intra-mart WebPlatform/AppFramework Ver.7.2

SAStruts+S2JDBC プログラミングガイド

2012/08/03 第 2 版

＜＜　変更履歴　＞＞

変更年月日	変更内容
2011/07/29	初版
2012/08/03	第 2 版 ・3.3.6 [参考] JSPカスタムタグの属性でオブジェクトを渡す場合の留意点を追加

＜＜　目次　＞＞

1　はじめに 2
1.1　用語解説 2
1.2　注意事項 2
1.3　準備 2
2　SAStruts+S2JDBC アーキテクチャ概要 3
2.1　URL とアクションクラスの関係 3
2.1.1　[参考] intra-mart 上での routingfilter マッピング設定 3
2.2　パッケージ構成・クラス構成 4
2.2.1　SMART deploy とパッケージ構成 4
2.2.2　アクションクラスとアクションフォーム 5
2.2.3　サービスクラスとエンティティクラス 5
2.2.4　[参考] ビジネスロジックをどのクラスで実装するか？ 5
3　アプリケーションの作成 6
3.1　Hello World を作ろう 7
3.1.1　名前入力画面 (JSP) 8
3.1.2　こんにちは画面 (JSP) 9
3.1.3　アクションフォーム HelloworldForm 10
3.1.4　アクションクラス HelloworldAction 11
3.1.4.1　[参考] JSP ファイル名に index.jsp を使用する際の注意点 12
3.2　Hello World を拡張してみよう 13
3.2.1　入力バリデーションの追加 13
3.2.1.1　アクションフォームのパラメータに@Required アノテーションを設定 13
3.2.1.2　実行メソッドのバリデータを有効化 13
3.2.1.3　バリデーションエラーメッセージを画面上に表示させる 14
3.2.1.4　メッセージファイルの編集 14
3.2.1.5　[参考] SAStruts 標準のバリデーション機能との違いについて 14
3.2.1.6　動作確認 15
3.2.2　クロスサイトスクリプティング対策 16
3.3　データ一覧参照画面を作ろう 17
3.3.1　テーブル「simple_todo」の作成 18
3.3.2　エンティティクラスおよびエンティティに対するサービスクラスの生成 18
3.3.3　サービスクラスを利用してレコードを取得・表示 22
3.3.4　[参考] アクションクラスの実行メソッド内でのエラー処理 24
3.3.5　[参考] トランザクション制御 25
3.3.5.1　UserTransaction#setRollbackOnly()を利用したロールバック 25
3.3.5.2　UserTransaction オブジェクトを使用したトランザクションの完全手動制御 26
3.3.6　[参考] JSP カスタムタグの属性でオブジェクトを渡す場合の留意点 27

1 はじめに

本ドキュメントは、intra-mart 上で Seasar プロダクトの SAStruts と S2JDBC を組合せたシステム開発のためプログラミングガイドです。SAStruts と S2JDBC を組合せたシステム開発では、SAStruts および S2JDBC は拡張性が高く、さまざまな活用方法がございます。本ドキュメントでは特に、画面デザイン、エラー処理、アクセスセキュリティ等、intra-mart 上でシステムを構築する上でポイントなるテーマに絞って紹介します。

1.1 用語解説

intra-mart WebPlatform Ver.7.2	以下、**IWP** と略します。 **IWP** をインストールしたディレクトリを**<%im_path%>**と略します。
intra-mart DebugServer Ver7.2	intra-mart e Builder で利用するデバッグ専用のサーバです。 以下、**imDS** と略します。 **imDS** をインストールしたディレクトリを**<%ds_path%>**と略します。
intra-mart e Builder Ver7.2	intra-mart 対応アプリケーション作成支援ツールです。 以下、**imEB** と略します。 **imEB** をインストールしたディレクトリを**<%eb_path%>**と略します。
intra-mart Server Manager	システム全体を管理するサーバです。以下、**imSM** と略します。 **imSM** をインストールしたディレクトリを**<%sm_path%>**と略します。
Application Runtime	アプリケーションの実行エンジンです。以下、**AppRuntime** と略します。 **AppRuntime** をインストールしたディレクトリを**<%app_path%>**と略します。
Seasar2	Seasar2 は DI (Dependency Injection) と AOP (Aspect Oriented Programming) をサポートした軽量コンテナです。 詳細はhttp://www.seasar.org/をご覧ください。
SAStruts	Super Agile に Struts と Seasar2 で開発するためのフレームワークです。 詳細はhttp://sastruts.seasar.org/をご覧ください。
S2JDBC	流れるようなインターフェースと 2Way-SQL をサポートした O/R Mapping フレームワークです。Seasar2 プロダクトの一部として提供されています。 詳細はhttp://s2container.seasar.org/2.4/ja/s2jdbc.htmlをご覧ください。

1.2 注意事項

- Struts のバージョンは 1.3.8 です。
- SAStruts を利用する場合、S2Struts が使えません。
- データソースには intra-mart のログイン中のユーザのログイングループデータソースを利用できるように AutoDetectedDataSource を利用します。この場合、intra-mart で定義されたシステムデータソースを S2JDBC から利用することができません。

1.3 準備

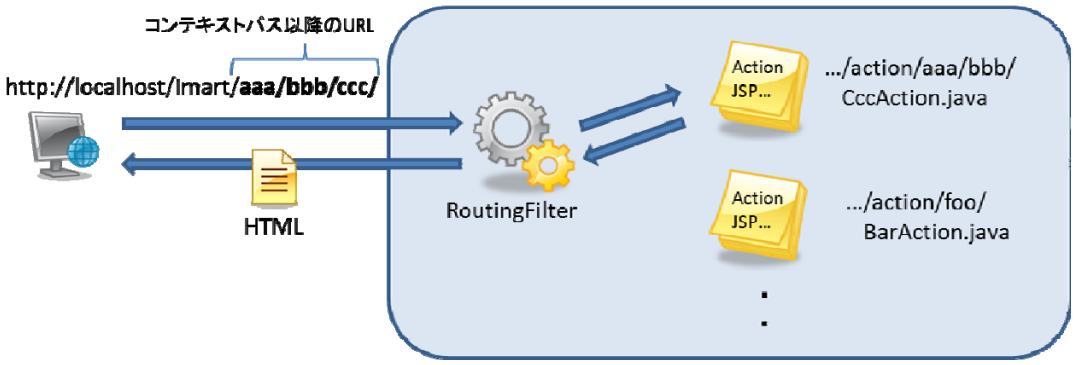
以下のドキュメントを参考に SAStruts+S2JDBC の開発環境をセットアップしてください。

- 「SAStruts+S2JDBC 開発・運用環境構築ガイド」

2 SAStruts+S2JDBC アーキテクチャ概要

2.1 URLとアクションクラスの関係

SAStruts+S2JDBC でアプリケーション開発を行う場合、URL 設計がポイントとなります。コンテキストパス以降の URL は、クラスのパッケージ、クラス、メソッド名に直接関連します。リクエストされた URL から呼び出すクラスおよびそのメソッドを決定しているのが RoutingFilter です。



たとえば上図のように、ホスト名が"localhost"、コンテキストパスが"imart"のとき、http://localhost/imart/aaa/bbb/ccc/ というリクエスト URL に対してマッピングされるクラスおよびメソッドは、以下の優先順位のルールで決まります。

1. jp.co.sample.app.action.**aaa.bbb.Ccc**Action#index()
2. jp.co.sample.app.action.**aaa.bbb**.IndexAction#**ccc**()
3. jp.co.sample.app.action.**aaa.bbb.ccc**.IndexAction#index()

上記例は Seasar2 の SMART deploy における Java ルートパッケージが"jp.co.sample.app"の場合です。

このようなルールに従い、URL から実行するクラスおよびメソッドは自動的に決まります。この実行されるクラスは、Struts におけるアクションクラスに一致します。Struts では、URL と実行するアクションクラスの関係を struts-config.xml に記述します。SAStruts では、上記ルールに従い実行するアクションクラスが決まるため、struts-config.xml の編集が不要です。また Struts ではアクションクラスは、Action クラスを継承する必要がありますが、SAStruts におけるアクションクラスでは特定の親クラスを継承する必要はありません。

