Seasar Conference
2008 Autumn

Teeda再考 ～使い方から拡張方法まで～

株式会社Abby　代表取締役
米林　正明

- 名前

  米林　正明 (id:yone098)

- 所属

  株式会社Abby

  社員募集中（ちょっとでも興味あれば是非）

- Blog

  よねのはてな

  http://d.hatena.ne.jp/yone098/

- コミッタ活動
  - Teedaコミッタ
  - S2JSFプロジェクトリーダー
- Teedaの執筆活動
  - JavaExpert#01　特集2
  - JavaExpert#02　特集1
  - 現在Teeda本執筆中

- Teedaとは
  - JSFとは
- Teedaの特徴
- TeedaAjax
- 再考と嵌りどころ
- Teedaの拡張
- Teedaの今後

～初めに～

本日のセッションは真面目です。
悪しき習慣である内輪ウケ狙いは一切しません。
Teeda再考とTeeda最高の駄洒落も無いです。
皆さまにお伝えしたいことが多いため(173ページ)
全般的に少し駆け足になることをご了承ください。

Teedaとは

ていーだ

沖縄の言葉で「太陽」

Java
Webアプリケーション フレームワーク

HTML
テンプレート

コンセプト

JSF meets DI x AOP

JSFとは

Java Server Faces

JSFとは
あくまで仕様

Sun Microsystemsが提案し JSR-127で仕様を策定したもの 仕様だけあって実装はない

JSF実装

MyFaces

※2008/08/30 Core1.2.4　Release

Mojarra(the JSF RI)

※2008/07/02にRelease

日本に
JSF実装は無い

まだ私が若かった頃

ないのなら
作ってみよう
日本初

米林一茶

Teedaは
日本初JSF実装

Teedaの構成

- Teeda Core
- Teeda Extention
- Teeda Ajax

Teeda Core

- JSF1.1実装
- JSFコンポーネント管理にSeasar2を利用

Teeda Extention

- Teeda Coreをベースに
  HTMLと規約に基づいた
  拡張を提供

Teeda Ajax

- Ajaxに特化したライブラリで独立して利用することも可能

まとめ

TeedaはJSF実装であり DIとAOP機能にSeasarを使ったフレームワーク

Teedaの特徴

1. HTMLテンプレート
2. PRGパターン
3. レスコンフィグレーション
4. SMART Deploy
5. ライフサイクル
6. 多機能

Teedaの特徴

HTMLテンプレート

Page駆動開発

HTMLを中心に

開発するスタイル

やり方

HTMLに対応する

Pageクラス作成

Eclipseプラグインである
Doltengを使えば
HTML右クリック一発で
Pageクラス作成可能

急がば回れ

Teedaを理解するために
あえてDoltengは使わない

重要

HTMLのid
Pageクラスの
プロパティ

【Pageクラス作成】

POJOを作るだけ

【Pageクラス命名規則】

HTML名の先頭を大文字にして

Pageというサフィックスをつける

test.html→TestPage.java

【Pageクラス作成】

単純明快

【test.html】

```
<body>
<form id="testForm">
 <input type="text" id="arg1" />
 <input type="text" id="arg2" />
 <input type="submit" id="doExec" />
</form>
</body>
```

