

Java EE 7 アプリケーション設計ガイド

- JSF(JavaServer Faces) 2.2 詳細編

日本アイ・ビー・エム システムズ・エンジニアリング株式会社

Disclaimer

- この資料は日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム システムズ・エンジニアリング株式会社の正式なレビューを受けておりません。
- 当資料は、資料内で説明されている製品の仕様を保証するものではありません。
- 資料の内容には正確を期するよう注意しておりますが、この資料の内容は2015年10月現在の情報であり、製品の新しいリリース、PTFなどによって動作、仕様が変わる可能性があるのでご注意ください。
- 今後国内で提供されるリリース情報は、対応する発表レターなどでご確認ください。
- I B M、I B Mロゴおよびibm.comは、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。他の製品名およびサービス名等は、それぞれ I B Mまたは各社の商標である場合があります。現時点での I B Mの商標リストについては、www.ibm.com/legal/copytrade.shtmlをご覧ください。
- 当資料をコピー等で複製することは、日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム システムズ・エンジニアリング株式会社の承諾なしではできません。
- 当資料に記載された製品名または会社名はそれぞれの各社の商標または登録商標です。
- JavaおよびすべてのJava関連の商標およびロゴは Oracleやその関連会社の米国およびその他の国における商標または登録商標です。
- Microsoft, Windows および Windowsロゴは、Microsoft Corporationの米国およびその他の国における商標です。
- Linuxは、Linus Torvaldsの米国およびその他の国における登録商標です。
- UNIXはThe Open Groupの米国およびその他の国における登録商標です。

目次

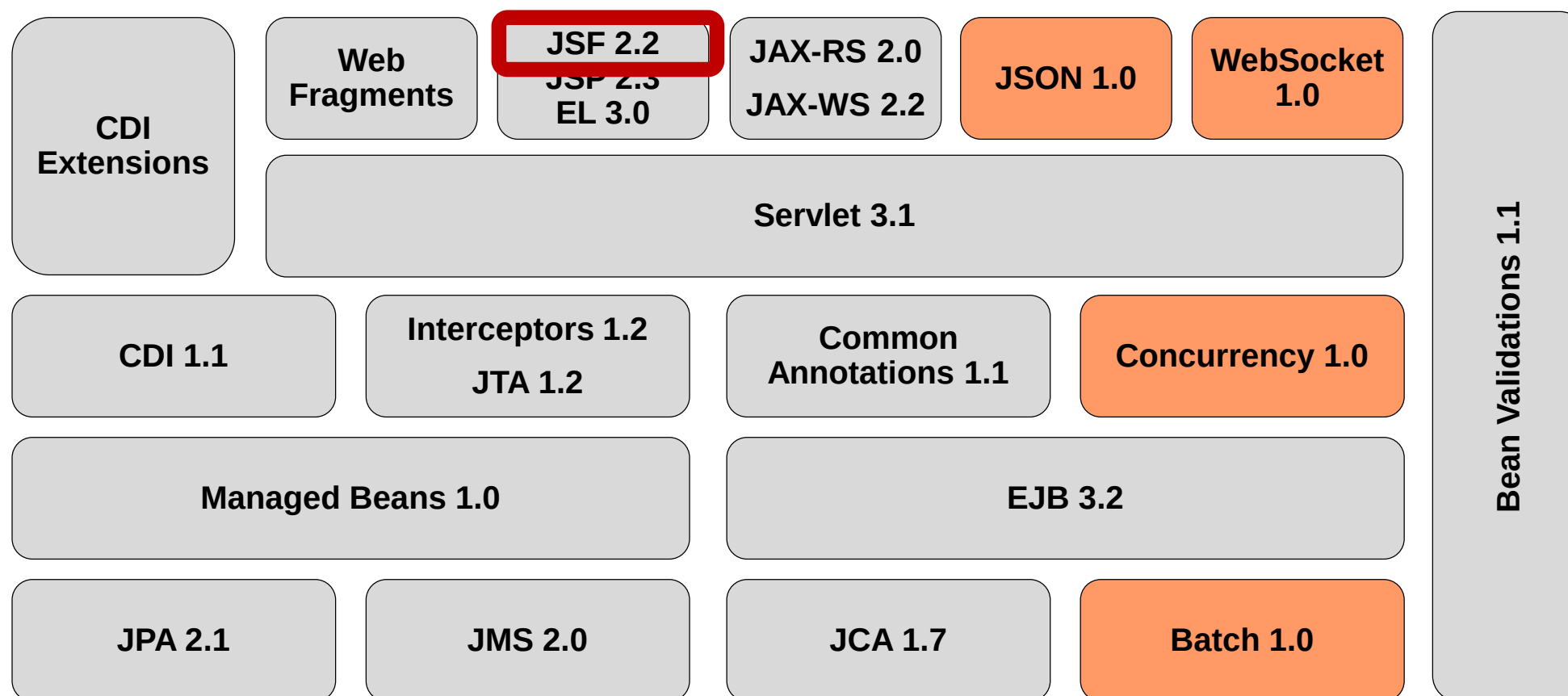
1. はじめに
2. 画面遷移
 1. Flash
 2. ConversationScoped
 3. ViewScoped(Big Ticket Feature)
3. Ajax
 1. Ajaxについて
 2. キュー制御
4. ファイル・アップロード
 1. ファイル・アップロード
 2. Ajaxによるファイル・アップロード
5. HTML5(Big Ticket Feature)
 1. パススルーアトリビュート
 2. パススルーエレメント
6. ViewActions
7. ステートレスビュー(Big Ticket Feature)
8. Faces Flow
9. Resource(Big Ticket Feature)
10. まとめ

1. はじめに



Java EE 7 に含まれる仕様 (JSR一覧)

- Java EE 7 では、4つの仕様が新規に追加され、3つの仕様がバージョンアップされている
- 当ガイドでは、Java EE7で機能拡張されたJSF 2.2を対象とする。



WebSphere Application Serverのサポート状況 (2015年10月現在)

- Liberty Profile
 - WAS V8.5.5.6でJSF2.2対応

- Full Profile (従来のWebSphere Application Server)
 - JSF2.0対応

2. 画面遷移

2-1.Flash

2-2.ConversationScoped

2-3.ViewScoped



JSFの画面遷移

- JSFの画面遷移は、Post-Redirect-Getによって画面遷移を実装することを推奨する。
- 画面遷移する際にリダイレクトさせない場合、表示される画面とURLに表示されるFaceletsが対応していない。リダイレクトを実装することでURLが実装した各Faceletsと対応したURLに切り替わる。
- リダイレクトの実装方法は、遷移先画面のページ名の後に"画面遷移先のページ+?faces-redirect=true"を付与する。（以下の例を参照）

管理Bean

```
...  
public String nextpage() {  
  
    return "nextpage?faces-redirect=true";  
}  
...
```

戻り値に遷移先ページは直接指定しない。
画面遷移先のページ+?faces-redirect=true"
指定してクライアント経由でリダイレクトさせる。

JSF管理BeanとCDI管理Beanのパッケージ一覧

- beanのスコープを定義することで、bean をコンテキストに関連付け、bean のライフサイクルを管理する。
- JSF管理Beanは2.2より後のリリースで廃止される予定になっているので、今後はJavaEEの標準であるCDI管理Beanを使用することを推奨する。
- CDIの詳細については、「Java EE7アプリケーション設計ガイド – CDI1.2編」を参照。

JSF管理Bean	CDI管理Bean
@javax.faces.bean.RequestScoped	@javax.enterprise.context. RequestScoped
(NA)	@javax.enterprise.context. ConversationScoped
@javax.faces.bean.SessionScoped	@javax.enterprise.context. SessionScoped
@javax.faces.bean.ApplicationScoped	@javax.enterprise.context.ApplicationScoped
(NA)	@javax.faces.flow.FlowScoped
@javax.faces.bean.ViewScoped	@javax.faces.view.ViewScoped

