# Сервлеты

Платформа Java EE

* Набор стандартов и спецификаций для создания корпоративных приложений на Java
* Спецификации Java EE реализуются серверами приложений: Apache Tomcat, GlassFish, RedHat

Сервлеты – серверные сценарии, написанные на Java. Жизненным циклом управляет веб-контейнер (он же контейнер сервлетов). В отличие от CGI, запросы обрабатываются в отдельных потоках (а не процессах) на веб-контейнере.

## Преимущества и недостатки сервлетов

Преимущества:

* Выполняются быстрее, чем CGI-сценарии.
* Хорошая масштабируемость.
* Надежность и безопасность (реализованы на JAVA)
* Платформенно-независимы
* Множество инструментов мониторинга и отладки

Недостатки:

* Слабое разделение уровня представления и бизнес-логики.
* Возможны конфликты при параллельной обработке запросов.

## Обработка HTTP-запроса

1. Браузер формирует HTTP-запрос и отправляет его на сервер.
2. Веб-контейнер создает объекты HttpServletRequest и HttpServletResponse.
3. Веб-контейнер вызывает метод service сервлета.
4. Сервлет формирует ответ и записывает его в поток вывода HttpServletResponse.

## Пример сервлета

public class SampleServlet extends HttpServlet {

 public void doGet (HttpServletRequest request, HttpServletResponse response)

 throws IOException {

 String pageTitle = “Пример сервлета”;

 response.setContentType(“text/html”);

out.println…

## Конфигурация сервлета

* Веб-контейнер создает один (строго) экземпляр сервлета на каждую запись в дескрипторе (servlet – servlet-name – servlet-class)
* Веб-контейнер перенаправляет запрос с URL на конкретный сервлет в соответствии с конфигурацией.

## Жизненный цикл сервлета

1. Загрузка класса
2. Создание экземпляра
3. Вызов метода init()
4. Вызов метода service()
5. Вызов метода destroy()

## Контекст сервлетов

* API, с помощью которого сервлет может взаимодействовать со своим контейнером
* Доступ к методам осуществляется через интерфейс javax.servlet.ServletContext
* У всех сервлетов внутри приложения общий контекст
* В контекст можно помещать общую для всех сервлетов информацию (методы getAttribute и setAttribute)
* Если приложение – распределенное, то на каждом экземпляре JVM контейнером создается свой контекст.
* ServletContext context=getServletContext();
* Strin driverName = context.getInitParameter(“dname”);

## HTTP-сессии

* HTTP-stateless-протокол
* javax.servlet.HttpSession – интерфейс, позволяющий идентифицировать конкретного клиента (браузер) при обработке множества HTTP-запросов от него
* Экземпляр HttpSession создается при первом обращение клиента к приложению и сохраняется некоторое (настраиваемое) время после последнего обращения.
* Идентификатор сессии либо помещается в cookie, либо добавляется к URL.
* Сессия «привязана» к конкретному приложению, у разных приложений – разные сессии.
* В распределенном окружении обеспечивается сохранение целостности данных в HTTP-сессии (независимо от количества экземпляров JVM)
* HttpSession session = request.getSession();

## Диспетчеризация запросов сервлетами

* Сервлеты могут делегировать обработку запросов другим ресурсам (сервлетам, JSP и HTML-страницам)
* Диспетчеризация осуществляется с помощью реализаций интерфейса javax.servlet.RequestDispatcher
* Два способа получения RequestDispatcher – через ServletRequest (абсолютный или относительный URL) и ServletContext (только абсолютный URL)
* Два способа делегирования обработки запроса – forward и include.

