

JavaServer Faces (JSF)

JSF – спецификация для построения компонентно-ориентированных пользовательских интерфейсов для веб-приложений Java (MVC)

Появилась 2004г. С 2006г. часть JavaEE
Версия - 2.3 (2017)

Основана на Facelets framework (или фейслетах)

Тэг- ориентированная (.xhtml)



ORACLE

JavaServer™ Faces Specification

Version 2.3

Ed Burns and Manfred Riem editors



Реализация библиотек компонентов

- ▶ RichFaces (RedHat)
- ▶ ADF Faces (Oracle)
- ▶ PrimeFaces
- ▶ Tomahawk
- ▶ Trinidad
- ▶ Tobago
- ▶ Orchestra
- ▶ ICEFaces
- ▶ OpenFaces
- ▶ OmniFaces
- ▶ PrettyFaces

Реализация JSF

- Mojarra (Oracle)
- MyFaces (Apache)

Основные пакеты JavaServer Faces

Пакет	Описание
javax.faces	Основные API JavaServer Faces
javax.faces.application	API, используемый для связи бизнес-логики приложения с JavaServer Faces
javax.faces.component	Основные API для работы с компонентами пользовательского интерфейса
javax.faces.context	Классы и интерфейсы, определяющие информацию о состоянии по запросу

Пакет	Описание
javax.faces.convert	Классы и интерфейсы, определяющие преобразователи
javax.faces.event	Интерфейсы, описывающие события и слушатели событий, а также конкретная реализация классов событий
javax.faces.flow	Классы и API среды выполнения для Faces Flow
javax.faces.lifecycle	Классы и интерфейсы, определяющие управление жизненным циклом для реализации JSF
javax.faces.render	Классы и интерфейсы, определяющие модель отрисовки
javax.faces.validator	Интерфейс, определяющий модель валидатора и конкретные реализации классов валидатора
javax.faces.webapp	Классы, требуемые для интеграции JSF в веб-приложения, включая стандартные сервлеты Faces

- ▶ Эталонная реализация JSF 2.3 – Mojarra (Oracle) доступен в GlassFish5
- ▶ <https://docs.oracle.com/javaee/7/tutorial/jsf-intro001.htm#BNAPK>

JSF vs JSP

- ▶ Усиливает разделение представления и логики
- ▶ UI компоненты (html – компоненты в JSP)
- ▶ Управляется событиями
- ▶ Простая навигация между web страницами
- ▶ Содержат конфигурируемые управляемые бины (bean)

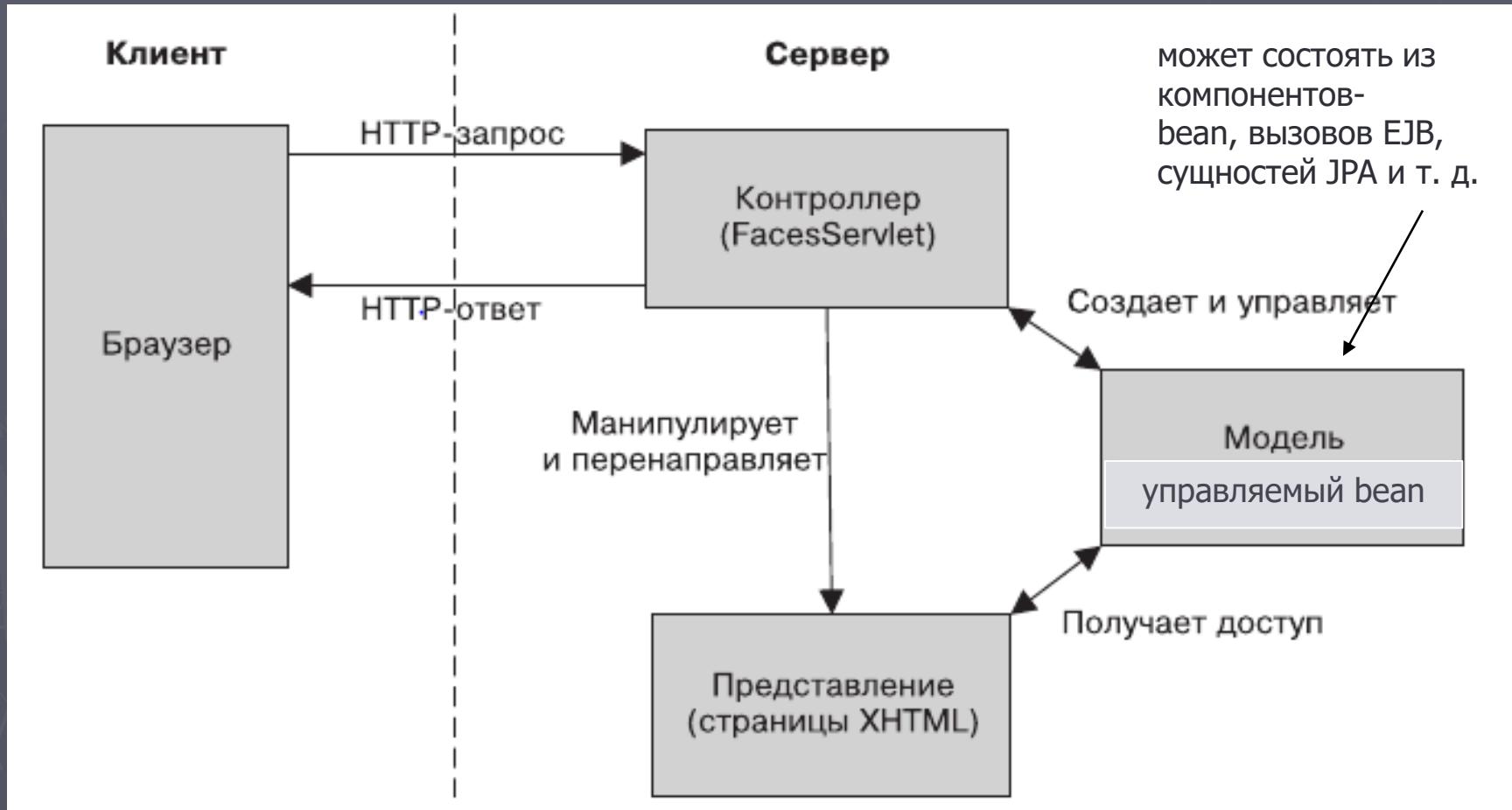
JSF vs JSP

- ▶ JSP — это язык страниц
- ▶ JSF — это слой компонентов, располагающийся над ним
- ▶ JSF - повторно использует JSP -> JSTL и EL (использует все технологии)

Архитектура JSF



Поддержка шаблона MVC

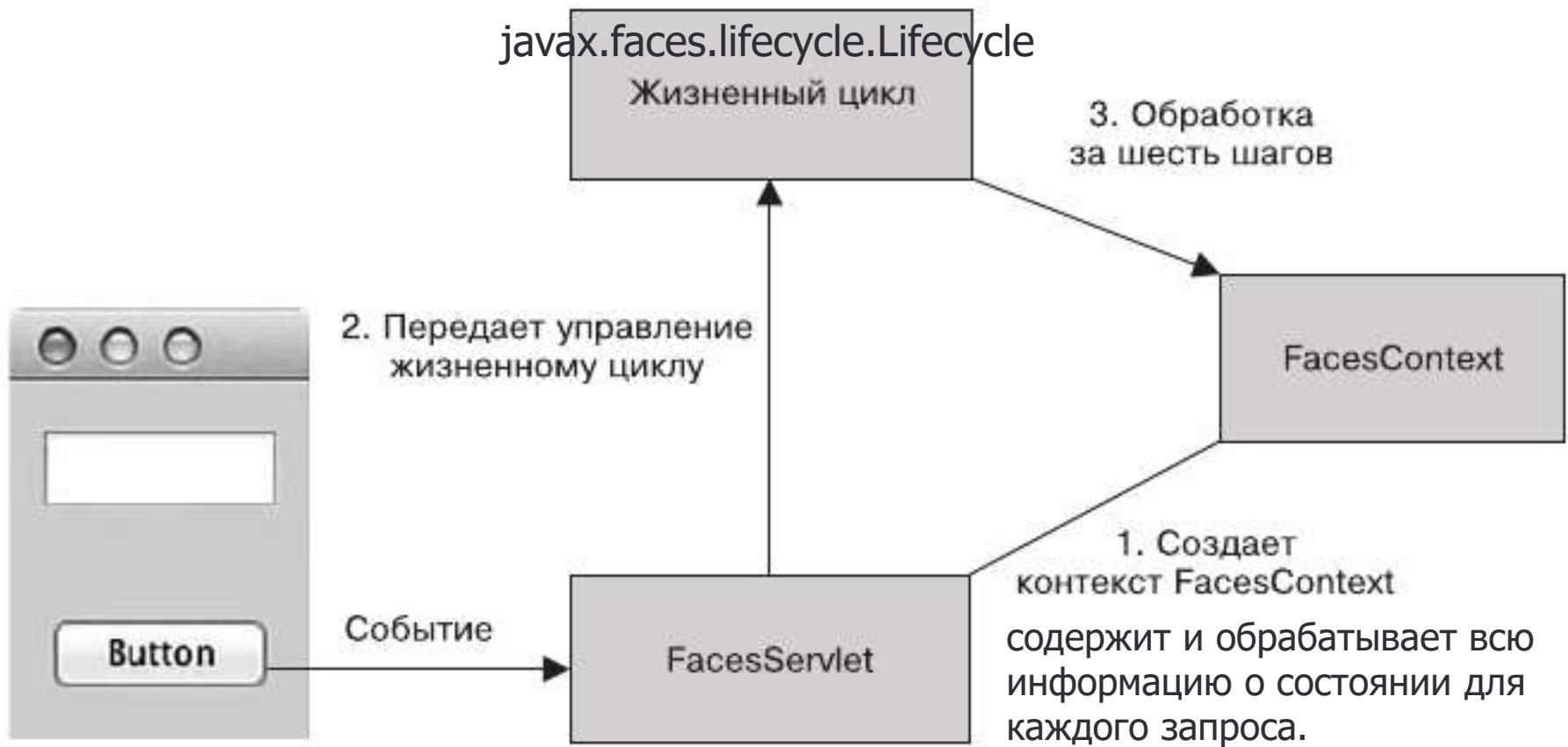


FacesServlet

Это реализация типа javax.servlet.Servlet.
Это контроллер - сервлет, управляющий
жизненным циклом обработки запросов для
веб-приложений , через который проходят
все запросы пользователей.

