```
                        result.close();
                    }
                    if(stmt != null){
                        stmt.close();
                    }
                } catch (SQLException e){
                }
            }

            //掲示板データ返却
            return notices;

        }

}
```

今回追加されたコードは select()メソッド、つまり掲示情報を取得してくるメソッドです。

登録処理同様、冒頭で SQL 文の定義を行っています。

```
private final String SQL_SELECT = "SELECT notice_cd,title,author,record_date,content" +
    " FROM tutorial_plus_notice ORDER BY record_date ";
```

DB データの取得もグループ ID に関連づけられた DB のコネクション（java.sql.Connection）を取得して、そのインスタンスに対して処理を行います。

```
//Connection の取得
try{
    con = getConnection();
} catch(Exception e){
    throw new DAOException(e.getMessage(), e);
}
```

次に、掲示板データを取得するための SQL 文を発行します。

```
stmt = con.prepareStatement(SQL_SELECT);
result = stmt.executeQuery();
```

取得した掲示板データは配列型で返却されます。取得したデータの件数分ループを回して、掲示板情報モデルクラス（NoticeInf）に格納し、返却値である notices(Vector)に設定しています。

```
while (result.next()) {
    NoticeInf notice = new NoticeInf();
    Date recordDate = new Date();
    notice.setTitle(result.getString("title"));
    notice.setAuthor(result.getString("author"));
    notice.setRegistDate(sdf.parse(result.getString("record_date")));
    notice.setContent(result.getString("content"));

    notices.add(notice);
}
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

ここで注意したいのは、dateStr です。NoticeIntramartDBDAO クラスの insert()メソッドを見てもわかるように、tutorial_plus_noticeテーブルのrecord_dateというカラムには、「年/月/日|時:分:秒」という形で文字列でデータが格納されています。掲示情報モデルオブジェクトでは Date クラスのインスタンスとして登録日付を持っていますので、これを Date クラスに変換する必要があります。このため、ここでは java.lang. SimpleDateFormat クラスのメソッドを利用して、文字列を Date 型に変換しています。

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd|HH:mm:ss");
・・・・・
notice.setRegistDate(sdf.parse(result.getString("record_date")));
```

dateStr は、「年/月/日|時:分:秒」という形式になった文字列です。これを、SimpleDateFormat の parse()メソッドにかけると、変数 recordDate に登録日付が Date 型のインスタンスとして格納されます。

以上でデータフレームワークの修正は終了です。

これらのファイルを保存して、メニューから掲示板参照を選んでみましょう。先ほど登録した掲示情報を参照できるようになりましたか？

2.10 処理概要

☆掲示板登録画面初期表示

エラー! 編集中のフィールド コードからは、オブジェクトを作成できません。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

☆掲示板登録画面登録処理（正常）

エラー！ 編集中のフィールド コードからは、オブジェクトを作成できません。

☆掲示板登録画面登録処理（例外）

エラー！ 編集中のフィールド コードからは、オブジェクトを作成できません。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

☆掲示板登録画面登録処理（モデル）

エラー！ 編集中のフィールド コードからは、オブジェクトを作成できません。

☆掲示板参照画面初期表示（概要）

エラー！ 編集中のフィールド コードからは、オブジェクトを作成できません。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

☆掲示板参照画面初期表示（詳細①）

エラー！ 編集中のフィールド コードからは、オブジェクトを作成できません。

☆掲示板参照画面初期表示（詳細②）

エラー！ 編集中のフィールド コードからは、オブジェクトを作成できません。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

3 Strutsとの連携

3.1 Struts連携の概要

intra-mart システム 4.1 から、JakartaプロジェクトのStrutsとの連携ができるようになりました。intra-mart システム 4.3 では、Struts1.1 に対応しています。Strutsは主にWeb層のフレームワークなので、JavaEE フレームワークの中ではサービスフレームワークが担当する処理に該当します。