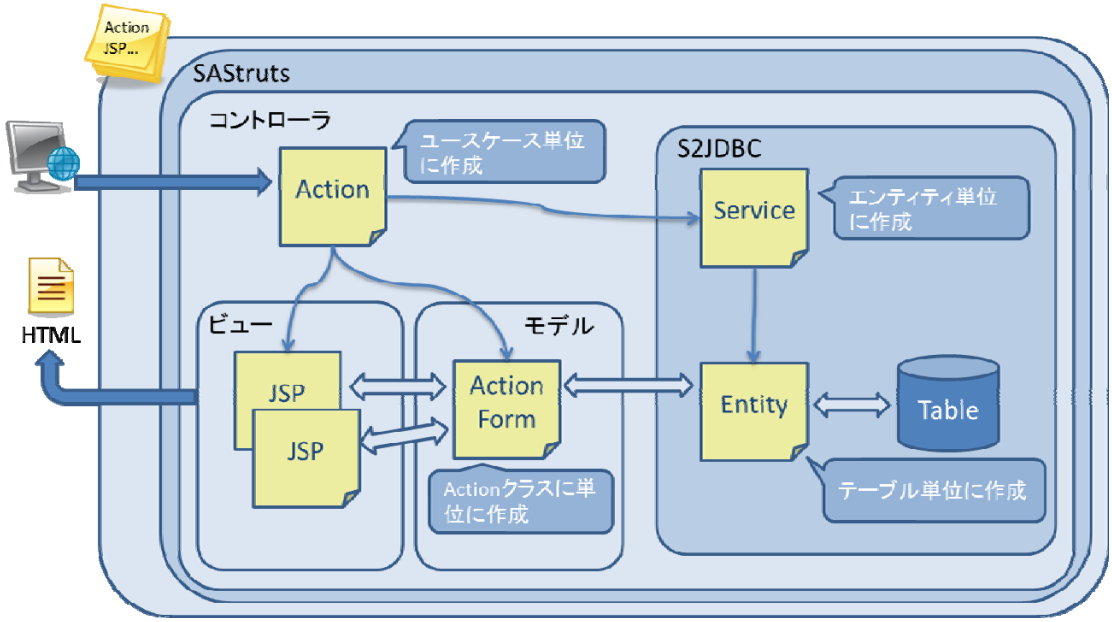
2.1.1 [参考] intra-mart上でのroutingfilterマッピング設定

