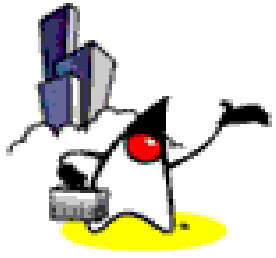
Компоненты для работы с данными



Таблица, связанная с источником данных

- Используется для отображения реляционных данных в виде таблицы

Quantity	Title	Price	
1	Web Servers for Fun and Profit	$40.75	Remove Item
1	Java Intermediate Bytecodes	$30.95	Remove Item
2	My Early Years: Growing up on *7	$30.75	Remove Item
3	Web Components for Web Developers	$27.75	Remove Item
3	Duke: A Biography of the Java Evangelist	$45.00	Remove Item
1	From Oak to Java: The Revolution of a Language	$10.75	Remove Item
Subtotal:$362.20			

Update Quantities

Компоненты UIData и UIColumn

- Компонент *UIData* поддерживает привязку к коллекции объектов данных
  - В цикле проходит по всем записям источника данных
  - Стандартный рендерер Table отображает данные в виде HTML-таблицы с помощью элемента *h:dataTable*
- Компонент *UIColumn* представляет столбец таблицы

Элемент h:dataTable

- Совмещает компонент UIData с рендерером Table
  - Поддерживает генерацию таблиц
  - CSS-стили
  - Произвольные модели данных

Атрибуты элемента *h:dataTable*

- first, rows – отображаемое “окно”
- value – указывает данные, включаемые в таблицу
  - Список бинов
  - Массив бинов
  - Единственный бин
  - Объект javax.faces.model.DataModel
  - Объект java.sql.ResultSet
  - Объект javax.servlet.jsp.jstl.sql.ResultSet
  - Объект javax.sql.RowSet
- var – переменная, в которую помещается текущая запись (строка данных)

Модель данных

- У всех источников данных для компонентов UIData есть оболочка типа DataModel
- Реализация JavaServer Faces по умолчанию создает оболочку DataModel для данных любого допустимого типа
- Типы моделей данных:
  - ArrayDataModel
  - ListDataModel
  - ResultDataModel
  - ResultSetDataModel
  - ScalarDataModel

Ячейка таблицы

- В ячейке таблицы может размещаться любой JSF-компонент
  - h:outputText, h:selectOneMenu, h:selectOneRadio, h:graphicImage, h:gridPanel
- JSF-компонент в ячейке работает так же, как и компонент вне таблицы
  - с ним можно связать обработчик события
  - вывод по условию, указываемому в атрибуте *rendered*

Пример: Таблица клиентов

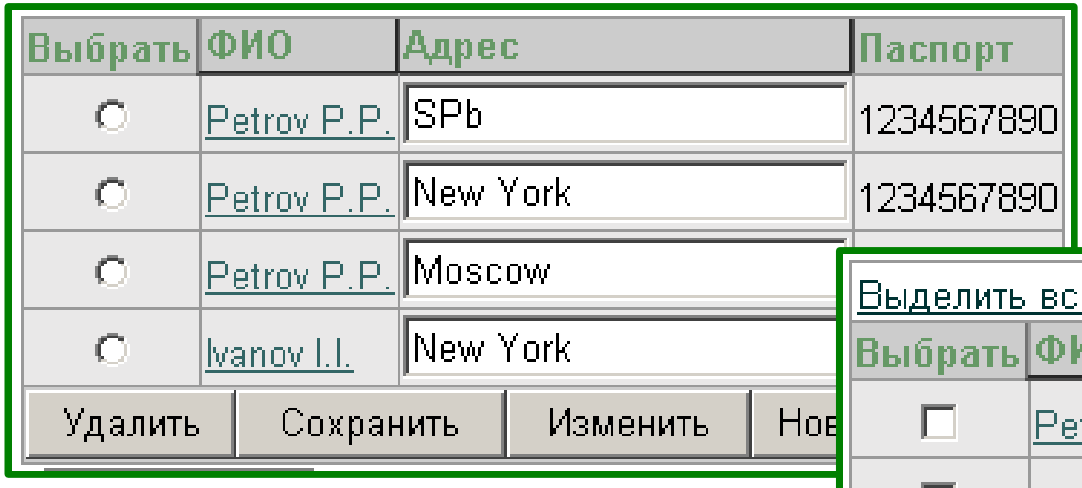
```
<h:dataTable value="#{clientDao.allClients}" var="client">
    <h:column headerClass="header">
        <f:facet name="header">
            <h:outputText value="ФИО"/>
        </f:facet>
        <h:outputText value="#{client.fio}"/>
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText value="Адрес"/>
        </f:facet>
        <h:outputText value="#{client.address}"/>
    </h:column>
</h:dataTable>
```

Недостатки таблиц в JSF

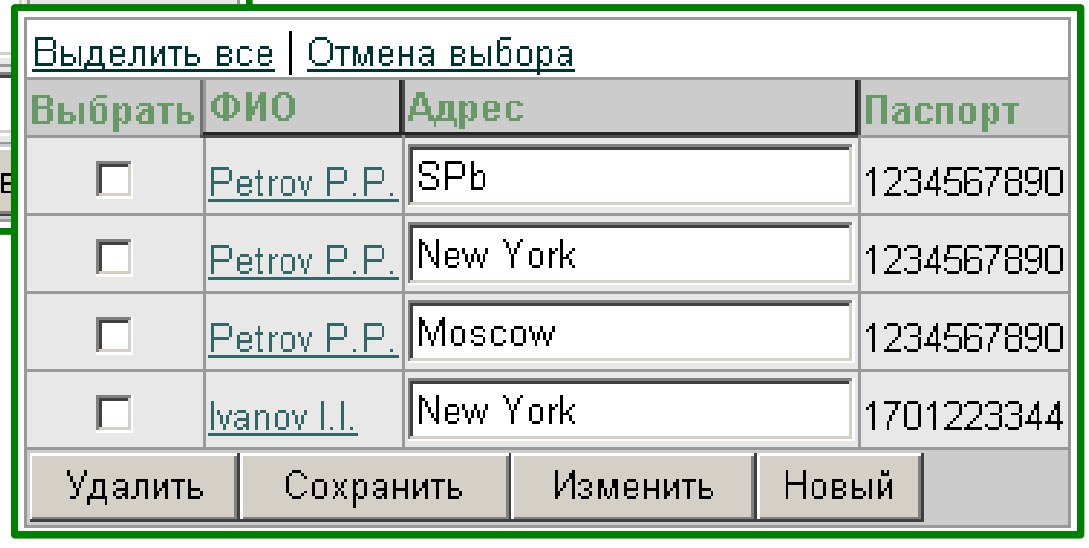
- Нет встроенной сортировки строк
- Нет встроенных средств выбора строк
  - Групповые операции с выделенными строками
- Решение: использовать стороннюю библиотеку компонентов
  - Apache Trinidad (ранее – Oracle ADF Faces)
  - Sun Visual JSF
  - Woodstock (проект на dev.java.net)
  - ICEfaces
  - ...