ここでは、掲示板の登録処理を行っている部分を対象にして、Strutsとの連携をどのように行うかを説明して行きます。

まず、掲示板の登録処理の処理概要を思い出してみましょう。

エラー! 編集中のフィールド コードからは、オブジェクトを作成できません。

図 3.1-a

JavaEE フレームワークでは前遷移ページからの最初の入り口として、ServiceController を利用します。Struts では、この ServiceController の位置を、ActionForm クラスと Action クラスがそれぞれ担当することになります。

このうち、ActionForm クラスは JSP ファイルとの情報の受け渡しに使用されるクラスで、基本的に受け渡しされる情報の getter メソッドと setter メソッドを持ちます。また、入力情報に対するチェック処理(ServiceController の check()メソッドに対応)も ActionForm の validate()メソッドが担当します。

次に Action クラスは、リクエストに対する処理を行います。Action クラスの execute メソッドがその処理を担当することになります(ServiceController の service()メソッドに対応)。ビジネスロジックが必要な場合には、Action クラスの execute()メソッドからビジネスロジックを行うクラスが呼び出されることになります。ビジネスロジックは JavaEE フレームワークではイベントフレームワークが担当することになりますので、Action クラスの execute()メソッドからイベントフレームワークを呼び出すことになります。

このとき、Struts の Action からイベントフレームワークを呼び出すために JavaEE フレームワークで用意されているのが ServiceUtils クラスで、このクラスが持っているスタティックメソッドを用いて Event クラスを取得したりイベントフレームワークの起動処理を行ったりします。

エラー! 編集中のフィールド コードからは、オブジェクトを作成できません。

図 3.1-b

それでは、Struts 対応の掲示情報登録画面を作成して行きましょう。

3.2 web.xmlの編集

Struts1.1 では、Struts の動作を決めるための定義ファイルを分割して設定することができるようになりました。まずは、この掲示板のために分割された定義ファイルである struts-tutorial_plus.xml を使用できるように web.xml を編集します。C:/imart/doc/imart/WEB-INF の直下にある web.xml を開いてください。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 0-a

```
  </servlet>
・・・・・
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>application</param-name>
      <param-value>sample/imart_struts/shopping/ApplicationResources</param-value>
    </init-param>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>config/j2ee/sample/imart_struts/shopping</param-name>
      <param-value>/WEB-INF/struts-shopping.xml</param-value>
    </init-param>
  <init-param>
      <param-name>config/notice</param-name>
      <param-value>/WEB-INF/struts-tutorial_plus.xml</param-value>
  </init-param>
  </servlet>
```

網掛けの部分を編集してください。この記述によって、あとで説明する struts-tutorial_plus.xml が有効になります。また、<param-name>の config/notice という値がこれ以降の定義ファイル(***.xml ファイル)で jsp ファイルなどを指定するときの相対パスの基準パスになることを覚えておいてください。

3.3 メニューへの登録

まずは、Struts 用の画面をメニューに登録しましょう。グループ管理者でログインし、[ログイングループ管理]-[メニュー管理] -[メニュー設定]から以下のような項目でメニュー情報を登録してください。

今回登録する画面は「掲示版登録(Struts)」です。

表示名	掲示板登録画面(Struts)
備考	チュートリアルプラスで利用する Struts 用の掲示板登録画面です。
クライアントタイプ	パソコン
URL	notice/tutorial/menu_to_notice_regist.do
ロール	(このフォルダにアクセスするユーザが保持するロール) ＊当研修では「guest」というロールを指定します

3.4 struts-tutorial_plus.xmlの編集

ここでは、掲示板に必要な定義ファイルである struts-tutorial_plus.xml を編集していきます。

FW4.1 以降をインストールすると、Strutsを使用する準備が整っています 。C:/imart/doc/imart/WEB-INF の直下に struts-tutorial_plus.xml というファイルを作成して、エディタで開いてください。