## Фильтры запросов

* Фильтры позволяют осуществлять пред- и постобработку запросов до и после передачи их ресурсу (сервлету, JSP или HTML-странице)
* Пример предобработки – допуск к странице только авторизованных пользователей
* Пример постобработки – запись в лог времени обработки запроса
* Реализуют интерфейс javax.servlet.Filter
* Ключевой метод doFilter
* Метод doFilter класса FilterChain передает управление следующему фильтру или целевому ресурсу; таким образом, возможна реализация последовательностей фильтров, обрабатывающих один и тот же запрос.

public class MyFilter implements Filter {

 public void init(FilterConfig arg0) throws ServletException {}

 public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws IOException, ServletException {

 PrinterWriter out = resp.getWriter();

 out.print(“filter is invoked before”);

 chain.doFilter(req, resp);

 out.print(“filter is invoked after”);

}

 public void destroy() {}

}

### Конфигурация

<web-app>

 <servlet>

 <servlet-name>s1</servlet-name>

 <sevlet-class>HelloServlet</servlet-class>

 </servlet>

 <servlet-mapping>

 <servlet-name>s1</servlet-name>

 <url-pattern>/servlet1</url-pattern>

 </servlet-mapping>

 <filter>

 <filter-name>f1</filter-name>

 <filter-class>MyFilter</filter-class>

 </filter>

 <filter-mapping>

 <filter-name>f1</filter-name>

 <url-pattern>/servlet1</url-pattern>

 </filter-mapping>

</web-app>

# Java Server Pages

* Страницы JSP – это текстовые файлы, содержащие статический HTML и JSP-элементы.
* JSP-элементы позволяют формировать динамическое содержимое.
* При загрузке в веб-контейнер страницы JSP транслируются компилятором (jasper) в сервлеты.
* Позволяют отделить бизнес-логику от уровня представления (если их комбинировать с сервлетами).

## Преимущества и недостатки JSP

Преимущества:

* Высокая производительность – транслируются в сервлеты.
* Не зависят от используемой платформы – код пишется на Java.
* Позволяют использовать Java API.
* Простые для понимания – структура похожа на обычный HTML.

Недостатки:

* Трудно отлаживать, если приложение целиком основано на JSP.
* Возможны конфликты при параллельной обработки нескольких запросов.

## Жизненный цикл JSP

1. Трансляция JSP в код сервлета.
2. Компиляция сервлета.
3. Загрузка класса сервлета.
4. Создание экземпляра сервлета.
5. Вызов метода jspinit().
6. Вызов метода \_jspService()
7. Вызов метода jspDestroy()

## JSP-элементы

* Комментарий - <%-- Comment --%>
* Директива - <%@ directive %>
* Объявление - <%! decl %>
* Скриплет - <% code %>
* Выражение <%= expr %>

## Предопределенные приложения

application, config, exception, out, page, PageContext, request, response, session

## JSP Actions

jsp:include – Включает в страницу внешний файл во время обработки запроса

jsp:useBean – Добавляет на страницу экземпляр Java Bean с заданным конекстом.

jsp:getProperty – Получение и установка свойств Java Bean

jsp:forward – Перенаправление на другую страницу

# Шаблоны проектирования в веб-приложениях

* Шаблон проектирования или паттерн – повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.
* Описывает подход к решению типовой задачи.
* Одну и ту же задачу часто можно решить с использованием разных шаблонов.

## Зачем нужны паттерны

* Позволяют избежать «типовых» ошибок при разработке типовых решений.
* Позволяют кратко описать подход к решению задачи.
* Легче поддерживать код – его поведение более предсказуемо.

# Java Server Faces

* JSF – фреймворк для разработки веб-приложений.
* Входит в состав платформы Java EE
* Основан на использовании компонентов.
* Для отображения данных используются JSP или XML-шаблоны (facelets)

## Достоинства JSF

* Четкое разделение бизнес-логики и интерфейса (фреймворк реализует шаблон MVC)
* Управление обменом данными на уровне компонент.
* Простая работа с событиями на стороне сервера.
* Доступность нескольких реализаций от различных компаний-разработчиков.
* Расширяемость (можно использовать дополнительные наборы компонентов).
* Широкая поддержка со стороны интегрированных средств разработки (IDE)

## Недостатки JSF

* Высокоуровневый фреймворк – сложно реализовывать не предусмотренную авторами функциональность.
* Сложности с обработкой GET-запросов (устранены в JDF 2.0).
* Сложность разработки собственных компонентов.