Таблица данных в Trinidad (1)

- Выбор строк:
  - `<tr:table ... rowSelection="single|multiple">`



`rowSelection="single"`



`rowSelection="multiple"`

Таблица данных в Trinidad (2)

- Сортировка по столбцам:
  - `<tr:column sortProperty="address" sortable="true">`

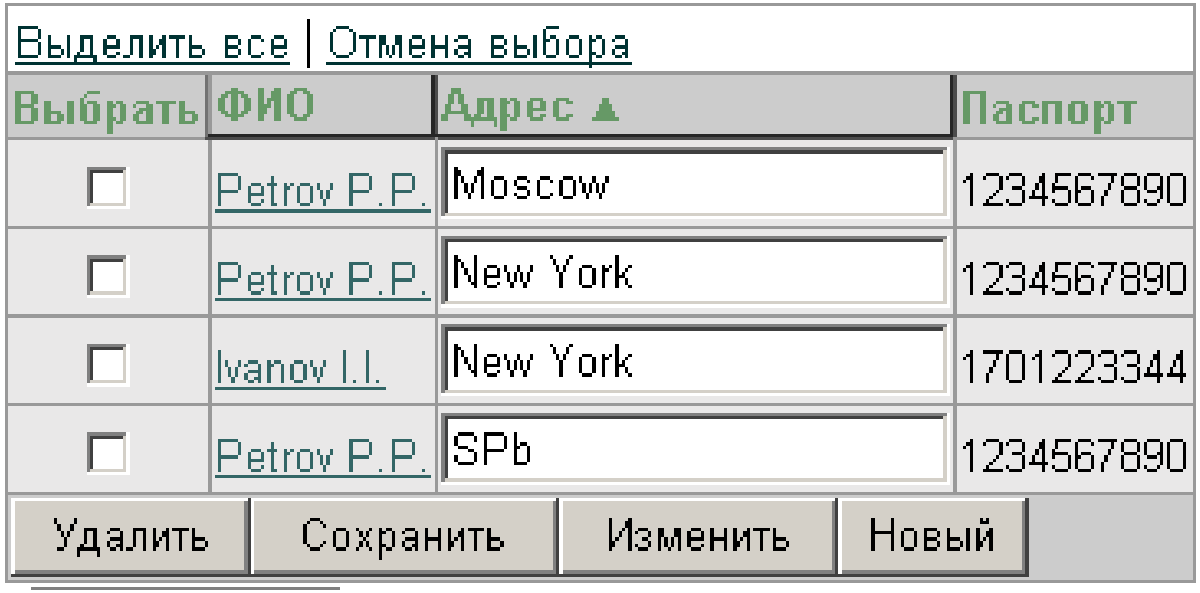


Таблица данных в Trinidad (3)

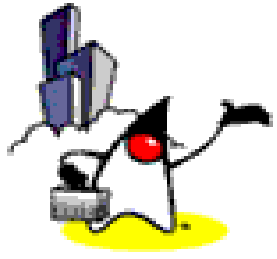
- Вывод подробных сведений о строке таблицы (например, в виде вложенной таблицы)
  -
```
<f:facet name="detailStamp">
    <tr:outputText value="#{client.fio}
        имеет идентификатор #{client.id}"/>
</f:facet>
```

Выделить все | Отмена выбора

Выбрать	Сведения	ФИО	Адрес ▲	Паспорт	ФИО	Адрес ▲	Паспорт
☐	▼Скрыть	Petrov P.P.	Moscow	1234567890	Petrov P.P.	Moscow	1234567890
Petrov P.P. имеет идентификатор 4							
☐	▶Показать	Petrov P.P.	New York	1234567890	Petrov P.P.	New York	1234567890
☐	▼Скрыть	Ivanov I.I.	New York	1701223344	Ivanov I.I.	New York	1701223344
Ivanov I.I. имеет идентификатор 5							
☐	▶Показать	Petrov P.P.	SPb	1234567890	Petrov P.P.	SPb	1234567890