Source 0-a

<C:/imart/doc/imart/WEB-INF/struts-tutorial_plus.xml>

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE struts-config PUBLIC
                "-//Apache Software Foundation//DTD Struts Configuration 1.0//EN"
                "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<struts-config>
        <!--=============Form Bean Definitions=================-->
  <form-beans>
    <form-bean name = "NoticeRegistForm"
                type = "notice.imart_struts.NoticeRegistForm" />
  </form-beans>

        <!--=============Global Forward Definitions===================-->
  <global-forwards>
    <forward name = "TutorialNoticeRegist"
              path = "/notice_regist_struts.jsp" />
  </global-forwards>

                <!--=============Action Mapping Definitions=================-->
  <action-mappings>
    <action path       = "/tutorial/menu_to_notice_regist"
            forward   = "/notice_regist_struts.jsp"/>
    <action path       = "/notice/notice_regist"
            name        = "NoticeRegistForm"
            validate = "true"
            scope      = "request"
            input      = "/notice_regist_struts.jsp"
            type        = "notice.imart_struts.NoticeRegistAction" />
  </action-mappings>
</struts-config>
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

<form-beans>タグの中の<form-bean>タグで、掲示板登録で使う ActionForm クラスの指定を行っています。すでに説明したように、ActionForm クラスは JSP ファイルと Action クラスとの間で情報の受け渡しを行うのに使用します。

```
<form-bean name = "NoticeRegistForm"
            type = "notice.imart_struts.NoticeRegistForm" />
```

ここでは、NoticeRegistForm という名前で notice.imart_struts.NoticeRegistForm というクラスを指定しています。

<action-mappings>タグの中の<action-mapping>タグでは同じく掲示板で使う Action クラスの指定を行っています。ここでは、先ほどメニュー設定で登録したメニュー遷移用の Aciton の登録と、掲示情報の登録処理用の Action の登録をしています。

```
<action path      = "/tutorial/menu_to_notice_regist"
        forward   = "/notice_regist_struts.jsp"/>
```

これは、メニュー遷移用の指定です。ここでは、Action クラスは使用しないので、ページ遷移のみを定義しています。「/tutorial/menu_to_notice_regist」という名前で、/notice_regist_struts.jsp へのページ遷移を指定します。

ここで、実際にファイルが存在するのは(APP Runtime のドキュメントルート)/notice/ notice_regist_struts.jsp であることに注意してください。先ほど、web.xml を編集したときに<param-name>に config/notice と設定したのを覚えていますでしょうか？config というのは、struts-config.xml(Struts の標準の定義ファイル)が指定されている値です。この config を除いた、/notice というのが struts-tutorial_plus.xml の中で指定されるパスの相対パスになります。つまり、上記で forward 属性に jsp ファイルを指定するときは(Application Runtime のドキュメントルート)/notice からの相対パスを指定することになります。つまり forward 属性の指定は「/notice_regist_struts.jsp」ということになります。

このpath属性の部分に「.do」という拡張子をつけたものが、メニューのページ引数として登録されます(3.2節を参照してください)。

```
<action path      = "/notice/notice_regist"
        name      = "NoticeRegistForm"
        validate = "true"
        scope     = "request"
        input     = "/notice_regist_struts.jsp"
        type      = "notice.imart_struts.NoticeRegistAction" />
```

これは、掲示情報登録用の Action クラスの指定です。path 属性はこの指定に与えられる名前のようなもので、JSP ファイルの<html:form>タグで action 属性が指定されるときに「.do」拡張子をつけて指定されます。name 属性はこの Action に対応する ActionForm を関連付けるものです。<form-bean>タグの name 属性と同じものが指定されていることに注意してください。validate 属性は、入力チェックを行うかどうかのフラグでここでは「true」をしていしています。scope 属性は、この Action に対応する ActionForm クラスのインスタンスの生存期間を設定しています。input 属性は入力チェック時のエラー画面の指定を行います。この場合は、入力エラーが発生したときには元の画面に戻るように設定されています。type 属性は、この<action>タグに紐付けられるクラスを指定します。