Настраивается только с использованием
внешних метаданных.

Взаимодействия FacesServlet



► настройка FacesServlet

■ web.xml

```
< servlet >
    < servlet-name >Faces Servlet</ servlet-name >
    < servlet-class >javax.faces.webapp.FacesServlet</ servlet-class >
    < load-on-startup >1</ load-on-startup >
</ servlet >
< servlet-mapping >
    < servlet-name >Faces Servlet</ servlet-name >
    < url-pattern >*.jsf</ url-pattern >
</ servlet-mapping >
```

все запросы,
которые имеют расширение .jsf,
управляются сервлетом
по умолчанию .faces

► Настройка дополнительных параметров

```
<context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
</context-param>

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

Параметр	Описание
javax.faces.DISABLE_FACELET_JSF_VIEWHANDLER	Значение true отключает Facelets как стандартный язык объявления страниц (PDL)
javax.faces.WEBAPP_RESOURCES_DIRECTORY	Если установлено значение этого параметра, среда выполнения JSF интерпретирует его как путь относительно корня веб-приложения, где должны располагаться ресурсы
javax.faces.LIBRARIES	Интерпретирует каждый файл, найденный в списке путей, разделенных двоеточием, как библиотеку тегов Facelets

Параметр	Описание
javax.faces.CONFIG_FILES	Определяет список разделенных запятой зависящих от контекста путей, по которым реализация JSF будет искать ресурсы
javax.faces.DEFAULT_SUFFIX	Позволяет веб-приложению определить список альтернативных суффиксов для страниц, имеющих содержимое JSF (то есть с расширением .jsf)
javax.faces.FACELETS_BUFFER_SIZE	Размер буфера ответа. По умолчанию равно -1
javax.faces.FACELETS_REFRESH_PERIOD	Когда запрашивается страница, этот параметр будет использоваться как интервал, по истечении которого компилятор будет проверять наличие изменений. Значение -1 отключает эту проверку
javax.faces.FACELETS_SKIP_COMMENTS	Если установлено значение true, среда выполнения гарантирует, что XML-комментарии на странице Facelets не будут доставлены клиенту
javax.faces.LIFECYCLE_ID	Идентифицирует объект типа Lifecycle как использованный при обработке запросов JSF
javax.faces.STATE_SAVING_METHOD	Определяет местоположение сохранения состояния. Корректными значениями являются server, которое задается по умолчанию (обычно сохраняется в HttpSession), и client (сохраняется в скрытом поле при последующей отправке формы)
javax.faces.PROJECT_STAGE	Описывает, на каком этапе жизненного цикла находится это конкретное приложение JSF (Development, UnitTest, SystemTest или Production). Этот параметр может быть использован реализацией JSF для кэширования ресурсов, например, чтобы улучшить производительность на производстве

FacesContext

- ▶ абстрактный класс `javax.faces.context.FacesContext` для представления контекстной информации, связанной с обработкой входящих запросов и создания соответствующего ответа.
- ▶ Позволяет взаимодействовать с пользовательским интерфейсом и остальной частью среды JSF.

► Доступ

1) Ссылка на неявный объект FacesContext

2) Ссылку в управляемый бин с помощью статического метода getCurrentInstance()

Метод	Описание
release	Высвобождает любые ресурсы, связанные с объектом типа FacesContext
renderResponse	Сигнализирует реализации JSF о том, что текущая фаза обрабатывающего запросы жизненного цикла была закончена, управление должно быть передано фазе «Отрисовать ответ», минуя любые фазы, которые еще не были выполнены
responseComplete	Сигнализирует реализации JSF о том, что HTTP-ответ для этого запроса уже был сгенерирован (например, переадресация HTTP), а также о том, что жизненный цикл обработки запросов должен прекратить свою работу, как только завершится текущая фаза

Метод	Описание
AddMessage	Присоединяет сообщение (информационное, предупреждающее, сообщение об ошибке либо сообщение о фатальной ошибке)
GetApplication	Возвращает объект типа Application, связанный с этим веб-приложением
GetAttributes	Возвращает объект типа Map, представляющий атрибуты, связанные с объектом типа FacesContext
getCurrentInstance	Возвращает объект типа FacesContext для запроса, который обрабатывается в текущем потоке
getELContext	Возвращает объект типа ELContext для текущего объекта типа FacesContext
getMaximumSeverity	Возвращает максимальную степень тяжести, записанную в любом FacesMessage, внесенном в очередь
GetMessages	Возвращает коллекцию объектов типа FacesMessage
getPartialViewContext	Возвращает объект типа PartialViewContext для заданного запроса. Он используется для внедрения логики в цикл управления обработкой/отрисовкой (например, для обработки AJAX)
getViewRoot	Возвращает корневой компонент, связанный с запросом

Страницы и компоненты

- ▶ PDL
- ▶ Facelets – предпочтительная
- ▶ Страницы Facelets:
 - состоят из дерева компонентов (текстовое поле, кнопки, списки и т. д.) стандартных или пользовательских
 - Имеет жизненный цикл(инициализация, события, отрисовка и т. д.).

XHTML-страница Facelets