CDI管理Beanアノテーションのスコープ

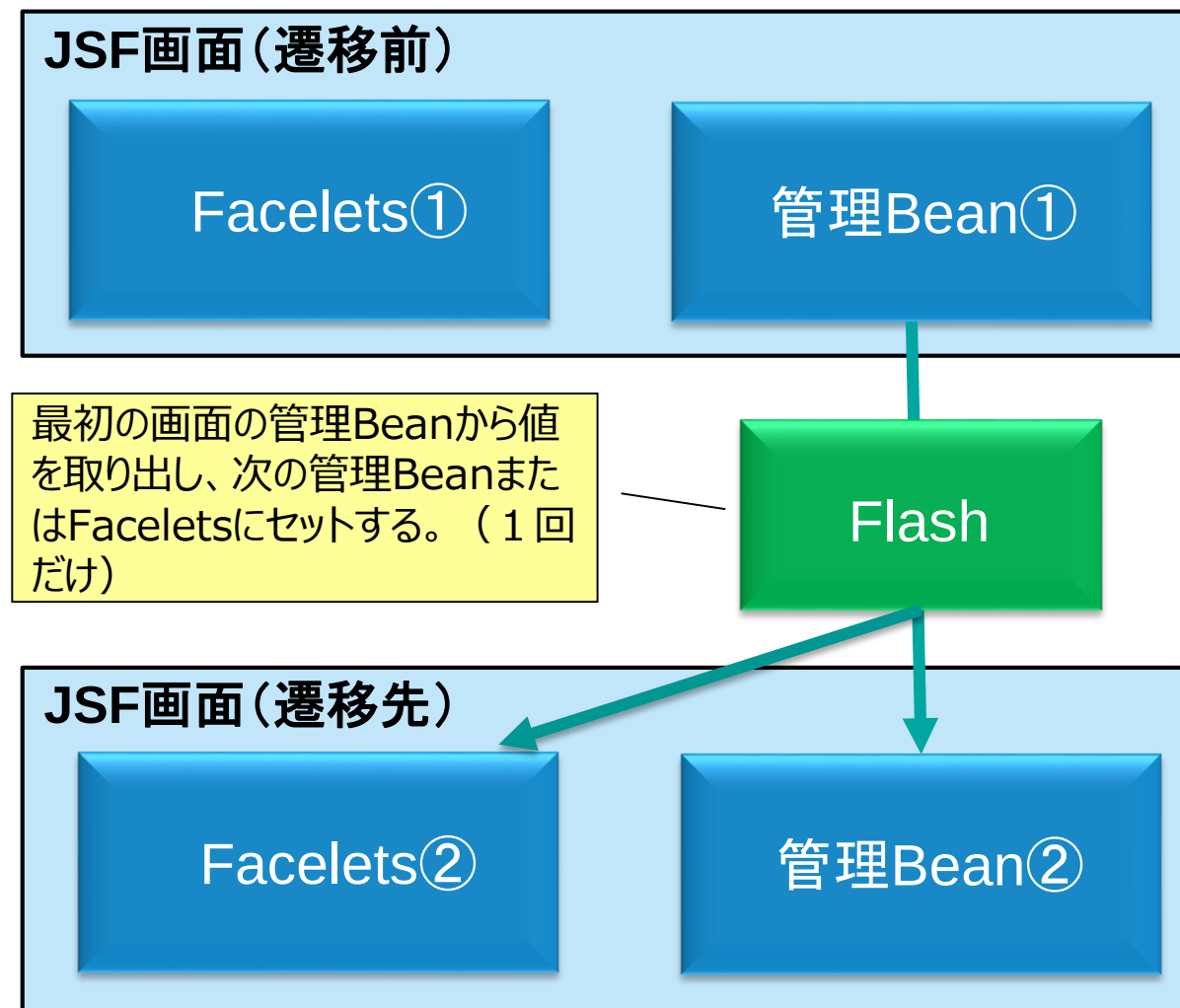
- CD管理Beanでは以下の6つがサポートされている。FlowScopedとViewScopedはJSF2.2から追加された。

CDI管理Beanアノテーション	スコープ
@javax.enterprise.context. RequestScoped	• 1回のHTTPリクエストの範囲
@javax.enterprise.context. ConversationScoped	• 複数回の HTTP リクエストの範囲 • 開発者が開始と終了を制御
@javax.enterprise.context. SessionScoped	• 1 つの HTTP セッションの範囲
@javax.enterprise.context.ApplicationScoped	• 1 つのアプリケーションの範囲
@javax.faces.flow.FlowScoped (JSF 2.2で追加)	• 1 つのフローの範囲
@javax.faces.view.ViewScoped (JSF 2.2で追加)	• 1 つのビューの範囲

2-1.Flash

Flashによる画面遷移の仕組み

- JSF で画面間のデータを受け渡す方法の一つとして Flash オブジェクトがある。Flashとは画面遷移の1度きりだけ有効になるマップのこと。
- 右図を参照
 - JSF画面（遷移前）の管理Bean①でこのFlash オブジェクトにデータをセットする。JSF画面（遷移先）の管理Bean②またはFacelets②では、画面が表示される前にFlashオブジェクトにアクセスし、Flash オブジェクトからデータを取得し、管理Beanのフィールドにセットする。
- 画面遷移をリダイレクトで実装した場合は、@RequestScopedではデータの受け渡しはできないが、@ConversationScopedや@SessionScopedにするほどではないときにFlashが有効。



Flash オブジェクトの使い方（１）

例① 最初の画面の管理BeanでFlashオブジェクトにデータをセットし、直接リダイレクト先に表示させる。

管理Bean①

```
public String next() {  
    Flash flash = FacesContext.getCurrentInstance().getExternalContext().getFlash();  
    flash.put("userID", user.getUserId());  
  
    return "nextpage?faces-redirect=true";  
}
```

次ページのレンダリングを行うのではなくリダイレクト先のURLをクライアントに返信する。

JSF2.2でFlashオブジェクトを利用するには、リダイレクト先に引き継ぎたいデータを以下のように設定する。
FacesContext.getCurrentInstance().getExternalContext().getFlash().put("フラッシュId",フラッシュデータ);

リダイレクト先のページにFlashスコープのデータを直接表示するには以下のように宣言する。
#{flash.フラッシュId}

Facelets②

```
<?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">  
    <head>  
        . . .  
    </head>  
    <body>  
        #{flash.userID}  
    </body>  
</html>
```

Flash オブジェクトの使い方（２）－１

例②

最初の画面の管理BeanでFlash
オブジェクトにデータをセットする。

管理Bean① (Bean1.java)

```
@Named
@RequestScoped
public class Bean1 implements Serializable{
    private String id;
    private String password;

    public String submit(){
        Flash flash = FacesContext.getCurrentInstance().getExternalContext().getFlash();
        flash.put("id", this.id);
        flash.put("password", this.password);

        return "view2?faces-redirect=true";
    }
}
```

1.リダイレクト先のURLをクライアントに送る

Facelets① (view1.xhtml)

```
...
<h:form>
    <h:panelGrid border="1" columns="2">
        id<h:inputText value="#{bean1.id}"></h:inputText>
        パスワード<h:inputText value="#{bean1.password}"></h:inputText>
        <h:commandButton value="キャンセル" action="#{bean1.goto_0}"></h:commandButton>
        <h:commandButton value="submit" action="#{bean1.submit}"></h:commandButton>
    </h:panelGrid>
</h:form>
...
```

2.Flash経由でデータを引き継ぐ

Flash オブジェクトの使い方（２） – ２

例② 続き

flash経由でデータを引き継いで画面に表示する。

3.Flashからデータを取り出す

管理Bean② (Bean2.java)

```
@Named
@RequestScoped

public class Bean2 implements Serializable{
    private String id;
    private String password;
```

@PostConstruct

```
public String load() {
    Flash flash = FacesContext.getCurrentInstance().getExternalContext().getFlash();
    this.id = (String) flash.get("id");
    this.password = (String) flash.get("password");
}
```

Facelets② (view2.xhtml)

```
...
<h:form>
    <h:panelGrid border="1" columns="2">
        名前<h:outputText value="#{bean2.id}"></h:outputText>
        住所<h:outputText value="#{bean2.password}"></h:outputText>
        <h:commandButton value="戻る" action="#{cS.goto_2}"></h:commandButton>
        <h:commandButton value="submit" action="#{cS.goto_0}"></h:commandButton>
    </h:panelGrid>
</h:form>
...
```

@PostConstruct
コンテナが管理Beanを初期化する時に実行させたいメソッドに付加する。

2-2. ConversationScoped

ConversationScopedとは

- @ConversationScopedはスコープの長さをカスタマイズすることができるスコープアノテーション。InjectしたConversationのオブジェクトに対してスコープの開始と終了を指示する。@RequestScoped より長く @SessionScoped よりも短く定義したい場合に有効。
- 例えば、ウェブで買い物をする場合、①商品を買物かごに入れる、②送り先などの情報を入力、③確認画面の表示、④注文を確定する、の流れが一般的である。この間、注文情報と利用者情報を保持し続ける必要がある。@SessionScopedでも実装することは可能だが、@ConversationScopedを使うことでスコープの開始と終了を@SessionScopedより短く実装することができる。
- @ConversationScopedを指定しても、開始の指示をするまでは@RequestScopedと同じ働きになる。
- ブラウザのタブ毎にConversationを制御することが可能。

