

Java EE開発におけるJSFの 活用について

株式会社構造計画研究所
製造ビジネス・ソリューション部
菊田 洋一

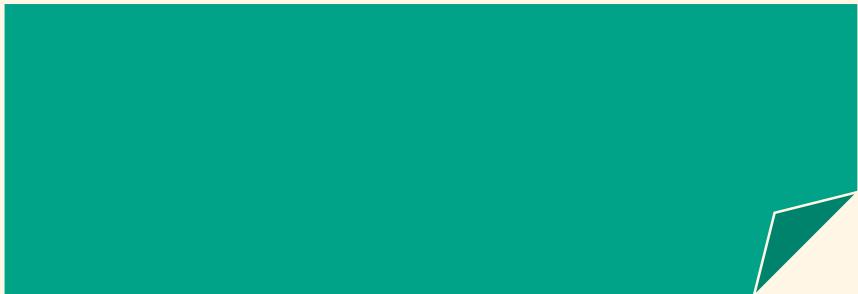
自己紹介

菊田洋一 Twitter :@kikutaro_

株式会社 構造計画研究所 製造ビジネス・ソリューション部

「**システム開発**」を利用した

Java歴は約  年



自己紹介

菊田洋一 Twitter : @kikutaro_

株式会社 構造計画研究所 製造ビジネス・ソリューション部

コンフィグレータを利用した製造業向け販売管理システム開発

Java歴は約  年

Challenge Java EE !

Java EEを中心に趣味や仕事における開発メモを書いています。

<http://kikutaro777.hatenablog.com/>

本日紹介するコード(一部)はGitHubにあります！
<https://github.com/kikutaro/JavaDayTokyo2015JSF>

JSFとは？

始め方

基本機能

JSF

JavaServer Faces

開発現場での
活用

PrimeFaces

JSF Tips

JSFとは？

始め方

基本機能

JSF

JavaServer Faces

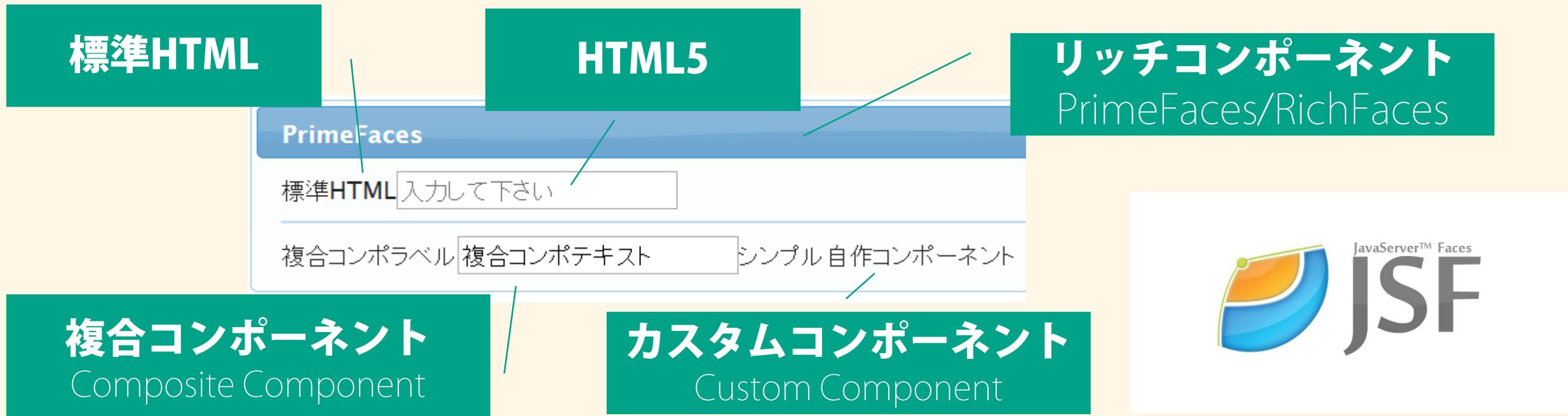
**開発現場での
活用**

PrimeFaces

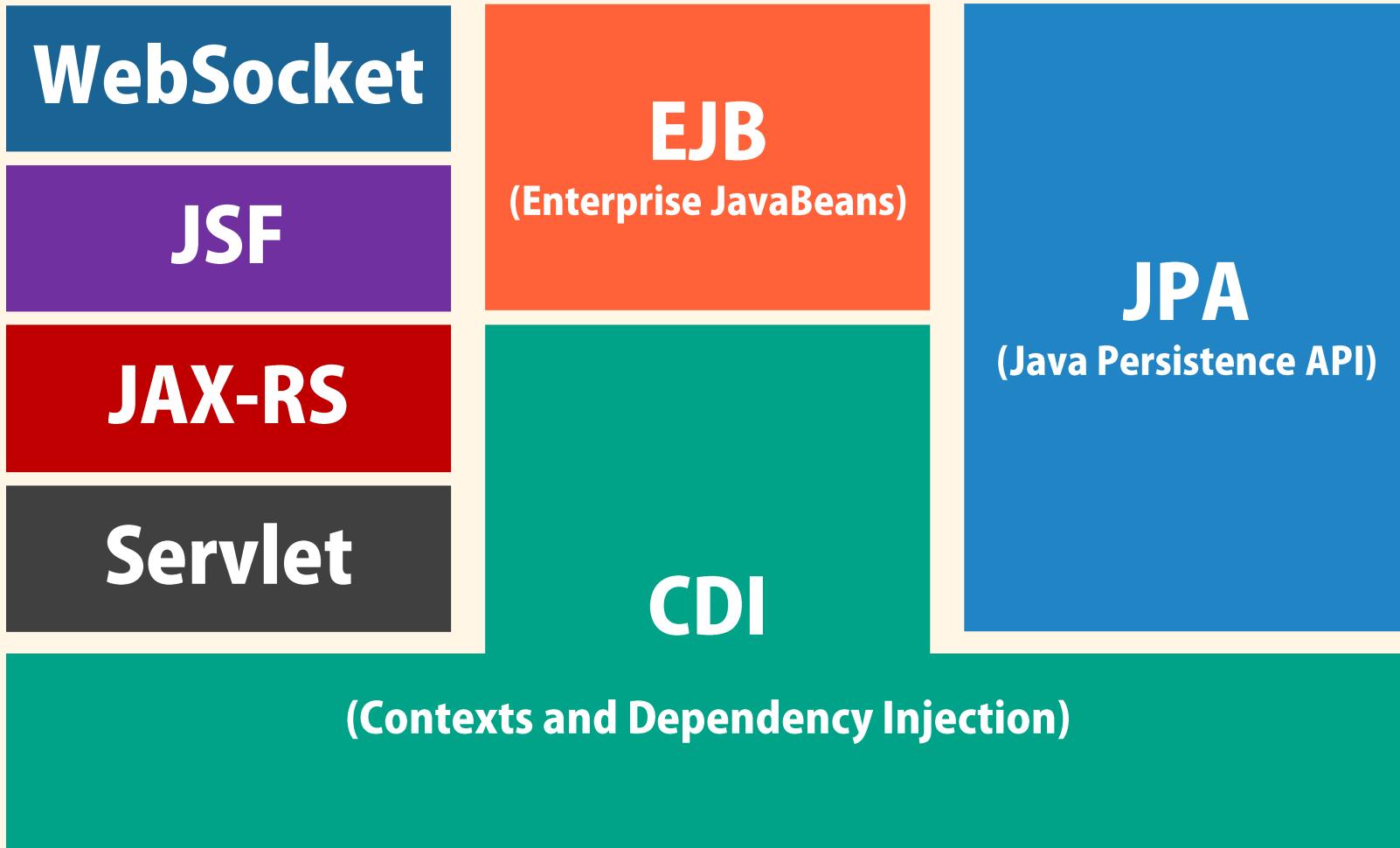
JSF Tips

JSFとは？

- JavaServer Faces
- Webアプリケーションの**ユーザインターフェース**構築技術
- コンポーネントベース (not アクションベース)
- 画面部品の**共通化**が容易、**再利用性**が高い



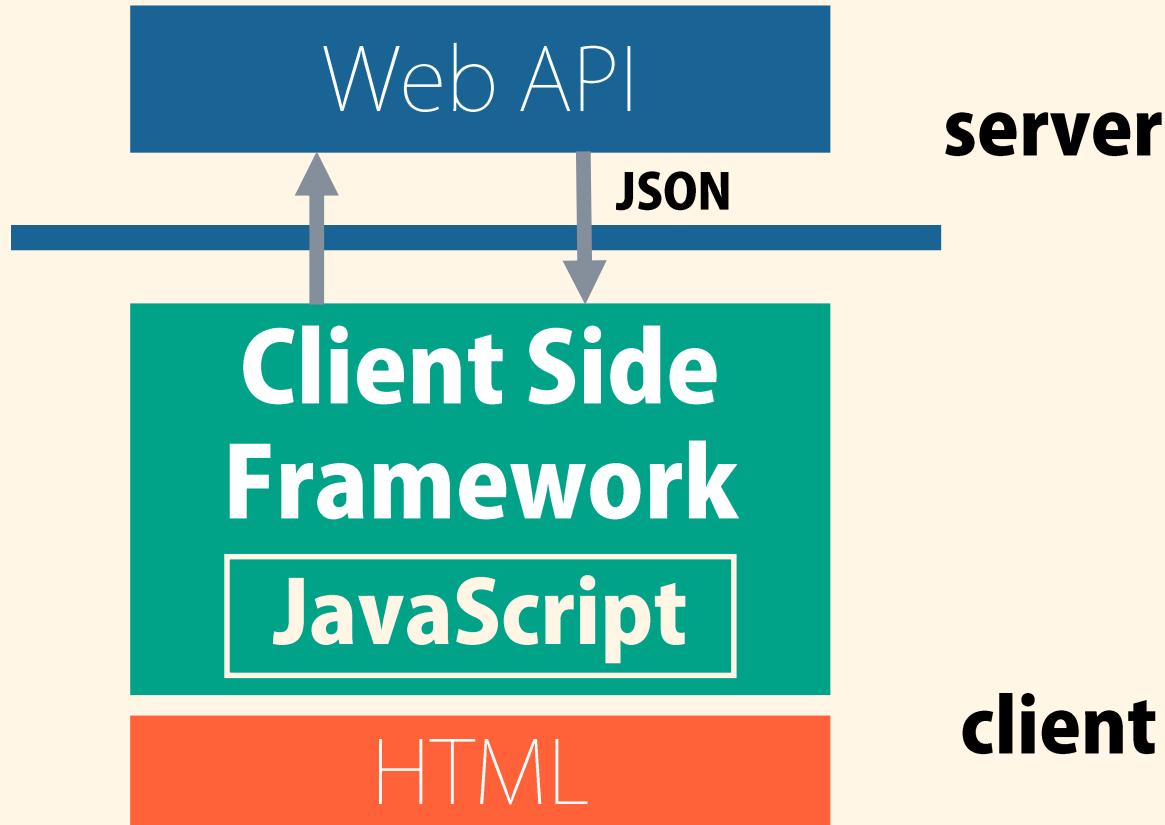
Java EEにおけるJSFの位置づけ



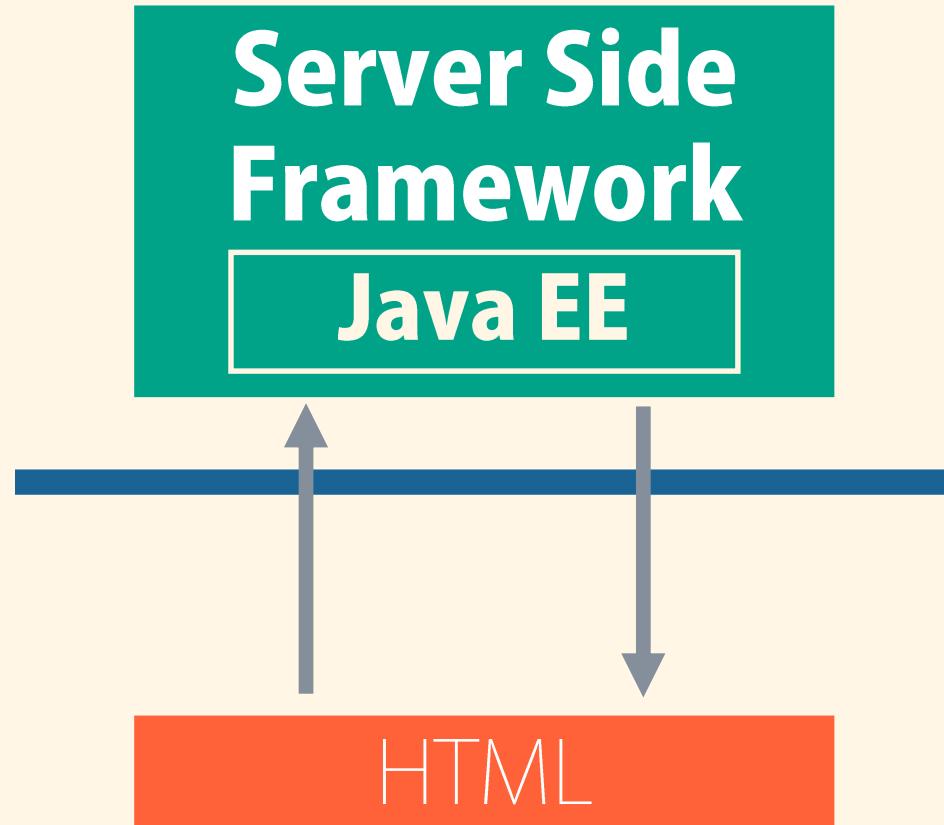
Web Architecture

Single **P**age **A**pplication

Model-**V**iew-**V**iew**M**odel(MVVM)



JSFはサーバサイドから
HTMLを出力



JSFの評判

Negative

Positive

Technology Radar

ThoughtWorks

Java 8
ADOPT

•ADOPT

- 83.Go language
- 84.Java 8

•TRIAL

- 85.AngularJS
- 86.Core Async
- 87.Dashing new
- 88.Django Rest new
- 89.HAL
- 90.Ionic Framework new
- 91.Nashorn new
- 92.Om
- 93.Q & Bluebird
- 94.R as Compute Platform
- 95.Retrofit new



•ASSESS

- 96.Flight.js new
- 97.Haskell Hadoop library new
- 98.Lotus new
- 99.React.js new
- 100.Reagent new
- 101.Rust
- 102.Spring Boot
- 103.Swift new

JSFは**HOLD**
動向に注意が必要

•HOLD

104.JSF

Java EE 8ではJSFと異なるアクションベースのMVCが加わる JSR371 Model-View-Controller(MVC 1.0) Specification

