Технологии разработки web-приложений с помощью технологии сервлетов и jsp

Согласно заявлению Sun Microsystems, на настоящий момент более 90% корпоративных систем поддерживают платформу Java Enterprise Edition.

Первый сервлет

Сервлеты – это компоненты приложений Java Enterprise Edition, выполняющиеся на стороне сервера, способные обрабатывать клиентские запросы и динамически генерировать ответы на них. Наибольшее распространение получили сервлеты, обрабатывающие клиентские запросы по протоколу HTTP.

Все сервлеты реализуют общий интерфейс **Servlet** из пакета **javax.servlet**. Для обработки HTTP-запросов можно воспользоваться

в качестве базового класса абстрактным классом **HttpServlet** из пакета **javax.servlet.http**.

Жизненный цикл сервлета начинается с его загрузки в память контейнером сервлетов при старте контейнера либо в ответ на первый запрос. Далее производятся инициализация, обслуживание запросов и завершение существования.

Первым вызывается метод **init()**. Он дает сервлету возможность инициализировать данные и подготовиться для обработки запросов. Чаще всего в этом методе программист помещает код, кэширующий данные фазы инициализации.

После этого сервлет можно считать запущенным, он находится в ожидании запросов от клиентов. Появившийся запрос обслуживается методом

service(HttpServletRequest req, HttpServletResponse res) сервлета, а все параметры запроса упаковываются в объект **req** класса **HttpServletRequest**, передаваемый в сервлет. Еще одним параметром этого метода является объект **res** класса **HttpServletResponse**, в который загружается информация для передачи клиенту. Для каждого нового клиента при обращении к сервлету создается независимый поток, в котором производится вызов метода **service()**. Метод **service()** предназначен для одновременной обработки множества запросов.

После завершения выполнения сервлета контейнер сервлетов вызывает метод **destroy()**, в теле которого следует помещать код освобождения занятых сервлетом ресурсов.

При разработке сервлетов в качестве базового класса в большинстве случаев используется не интерфейс **Servlet**, а класс **HttpServlet**, отвечающий за обработку запросов HTTP. Этот класс уже имеет реализованный метод

service().

Метод **service()** класса **HttpServlet** служит диспетчером для других методов, каждый из которых обрабатывает методы доступа к ресурсам. В спецификации HTTP определены следующие методы: **GET**, **HEAD**, **POST**, **PUT**, **DELETE**, **OPTIONS** и **TRACE**. Наиболее часто употребляются методы **GET** и **POST**, с помощью которых на сервер передаются запросы, а также параметры для их выполнения.

При использовании метода **GET** (по умолчанию) параметры передаются как часть URL, значения могут выбираться из полей формы или передаваться непосредственно через URL. При этом запросы кэшируются и имеют ограничения на размер. При использовании метода **POST** (**method=POST**) параметры (поля формы) передаются в содержимом HTTP-запроса и упакованы согласно полю заголовка **Content-Type**.

По умолчанию в формате:

<имя>=<значение>&<имя>=<значение>&...

Однако форматы упаковки параметров могут быть самые разные, например в случае передачи файлов с использованием формы

enctype="multipart/form-data".

В задачу метода **service()** класса **HttpServlet** входит анализ полученного через запрос метода доступа к ресурсам и вызов метода, имя которого сходно с названием метода доступа к ресурсам, но перед именем добавляется префикс **do**: **doGet()** или **doPost()**. Кроме этих методов, могут использоваться методы **doHead()**, **doPut()**, **doDelete()**, **doOptions()** и **doTrace()**. Разработчик должен переопределить нужный метод, разместив в нем функциональную логику.

В следующем примере приведен готовый к выполнению шаблон сервлета:

```
// пример # 1 : простейший сервлет : MyServlet.java
package chapt17;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class MyServlet extends HttpServlet {
      public MyServlet() {
            super();
      }
      public void init() throws ServletException {
      }
      protected void doGet(
            HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            out.print("This is ");
            out.print(this.getClass().getName());
            out.print(", using the GET method");
            }
      protected void doPost(
            HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            out.print("This is ");
            out.print(this.getClass().getName());
            out.print(", using the POST method");
            }
      public void destroy() {
            super.destroy(); // Just puts "destroy" string in log
            }
}
```

Практика включения HTML-кода в код сервлета не считается хорошей, так как эти действия “уводят” сервлет от его основной роли – контроллера приложения. Это приводит к разрастанию размеров сервлета, которое на определенном этапе становится неконтролируемым и реализует вследствие этого анти-шаблон “Волшебный сервлет”. Даже приведенный выше маленький сервлет имеет признаки анти-шаблона, так как содержит метод **print()**, используемый для формирования кода HTML. Сервлет должен использоваться только для реализации бизнес-логики приложения и обязан быть отделен как от непосредственного формирования ответа на запрос, так и от данных, необходимых для этого. Обычно для формирования ответа на запрос применяются возможности JSP, JSPX или JSF. Признаки наличия анти-шаблонов все же будут встречаться ниже, но это отступление сделано только с точки зрения компактности примеров.

Сервлет является компонентом Web-приложения, который будет называться **FirstProject** и размещен в папке **/WEB-INF/classes** проекта.

Запуск контейнера сервлетов и размещение проекта

Здесь и далее применяется контейнер сервлетов Apache Tomcat в качестве обработчика страниц JSP и сервлетов. Последняя версия может быть загружена

с сайта **jakarta.apache.org**.