- intra-mart 上で SAStruts を利用する場合、SAStruts のアプリケーションは、他の Servlet と衝突しないように、スラッシュで終わるリクエストに対してマッピングしています。これ以外の設定での動作検証は行っておりません。
  - [参考] web.xml での設定内容

```
<filter-mapping>
    <filter-name>routingfilter</filter-name>
    <url-pattern>*/</url-pattern>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

2.2 パッケージ構成・クラス構成

本節では、SAStruts+S2JDBC で1つのアプリケーションを開発する場合の基本的なパッケージ構成・クラス構成について説明します。下図は、SAStruts における MVC 構造(モデル:ActionForm、ビュー:JSP、コントローラ:Action)のイメージと Action クラスから S2JDBC を利用するイメージを表現しています。



2.2.1 SMART deployとパッケージ構成

SAStruts では、SMART deploy 対象のルートパッケージの配下に action などのパッケージを作成し、そのパッケージ配下に必要なファイルを格納します。SMART deploy とは、「HOT deploy」、「COOL deploy」、「WARM deploy」の3種類の総称です。Seasar2.4 から奨励されているパッケージ構成等の規約を守る事によって、設定ファイルを追記する必要なく開発を進めることができます。特に「Hot deploy」ではデバッグサーバを再起動せず開発を進めることができます。基本的なパッケージ構成は以下の通りです。

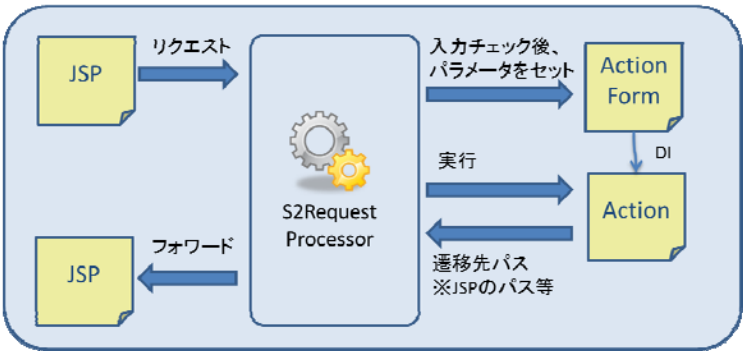
命名規則	格納するクラス
ルートパッケージ.action.*Action	リクエストを処理する SAStruts のアクションクラス
ルートパッケージ.form.*Form	リクエストパラメータを受け取る SAStruts のアクションフォーム
ルートパッケージ.entity.*	テーブルと 1 対 1 となるエンティティクラス
ルートパッケージ.service.*Service	業務ロジックやエンティティを操作するサービスクラス
ルートパッケージ.condition.*	データベースにアクセスする条件を設定するためのクラス
ルートパッケージ.dto.*Dto	アクションフォームやエンティティクラス以外のデータを格納するクラス

SMART deploy に関する詳細な仕様は、Seasar 2.4 の以下のサイトをご覧ください。

- http://s2container.seasar.org/2.4/ja/DIContainer.html

2.2.2 アクションクラスとアクションフォーム

アクションクラスの役割は、リクエストを処理し、遷移先パスを返すことです。具体的には、リクエストを受け取り、ビジネスロジックを実行し、実行結果に応じて遷移先パスを返します。アクションフォームは、リクエストパラメータを受け取り、アクションクラスから利用されます。アクションフォームの各フィールドにアノテーションを付与することで、アクションが実行される前に、入力値バリデーションを行うことができます。



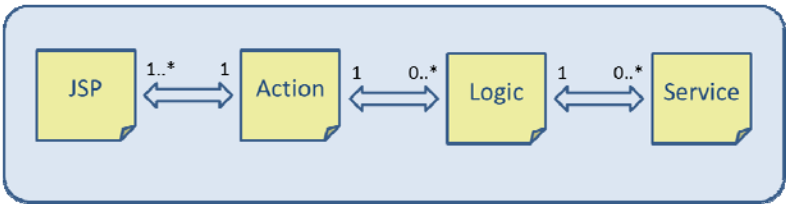
アクションクラスは、アクションフォームを1つだけ持つことができます。アクションクラスは通常、1つのユースケースに対して1つのアクションクラスを作成します。そのため、SAStruts+S2JDBC のアプリケーション開発において、ユースケースの粒度およびアクションクラスの粒度が、設計および実装の重要なポイントとなります。

2.2.3 サービスクラスとエンティティクラス

サービスクラスは、さまざまなビジネスロジックやエンティティを操作するためのクラスです。エンティティクラスは、テーブルと 1 対 1 で作成します。エンティティクラス名はテーブル名に一致します。またエンティティクラスの各プロパティはテーブルのカラム名に一致します。

2.2.4 [参考] ビジネスロジックをどのクラスで実装するか？

アクションクラスから実行するビジネスロジックの実装は、アクションクラス内で行ってもよいし、サービスクラス内で行ってもよいです。サービスクラス内で業務ロジックを実装する場合、さらに im-JavaEE Framework のように、データベースにアクセスする層とビジネスロジック層を完全に分けるような 3 層構造のクラス構成で開発することもできます。



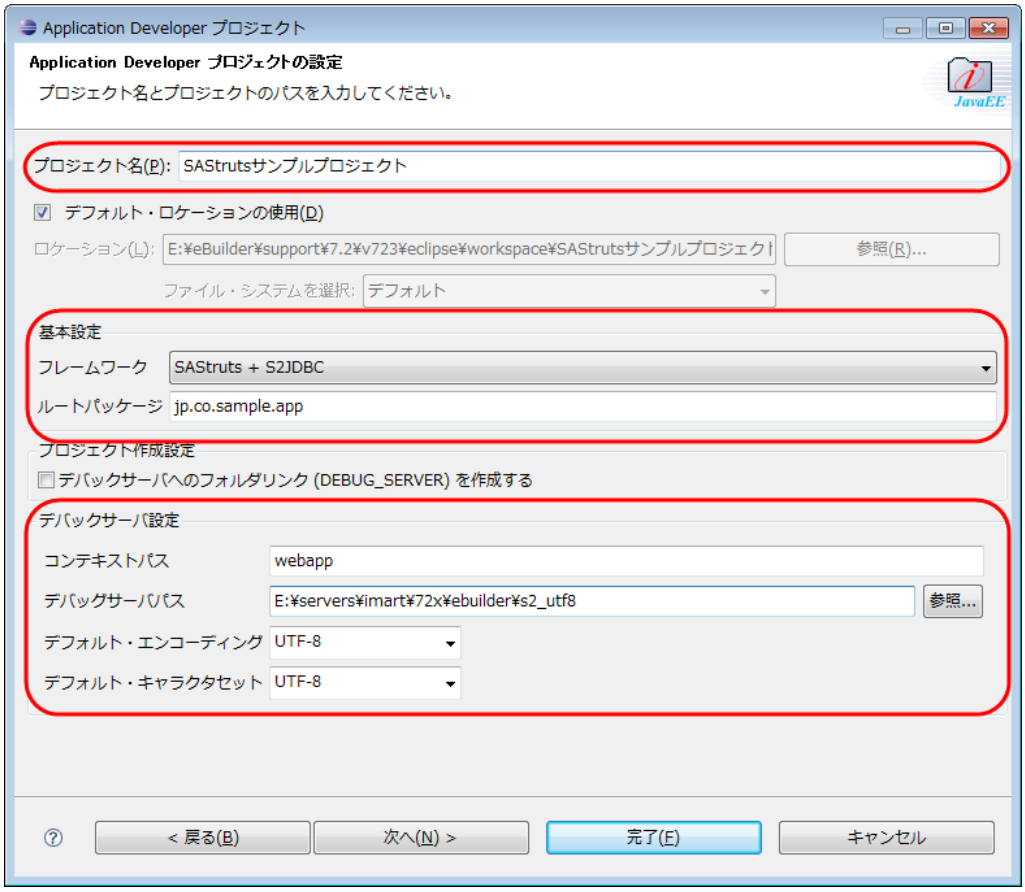
※e Builder Application Producer で開発した場合、上記クラス構成のソースコードが自動生成されます

3 アプリケーションの作成

本節では、e Builderを利用して、intra-mart 上でSAStruts+S2JDBCを中心としたアプリケーション作成方法を紹介します。準備として、以下を行ってください。

1. e Builder を起動してください。
2. Application Developer プロジェクトを作成してください。

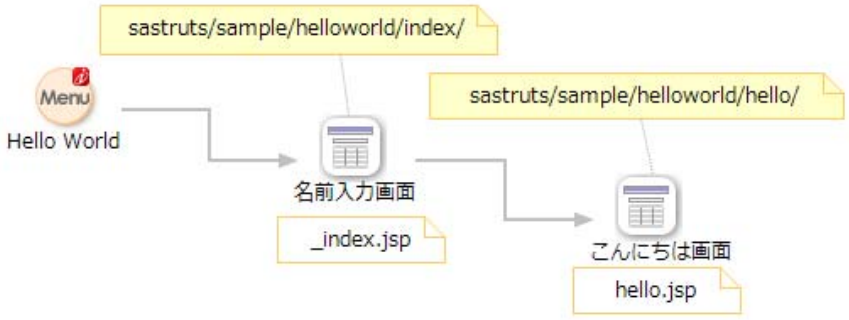
ウィンドウメニュー[ファイル]-[新規]-[プロジェクト]から[intra-mart Application Developer]-[Application Developer プロジェクト]を選択し、下記のように必要な情報を入力してください。



※以降、ルートパッケージには「jp.co.sample.app」を指定した前提で説明します。

3.1 Hello Worldを作ろう

本節では、以下のような画面遷移を行う「HelloWorld」を作成します。



それぞれの画面には以下の URL でアクセスできるものとします。

画面名	URL
名前入力画面	sastruts/sample/helloworld/index/
こんにちは画面	sastruts/sample/helloworld/hello/

Hello World のサンプル実装を以下に示します。作成する JSP ファイルおよび Java クラスファイルは以下とします。

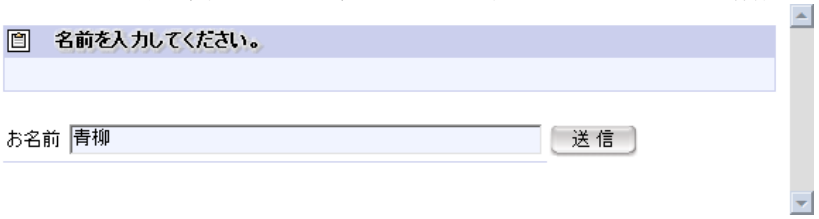
作成ファイル	JSP ファイル名/クラス名
名前入力画面 (JSP)	sastruts/sample/helloworld/_index.jsp
こんにちは画面 (JSP)	sastruts/sample/helloworld/hello.jsp
アクションフォーム (Java)	jp/co/sample/app/form/sastruts/sample/HelloworldForm.java
アクションクラス (Java)	jp/co/sample/app/action/sastruts/sample/HelloworldAction.java

処理の流れは以下のようになります。

1. メニューをクリックすると、/sastruts/sample/helloworld/index/ がリクエストされる。
2. sastruts/sample/helloworld/index/ がリクエストされると実行メソッド HelloworldAction#index()が実行され、_index.jsp にフォワードされ、「名前入力画面」が表示される。
3. 「名前入力画面」で名前を入力し、送信ボタンをクリックすると、/sastruts/sample/helloworld/hello/ がリクエストされる。
4. /sastruts/sample/helloworld/hello/ がリクエストされると実行メソッド HelloworldAction#hello()が実行される。このとき、アクションクラスに設定したアクションフォームに、入力された名前が設定されている。
5. HelloworldAction#hello()は、hello.jsp にフォワードします。hello.jsp では「名前入力画面」で入力された名前をアクションフォームから参照し、「こんにちは、XX さん」とあいさつ文として表示する。

3.1.1 名前入力画面 (JSP)

以下の画面イメージように、名前を入力し、その入力した名前を送信できる画面を作成します。



■ sastruts/sample/helloworld/_index.jsp

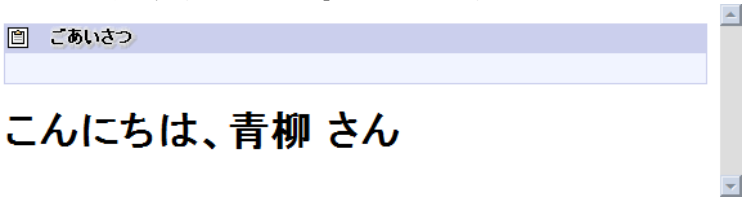
```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/foundation/imarttag"%>
<%@ taglib prefix="html" uri="http://struts.apache.org/tags-html"%>
<%@taglib prefix="s" uri="http://sastruts.seasar.org"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="f" uri="http://sastruts.seasar.org/functions"%>

<html>
  <head>

      <title>タイトル</title>
      <imarttag:imartDesignCss />
      <script type="text/javascript">
      </script>
  </head>
  <body>

      <!-- タイトルバー -->
      <imarttag:imartTitleBar title="名前を入力してください。" />
      <imarttag:imartToolbarFrame>
      </imarttag:imartToolbarFrame>

      <br/>

      <s:form action="hello/" styleId="sayHelloForm">
          <table class="edit">
              <tr>
                  <!-- 名前入力項目 -->
                  <imarttag:imartItemNameTd attr="width=¥"150¥" " name="お名前" />
                  <imarttag:imartInputTd name="inputedName" size="50" type="text" />
                  <!-- 送信ボタン -->
                  <td><table class="button"><tr>
                      <td class="button_padding"><img src="${f:url('/images/standard/button_left.gif')}" /></td>
                      <td class="button_bg"><input class="button_bg" type="submit" value=" 送 信 " /></td>
                      <td class="button_padding"><img src="${f:url('/images/standard/button_right.gif')}" /></td>
                  </tr></table></td>
              </tr>
          </table>
      </s:form>