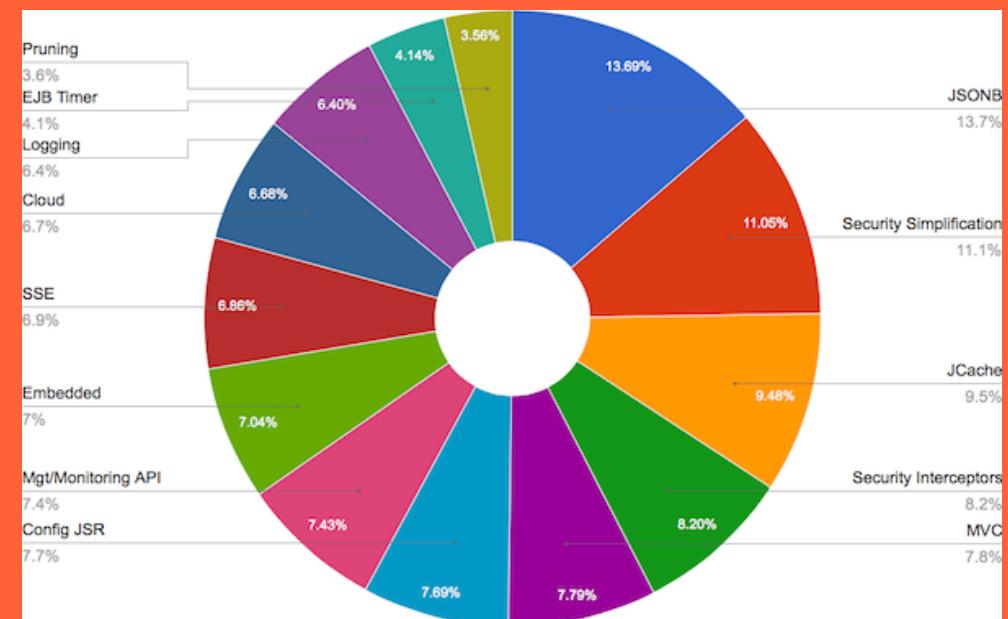
The screenshot shows a web page from the Oracle Technology Network. The top navigation bar includes links for Account, Sign Out, Help, Country, Communities, I am a..., I want to..., Search, Products, Solutions, Downloads, Store, Support, Training, Partners, About, and OTN. The main content area is titled "Why Another MVC?" by Ed Burns. It discusses the recent filing of a new JSR for an MVC 1.0 framework in Java EE 8 and its relationship to JSF. A diagram titled "UI Component Oriented MVC == JSF" illustrates the architecture, showing a "Controller" (oval) connected to multiple "View" (triangle) components, which in turn interact with "CDIBean1", "CDIBean2", and "CDIBean3" (rectangles). These beans are connected to "Model Objects" (dashed box). A note explains that with JSF, the model is CDI, the View is your Facelet pages, and the controller is the JSF lifecycle. The page also lists various Oracle technology categories in a sidebar.

Why Another MVC? EdBurnさん

<http://www.oracle.com/technetwork/articles/java/mvc-2280472.html>

Final Results from our Java EE 8 Community Survey

https://blogs.oracle.com/l demichel/entry/results_from_th e_java_ee



Java EE 8 & JSF 2.3

JSR372 機能拡張は継続

※注意：仕様策定中のため変わる可能性があります。
まだ策定初期にて今後色々増えていきます。

- FacesContextやExternalContextなどがInject可能になる

```
@Inject  
private FacesContext context;
```

- ConverterやValidatorの中でCDIによるInjectが可能に

```
@FacesConverter(value="myConverter", managed=true)  
public class MyConverter implements Converter{  
    @Inject  
    private BizLogic bizlogic;  
  
    //override の getAsObjectやgetAsStringは略  
}
```

サードパーティの動き

2015年3月9日 RichFaces 4.5.3 Final Release

2015年4月6日 PrimeFaces 5.2 RC3 Release

モバイル対応・レスポンシブWebに対応したコンポーネントライブラリも



2015年2月27日 Sentinel Theme

2015年4月3日 SPARK

JSFの評判

Negative

Technology Radarでは**Hold**

Java EE 8で**新たなMVC**も入る

仕組みが**複雑**(HTTPの観点で)

採用判断になるか微妙ですが…Java EE 7で素直に作ればJSFかと

Positive

Java EE 8の**JSF2.3** 繼続拡張

サードパーティも**進化**

海外情報が多い

(StackOverflow / Forum等)

JSFとは？

始め方

基本機能

JSF

JavaServer Faces

開発現場での
活用

PrimeFaces

JSF Tips

実際に作ってみましょう

Java EEってアプリケーションサーバに入れたり
開発環境の構築だけでも大変そう

実際に作ってみましょう

Java EEってフレームワークやショットカーバに入れたり
開発環境構築も大変そう

NO!

実際に作ってみましょう

NetBeans Demo

実際に作ってみましょう

<https://ja.netbeans.org/>

The screenshot shows the NetBeans IDE 8.0.2 download page. At the top, there's a navigation bar with the NetBeans logo and links for NetBeans IDE, NetBeans Platform, Plugins, and Docs & Support. Below the navigation bar, there's a breadcrumb trail: netbeans.org > projects > ja > Website. The main content features a large blue cat-like mascot named 'ねこび~ん'. To its right, the text reads 'NetBeans IDE' and 'The Smarter and Faster Way to Code'. Below this, it says 'デスクトップ、モバイル、ウェブアプリケーションを Java、' and 'NetBeans IDE は無償のオープンソースソフトウェアで全世界のコミュニティーウェブユーザーと開発者に支援されています。' A section titled 'ユーザーインターフェースは日本語に翻訳されているのでと' follows. A red box highlights the '最新版(日本語版)ダウンロード' button. At the bottom, there are sections for 'NetBeans 8.0.2 (最新版)', 'ダウンロード(日本語版)', 'リリース情報', and 'リリースノート'.

The screenshot shows the 'NetBeans IDE 8.0.2 ダウンロード' page. At the top, there are dropdown menus for 'IDE の言語' (Japanese) and 'プラットフォーム' (Windows). Below this, there's a note: '注: グレイアウトされているテクノロジーはこのプラットフォームではサポートされていません' (Note: Layouted technologies are not supported on this platform). On the left, there's a list of supported technologies: NetBeans Platform SDK, Java SE, Java FX, Java EE, Java ME, HTML5, Java Card(TM) 3 Connected, C/C++, Groovy, PHP, GlassFish Server Open Source Edition 4.1, and Apache Tomcat 8.0.15. To the right, there's a grid for selecting a download bundle. The columns are labeled 'Java SE', 'Java EE', 'C/C++', 'PHP', and 'すべて'. Three specific rows in the grid are highlighted with red boxes: the first row under Java EE, the second row under C/C++, and the third row under PHP. Each row has a 'ダウンロード' button at the bottom.



- PC
 - ダウンロード
 - デスクトップ
 - ドキュメント
 - ピクチャ
 - ビデオ
 - ミュージック
 - ローカル ディスク (C:)
 - ボリューム (D:)
- ネットワーク



1 個の項目 1 個の項目を選択 2.05 KB



22:03
2015/04/02

localhost:8080/JavaDayTokyo2015/



検索



入力欄





Java Day Tokyo 2015

j_idt5:j_idt8: 検証エラー: 値が必要です。

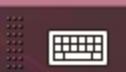
 送信

▼ A

A

▶ B

▶ C



JSFとは？

始め方

基本機能

JSF

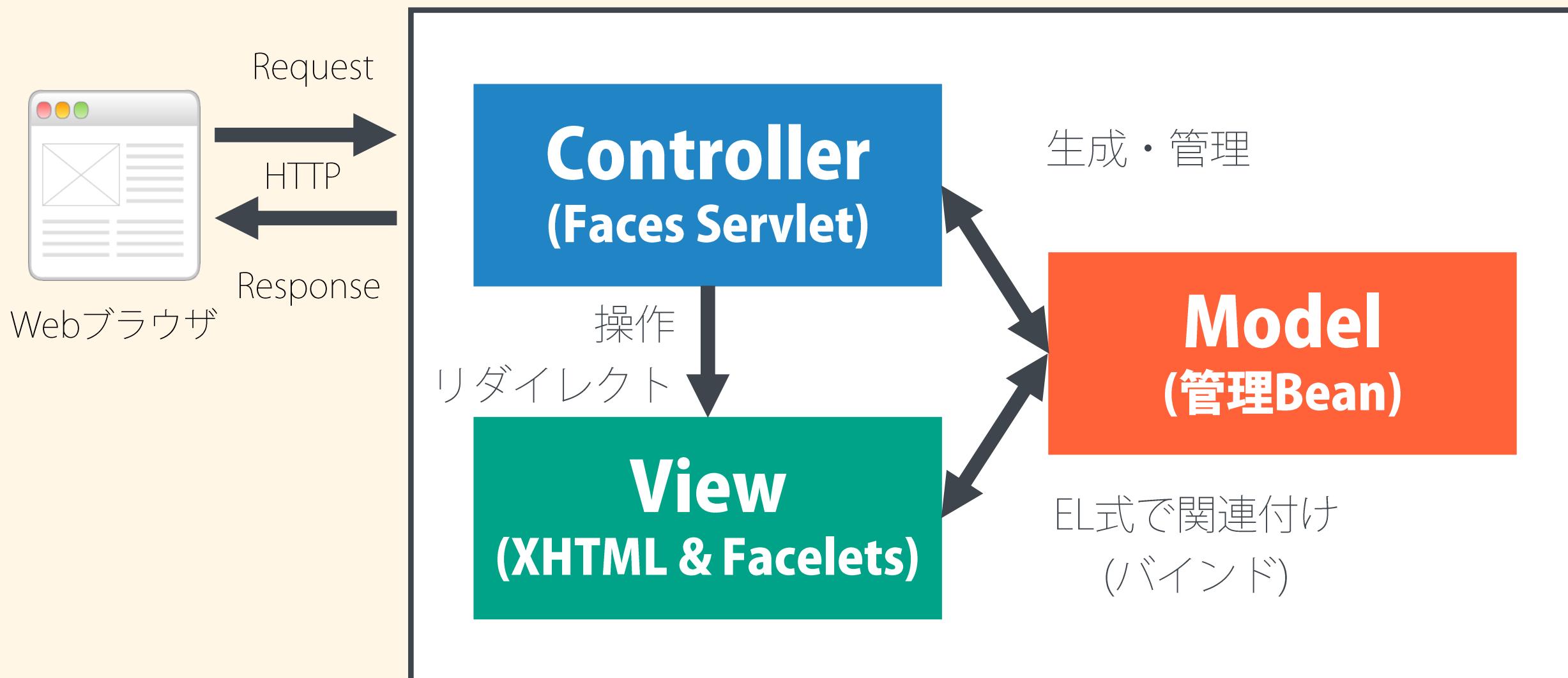
JavaServer Faces

開発現場での
活用

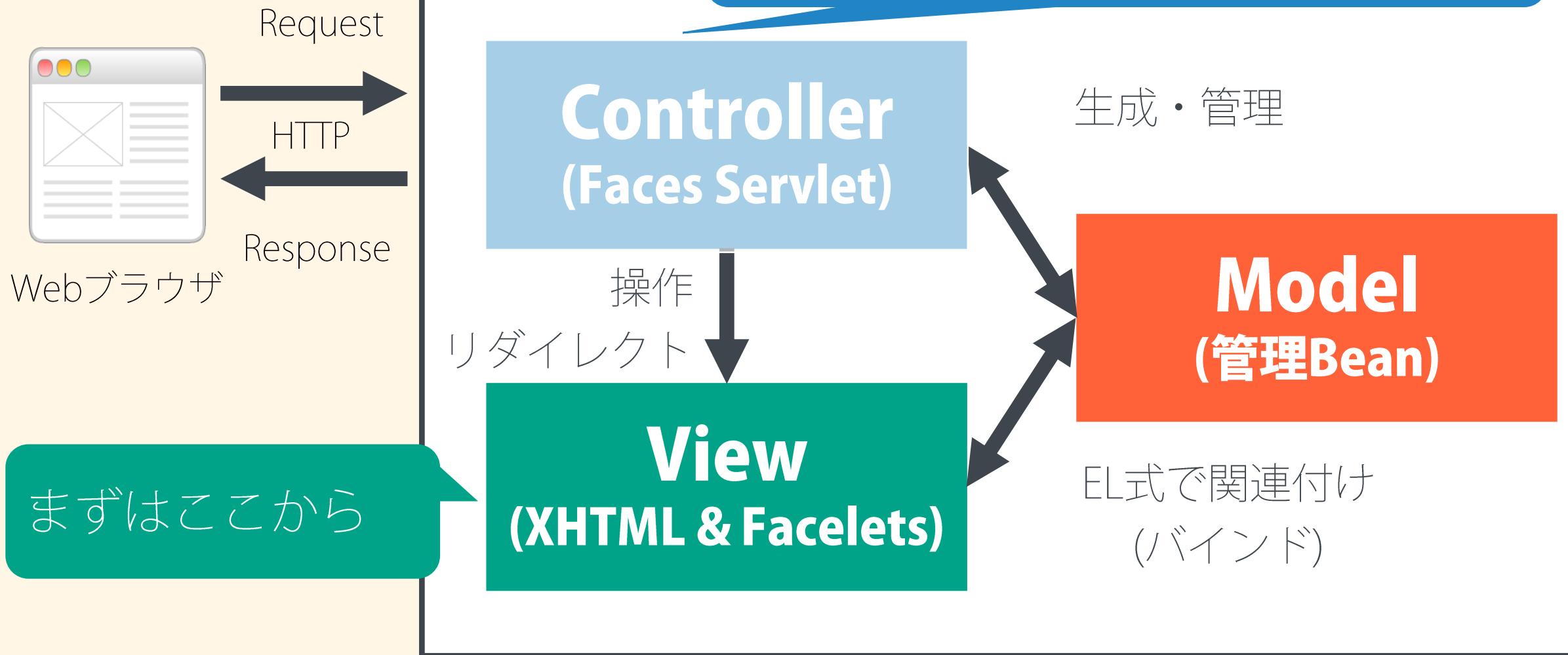
PrimeFaces

JSF Tips

基本構成



基本構成



Facelets & XHTML

JSFタグを利用したビュー定義

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head><title>TODOリスト</title></h:head>
  <h:body>
    <h:form id="frm">
      <h:outputLabel value="TODOを入力" />
      <h:inputText value="#{todoBean.todo}" />
      <h:commandButton value="登録"
                        action="#{todoBean.register()}" />
    </h:form>
  </h:body>
</html>
```

Namespace

HTMLタグ(ページ関連)

HTMLタグ(フォーム)

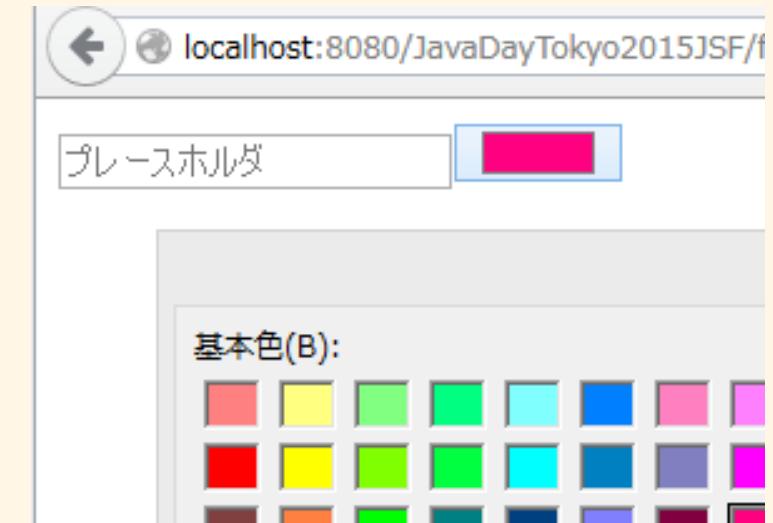
HTMLタグ(コントローラ)

TODOを入力

Facelets & HTML5

PassThrough AttributeによるHTML5対応

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://xmlns.jcp.org/jsf/passthrough">
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>
    <h:form>
      <h:inputText p:placeholder="プレースホルダ" />
      <h:inputText p:type="color" />
    </h:form>
  </h:body>
</html>
```



