



S2JSF プレゼンテーション層の革命

ひがやすを
電通国際情報サービス



S2JSFにむいているのはこんな人

- **Question**

- JSPを使っていてブラウザで直接レイアウトを確認できないことに苛立ちを覚えたことはありませんか
- HTMLをJSPに変換するという単調だけど間違えやすい作業にうんざりしたことはありませんか
- JSPってデザイナーと連携しづらいよなあって思ったことはありませんか
- JSFってStrutsより便利そうだけど乗り換えるほどじゃないよなあって思ったことはありませんか

- **Answer**

- そんなあなたにS2JSFをお勧めします



JSFのメリット

- リクエストの内容を受け取るオブジェクト(StrutsのActionForm)、ロジックを起動するオブジェクト(StrutsのAction)をPOJOにすることができる
 - Webに依存しないのでテストがしやすくなる
- ブラウザで入出力する値がサーバサイドのJavaBeansのプロパティの値と連動する
 - 不正な値が入力された場合は、JavaBeansのプロパティの値は更新されず、入力された値をそのままブラウザで再表示が出来る
 - Strutsではこれが出来ないため、プロパティの型をStringにしていったんリクエストの内容を格納する必要があった
 - ActionFormからビジネスオブジェクトへの変換が不要
- イベントの処理が簡単
 - ボタンやリンクでコンポーネントのメソッド名を指定することで簡単にイベントを処理することが出来る
 - リクエストの中身を解析することは不要



- 標準ではJSPをテンプレートに使っているためJSPのデメリットをそのまま受け継いでいる
 - ブラウザで直接レイアウト・画面遷移を確認することが出来ない
 - デザイナーにJSPを直接書いてもらうことが出来ない
 - HTMLで作ったモックをJSPに変換する作業が単調で間違えやすい
 - JSPに変換後デザインに大きな変更が入るとせっかく作ったJSPを捨ててもとのHTMLを修正し、再度JSPに変換する必要がある



S2JSFの特徴

- テンプレートはHTML
 - ブラウザで直接レイアウト・画面遷移を確認出来る
 - デザイナーにHTMLを直接書いてもらうことが出来る
 - HTMLで作ったモックをJSPに変換する作業は不要でいくつか属性を追加するだけ
 - HTMLにいくつか属性を追加するだけなので後でデザインにおきな変更が入っても、その属性追加後のHTMLをデザイナーに渡すことが出来る
- HTMLを実行時にJSFのタグに変換して実行
 - イメージ
 - HTML:<input type="text" m:value="#{hoge.aaa}"/>
 - 変換後のJSFのタグ:<h:inputText value="#{hoge.aaa}"/>
 - 明示的にタグ(S2JSFが知らないタグでも)を指定することも可能:
<input type="text" m:inject="h:inputText" m:value="#{hoge.aaa}"/>
 - HTMLに幾つか属性を追加するだけでJSFアプリケーションに変身



- 特別なツールは必要なくHTMLのEditorがあれば良い
 - S2JSFのプラグインも開発中
- リクエスト、セッションの属性をアクションクラスのプロパティに自動バインディング
 - リクエスト、セッションの属性名とアクションクラスのプロパティ名をあわせておけば自動的にバインディングされる
 - リクエスト、セッション、JSFのAPIを意識することなくリクエストやセッションの属性にアクセスできる
 - JSFの場合、リクエストやセッションは直接意識する必要はないがJSFのAPIを使う必要がありテストがしにくくなる
- どのAPI(もちろんS2)にも依存しない完全なPOJOで開発することが出来る



- MyFaces + Seasar2 + NekoHTML(Nirvana版) + S2JSF
 - NekoHTML
 - HTMLをSAXのイベントに変換
 - 閉じタグ忘れなどもきちんと処理してくれる
 - inputタグに子タグがかけないなどJSF用には一部問題があるので NirvanaプロジェクトのNekoHTMLを使用
 - <https://nirvana.dev.java.net/>
 - MyFacesに依存しているわけではないが稼動確認は今のところ MyFacesのみ



S2JSFの仕組み(概念)

- HTMLのコンパイル
 - HTMLをSAXのイベントに変換
 - SAXのイベントハンドラでHTMLのタグをJSFのタグに変換
 - JSFのタグのツリーを構築
- HTMLの表示
 - HTMLを最初にアクセスされたとき、および変更されたときにコンパイルしてタグのツリーを構築
 - コンパイルされた結果はキャッシュ
 - JSFのタグのツリーをJSPをエミュレートして実行



登場人物

- テンプレートHTML
- DTO(Data Transfer Object)
 - HTMLの入出力項目と自動バインディング
- Action
 - ボタンやリンクをクリックしたときに呼び出されるメソッドを定義
 - リクエストやセッションに格納されているDTOが自動的にプロパティに設定されている
 - そのDTOを業務ロジックにわたし戻り値をプロパティに格納する
 - 戻り値はリクエストやセッションに自動的に格納される



hello.html

```
<html xmlns:m="http://www.seasar.org/maya">
<head>
<meta http-equiv="Content-Type" content="text/html;
  charset=Windows-31j" />
<title>Hello</title>
</head>
<body>
  こんにちは<span m:value="{message}">hoge</span>
</body>
</html>
```