  </body>
</html>
```

JSP では、imartTitleBar タグ等の intra-mart の画面系のタグライブラリを利用できます。そのため、intra-mart の画面デザインガイドラインに沿った画面を開発することができます。

フォームタグに関して、上記実装は SAStruts で提供される<s:form>タグを利用しています。<s:form>タグでは、action 属性に指定したパスが「/」で始まっていない場合、JSP ファイルから見た相対パスが計算されて設定されます。つまり、"hello/"が再計算されるとここでは"/sastruts/sample/helloworld/hello/"となります。また action 属性に指定したパスが「/」で始まっている場合、コンテキストルートから見た絶対パスになります。

3.1.2 こんにちは画面 (JSP)

以下の画面イメージように、「名前入力画面」で入力された名前を画面上にそのまま表示します。



■ sastruts/sample/helloworld/hello.jsp

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/foundation/imarttag"%>
<%@ taglib prefix="html" uri="http://struts.apache.org/tags-html"%>
<%@taglib prefix="s" uri="http://sastruts.seasar.org"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="f" uri="http://sastruts.seasar.org/functions"%>

<html>
  <head>

      <title>ごあいさつ</title>
      <imarttag:imartDesignCss />
  </head>
  <body>

      <!-- タイトルバー -->
      <imarttag:imartTitleBar title="ごあいさつ" />
      <imarttag:imartToolbarFrame>
      </imarttag:imartToolbarFrame>

      <h1>こんにちは、${helloworldForm.inputedName} さん</h1>

  </body>
</html>
```

JSP では、アクションクラスの public なフィールドの値を参照することができます。

3.1.3 アクションフォーム HelloworldForm

■ jp/co/sample/app/form/sastruts/sample/HelloworldForm.java

```
package jp.co.sample.app.form.sastruts.sample;

import java.io.Serializable;
import org.apache.commons.lang.builder.ReflectionToStringBuilder;
import org.apache.commons.lang.builder.ToStringStyle;

/**
 * アクションフォーム。
 */
public class HelloworldForm implements Serializable {

   private static final long serialVersionUID = 1L;

   /**
    * 入力された名前。
    */
   public String inputedName;

   @Override
   public String toString() {
         return new ReflectionToStringBuilder(this,
                     ToStringStyle.MULTI_LINE_STYLE).toString();
   }
}
```

アクションフォームは、リクエストパラメータを格納するクラスです。そのため、リクエストパラメータと同じ名前のプロパティをアクションフォームに定義します。本サンプルでは、「名前入力画面」でリクエスパラメータ名を「inputedName」としているため、アクションフォームのプロパティ名は「inputedName」となります。

Seasar2 では public フィールドをプロパティとみなすことができるため、getter および setter メソッドは記述する必要はありません。また#toString()メソッドをオーバライドしておくことでデバッグしやすくなります。

3.1.4 アクションクラス HelloworldAction

■ jp/co/sample/app/action/sastruts/sample/HelloworldAction.java

```
package jp.co.sample.app.action.sastruts.sample;

import javax.annotation.Resource;
import jp.co.sample.app.form.sastruts.sample.HelloworldForm;
import org.seasar.struts.annotation.ActionForm;
import org.seasar.struts.annotation.Execute;

/**
 * Hello World アクションクラス。
 *
 * @author 株式会社 NTT データイントラマート
 * @version 0.1
 */
public class HelloworldAction {

  /**
   * アクションフォーム。
   */
  @ActionForm
  @Resource
  public HelloworldForm helloworldForm;

  /**
   * 名前入力画面表示処理。
   * @author 株式会社 NTT データイントラマート
   * @version 0.1
   */
  @Execute(validator = false)
  public String index() {
        // 名前入力画面
        return "_index.jsp";
  }

  /**
   * こんにちは画面表示処理。
   * @author 株式会社 NTT データイントラマート
   * @version 0.1
   */
  @Execute(validator = false)
  public String hello() {
        // こんにちは画面
        return "hello.jsp";
  }
}
```

アクションクラスの役割は、リクエストを処理し、遷移先パスを返すことです。その処理を行うメソッドを「実行メソッド」と呼びます。実行メソッドは、名前は任意、 @Execute アノテーションがついていて、戻り値は String、引数は無しとなります。本サンプルでは2つの実行メソッドが定義・実装されています。実行メソッドの戻り値は、遷移先のパスです。デフォルトでは、フォワードで遷移します。リダイレクトで遷移したい場合、パスの最後に redirect=true を追加してください。実行メソッドの詳しい説明は、以下のサイトをご覧ください。

http://sastruts.seasar.org/featureReference.html#ExecuteMethod

アクションフォームを利用するには、@ActionForm と@Resource アノテーションをつけます。@Resource アノテーションは、フィールドを設定するように Seasar2 に指示するためのアノテーションです。フィールド名は、クラス名の先頭を小文字にしたものにします。

3.1.4.1 [参考] JSPファイル名にindex.jsp を使用する際の注意点

Application Server に Resin を使用する場合、JSP のファイル名にはご注意ください。Resin の場合、標準設定では、スラッシュで終わるリクエスト URL に対してインデックスページ(welcome-file)が登録されています。これは conf/app-default.xml で設定されています。SAStruts の routingfilter もスラッシュで終わるリクエスト URL に対して設定されていますが、スラッシュで終わるリクエストURL の場合、そのパスに index.jsp が存在すると index.jsp が優先されます。たとえば、http://localhost/imart/foo/bar/ というリクエストがされたときに、foo/bar/index.jsp が存在すれば、その index.jsp が直接実行されます。このように、アクションクラスの実行メソッドが実行されず、直接 index.jsp が実行され、予期しない動作をすることがあります。

JSP ファイルを webapp 以下に配置する場合には、この resin の標準機能を避けるために、app-default.xml で welcome-file から index.jsp の設定を削除するか、JSP のファイル名に index.jsp を使わないようにしてください。
※3.1.1節「名前入力画面」のJSPファイル名には、"index.jsp"ではなく"_index.jsp"としています。

また SAStruts の機能には、JSP などの View 用のファイルを置くディレクトリを指定できる機能があります。これは web.xml で sastruts.VIEW_PREFIX パラメータを指定します。JSP ファイルは通常、ブラウザから直接アクセスされないように/WEB-INF 配下のディレクトリを指定するのが良いとされています。/WEB-INF 配下であれば、app-default.xml で welcome-file に index.jsp が設定されていても、JSP ファイル名に index.jsp を利用できます。

Application Producer のソースコード生成機能や業務スケルトンでは、sastruts.VIEW_PREFIX の値が設定されていると、index.jsp の生成ルールが自動的に変わるようになっています。具体的には以下の通りです。

- sastruts.VIEW_PREFIX が設定されていない場合、"_index.jsp"で生成されます。
- sastruts.VIEW_PREFIX が設定されている場合、"index.jsp"で生成されます。

sastruts.VIEW_PREFIX については、以下のページを参考にしてください。

- http://sastruts.seasar.org/fileReference.html

3.2 Hello Worldを拡張してみよう

ここでは、前節で作成した Hello World を拡張します。

3.2.1 入力バリデーションの追加

ここでは、名前が入力されずに送信ボタンをクリックされたときに、「お名前を入力してください」と入力バリデーション設定をする方法を紹介します。

SAStruts では、アクションフォームの各プロパティに入力バリデーション用のアノテーションを設定することで簡単にバリデーションを設定できます。ここでは、@Required アノテーションを利用します。以下その手順を紹介します。

3.2.1.1 アクションフォームのパラメータに@Requiredアノテーションを設定

■ jp/co/sample/app/form/sastruts/sample/HelloworldForm.java

```
import org.seasar.struts.annotation.Required;

/**
 * アクションフォーム。
 */
public class HelloworldForm implements Serializable {

  :（省略）

  /**
   * 入力された名前。
   */
  @Required
  public String inputedName;

  :（省略）

}
```