【TestPage.java】

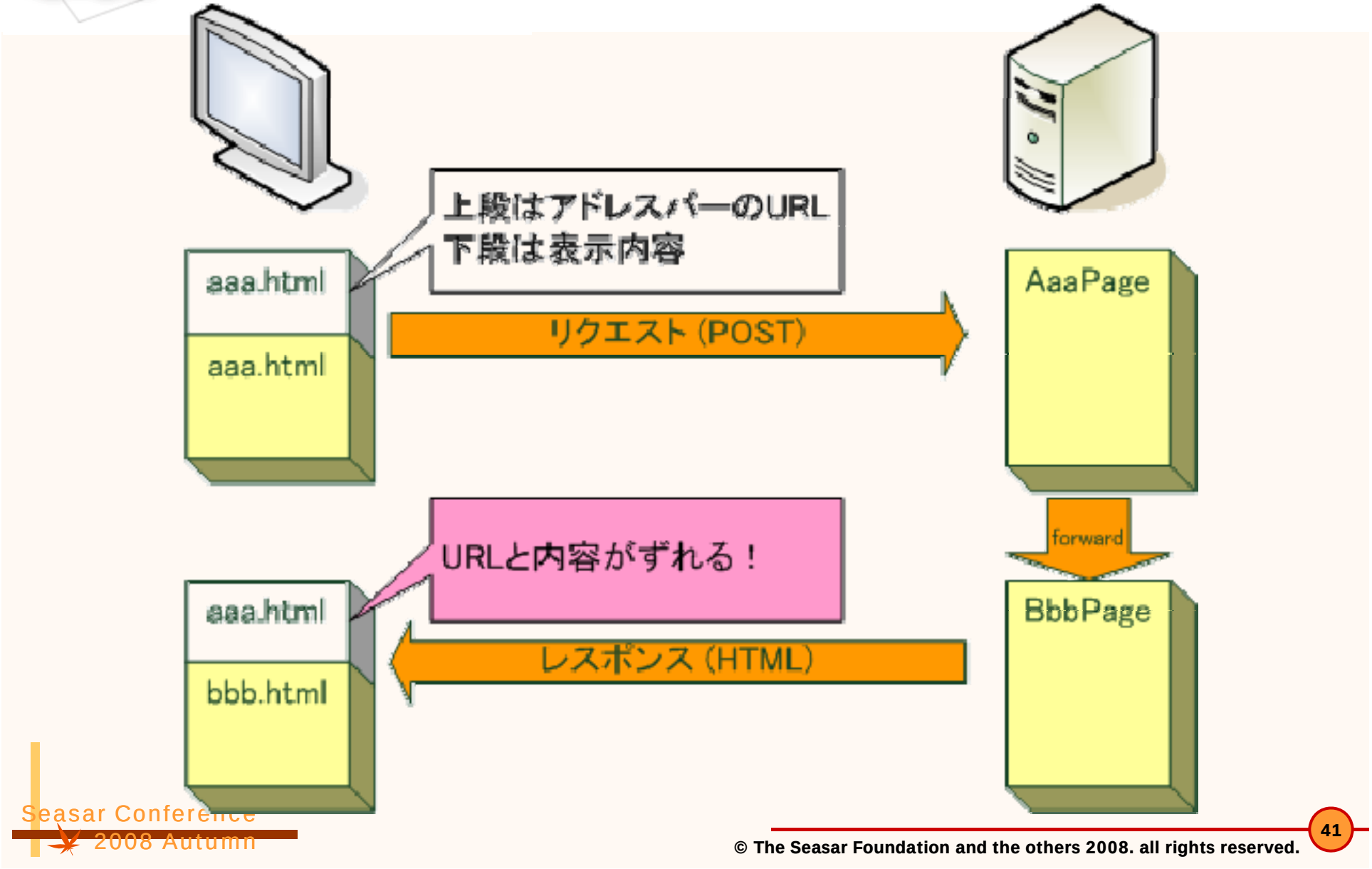
```
public class TestPage {
  public String arg1;
  public String arg2;
  public Class doExec() {
  }
}
```

