Seasar Conference
2006 Autumn

S2Struts入門

2006.11.12

S2Struts コミッタ

永島　克彦

自己紹介

- 名前:永島　克彦
- ブログ:http://d.hatena.ne.jp/kanag/
- 所属:(株)広島情報シンフォニー
- メール:
  katsuhiko.nagashima@gmail.com

アジェンダ

- S2Struts概要
- S2とStrutsの連動
- 無設定～省設定
- POJO化
- 拡張TagLib
- HOT deploy
- まとめ

S2Strutsとは

- Strutsのノウハウをそのまま活用してS2と簡単に連動

さらに

- 無設定～省設定・・・記述量を減らす
- POJO化・・・テストしやすく
- 拡張TagLib・・・より便利に
- HOT deploy・・・テンポのよい開発

必要な機能のみ利用できる

S2とStruts連動の特徴

- ActionクラスをS2Containerで管理
  - DI x AOP
- 既存Actionクラスに変更なく適用可能
  - ただし、設定ファイルの変更/追加は必要
    - web.xmlの変更
    - struts-config.xmlの変更
    - Actionの登録（diconファイルの追加）

連動の設定1

- S2ContainerFilter、S2StrutsFilterの追加
- S2ContainerServletの追加
  - ActionServletは変更する必要なし。ただし、先にS2ContainerServletを初期化すること

```
<servlet>
  <servlet-name>s2container</servlet-name>
  <servlet-class>....S2ContainerServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>....ActionServlet</servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>
```

連動の設定2

- S2RequestProcessorへの変更

```
<controller
  processorClass="org.apache.struts.action.RequestProcessor"/>
    ↓
<controller
  processorClass="org.seasar.struts.processor.S2RequestProcessor"/>
```

Actionの登録方法1

diconファイルの作成1

- AutoRegisterを利用してカンタンに登録

```
<component class="....FileSystemComponentAutoRegister">
        :
  <initMethod name="addClassPattern">
    <arg>"example.app.web"</arg>
    <arg>".*Action"</arg>
  </initMethod>
</component>
```

**Actionクラスを
格納しているパッケージと
自動登録したいクラスパターンを
指定すればOK**

diconファイルの作成2

- SMART deployでもっとカンタンに登録
  - ただし、推奨パッケージ構成に合わせる必要がある

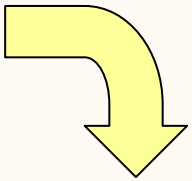
```
<component class="....NamingConventionImpl">
  <initMethod name="addRootPackageName">
    <arg>"example.app"</arg>
  </initMethod>
</component>
```

アプリケーションの
ルートパッケージを
指定するのみでOK

ActionでのAOP

- 既存のActionクラスに修正なく適用可能

```
<component class="....AspectAutoRegister">
  <property name="interceptor">traceInterceptor</property>
  <property name="pointcut">"execute"</property>
  <initMethod name="addClassPattern">
    <arg>"example.app.web"</arg>
    <arg>".*Action"</arg>
  </initMethod>
</component>
```



トレースログ出力内容

```
DEBUG ... [...] BEGIN ...Action#execute(ActionConfig[...)
      :
DEBUG ... [...] END ...Action#execute(ActionConfig[...) : ForwardConfig[...]
```

ActionでのDI

- 既存のActionクラスを修正する必要あり
  - プロパティの追加
    - ActionクラスはRequest単位で生成するため、プロパティを持っても問題なし
- 開発しやすくなる
  - ロジック、サービスクラスをMockにできる
    - テストしやすくなる
    - はやめにJSPの動作を確認できる
  - S2DaoなどのS2ファミリーが使いやすくなる