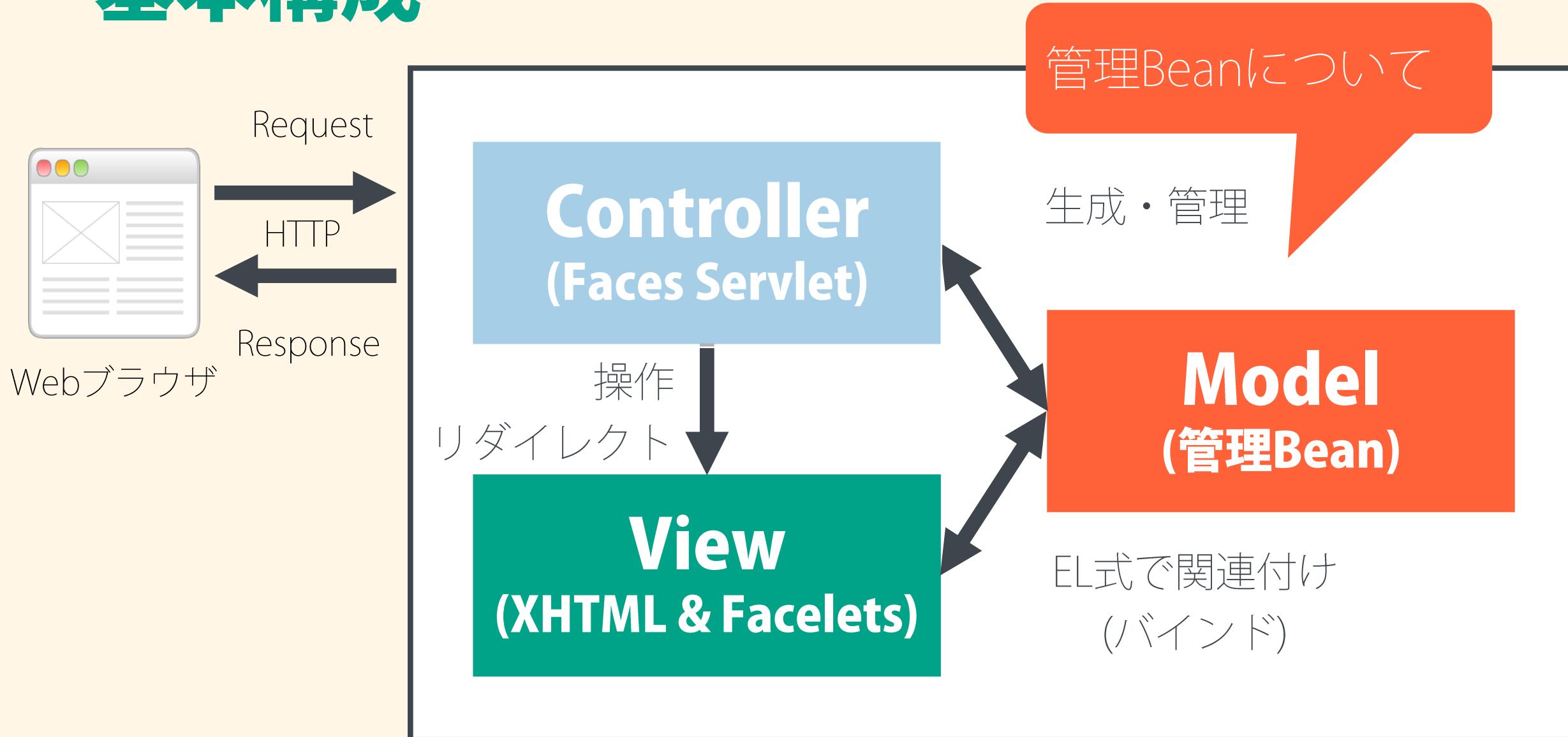
Facelets & HTML

デザイナさんと協業などで素のHTMLを記述したい場合は
JSF属性を利用

```
<input type="text" jsf:value="#{sampleBean.value}" />
```

私自身、実開発では利用したことないので、もし
ご利用された方は情報発信を是非(^^;

基本構成



管理Bean(Managed Bean)

ビューとバインド(関連付け)するデータや処理を持つクラス

```
<h:outputLabel value="TODOを入力"/>
<h:inputText value="#{todoBean.todo}" />
<h:commandButton value="登録"
                  action="#{todoBean.register()}" />

<h:dataTable var="t" value="#{todoBean.listTodo}">
  <h:column>
    <h:outputText value="#{t}" />
  </h:column>
</h:dataTable>
```

Facelets

TODOを入力	JJUG CCC参加	登録
Java Day Tokyo参加		
JJUG CCC参加		

```
@Named(value = "todoBean")
@ViewScoped
public class TodoBean implements Serializable {
  @Getter @Setter
  private String todo;

  @Getter @Setter
  private List<String> listTodo;

  public void register(){
    listTodo.add(todo);
  }
}
```

管理Bean

※@Getter @SetterはLombokライブラリ利用

管理Bean(Managed Bean)

```
@Named(value="hogeBean")  
@RequestScoped
```

```
public class HogeBean {  
    @Inject  
    private BizLogic bizlogic;  
}
```

@Named : CDI管理Beanであることを表す

@xxScoped : 管理Beanのスコープを表す

CDIを利用したインジェクションが可能
(Context and Dependency Injection)

newする必要がない
EJBや他のCDIインスタンス等

依存性の注入により
インスタンス生成

Java EEには現在、**CDI管理Bean**と**JSF管理Bean**(@ManagedBean)の2種類があり、利用するアノテーションを混同しやすいので注意です。Java EE 7以降ではCDIの管理Beanを推奨。**@Namedを確認**すればOKです。

管理Beanのスコープ

管理Beanインスタンスの生存期間

Request Scope

View Scope

Conversation Scope

Flow Scope

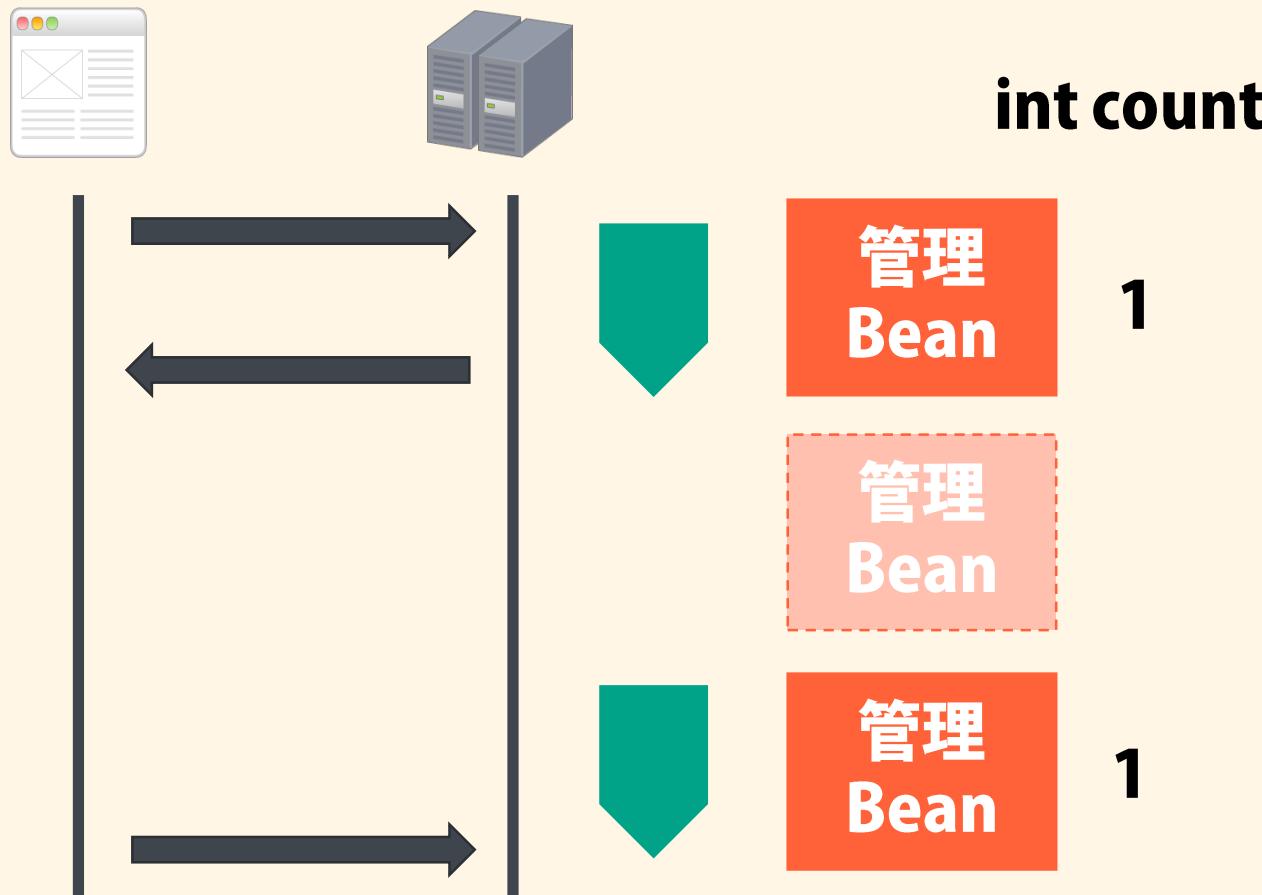
Session Scope

Application Scope

Dependent

Request Scope

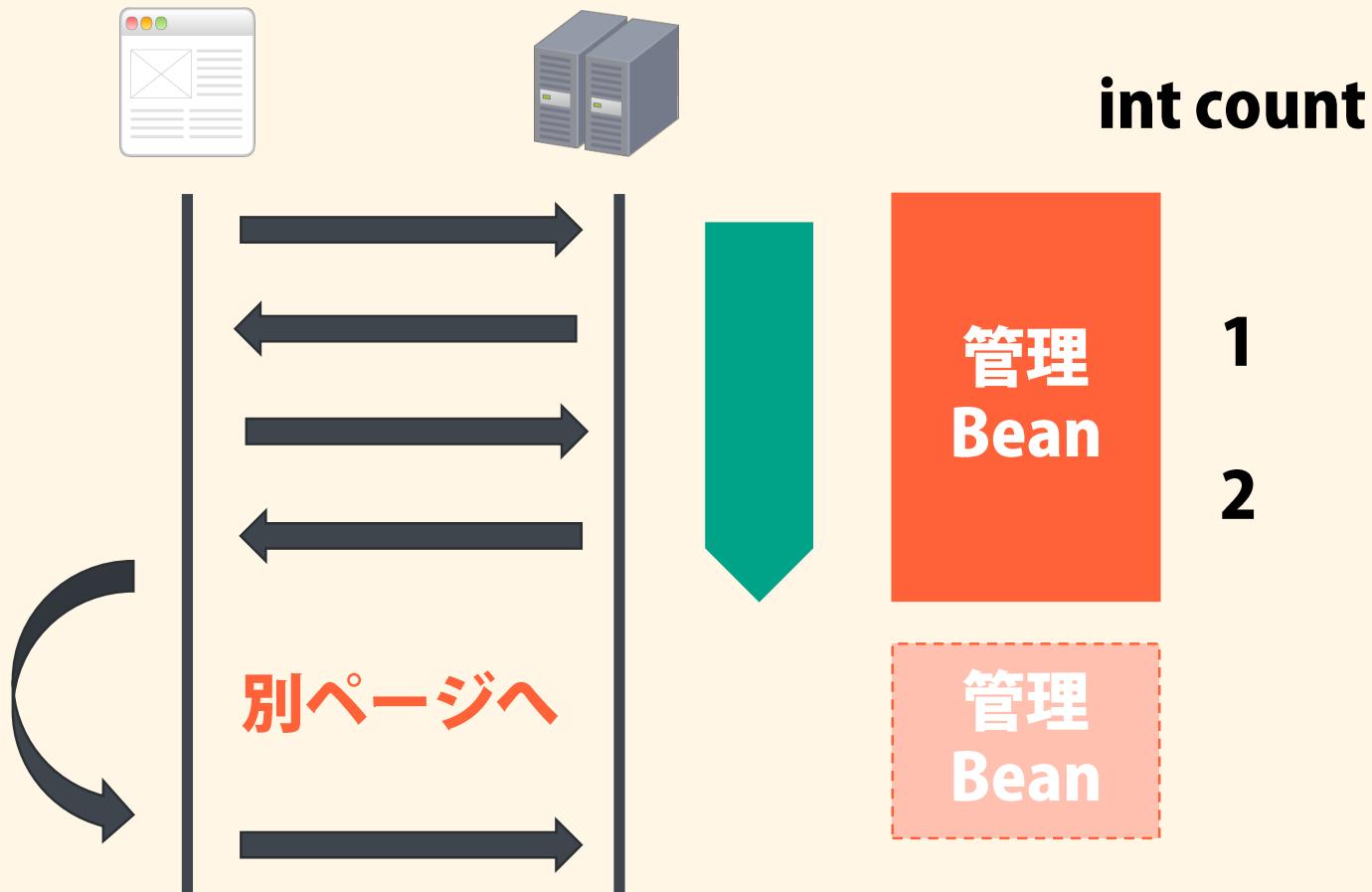
1回のリクエスト/レスポンス間のみ生存



```
@Named  
@RequestScoped  
public class HogeBean {  
}
```