```
<?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"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
<h:head>
    <title> Do payment</title>
</h:head>
<h:body>
    <h1> Do payment </h1>
    <hr/>
    <h:form>
        <table border="0">JSF-компонентов
            <tr>
                <td><h:outputLabel value="Card # : ">/</td>
                <td><h:inputText value="#{cardController.card.number}" /></td>
            </tr>
            <tr>
                <td><h:outputLabel value="Name : ">/</td>
                <td><h:inputText value="#{cardController.card.name}" /></td>
            </tr>
        </table>
        <h:commandButton value="Pay"
                         action="#{cardController.doPay}" styleClass="submit"/>
    </h:form>
    <hr/>
    <em> Java EE 7</em>
</h:body>
</html>
```

Использует теги JSF

JSF-компоненты

чистые HTML-теги

Отрисовщики

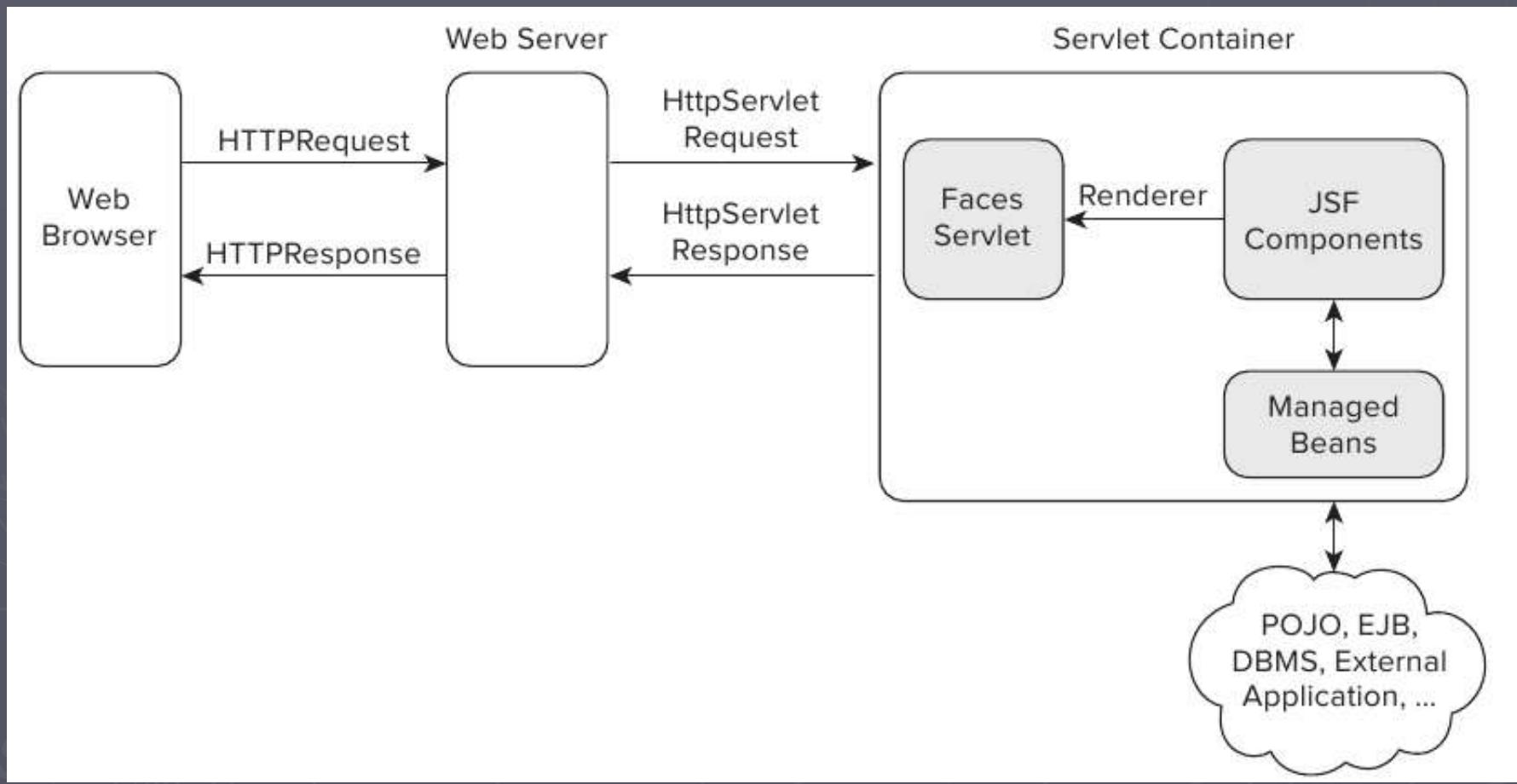
- ▶ JSF поддерживает две модели программирования для отображения компонента:
 - непосредственную
 - ▶ компоненты должны декодировать и закодировать себя в графическое представление
 - делегированную
 - ▶ операции делегируются отрисовщику, который позволяет компоненту не зависеть от технологии отрисовки (браузер, переносное устройство и т. д.) и иметь несколько графических представлений.

Преобразователи и валидаторы

- ▶ Преобразователи - преобразуют объект (Integer, Date, Boolean и т. д.) в строку для отображения и из входной строки обратно в объект
 - Стандартные преобразователи для общих типов (`javax.faces.convert`)
 - Пользовательские
 - Сторонние преобразователи

Преобразователи и валидаторы

- ▶ Валидаторы отвечают за то, чтобы значение, введенное пользователем, было корректным.
 - Стандартная поставка (LengthValidator, RegexValidator и т. д.).
 - Пользовательские валидаторы на основе аннотированных классов.
 - Делегирована Bean Validation



Managed JSF beans

- ▶ JavaBeans, которые могут быть напрямую связаны с JSF компонентами.
- ▶ - хранит информацию (в атрибутах)
- ▶ - содержит данные о навигации страниц
- ▶ - выполняют бизнес-логику (или делегируют ее выполнение компонентам EJB) (метод).

Управляемый бин и навигация

- ▶ Управляемый бин — это специализированный класс Java, который синхронизирует значения с компонентами (через свойства или EL), обрабатывает бизнес-логику и навигацию между страницами

Введенное текстовое значение
синхронизировано со свойством
card.name bean CardController

```
<h:inputText value="#{cardController.card.name}" />  
  
<h:commandButton value="Pay"  
action="#{cardController.doPay}" styleClass="submit"/>
```

Bean обрабатывает событие
путем генерации HTTP-запроса POST

Bean (компонент)

изменение имени по умолчанию влияет на способ вызова свойства или метода

```
@ManagedBean (name="PnvBean")  
@RequestScoped  
public class CardController {  
  
    private Card card = new Card();  
  
    public String doPayment()  
    {  
        return "listCards.xhtml";  
    }  
    // Методы работы со свойствами  
}
```

обладает состоянием

в течение конкретного периода времени (область действия)

определяет методы действий

обрабатывает навигацию

аннотированный POJO

POJO (Plain Old Java Object — простой Java-объект в старом стиле)

@Named

@RequestScoped

@Documented

@Stereotype

@Target({ TYPE, METHOD, FIELD })

@Retention(RUNTIME)

Области действия управляемых бинов

@ApplicationScoped

@SessionScoped

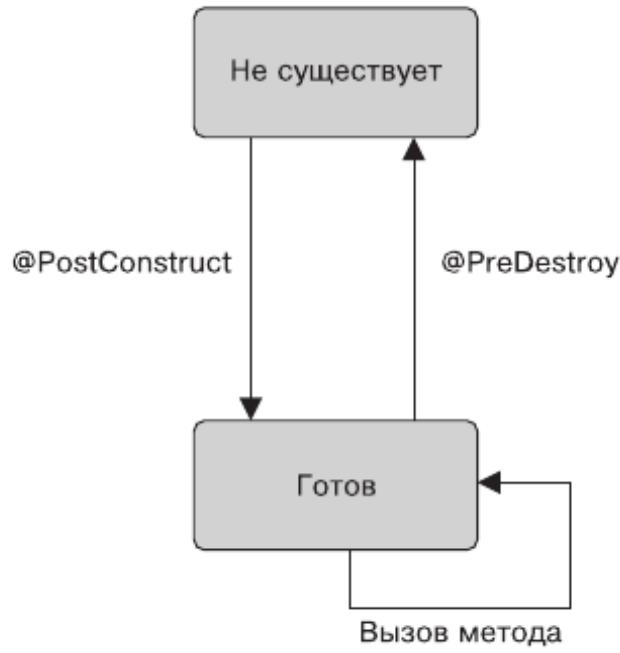
@ViewScoped

@RequestScoped

- ▶ Созданные объекты доступны во всех циклах запросов/ответов для всех клиентов, использующих веб-приложение, до тех пор, пока приложение активно.
- ▶ объекты доступны для любых циклов запросов/ответов, которые принадлежат сессии клиента
- ▶ объекты доступны в пределах заданного представления, пока оно не изменится, и их состояние сохраняется до тех пор, пока пользователь не перейдет к новому представлению
- ▶ состояние доступно в начале запроса и до тех пор, пока клиенту не был отправлен ответ

Поток (@FlowScoped) и момент

Аннотации функций обратного вызова



```
public class CardController {  
    private Card defaultCard;  
  
    @PostConstruct  
    private void init() {  
        defaultCard = new Card(0, "No", 0);  
    }  
}
```

Обработка исключений и сообщений

- ▶ <h:message> и <h:messages>

Создать и поместить сообщение в очередь

- ▶ facesContext.addMessage(String clientId, FacesMessage message)

Идентификатор клиента - он относится к расположению компонента пользовательского интерфейса
Null - сообщение не относится ни к одному компоненту и считается глобальным на всей странице.

- ▶ FacesMessage(Severity severity, String summary, String detail)

обобщенный текст, подробного текста и степени тяжести

```
public String createCard() {
```

предупреждающее сообщение (SEVERITY_WARN), связанное
компонентом пользовательского интерфейса

```
    FacesContext ctx = FacesContext.getCurrentInstance();
```

```
    if (card.getName() == null || "" .equals(card.getName())) {  
        ctx.addMessage("cardForm:name",
```

```
                    new FacesMessage(FacesMessage.SEVERITY_WARN,
```

```
                    "Неверное имя", "Вы должны ввести имя"));
```

```
}
```

```
    if (ctx.getMessageList().size() != 0)
```

```
        return null;
```

глобальные сообщения

```
    ctx.addMessage(null, new FacesMessage(FacesMessage.SEVERITY_ERROR,
```

```
                    "карта не может быть создана", ""));
```

*.xhtml

```
<h:inputText value="#{cardController.card.name}" />  
<h:message for="cardForm:name" />
```

Навигация

1) Явная навигация

FacesServlet, выступая в качестве контроллера

<h:button>, <h:link> и <h:outputLink>

```
<h:link value="Вернуться" outcome="newPage.xhtml"/>
```

<h:commandButton> и <h:commandLink>

атрибут action позволяет выбирать методы bean

```
<h:commandButton value="Create card"  
action="#{cardController.createCard}">
```

2) Правила навигации

может быть определена внешне в файле
faces-config.xml

<navigation-rule> -

определяет начальную страницу, условие и
целевую страницу, на которую нужно перейти
при срабатывании условия