入力項目や
ボタンのアクションは
HTMLのidにより
Pageクラスと関連付け

Teedaの特徴

PRGパターン

JSF forwardベースアーキテクチャ

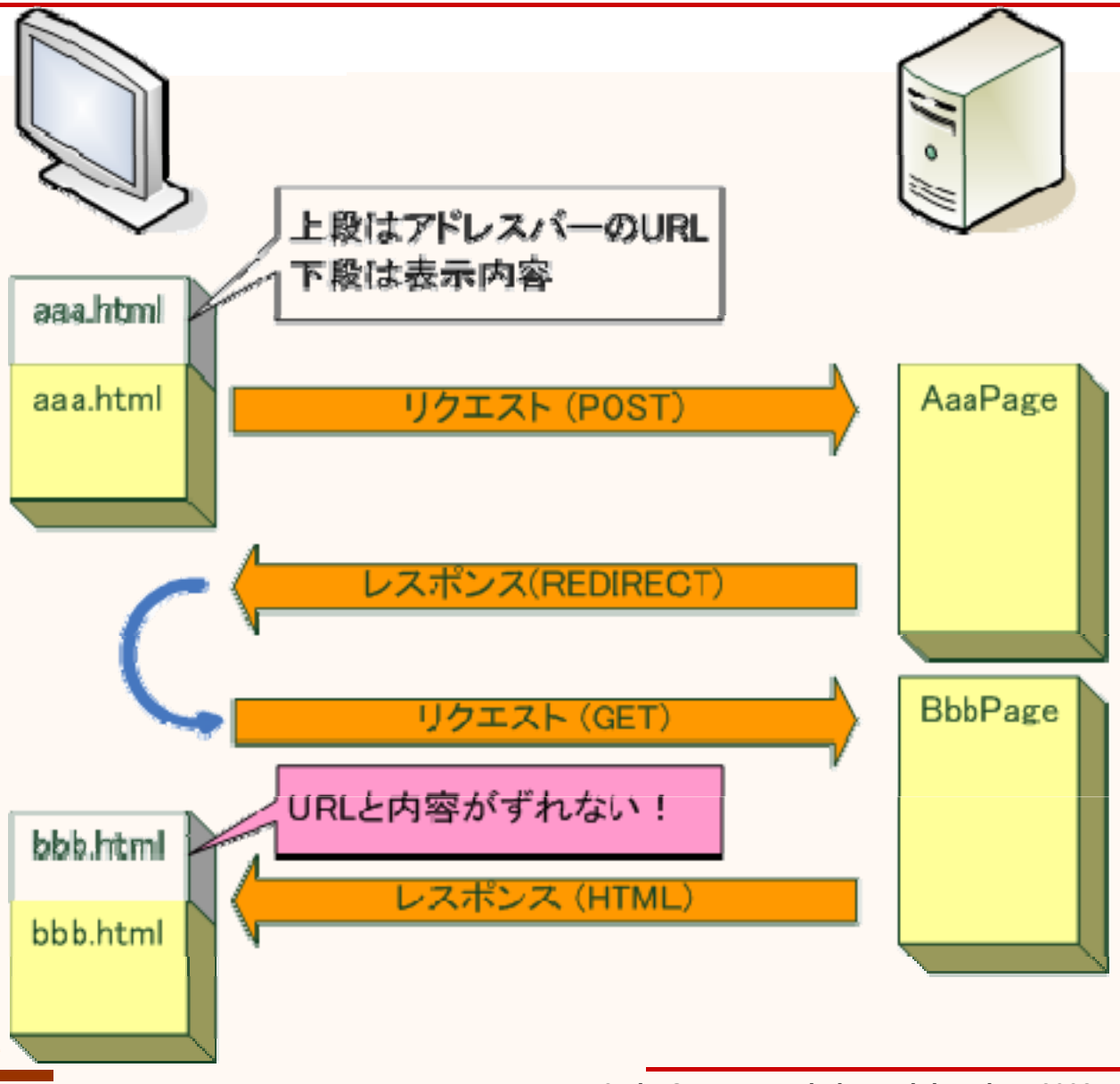


PRGパターンとは？

POST

REDIRECT

GET



【PRGパターンのメリット】

- ブラウザのリロードによる予期せぬ更新を防げる
- ブラウザの戻るボタン対応
- 表示されるURLがずれない

Teedaの特徴
レスコンフィグレーション

【レスコンフィグレーション】

Struts：struts-config.xml

JSF：faces-config.xml

Teeda：

遷移先の設定を書かない

リンクでのページ遷移

一覧

list.htmlへ

メソッドによるページ遷移

return ListPage.class

list.htmlへ

コンポーネント登録
ルートパッケージ登録のみ

```
<initMethod
        name="addRootPackageName">
    <arg>"jp.co.abby.demo"</arg>
</initMethod>
```

Teedaの特徴

SMART Deploy

Seasar2.4の機能

Tomcat等のWebコンテナを起動したまま画面の追加変更を行い即座に確認可能

便利だけど
はまると
難しい＞＜

Teedaでの
はまりパターン

COOL Deploy時 Pageクラスの メソッドが呼ばれない

Pageクラスを継承した際親クラスが抽象クラスじゃないと正しく登録されない

public abstract XxxPage

独自クラスが SMART Deploy 対象にならない

自分でdiconファイルに登録しているか？
登録した場合、SMART Deploy対象にならない

メモリリーク

><

以下の記述がないか？

```
S2Container container;
container =
    S2ContainerFactory.create(DICON_PATH);
container.init();
```

リクエストの度にコンテナ生成することはやってはいけません

どうしても特殊な場合

```
public String previousViewId;
public S2Container container;
public NamingConvention namingConvention;
String pageName = namingConvention
        .fromPathToPageName(previousViewId);
Object previousPage =
          container.getComponent(pageName);
```

S2ContanerをDIしてもらう

別アプローチ

はまるなら
外してみよう
新機能

米林一茶

Teedaの特徴

ライフサイクル

JSFに密接

JSFは難しいから一旦 JSFは頭からはずします

【6つのフェーズ】

1. ビュー復元
2. リクエスト適用
3. Validator/Converter適用
4. Pageクラス更新
5. PageクラスのAction実行
6. 描画