Удалить | Сохранить | Изменить | Новый

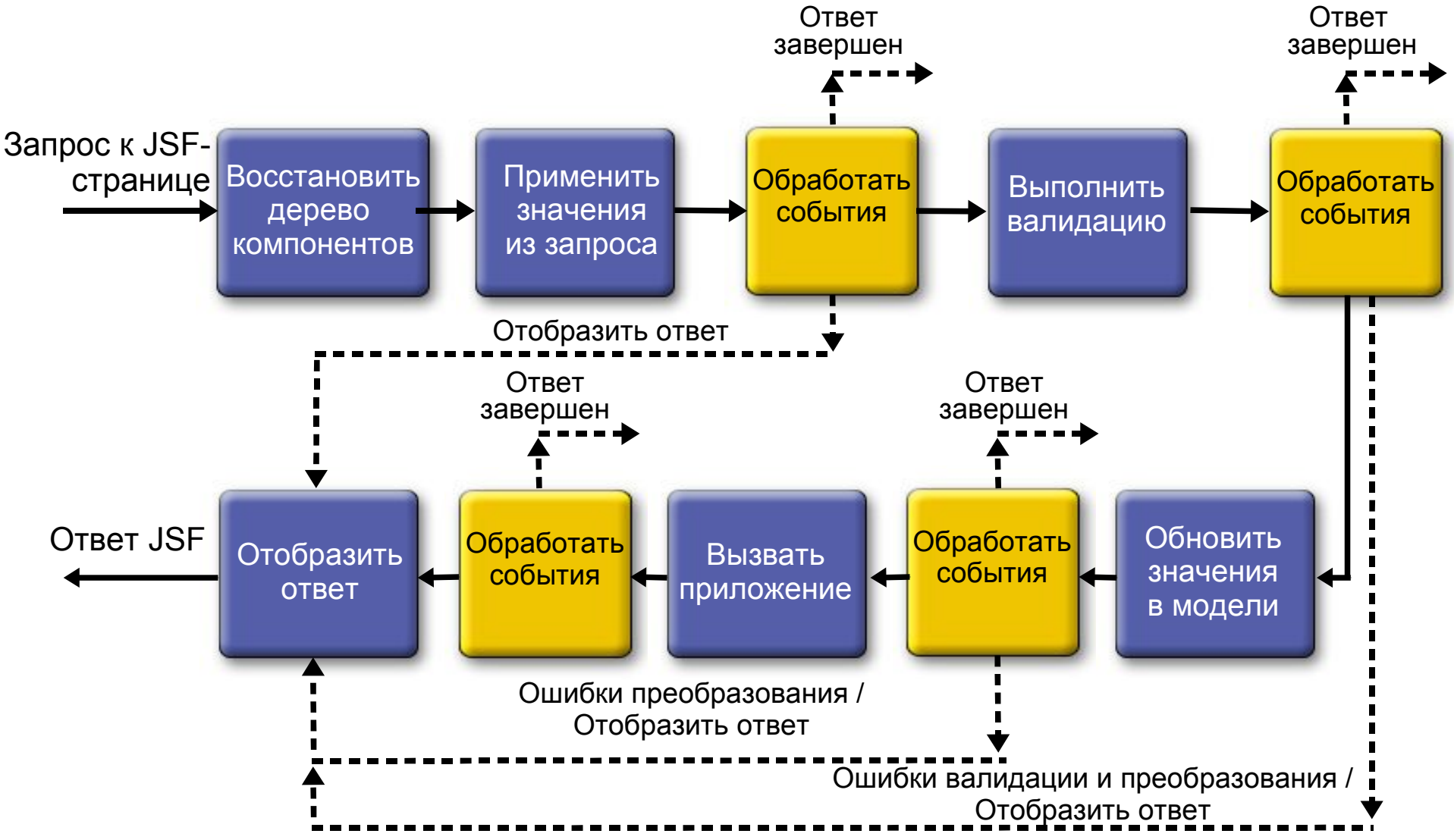
Цикл обработки запроса в JSF



Цикл обработки запроса

- JSF-страница представляется в виде дерева компонентов UI, называемого view (представление)
- Цикл начинается, когда клиент запрашивает страницу
- В рамках цикла реализация JSF должна построить представление с учетом состояния, сохраненного при повторном запросе
- Когда клиент выполняет повторный запрос, реализация JSF должна выполнить
  - валидацию и
  - преобразование данных

Цикл обработки запроса



Обработка запроса

- Два типа запросов
  - Начальный и повторный
- Начальный запрос
  - Пользователь запрашивает страницу впервые
  - Выполняются только шаги “восстановить дерево компонентов” и “отобразить ответ”
- Повторный запрос
  - Пользователь отправляет данные формы со страницы, полученной в результате начального запроса
  - Выполняются все шаги обработки запроса

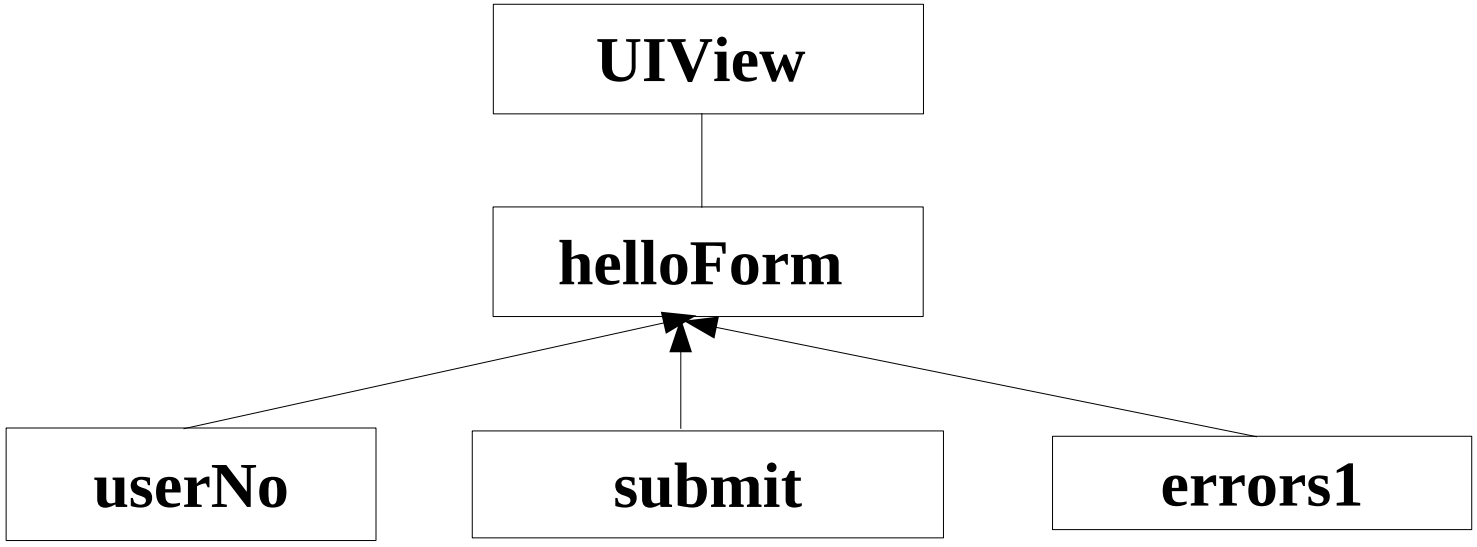
Пример страницы greeting.jsp

```
<HTML>
 <HEAD> <title>Hello</title> </HEAD>
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <body bgcolor="white">
 <f:view>
 <h:form id="helloForm" >
  <h2>Hi. My name is Duke.  I'm thinking of a number from
  <h:outputText value="#{UserNumberBean.minimum}"/> to
  <h:outputText value="#{UserNumberBean.maximum}"/>.  Can you guessit?</h2>
  <h:graphicImage id="waveImg" url="/wave.med.gif" />
  <h:inputText id="userNo" value="#{UserNumberBean.userNumber}">
              <f:validateLongRange minimum="0" maximum="10" />
  </h:inputText>
  <h:commandButton id="submit" action="success" value="Submit" />
  <p>
  <h:message style="color: red; font-family: 'New Century Schoolbook', serif;
     font-style: oblique; text-decoration: overline" id="errors1" for="userNo"/>
 </h:form>
 </f:view>
</HTML>
```