View Scope

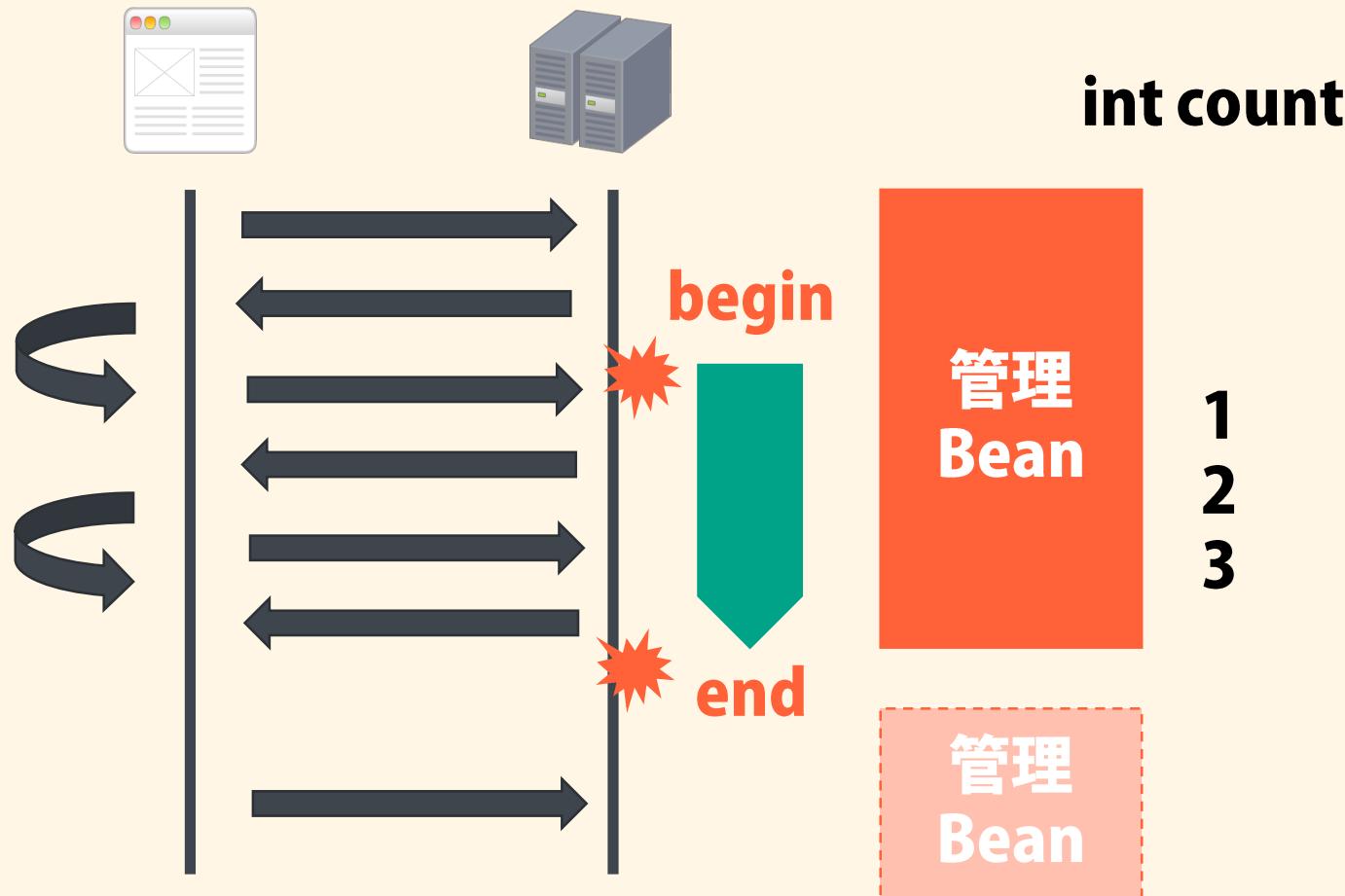
ビューが変わらない間は生存



```
@Named  
@ViewScoped  
public class HogeBean  
implements Serializable{  
}
```

Conversation Scope

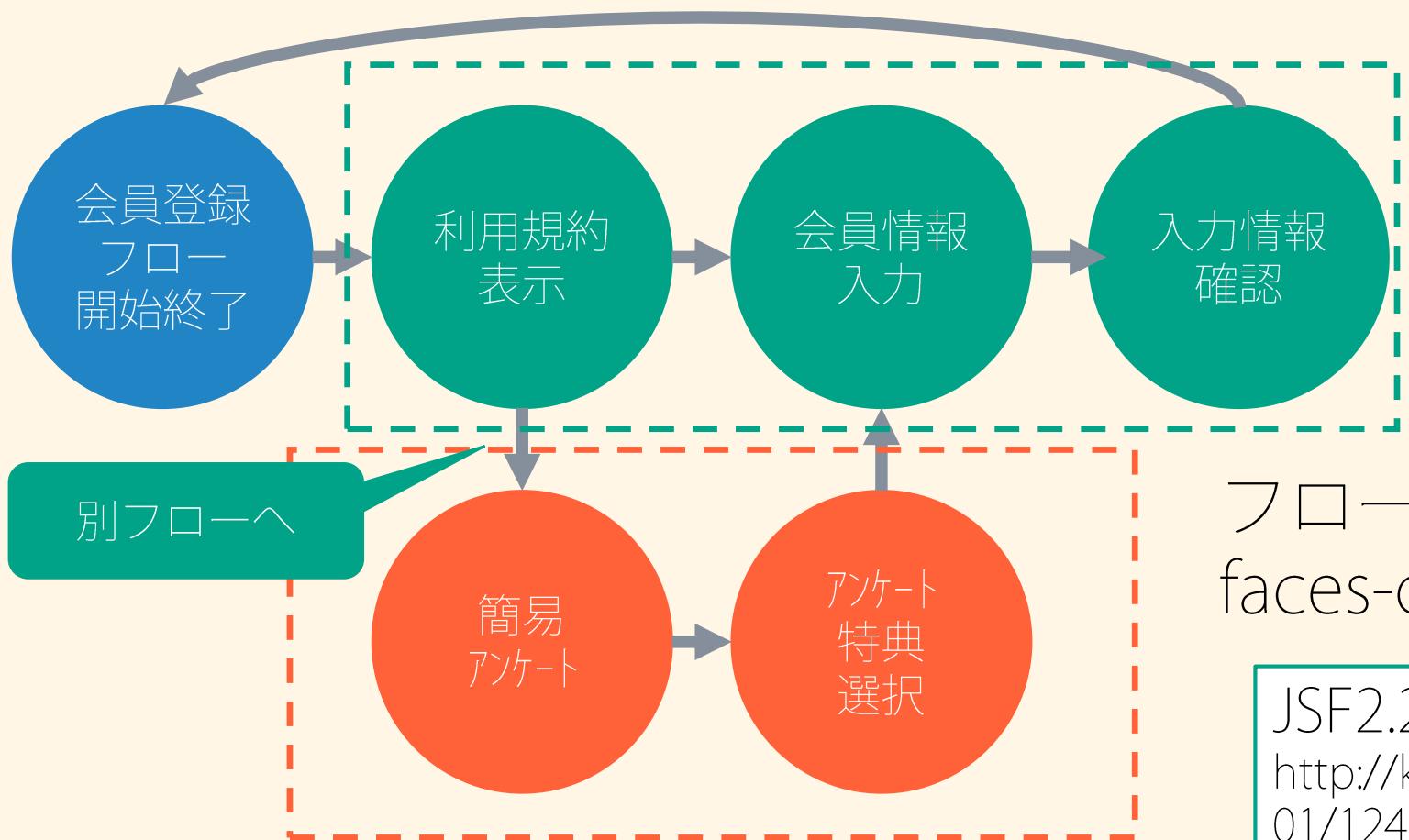
会話の開始と終了を指定して、その会話の間は生存



```
@Named  
@ConversationScoped  
public class HogeBean  
    implements Serializable{  
}
```

Flow Scope

フローを定義して、そのフローの範囲で生存



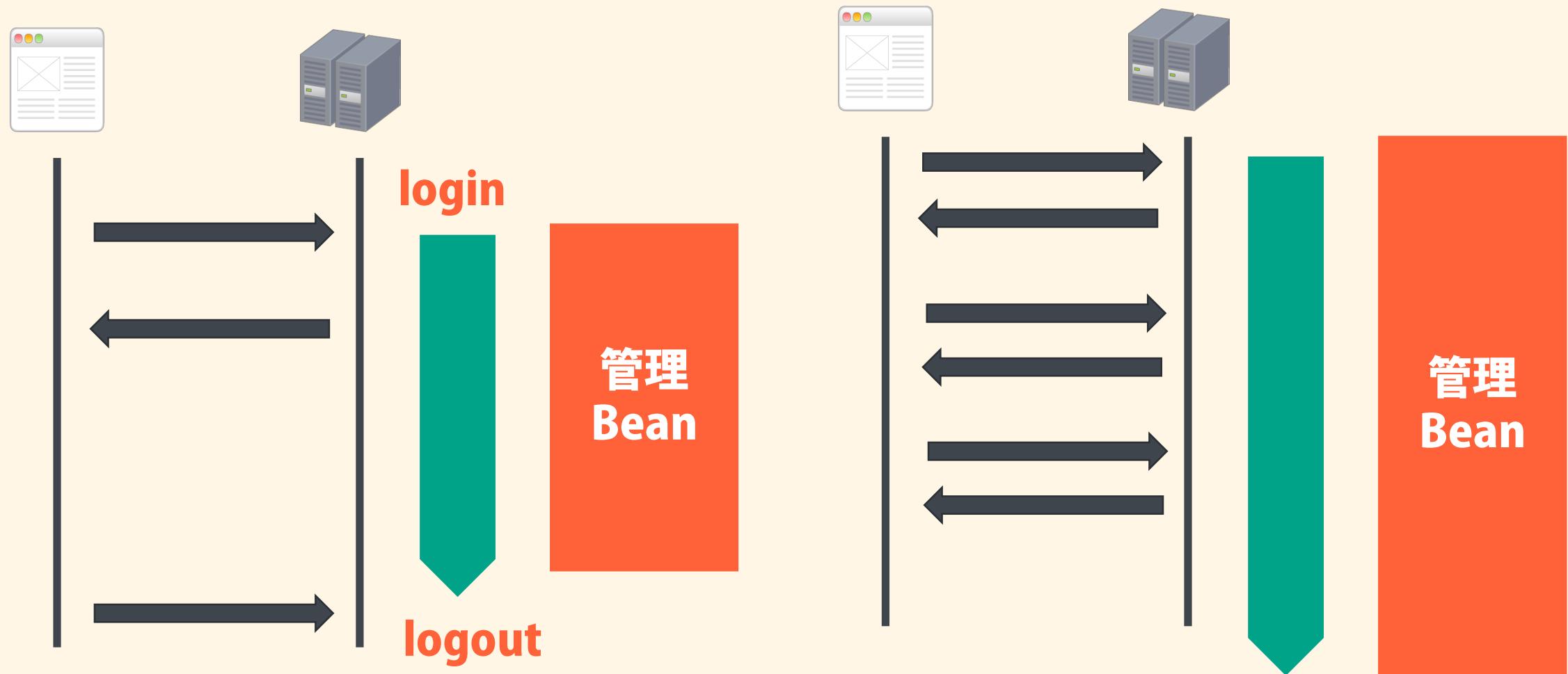
```
@Named  
@FlowScoped  
public class HogeBean  
    implements Serializable{  
}
```

フローの定義は
faces-config.xmlまたはFlowBuilder

JSF2.2のFaces Flows (FlowScoped)
<http://kikutaro777.hatenablog.com/entry/2013/12/01/124358>

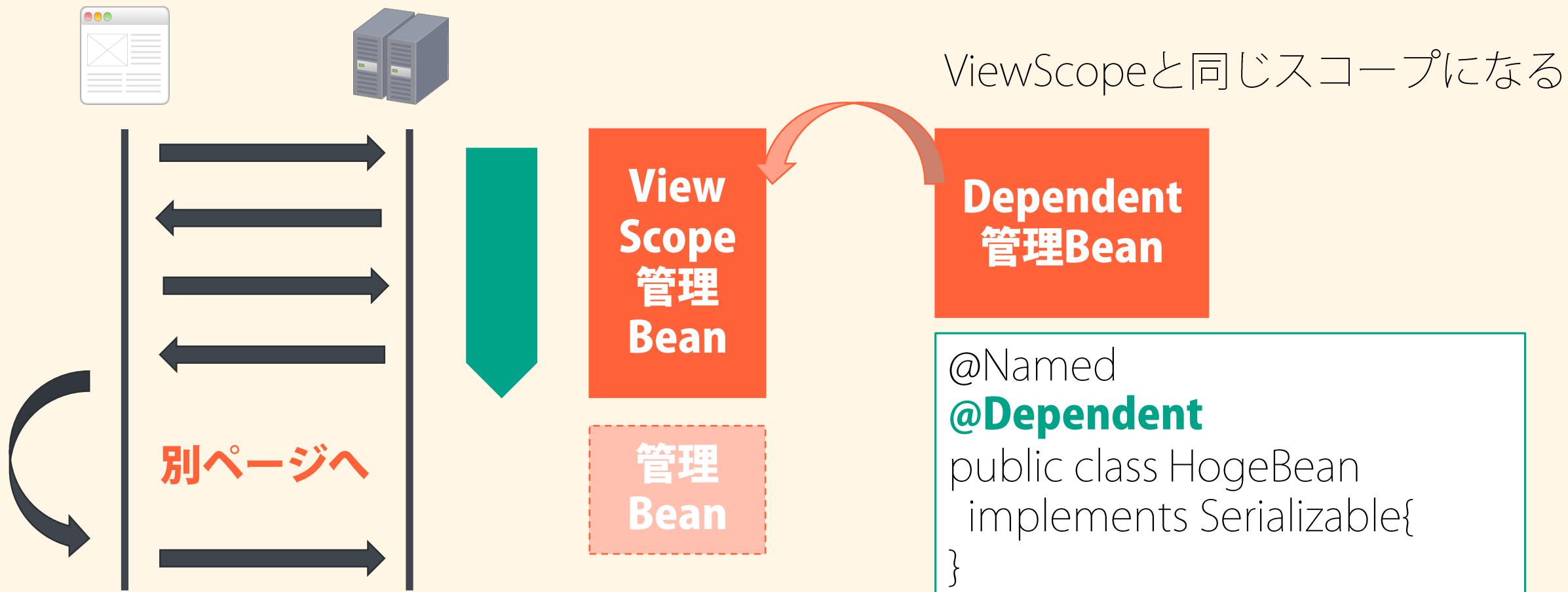
Session Scope / Application Scope

セッションが生きてる間 / アプリケーションが生きてる間



Dependent

インジェクション先のスコープに依存



管理Beanのスコープ

Request Scope

1回のやりとりで
処理が済む場合

View Scope

RequestとSessionの間
一番便利でよく使う

Session Scope

Application Scope

期間が長いのでメモリ意識
特にApplication Scope

Conversation Scope

Dependent

Flow Scope ウィザードのような
画面で有用

個人的にはFlowScopeのほうが使い勝手良い

インジェクト先に依存
させたい場合

Expression Language

ビューと管理BeanをBind(結びつける)する式・言語

```
<h:outputLabel value="TODOを入力" />
<h:inputText value="#{todoBean.todo}" />
<h:commandButton value="登録"
                  action="#{todoBean.register()}" />

<h:dataTable var="t" value="#{todoBean.listTodo}">
    <h:column>
        <h:outputText value="#{t}" />
    </h:column>
</h:dataTable>
```