```
@ManagedBean(name="cardController")
@RequestScoped
public class CardController {
    public String doMethod() {

        return "ok";
    }
}
```

Если было возвращено логическое имя

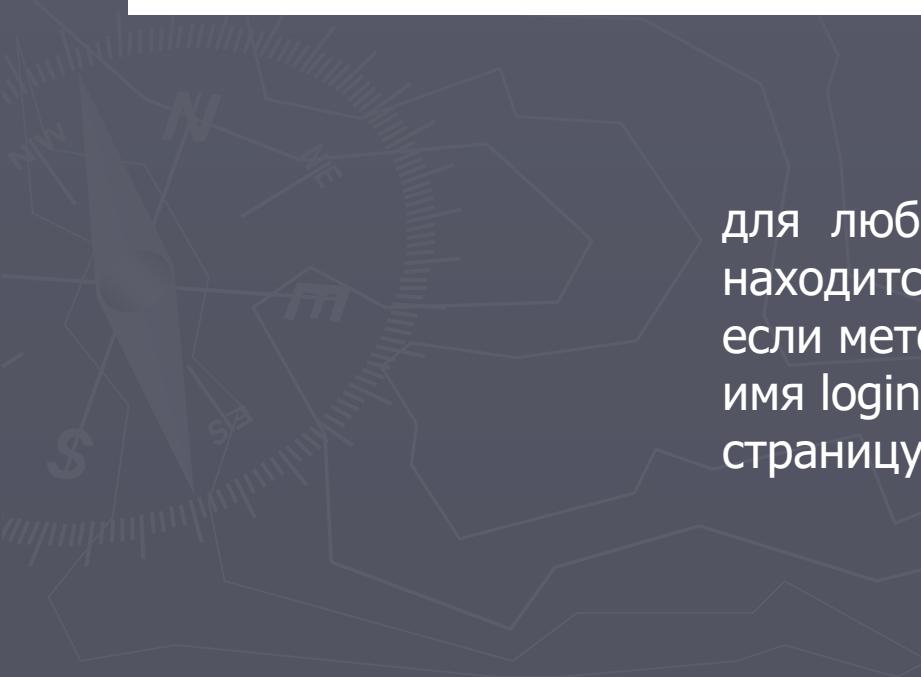


```
<navigation-rule>
    <from-view-id>newCard.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>ok</from-outcome>
        <to-view-id>listCard.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
</faces-config>
```

перейдет к вызову страницы

► Глобальные правила навигации

```
<navigation-rule>
    <from-view-id>*</from-view-id>
    <navigation-case>
        <from-outcome>login</from-outcome>
        <to-view-id>login.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```



для любой страницы, на которой находится пользователь, в том случае, если метод бина возвращает логическое имя login, следует перенаправить на страницу

Язык выражений

► #{}expr

```
# ${empty card}  
# ${empty card.number}
```

```
# {card.pay() }  
# {card.pay(200) }
```

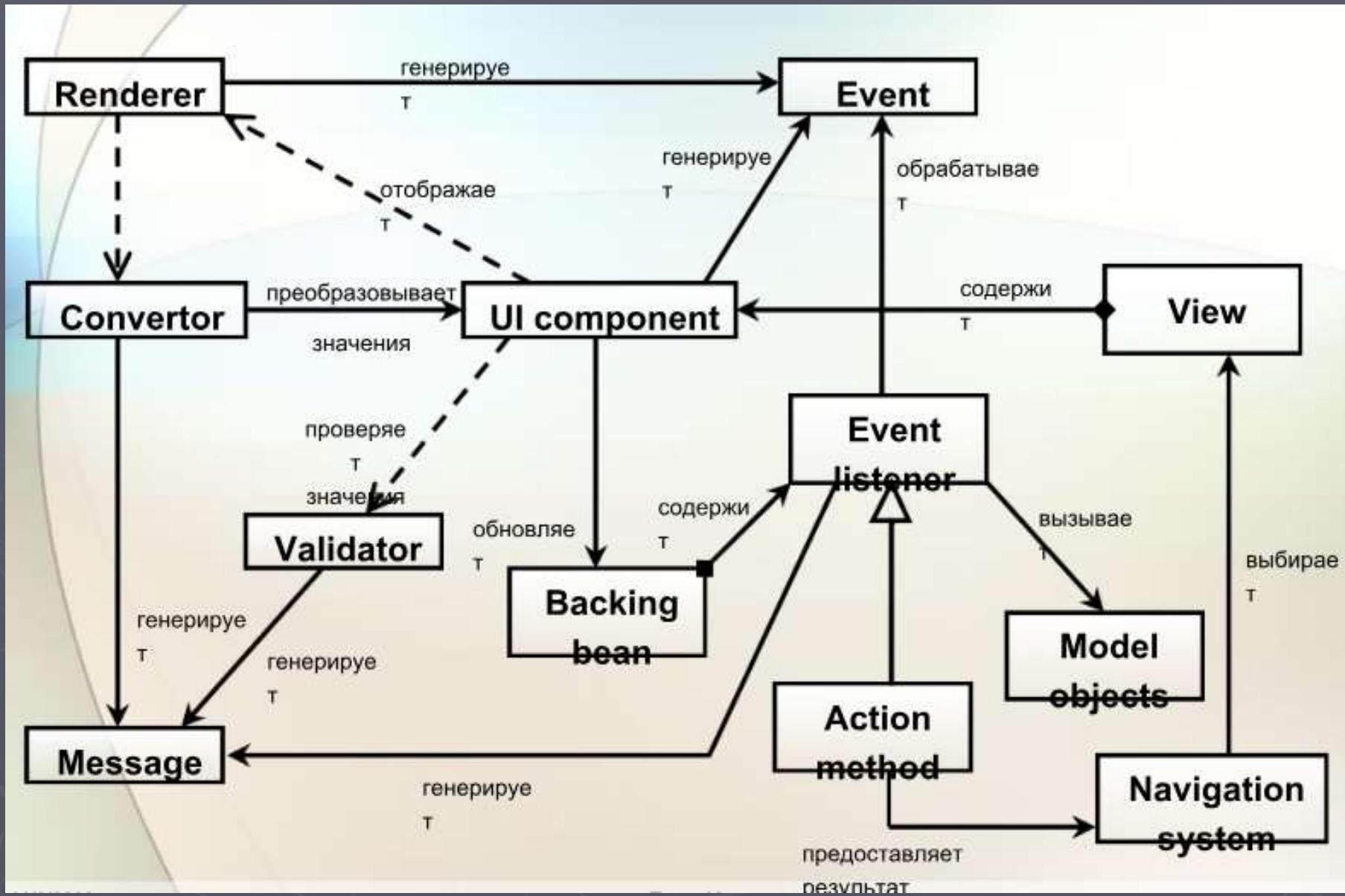
Поддержка AJAX

- ▶ jsf.js
- ▶ теге <f:ajax>

```
<h:commandButton value="Do payment"  
                  action="#{cardController.doPayment}">  
  
    <f:ajax execute="@form" render=":cardlist"/>  
  
</h:commandButton>
```

UI Component	Объект с состоянием, методами, событиями, который содержитя на сервере и отвечает за взаимодействие с пользователем.
Renderer	Отвечает за отображение компонента и преобразование ввода пользователя.
Validator	Проверяет правильность введенного пользователем значения.
Convertor	Преобразует свойства компонента в/из строки для отображения.
Backing bean	Специальный JavaBean, который собирает значения из компонент, реагирует на события, взаимодействует с бизнес-логикой.
Events and Listeners	Компоненты генерируют события, слушатели реагируют на них.
Messages	Сообщения, генерируемые любым объектом JSF и отображаемые пользователям
Navigation	Схема навигации между страницами

Взаимодействие компонентов JSF



Структура страницы JSF

- ▶ JSF - серверная технология (отрисовывается сервером)
- ▶ Страница JSF — это практически файл XHTML, определяющий список библиотек тегов в заголовке и содержит графическое представление.

```
<?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"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    xmlns:f="http://xmlns.jcp.org/jsf/core">
    объявление типа
    документа (DTD)
    xhtml1-transitional.dtd
    импортирует библиотеку тегов

<h:head>
    <title> Create card</title>
</h:head>
<h:body>
    <h1>Create card</h1>
    <hr/>
    <h:form>
        <h:panelGrid columns="2">
            <h:outputLabel value="Number : "/>
            <h:inputText value="#{cardController.card.number}" />
            <h:outputLabel value="Name : "/>
            <h:inputText value="#{cardController.card.name}" />
            <h:outputLabel value="Sum : "/>
            <h:inputText value="#{cardController.card.sum}" />
            <h:outputLabel value="Other : "/>
            <h:inputTextarea value="#{cardController.card.other}" cols="20"
                rows="5"/>
            <h:outputLabel value="Password : "/>
            <h:inputText value="#{cardController.card.passw}" />
        </h:panelGrid>
        <h:commandButton value="Create card" action="#{cardController.doCreateCard}" />
    </h:form>
    <hr/>
```

Библиотеки тегов, допускаемые к использованию в Facelets PDL

URI	Префикс	Описание
http://xmlns.jcp.org/jsf/html	h	Библиотека классов содержит компоненты и их HTML-отрисовки (h:commandButton, h:comandLink, h:inputText и т. д.)
http://xmlns.jcp.org/jsf/core	f	Библиотека содержит пользовательские действия, которые независимы от созданных ранее отрисовок (f:selectItem, f:validateLength, f:convertNumber и т. д.)
http://xmlns.jcp.org/jsf/facelets	ui	Теги этой библиотеки добавляют поддержку шаблонов
http://xmlns.jcp.org/jsf/composite	composite	Библиотека тегов применяется для объявления и определения составных компонентов
http://xmlns.jcp.org/jsp/jstl/core	c	Facelets-страницы могут использовать некоторые основные библиотеки тегов JSP (<c:if/>, <c:forEach/> или <c:catch/>)
http://xmlns.jcp.org/jsp/jstl/functions	fn	Facelets-страницы могут использовать все функции библиотек тегов JSP