フィールド inputedName に@Required アノテーションを設定します。

3.2.1.2 実行メソッドのバリデータを有効化

■ jp/co/sample/app/action/sastruts/sample/HelloworldAction.java

```
/**
 * Hello World アクションクラス。
 *
 * @author 株式会社 NTT データイントラマート
 * @version 0.1
 */
public class HelloworldAction {

  :（省略）

  /**
   * こんにちは画面表示処理。
   * @author 株式会社 NTT データイントラマート
   * @version 0.1
   */
  @Execute(validator = true, input = "/index.jsp")
  public String hello() {
      // こんにちは画面
      return "hello.jsp";
  }
}
```

@Execute アノテーションの validator 属性を true に変更し、バリデーションエラー時の遷移先ページを input 属性で指定します。

3.2.1.3 バリデーションエラーメッセージを画面上に表示させる

■ sastruts/sample/helloworld/_index.jsp

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

  :(省略)

<html>
  <head>

  :(省略)

  </head>
  <body>

      <!-- 入力バリデーションエラー情報 -->
      <html:errors />

      <!-- タイトルバー -->
      <imarttag:imartTitleBar title="名前を入力してください。" />

  :(省略)
</html>
```

バリデーションエラーメッセージを画面上に表示させるために、上記のように表示させたい箇所に<html:errors /> タグを配置します。

3.2.1.4 メッセージファイルの編集

バリデーションエラー時のメッセージは、以下のファイルで定義されています。

■ <%sm_path%>/conf/message/sastruts-validator-message_ja.properties

```
 :(省略)
# 入力項目ラベル
labels.inputedName=お名前
```

入力項目ラベルの定義を、このファイル内、もしくは入力項目ラベル用に properties ファイルを作成しそのファイル内に追加してください。入力項目ラベルのキー名は「"labels" + "." + "プロパティ名"」で構成されます。この例では、アクションフォームにおけるプロパティ名が"inputedName"であることから、キー名は"labels.inputedName"となります。

3.2.1.5 [参考] SAStruts 標準のバリデーション機能との違いについて

intra-mart では、SAStruts のバリデーション機能を以下のように拡張しております。

■ ログインユーザのロケールに応じたエラーメッセージを取得

■ 複数パラメータの場合にメッセージに対してパラメータのインデックス(1～)を渡せるように拡張

「SAStruts+S2JDBC 環境構築ツール」で環境を構築した場合、上記機能を利用することができます。SAStruts 標準のバリデーションを利用したい場合は以下のファイルの設定を SAStruts 標準の内容に変更してください。

■ <%app_path%>/doc/imart/WEB-INF/validator-rules.xml の置き換え
※各バリデータは IMFieldChecks クラスのメソッドを利用するように設定されています。S2FieldChecks を使った定義に変更してください。

■ <%app_path%>/doc/imart/WEB-INF/struts-config.xml の message-resources タグの変更
※以下のように変更してください。

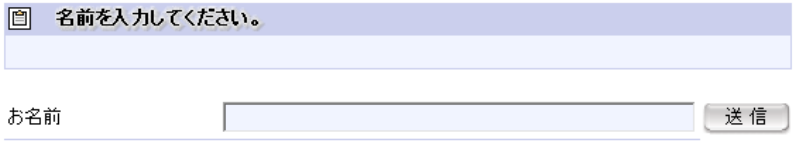
```
<message-resources parameter="application"
    factory="org.seasar.struts.util.S2PropertyMessageResourcesFactory"/>
```

■ <%app_path%>/doc/imart/WEB-INF/classes/application.properties の配置

3.2.1.6 動作確認

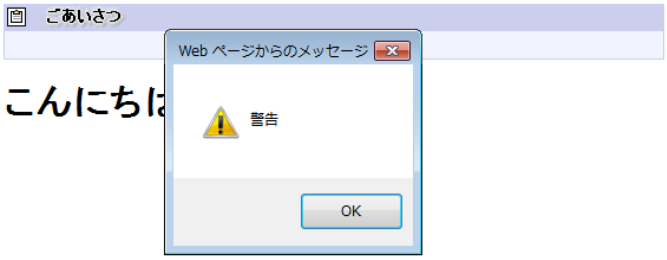
1. 名前を入力せず送信ボタンをクリックする
2. 下図イメージのように、タイトルバーの上部にエラーメッセージが表示される

- 「お名前」は必ず入力してください。



3.2.2 クロスサイトスクリプティング対策

現在のサンプルプログラムでは、クロスサイトスクリプティング対策がされておりません。たとえば「名前入力画面」でお名前の入力欄に「<script>alert("警告")</script>」と入力して送信ボタンをクリックすると、遷移先の「こんにちは画面」が表示されたタイミングで以下のようにアラートが表示されてしまいます。



これを解決するためには、遷移先の「こんにちは画面」で受け取ったパラメータを HTML エスケープする必要があります。SAStruts では、JSP でこの対策を行えるように EL 関数「$f:h(プロパティ名)」が用意されております。

■ sastruts/sample/helloworld/hello.jsp

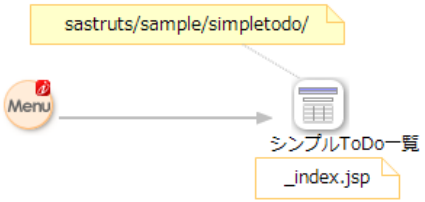
```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

  :(省略)
<%@ taglib prefix="f" uri="http://sastruts.seasar.org/functions"%>

<html>
  <head>

  :(省略)

      <h1>こんにちは、${f:h(helloworldForm.inputedName)} さん</h1>

  </body>
</html>
```

上記のように対策することで、以下のように、script タグがエスケープされ、アラートが表示されません。

3.3 データー覧参照画面を作ろう

本節では、S2JDBC を利用して、以下のような「シンプル ToDo」テーブルのデータを一覧表示するアプリケーションを作成します。

■ 画面遷移



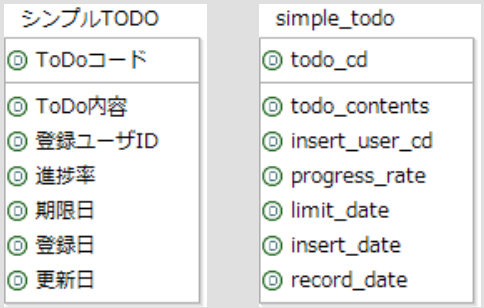
■ 一覧表示画面イメージ

シンプルToDo一覧

ToDo内容	登録者ユーザID	進捗率	期限日	登録日	更新日
定時退社	aoyagi	0		2010/06/28	2010/06/30
見積書作成	aoyagi	50	2010/06/30	2010/06/27	2010/06/29
勤怠申請	aoyagi	0	2010/07/01	2010/06/28	2010/06/30

■ テーブル構造