駆け足

ビュー復元（RestoreView）フェーズ

- Pageの状態復元
- ポストバック機能の初期化
- スコープ管理機能
- 中間モデルの復元

リクエスト適用（ApplyRequestValues）フェーズ

- UIコンポーネントに
  リクエスト内容を適用

Validator/Converter適用(ProcessValidations)フェーズ

- Converterの起動
- Validatorの起動
- UIコンポーネント内での値の持ち替え

Page更新(UpdateModelValues)フェーズ

・Pageクラスのプロパティ更新

PageクラスAction実行（InvokeApplications）フェーズ

- Pageクラスのdoメソッドの実行
- 戻り値による遷移先の自動判別
- Pageクラスのプロパティを保存

描画(RenderResponse)フェーズ

- initializeメソッド(初回のみ)
- prerenderメソッド(毎回)
- レスポンス描画

フェーズとか難しいので Teedaライフサイクルのまとめ

1. HTMLにアクセス
2. Teedaの処理開始
3. リクエスト値の変換と入力チェック
4. Pageクラスにリクエスト値を反映
5. Pageクラスのメソッドを順に実行
   initialize->doXxx->prerender

Teedaの特徴

多機能

ボタン関連

```
<input type="button" id="doHoge" />
```

doHogeメソッドが呼ばれる

```
<input type="button" id="doOnceHoge" />
```

doOnceHogeに変えると

ダブルサブミット防止

バリデーションのボタン指定

```
<input type="button" id="doHoge" />
```

```
@Required(target = "doHoge")
private String arg1;
```

targetで指定したidのボタン押下時のみバリデーションが動作

コンバータのボタン指定

```
<input type="button" id="doHoge" />
```

```
@DateTimeConverter(pattern="yyyy/MM/dd",
    target="doHoge")
private Date arg2;
```

targetで指定したidのボタン押下時のみコンバータが動作

TeedaAjax

【TeedaAjax】

簡単にPageクラスのメソッドをAjaxで呼び出せる

出来た経緯

Ajaxが流行った数年前

Teedaでも呼ばせてみよう非同期で

米林一茶

【AjaxPage.java】

```
package teeda.web.test;

public class AjaxPage {
  // メソッド名はajaxから始めること
  public String ajaxHello() {
    return “Hello Ajax!”;
  }
```


```
// 関数名にPageクラスとメソッド名を指定
function ajax_ajaxPage_ajaxHello(response) {
  alert(response); // HelloAjax!が表示される
}
function test() {
  Kumu.Ajax.executeTeedaAjax(
    ajax_ajaxPage_ajaxHello, []); //callback関数設定
}
<input type="button" onclick="test();" />
```

JSの関数名により呼び出すコンポーネント解決

サブアプリケーション名＋”_”
Pageクラス名＋”_”
メソッド名

再考と嵌りどころ

そもそも DIは何が 嬉しいのか？

出来た経緯

ロッドジョンソンがその昔

使いたい
そんなあなたに
インジェクション

ロッドー茶

DIの醍醐味

```
public class AaaPage {
  public HttpSession sessionScope;
  public HttpServletRequest requestScope;
  public LoginDao loginDao;
  public LoginLogic loginLogic;
}
```

newせず注入してもらう

TeedaのDI

```
public class AaaPage {
  public Cookie cookie;
  public Map header;
}
```

