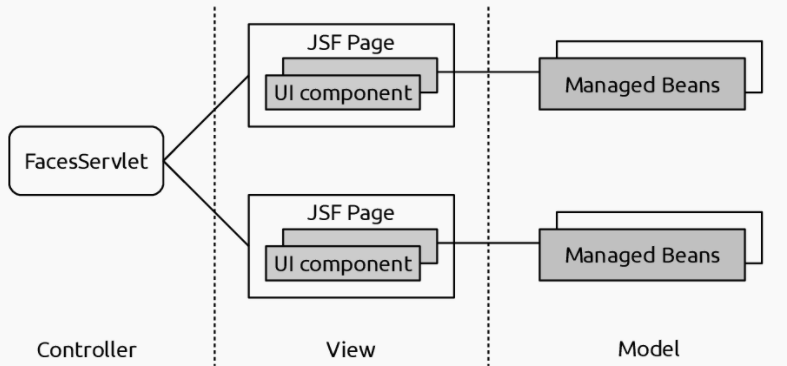
**JSF**

-Веб-MVC фреймворк, который основан на использовании компонентов и является частью JAVA EE.



* View - За представление в JSF отвечают JSP страницы или Facelets шаблоны. По сути Facelets, это xml документы, который описывают страницы и умеют в jsf компоненты.
* Controller – FacesServlet - это сервлет, который управляет жизненным циклом обработки запросов для веб-приложений, использующих JavaServer Faces для создания пользовательского интерфейса.
* Model – бины

**FacesServlet** – обрабатывает запросы от браузера, формирует события и вызывает слушателей. Реализация данного фрейма позволяет нам перейти от типичного http запроса с его request – response на событие – обработчик события – FacesServlet реализует этот переход. Занимается синхронизацией состояний между представлением и DOM.

Его конфигурация задается как для обычного сервлета. Мы прописываем что все запросы будут приходить на данный сервлет.

Интерфейс JSF приложения строится из компонентов. Компонент – кусок html и java код который обрабатывает запрос на серверной стороне. Все компоненты реализуют интерфейс UIComponent. Располагаются компоненты на jsp или facelets шаблонах. Компоненты объединены в древовидную структуру – представление, корень – экземпляр класса UIViewRoot.

Facelets – JSFстраницы

Изображение выглядит как текст

Автоматически созданное описаниеОбычный xhtml с набором дополнительных тегов. Xhtml – xml документ с дополнительным набором html тегов. Namespace – набор тегов котоые могут быть использованы в xml для тегов мы можем назначать свои префиксы.

* F – core f.e. ajax
* H – html элементы

Браузер на вход получает обычный html или xhtml. Поэтому FacesServlet преобразует JSF страницы в xhtml. Поэтому мы получаем две структуры – DOM на стороне клиента, который формируется из html самим браузером и представление со стороны сервера.

Навигация между страницами

* Изображение выглядит как текст

  Автоматически созданное описаниеВ файле faces-config.xml. Тут мы просто задаем navigation rule.
* Прописать в action – url нашей страницы, тогда jsf поймет что это страница при перейдет туда. <h:commandButton action = "page2" value = "Page2"/>
* Используя управляемый компонент, мы можем очень легко управлять навигацией

public class NavigationController implements Serializable {

@ManagedProperty(value = "#{param.pageId}")

private String pageId;

public String showPage() {

if(pageId == null) {

return "home";

Изображение выглядит как текст

Автоматически созданное описание }

if(pageId.equals("1")) {

return "page1";

}else if(pageId.equals("2")) {

return "page2";

}else {

return "home";

}

}

}

* Добавить в bean метод, который возвращает строку с нужным url. И поместит его в action

<h:commandButton action = "#{navigationController.moveToPage1}"

public class NavigationController implements Serializable {

public String moveToPage1() {

return "page1";

}

}

Управляемые бины

Классы, которые хранят состояние нашего приложения. Формально – java классы, которые удовлетворяют спецификации.

* Содержат параметры и методы для обработки данных с компонентов.
* Используются для обработки событий UI и валидации данных.
* Жизненным циклом управляет JSF Runtime Envronment- т.е. мы не управляем их жизнью.
* Доступ из JSF-страниц осуществляется с помощью элементов EL.
* Конфигурация задаётся в faces-config.xml (JSF 1.Х), либо с помощью аннотаций (JSF 2.0).

Пример

@ManagedBean

@SessionScoped

public class HelloBean implements Serializable {

private static final long serialVersionUID = 1L;

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this. name = name;

}

public String getSayWelcome(){

if("".equals(name) || name == null){ //check if null?

return "";

}else{

return "Ajax message : Welcome " + name;

}

}

}

При создании бина мы задаем его время жизни в параметре scope с помощью faces-config.xml или с помощью аннотаций.