S2とStrutsの連動のポイント

開発中又は開発済みのアプリケーション

- AOPのみならActionクラスの修正なく適用可能
  - 今すぐに試すことができる

これから開発のアプリケーション

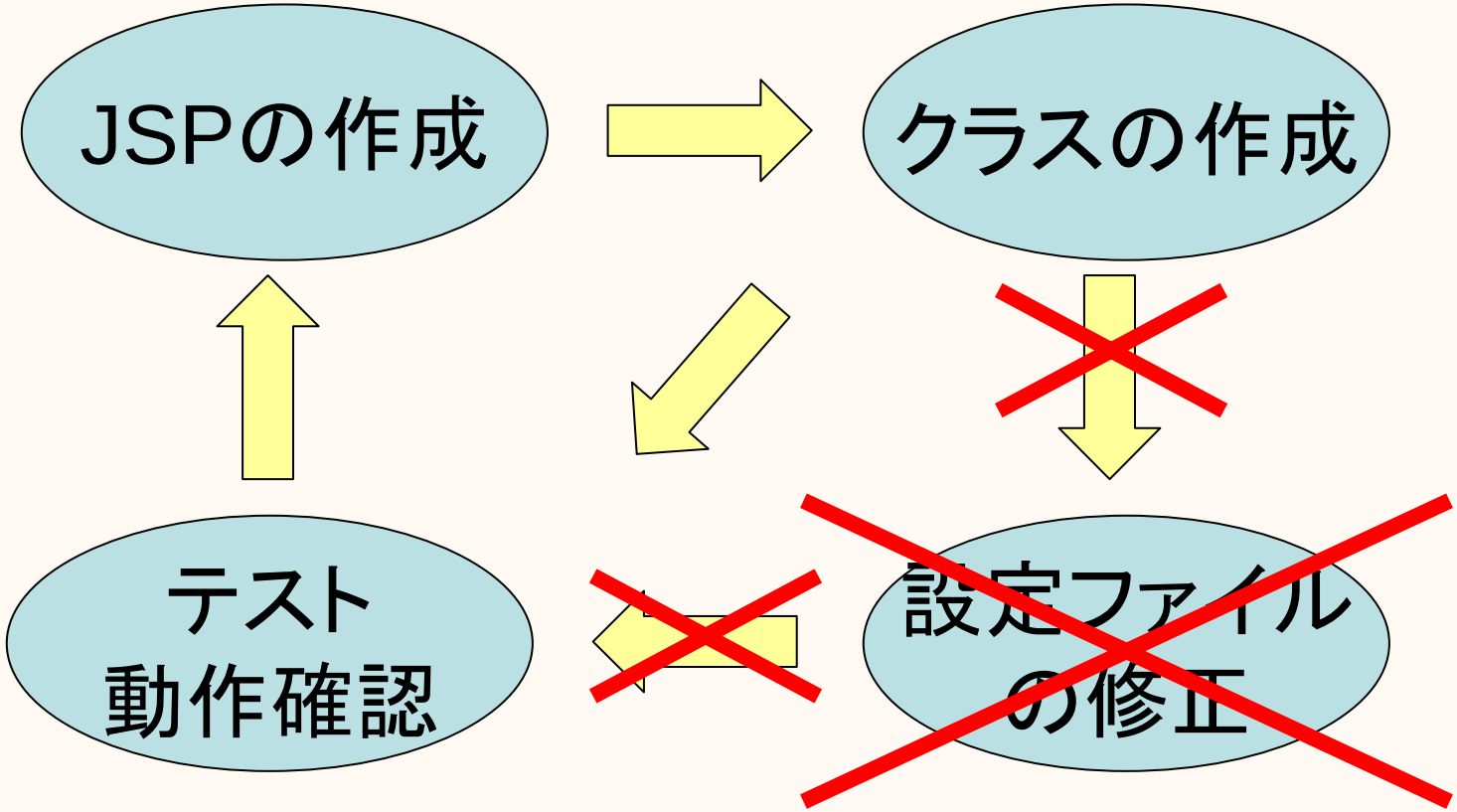
- Strutsを使って、S2DaoなどのS2ファミリーも使うなら利用したほうが楽になる

無設定～省設定の特徴

- 規約に従うことで無設定で動作
  - Actionクラスを作るのみでOK
- 規約に従えない場合はカスタマイズ可能
  - アノテーションによるカスタマイズ
    - 定数アノテーション
    - Tigerアノテーション
  - 従来どおりstruts-config.xmlによる設定も可能
    - struts-config.xmlの内容を優先

無設定～省設定

- ストレスの少ない開発



無設定～省設定の設定

- struts-config.xmlの変更
  - AutoStrutsConfigRegisterPlugInの追加
    - 無設定情報となるプロパティも設定

```
<plug-in className="....ValidatorPlugIn">
    :
</plug-in>
<plug-in className="....AutoStrutsConfigRegisterPlugIn">
    :
</plug-in>
```

ValidatorPlugInの次に
AutoStrutsConfigRegisterPlugInを追加

無設定

- 規約に従い自動的に設定

```
package example.app.form;

public class AddFrom
    extends ValidatorForm {
      :
}
```

```
package example.app.web;

public class AddAction
    extends Action {
  public ActionForward execute(...) {
            :
  }
}
```

```
<form-bean
  name=“addForm”
  type=“example.app.form.AddForm”
/>
    :
<action
  path="/add"
  type="example.app.web.AddAction"
  name="addForm"
  scope="request"
  validate="true">
  <forward name="success"
    path="/add.jsp" />
</action>
```

form-beanの規約

- form-bean nameの規約

```
<form-bean
  name="addForm"
  type=
    "example.app.form.AddForm"
/>
```