<global-forwards>タグの<forward>タグの中では処理が終わった後に遷移するページを指定しています。

```
<forward name = "TutorialNoticeRegist"
        path = "/notice_regist_struts.jsp" />
```

name 属性は、Action クラスのインスタンスから次画面に遷移する時に指定されるキーです。path 属性は遷移先のページを指定しています。

3.5 Struts用のJSPファイルの編集

次に JSP ファイルの修正を行いましょう。前の章で作成した C:/imart//notice/notice_regist.jsp をコピーして、新しく C:/imart//notice/notice_regist_struts.jsp というファイルを作成します。

次にこの新しく作成したファイルをエディタで開いてください。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 3.5-b

<C:/imart//notice/notice_regist_struts.jsp>

```
<%@ page contentType="text/html; charset=Windows-31J" pageEncoding="Windows-31J" %>
<%@ taglib prefix="imartj2ee" uri="http://www.intra-mart.co.jp/taglib/core/framework" %>
<%@ taglib prefix="html" uri="/WEB-INF/struts-html.tld" %>

<html:html>

<HEAD>
  <TITLE>個人掲示板登録画面</TITLE>
</HEAD>

<BODY>

<!-- タイトル-->
<TABLE bgcolor='#00AAFF' width='100%'>
  <TR><TD>
      <FONT color='white' size="+1"><b>  個人掲示板 -登録-  </b></FONT>
  </TD></TR>
</TABLE>

<BR>
<BR>

<html:form action="/notice/notice_regist.do" scope="request" >

<CENTER>

<TABLE border="1">
  <TR>
      <TH bgcolor="lightskyblue" align="center">
          タイトル
      </TH>
      <TD>
          <INPUT name="title" type="text" size="50">
      </TD>
  </TR>
  <TR>
      <TH bgcolor="lightskyblue" align="center">
          内容
      </TH>
      <TD>
          <TEXTAREA name="content" rows="9" cols="50"></TEXTAREA>
      </TD>
  </TR>
</TABLE>

<BR>

<INPUT type="submit" value=" 登 録 ">

</CENTER>

</html:form>

</BODY>

</html:html>
```

例によって、網掛けの部分が今回修正を加えた部分です。

```
<%@ taglib prefix="html" uri="/WEB-INF/struts-html.tld" %>
```

ここでは、struts-html.tld というタグライブラリ定義ファイルに定義されているタグライブラリの使用を宣言しています。このタグライブラリは、Struts で提供されているタグライブラリです。この宣言をすることで、prefix が html:というタグライブラリをしようすることができるようになります。

```
<html:html>
・・・
</html:html>
```

<html:html>タグは、ユーザの Locale オブジェクトから、適切な言語属性を持つ<html>タグを生成します。

```
<html:form action="/notice/notice_regist.do" scope="request" >
```

<html:form>タグはJSPファイルの中でStrutsと連携して動作する<form>タグを生成します。action 属性には、*.do が指定されたときには、struts-tutorial_plus.xml で定義した ActionMapping を呼び出します。ここでは、
/notice/notice_regist
という名前で定義されている ActionMapping が呼び出されます。上で編集した、struts-tutorial_plus.xml と見比べてみてください。action 属性で拡張子を省いた値が、struts-tutorial_plus.xml の<action>タグで指定した path 属性の値と同じということが確認できると思います。

scope 属性は、ActionForm インスタンスの生存期間を設定します。

3.6 ActionFormクラスの編集

次は、ActionForm クラスです。
C:/imart/doc/imart/WEB-INF/classses/notice/imart_struts/NoticeRegistForm.java
というファイルを作成して、エディタで開いてください。

このクラスでは、JSP ファイルと Action クラスの間で情報を受け渡しするための ActionForm を提供します。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