Шаг 1: Восстановить дерево компонентов (представление)

- Запрос проходит через сервлет-контроллер *FacesServlet*, который выделяет из запроса ID представления (название JSP-страницы).
- Контроллер использует этот ID для поиска представления и всех его компонентов.
  - Если представление еще не существует, то контроллер создает его.
  - Если представление уже существует, то контроллер использует его.
- Контроллер подключает к представлению обработчики событий, валидаторы и конвертеры
- Контроллер сохраняет представление в FacesContext

Пример: Дерево компонентов (пред-ставление) страницы greeting.jsp



Шаг 2: Применить значения из запроса

- Каждый компонент выделяет свое новое значение из параметров запроса с помощью своего метода decode()
- В методе decode() значение преобразуется к правильному типу и сохраняется в компоненте
  - Например, значение компонента userNo на странице greeting.jsp преобразуется из типа String в тип Integer
  - Ошибки преобразования запоминаются в FacesContext

Шаг 3: Выполнить валидацию

- Реализация JSF опрашивает все зарегистрированные для компонентов валидаторы
- Если есть ошибки валидации
  - Сообщения об ошибках запоминаются в FacesContext
  - Выполняется переход к шагу 6: “Отобразить ответ”
- Пример
  - Значение userNo должно быть от 1 до 10

Шаг 4: Обновить значения в модели

- Реализация JSF проходит по дереву компонентов и синхронизирует состояние компонентов и модели
  - Изменяется свойство управляемого бина, на которое указывает атрибут value компонента
  - Только для компонентов типа EditableValueHolder (в т.ч. все UIInput)
  - Значение компонента преобразуется к типу свойства модели
- Пример в greeting.jsp
  - Свойству UserNumberBean.userNumber присваивается значение компонента userNo

Шаг 5: Вызвать приложение

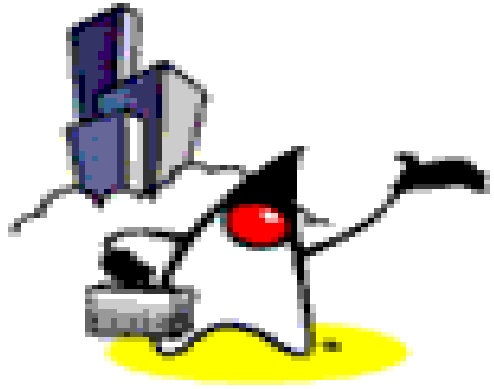
- Выполняются обработчики событий уровня приложения (например, для обработки формы или ссылки на другую страницу)

Шаг 5: Пример в GuessNumber

- В странице greeting.jsp есть одно событие уровня приложения, связанное с компонентом UICommand
- Реализация ActionListener по умолчанию получает строку результата "success" из атрибута action компонента
- Обработчик передает строку результата обработчику навигации (NavigationHandler) по умолчанию
- NavigationHandler определяет следующую отображаемую страницу, сопоставляя строку результата и определенные правила навигации
- Реализация JSF устанавливает в качестве представления ответа (response view) представление следующей отображаемой страницы

Шаг 6: Отобразить ответ

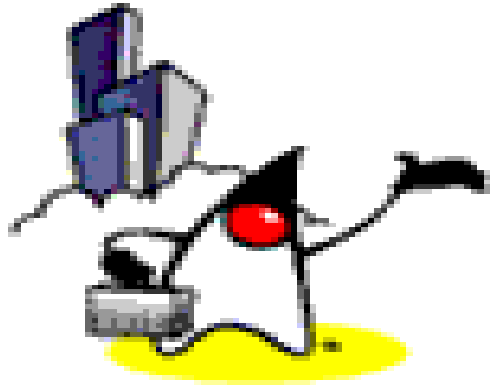
- Реализация JSF проходит по сохранен-ному в FacesContext дереву компонентов и вызывает у всех компонентов метод encode()
  - Создание правильной разметки по дереву компонентов
  - Если на предыдущих шагах были ошибки, то отображается исходная страница с сохраненными сообщениями об ошибках
- Состояние ответа сохраняется так, чтобы оно было доступно на шаге “восстановление дерева компонентов” при обработке последующих запросов



Изменение последовательности обработки запроса

Атрибуте immediate

- Может применяться для компонентов:
  - *UICommand*
  - *UIInput*
- Каждому типу компонентов соответствует особое поведение атрибута