Стандартные компоненты пользовательского интерфейса JSF

► наследуют от UIComponent

Компонент	Описание
UIColumn	Представляет колонку в родительском компоненте UIData
UICommand	Представляет графические компоненты, такие как кнопки, гиперссылки или меню
UIComponent	Базовый класс для всех компонентов пользовательского интерфейса в JSF
UIComponentBase	Класс, созданный для удобства, реализующий конкретное поведение по умолчанию для всех методов, определенных в UIComponent
UIData	Поддерживает связывание данных с коллекциями объектов, часто применяется для отрисовки таблиц, списков и деревьев
UIForm	Представляет форму пользовательского ввода и является контейнером других компонентов
UIGraphic	Выводит изображения
UIInput	Представляет компоненты, предназначенные для ввода данных, такие как поля ввода, текстовые области и т. д.
UIMessage, UIMessages	Отвечает за вывод одного или нескольких сообщений для определенного UIComponent

Компонент	Описание
UISelectOne, UISelectMany	Представляет такие компоненты, как раскрывающиеся списки или группы флажков, и позволяют выбирать один или несколько элементов
UIViewAction	Представляет вызов метода, который происходит при запросе обработки жизненного цикла
UIViewParameter	Представляет двунаправленное связывание между параметром запроса и свойством компонента-подложки
UIViewRoot	Представляет корень дерева компонентов и не имеет графической отрисовки
UIOutcomeTarget	Представляет графические кнопки и гиперссылки, позволяющие добавлять страницу в закладки
UIOutput	Представляет компоненты, предназначенные для вывода, такие как метки или другие формы вывода текста
UIPanel	Представляет компоненты пользовательского интерфейса, которые служат контейнерами для других компонентов, не требуя при этом отправки формы
UIParameter	Представляет информацию, которая не требует отрисовки
UISelectBoolean	Представляет флажки
UISelectItem, UISelectItems	Представляет один или несколько элементов в списке

<h:outputLabel>



javax.faces.component.html.HtmlOutputLabel,

UIOutput

наследуется

<h:panelGrid>



javax.faces.component.html.HtmlPanelGrid

UIPanel

Жизненный цикл JSF

- ▶ Восстановление вида
- ▶ Применение параметров запроса
- ▶ Проверка процесса
- ▶ Обновление значений модели
- ▶ Вызов приложения
- ▶ Отрисовка ответа

Компоненты страниц

- ▶ встроенные компоненты JSF: HTML, Core и Templating;
- ▶ теги JSTL;
- ▶ собственные компоненты;
- ▶ компоненты сторонних производителей (с открытым исходным кодом или коммерческие).

HTML-компоненты JSF

1) Команды

По умолчанию commandButton имеет тип submit

```
<h:commandButton value="Отправить" />
<h:commandButton type="reset" value="Очистить" />
<h:commandButton image="foto.gif" title="Изображение" />
<h:commandLink>Гиперссылка</h:commandLink>
```

вызывают действие на managed-beans путем генерации HTTP-запроса POST

2) Цели

Переход с одной страницы на другую, не вызывая managed beans



```
<h:button outcome="newBook.xhtml"  
           value="button link"/>  
<h:link outcome="newBook.xhtml" value="link"/>
```

создания HTTP-запроса GET

3) Компоненты ввода

отображают текущее значение пользователю и позволяют ему вводить различные виды текстовой информации.

```
<h:inputHidden value="Скрытые данные"/>
<h:inputSecret maxlength="8"/>
<h:inputText value="Введенный текст"/>
<h:inputText size="40" value="Чуть более длинный введенный текст"/>
<h:inputTextarea rows="4" cols="20" value="Текстовая область"/>
<h:inputFile/>
```

4) Компоненты вывода

отображают значение,
при необходимости извлекаемое из managed beans,
выражений или фиксированного текста.

```
<h:outputLabel value="#{cardController.card.number}" />
<h:outputText value="Текст"/>
<h:outputLink value=" http://www.ali.com/">Ссылка</h:outputLink>
<h:outputFormat value="Добро пожаловать {0}.
    Вы можете купить {1} товаров">
    <f:param value="#{user.firstName}" />
    <f:param value="#{user.Number}" />
</h:outputFormat>
```

5) Компоненты выбора

```
<h:selectOneMenu>
    <f:selectItem itemLabel="Срочный" />
    <f:selectItem itemLabel="Пенсионный" />
    <f:selectItem itemLabel="Детский" />
    <f:selectItem itemLabel="Под 10%" />
</h:selectOneMenu>
```

```
<h:selectBooleanCheckbox>
<h:selectManyCheckbox>
<h:selectManyListbox>
<h:selectManyMenu>
<h:selectOneRadio>
<h:selectOneMenu>
```

6) Графика

```
<h:graphicImage value="foto.gif" height="200" width="320"/>
```

7) Сетка и таблицы

```
<h:panelGrid columns="3" border="1">
    <h:outputLabel value="1"/>
    <h:outputLabel value="2"/>
    <h:outputLabel value="3"/>
    <h:outputLabel value="4"/>
    <h:outputLabel value="5"/>
    <h:outputLabel value="6"/>
</h:panelGrid>
```

```
<h:dataTable>
    <h:column>
```

8) Сообщения об ошибках

```
<h:message>  
<h:messages>
```

привязан к конкретным компонентам

может обеспечить глобальный механизм оповещения для всех компонентов на странице.

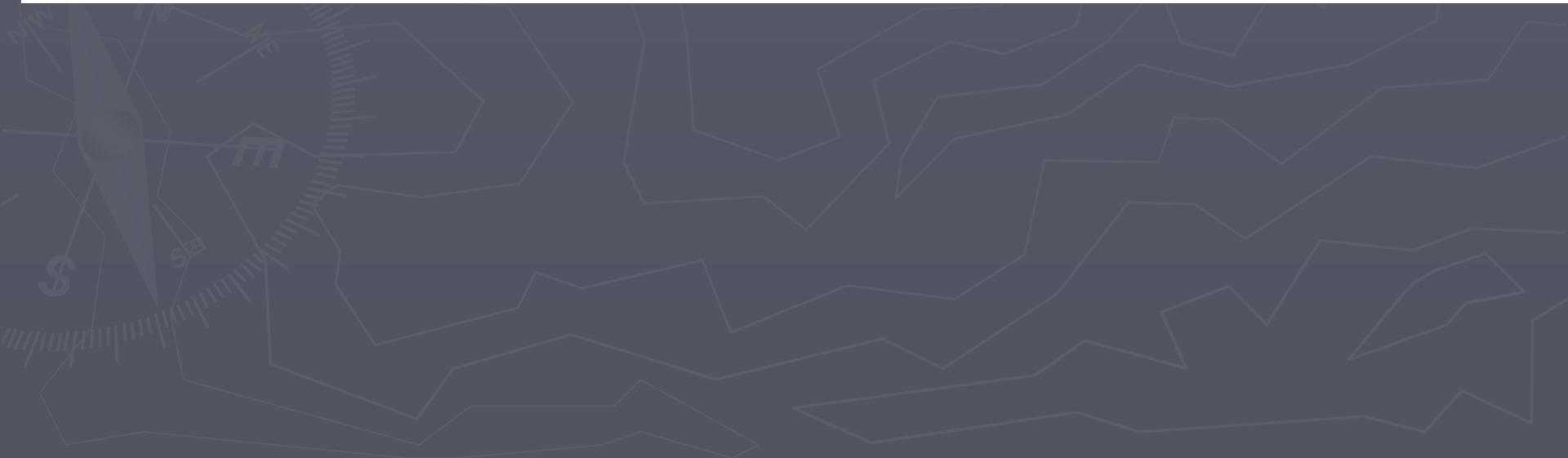
степень важности : INFO, WARN, ERROR и FATAL

свойство обязательно для заполнения,
M б сообщение об ошибке

```
<h:messages style="color:red"/>  
<h:form>  
    enter number:  
    <h:inputText value="#{cardController.number}" required="true"/>  
    <h:commandButton action="#{cardController.save}" value="Сохранить"/>  
</h:form>
```

9) Базовые атрибуты

Атрибут	Описание
id	Идентификатор компонента
rendered	Двоичное значение, указывающее, следует ли отрисовывать компонент
value	Значение компонента (текст или утверждение EL)
converter	Имя класса преобразователя
validator	Имя класса валидатора
required	Если имеет значение true, требует введения значения в соответствующее поле



Основные теги JSF

- не имеют никакого визуального представления и используются для передачи атрибутов или параметров других компонентов, а также для преобразования или валидации

Тег	Описание
<f:facet>	Добавляет фасет к компоненту
<f:attribute>	Добавляет атрибут компоненту
<f:param>	Создает компонент-потомок с параметром
<f:actionListener>	Добавляет компоненту слушатель действий, изменений значений или действий над свойствами
<f:valueChangeListener>	Добавляет компоненту слушатель действий, изменений значений или действий над свойствами
<f:propertyActionListener>	
<f:phaseListener>	Добавляет на страницу слушатель фаз
<f:converter>	Добавляет компоненту произвольный преобразователь или использует особый преобразователь даты/времени и чисел
<f:convertDateTime>	
<f:convertNumber>	