Source 3.6-b

<C:/imart/doc/imart/WEB-INF/classes/notice/imart_struts/

NoticeRegistForm.java>

```
package notice.imart_struts;

import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

/**
 * @author NTTDATA intra-mart
 *
 */
public class NoticeRegistForm extends ActionForm {
  private String title;
  private String content;

  /**
   * デフォルトコンストラクタ
   */
  public NoticeRegistForm() {
        super();
  }

  /**
   * Returns the content.
   * @return String
   */
  public String getContent() {
        return content;
  }

  /**
   * Returns the title.
   * @return String
   */
  public String getTitle() {
        return title;
  }

  /**
   * Sets the content.
   * @param content The content to set
   */
  public void setContent(String content) {
        this.content = content;
  }

  /**
   * Sets the title.
   * @param title The title to set
   */
  public void setTitle(String title) {
        this.title = title;
  }

  /**
   * @see org.apache.struts.action.ActionForm#validate(ActionMapping, HttpServletRequest)
   */
  public ActionErrors validate(ActionMapping arg0, HttpServletRequest arg1) {
        ActionErrors errors = new ActionErrors();
```

```
        if(title != null && title.equals("")){
            ActionError error = new ActionError("title_null_error");
            errors.add(ActionErrors.GLOBAL_ERROR, error);
        }

        if(content != null && content.equals("")){
            ActionError error = new ActionError("content_null_error");
            errors.add(ActionErrors.GLOBAL_ERROR, error);
        }

        return errors;
    }

}
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

まず、NoticeRegistForm クラスのプライベート変数である、title と content に対する setter メソッドと getter メソッドが記述されています。これは、notice_regist_struts.jsp からページ引数として渡される属性名に対応しています。

次に、validate メソッドは、入力チェックを行うためのメソッドです。ここでは、タイトルと内容が空のときに、エラーを返しています。返り値には ActionErrors クラスのインスタンスが返却されます。

ここで、struts-tutorial_plus.xml の<action>タグの内容を振り返ってみましょう。

```
<action path     = "/notice/notice_regist"
        name     = "NoticeRegistForm"
        validate = "true"
        scope    = "request"
        input    = "/notice_regist_struts.jsp"
        type     = "notice.imart_struts.NoticeRegistAction" />
```

エラーが発生したときは、<action>タグの中で、input 属性に指定されたパスの JSP ファイルが呼ばれます。ここでは、掲示板登録画面自身を指定していますので、インプットチェックに引っかかったときには掲示情報をなにも登録せず、元の画面に戻るようになっています。

3.7 Actionクラスの編集

次に Action クラスを継承した NoticeRegistAction クラスを見ていきましょう。
C:/imart/doc/imart/WEB-INF/classses/notice/imart_struts/ NoticeRegistAction.java
というファイルを作成して、次のように記述してください。

Source 3.7-b

<C:/imart/doc/imart/WEB-INF/classes/notice/imart_struts/

NoticeRegistAction.java>

```
package notice.imart_struts;

import java.io.IOException;
import java.util.Date;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import jp.co.intra_mart.framework.extension.common.util.ServiceUtils;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import notice.model.event.NoticeRegistEvent;
import notice.model.object.NoticeInf;

/**
 * @author NTTDATA intra-mart
 *
 */
public class NoticeRegistAction extends Action {

  /**
   * デフォルトコンストラクタ
   */
  public NoticeRegistAction() {
        super();
  }

  /**
   * @see org.apache.struts.action.Action#execute(ActionMapping, ActionForm, ServletRequest, ServletResponse)
   */
  public ActionForward execute(
              ActionMapping mapping,
              ActionForm form,
              HttpServletRequest request,
              HttpServletResponse response)
              throws IOException, ServletException {
        NoticeRegistForm registForm = (NoticeRegistForm)form;
        String title;
        String content;
        Date date;
        String author;

        try {
              // 掲示板情報登録用のインスタンスの初期化
              NoticeInf notice = new NoticeInf();

              // タイトル、内容を ActionForm から取得
              title = registForm.getTitle();
              author = ServiceUtils.getLoginUserID(request, response);
              date = new Date();
              content = registForm.getContent();

              // 掲示板情報を作成
              notice.setTitle(title);
              notice.setAuthor(author);
              notice.setRegistDate(date);
              notice.setContent(content);