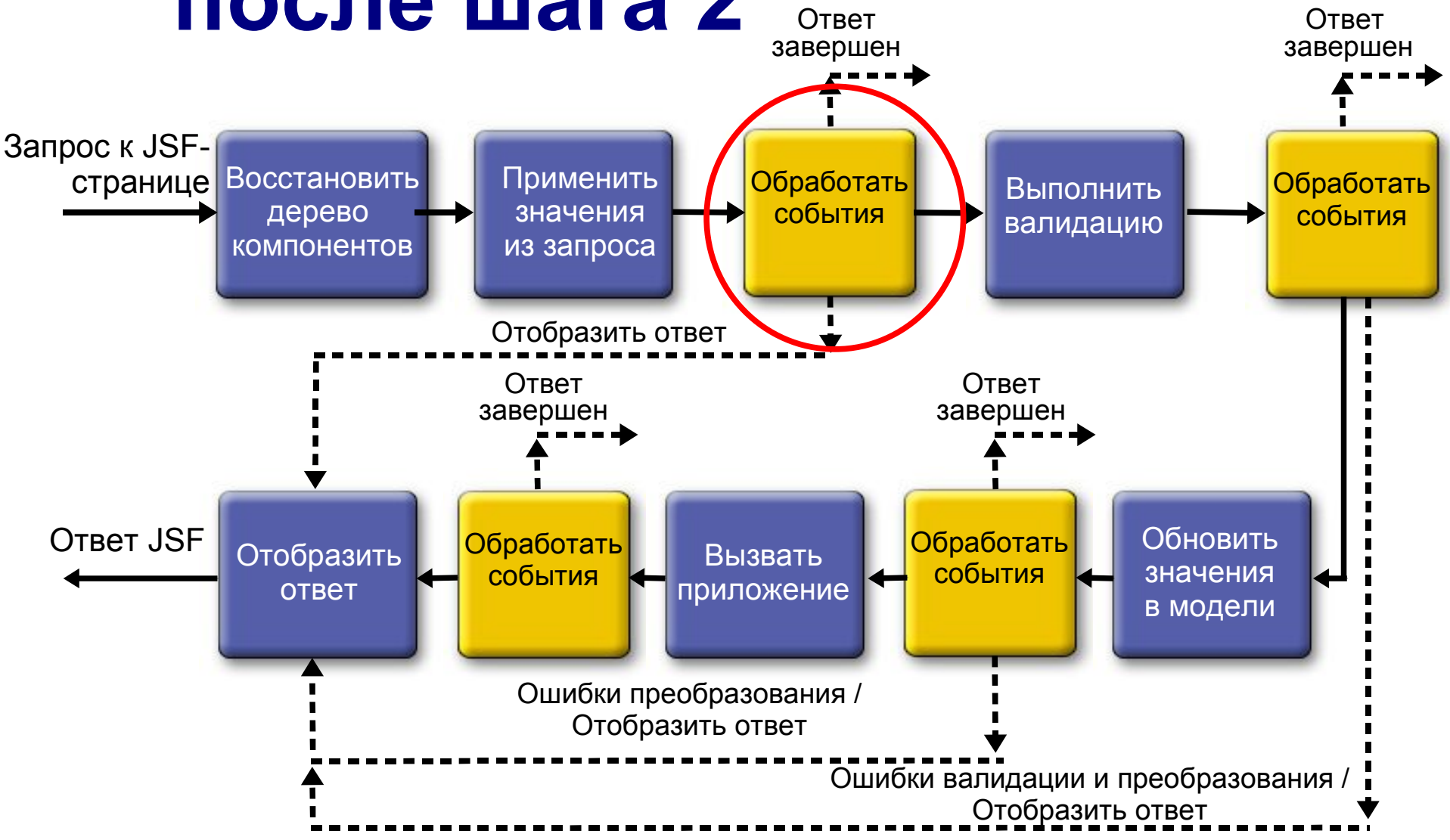


Атрибут immediate для компонентов UICommand

Атрибут immediate для компонентов UICommand

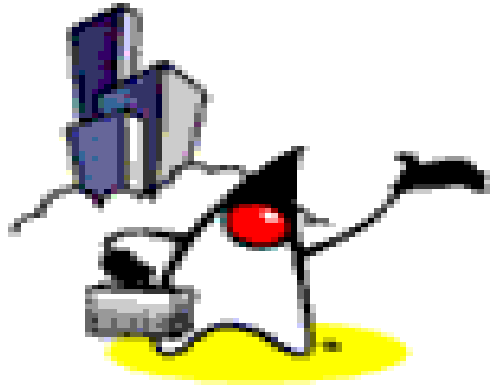
- Обработчик действия вызывается в конце шага 2: “Применить значения из запроса”
  - В обычном режиме он вызывается на шаге 5: “Вызов приложения”
  - Обработчик действия вызывается без валидации (шаг 3) и обновления значений в модели (шаг 4)
  - Обработчик действия может затем перейти напрямую к шагу 6: “Отобразить ответ”

Вызов обработчика действия после шага 2



Примеры использования

- Необязательные шаги в мастере (например, кнопку "Пропустить" для перехода к следующему окну)
- Пользователь хочет отменить ввод данных в форму
  - В этом случае нет смысла проводить валидацию и изменять свойства модели. Все что нужно сделать – перейти к следующей операции