CookieやHTTPヘッダ情報も
Pageクラスに定義するだけ

スコープ

従来のWebアプリケーションスコープ

- Requestスコープ
- Sessionスコープ
- Applicationスコープ

開発者が

setAttributeしたり

removeAttributeしたり

ちょっと不便

Teeda独自スコープ

- デフォルトスコープ
- Pageスコープ
- Redirectスコープ
- SubApplicationスコープ

駆け足

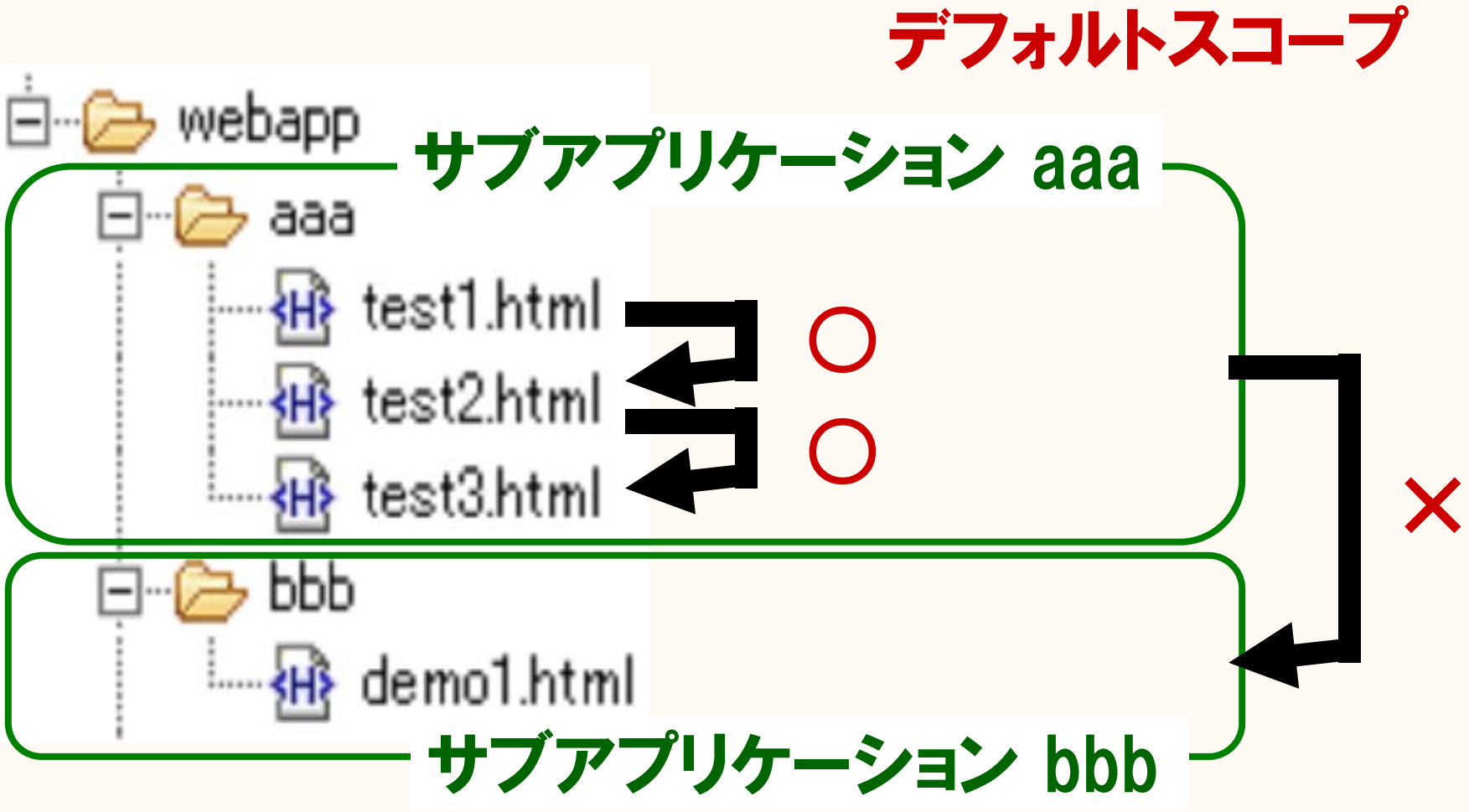
デフォルトスコープ

Test1PageとTest2Pageで
同一プロパティを自動で受け継ぐ

【条件】

・同一サブアプリケーションであること

・inputタグの項目であること



Pageスコープ

同一Pageの間だけ保持

別ページに遷移すると消去

```
package teeda.web.aaa;

public class AaaPage {

  @PageScope

  private String message;
```

Pageスコープ

Redirectスコープ

Redirect1回されるまで保持

Redirect後の画面リロードで消去

```
package teeda.web.aaa;

public class AaaPage {

  @RedirectScope

  private String message;
```

Redirectスコープ

SubApplicationスコープ

Teedaのサブアプリケーション単位

ユースケース単位で引き継がれる

HTMLにinput項目が無くても引き継ぐ

```
package teeda.web.aaa;

public class AaaPage {

  @SubapplicationScope

  private String message;
```

SubApplicationスコープ

コンポーネントの
スコープを
Sessionにしたい


```
// classにアノテーションを指定するだけ
@Component(instance = InstanceType.SESSION)
public class UserDto implements Serializable {
  // 省略
}
```