リンクによるページ遷移

```
<a href="hello/hello.html">あいさつ  
  <span m:inject="f:param" m:name="message" m:value="World"/>  
</a>
```

f:paramの部分は論理的には下記のように解釈される

```
<f:param name="message" value="World"/>
```



```
<form>
<span m:inject="h:messages" m:globalOnly="false"
      m:showDetail="true"/>
<input type="text" m:value="#{addDto.arg1}"/> +
<input type="text" m:value="#{addDto.arg2}"/> =
<span m:value="#{addDto.result}"/>
<input type="submit" value="計算"
      m:action="#{addAction.calculate}"/>
<input type="button" value="戻る" m:action="home"
      onclick="location.href='../index.html'"
      m:immediate="true"/>
</form>
```



faces-config.xml(homeの定義)

```
<navigation-rule>  
  <navigation-case>  
    <from-outcome>home</from-outcome>  
    <to-view-id>/index.html</to-view-id>  
    <redirect/>  
  </navigation-case>  
</navigation-rule>
```



繰り返し

```
<span m:inject="s:forEach" m:items="#{forEachDtoList}"
  m:var="e">
  <tr>
  <td><span m:value="#{e.key}">111</span></td>
  <td><span m:value="#{e.name}">あああ</span></td>
  <td><a href="forEachResult.html"
m:action="forEachResult">
    結果画面へ
    <span m:inject="f:param" m:name="selectKey"
      m:value="#{e.key}"/>
    <span m:inject="f:param" m:name="selectName"
      m:value="#{e.name}"/>
  </a>
  </td>
  </tr>
</span>
```



Converterの定義(diconファイル)

```
<component name="inputDateTimeConverter"  
    class="javax.faces.convert.DateTimeConverter">  
    <property name="pattern">"yyyyMMdd"</property>  
</component>  
<component name="outputDateTimeConverter"  
    class="javax.faces.convert.DateTimeConverter">  
    <property name="pattern">"yyyy/MM/dd"</property>  
</component>
```



Converterの組み込み

```
<input type="text" m:value="#{converterDto.aaa}"  
      m:converter="#{inputDateTimeConverter}"/><br />  
<span m:value="#{converterDto.aaa}"  
      m:converter="#{outputDateTimeConverter}"/><br />
```



Validatorの定義(diconファイル)

```
<component name="userNameLengthValidator"  
  class="javax.faces.validator.LengthValidator">  
  <property name="minimum">2</property>  
</component>
```



Validatorの組み込み

```
<span m:inject="h:messages" m:globalOnly="true"/>
<input id="userName" type="text"
      m:value="#{validatorDto.userName}"
      m:required="true">
  <span m:inject="s:validator"
        m:binding="#{userNameLengthValidator}"/>
</input>
<span m:inject="h:message" m:for="userName"/>
```



checkbox

```
<input type="checkbox"  
m:value="#{checkboxDto.aaa}"/>hoge
```

checkboxをOFFにしたことがちゃんとサーバサイドに伝わる
ことがこれまでのcheckboxとの違い



select

```
<select m:value="#{selectOneMenuDto.aaa}">
  <option value="1">One</option>
  <option value="2">Two</option>
  <option value="3">Three</option>
</select>
```



select2

```
<select m:value="#{selectOneMenuDto.bbb}"  
  m:items="#{bbbSelectItems}">  
  <option value="">選択してください</option>  
  <option value="1">One</option>  
  <option value="2">Two</option>  
</select>
```



select2(SelectItem)

```
<component name="bbbSelectItems" class="java.util.ArrayList">
  <initMethod name="add" >
    <arg>
      <component class="javax.faces.model.SelectItem">
        <property name="label">"選択してください"</property>
      </component>
    </arg>
  </initMethod>
  <initMethod name="add" >
    <arg>
      <component class="javax.faces.model.SelectItem">
        <property name="value">1</property>
        <property name="label">"One"</property>
      </component>
    </arg>
  </initMethod>
  ...
</component>
```



select3

```
<select m:value="#{selectOneMenuDto.deptno}"
  m:items="#{bbblItems}"
  m:itemValue="deptno"
  m:itemLabel="dname"
  m:nullLabel="選択してください">
  <option value="">選択してください</option>
  <option value="10">ACCOUNTING</option>
</select>
```



select3(JavaBeans)

```
<component name="bbbItems" class="java.util.ArrayList">
  <initMethod name="add" >
    <arg>
      <component class="examples.jsf.entity.Department">
        <property name="deptno">10</property>
        <property name="dname">"ACCOUNTING"</property>
      </component>
    </arg>
  </initMethod>
  ...
</component>
```



ご参考

- ひがやすを
 - <http://d.hatena.ne.jp/higayasuo/>
 - higa@isid.co.jp
- Seasar.org
 - <http://www.seasar.org>
- Seasar Project
 - <http://sourceforge.jp/projects/seasar>