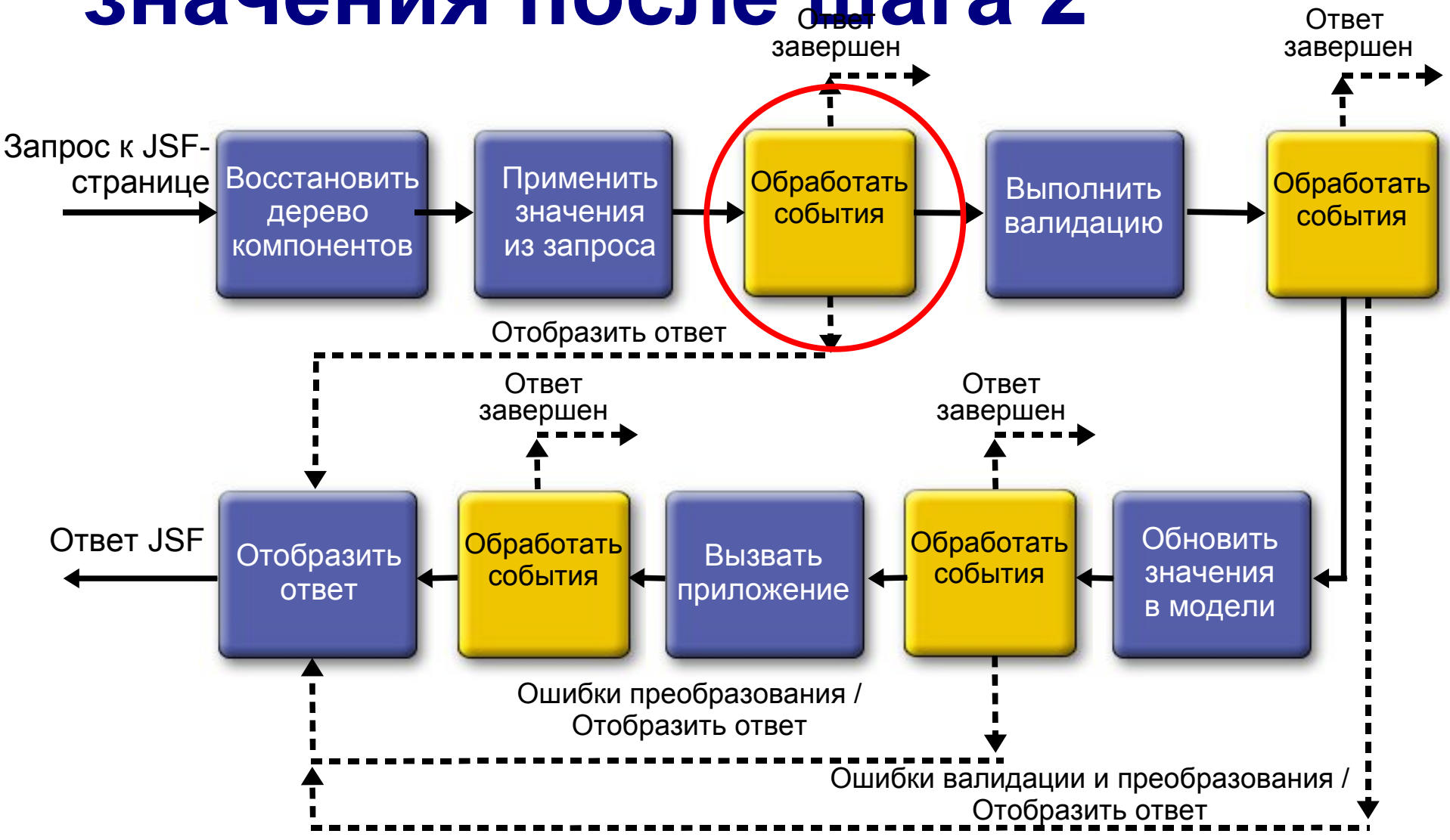


Атрибут immediate для компонентов UIInput

Атрибут immediate для компонентов UIInput

- Преобразование/валидация значения компонента происходит на шаге 2: “Применить значения из запроса”
  - В обычном режиме он вызывается на шаге 3: “Выполнить валидацию”
  - События изменения значения (ValueChangeEvent) также помещаются в очередь на шаге 2
- Конвертеры/валидаторы для не-"immediate" компонентов выполняются как обычно, на шаге 3

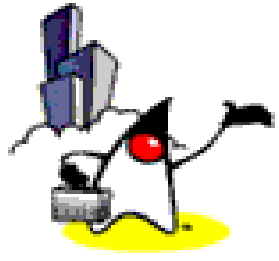
Вызов обработчика изменения значения после шага 2



Пример использования

- Пусть на странице есть выпадающий список и текстовое поле (и то, и другое – компоненты типа UIInput).
- При изменении значения списка требуется устанавливать и валидировать значение текстового поля.
- Для этого можно сделать список "immediate", и в его обработчике типа ValueChangeListener устанавливать значение текстового поля.

Обработка событий



Модель обработки событий

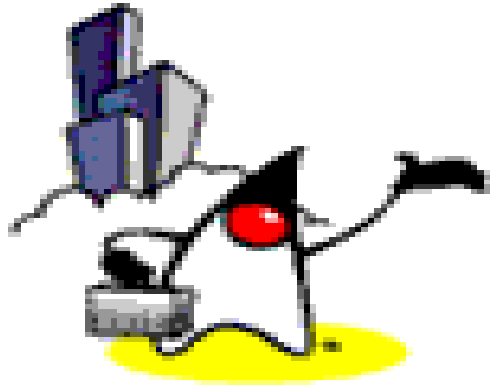
- Аналогична модели событий, принятой в JavaBeans
  - Интерфейсы обработчиков и классы событий
  - Объект события определяет компонент, вызвавший событие, и содержит информацию о событии
  - Чтобы получать уведомления о событиях некоторого компонента, приложение должно реализовать обработчик соответствующего типа и зарегистрировать его в компоненте
  - Событие запускается, когда пользователь активирует компонент (например, нажимает кнопку)

Три типа событий

- Действие (ActionEvent)
- Изменение значения (ValueChangeEvent)
- Событие цикла обработки запроса (PhaseEvent)

Два подхода к обработке событий

- Подход 1:
  - Реализовать метод обработки события в интерфейсном бине
  - Автор страницы указывает метод с помощью отложенного EL-выражения в атрибуте действия JSP
- Подход 2:
  - Реализовать класс обработчика событий
    - javax.faces.event.ActionListener
    - javax.faces.event.ValueChangeListener
  - Автор страницы регистрирует обработчик для компонента с помощью вложенного действия JSP



Обработка события действия

Событие действия

- Происходит, когда пользователь активирует компонент типа ActionSource
  - в т.ч. UICommand (кнопки и ссылки)
- Класс события javax.faces.event.ActionEvent
- Обработчик реализует интерфейс javax.faces.event.ActionListener, либо обработка выполняется методом, принимающем параметр типа ActionEvent
- Обрабатывается на шаге 2: “Применить значения из запроса” или на шаге 5: “Вызвать приложение”

Подход 1: Метод интерфейсного бина для ActionEvent

- Шаг 1: Метод обработки события в интерфейсном бине

```
public void chooseLocaleFromLink(ActionEvent event) {
  String current = event.getComponent().getId();
  FacesContext context = FacesContext.getCurrentInstance();
  context.getViewRoot().
    setLocale((Locale) locales.get(current));
  resetMaps();
}
```