開発時に

動作しない時

困ったら

疑うポイント

- Pageクラスが存在するか？
- FormのidはxxxFormで終わっているか？
- エラーが起きていないか？

  →バリデーション・コンバータ

- PageクラスがS2に登録されている？
- Formがネストしていないか？

調べても分からない場合

- BLOGで現象を書く
- メーリングリストで聞く
- 2chに書いてみる

それでも

解決しない場合

><

1.山手線に乗る

2.恵比寿駅で降りる

3.弊社に面接しに来る

4.弊社に入社→解決

Teedaの拡張

鉄板で
今日の目玉

- 独自バリデーション・コンバータ作成
- フェーズへの処理差し込み方法
- TeedaAjax拡張(prototype.js/jquery)
- SMART Deployを使わない
- Ovalでバリデーション
- カスタムコンポーネント作成（Jython)

駆け足

m(_ _)m

独自バリデーション

JSFのバリデーターを拡張して

作成可能です。

ハマりどころも少ない為

今日は割愛します。

独自コンバータ

JSFのコンバータを拡張して

作成可能です。

ハマりどころも少ない為

今日は割愛します。

フェーズへの処理差し込み方法

JSFのフェーズって難しい＞＜

でもTeedaで開発すると少し

は理解しないと困る場面が…

その為にPhaseListenerを作成

JSFのPhaseListener

```
public class MyPhaseListener implements PhaseListener {
 // フェーズ後に実行されます
 public void afterPhase(PhaseEvent event) {
  System.out.println("# AfterPhase[" + event.getPhaseId() + "]");
 }
 // フェーズ前に実行されます
 public void beforePhase(PhaseEvent event) {
  System.out.println("# BeforePhase[" + event.getPhaseId()
                                      + "]");
 }
 public PhaseId getPhaseId() {
  //return PhaseId.RENDER_RESPONSE;
  return PhaseId.ANY_PHASE; // 全フェーズ
 }
}
```

【faces-config.xml】

```
<lifecycle>
   <phase-listener>
      teeda.MyPhaseListener
   </phase-listener>
</lifecycle>
```

Html表示時に出力

```
# BeforePhase[RESTORE_VIEW:1]
DEBUG 2008-09-06 03:22:24,453 [http-8080-1] クラス
(examples.teeda.web.sample.Sample1Page[sample_sample1Page])のコンポーネン
ト定義を登録します
# AfterPhase[RESTORE_VIEW:1]
# BeforePhase[APPLY_REQUEST_VALUES:2]
# AfterPhase[APPLY_REQUEST_VALUES:2]
# BeforePhase[PROCESS_VALIDATIONS:3]
# AfterPhase[PROCESS_VALIDATIONS:3]
# BeforePhase[UPDATE_MODEL_VALUES:4]
# AfterPhase[UPDATE_MODEL_VALUES:4]
# BeforePhase[INVOKE_APPLICATIONS:5]
# doExecActionが呼ばれたよ
# AfterPhase[INVOKE_APPLICATIONS:5]
# BeforePhase[RENDER_RESPONSE:6]
# AfterPhase[RENDER_RESPONSE:6]
```

TeedaAjaxの拡張

以下のPageクラスのメソッドをコール

```
package teeda.web.ajax;

public class AjaxPage {
  public String ajaxStartAjax() {
    return "Start Ajax";
  }
}
```

prototype.jsでTeedaAjax

```
var test = function(){
  function showResponse(res){
    $('result').innerHTML = res.responseText;
  }
  var url = "/teeda-html-example/teeda.ajax";
  var send_param =
        "component=ajax_ajaxPage&action=ajaxStartAjax";
  var param = {parameters:send_param,
         onComplete:showResponse};
  var req = new Ajax.Request(url,param);
};
```

jqueryでTeedaAjax

```
var test = function(){
  $.ajax({
    url:"./teeda.ajax",
    type:"GET",
    data:"component=ajax_ajaxPage&action=ajaxStartAjax",
    success:function(data){
      $('#result').text(data);
    }
  });
};
```