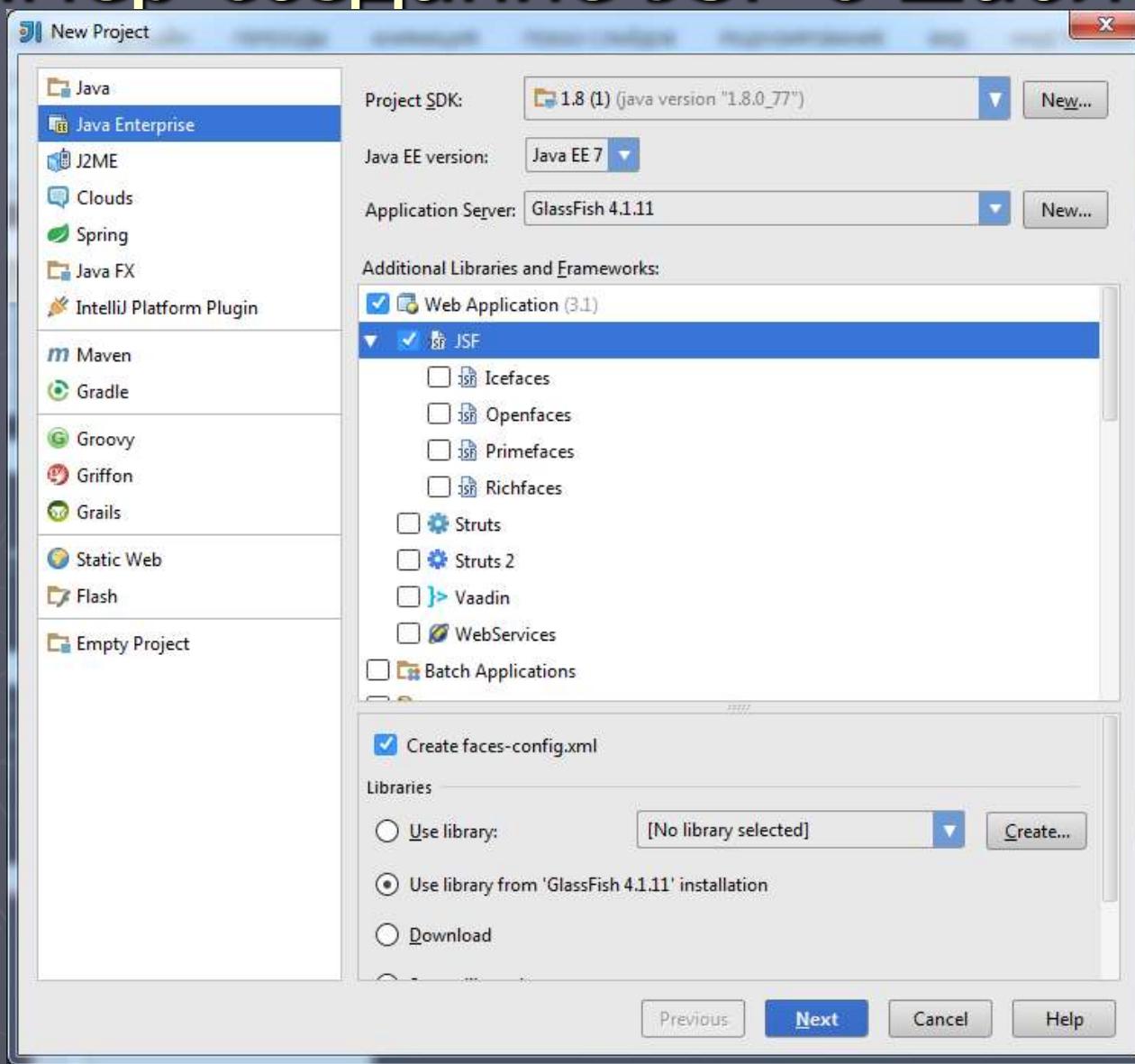
Тег	Описание
<f:validator>	Добавляет компоненту произвольный валидатор или использует определенный валидатор длины или регулярных выражений
<f:validatorDoubleRange>	
<f:validatorLongRange>	
<f:validateLength>	
<f:validateRegex>	
<f:validateBean>	Определяет группы проверки для Bean Validation (см. главу 3)
<f:loadBundle>	Загружает ресурсы
<f:selectItem>	Определяет один или несколько элементов для выбора
<f:selectItems>	одного или нескольких компонентов
<f:ajax>	Включает поведение AJAX

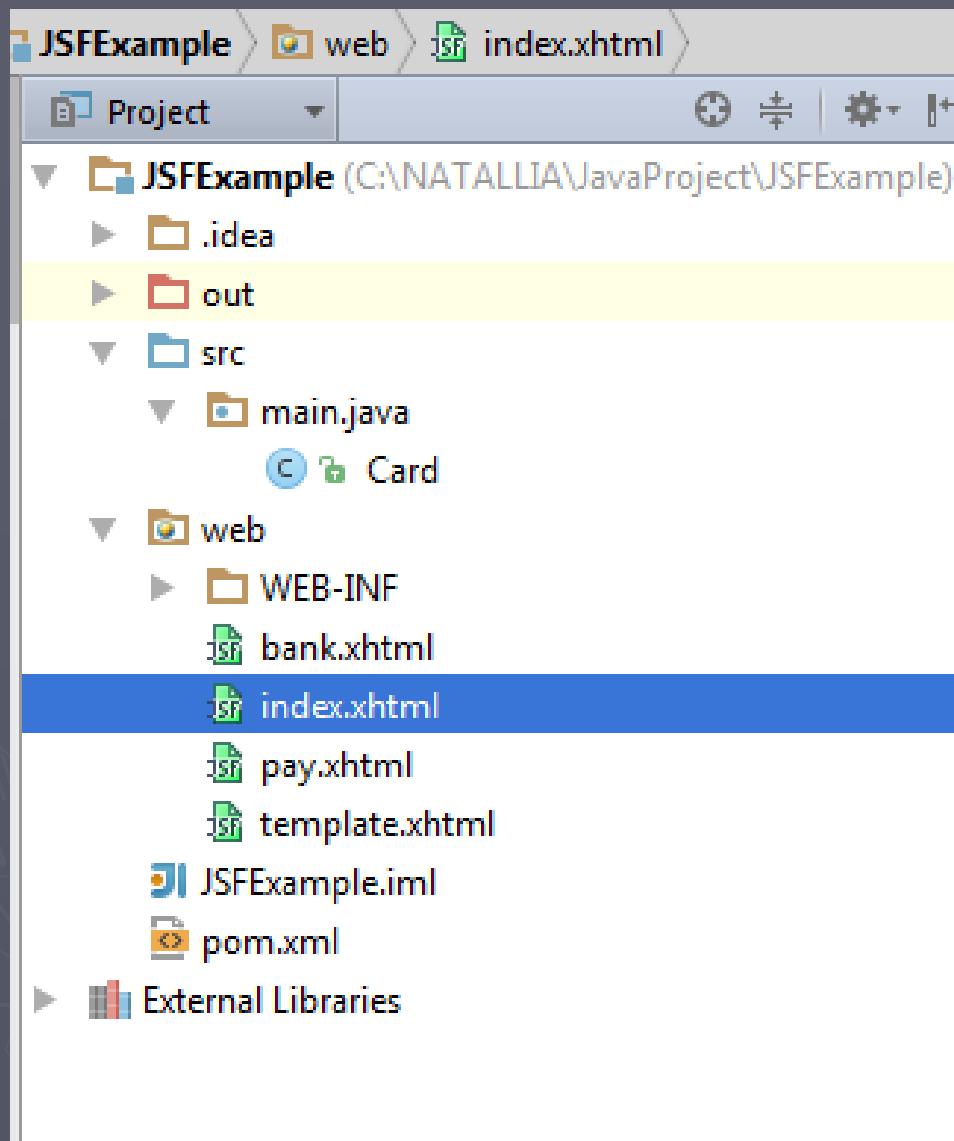
Теги шаблонов JSF

- ▶ позволяет определить макет в файле шаблона, который может быть использован для всех страниц в веб-приложении.
 - определяет области (<ui:insert>), где содержимое может быть заменено.
 - клиентская страница использует теги <ui:component>, <ui:composition>, <ui:fragment> или <ui:decorate> для встраивания

Тег	Описание
<ui:composition>	Определяет композицию, которая может использовать шаблон. Несколько композиций могут применять один и тот же шаблон
<ui:component>	Создает компонент
<ui:debug>	Собирает информацию об отладке
<ui:define>	Определяет содержимое, которое вставлено на страницу с помощью шаблона
<ui:decorate>	Позволяет декорировать содержимое на странице
<ui:fragment>	Добавляет фрагмент страницы
<ui:include>	Инкапсулирует и повторно использует содержимое среди нескольких страниц XHTML
<ui:insert>	Вставляет содержимое в шаблон
<ui:param>	Передает параметры во включенный файл (с использованием тега <ui:include> или шаблона)
<ui:repeat>	Проходит по всему списку объектов
<ui:remove>	Удаляет содержимое страницы