- Шаг 2: Ссылка на метод в атрибуте actionListener (только для компонентов класса UICommand и компонентов типа ActionSource)

```
<h:command_link id="NAmerica" action="storeFront"
  actionListener="#{carstore.chooseLocaleFromLink}">
```

Подход 2: Обработчик типа ActionListener

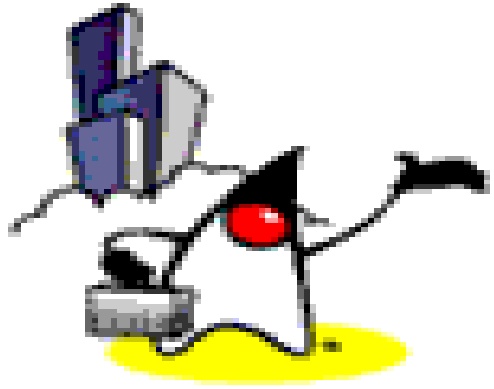
- Шаг 1: Реализовать интерфейс ActionListener
  - Метод processAction(ActionEvent) вызывается реализацией JSF, когда происходит событие ActionEvent – после приема значений из запроса
- Шаг 2: Зарегистрировать обработчик для компонента типа UICommand
  - Вложенное действие <f:actionListener>
  - Атрибут type указывает полноквалифицированное имя класса обработчика

Подход 2: пример обработчика LocaleChange

```
public class LocaleChange extends Object implements
 ActionListener {

 public void processAction(ActionEvent event)
  throws AbortProcessingException {

  String current = event.getComponent().getId();
  FacesContext context =
   FacesContext.getCurrentInstance();
  context.getViewRoot().setLocale((Locale)
   locales.get(current));
  resetMaps();
 }
}
```

На JSP-странице:

```
<h:command_link id="NAmerica" action="storeFront">
  <f:actionListener type="carstore.LocaleChange" />
</h:command_link>
```



Обработка события изменения значения

Событие изменения значения

- Приводит к изменению значения компонента типа UIInput
  - Например, событие происходит при вводе текста в текстовое поле
- Класс javax.faces.event.ValueChangeEvent
- Обработчик реализует интерфейс javax.faces.event.ValueChangeListener, либо обработка выполняется методом, принимающем параметр типа ValueChangeEvent
- Обрабатывается на шаге 3: “Выполнить валидацию”

Подход 1: Метод интерфейсного бина для ValueChangeEvent

- Шаг 1: Метод обработки события в интерфейсном бине

```
public void firstNameChanged(ValueChangeEvent event) {
    String current = event.getComponent().getId();
    ...
}
```

- Шаг 2: Ссылка на метод в атрибуте valueChangeListener

```
<h:inputText  id="firstName" value="#{customer.firstName}"
  required="true"
  valueChangeListener="#{carstore.firstNameChanged}" />
```

Подход 2: Обработчик типа ValueChangeListener

- Шаг 1: Реализовать интерфейс ValueChangeListener
  - Метод processValueChange(ValueChangeEvent) вызывается реализацией JSF, когда происходит событие ValueChangeEvent – после валидации значения компонента
  - Объект ValueChangeEvent содержит старое и новое значения компонента, вызвавшего событие
- Шаг 2: Зарегистрировать обработчик для компонента типа UIInput
  - Вложенное действие <f:valueChangeListener>
  - Атрибут type указывает полноквалифицированное имя класса обработчика

Подход 2: пример обработчика FirstNameChanged

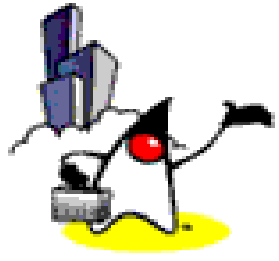
```
public class FirstNameChanged extends Object implements
    ValueChangeListener {

   public void processValueChange(ValueChangeEvent event)
      throws AbortProcessingException {
      if (null != event.getNewValue()) {
         FacesContext.getCurrentInstance().getExternalContext()
            .getSessionMap()
            .put("firstName", event.getNewValue());
      }
   }

   public PhaseId getPhaseId() {
      return PhaseId.ANY_PHASE;
   }
}
```

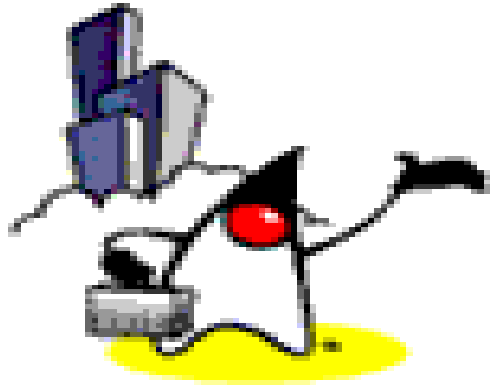

Событие обрабатывается после указанного шага обработки запроса

Подход 2: пример JSP-страницы customerInfo.jsp

```
<h:inputText  id="firstName" value="#{customer.firstName}"
  required="true">
  <f:valueChangeListener
    type="carstore.FirstNameChanged" />
</h:inputText>
```

Управляемые бины и интерфейсные бины





Методы бинов

Типы методов бинов

- Get- и set-методы свойств
  - Соответствуют соглашениям JavaBeans
- Методы, относящиеся к JSF
  - Методы валидации
  - Методы обработки действия
  - Методы обработки изменения значения
  - Методы навигации (Action-методы)

Методы валидации

- Метод валидации принимает два параметра: *FacesContext* и компонент типа *UIInput*
  - Так же как метод *validate()* интерфейса *Validator*
- Можно проверять только значения компонентов типа *UIInput* (и подтипов)
- Компонент ссылается на этот метод с помощью атрибута *validate*