## Структура JSF-приложения

* JSP или XHTML-страницы, содержащие компоненты GUI.
* Библиотеки тегов.
* Управляемые бины.
* Дополнительные объекты (компоненты, конвертеры и валидаторы).
* Дополнительные теги.
* Конфигурация – faces-config.xml (опционально)
* Дескриптор развертывания – web.xml

## FacesServlet

* Обрабатывает запросы с браузера
* Формирует объекты-события и вызывает методы-слушатели.

Конфигруация:

<!--Faces Servlet -->

<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>/faces/\*</url-pattern>

</servlet-mapping>

## Страницы и компоненты UI

* Интерфейс строится из компонентов.
* Компоненты расположены на Facelets-шаблонах или страницах JSP.
* Компоненты реализуют интерфейс javax.faces.component.UIComponent.
* Можно создавать собственные компоненты.
* Компоненты на странице объединены в древовидную структуру – представление.
* Корневым элементом представления является экземпляр класса javax.faces.component.UIViewRoot.

## Пример страницы

 <?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:f="http://java.sun.com/jsf/core"

 xmlns:h="http://java.sun.com/jsf/html">

 <h:body>

 <h3>JSF 2.0 + Ajax Hello World Example</h3>

 <h:form>

 <h:inputText id="name" value="#{helloBean.name}"></h:inputText>

 <h:commandButton value="Welcome Me">

 <f:ajax execute="name" render="output" />

 </h:commandButton>

 <h2>

 <h:outputText id="output" value="#{helloBean.sayWelcome}" />

 </h2>

 </h:form>

 </h:body>

</html>

## Навигация между страницами JSF

* Реализуется экземплярами класса NavigationHandler
* Правила задаются в файле faces-config.xml

 <navigation-rule>

 <from-view-id>/pages/inputname.xhtml</from-view-id>

 <navigation-case>

 <from-outcome>sayHello</from-outcome>

 <to-view-id>/pages/greeting.xhtml</to-view-id>

 </navigation-case>

 <navigation-case>

 <to-view-id>/pages/goodbye.xhtml</to-view-id>

 </navigation-case>

 </navigation-rule>

* Пример перенаправления на другую страницу:

<h:commandButton id=”submit” action=”sayHello” value = “Submit” />

# Управляемые бины

* Содержат параметры и методы для обработки данных с компонентов.
* Используются для обработки событий UI и валидации данных.
* Жизненным циклом управляет JSF Runtime Environment
* Доступ из JSF-страниц осуществляется с помощью элементов EL.
* Конфигурация задается в faces-config.xml, либо с помощью аннотаций.

Пример:

package org.itmo.sample;

import javax.faces.bean.ManagedBean;

import javax.faces.bean.SessionScoped;

import java.io.Serializable;

@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?

 eturn "";

 }else{

 return "Ajax message : Welcome " + name;

 }

 }

}

## Контекст (scope) управляемых бинов

* Задается через faces-config.xml или с помощью аннотаций.
* 6 вариантов конфигурации:

@NoneScoped – контекст не определен, жизненным циклом управляют другие бины.

@RequestScoped (применяется по умолчанию) – контекст-запрос.

@ViewScoped – контекст-страница

@SessionScoped – контекст-сессия

@ApplicationScoped – контекст-приложение

@CustomScoped – бин сохраняется в Map; программист сам управляет его жизненным циклом.

## Конфигурация управляемых бинов

<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>

## Конвертеры данных

* 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

## Назначение конвертеров

 ● Автоматическое (на основании типа данных):

<h:inputText value="#{user.age}"/>

● С помощью атрибута converter:

<h:inputText converter="#{javax.faces.DateTime}"/>

● С помощью вложенного тега:

<h:outputText value="#{user.birthDay}">

 <f:converter

 converterId="#{javax.faces.DateTime}"/>

</h:outputText>

## Валидация данных JSF-компонентов

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