Value Expression

#{{todoBean.todo}}

Method Expression

#{{todoBean.register()}}

Expression Language

ビューと管理BeanをBind(結びつける)する式・言語

種類	式
算術式	+,-,*/,%,
論理式	&&(and), (or),!(not)
関係式	==(eq),!=(ne),<(lt),>(gt),<=(le),>=(ge)
ラムダ式	->
その他	(),empty,[],A ? B : C,+=,;

```
<h:outputText  
    value="#{(1 * 5) + (10 / 2) - 1}" />  
  
    "#{{((x,y) -> x + y)(3, 4)}"  
  
    "#{elBean.listName.stream()  
        .filter(n -> n.contains('ai')).toList()}"
```

ELで非常に柔軟な処理記述ができます。が、あくまでもビュー側であることを意識して、どこまで記述するか、バランスを大切に

Ajax(Aynchronous JavaScript + XML)

- ・ビューの部分更新
- ・<f:ajax>タグを入れるだけ



Non - Ajaxなボタン

```
<h:commandButton id="btnNonAjax" value="non-ajax送信"  
actionListener="#{ajaxBean.combine()}" />
```

Ajax利用したボタン

```
<h:commandButton id="btnAjax" value="ajax送信"  
actionListener="#{ajaxBean.combine()}>  
  <b><f:ajax execute="@this frm:msg1 frm:msg2" render="frm:output" /></b>  
</h:commandButton>
```

executeで送信対象のコンポ、renderで書き換え対象のコンポを指定

Ajax(Asynchronous JavaScript + XML)

Non - Ajaxレスポンス

localhost:8080/JavaDayTokyo/faces/ajax/ajax.xhtml

Java Day Tokyo 2015

non-ajax送信 ajax送信 Java Day Tokyo2015

	inspect	console	debug	style editor	
✓	メソッド	ファイル	ドメイン		
● 200 POST	ajax.xhtml	localhost:8080			
● 200 POST	ajax.xhtml	localhost:8080			

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"><head><link type="text/css" rel="stylesheet" href="/JavaDayTokyo/faces/javax.faces.resource/theme.css?ln=primefaces-redmond" /><script type="text/javascript" src="/JavaDayTokyo/faces/javax.faces.resource/jsf.js?ln=javax.faces"></script>
<title>Facelet Title</title></head><body>
<form id="frm" name="frm" method="post" action="/JavaDayTokyo/faces/ajax/ajax.xhtml" enctype="application/x-www-form-urlencoded">
<input type="hidden" name="frm" value="frm" />
<table>
<tbody>
<tr>
<td><input id="frm:msg1" type="text" name="frm:msg1" value="Java Day Tokyo" /></td>
<td><input id="frm:msg2" type="text" name="frm:msg2" value="2015" /></td>
</tr>
</tbody>
</table>
<input id="frm:btnNonAjax" type="submit" name="frm:btnNonAjax" value="non-ajax送信" /><input id="frm:btnAjax" type="submit" name="frm:btnAjax" value="ajax送信" onclick="mojarra.ab(this,event,'action','@this frm:msg1 frm:msg2','frm:output');return false;" /><span id="frm:output">Java Day Tokyo2015</span><input type="hidden" name="javax.faces.ViewState" id="j_id1:javax.faces.ViewState:0" value="-5245659688803276564:3884769283749958235" autocomplete="off" />
</form></body>
</html>
```

Ajaxレスポンス

```
<?xml version='1.0' encoding='UTF-8'?>
<partial-response id="j_id1"><changes><update id="frm:output"><![CDATA[<span id="frm:output">Java Day Tokyo2015</span>]]></update><update id="j_id1:javax.faces.ViewState:0"><![CDATA[-5245659688803276564:3884769283749958235]]></update></changes></partial-response>
```

Ajax(Aynchronous JavaScript + XML)

eventやlistener属性を利用

```
<h:inputText>
  <f:ajax event="blur" listener="#{ajaxEventBean.executeFromBlur()}" />
</h:inputText>
```

listenerはEL式で処理を呼び出し

複数のコンポに一括でAjax処理

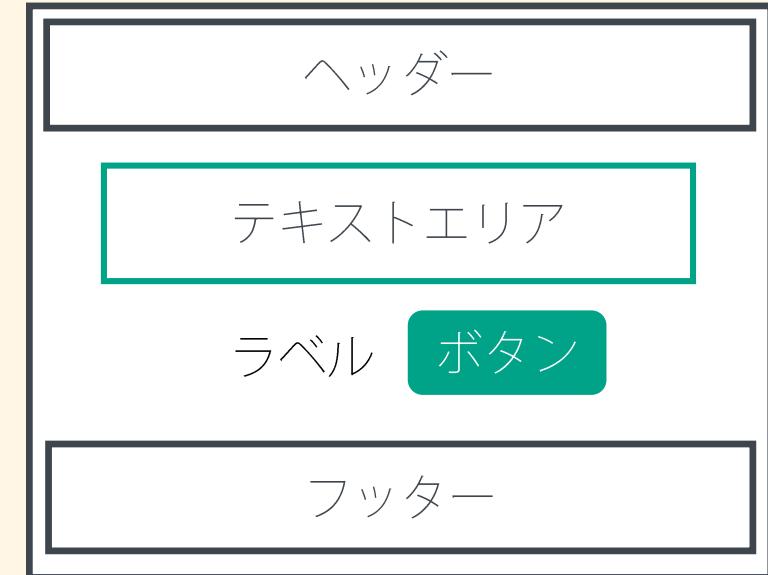
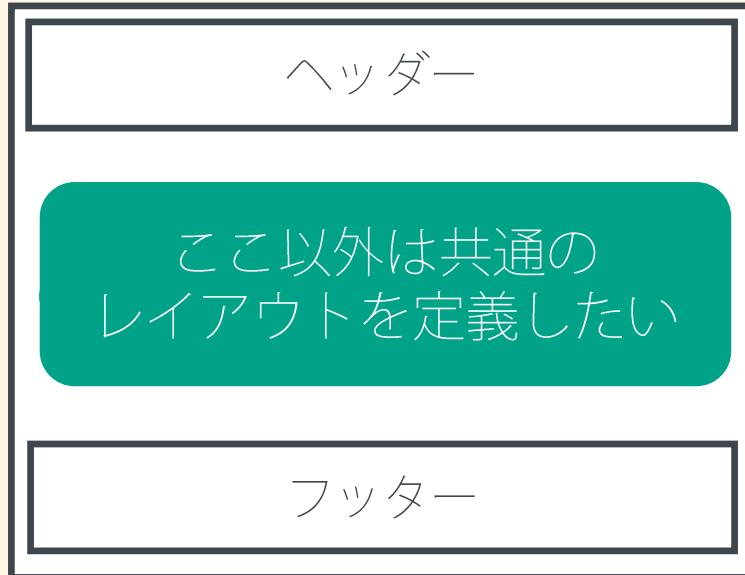
```
<f:ajax event="click"
  listener="#{ajaxEventBean.executeFromClick()}">
  <h:inputText />
  <h:inputTextarea />
</f:ajax>
```

eventでは
blur:フォーカスアウト
click:クリック
dblclick:ダブルクリック
keydown:キー入力
mouseover:マウスオーバー
等(その他にも様々イベントあり)

ajaxタグ内で複数コンポを定義

Template

レイアウトの共通部分をテンプレート化



Template

テンプレートの定義

```
<h:form>
  <ui:insert name="header">
    <h:outputText value="ヘッダー" />
    <p:separator />
  </ui:insert>

  <ui:insert name="content" />

  <ui:insert name="footer">
    <p:separator />
    <h:outputText value="フッター" />
  </ui:insert>
</h:form>
```

テンプレートを利用する側

```
<h:body>
  <ui:composition
    template="template.xhtml">
    <ui:define name="content">
      <h:outputText value="コンテンツ" />
      <h:commandButton value="ボタン" />
    </ui:define>
  </ui:composition>
</h:body>
```



Bookmarkable

GETパラメータのハンドリング

http://app/myService?param1=JavaDay¶m2=Tokyo2015

```
<f:metadata>
  <f:viewParam name="param1"
    value="#{toBean.param1}" />
  <f:viewParam name="param2"
    value="#{toBean.param2}" />
  <f:viewAction action="#{toBean.action()}" />
  <f:event type="preRenderView"
    listener="#{toBean.preRenderView()}" />
</f:metadata>
```

パラメータを利用して処理
をキックする場合

```
@Getter @Setter
private String param1;
```

```
@Getter @Setter
private String param2;
```

```
public void action(){}
 (INVOKE_APPLICATION)
```

```
public void preRenderView(){}
  (RENDER_RESPONSE)
```

Composite Component (複合コンポーネント)

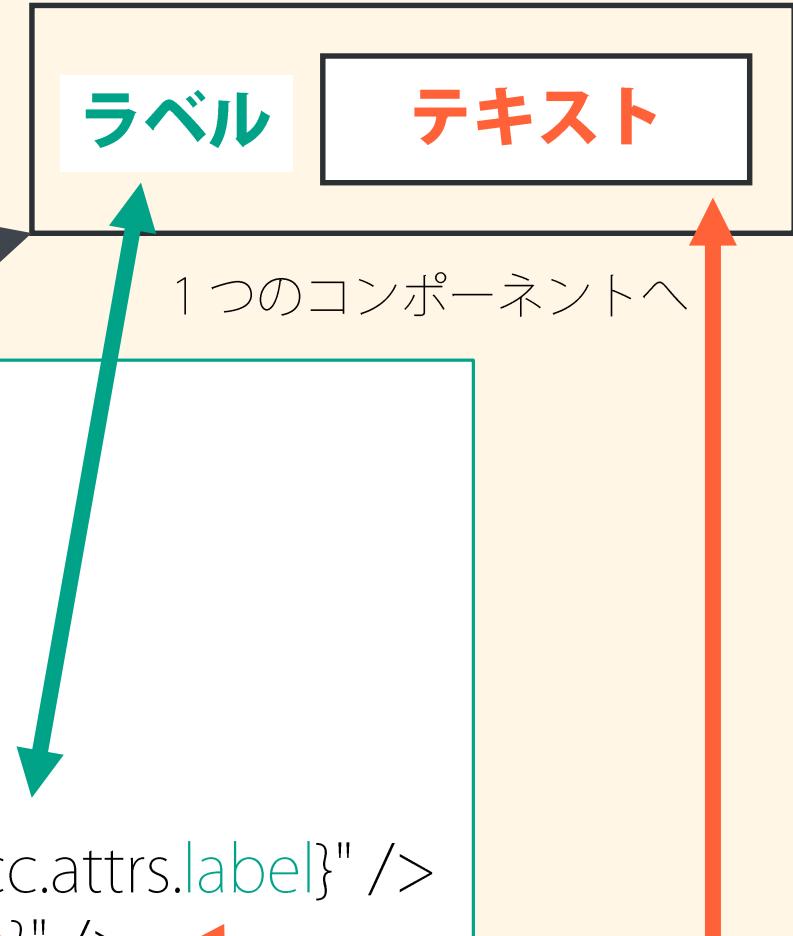
- 複数のコンポーネントを1つに束ねる
- 画面部品の共通化、再利用性を高める

1つに束ねたコンポーネントのインターフェース (属性)

```
<composite:interface>
    <composite:attribute name="label" />
    <composite:attribute name="value" />
</composite:interface>

<composite:implementation>
    <h:outputLabel id="lbl" for="txt" value="#{cc.attrs.label}" />
    <h:inputText id="txt" value="#{cc.attrs.value}" />
</composite:implementation>
```

複合コンポーネントの定義

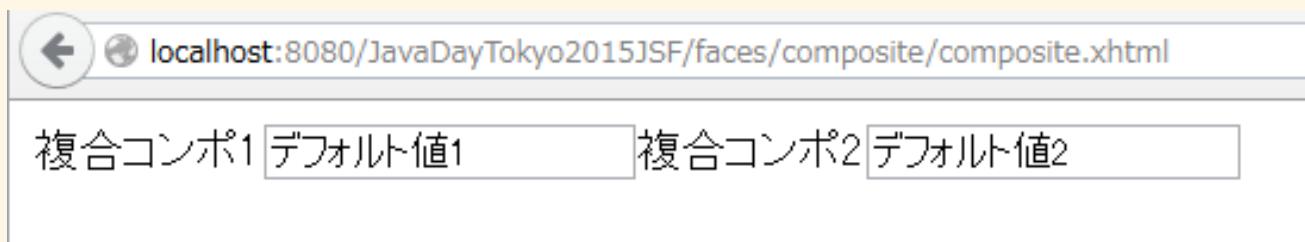


Composite Component (複合コンポーネント)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:mycomp="http://xmlns.jcp.org/jsf/composite/mycomp">

<h:form>
    <mycomp:labelText label="複合コンポ1" value="デフォルト値1"/>
    <mycomp:labelText label="複合コンポ2" value="デフォルト値2"/>
</h:form>
```

複合コンポーネントを利用する例



Custom Component

自作のコンポーネントを定義

@FacesComponent(createTag = true)

```
public class SimpleCustomComponent extends UIComponentBase{
```

```
    @Override  
    public String getFamily() {  
        return "simple.component";  
    }
```

```
    @Override  
    public void encodeBegin(FacesContext context) throws IOException{  
        String value = (String) getAttributes().get("value");  
        if(value != null){  
            ResponseWriter writer = context.getResponseWriter();  
            writer.write(value + "自作コンポーネント");  
        }  
    }  
}
```

カスタムコンポーネントの定義

デフォルトのnamespace
namespace属性で決めることも可能

カスタムコンポーネントを利用する

xmlns:s="http://xmlns.jcp.org/jsf/component"

```
<s:simpleCustomComponent value="シンプル" />
```

localhost:8080/JavaDayTokyo20

シンプル 自作コンポーネント

tagName属性で指定しない場合は
先頭小文字のクラス名

Validation / Converter

- JSF標準のバリデータ、コンバータ
- 自分で定義するカスタムのバリデータ、コンバータ
- Bean Validation アノテーションによる制約記述

@NotNull(message = "入力必須です")

```
private String 必須;
```

@Size(max=5, message = "文字数オーバー")

```
private String 文字数;
```

@Pattern(regexp = "[a-zA-Z0-9]*", message = "英数字のみ")

```
private String 正規表現;
```

The screenshot shows a browser window at the URL `localhost:8080/JavaDayTokyo/faces/validation/validation`. The page displays validation messages for three input fields:

- 必須: 入力必須です
- 文字数5文字まで: 文字数オーバー
- 英数字のみ: 英数字のみ

The input fields are labeled and filled as follows:

- 必須:
- 文字数5文字まで:
- 英数字のみ:

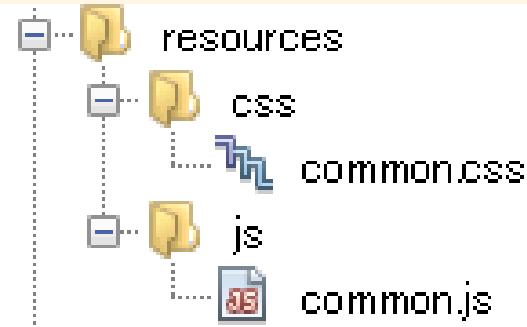
A send button is visible below the inputs.