@ConversationScopedの使用例

■ サンプルアプリケーション

– スタート画面の"スタート"ボタンをクリックしたときに、@ConversationScopedを開始する。画面①、画面②で値を入力し、画面③で入力値を確認する。画面③で"購入する"を押すと@ConversationScopedを終了する。

index.xhtml

スタート画面	
スタート	

view1.xhtml

画面①	
商品名	
数量	
キャンセル	画面②へ移る

view2.xhtml

画面②	
名前	
住所	
戻る	画面③へ移る

view3.xhtml

画面③	
名前	
住所	
商品名	
数量	
戻る	購入する

会話スコープの範囲

"購入する"をクリックすると、Conversationを終了し、スタート画面に戻る。

@ConversationScopedの使い方

管理Bean (CS.java)

@Named

@ConversationScoped

```
public class CS implements Serializable {
```

```
    private String product;
```

```
    private Integer qty;
```

```
    private String name;
```

```
    private String address;
```

@Inject

Conversation conv;

```
    public String goto_1() {
```

```
        if(conv.isTransient()) {
```

```
            conv.begin();
```

```
            System.out.println("**画面① 開始**");
```

```
        } else {
```

```
            System.out.println("**画面①**");
```

```
        }
```

```
        return "view1.xhtml";
```

```
    }
```

•isTransientメソッドは
@ConversationScopedが
開始されていない場合に
"true"を返す。

•beginメソッドで
@ConversationScopedを
開始する。

(続き)

```
    public String goto_2() {
```

```
        System.out.println("画面②");
```

```
        return "view2.xhtml";
```

```
    }
```

```
    public String goto_3() {
```

```
        System.out.println("画面③");
```

```
        return "view3.xhtml";
```

```
    }
```

```
    public String goto_0() {
```

```
        conv.end();
```

```
        System.out.println("画面① 終了");
```

```
        return "index.xhtml";
```

```
    }
```

//ゲッターとセッターは省略

endメソッドで
@ConversationScoped
を終了する。

<参考>サンプルアプリケーション ソースコード

[view1.xhtml](#)

```
...
<h:form>
    <h:panelGrid border="1" columns="2">
        商品名<h:inputText value="#{cS.product}"></h:inputText>
        数量<h:inputText value="#{cS.qty}"></h:inputText>
        <h:commandButton value="キャンセル" action="#{cS.goto_0}"></h:commandButton>
        <h:commandButton value="画面②へ移る" action="#{cS.goto_2}"></h:commandButton>
    </h:panelGrid>
</h:form>
...
```

[view2.xhtml](#)

```
...
<h:form>
    <h:panelGrid border="1" columns="2">
        名前<h:inputText value="#{cS.name}"></h:inputText>
        住所<h:inputText value="#{cS.address}"></h:inputText>
        <h:commandButton value="戻る" action="#{cS.goto_1}"></h:commandButton>
        <h:commandButton value="画面③へ移る" action="#{cS.goto_3}"></h:commandButton>
    </h:panelGrid>
</h:form>
...
```

<参考>サンプルアプリケーション ソースコード

view3.xhtml

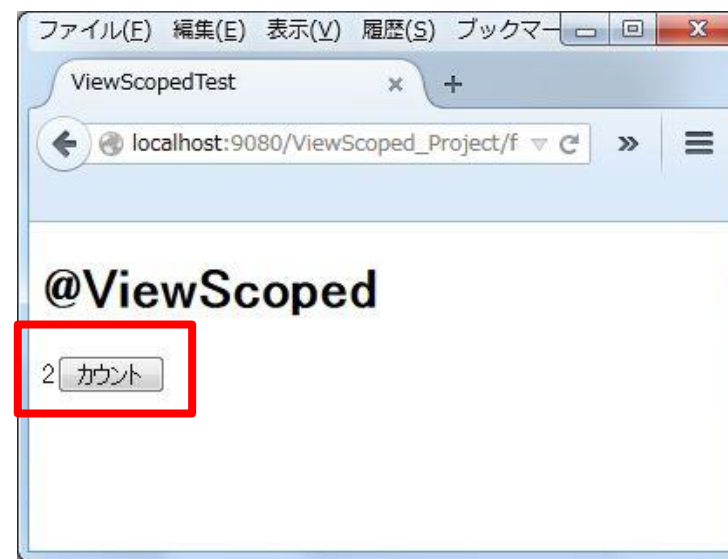
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http:
//www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
<h:head>
  <title>画面③</title>
</h:head>
<h:body>
<h1>画面③</h1>
```

```
<h:form>
  <h:panelGrid border="1" columns="2">
    名前<h:outputText value="#{cS.name}"></h:outputText>
    住所<h:outputText value="#{cS.address}"></h:outputText>
    商品名<h:outputText value="#{cS.product}"></h:outputText>
    数量<h:outputText value="#{cS.qty}"></h:outputText>
    <h:commandButton value="戻る" action="#{cS.goto_2}"></h:commandButton>
    <h:commandButton value="購入する" action="#{cS.goto_0}"></h:commandButton>
  </h:panelGrid>
</h:form>
</h:body>
</html>
```

2-3. ViewScoped

@ViewScopedとは

- JSF2.2は、CDI管理Beanの@ViewScopedをサポート。
- @ViewScopedは、画面遷移しない間であればリクエストをまたがってもフィールドの情報が保持されるスコープアノテーション。
- 画面をリロードするとビューが変わるため、スコープからはずれる。
- @ViewScopedはAjaxなど同一画面内で処理し続ける場合に有効。
- サンプルアプリケーション
 - "カウント"ボタンをクリックするとカウントされる。画面をリロードするとカウントは0になる。（次頁にサンプルアプリケーションのソースコードを記載）



@ViewScopedを使用例

ViewScopedBean.java

```
package viewScoped;
... (省略)
@Named
@ViewScoped
//@RequestScoped
public class ViewScopedBean implements Serializable{
    private int count;

    public int getCount() {
        return count;
    }
    public void setCount(int count) {
        this.count = count;
    }
    public void countUp(){
        count++;
    }
    public ViewScopedTest() {
    }
}
```

- CDI管理Beanに@ViewScopedを指定する。
- @RequestScopedを指定すると、リクエストの度に新たなbeanが生成、初期化されるので、countUpが実行される度に"1"になる。(アプリケーションの動作は"カウント"ボタンをクリックしても1から変わらないように見える。)

view1.xhtml