コンポート名を設定

クラス名：AddForm
↓
コンポーネント名：addForm
↓
form-bean name：addForm

actionの規約1

- action pathの規約

```
<action
  path="/add"
  type=
    "example.app.web.AddAction"
  name="addForm"
  scope="request"
  validate="true">
  <forward name="success"
      path="/add.jsp" />
</action>
```

“/” + “Action”を
　　消したコンポート名を設定

クラス名：AddAction
↓
コンポーネント名：addAction
↓
action path：/add

actionの規約2

- action nameの規約

```
<action
  path="/add"
  type=
    "example.app.web.AddAction"
  name="addForm"
  scope="request"
  validate="true">
  <forward name="success"
      path="/add.jsp" />
</action>
```

“Action”を消したコンポート名
　　＋　“Form” or “Dto”を設定

クラス名：AddAction
↓
コンポーネント名：addAction
↓
action name：addForm

actionの規約3

- forwardの規約

```
<action
  path="/add"
  type=
    "example.app.web.AddAction"
  name="addForm"
  scope="request"
  validate="true">
  <forward name="success"
      path="/add.jsp" />
</action>
```

パッケージとクラス名からViewを検索しあった場合は“success”のforwardとして設定

クラス名：example.app.web.AddAction
docRoot　　：/
ViewExtension　：jsp
の場合の検索順

1. /example/app/web/add.jsp
2. /app/web/add.jsp
3. /web/add.jsp
4. /add.jsp

actionの規約4

- その他はデフォルト値を割り当てる

```
<action
  path="/add"
  type=
    "example.app.web.AddAction"
  name="addForm"
  scope="request"
  validate="true">
  <forward name="success"
      path="/add.html" />
</action>
```

scope	request
validate	true
input	null
parameter	null
attribute	null
forward	null
include	null
prefix	null
suffix	null
unknown	null
roles	null
cancellable	false

バリデーション

- バリデーションの設定もアノテーションでかける
  - クラスとバリデーション設定の記述が近く管理しやすい（わかりやすくなる）
- 独自のバリデーションをアノテーションとして作成することもできる

```
public class AddForm
    extends ValidatorForm {
@Required
@EmailType
public void setValue(String valu
     :
 }
}
```

```
public class AddForm
    extends ValidatorForm {
 public static final String value_VALIDATOR_0 = "required";
 public static final String value_VALIDATOR_1 = "email";
 public void setValue(String value) {
     :
 }
}
```

無設定～省設定のポイント

省設定でシンプルに

- 基本的な設定は無設定にまかせ、一部の特殊なものをアノテーションで設定
- アノテーションで設定する項目を限定する
  - ちょっとした設定で思わぬ時間をとられることを防ぐ
  - 限定の例
    - form-beanは無設定で行う
    - action-mappingは、nameとscopeのみアノテーションで設定する etc

ActionをPOJO化したときの特徴

- テストしやすくなる
- 特定のAPIに依存しなくなる
  - import文からStruts関連のクラスがなくなる
- ページを中心とした開発
  - 拡張TagLibを利用することで可能となる
    - Initタグによる画面初期化
    - MethodBindingによるAction処理

POJO Action例

```
@StrutsAction(name="calcForm")
public class AddAction {

  @StrutsActionForward(path="/calcResult.jsp"
  public static final String SUCCESS = "success";

  public void setCalcForm(CalcForm calcForm) { ...}
  public CalcForm getCalcForm() { ... }

  @ExportToSession
  public List getCalcHistory() { ... }

  public void initialize() { ... }

  public String doCalc() { ... }

}
```

POJO Action規則1

- Actionメソッド（引数なし、戻り値Stringのメソッド）
  - Forward名を返却する
  - 1つのActionメソッドのみを定義している場合
    - 通常Actionと同様の方法による実行
    - MethodBindingによる実行
  - 複数のActionメソッドを定義している場合
    - DispatchActionと同様の方法による実行
    - MethodBindingによる実行

```
<s2struts:submit action="#{addAction.doCalc}" />
```

MethodBindingでは
実行するActionメソッドを
コンポーネント名.メソッド名
で指定

POJO Action規則2

- Initializeメソッド（引数なし、戻り値voidのメソッド）
  - 画面の初期化を行う
  - メソッド名は自由
  - Initタグで呼び出されるメソッド

```
<s2struts:init action="#{addAction.initialize}" />
```

Initタグでは
Initializeメソッドを
コンポーネント名.メソッド名
で指定

POJO Action規則3

- FormBeanのバインディング
  - FormBean名のプロパティを定義
  - セッターメソッドでViewから受け取れる
  - ゲッターメソッドでViewへ渡す
  - スコープはActionMappingに従う