```
CREATE TABLE simple_todo (
    todo_cd             VARCHAR(15) NOT NULL,
    todo_contents       VARCHAR(400) NOT NULL,
    insert_user_cd      VARCHAR(50) NOT NULL,
    progress_rate       DECIMAL(3),
    limit_date          DATE,
    insert_date         TIMESTAMP,
    record_date         TIMESTAMP,
    PRIMARY KEY (todo_cd)
);
```



作成する JSP ファイルおよび Java クラスファイルは以下とします。

作成ファイル	JSP ファイル名/クラス名	備考
一覧表示画面 (JSP)	sastruts/sample/simpletodo/_index.jsp	
アクションクラス (Java)	jp/co/sample/app/action/sastruts/sample/SimpletodoAction.java	
エンティティクラス (Java)	jp/co/sample/app/entity/sastruts/sample/SimpleTodo.java	S2JDBC-GEN より生成
サービスクラス (Java)	jp/co/sample/app/service/sastruts/sample/SimpleTodoService.java	S2JDBC-GEN より生成

作成手順は以下の通りです。

1. ログイングループデータベースにテーブル「simple_todo」を作成します。
2. 業務スケルトン[Seasar]-[S2JDBC]-[エンティティ生成]を利用してエンティティクラスおよびエンティティに対するサービスクラスを生成します。
3. アクションクラスおよび JSP を作成し、サービスクラスを利用してレコードを取得し表示する処理を追加します。

3.3.1 テーブル「simple_todo」の作成

ログイングループデータベースに以下のテーブル「simple_todo」を作成します。

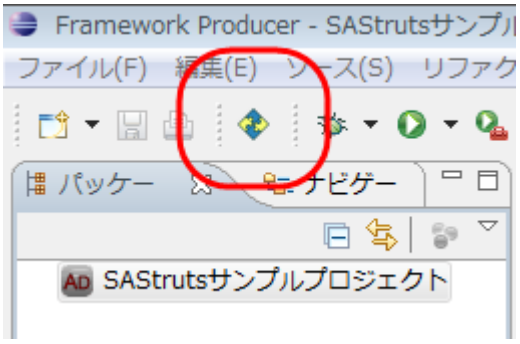
```
CREATE TABLE simple_todo (
   todo_cd             VARCHAR(15) NOT NULL,
   todo_contents       VARCHAR(400) NOT NULL,
   insert_user_cd      VARCHAR(50) NOT NULL,
   progress_rate       DECIMAL(3),
   limit_date          DATE,
   insert_date         TIMESTAMP,
   record_date         TIMESTAMP,
   PRIMARY KEY (todo_cd)
);
```

次節で説明する業務スケルトン[Seasar]-[S2JDBC]-[エンティティ生成]を利用するためには、事前にデータベースにテーブルが作成されている必要があります。

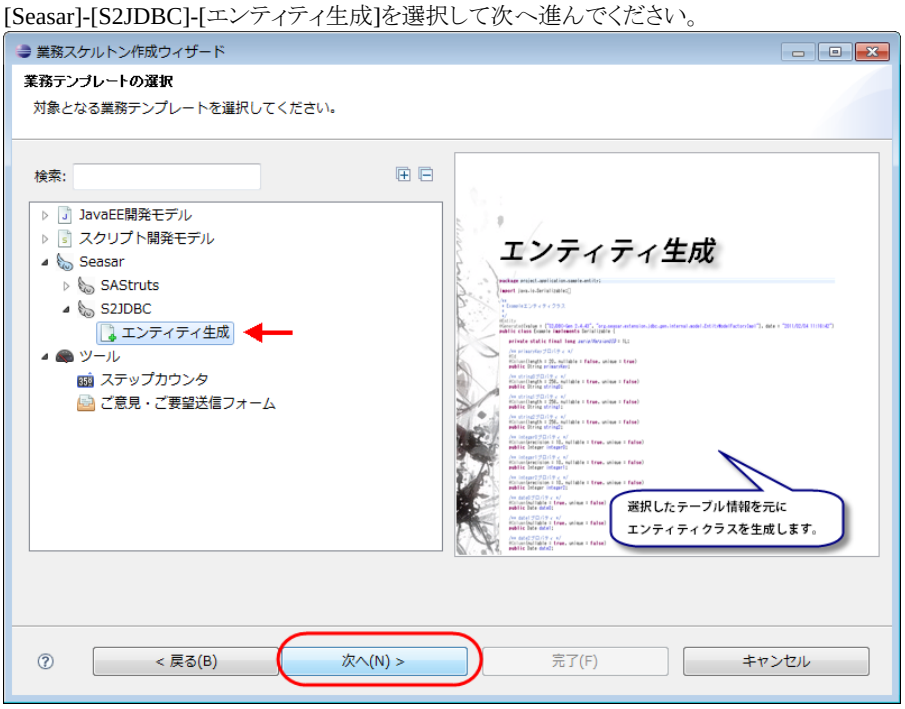
3.3.2 エンティティクラスおよびエンティティに対するサービスクラスの生成

業務スケルトン[Seasar]-[S2JDBC]-[エンティティ生成]を利用して、エンティティクラスおよびエンティティに対するサービスクラスを生成します。

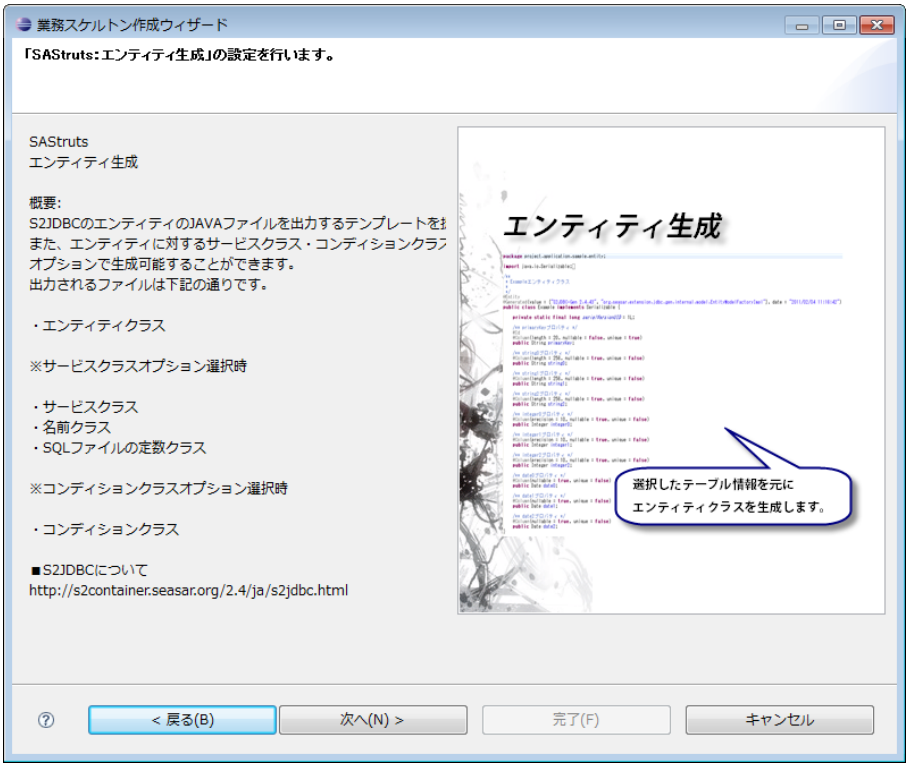
1. Application Developer プロジェクト「SAStruts サンプルプロジェクト」を選択し、下図アイコンをクリックして業務スケルトンウィザードを開いてください。



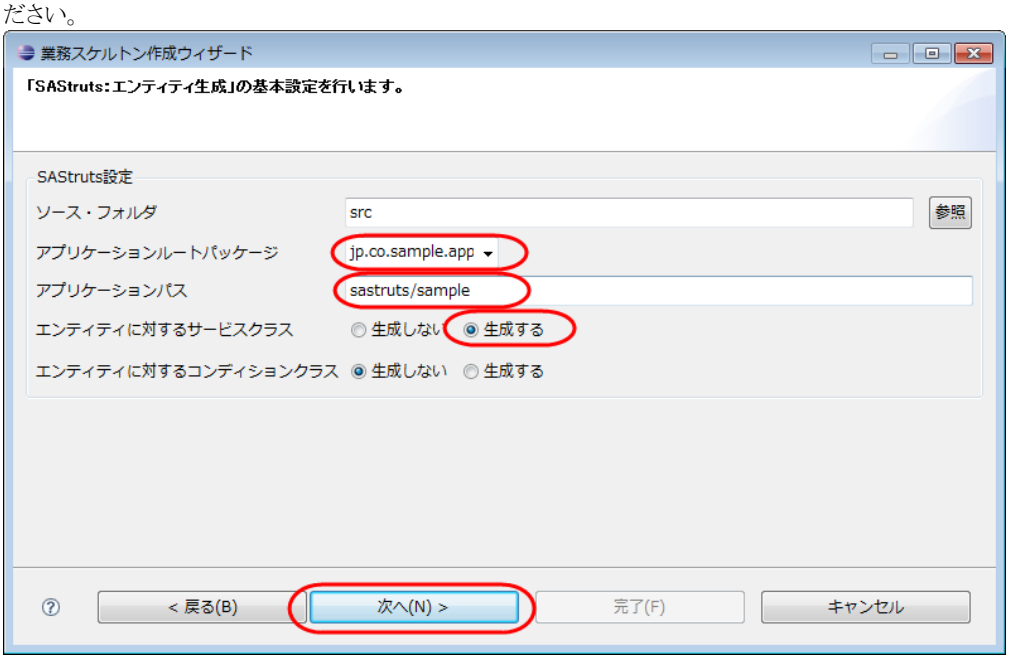
2. 起動した業務スケルトンウィザードから「業務テンプレートの選択」ページまで進み、[Seasar]-[S2JDBC]-[エンティティ生成]を選択して次へ進んでください。



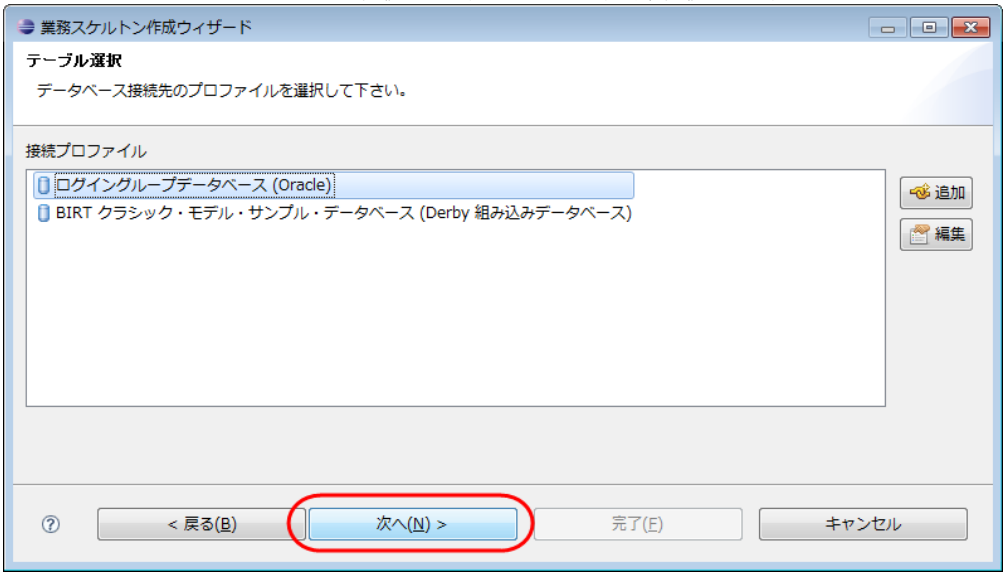
3. 説明のページが表示され、次へ進んでください。



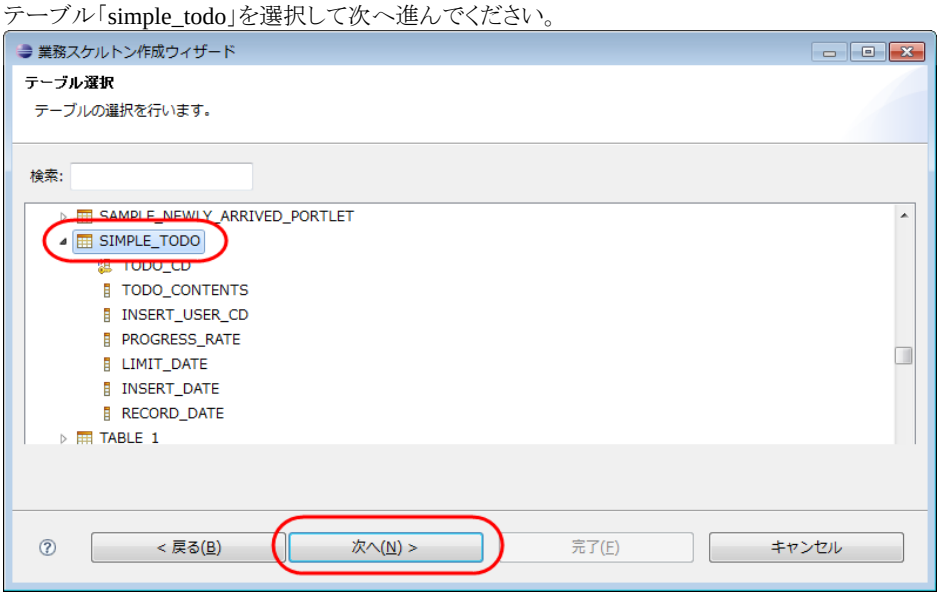
4. 「基本設定」ページで、アプリケーションルートパッケージに「jp.co.sample.app」、アプリケーションパスに「sastruts/sample」、エンティティに対するサービスクラスを「生成する」にそれぞれ選択して次へ進んでください。



5. ログイングループデータベースの接続プロファイルを選択して次へ進んでください。
※ログイングループデータベースの接続プロファイルがない場合、接続プロファイルを追加してください。



6. テーブル「simple_todo」を選択して次へ進んでください。



「設定項目の確認」ページで設定に誤りがないか確認し、問題なければ「完了ボタン」をクリックしてください。
クリックすると、以下のファイルが生成されます。

- src/jp/co/sample/app/entity/sastruts/sample/Names.java
- src/jp/co/sample/app/entity/sastruts/sample/SimpleTodo.java
- src/jp/co/sample/app/entity/sastruts/sample/SimpleTodoNames.java
- src/jp/co/sample/app/service/sastruts/sample/AbstractService.java
- src/jp/co/sample/app/service/sastruts/sample/SimpleTodoService.java
- src/jp/co/sample/app/service/sastruts/sample/SqlFiles.java

3.3.3 サービスクラスを利用してレコードを取得・表示

アクションクラス SimpletodoAction の index メソッドで、サービスクラスを利用してテーブル「simple_todo」のレコードを取得し、取得したレコードを JSP で一覧表示する処理を実装します。実装例は以下の通りです。

■ jp/co/sample/app/action/sastruts/sample/SimpletodoAction.java