* @NoneScoped - контекст не определён, жизненным циклом управляют другие бины. Т.е. просто мы к нему не можем обратиться. Мы можем его заинжектить в другой бин с заданным scope и работать. Встраивание
* @RequestScoped (применяется по умолчанию) - контекст - запрос. Мгновенно
* @ViewScoped (JSF 2.0) - контекст - страница. Пока мы не уйдем с этой страницы мы живем
* @SessionScoped - контекст - сессия.
* @ApplicationScoped - контекст - приложение. Един на все приложение - синглтон
* @CustomScoped (JSF 2.0) - бин сохраняется в Мар программист сам управляет его жизненным циклом.

Конфигурация бинов

|  |  |
| --- | --- |
| **Способ 1 - через faces-config.xml:**  <managed-bean>  <managed-bean-name>customer</managed-bean-name>  <managed-bean-class>CustomerBean</managed-bean-class>  <managed-bean-scope>request</managed-bean-scope>  <managed-property>  <property-name>areaCode</property-name>  <value>#{initParam.defaultAreaCode}</value>  </managed-property>  </managed-bean> | **Способ 2 (JSF 2.0) - с помощью аннотаций**:  @ManagedBean(name="customer")  @RequestScoped  public class CustomerBean {  …  @ManagedProperty(value="#{initParam.defaultAreaCode}"  name="areaCode")  private String areaCode; } |

Доступ к бинам из xhtml

Осуществляется с помощью el выражений. # - read/write; $ - read. Примеры

* Value=”#{beanname.name}” – вызов функции setName()
* Validator=”#{beanname.validate}” – определяет валидность введенного значения
* Binding=”#{beanname.nameField}” – типа value
* Action=”#{beanname.save}” – метод

Конвертация

Http – текстовый протокол, все данные в нем – строк. JSF работает на уровне java, поэтому ему надо конвертировать данные типы данных java. Для этого нужен конвертор.

• Используются для преобразования данных компонента в заданный формат (дата, число и т. д.).

• Реализуют интерфейс javax.faces.convert.Converter.

• Существуют стандартные конвертеры для основных типов данных.

• Можно создавать собственные конвертеры.

Реализация

* Автоматическая - <h:inputText value="#{user.age}"/> - age – Integer в бине
* С помощью атрибута converter: <h:inputText converter="#{javax.faces.DateTime}"/> - в теге
* С помощью вложенного тега: <h:outputText value="#{user.birthDay}">

<f:converter converterId="#{javax.faces.DateTime}"/>

</h:outputText>

Валидация

Когда мы уже получили данные нужного нам типа мы можем их провалидировать.

* Осуществляется перед обновлением значения компонента на уровне модели.
* Класс, осуществляющий валидацию, должен реализовывать интерфейс Validator
* Существуют стандартные валидаторы для основных типов данных.
* Можно создавать собственные валидаторы.

Изображение выглядит как текст

Автоматически созданное описаниеРеализация

* Изображение выглядит как текст

  Автоматически созданное описаниеС помощью логики на уровне управляемого бина.

Обработка запроса

1. Request
2. Формирование представления
   * JSF Runtime формирует представление (начиная с UIViewRoot):
     + Создаются объекты компонентов.
     + Назначаются слушатели событий, конвертеры и валидаторы.
     + Все элементы представления помещаются в FacesContext.
   * Если это первый запрос пользователя к странице JSF, то формируется пустое представление.
   * Если это запрос к уже существующей странице, то JSF Runtime синхронизирует состояние компонентов представления с клиентом.
3. Получение значений компонентов

* На стороне клиента все значения хранятся в строковом формате - нужна проверка их корректности:
  + Вызывается конвертер в соответствии с типом данных значения.
* Если конвертация заканчивается успешно, значение сохраняется в локальной переменной компонента.
* Если конвертация заканчивается неудачно, создаётся сообщение об ошибке, которое помещается в FacesContext.

1. Валидация значений
   * Вызываются валидаторы, зарегистрированные для компонентов представления.
   * Если значение компонента не проходит валидацию, формируется сообщение об ошибке, которое сохраняется в FacesContext.
2. Обновление значений компонентов
   * Если данные валидны, то значение компонента обновляется.
   * Новое значение присваивается полю объекта компонента.
3. Вызов приложения
   * Управление передается слушателям событий
   * Формируются новые значения компонентов
4. Формирование ответа от сервера
   * JSF Runtime обновляет представление в соответствии с результатами обработки запроса.
   * Если это первый запрос к странице, то компоненты помещаются в иерархию представления.
   * Формируется ответ сервера на запрос. На стороне клиента происходит обновление страницы.
5. Response