POJO Action規則4

- その他のオブジェクトのバインディング
  - セッターメソッドでViewから受け取れる
    - 受け取る優先順位
      - HttpServletRequest#getParameter(プロパティ名)
      - HttpServletRequest#getAttribute(プロパティ名)
      - HttpSession#getAttribute(プロパティ名)
  - ゲッターメソッドでViewへ渡す
    - デフォルトはRequestスコープに値を設定
    - Sessionスコープに値を設定する場合はアノテーションで指定する

POJO Actionの疑問

- interfaceは必要なの？
  - S2Struts1.3からinterface不要
  - ただinterfaceを定義すると設定と実装が分離され明確になる
- InitializeメソッドとActionメソッドは別々のActionクラスで定義しないといけないの？
  - 1つのActionクラスとして定義可能

POJO化のポイント

トレードオフを考慮する

- Strutsから距離をおく
  - 移行しやすいかも
  - でも、Strutsの知識は必須
- POJO化するだけの価値はあるか
  - テストしやすくなる
  - 拡張TagLibを使うことで開発しやくなる
    - でも、新しく覚えることが増える

拡張TagLibの特徴

- Strutsのタグをより便利に
  - indexIdの追加
  - チェックがない場合でも値をセットするcheckboxタグ
  - バリデーションを行わなくするcancel属性の追加
- 無設定の支援
- ページを中心とした開発のための機能

無設定の支援

Pageタグ

- エラー発生時、自画面を再表示する
  - action inputを指定しなくてよい

cancel属性

- バリデーション有無をタグで指定
  - action validateは常にtrue（デフォルト値）で問題なし

ページ中心とした開発のための機能

Initタグ

- 画面の初期化処理を行う
- ページの初期化についてはそのページで指定

MethodBinding

- POJO Actionのメソッドを直接指定する
  - DispatchActionなどの代替え手段
  - cancel属性と併用することによりバリデーション有無を制御

拡張TagLibのポイント

POJO化しない場合

- Strutsタグの拡張部分のみを利用
  - 違和感なく便利に利用できる
- Initタグは利用しないほうがよいかも
  - 利用したらPOJO化したくなる

POJO化する場合

- 積極的に利用したほうが便利
  - Pageタグを利用すればInitタグもほしくなる
  - Initタグを利用すればMethodBindingしたくなる

HOT deployの特徴

- テンポのよい開発
  - クラスの追加／変更がすぐに反映
  - 設定ファイルの変更もすぐに反映
    - struts-config.xml
    - validation.xml
    - application.properties
- ストレスの少ない開発
  - アプリケーションサーバーを起動したまま開発

HOT deployの設定

- web.xmlの変更
  - HotdeployFilterの追加

```
<filter>
  <filter-name>hotdeployfilter</filter-name>
  <filter-class>....HotdeployFilter</filter-class>
</filter>
  :
<filter-mapping>
  <filter-name>hotdeployfilter</filte
  <url-pattern>/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

REQUESTとFORWARDに対してフィルタをかけるため Servlet2.4が必要

HOT deployの設定

- struts-config.xmlの変更
  - HotdeployPlugInの追加

```
<plug-in className="....ValidatorPlugIn">
    :
</plug-in>
<plug-in className="....HotDeployPlugIn">
    :
</plug-in>
```

一番最後に
HotDeployPlugInを追加する

HOT deployの仕組み

- 設定ファイル
  - Requestのたびに毎回読み込む
- 無設定～省設定の場合
  - ActionConfigの求め方
    - RequestのたびにActionConfigを生成する
    - Requestパス→コンポーネント名→クラス→ActionConfig
  - FormBeanConfigの求め方
    - RequestのたびにFormBeanConfigを生成す
    - ActionConfig name→コンポーネント名→クラス →FormBeanConfig

HOT deployの制限

- 無設定～省設定の場合の制限
  - action pathは規約による設定とする
    - pathからActionクラスを求めるため
  - form-bean nameは規約による設定とする
    - nameからFormBeanクラスを求めるため

HOT deployの注意点

- 対応していない設定ファイルがある
  - web.xml、tiles-defs.xmlについては未対応
- 開発のときのみ利用
  - Requestのたびに設定ファイルを読み込むため性能が悪くなる
  - 本番運用時にはCOOL deployに切り替える
    - env.txtの内容を”ut”以外に変更

まとめ

- 必要な機能のみを利用
  - まずはS2とStrutsの連動とHOT deployを試してほしい
- 開発効率のアップ
  - 無設定～省設定により設定ファイルの管理を不要にでき、HOT deployにより動作確認がすぐにできるようになる
- POJO化による利点
  - ActionをPOJO化することによりテストがしやすくなり、拡張TagLibも一緒に使うことによりページ中心の開発が可能となる

ご清聴
ありがとうございました