Пример создания JSF с шаблонами





A. Создание managed bean

```
@ManagedBean  
@RequestScoped  
public class Card implements Serializable {  
    private long cardId;  
    private String name; ← Поля  
    private int passw;  
  
    public long getCardId() {  
        return cardId;  
    }  
    public void setCardId(long cId) {  
        this.cardId = cId;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public int getPassw() {  
        return passw;  
    }  
    public void setPassw(int passw) {  
        this.passw = passw;  
    }  
}
```

Аннотация регистрирует класс Card как JSF ресурс

Аннотация – контейнер должен создавать новый instance бина для каждого запроса пользователя

Сгенерировать
get и set

faces-config.xml в WEB-INF
- переопределит
аннотированные значения

Б. Создание JSF страниц

namespace для JSF

```
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
xmlns:f="http://xmlns.jcp.org/jsf/core">
```

```
<!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:ui="http://java.sun.com/jsf/faclets">
<head>
    <title><ui:insert name="title">Bank Portal</ui:insert></title>
</head>
<body>
    header and footer будут сохраняться
    на каждой странице, меняется только контент
    <div id="header">
        <ui:insert name="header">
            <h1>Welcome to bank WWW</h1>
        </ui:insert>
    </div>

    <div id="content">
        <ui:insert name="content">
            The content of bank  </ui:insert>
    </div>

    <div id="footer">
        <ui:insert name="footer">
            Copyright © 2017, All rights reserved
        </ui:insert>
    </div>
</body>
</html>
```



template.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://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:f="http://xmlns.jcp.org/jsf/core">

<ui:composition template="template.xhtml"> ← Связывает страницу с шаблоном
    <ui:define name="content">

        <h:form>

            <h:outputText value="Enter your name:<br/>" />
            <h:inputText value="#{card.name}" title="name" id="name" /> <br/><br/>

            <h:outputText value="Enter card number:<br/>" />
            <h:inputText value="#{card.cardId}" title="id" id="cardId" /> <br/><br/>

            <h:outputText value="Enter password:<br/>" />
            <h:inputText value="#{card.passw}" title="passw" id="passw" /> <br/><br/>
            <h:commandButton action="pay" value = "pay" /><br/>

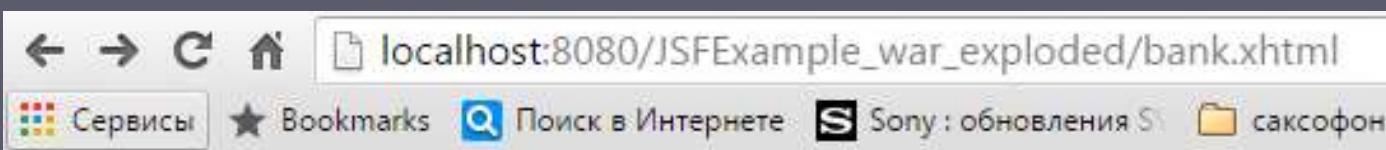
        </h:form>

    </ui:define>
</ui:composition>
</html>
```

Связь со свойствами Card бина

После обработки формы, выполняется навигация на JSF с назначением **pay**

JSF реализация не найдет header footer, и реализация будет взята из **template**



Welcome to bank WWW

Enter your name:

Enter card number:

Enter password:

Copyright © 2017. All rights reserved

По умолчанию включается
FacesServlet, который знает как
интерпретировать все JFS теги

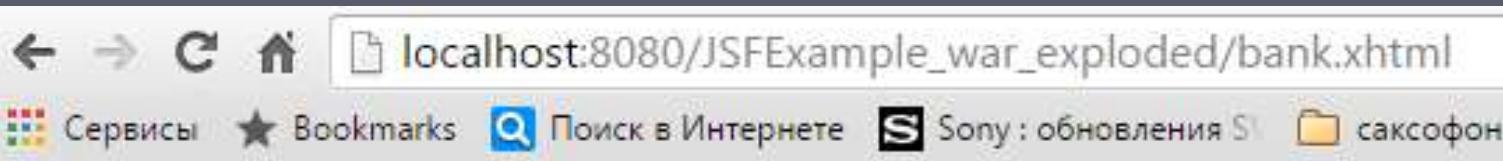


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
  xmlns:f="http://xmlns.jcp.org/jsf/core">

<ui:composition template="template.xhtml">
<ui:define name="content">

<h:outputText value="#{card.name}"/>, Thank you for payment with card nubr
<h:outputText value="#{card.cardId}" /><br/><br/>

</ui:define>
</ui:composition>
</html>
```



```
<h:commandButton value="Do Something" action="#{card.doSomething}"
```

В. Добавление валидаторов и преобразователей

```
<h:inputText value="#{card.name}"  
             title="name" id="name" > <br/><br/>  
<f:validateLength minimum="2" />  
</h:inputText>
```

Вставка стандартных
валидаторов JSF

Валидаторы действуют как первый уровень
контроля, проверяя значения компонентов
пользовательского интерфейса перед
тем, как они будут обработаны bean.

```
<h:inputText value="#{cardController.card.number}" required="true"/>
```

тег требует, чтобы было введено значение
в текстовом поле ввода

```
<h:inputText value="#{cardController.card.name}" required="true">  
    <f:validateLength minimum="2" maximum="20"/>  
</h:inputText>
```

```
<h:inputText value="#{cardController.card.sum}">  
    <f:validateLongRange minimum="1" maximum="500"/>  
</h:inputText>
```

имя владельца карты составляет от 2 до 20 символов в длину,

А сумма — от 1 до 500

Пользовательские валидаторы

1) Создать класс

```
public class CardValidator implements Validator {  
    @Override  
    public void validate(FacesContext facesContext,  
        UIComponent uiComponent, Object o) throws ValidatorException {  
    }  
}
```

2) Зарегистрировать в faces-config.xml или аннотация

```
<validator>  
    <validator-id>CValidator</validator-id>  
    <validator-class>main.java.CardValidator</validator-class>  
</validator>
```

3) Использовать

```
<h:outputText value="Enter card number:"/><br/>
<h:inputText value="#{card.cardId}" title="id" id="cardId"> <br/>
<f:validator id="CValidator"/>
</h:inputText>
```

```
import javax.faces.validator.FacesValidator;
```

```
@FacesValidator("idValidator")
public class IDValidator implements javax.faces.validator.Validator {

    private Pattern pattern;
    private Matcher matcher;

    @Override
    public void validate(FacesContext context, UIComponent component,
                         Object value) throws ValidatorException {

        String componentValue = value.toString();
        pattern = Pattern.compile("(?=[-0-9хХ]{13}$)");
        matcher = pattern.matcher(componentValue);
        if (!matcher.find()) {
            String message = MessageFormat.format("{0} неверный формат id",
                                                   componentValue);
            FacesMessage facesMessage =
                new FacesMessage(FacesMessage.SEVERITY_ERROR, "___", "___");
            throw new ValidatorException(facesMessage);
        }
    }
}
```

реализует интерфейс Validator
и добавляет логику проверки в метод validate()

сообщение добавляется в контекст и генерируется исключение

преобразовывать входное
значение к числу (по умолчанию),
валюте или процентам

Преобразователи

```
<h:inputText value="#{cardController.card.sum}">  
    <f:convertNumber currencySymbol="$" type="currency"/>  
</h:inputText>
```

необходимо вложить преобразователь
внутрь любого из тегов ввода или
вывода

```
<h:inputText value="#{cardController.card.expered}">  
    <f:convertDateTime pattern="MM/yy"/>  
</h:inputText>
```

может конвертировать даты в
различных форматах

Пользовательские преобразователи

класс, реализующий интерфейс
javax.faces.convert.Converter

Зарегистрировать — объявление
преобразователя в файле faces-config.xml или
использование аннотации @FacesConverter.

```
@FacesConverter("rubConverter")
public class RubConverter implements Converter {

    @Override
    public Object getAsObject(FacesContext context,
        UIComponent component, String
        value) {
        return value;
    }

    @Override
    public String getAsString(FacesContext ctx,
        UIComponent component, Object value) {
        float amountInDollars = Float.parseFloat(value.toString());
        double amountInRub = amountInDollars * 2;
        DecimalFormat df = new DecimalFormat("###,##0.##");
        return df.format(amountInRub);
    }
}
```

преобразует строковое значение
компоненты пользовательского интерфейса
в соответствующий поддерживаемый тип
возвращает новый
экземпляр

преобразует объект предоставленного типа в строку,
которая будет отрисована с помощью языка разметки (на-
пример, XHTML)

```
<h:outputText value="#{card.sum}"> // конвертируем в руб
  <f:converter converterId="rubConverter"/>
</h:outputText>
```

```
<h:outputText value="#{card.sum}" converter="rubConverter"/>
```

Неявные объекты JSF

- ▶ Неявные объекты — это специальные идентификаторы, которые сопоставляются с конкретными часто используемыми объектами.
Страница имеет к ним доступ и может использовать их без необходимости их явного объявления или инициализации

```
# {resources [ 'foto.gif' ] }
```

Неявный объект	Описание	Возвращаемый тип
application	Представляет среду веб-приложения. Используется для получения конфигурационных параметров приложения	Object
applicationScope	Преобразует имена глобальных атрибутов приложения в их значения	Map
component	Указывает на текущий компонент	UIComponent
cc	Указывает на текущий составной компонент	UIComponent
cookie	Определяет объект типа Map, содержащий имена cookies (являющиеся ключами) и объекты типа Cookie	Map
facesContext	Указывает на объект типа FacesContext для текущего запроса	FacesContext