              // イベントの呼び出し
```


Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

```
            NoticeRegistEvent event = (NoticeRegistEvent)ServiceUtils.createEvent("notice.conf.notice", "notice_regist",
request, response);
            // イベントへの掲示情報の登録
            event.setNotice(notice);
            // イベントの実行
            ServiceUtils.dispatchEvent(event);
        } catch (Exception e) {
            throw new ServletException(e.getMessage(), e);
        }

        return mapping.findForward("TutorialNoticeRegist");
    }

}
```

このクラスでは、Action クラスから execute()メソッドをオーバーライドしています。このメソッドは ServiceController でいう service()メソッドに相当します。

NoticeRegistForm registForm = (NoticeRegistForm)form;

JSP ファイルから渡される、title と content は NoticeRegistForm クラスを介して取得します。NoticeRegistForm クラスのインスタンスは引数で渡されますので、この行では NoticeRegistForm クラスにキャストしています。

title = registForm.getTitle();
content = registForm.getContent();

値の取得は、NoticeRegistForm クラスで作成した getter メソッドを使用して取得します。

次に、ServiceController で行ったのと同じように、NoticeInf クラスのインスタンスに掲示情報の登録に必要な情報をセットしていきます。

次に Event クラスのインスタンスを取得します。掲示情報登録用の Event クラスは NoticeRegistEvent でした。Struts 上で Event クラスを取得したり、実行(dispatch()メソッド)したりするときには、JavaEE フレームワークで用意されている、ServiceUtils クラスのメソッドを使用します。

NoticeRegistEvent event =
(NoticeRegistEvent)ServiceUtils.createEvent("notice.conf.notice", "notice_regist", request, response);

createEvent()メソッドは、Event クラスのインスタンスを取得するメソッドです。ServiceController で使う createEvent()メソッドのように、第 1 引数と第 2 引数はアプリケーション ID とサービス ID です。ServiceUtils の createEvent()メソッドはこの他に、HttpServletRequest クラスのインスタンスと HttpServletResponse クラスのインスタンスを引数にとります。これは、execute メソッドの引数として渡されているので、そのまま渡します。

ServiceController のところでも説明しましたが、イベントフレームワークにおいて Event クラスのインスタンスは、イベントフレームワークに対して受け渡される情報のいれものとなります。ここでは、掲示情報(NoticeInf クラス)を Event クラスのインスタンスにセットしています。

最後に ServiceUtils クラスの dispatchEvent()でイベントを起動して、execute メソッドの記述は終了です。

Copyright 2000-2011 株式会社 NTT データ イントラマート All rights Reserved.

4 おわりに

以上でチュートリアル plus の内容は終了です。
ここまで読み進んできた読者は、intra-mart JavaEE フレームワークの機能を利用したアプリケーション作成の流れが理解できたと思います。

読んでみただけという方はぜひ 1 度本書のアプリケーションを作成し、動作させてみてください。
intra-mart JavaEE フレームワークに対する理解がより深まると思います。

冒頭にお話ししたように、本書は intra-mart JavaEE フレームワークの仕組みについて概説的に説明したものです。
「より理解を深めたい！」という方は、本書を参考に類似したアプリケーションをご自分の力で作ってみることをお薦めします。

intra-mart JavaEE フレームワークの機能や仕組み、より具体的な開発手法に関して興味を持ってくださった方には**「intra-mart JavaEE フレームワーク中級研修コース」**
の受講をお薦めします。

本書では触れられていない、
ソースコード自動出力機能を利用した効率のよいコンポーネント作成
intra-mart e-builder Framework Producer を利用した開発手法
デバッグ（文法チェック・ブレークポイントを設定したプログラムのトレース実行など）
GUI でのプロパティ設定
などを体験することができます。

intra-mart WebPlatform/AppFramework
im-JavaEE Framework チュートリアル

2011/06/30 第3版

Copyright 2000-2011 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/