```
...
<h:form>
    <h1>@ViewScoped</h1>
    <h:form>
        <h:outputLabel value="#{viewScopedBean.count}" />
        <h:commandButton value="カウント" action="#{viewScopedTest.countUp()}" />
    </h:form>
</h:form>
...
```


3. Ajax

3-1.Ajax

3-2.Ajaxによるキュー制御



3-1. Ajax

Ajax

■ Ajax とは

– Ajaxとは、Webブラウザに実装されているJavaScriptのHTTP通信機能を使って、Webページのリロードを伴わずにサーバーとXML形式のデータのやり取りを行って処理を進めていく対話型Webアプリケーションの実装形態。

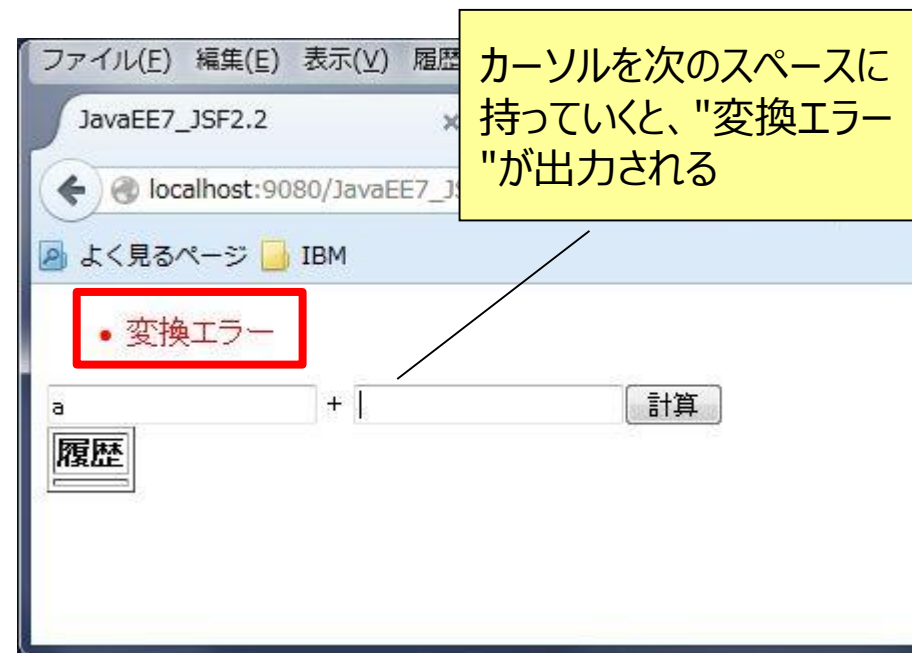
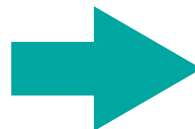
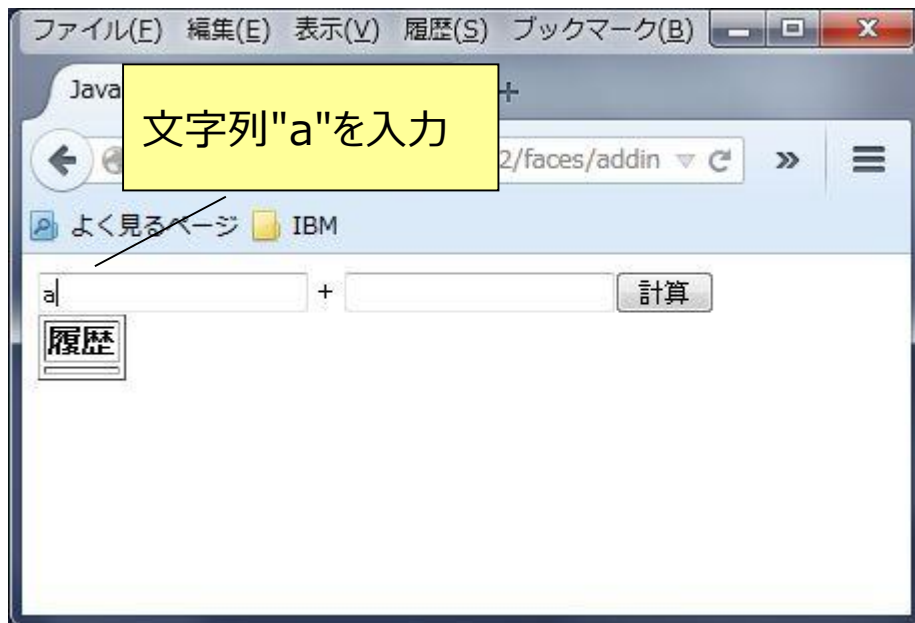
(出展：IT用語辞典 e-Words)

- 例えば、JSFページで送信ボタンを押すとフォーム内の全データがサーバーに送信される。サーバーはデータを受け取り、次に表示するJSFページの全情報をクライアントに送り返す。
- JSFのAjax機能を使うとJSFページ内の指定のデータだけをサーバーとやり取りすることができるようになる。データ書き換え時や、指定したフィールドにマウスがのポインターが乗ったときなどの指定したイベントが発生したときに、特定のフィールドのみ書き換えてメッセージを表示するなどができる。
- Ajaxの解説にあたり、「Java EE 7 アプリケーション設計ガイド – JSF(JavaServer Faces)2.2 入門編」で使ったサンプルアプリケーションを次ページで使用する。

Ajaxの使用例

■ サンプルアプリケーション

- 入力スペースに、数字以外を入力して、カーソルを次の入力スペースに移す。
- この時点で入力ミスがあった場合、画面にエラーが出力される。



Ajaxの使い方 (1)

- 「Java EE 7 アプリケーション設計ガイド – JSF(JavaServer Faces)2.2 入門編」でを使用したサンプルアプリケーションに以下の行を追加することで、Ajax機能が使えるようになる。

Facelets (adding.xhtml)

```
...
<h:body>
  <h:messages id="messages" style="color:#ff0000;" />
  <h:form>
    <h:inputText required="true" id="augend" value="#{addingBean.augend}">
      <f:converter converterId="javax.faces.Integer" />
      <f:validateLongRange minimum="0" maximum="100" />
      <f:ajax event="blur" execute="@this" render="messages" />
    </h:inputText>
    +
    <h:inputText required="true" id="addend" value="#{addingBean.addend}">
      <f:converter converterId="javax.faces.Integer" />
      <f:validateLongRange minimum="0" maximum="100" />
      <f:ajax event="blur" execute="@this" render="messages" />
    </h:inputText>
    <h:commandButton value="計算" action="#{addingBean.add}" />
  </h:form>
  ...
```

入力値の検査にAjaxを利用している。
Ajaxタグを追加する前のソースコードでは、コンバージョンとバリデーションは、全ての項目を入力し、計算ボタンを押したときに実行される。この動作では間違いがあっても全体を入力した後でしかわからない。
ここでは、項目を入力すると、その場ですぐにチェックできるようにAjax機能を使ってコンバージョンとバリデーションを実行している。

Ajaxの使い方 (2)

- 以下、サンプルアプリケーションのAjaxタグに指定した属性について示す。
- `event="blur"`
 - `event`属性はAjax処理を起動するトリガとなるイベントを指定する。ここでは"blur"を指定して、「コンポーネントからフォーカスが外れた」時にAjax処理が実行されるように指定している。サンプルアプリケーションでは、「文字（数字）」の入力を終えて次の項目に移った瞬間に、Ajax処理が実行される。
- `execute="@this"`
 - `execute`属性はサーバーに送信するコンポーネントのidを指定する。"@this"を指定することで、ここではこのAjaxタグが追加されただけをサーバーに送信する。サーバーでは受け取ったコンポーネントに指定された変換・検証処理を実施する。
 - `execute="@this"`は既定値であるため、省略してもよい。
- `render="messages"`
 - `render`属性は、サーバーからの返信を受けて、ブラウザ側で再描写するコンポーネントのidを指定する。サンプルアプリケーションでは、"messages"を指定しているので、h:messagesタグの値が返される。ブラウザは受け取った要素の値を使って、h:messagesの表示を更新する。そのため、エラーがあった場合にはメッセージが画面に表示される。エラーがなければ、h:messagesの値はnullであるため、画面表示はかわらない。

<参考>f:ajax タグの属性に指定できる値

- ここでは、f:ajax タグの属性に指定できる代表的なプロパティを示す。

プロパティ	意味と使用できる値
event	• Ajax 処理を行うイベントを指定する。
execute	• サーバーに送信するコンポーネントのidを指定する。 • もしくは、@all、@form、@none、@thisのシンボルを指定する。 • 指定を省略した場合には、@thisがデフォルト値になる。
render	• サーバーからの返信を受けて、ブラウザ側で再描写するコンポーネントのidを指定する。 • 指定できるシンボルには、@all、@form、@none、@thisがある。指定を省略した場合には、@none（レンダリングしない）がデフォルト値になる。
listener	• Ajax処理でサーバー側で実行させたいメソッドを指定する。
onevent	• Ajax処理の起動時に、ブラウザ側で起動するJavaScriptのメソッドを指定する。
onerror	• Ajax処理が失敗した時に、ブラウザ側で起動するJavaScriptのメソッドを指定する。

<参考>f:ajax タグの属性に使用できるシンボル

- ここでは、f:ajax タグのexecute、render属性に指定できるシンボル。

値	意味
@all	すべてのコンポーネント（のid）を意味する。
@form	コンポーネントを含むh:formを意味する
@none	指定しない（なし）を意味する
@this	Ajax処理を起動したコンポーネント自身を指す（既定値、省略値）

3-2. Ajaxによるキュー制御

Ajaxによるキュー制御

- JSF2.2では、f:ajaxタグの"delay"属性を指定することでキューを制御することができる。
 - delay属性に指定した期間(単位:milliseconds)に複数のリクエストがあった場合、最後のリクエストのみをサーバーに転送する。
 - デフォルトは、300milliseconds。
 - "none"はこの設定を"disable"にする。

Facelets

```
<h:commandButton id="button" value="submit" action="#{someBean.someAction}">  
    <f:ajax execute="@form" delay="none" />  
</h:commandButton>
```

4. ファイルアップロード

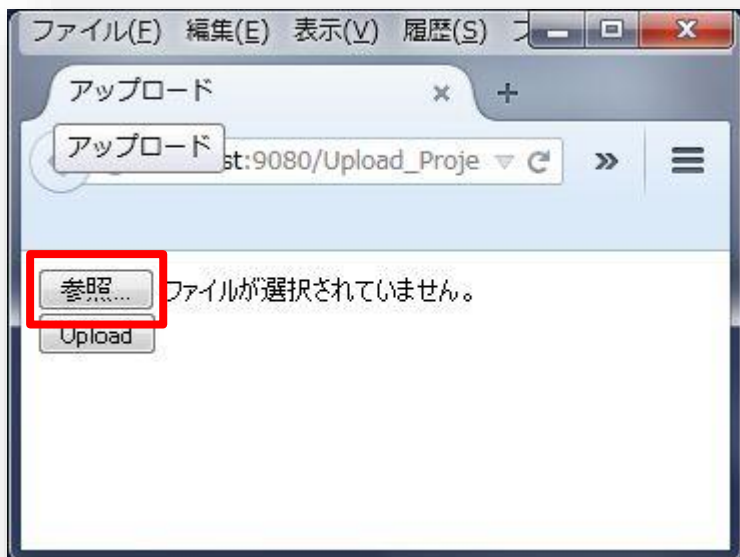


ファイル・アップロードの使用例

■ サンプルアプリケーション

- "参照"ボタンをクリックして、アップロードするファイルを選択する(図①)。
- "Upload"ボタンをクリックして、ファイルをアップロードする(図②)。
- 当サンプルアプリケーションでは、アップロードしたファイル名が画面に出力される(図③)。
- "Upload"ボタンをクリックするとファイル・アップロードとアップロード後の処理（画面にファイル名を表示）が実行される。

①



②



③



ファイル・アップロード

- JSF 2.2はファイル・アップロードコンポーネント(<h:inputFile>)を標準でサポート。

uploadview.xhtml

... (省略)

<h:body>