SMART Deployを使わない

こんな人に向けて

Seasar2.3のAutoRegisterが好き

AutoRegisterで自分で登録したい！

Pageクラスを全てAutoRegisterする AutoNamingクラス作成

```
public class MyRegistNaming extends AbstractAutoNaming {
  protected String makeDefineName(String packageName, String shortClassName) {
    NamingConvention nc = SingletonS2Container
      .getComponent(NamingConvention.class);
    for (String viewRoot : nc.getRootPackageNames()) {
      String web = nc.getSubApplicationRootPackageName();
      String root = viewRoot + "." + web + ".";
      if (packageName.startsWith(root)) {
        String s = packageName.substring(root.length(), packageName
          .length());
        s = s.replace(".", "_") + "_"
          + StringUtil.decapitalize(shortClassName);
       return s;
      }
    }
    return applyRule(shortClassName);
  }
}
```

app.diconに設定

```
<component
 class="org.seasar.framework.container.autoregister.ComponentAutoRegister">
 <property name="externalBinding">true</property>
 <property name="instanceDef">
  @org.seasar.framework.container.deployer.InstanceDefFactory@REQUEST
 </property>
 <property name="autoNaming">
  <component class="examples.teeda.naming. MyRegistNaming" />
 </property>
 <initMethod name="addReferenceClass">
  <arg>@examples.teeda.web.add.AddPage@class</arg>
 </initMethod>
 <initMethod name="addClassPattern">
  <arg>"examples.teeda.web"</arg>
  <arg>".*Page"</arg>
 </initMethod>
</component>
```

Ovalでバリデーション

こんな人に向けて

JSFの影響を受けて、コンバートエラー時

バリデーションエラー時にPageクラスに

値が更新されないのが嫌だ。

エラーでもPageクラスのメソッド呼びたい。

Ovalとは？

OValはJavaで開発された検証フレームワーク

http://oval.sourceforge.net/

【使い方】アノテーションベース

```
public class Test {
  @NotNull
  @NotEmpty
  @Length(max=32)
  private String name;
}
```

sample1.html

```
<form id="sample1Form">
 <span id="allMessages"></span>
 <input type="text" id="arg1" />
 <input type="text" id="arg2" />
 <input type="submit" value="teeda" id="doTeeda"/>
</form>
```

sample2.html

```
<form id="sample1Form">
 <span id="allMessages"></span>
 <input type="text" id="arg1" />
 <input type="text" id="arg2" />
 <input type="submit" value="oval" id="doOval"/>
</form>
```

Sample1Page.java

```
public class Sample1Page {

  @Required
  public String arg1;

  @Length(maximum=2)
  public String arg2;

  public Class doTeeda() {
    System.out.println("##### doTeeda[" + this.arg1 + "] [" + this.arg2 + "]");
    return null;
  }
}
```

arg1がnullだったりarg2が3文字以上でdoTeedaは呼ばれない

Sample2Page.java

```
public class Sample1Page {
  @NotNull
  public String arg1;

  @Length(max = 2)
  public String arg2;

  public Class doOval() {
    System.out.println("##### doOval[" + this.arg1 + "] [" + this.arg2 + "]");
    Validator validator = new Validator();
    List<ConstraintViolation> violations = validator.validate(this);
    if (violations.size() > 0) {
      for (ConstraintViolation c : violations) {
        FacesContext.getCurrentInstance().addMessage(null,
        new FacesMessage(null, c.getMessage()));
      }
    }
    return null;
  }
}
```

arg1がnullでもarg2が3文字以上でもdoOvalは呼ばれる

カスタムコンポーネント作成

こんな人に向けて

JSFのカスタムコンポーネントを作成したい

凝ったコンポーネントが必要

カスタムコンポーネント作成

ちょっとだけJSFを
意識する必要があります
またJSF＞＜

ここにきて
JSFを
意識する

米林一茶

カスタムコンポーネント作成

現在時刻を表示するコンポーネント

```
<form id="testForm">
  <original:now />
</form>
```

カスタムコンポーネント作成

作るもの

- UIコンポーネント
- Tagクラス
- レンダラ

カスタムコンポーネント作成

設定ファイル

- TLDファイル
- faces-config.xml

UIコンポーネント

```
import javax.faces.component.UIComponentBase;

public class UINow extends UIComponentBase {
    public String getFamily() {
        return "NowFamily";
    }
}
```

Tagクラス

```
import javax.faces.component.UIComponent;
import javax.faces.webapp.UIComponentTag;

public class NowTag extends UIComponentTag {

    public String getComponentType() {
        return "Now";
    }

    public String getRendererType() {
        return "NowRenderer";
    }
}
```