```
package jp.co.sample.app.action.sastruts.sample;

import java.util.List;
import javax.annotation.Resource;
import jp.co.intra_mart.framework.extension.seasar.struts.exception.ApplicationRuntimeException;
import jp.co.sample.app.entity.sastruts.sample.SimpleTodo;
import jp.co.sample.app.service.sastruts.sample.SimpleTodoService;

import org.seasar.struts.annotation.Execute;

public class SimpletodoAction {

   /**
    * サービスクラス。
    */
   @Resource
   protected SimpleTodoService simpleTodoService;

   /**
    * Todo レコード
    */
   public List<SimpleTodo> records;

   /**
    * シンプル ToDo 一覧画面表示アクション。<br/>
    */
   @Execute(validator = false)
   public String index() {
         records = simpleTodoService.findAll();
         return "/sastruts/sample/simpletodo/_index.jsp";
   }

}
```

SimpleTodoService#findAll()メソッドを利用して全データを取得します。取得したデータを public なプロパティ records に格納しています。public なプロパティに格納することで、JSP から参照することができます。
またプロパティ simpleTodoService は、@Resource アノテーションを付与することで自動バインディングされます。そのためサービスクラスのインスタンス生成は行う必要がありません。

■ sastruts/sample/simpletodo/_index.jsp

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="imarttag" uri="http://www.intra-mart.co.jp/taglib/foundation/imarttag"%>
<%@ taglib prefix="imartj2ee" uri="http://www.intra-mart.co.jp/taglib/core/framework"%>
<%@ taglib prefix="imart" uri="http://www.intra-mart.co.jp/taglib/core/standard"%>
<%@ taglib prefix="eb" uri="http://www.intra-mart.jp/taglib/eb/functions"%>
<%@taglib prefix="s" uri="http://sastruts.seasar.org"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="f" uri="http://sastruts.seasar.org/functions"%>

<html>
  <head>

      <title>シンプル ToDo 一覧</title>
      <imarttag:imartDesignCss/>
  </head>
  <body>

      <!-- タイトルバー -->
      <imarttag:imartTitleBar title="シンプル ToDo 一覧"/>
      <imarttag:imartToolbarFrame>
          <imarttag:imartToolbarLeft>
          </imarttag:imartToolbarLeft>
          <imarttag:imartToolbarRight>
          </imarttag:imartToolbarRight>
      </imarttag:imartToolbarFrame>

      <br/>

      <!-- 一覧表示 -->
      <table class="table_border_bg" width="100%">
          <tr>
              <td class="list_title_bg_center" nowrap="nowrap">ToDo 内容</td>
              <td class="list_title_bg_center" nowrap="nowrap">登録者ユーザ ID</td>
              <td class="list_title_bg_center" nowrap="nowrap">進捗率</td>
              <td class="list_title_bg_center" nowrap="nowrap">期限日</td>
              <td class="list_title_bg_center" nowrap="nowrap">登録日</td>
              <td class="list_title_bg_center" nowrap="nowrap">更新日</td>
          </tr>

          <c:forEach items="${records}" var="simpletodo" varStatus="idx">
          <tr>
              <!-- シンプル ToDo -->
              <td class="list_data_bg_left" nowrap="nowrap">
                  ${f:h(simpletodo.todoContents)}
              </td>
              <td class="list_data_bg" nowrap="nowrap">
                  ${f:h(simpletodo.insertUserCd)}
              </td>
              <td class="list_data_bg_right" nowrap="nowrap">
                  ${f:h(simpletodo.progressRate)}
              </td>
              <td class="list_data_bg" nowrap="nowrap">
                  ${eb:formatDate(simpletodo.limitDate, "yyyy/MM/dd")}
              </td>
              <td class="list_data_bg" nowrap="nowrap">
                  ${eb:formatDate(simpletodo.insertDate, "yyyy/MM/dd")}
              </td>
              <td class="list_data_bg" nowrap="nowrap">
                  ${eb:formatDate(simpletodo.recordDate, "yyyy/MM/dd")}
              </td>
          </tr>
          </c:forEach>
      </table>

  </body>
</html>
```