```
<h:form enctype="multipart/form-data">
```

```
<h:inputFile id="file" value="#{uploadBean.file}">
```

```
</h:inputFile><br />
```

```
<h:commandButton value="Upload" action="#{uploadBean.upload}" />
```

```
</h:commandButton><br />
```

```
</h:form>
```

```
<h:outputText id="fileName" value="#{upload.file == null ? '' :  
upload.file.submittedFileName}" />
```

```
</h:body>
```

・form要素に"enctype"を追加し、MIMEタイプにマルチパート (enctype="multipart/form-data") を指定する。

・ファイルはServlet3.0のファイル・アップロードと同様に javax.servlet.http.Partとして、valueにEL式で指定したフィールド ("file") にセットされる。

UploadBean.java

```
package upload;  
import javax.inject.Named;  
import javax.enterprise.context.RequestScoped;  
import javax.servlet.http.Part;  
@Named  
@RequestScoped  
public class UploadBean {  
    private Part file;  
    public Part getFile() {  
        return file;  
    }  
    public void setFile(Part file) {  
        this.file = file;  
    }  
    public void upload() {  
    }  
}
```

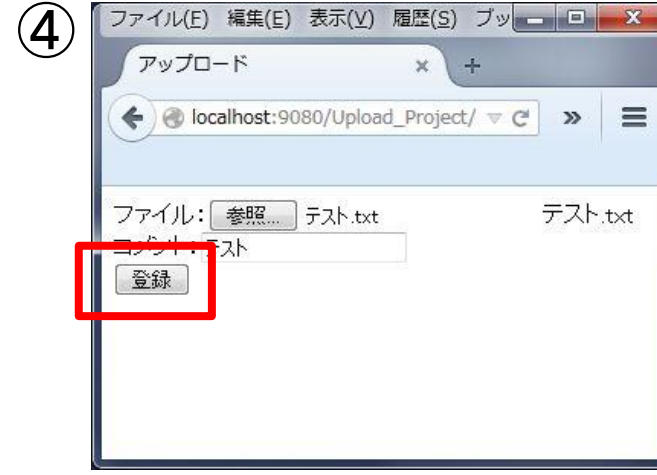
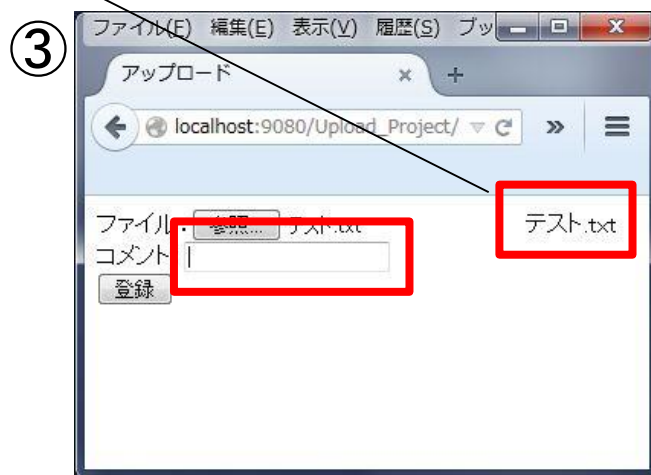
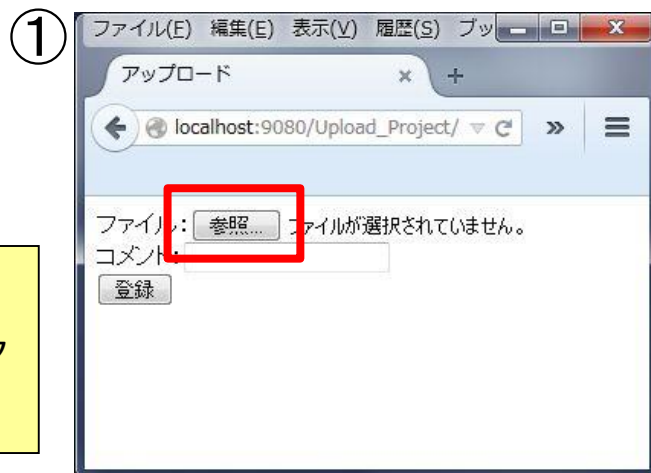
・Partとして受け取ったファイルのデータを使う処理をここに書く。

Ajaxによるファイル・アップロードの使用例

■ サンプルアプリケーション

- "参照"ボタンをクリックしてアップロードするファイルを選択する。カーソルをコメント欄に移してコメントを入力して"登録"ボタンをクリックする。

・カーソルをコメント欄に移すと、選択したファイルが表示される。



Ajaxによるファイル・アップロード

- Ajaxタグを以下の例のように追加することでファイル・アップロードすることが可能。管理Beanはp.37に記載してあるものと同様。

uploadview.xhtml

... (省略)

```
<h:body>
```

```
  <h:form enctype="multipart/form-data">
```

```
    ファイル: <h:inputFile id="file" value="#{uploadBean.file}">
```

```
      <f:ajax event="blur" execute="@this" render="fileName" />
```

```
    </h:inputFile>
```

```
      <h:outputText id="fileName" value="#{uploadBean.file == null ? '' : uploadBean.file.submittedFileName}" /><br/>
```

```
  </h:form>
```

```
  <h:form>
```

```
    コメント: <h:inputText id="userName" value="#{uploadBean.comment}" /><br/>
```

```
    <h:commandButton value="登録" action="#{uploadBean.upload}" >
```

```
  </h:commandButton><br />
```

```
</h:form>
```

```
</h:body>
```

・フォーカスがはずれた瞬間にファイルが送信される。

・ファイルのアップロードとデータの処理を別のタイミングにできる。

・"登録"ボタンをクリックするとuploadメソッドを実行する。その中でAjaxで送信済みのデータ処理するロジックがある。

5.HTML5

4-1.パススルーアトリビュート

4-2.パススルーエレメント



5-1. パススルーアトリビュート

パススルーアトリビュート（１）

- JSF2.2では、HTML5へのサポートが強化され、HTML5の仕様に追加されたタグや属性をパススルーすることで対応できる。
- 「パススルーアトリビュート」は、定義が未確定なタグ属性についてJSFでは特に処理を行わず、そのままHTMLで出力させることができる。パススルーアトリビュートの指定方法は３つある。
- 方法１
 - ネームスペースに「xmlns:p="http://xmlns.jcp.org/jsf/passthrough"」を宣言する。
 - 入力された値をそのままHTMLに変換する箇所を「p:属性名=XXXX」と指定する。
 - p:を付けた属性がFaceletsタグの中にあると、その属性はFaceletsの属性ではなく、レンダリング後のHTMLに適用する属性として扱われる。

Facelets

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://java.sun.com/jsf/passthrough">
  <h:form>
    <h:inputText value="#{bean.value}" p:placeholder="Enter text"/>
  </h:form>
</html>
```

・"placeholder"はHTML5で使えるようになった属性で、入力ガイド（入力例）を表示する。テキストを入力すると、ガイド文字列は消える。

・ここでは、p:を付けた属性であることからHTMLに適用する属性として扱われる。

パススルーアトリビュート（２）

■ 方法 2

- ネームスペースに「xmlns:f="http://java.sun.com/jsf/core"」を宣言する。
- <f:passThroughAttribute>タグを指定する。
- <f:passThroughAttribute>タグを付けると、そのタグはFaceletsの属性ではなく、レンダリング後のHTMLに適用する属性として扱われる。

Facelets

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
  <h:form>
    <h:inputText value="#{bean.value}" >
      <f:passThroughAttribute name="placeholder" value="Enter text" />
    </h:outputText>
  </h:form>
</html>
```

パススルーアトリビュート（３）

- 方法３：複数のアトリビュートを指定できる。
 - ネームスペースに「xmlns:f="http://java.sun.com/jsf/core"」を宣言する。
 - f:passThroughAttributesタグを指定する。
 - 方法２と同様に、<f:passThroughAttributes>タグを付けると、そのタグはFaceletsの属性ではなく、レンダリング後のHTMLに適用する属性として扱われる。
 - EL式でvalueにマップオブジェクトを指定する。

Facelets

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
  <h:form>
    <h:inputText value="#{bean.value}" />
    <f:passThroughAttributes value="#{bean.manyAttributes}" />
  </h:inputText>
</h:form>
</html>
```