Неявный объект	Описание	Возвращаемый тип
flash	Представляет объект, использующий флеш (подробнее об областях видимости читайте в главе 11)	Object
header	Преобразует имя HTTP-заголовка в значение заголовка типа String	Map
headerValues	Преобразует имя HTTP-заголовка в набор всех значений заголовка типа String[]	Map
initParam	Преобразует имена параметров инициализации контекста к значениям типа String	Map
param	Преобразует имена параметров запроса к одному значению параметра типа String	Map
paramValues	Преобразует имена параметров запроса к набору всех значений параметра типа String[]	Map
request	Представляет объект запроса HTTP	Object
requestScope	Преобразует имена атрибутов запроса к их значениям	Map
resource	Представляет объект ресурса	Object
session	Представляет объект сеанса HTTP	Object
sessionScope	Преобразует имена атрибутов сеанса к их значениям	Map
view	Представляет текущее представление	UIViewRoot
viewScope	Преобразует имена атрибутов представления к их значениям	Map

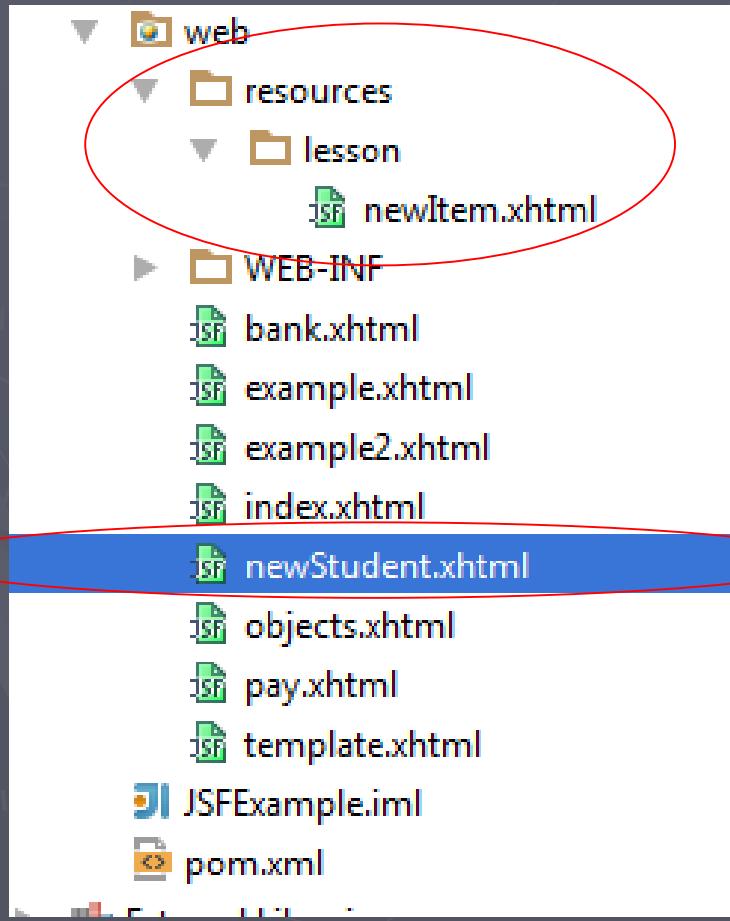
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:c="http://xmlns.jcp.org/jsp/jstl/core">
<h:body>
    <h1>Неявные объекты</h1>
    <hr/>
    <h3>Локаль</h3>
    <h:outputText value="#{view.locale}" />
    <h3>headerValues</h3>
    <c:forEach var="parameter" items="#{headerValues}">
        <h:outputText value="#{parameter.key}" /> =
        <c:forEach var="value" items="#{parameter.value}">
            <h:outputText value="#{value}" escape="false"/><br />
        </c:forEach>
    </c:forEach>
    <hr />
    <h2 style="text-align: center;">Неявные объекты</h2>
    <hr style="border-top: 1px solid black; margin-top: 10px;" />
    <h3>Локаль</h3>
    <hr style="border-top: 1px solid black; margin-top: 10px;" />
    <h3>headerValues</h3>
    <hr style="border-top: 1px solid black; margin-top: 10px;" />
    host = localhost:8080
    connection = keep-alive
    upgrade-insecure-requests = 1
    user-agent = Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36
    accept = text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
    accept-encoding = gzip, deflate, sdch
    accept-language = ru_RU,ru;q=0.8,en-US;q=0.6,en;q=0.4
    cookie = JSESSIONID=abe9aa646a974c525977eef0fd2d; __AntiXsrfToken=5027de3aacc84313bb4f3f50df1ae43e; JSESSIONID=85a53d4ff3
    hi=treeForm:tree:applicationServer
    <hr style="border-top: 1px solid black; margin-top: 10px;" />
```

Составные компоненты

- ▶ страница XHTML(рассматривается как реальный компонент, который может поддерживать работу с валидаторами, преобразователями и слушателями), которая содержит компоненты может быть использована ее в качестве компонента на других страницах.
- ▶ Внутри составного компонента разрешено использовать любую допустимую разметку, в том числе шаблоны.
- ▶ Составные компоненты обрабатываются как ресурсы и должны находиться в пределах стандартных каталогов resources.

Тер	Описание
<composite:interface>	Описывает контракт компонента
<composite:implementation>	Определяет реализацию компонента
<composite:attribute>	Определяет атрибут, который может быть задан экземпляру компонента. Их может не быть вообще либо быть сразу несколько внутри раздела <composite:interface>
<composite:facet>	Указывает, что этот компонент поддерживает фасет
<composite:insertFacet>	Используется в разделе <composite:interface>. Вставленный фасет будет отрисован в компоненте
<composite:insertChildren>	Применяется в разделе <composite:interface>. Любые компоненты-потомки или шаблоны внутри компонента будут вставлены в отрисовываемый результат
<composite:valueHolder>	Указывает, что компонент, чей контракт определен в разделе <composite:interface>, в который этот компонент вложен, предоставляет реализацию типа ValueHolder
<composite:renderFacet>	Используется в разделе <composite:interface> для отрисовки фасета
<composite:extension>	Применяется в разделе <composite:interface> для того, чтобы включить XML-содержимое, не определенное в спецификации JSF
<composite:editableValueHolder>	Указывает, что компонент, чей контракт определен в разделе <composite:interface>, в который этот компонент вложен, предоставляет реализацию типа EditableValueHolder

Тер	Описание
<composite:clientBehavior>	Определяет контракт для поведений, которые могут изменить отрисованное содержимое компонента
<composite:actionSource>	Указывает, что компонент, чей контракт определен в разделе <composite:interface>, в который этот компонент вложен, предоставляет реализацию типа ActionSource





newItem.xhtml

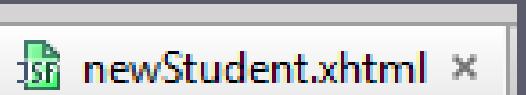
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:composite="http://xmlns.jcp.org/jsf/composite">
```

```
<composite:interface> ←
    <composite:attribute name="item" required="true" />
    <composite:attribute name="style" required="false" />
</composite:interface>
```

Интерфейс – отправная
точка для компонента

```
<composite:implementation> ←
    <h:panelGrid columns="2" style="#{cc.attrs.style}">
        <h:outputLabel value="Name : "/>
        <h:inputText value="#{cc.attrs.item.name}" />
        <h:outputLabel value="Number : "/>
        <h:inputText value="#{cc.attrs.item.number}" />
        <h:outputLabel value="Other : "/>
        <h:inputTextarea value="#{cc.attrs.item.description}"
                         cols="20" rows="5" />
    </h:panelGrid>
</composite:implementation>
</html>
```

Реализация
компонента



```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:pnv="http://xmlns.jcp.org/jsf/composite/lesson">
<h:head>
    <title>Add information </title>
</h:head>
<h:body>
    <h1>Enrolled</h1>
    <hr/>
    <h:form>
        <pnv:newItem item="#{itemController.student}" />
        <h:panelGrid columns="2">
            <h:outputLabel value="Department : "/>
            <h:inputText value="#{itemController.student.dep}" />
            <h:outputLabel value="Level : "/>
            <h:inputText value="#{itemController.student.level}" />
            <h:outputLabel value="Group : "/>
            <h:selectBooleanCheckbox value="#{itemController.student.group}" />
        </h:panelGrid>
        <h:commandButton value="Submit" action="#{itemController.doCreateStudent}" />
    </h:form>
    <hr/>
</h:body>
</html>
```

Объявляем библиотеку

Включаем компонент и определим параметр



localhost:8080/JSFExample_war_exploded/newStudent.xhtml

Сервисы

Bookmarks



Поиск в Интернете



Sony : обновления



саксофон



KVITKO

Enrolled

Name :

Number :

Other :

Department :

Level :

Group :

Submit

Резюме

- ▶ Были рассмотрены графический аспект и динамический JSF (тэги и навигация)
- ▶ Управляемый бин лежит в основе JSF
- ▶ Существуют стандартные и пользовательские перобразователи и валидаторы