自分で定義するカスタムバリデーターションも

Resources - JavaScript,CSS -

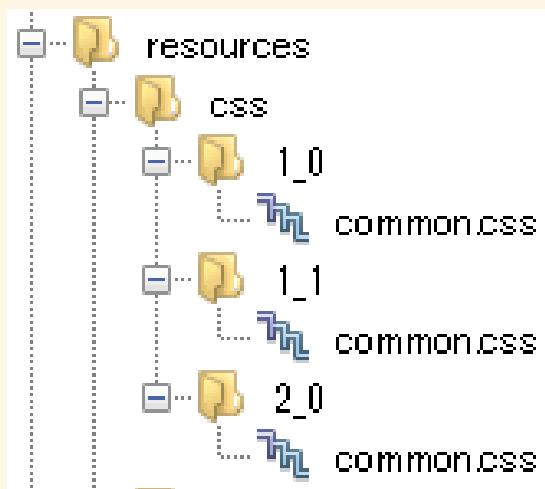
CSSの利用

```
<h:outputStylesheet library="css" name="common.css" />
```



JavaScriptの利用

```
<h:outputScript library="js" name="common.js" />
```

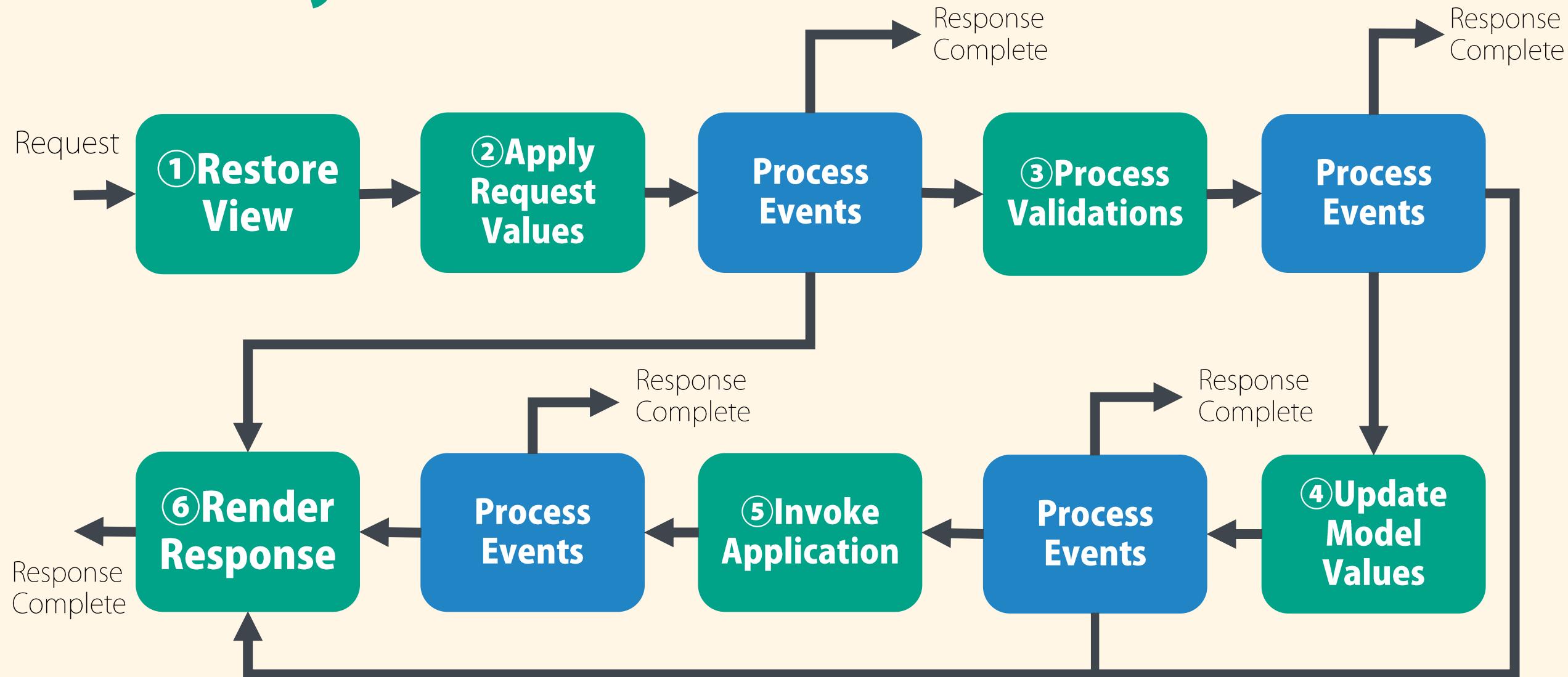


1_0、1_1のように名前付けしてバージョニング可能

JSF 2.2では**Resource Library Contract**によって
リソース単位にCSSやテンプレートの切り替えが可能

Java EE 7のJSF2.2で「Resource Library Contracts」チュートリアルをやってみました
<http://kikutaro777.hatenablog.com/entry/2013/08/03/212314>

LifeCycle PhaseListener



LifeCycle PhaseListener

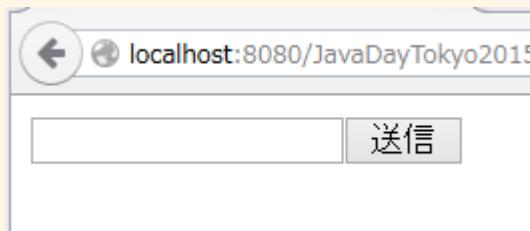
```
public class MyPhaseListener implements PhaseListener{  
    @Override  
    public void afterPhase(PhaseEvent event) {  
        System.out.println(event.getPhaseId() + "の後");  
    }  
  
    @Override  
    public void beforePhase(PhaseEvent event) {  
        System.out.println(event.getPhaseId() + "の前");  
    }  
  
    @Override  
    public PhaseId getPhaseId() {  
        return PhaseId.ANY_PHASE;  
    }  
}
```

普段あまり強く意識しなくても開発はできますが、図を理解するために、一度はPhaseListenerを実装して、各フェーズを確認してみるとよいかも

```
faces-config.xml  
<lifecycle>  
    <phase-listener>  
        my.app.javadaytokyo.phase.MyPhaseListener  
    </phase-listener>  
</lifecycle>
```

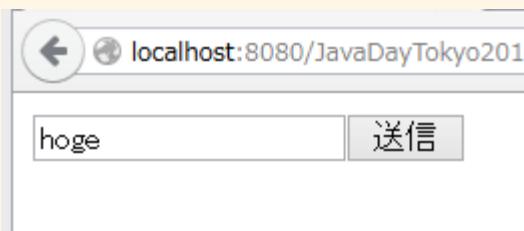
LifeCycle PhaseListener

画面の初期表示



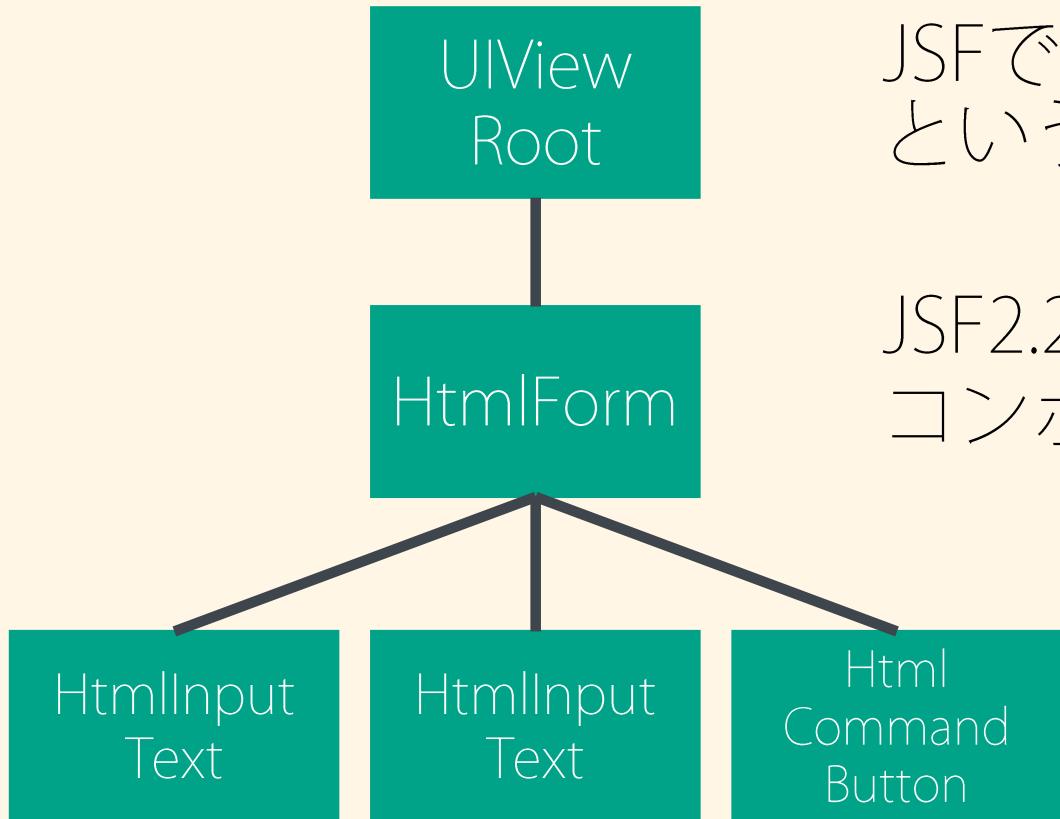
情報: RESTORE_VIEW 1の前
情報: RESTORE_VIEW 1の後
情報: RENDER_RESPONSE 6の前
情報: Static初期化
情報: 管理Bean初期化
情報: Getter
情報: RENDER_RESPONSE 6の後

値を入れて送信ボタン



情報: RESTORE_VIEW 1の前
情報: RESTORE_VIEW 1の後
情報: APPLY_REQUEST_VALUES 2の前
情報: APPLY_REQUEST_VALUES 2の後
情報: PROCESS_VALIDATIONS 3の前
情報: Getter
情報: PROCESS_VALIDATIONS 3の後
情報: UPDATE_MODEL_VALUES 4の前
情報: Setter
情報: UPDATE_MODEL_VALUES 4の後
情報:(INVOKE_APPLICATION 5の前
情報: ActionListenerから呼び出し
情報: Actionから呼び出し
情報:(INVOKE_APPLICATION 5の後
情報: RENDER_RESPONSE 6の前
情報: Getter
情報: RENDER_RESPONSE 6の後

コンポーネントツリー



JSFではサーバサイドでコンポーネントツリーという形で画面情報をサーバサイドで保持

JSF2.2ではStateless Viewsという機能も入り、コンポーネントツリーを保持しない選択も

`<f:view transient = "true">`

私自身まだ実際には試したことがないので是非性能比較などレポートを(^^;

コンポーネントツリーを簡単に確認する方法はJSF Tipsの部分で

JSFとは？

始め方

基本機能

JSF

JavaServer Faces

開發現場での
活用

PrimeFaces

JSF Tips

自分の開発現場

- ・受託開発案件(隣の室ではパッケージも)
- ・ウォーターフォール
- ・Webデザイナさんはいない
- ・JavaScript(ちゃんと)書けるメンバがいない
- ・元々.NETのWindowsForm経験者が多い

自分の開発現場



デモ画面を急いで
作って欲しい

社内営業担当

デザインや操作感が
思ってたのと違う…



開発者



お客様

業務系システム
画面中心になることが多い

営業 ▶ 受注 ▶ 要件定義 ▶ 基本/詳細設計 ▶ 開発/テスト ▶ 納品…

Excelの画面だと
イメージしにくい…



お客様



社内SE

どういう画面なら
作れるのか？

最近のWebサイトは
色々リッチですよね

開発効率あげる
準備を早めにしたい

技術的不安は早めに
払拭したい



開発者

自分の開発現場



簡単なプロト作成は
コンポ並べてㄌㄌと

社内営業担当

プロトベースに
打合せ



開発者



お客様

業務系システム
画面中心になることが多い

見て触れるコンポーネントを揃えやすく、細設計段階から土台として共有

プロトタイプ画面で
イメージ共有



お客様



社内SE

ShowCaseで
共有

リッチコンポの
サードパーティ利用

複合コンポや
カスタムコンポ準備

サードパーティ等
早めに検証



開発者

これから

jQuery UIやBootstrapを利用したり、最近ではWeb上で簡単にモックアップ・プロトタイプが作れるサービスも多々あります

JSFではプロトで作ったものを、そのまま本開発で使うことも可能ただし、プロトレベルはサクサク作れるけど、実開発となると細かい所で苦戦することも(主にサードパーティ利用時)

JSFを3年ほど触ってきて、チームメンバのスキルも高いため、暫くはJSFで行く予定ですが、今後のWeb動向も気にしつつJavaScriptなどもしっかり学んでいきたい所

JSFとは？

始め方

基本機能

JSF

JavaServer Faces

開発現場での
活用

PrimeFaces

JSF Tips

PrimeFacesとは？

JSF向けにリッチコンポーネントを揃えたライブラリ



100種類以上の
コンポーネント

Apache
License
Version 2.0

ドキュメント
500ページ超

クロスブラウザ

商用サポート選択可
Elite/Pro

Extension

Mobile



PrimeFaces Cookbook
Over 90 practical recipes to learn PrimeFaces – for rapidly evolving, leading JSF component suite



PrimeFaces Starter
Design and develop responsive web user interfaces for the desktop and mobile devices with PrimeFaces and its many practical, flexible and easy-to-use components



PrimeFaces
Get your JSF-based projects up and running with this easy-to-implement guide on PrimeFaces