{bean.manyAttributes} はMap<String, Object>を参照する。
EL式で指定するので以下のような使い方も可能。
value="{\"one\":1, \"two\":2, \"three\":3}"

<参考>パススルーアトリビュート

- 以下は方法 1、2 で示したFaceletsのサンプルソースコードをHTMLにレンダリングしたもの。
- パススルーアトリビュートを指定した"placeholder"属性は、JSFの処理は行わず、そのままHTMLとして処理されている。

HTML

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<form id="j_id_4" name="j_id_4" method="post" action="/Pass_through_attributes/faces/passthrough.xhtml"
enctype="application/x-www-form-urlencoded">
<input id="j_id_4:j_id_5" name="j_id_4:j_id_5" type="text" value="" placeholder="Enter text" />
<input type="hidden" name="j_id_4_SUBMIT" value="1" />
<input type="hidden" name="javax.faces.ViewState" id="j_id__v_0:javax.faces.ViewState:1" value=
"ADygs64mPGMsV+4VgNemnZXbmByIQIKl3LrUxuRt+a50eE+p" autocomplete="off" /></form>

</html>
```

5-2. パススルーエレメント

パススルーエレメント

- パススルーエレメントは、HTMLフレンドリーなマークアップ方法を提供。
- JSF2.1ではjsfc属性を組み込むことでJSFコンポーネントをHTMLとして認識させていた(次頁参照)。JSF2.2ではjsf属性を使うことでコンポーネント名を書かなくてもJSFコンポーネントとして認識させることができる。
- ネームスペースにxmlns:jsf="http://xmlns.jcp.org/jsf"を宣言し、以下のようにjsf属性を使ってHTMLタグの属性として組み込む。
- 注意：タグに直接置き換わるものはない。

JSF 2.2

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:jsf="http://xmlns.jcp.org/jsf">
  <body>
    <form jsf:id="form">
      Welcome, #{loggedInUser.name}
      <input type="text" jsf:value="#{bean.property}" />
      <input type="submit" value="OK" jsf:action="#{bean.doSomething}" />
    </form>
  </body>
</html>
```

参考 JSF2.1 jsfc属性

- JSF2.1ではHTMLタグにjsfc属性を組み込むことで、JSFのコンポーネントをHTMLとして認識させることができる。
- jsfc属性にはJSFのコンポーネント名を記述する。

JSF 2.1

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html">
  <body>
    <form jsfc="h:form">
      <span jsfc="h:outputText" value="Welcome, #{loggedInUser.name}" />
      <input type="text" jsfc="h:inputText" value="#{bean.property}" />
      <input type="submit" jsfc="h:commandButton" value="OK" action="#{bean.doSomething}" />
    </form>
  </body>
</html>
```


6. View Actions



View Actions

- ビューを表示するときに呼び出す処理を指定するコンポーネント(`f:viewAction`)が追加された。
- 例えば、画面表示のタイミングでアクションを実行したいときなどに有効。
 - 以下のようなサンプルアプリケーションのURLにアクセスすると、自動的にカウントされる。



- 初期リクエストで処理を実行できる。
- 2回目以降も処理を実行するように指定できる。
- JSFライフサイクルのどのフェーズで処理を実行するか指定することができる。
 - `APPLY_REQUEST_VALUES`, `PROCESS_VALIDATIONS`, `UPDATE_MODEL_VALUES`, `INVOKE_APPLICATION` (デフォルトは`INVOKE_APPLICATION`)

Facelets

```
<f:metadata>
  <f:viewAction action="#{someBean.someAction}" />
</f:metadata>
```

- `f:viewAction`は`metadata`ファセット`<f:metadata>`の子として宣言する。
- ボタンを押したときにしかアクションを実行できなかったのに対して、画面表示するだけのときにもこの機能でアクションを指定できるようになった。

ViewActionの使用例

- 以下は前頁のサンプルアプリケーションのソースコード。

GetSupport.xhtml

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

... (省略)

```
<h:body>
```

```
<f:metadata>
```

```
  <f:viewAction action="#{viewActionBean.init()}" onPostback="true"
    phase="APPLY_REQUEST_VALUES" />
```

```
</f:metadata>
```

```
<div>Count: #{viewActionBean.count}</div>
```

```
<h:form>
```

```
  <h:commandButton value="Click" />
```

```
</h:form>
```

```
</h:body>
```

```
</html>
```

・f:viewActionは初期表示時の1回のみ動作する。このため、処理を続ける（ボタンをクリックすることでカウントする）動作にするには、f:viewActionタグの属性に、onPostback="true"を追加する。
・ここでは、APPLY_REQUEST_VALUESフェーズで処理を実行するように指定。

ViewActionBean.java

```
package viewaction;
```

```
import javax.faces.view.ViewScoped;
import javax.inject.Named;
```

```
@Named
```

```
@ViewScoped
```

```
public class ViewActionBean {
    private int count = 0;
    public int getCount() {
        return count;
    }
    public void init() {
        count++;
    }
}
```

7. ステートレスビュー



ステートレスビュー

- 従来のJSFでは、サーバ側でコンポーネント・ツリーを保持するステートフルが前提。（コンポーネントツリーについては、「Java EE 7 アプリケーション設計ガイド – JSF(JavaServer Faces)2.2 入門編」を参照。）
- ステートレスモードではサーバ側ではコンポーネントツリーを保持せず、「クライアントからのアクセスがある度にコンポーネントツリーを作り直す」という動作を行う。
- 注意：ステートを保有するコンポーネントが使用できなくなる。例、@ViewScoped

Facelets

```
<f:view xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core"
  transient="true">
  <html>
    <h:head/>
    <h:body>
      <h:form>
        <!-- Input components here -->
      </h:form>
    </h:body>
  </html>
</f:view>
```

ネームスペースに「transient="true"」を宣言することでステートレスモードに設定。

8. Faces Flow



Faces Flowとは

- Faces Flowは画面遷移の一連の流れをまとまり（フロー）として管理する仕組み。
 - フロー内の画面を遷移している間は同一のスコープが保持される。
- スコープにはCDI管理Beanの"@FlowScoped"を使用（右図）
- フローの定義方法
 - XMLによる定義
 - プログラムによる定義（当資料では省略）
 - FlowBuilderクラスを使ってプログラムでフローを定義

CDI 管理Bean