Пример метода валидации

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

    if ((context == null) || (component == null)) {
        throw new NullPointerException();
    }

    if (value != null) {
        try {
            int converted = intValue(value);
            if (maximumSet &&
                (converted > maximum)) {
                if (minimumSet) {
```

Методы обработки действия

- Обрабатывают *ActionEvent*
  - Принимает *ActionEvent* и возвращает void
- Компонент ссылается на этот метод с помощью атрибута *actionListener*
- Этот метод можно указывать только компонентах типа *ActionSource*
  - *UICommand*
  - *UIButton*

Пример метода обработки действия

```
public void chooseLocaleFromLink(ActionEvent event) {
  String current = event.getComponent().getId();
  FacesContext context = FacesContext.getCurrentInstance();
  context.getViewRoot().
    setLocale((Locale) locales.get(current));
    resetMaps();
}
```

Метод навигации

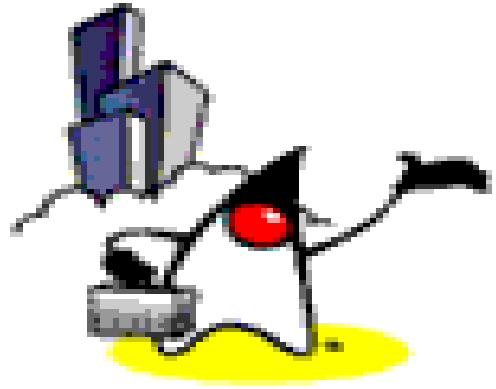
- Не принимает параметров и возвращает строку результата (String)
- Компонент ссылается на этот метод с помощью атрибута *action*
- Пример:

```
public String buyCurrentCar() {
  getCurrentModel().getCurrentPrice();
  return "confirmChoices";
}
```

Как использовать метод бина в компоненте?

- Указать в виде отложенного EL-выражения в одном из следующих атрибутов
  - *action*
  - *actionListener*
  - *validator*
  - *valueChangeListener*
- Пример:

```
<h:inputText ...
  validator="#{CarBean.validateInput}"
  valueChangeListener="#{CarBean.processValueChange}" />
```



Связывание значений и экземпляров компонентов с внешними источниками данных

Виды связывания

- Связывание значения компонента со свойством бина
- Связывание экземпляра компонента со свойством бина
  - Интерфейсный бин
- Связывание значения компонента со встроенным объектом

Связывание значения компонента со свойством бина

- Укажите имя бина и его свойство в атрибуте *value* с помощью отложенного EL-выражения #{X.Y}
  - X – значение <managed-bean-name>
  - Y – значение <property-name>
- Бин и его свойства объявляются в конфигурационном файле (*faces-config.xml*)

Пример:

- Вызывающая страница

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

- Объявление бина в конфигурационном файле

```
<managed-bean>
  <managed-bean-name>CarBean</managed-bean-name>
  <managed-property>
    <property-name>carName</property-name>
    <value>Jalopy</value>
  </managed-property>
  ...
</managed-bean>
```

Основания для связывания значения компонента со свойством бина

- У автора страницы есть контроль над атрибутами компонента, в отличие от разработчика бина.
- В управляемом бине не будет зависимостей от JavaServer Faces API, что обеспечивает лучшее разделение представления и модели.
- Реализация JSF может выполнять автоматическое преобразование данных к типу свойства бина.

Связывание экземпляра компонента со свойством бина

- Атрибут *binding* компонента
- Когда экземпляр компонента связан со свойством интерфейсного бина, то свойство содержит ссылку на сам компонент.
  - Когда значение компонента связано со свойством бина, то свойство содержит значение из модели, которое обновляется только на соответствующем шаге цикла обработки запроса.

Пример

- Вызывающая страница

```
<h:selectBooleanCheckbox
   id="fanClub" rendered="false"
      binding="#{cashier.specialOffer}" />
```

- Класс бина

```
public class CashierBean extends AbstractBean {
   protected Date shipDate;
   protected String name = null;
   protected String shippingOption = "2";
   protected String[] newsletters = new String[0];

   UIOutput specialOfferText = null;
   UIOutput thankYou = null;
   UISelectBoolean specialOffer = null;

   private CreditCardConverter creditCard = null;
   private String stringProperty = "This is a String property";
```

Основания для связывания экземпляра компонента со свойством бина

- Разработчик интерфейсного бина может программно изменять атрибуты компонента
- Изменение состояния компонента не отражается на состоянии модели
  - Например, множество выбранных в таблице строк

Связывание значения компонента со встроенным объектом

- Встроенные объекты
  - application, applicationScope
  - cookie
  - facesContext
  - header
  - headerValues
  - initParam
  - param
  - paramValues
  - request, requestScope
  - session, sessionScope
  - view

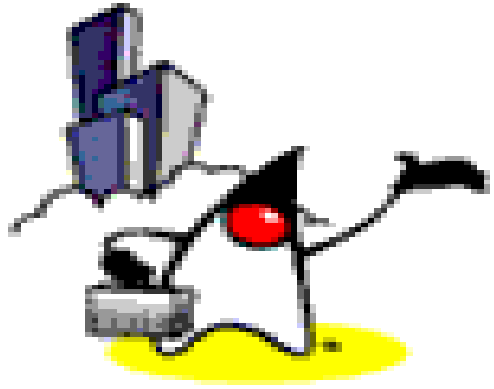
Пример:

- Вызывающая страница

```
<h:outputText id=version value="#{initParam.versionNo}"
```

- Установочный дескриптор web.xml

```
<context-param>
  <param-name>versionNo</param-name>
  <param-value>1.05</param-name>
</context-param>
```



Сравнение управляемых и интерфейсных бинов

Сравнение управляемых и интерфейсных бинов

- Управляемый бин – это JavaBean, зарегистрированный в файле faces-config.xml
  - Свойства этих бинов связываются со значениями компонентов UI
  - Используется атрибут *value*
  - <h:inputText value="#{ManagedBean.propertyName}" ... />

Сравнение управляемых и интерфейсных бинов

- Интерфейсный бин – особый тип управляемого бина, свойствами которого являются компоненты UI
  - Свойства бина связаны непосредственно с компонентом UI, а не с его значением
  - Используется атрибут *binding*

```
<h:inputText
  binding="#{BackingBean.someUIInputInstance}" ... /
  >
```