PrimeFaces
Blueprints
Create your very own portfolio of customized web applications with PrimeFaces



Learning PrimeFaces
Extensions Development
Develop advanced frontend applications using PrimeFaces Extensions components and plugins



Basic Component

入力

AutoComplete

InputMask

InputText

InputTextarea

Password



選択肢

SelectOneMenu

Checkbox

Radio



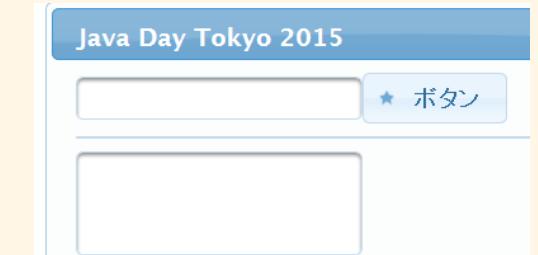
パネル

Accordion

Panel

Fieldset

TabView

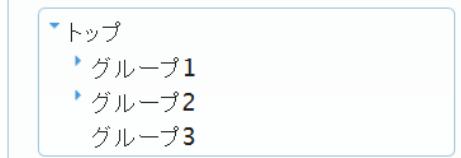


データ

DataGrid

DataTable

Tree



メニュー

Menu

Menubar

MegaMenu



ボタン・リンク

CommandButton

CommandLink

メッセージ

Messages

Basic Component

DataTableはかなり多機能

The screenshot shows a 'Employee List' (社員一覧) table with the following data:

名前	年齢	所属組織	メール
西野七瀬	26	Sweet Power	makimaki@love.cute.com
深川麻衣	24	乃木坂46	@nogizaka46
秋元真夏	21	乃木坂46	mamai@nogizaka46
橋本奈々未	22	乃木坂46	nanamin@nogizaka46
山本舞香	18	乃木坂46	nanase@nogizaka46
	21		

Callout bubbles highlight various features:

- ソート (Sort): Points to the column headers.
- フィルタ (Filter): Points to the search input field.
- 行固定 (Row Fix): Points to the first row.
- カラム固定 (Column Fix): Points to the first column.
- ページング (Paging): Points to the page navigation buttons.
- ヘッダ (Header): Points to the top header bar.
- 縦横スクロール (Vertical/Horizontal Scroll): Points to the scroll bars.
- 行展開(Expand Row): Points to the expanded row for 山本舞香.
- その他の... (More...): Points to the ellipsis button.
- フッタ (Footer): Points to the bottom footer bar.
- 行選択 (Single/Multi): Points to the selection checkboxes in the context menu.
- 表示カラム選択 (Column Selection): Points to the column selection dropdown.
- Drag&Drop 行の入れ替え (Drag&Drop Row Swap): Points to the row being moved.
- Drag&Drop カラム入れ替え (Drag&Drop Column Swap): Points to the column being moved.

Modern

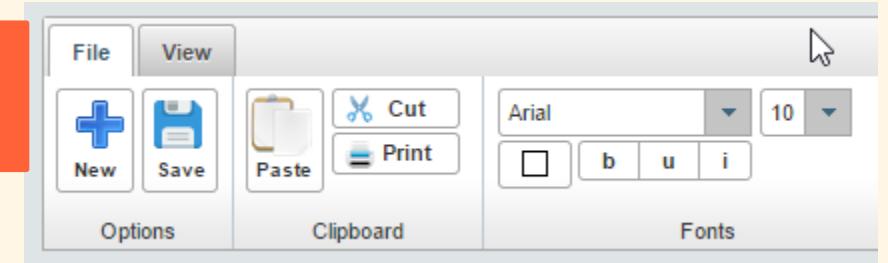
ContentFlow

ContextMenu

Ribbon



Drag & Drop



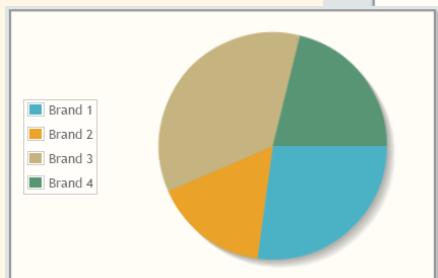
Growl

BlockUI

frm:j_idt16: 検証エラー:
値が必要です。

Barcode/QR

Graph



Rating



OverlayPanel



Dialog Framework

単純なYes/NoダイアログはConfirm Dialogコンポで作成可能

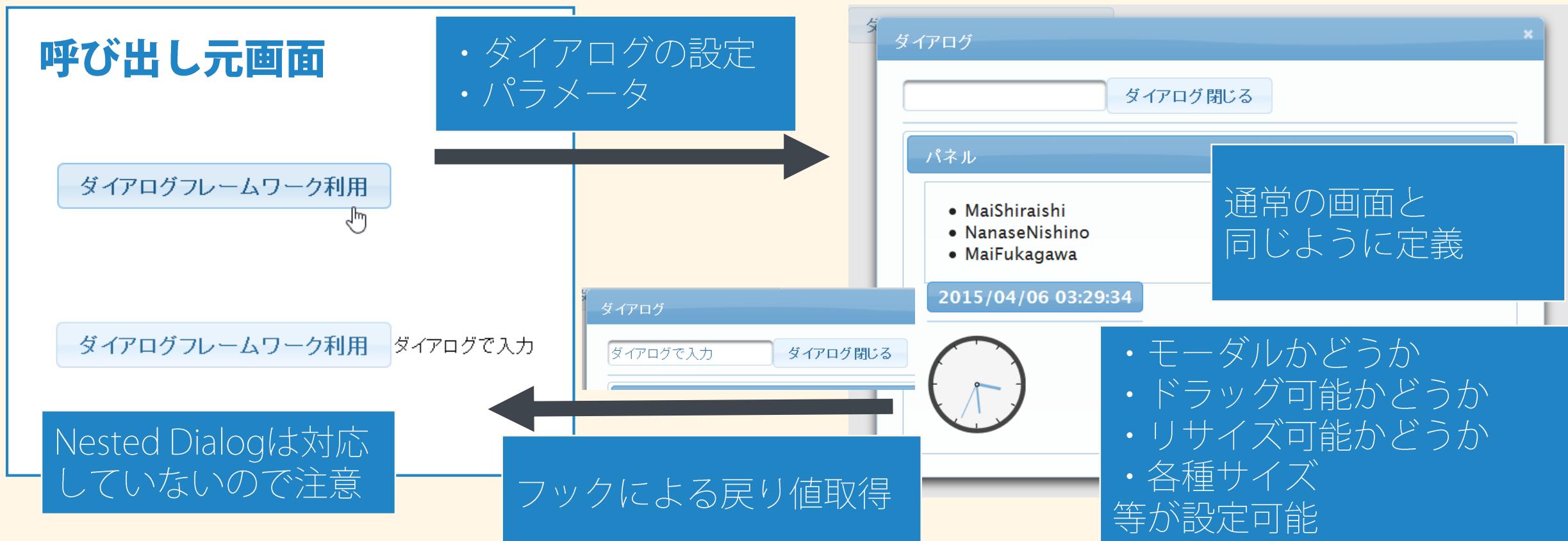
```
<p:confirmDialog message="ConfirmDialogコンポを利用しています"  
header="ConfirmDialogコンポ" severity="info" widgetVar="dlg">  
    <p:commandButton value="はい" oncomplete="PF('dlg').hide()" />  
    <p:commandButton value="いいえ" oncomplete="PF('dlg').hide()" />  
</p:confirmDialog>
```



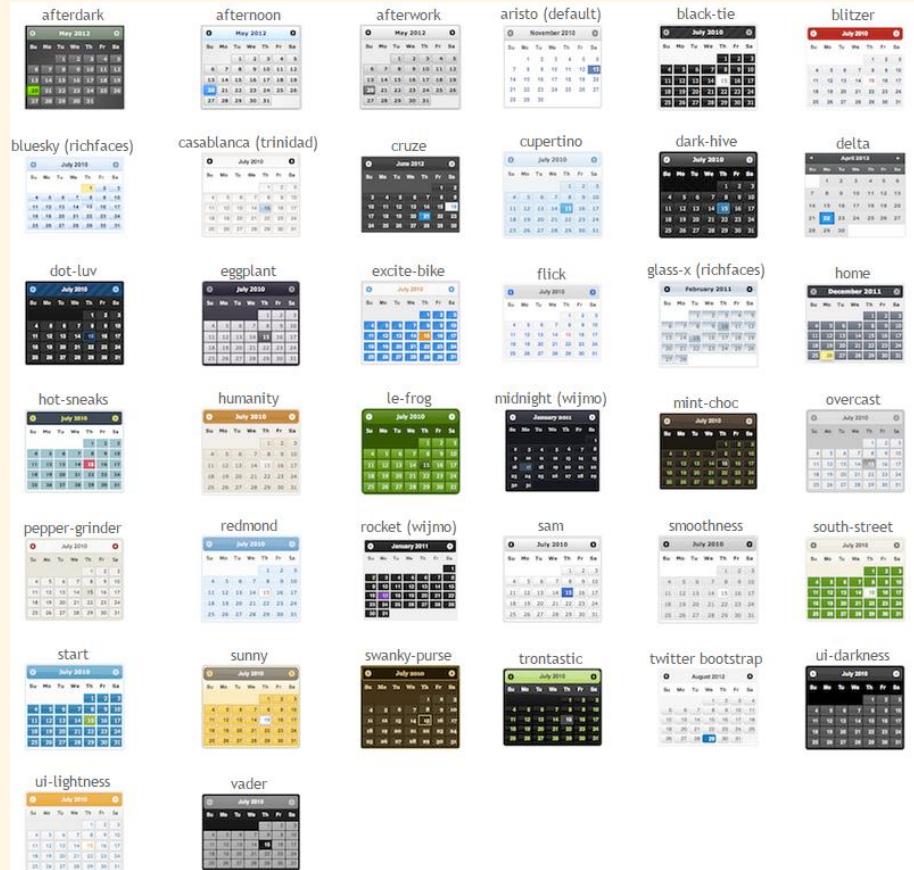
もう少し複雑なダイアログはDialog Frameworkが便利

Dialog Framework

- Dialogを通常の画面同様に定義(xhtml, 管理Bean)
- ダイアログの呼び出し元画面とデータのやりとり



Theme



Community Theme 30 以上

有償のTheme
• レスポンシブ
• HTML5 & CSS3

The image shows the SentinelTheme website, which offers over 100 JSF components. It also displays the SPARK dashboard, a PrimeFaces application, running on a desktop monitor and two mobile devices (an iPhone and a tablet). The SPARK dashboard features a network graph, a chart, and various data tables and reports.

jQuery UIのThemeRoller
を利用した自作も可能

The image shows the ThemeRoller interface, a tool for creating custom jQuery UI themes. It includes sections for Font Settings, Corner Radius, Header/Toolbar, Content, and various theme-specific options like Clickable states and Highlight.

良い所

- JavaScriptやCSSを定義することなく手軽にリッチな画面作成が可能
- オープンソースなので中身は確認可能
- 商用サービスも選べる（ただし英語）
- 情報が多くある（ただし英語）
- Cagatay CiviciさんはJSFのExpert Group



PrimeFaces Community Forum

28011 Topics
113496 Posts

注意点

- 上手く動かなかった場合の原因調査や回避方法の検討に苦労する
- prime technologyというトルコの企業（Javaのコンサルやトレーニング等）が開発しているが、どこまで継続されるかは正直わからない
- 情報が多くある=ハマることもある…かも

JSFとは？

始め方

基本機能

JSF

JavaServer Faces

開発現場での
活用

PrimeFaces

JSF Tips

画面遷移でパラメータの受け渡し

画面A



画面B



Session利用

Getパラメータ

Flashオブジェクト
利用(おススメ)

画面遷移でパラメータの受け渡し

画面A Bean

```
@Inject  
private Parameter param;
```

画面A



画面B



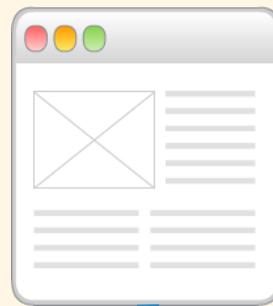
画面B Bean

```
@Inject  
private Parameter param;
```

```
@Named(value = "parameter")  
@SessionScoped  
public class Parameter implements Serializable {  
    @Getter @Setter  
    private String value;  
}
```

画面遷移でパラメータの受け渡し

画面A



画面B



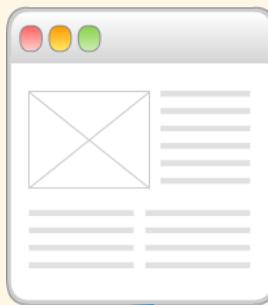
Flashオブジェクト利用(EL式)

```
<h:inputText value="#{flash.value}" />
```

```
<h:outputText value="#{flash.value}" />
```

画面遷移でパラメータの受け渡し

画面A



画面B



Flashオブジェクト利用(コード)

```
FacesContext.getCurrentInstance()  
.getExternalContext().getFlash()  
.put("parameter", value);
```

```
FacesContext.getCurrentInstance()  
.getExternalContext().getFlash()  
.get("parameter").toString();
```

actionとactionListener

- CommandButtonやCommandLinkが持つ属性
- 呼び出される順序は**actionListener → action**

actionListener

特に画面遷移を伴わず、処理を実行する際
画面遷移する前に処理や判定を行いたい場合

action

主に画面遷移を伴う処理
単に処理を実行する場合も利用可能

web.xml

ビューのコメントを出力しない

```
<context-param>
  <param-name>javax.faces.FACELETS_SKIP_COMMENTS</param-name>
  <param-value>true</param-value>
</context-param>
```

ページの変更チェック 本番環境では refresh -1

```
<context-param>
  <param-name>javax.faces.FACELETS_REFRESH_PERIOD</param-name>
  <param-value>-1</param-value>
</context-param>
```

Project Stage

State Saving Method

Response Buffer

NumbersOfViewsInSession

NumberOfLogicalViews

…etc 実装やライブラリ依存も

暗黙的なオブジェクト

#{}application}

#{}applicationScope}

#{}component}

#{}cc}

#{}cookie}

#{}facesContext}

#{}flash}

#{}flowScope}

#{}header}

#{}headerValues}

#{}initParam}

#{}param}

#{}paramValues}

#{}request}

#{}resource}

#{}session}

#{}sessionScope}

#{}view}

#{}viewScope}

エラーハンドリング

web.xmlによるハンドリング

```
<error-page>
  <error-code>404</error-code>
  <location>/error/notFound.xhtml</location>
</error-page>
<error-page>
  <error-code>500</error-code>
  <location>/error/error.xhtml</location>
</error-page>
<error-page>
  <exception-type>java.lang.Exception</exception-type>
  <location>/error/error.xhtml</location>
</error-page>
```