```
@Named
@FlowScoped("someFlowName")
public class SomBean {
    public String someReturn() {
        return "/somePage.xhtml";
    }
}
```

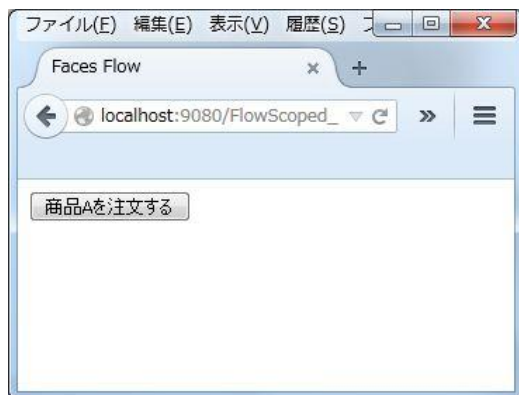
- 特徴
 - @RequestScopedや@ViewScopedより長く、@SessionScopedより短く定義したい場合に有効。この点は@ConversationScopedも同様に有効。
 - フローの単位（Viewと画面遷移）でパッケージングでき、再利用可能。
 - ブラウザのタブ毎にフローを制御することが可能。

Faces Flowの使用例

■ サンプルアプリケーション

- 画面①で"商品Aを注文する"をクリックすると順番に注文数を入力する画面（画面②）、配送先を入力する画面（画面③）に遷移する。最後の画面（画面④）で注文内容を確認して"注文確定"させる。

① *home.xhtml*



"注文確定"を押すと、フローを終了し、スタート画面に戻る。

④ *order-confirmation.xhtml*



② *order-flow.xhtml*



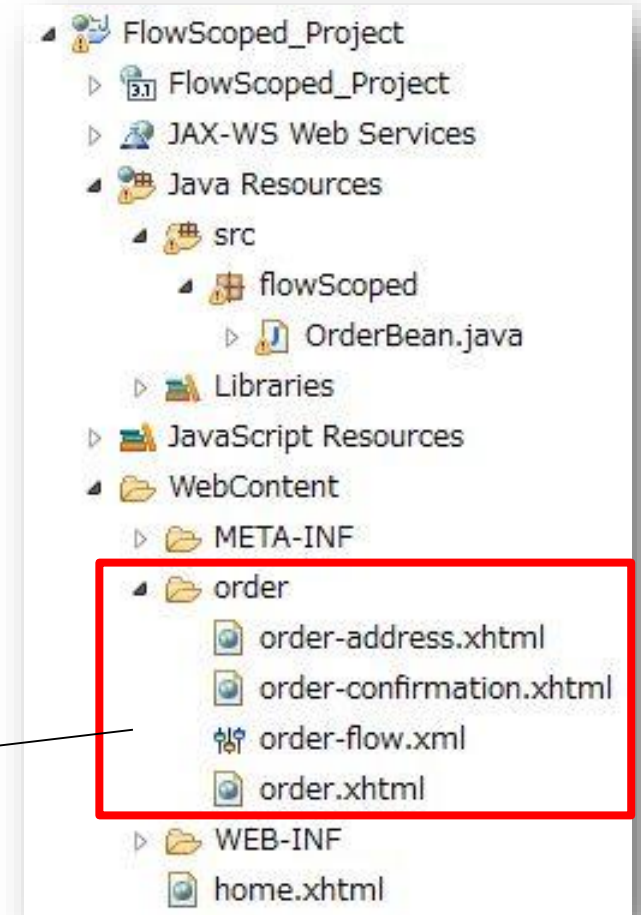
③ *order-address.xhtml*



用意するソースファイル

- 当サンプルアプリケーションを作成するにあたり、以下のソースファイルを用意する。
 - order-flow.xml : フローを定義する
 - home.xhtml : フローの入り口となるノード
 - order-flow.xhtml
 - order-address.xhtml
 - order-confirmation.xhtml : フローの呼び出し元に戻るノード
 - OrderBean.java
- WebContentにorderフォルダーを作成し、XMLファイルとスコープ範囲内となるビュー(XHTMLファイル)を作成する。

・order配下に置いたビュー(xhtmlファイル)はフローの範囲内になる。



XMLによるフローの定義

- 当サンプルアプリケーションではXMLにフローを定義する。（右図）
- フォルダー名とidを一致させる（必須）。

・フローidを指定する。

・当サンプルアプリケーションではXMLファイルにフローの開始は定義しない。フローidと同じ名前のビュー(order.xhtml)を"order"フォルダーに用意することでデフォルトの開始ビューとして使われる。

・フローの終了を定義する。フローの呼び出し元に出力を返す定義をする。
フローの呼び出し元(home.xhtml)に戻る。

order-flow.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config version="2.2"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">

  <flow-definition id="order">

    <flow-return id="returnFromOrderFlow">
      <from-outcome>#{orderBean.returnValue}</from-outcome>
    </flow-return>

  </flow-definition>

</faces-config>
```

<参考>サンプルアプリケーションのソースコード

OrderBean.java

```
package flowScoped;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.faces.flow.FlowScoped;
import javax.inject.Named;
import java.io.Serializable;

@Named
@FlowScoped("order")
public class OrderBean implements Serializable {

    private int itemCount;
    private String address;
    public int getItemCount() {
        return itemCount;
    }
}
```

@FlowScopedを指定する。フロー-id
を指定する（必須）。

```
public void setItemCount(int itemCount) {
    this.itemCount = itemCount;
}
public String getAddress() {
    return address;
}
public void setAddress(String address) {
    this.address = address;
}
public String getReturnValue() {
    return "home";
}
}
```

<参考>サンプルアプリケーションのソースコード

home.xhtml

```
... (省略)
<h:head>
  <title>FlowScoped</title>
</h:head>
<h:body>
  <h:form>
    <h:commandButton value="商品Aを注文する"
      action="order" />
  </h:form>
</h:body>
</html>
```

order-flow.xmlで定義したフローidを指定してフローが開始する。

order-confirmation.xhtml

```
... (省略)
<h:head>
  <title>注文確認</title>
</h:head>
<h:body>
  <h:form>
    <h:outputText value="商品Aの個数: #{orderBean.itemCount}" />
    <br />
    <h:outputText
      value="配送先: #{orderBean.address}" />
    <br />
    <h:commandButton value="注文確定" action="returnFromOrderFlow" />
  </h:form>
</h:body>
</html>
```

order-flow.xmlの<flow-return>のid属性で定義した名前を指定する。これにより、フローから出ることになる。

<参考>サンプルアプリケーションのソースコード

order.xhtml

```
... (省略)
<h:head>
<title>注文数</title>
</h:head>
<h:body>
<h:form>
  <h:outputText value="商品の数: " />
  <br />
  <h:inputText value="#{orderBean.itemCount}" />
  <br />
  <h:commandButton value="続ける" action="order-address" />
</h:form>
</h:body>
```

order-address.xhtml

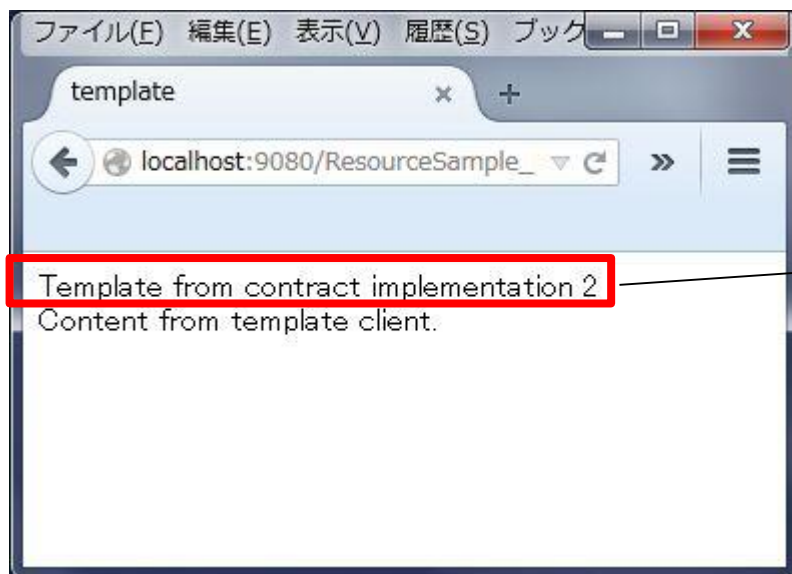
```
... (省略)
<h:head>
<title>住所</title>
</h:head>
<h:body>
<h:form>
  <h:outputText value="商品Aの個数: #{orderBean.itemCount} " />
  <br />
  <h:outputText value="配送先: " />
  <br />
  <h:inputText value="#{orderBean.address}" />
  <h:commandButton value="Proceed" action="order-confirmation" />
</h:form>
</h:body>
</html>
```

9. Resource



リソースライブラリコントラクト

- 「リソースライブラリコントラクト」は、1つの Web アプリケーションに対して、複数の画面デザインのテンプレート(CSS等)を事前に作成しておく事で、画面のルック・アンド・フィールを簡単に(動的に)切り替えることができる機能。
- 切り替える方法として、faces-configを使ってurlパターン毎に使用するセットを切り替える。もしくは、アプリケーションで動的に切り替えることもできる。
- サンプルアプリケーション①
 - "contract-client.xhtml"をブラウザで開くと以下の画面を表示。(contract-client.xhtmlは次頁参照)



複数用意したテンプレートの中から指定したテンプレートで表示する。

リソースライブラリコントラクトの使用例① - 1

- 右図のようなディレクトリ構成でソースファイルを用意する。

- /contracts/contract1/template.xhtml
- /contracts/contract2/template.xhtml
- contract-client.xhtml

/contract1/template.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

  <body>
    Template from contract implementation 1 <br/>

    <ui:insert name="content"/>
  </body>
</html>
```

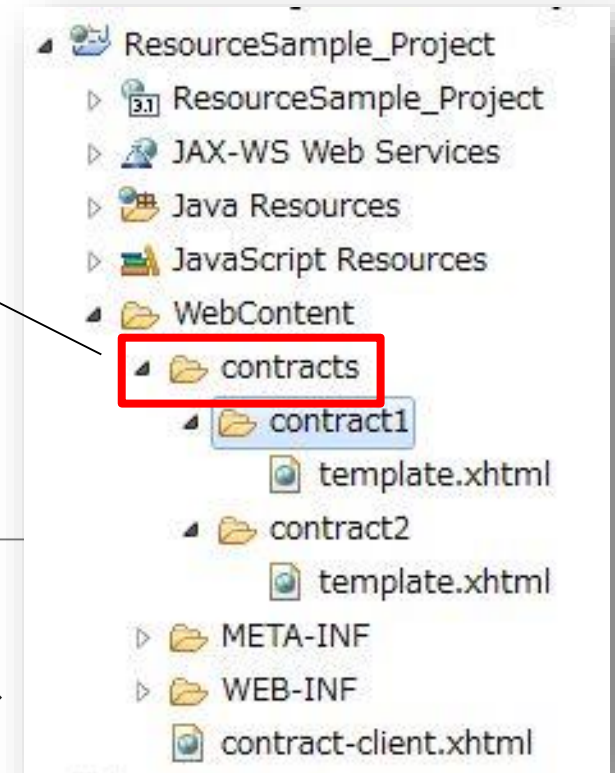
/contract2/template.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

  <body>
    Template from contract implementation 2 <br/>

    <ui:insert name="content"/>
  </body>
</html>
```