3.3.4 [参考] アクションクラスの実行メソッド内でのエラー処理

Struts では、アクションクラスの実行メソッド内でスローされた例外クラスの種類に応じて、適切なエラー処理を行うように設定を組み込む仕組みがあります。たとえば、特定の例外がスローされたときにだけシステム管理者に通知する処理を組み込むこともできます。このように特定の例外に対して、エラー処理を組み込む設定は struts-config.xml の global-exceptions で行います。

標準では、以下のような3種類のエラー処理が設定されています。

エラー	エラー処理内容
アプリケーションエラー ApplicationRuntimeException	ApplicationRuntimeException に対して、ApplicationRuntimeExceptionHandler を設定しています。ApplicationRuntimeExceptionHandler では、例外が発生したアクションクラスでロガーを取得して INFO レベルでログ出力し、アプリケーションエラー用のエラーページに遷移します。 ログの内容は、ApplicationRuntimeException#getMessage()で取得できる内容です。またログの出力レベルを DEBUG にすると、スローされた例外情報(スタックトレース)も出力します。
システムエラー SystemRuntimeException	SystemRuntimeException に対して、SystemRuntimeExceptionHandler を設定しています。SystemRuntimeExceptionHandler では、例外が発生したアクションクラスでロガーを取得して ERROR レベルでログ出力し、システムエラー用のエラーページに遷移します。 ログの内容は、SystemRuntimeException#getMessage()とスローされた例外情報(スタックトレース)です。
上記以外のすべてのエラー Exception	Exception に対して、システムエラーと同等と判断し、SystemRuntimeExceptionHandler を設定しております。そのため上記以外の例外が発生した際の動作は、SystemRuntimeExceptionHandler と同じです。

■ struts-config.xml の global-exceptions の設定内容

```
<global-exceptions>
    <!-- ApplicationException -->
    <exception key="errors.application"
        handler="jp.co.intra_mart.framework.extension.seasar.struts.exception.ApplicationRuntimeExceptionHandler"
        type="jp.co.intra_mart.framework.extension.seasar.struts.exception.ApplicationRuntimeException"
        path="/j2ee/document/error/s2_application_error.jsp" />
    <!-- SystemException -->
    <exception key="errors.system"
        handler="jp.co.intra_mart.framework.extension.seasar.struts.exception.SystemRuntimeExceptionHandler"
        type="jp.co.intra_mart.framework.extension.seasar.struts.exception.SystemRuntimeException"
        path="/j2ee/document/error/s2_system_error.jsp" />
    <!-- Exception -->
    <exception key="errors.exception"
        handler="jp.co.intra_mart.framework.extension.seasar.struts.exception.SystemRuntimeExceptionHandler"
        type="java.lang.Exception"
        path="/j2ee/document/error/s2_system_error.jsp" />
</global-exceptions>
```

3.3.5 [参考] トランザクション制御

トランザクション制御の設定は customizer.dicon で行っています。intra-mart 標準のトランザクション制御の設定では、アクション、ロジック、サービスのメソッドに TxAttributeCustomizer が設定されております。この設定により、メソッドの開始前にトランザクションが開始されていなければ開始され、実行メソッドが正常に終了した場合はコミットされ、例外がスローされた場合にはロールバックされます。

3.3.5.1 UserTransaction#setRollbackOnly()を利用したロールバック

トランザクションをロールバックしたいが例外をスローしたくないというケースがよくがあります。このような場合、UserTransaction#setRollbackOnly()を利用して、例外をスローせずトランザクションをロールバックさせることができます。具体的な実装例は以下の通りです。

```
public class SimpletodoAction {

  @Resource
  @ActionForm
  public SimpletodoForm simpletodoForm;

  @Resource
  protected UserTransaction userTransaction;

  @Execute(validator = true)
  public String add() {
        SimpleTodo entity = new SimpleTodo();
        BeanUtil.copyProperties(simpletodoForm, entity);
        try {
              simpleTodoService.insert(entity);
        } catch (Exception e) {
              // トランザクションをロールバック
              try {
                    userTransaction.setRollbackOnly();
              } catch (Exception e) {
                    throw new SystemRuntimeException(e);
              }
        }

        return "/sastruts/sample/simpletodo/";
  }
}
```

3.3.5.2 UserTransactionオブジェクトを使用したトランザクションの完全手動制御

トランザクションの開始、コミット、ロールバックは、UserTransaction オブジェクトを使用することで、完全に制御することができます。下記例では、TransactionAttributeType.NEVER を指定することで実行メソッドをトランザクション対象外にした上で、UserTransaction オブジェクトを使用したトランザクション制御を行っています。

```
public class SimpletodoAction {

  @Resource
  @ActionForm
  public SimpletodoForm simpletodoForm;

  @Resource
  protected UserTransaction userTransaction;

  @Execute(validator = true)
  @TransactionAttribute(TransactionAttributeType.NEVER)
  public String add() {
      SimpleTodo entity = new SimpleTodo();
      BeanUtil.copyProperties(simpletodoForm, entity);

      // トランザクション開始
      userTransaction.begin();
      try {
          simpleTodoService.insert(entity);
      } catch (Exception e) {
          // finally 句でトランザクションをロールバックさせるため、ロールバックを予約しておく
          userTransaction.setRollbackOnly();
      } finally {
          if (userTransaction.getStatus() == Status.STATUS_ACTIVE) {
              // コミット
              userTransaction.commit();
              return "/sastruts/sample/simpletodo/";
          } else {
              // ロールバック
              userTransaction.rollback();
              return "/sastruts/sample/error.jsp";
          }
      }
  }
}
```

3.3.6 [参考] JSPカスタムタグの属性でオブジェクトを渡す場合の留意点

JSP カスタムタグの属性値に対しては、文字列、数値、日付、論理値等の場合は通常、EL 式で設定することができます。しかし、上記以外の値を属性値に渡す場合があります。たとえば、intra-mart で提供している <imarttag:imartListHeader>タグの headerData 属性には、List<ListHeaderObject>のインスタンスを設定します。JSP で List<ListHeaderObject>のインスタンスを EL 式で取得した場合、BeanWrapper でラップされます。そのため、JSP で<imarttag:imartListHeader>の headerData 属性値には EL 式で設定するには、下記例のように、Action クラスでリクエストスコープに格納してください。

FooAction.java

```
<-- 中略  -->

    @Resource
    protected HttpServletRequest request;

<-- 中略 -->

    request.setAttribute("listHeaders", listHeaders);
```

index.jsp

```
<imarttag:imartListHeader
    headerData="${listHeaders}"
    sortKey="ソートキーとなる文字列" />
```

intra-mart WebPlatform/AppFramework Ver.7.2
SAStruts+S2JDBC プログラミングガイド

2012/08/03 第2版

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