ExceptionHandlerの利用

ExceptionHandlerWrapperを継承して処理を実装

これら辺に関しては以下サイトで非常に綺麗にまとめています

JSF2.0のエラーハンドリング
<http://n-agetsuma.hatenablog.com/entry/2013/02/11/134531>

Security

HTMLエスケープ(サニタイジング)

escape属性(デフォルトでtrue)

< → <

> → >

& → &

<h:inputText escape="true" />

CSRF(クロスサイトリクエストフォージィング)

faces-config.xml

```
<protected-views>
  <url-pattern>result.xhtml</url-pattern>
</protected-views>
```

/result.xhtml?javax.faces.Token=1427748820838&user=kikuta

An Error Occurred:

javax.faces.application.ProtectionViewException

Test



モックの利用

Arquillian Extension
warpやdroneなど

管理BeanはPOJOなのでテストしやすい！

とは聞くものの、実際には難しい面も

- FacesContext
- Inject

テストしやすい作りを意識した実装を



負荷試験 ViewStateなどに工夫が必要

JMeterでJSFの負荷試験にチャレンジ
<http://kikutaro777.hatenablog.com/entry/2013/05/04/151951>



OmniFaces

To make JSF life easier

JSF開発の便利ライブラリ

FacesContextを扱いやすくするFacesクラス

before

```
if(FacesContext.getCurrentInstance() != null)
```

after

```
if(Faces.hasContext())
```

before

```
FacesContext.getCurrentInstance()  
    .getExternalContext().getInitParameter("hoge");
```

after

```
Faces.getInitParameter("hoge");
```

OmniFaces

リソースを一括りにしてリクエスト回数を減らす
CombinedResourceHandler

```
<resource-handler>  
    org.omnifaces.resourcehandler.CombinedResourceHandler  
</resource-handler>
```

基本的には設定追加のみなので
非常に手軽に実現できる

HTTPレスポンスの圧縮フィルタ
GzipFilter

```
<filter>  
    <filter-name>gzipResponseFilter</filter-name>  
    <filter-class>  
        org.omnifaces.filter.GzipResponseFilter  
    </filter-class>  
</filter>
```

コードから生成

Windows Formのようにコードによる生成も…

これだけで実装してしまうとFaceletsが意味をなさず、生成されるビューもイメージしづらいので個人的には奥の手

```
HtmlOutputLabel label = new HtmlOutputLabel();
label.setValue("コードから生成したラベル");
HtmlInputText text = new HtmlInputText();
HtmlCommandButton btn = new HtmlCommandButton();
btn.setValue("コードから生成したボタン");
List<UIComponent> listComp
    =FacesContext.getCurrentInstance().getViewRoot().getChildren();
listComp.add(label);
listComp.add(text);
listComp.add(btn);
```

Debug

<ui:debug>というタグをページのどこかに定義

Ctrl + Shift + dで以下ページが表示され、コンポーネントツリーなどを簡単に確認可能

Debug Output

/componenttree/componenttree.xhtml

+ Component Tree

+ Scoped Variables

+ View State

2015/04/07 22:41:49 - Generated by Mo

dのキーを変更したい場合は**hotkey**

属性で変更したいキーを指定

<ui:debug **hotkey = "i"** />

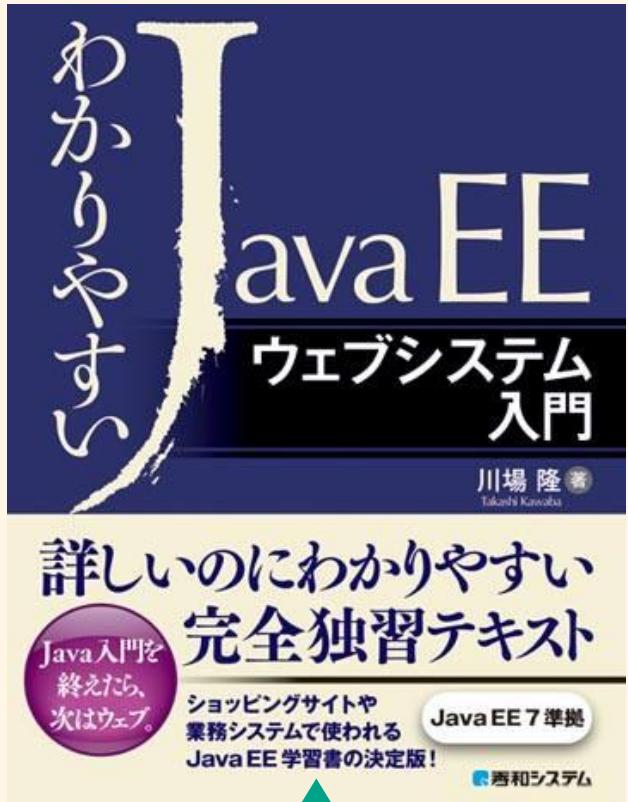
- Component Tree

```
<UIViewRoot id="j_id1" inView="true" locale="ja_JP" renderKitId="HTML_BASIC" rendered="true"
transient="false" viewId="/componenttree/componenttree.xhtml">
<html xmlns="http://www.w3.org/1999/xhtml">
<UIOutput id="j_idt2" inView="true" rendered="true" transient="false">
<title>Facelet Title</title>
</UIOutput>
<HtmlBody id="j_idt4" inView="true" rendered="true" transient="false">
<UIDebug hotkey="D" id="j_idt5" inView="true" rendered="true" transient="true"/>
<HtmlForm enctype="application/x-www-form-urlencoded" id="frm" inView="true"
prependId="true" rendered="true" submitted="false" transient="false">
```

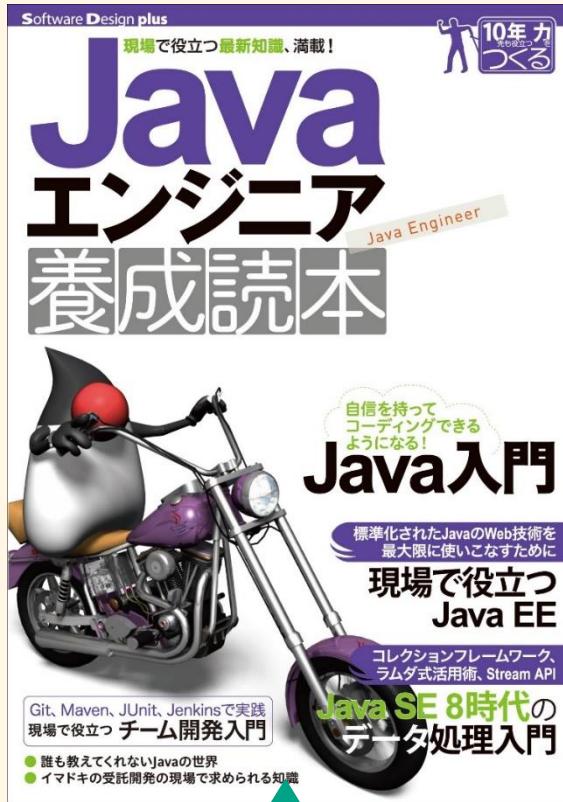
気を付ける所

- ・ 大量のコンポーネントを1画面に配置しない
→コンポーネントツリーのサイズが大きくなってしまう
- ・ EL式で時間がかかる処理のロジックを直接バインドする際は要注意
→レンダリングの際に大量に呼ばれてないか確認
- ・ なんでもかんでもリッチコンポ、ではなく、使い分けも

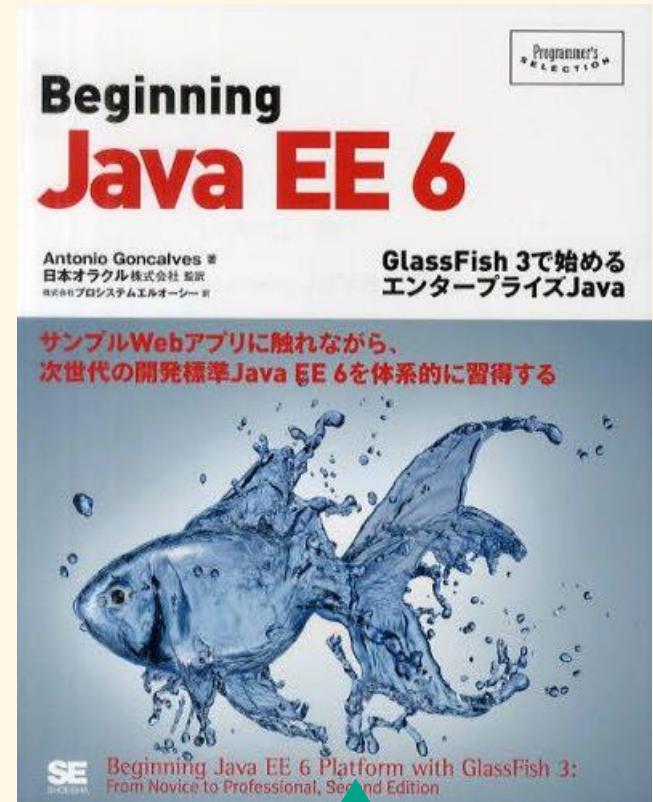
Book(日本語)



JSFの詳細解説あり
Java EE初心者にオススメ

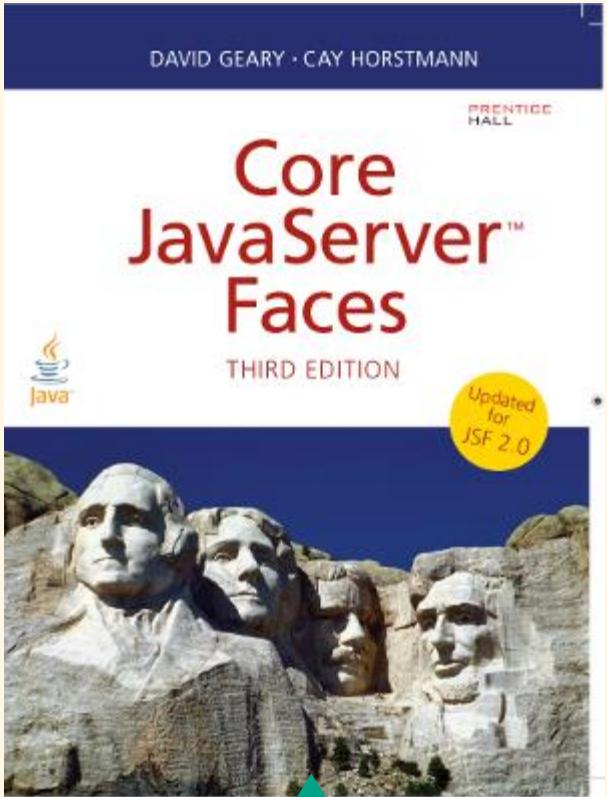


Java EE簡易チュートリアルあり
他のJavaパートも有益！

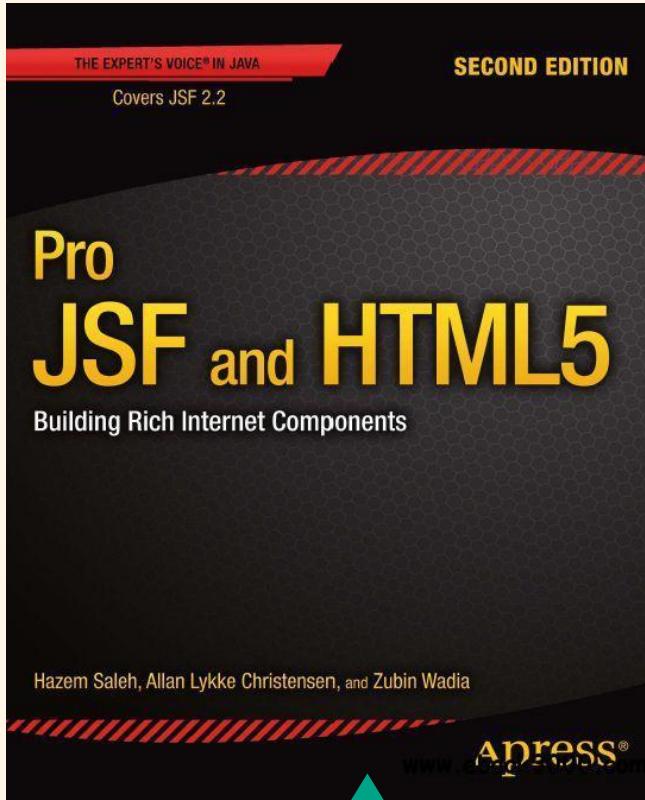


Java EE 6ですが非常に
読み応えある1冊

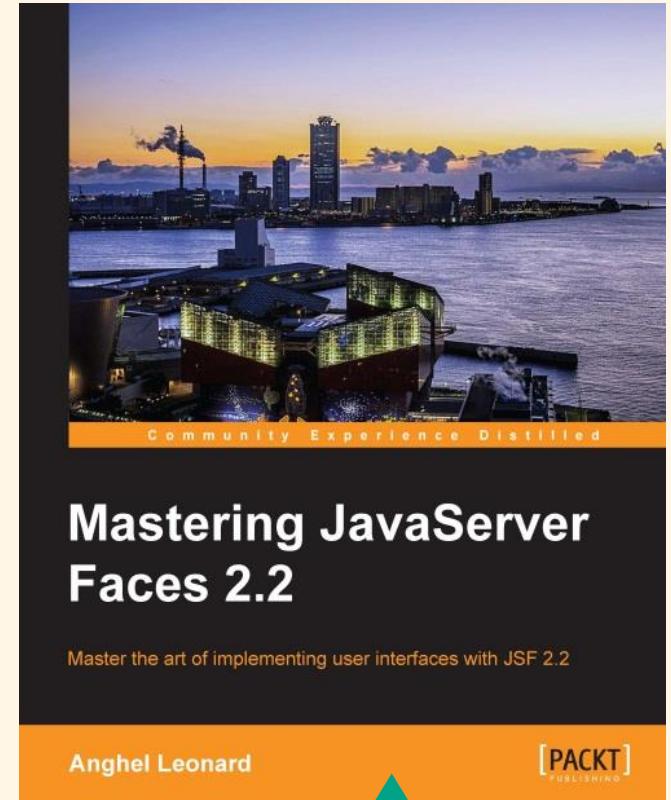
Book(英語)



JSF2.0ですが実用的な
内容が多く挿入



JSF2.2を含み,JSF設計の
勘所なども充実



JSF2.2に対応した
書籍

今日のサンプル

全ての例を網羅してはいませんが、GitHubに本日のサンプルコードを置きましたので、是非NetBeansを入れて色々試してみてください！

<https://github.com/kikutaro/JavaDayTokyo2015JSF>

Java EE開発におけるJSFの 活用について

ご清聴ありがとうございました

株式会社構造計画研究所
製造ビジネス・ソリューション部
菊田 洋一