フォルダー名"contracts"(必須)
を作成する。



リソースライブラリコントラクトの使用例① - 2

- "contract-client.xhtml"をブラウザで開くと、画面の一部を/contracts/contract2/template.xhtmlのテンプレートから表示する。

contract-client.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
```

```
<f:view contracts="contract2">
<ui:composition template="/template.xhtml">
```

```
<ui:define name="content">
    Content from template client.
```

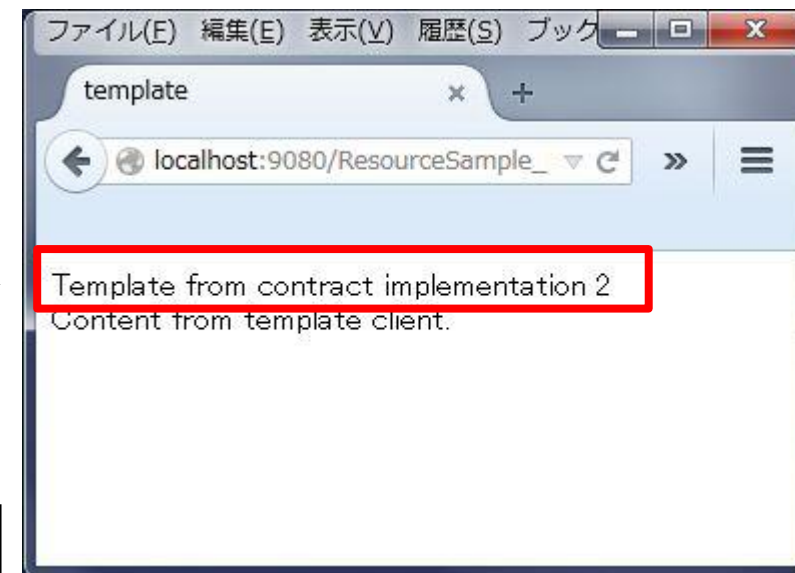
```
</ui:define>
```

```
</ui:composition>
```

```
</f:view>
```

```
</html>
```

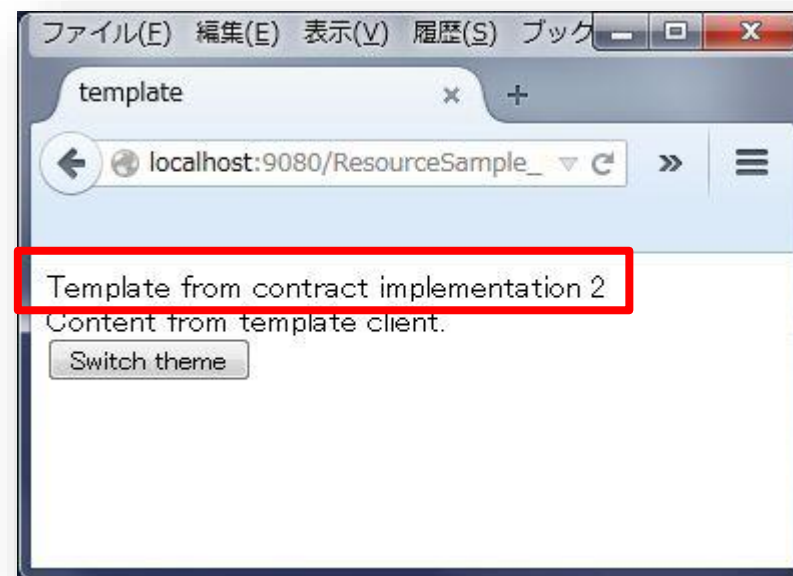
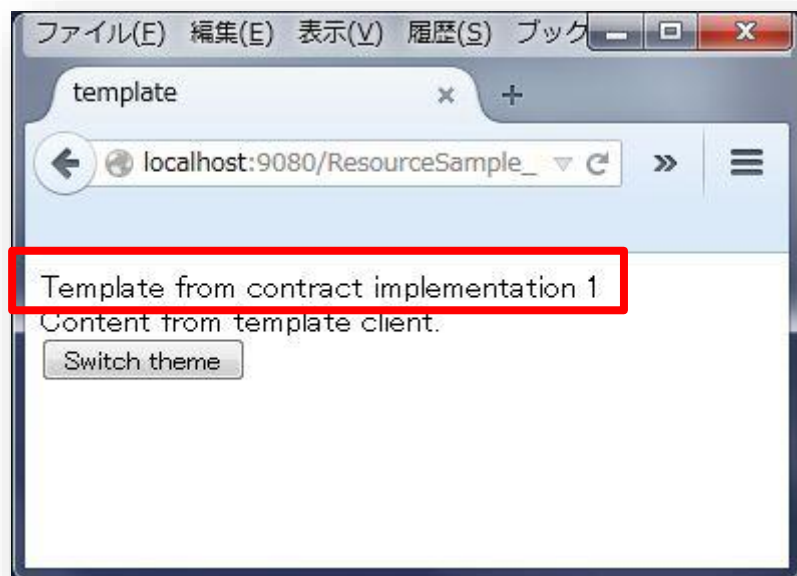
<f:view>タグのcontracts属性に、使用するcontractを指定する。
<ui:composition>タグに指定されたテンプレートは上記で指定したcontractのものが使われる。



リソースライブラリコントラクトの使用例② - 1

■ サンプルアプリケーション②

- 前頁"contract-client.xhtml" の<f:view>タグのcontracts属性にEL式を指定して、テンプレートを動的に変更する。
- "Switch theme"ボタンをクリックすると表示するテンプレートが切り替わる。



リソースライブラリコントラクトの使用例② - 2

- contract-client.xhtmlを以下のように修正する。

contract-client.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
  <f:view contracts="#{themeSwitcher.theme}">
    <ui:composition template="/template.xhtml">
      <ui:define name="content">
        Content from template client.
        <h:form>
          <h:commandButton action="#{themeSwitcher.switchTheme}" value="Switch theme" />
        </h:form>
      </ui:define>
    </ui:composition>
  </f:view>
</html>
```

<f:view>タグのcontracts属性には
EL式を指定することも可能。

リソースライブラリコントラクトの使用例② - 3

- CDI管理Beanを用意する。 *ThemeSwitcher.java*

```
import java.io.Serializable;
import javax.enterprise.context.SessionScoped;
import javax.inject.Named;
@Named
@SessionScoped
public class ThemeSwitcher implements Serializable {
    private static final long serialVersionUID = -1L;
    private String theme = "contract1";
    public void switchTheme() {
        if ("contract1".equals(theme)) {
            theme = "contract2";
        } else {
            theme = "contract1";
        }
    }
    public String getTheme() {
        return theme;
    }
}
```

・"Switch theme"ボタンがクリックされる度に "theme"がかわる。 contract-client.xhtml の<f:view>タグの contracts属性の値もかわるため、画面表示に使用するテンプレートがかわる。

10. まとめ



まとめ

- 当ガイドでは Java EE 7 アプリケーション設計ガイドとして、JSF 2.2 の詳細編について解説を行いました。

JSF 2.2

- JSF 2.2はBig Ticket Featureが追加
 - Faces Flow
 - パススルーアトリビュート
 - パススルーエレメント
 - ステートレスビュー
 - リソースライブラリーコントラクト
- CDIとの親和性が向上
 - @FlowScoped, @ViewScoped

参考文献

- JSR 344: JavaServerTM Faces 2.2
 - <https://jcp.org/en/jsr/detail?id=344>
- Java EE 7 Specification APIs
 - <http://docs.oracle.com/javaee/7/api/>
- The Java EE 7 Tutorial
 - <http://docs.oracle.com/javaee/7/tutorial/index.html>
- IBM Knowledge Center - WebSphere Application Server Liberty Core 8.5.5
 - <http://www-01.ibm.com/support/knowledgecenter/SSD28V/welcome>
- Liberty Profile Beta
 - <https://developer.ibm.com/wasdev/>
- IBM Knowledge Center - WebSphere Application Server Liberty Profile Beta
 - http://www-01.ibm.com/support/knowledgecenter/was_beta_liberty/as_ditamaps/wasbeta_welcome_wlp.html