レンダラ

```
import java.util.Date;
import java.text.DateFormat;
import java.io.IOException;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.context.ResponseWriter;
import javax.faces.render.Renderer;

public class NowRenderer extends Renderer {
   public void encodeBegin(
            FacesContext context, UIComponent component)
            throws IOException {
      if ((context == null) || (component == null)) {
         throw new NullPointerException();
      }
      DateFormat df = DateFormat.getDateTimeInstance(
           DateFormat.FULL, DateFormat.FULL);
      ResponseWriter writer = context.getResponseWriter();
      writer.write(df.format(new Date()));
   }
}
```

now.tld

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE taglib
  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
  "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
   <tlib-version>1.0</tlib-version>
   <jsp-version>1.2</jsp-version>
   <short-name>now</short-name>
   <uri>http://teeda/example</uri>
   <tag>
      <name>now</name>
      <tag-class>examples.teeda.taglib.NowTag</tag-class>
      <body-content>empty</body-content>
   </tag>
</taglib>
```

faces-config.xml

```
<component>
 <component-type>Now</component-type>
 <component-class>
 examples.teeda.taglib.UINow</component-class>
 </component>
 <render-kit>
  <renderer>
   <component-family>NowFamily</component-family>
   <renderer-type>NowRenderer</renderer-type>
  <renderer-class>
   examples.teeda.taglib.NowRenderer</renderer-class>
  </renderer>
</render-kit>
```

使い方

```
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:original="http://teeda/example"
 xml:lang="ja" lang="ja">
<html>
<body>
<form id=“tldForm”>
 <original:now />
</form>
</body>
</html>
```

このページを表示すると 2008年9月6日 04時27分16秒 JST が表示

少しノッってきたので

ネタ的に

Jythonで出来るのでは？

自論

出来ないことはない

UIコンポーネント UIHello.py

```
from javax.faces import component

class UIHello(component.UIComponentBase):
  def getFamily(self):
    return "Yone"
```

Tagクラス HelloTag.py

```
from javax.faces import webapp

class HelloTag(webapp.UIComponentTag):

  def getComponentType(self):
    return "Hello"

  def getRendererType(self):
    return "HelloRenderer"
```

レンダラ HelloRenderer.py

```
from javax.faces import render
import time

class HelloRenderer(render.Renderer):
  def encodeBegin(self, context, component):
    writer = context.responseWriter;
    writer.write(time.ctime(time.time()))
```

記述量

少ない！

tldに追加

```
<tag>
      <name>jython</name>
      <tag-class>tags.HelloTag</tag-class>
      <body-content>empty</body-content>
   </tag>
```

faces-configに追加

```
<component>
 <component-type>Hello</component-type>
  <component-class>tags.UIHello</component-class>
</component>
<renderer>
 <component-family>Yone</component-family>
 <renderer-type>HelloRenderer</renderer-type>
 <renderer-class>tags.HelloRenderer</renderer-class>
</renderer>
```

使い方

```
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:original="http://teeda/example"
 xml:lang="ja" lang="ja">
<html>
<body>
<form id=“tldForm”>
 <original:jython />
</form>
</body>
</html>
```

このページを表示すると Fri Sep 6 04:37:16 2008 が表示

意外に出来る

Jythonで
カスタムコンポーネント

少し脱線

少し前に
LL Struts
作りました

StrutsのActionを JythonやJRubyで 記述出来ます

TestAction.py

```
import sys
import org.apache.struts.action

def execute():
  mapping = struts.getMapping()
  struts.setForward(mapping.findForward("success"))

execute()
```

TestAction.rb

```
require 'java'

mapping = $struts.getMapping
$struts.setForward(mapping.findForward('success'))
```

そもそもJythonやJRubyで実装する意味は無いです。実用性も無さそうですね。単純に興味本位です。

TeedaのPageやActionも工夫次第で出来るはず。誰かTeedaにLLな風を吹かせて下さい。

脱線終わり

Teeda拡張のまとめ

カスタマイズし放題

Teedaの今後

1.0系

開発は1.0.13で終了

今後はサービスパック

1.0.13-spnで提供

1.1系

Teeda Extentionの向上

以下新機能予定

- ネストしたプロパティ対応
- Forwardによる画面遷移対応
- コマンドリンク対応

1.2系

Teeda CoreがJSF1.2準拠

動作環境

- Java SE 5.0以降
- Servlet 2.5
- JSP 2.1

2008年中くらいには実装完了予定

最後に

探求心

大切

これなんだろう？

と思ったら

手を動かしてみてください

何か

生み出せます

最後に一句

探求心
持たなくなったら
ホトトギス

米林一茶

ご清聴
ありがとう
ございました