При установке Tomcat предложит значение порта по умолчанию **8080**, но во избежание конфликтов с иными Application Server рекомендуется присвоить другое значение, например **8082**.

Ниже приведены необходимые действия по запуску сервлета из предыдущего примера с помощью контейнера сервлетов Tomcat 5.5.20, который установлен в каталоге **/Apache Software Foundation/Tomcat5.5**. В этом же каталоге размещаются следующие подкаталоги:

/bin – содержит файлы запуска контейнера сервлетов **tomcat5.exe**,

tomcat5w.exe и некоторые необходимые для этого библиотеки;

/common – содержит библиотеки служебных классов, в частности Servlet API;

/conf – содержит конфигурационные файлы, в частности конфигурационный файл контейнера сервлетов **server.xml**;

/logs – помещаются log-файлы;

/webapps – в этот каталог помещаются папки, содержащие сервлеты и другие компоненты приложения.

В каталог **/webapps** необходимо поместить папку **/FirstProject** с вложенным в нее сервлетом **MyServlet**. Кроме того, папка **/FirstProject** должна содержать каталог **/WEB-INF**, в котором помещаются подкаталоги:

/classes – содержит класс сервлета **chapt17.MyServlet.class**;

/lib – содержит используемые внешние библиотеки (если они есть), упакованные в JAR-файлы (архивы java);

/src – содержит исходный файл сервлета **MyServlet.java** (опционально);

а также **web.xml** – дескриптор доставки приложения располагается в каталоге **/WEB-INF**.

В файле **web.xml** необходимо прописать имя и путь к сервлету. Кроме того, в дескрипторном файле можно определять параметры инициализации, MIME-типы, mapping сервлетов и JSP, стартовые страницы и страницы с сообщениями об ошибках, а также параметры для безопасной авторизации и аутентификации. Этот файл можно сконфигурировать так, что имя сервлета в браузере не будет совпадать с истинным именем сервлета. Например:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
      <display-name>FirstProject</display-name>
<servlet>
  <display-name>MyServletdisplay</display-name>
  <servlet-name>MyServletname</servlet-name>
  <servlet-class>chapt17.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>MyServletname</servlet-name>
  <url-pattern>/MyServlettest</url-pattern>
</servlet-mapping>
<session-config>
        <session-timeout>30</session-timeout>
</session-config>
      <welcome-file-list>
            <welcome-file>index.jsp</welcome-file>
      </welcome-file-list>
 <login-config>
  <auth-method>BASIC</auth-method>
 </login-config>
</web-app>
```

Здесь указано имя сервлета **MyServletname**, путь к откомпилированному классу сервлета **MyServlet.class**, а также URL-имя сервлета, по которому происходит его вызов **MyServlettest**.

Таким образом, требуется выполнить следующие действия:

1. Компиляцию сервлета с указанием в `-cp` пути к архиву
2. `servlet-api.jar`;
3. Полученный файл класса **MyServlet.class** поместить в папку **/FirstProject/WEB-INF/classes/chapt18**;
4. В папку **/MyProject/WEB-INF** поместить файл конфигурации **web.xml**;
5. Переместить папку **/FirstProject** в каталог **/webapps** контейнера сервлетов Tomcat;
6. Стартовать Tomcat;
7. Запустить браузер и ввести адрес:
   http://localhost:8082/FirstProject/MyServlettest
   При обращении к сервлету из другого компьютера вместо **localhost** следует указать IP-адрес или имя компьютера.
8. Если вызывать сервлет из **index.jsp**, то тег **FORM** должен выглядеть следующим образом:

```
<FORM action="MyServlettest">
      <INPUT type="submit" value="Execute">
</FORM>
```

Файл **index.jsp** помещается в папку **/webapps/FirstProject**
и в браузере набирается строка:

http://localhost:8082/FirstProject/index.jsp

Сервлет будет вызван из JSP-страницы по URL-имени **MyServlettest**,

и в результате в браузер будет выведено:

Рис. 1. Вывод сервлета после вызова метода doGet()

Первая JSP

Java Server Pages (JSP) обеспечивает разделение динамической и статической частей страницы, результатом чего является возможность изменения дизайна страницы, не затрагивая динамическое содержание. Это свойство используется при разработке и поддержке страниц, так как дизайнерам нет необходимости знать, как работать с динамическими данными.

Процессы, выполняемые с файлом JSP при первом вызове:

1. Браузер делает запрос к странице JSP.
2. JSP-engine анализирует содержание файла JSP.
3. JSP-engine создает временный сервлет с кодом, основанным на исходном тексте файла JSP, при этом контейнер транслирует операторы Java в метод **`_jspService()`**. Если нет ошибок компиляции, то этот метод вызывается для непосредственной обработки запроса. Полученный сервлет ответствен за исполнение статических элементов JSP, определенных во время разработки в дополнение к созданию динамических элементов.
4. Полученный код компилируется в файл **`*.class`**.
5. Вызываются методы **`init()`** и **`_jspService()`**, и сервлет логически исполняется.
6. Сервлет на основе JSP установлен. Комбинация статического HTML
7. и графики вместе с динамическими элементами, определенными в оригинале JSP, пересылаются браузеру через выходной поток объекта ответа **`ServletResponse`**.

Последующие обращения к файлу JSP просто вызовут метод **`_jspService()`** сервлета. Сервлет используется до тех пор, пока сервер не будет остановлен и сервлет не будет выгружен вручную либо пока не будет изменен файл JSP. Результат работы JSP можно легко представить, зная правила трансляции JSP в сервлет, в частности в его **`_jspService()`**-метод.

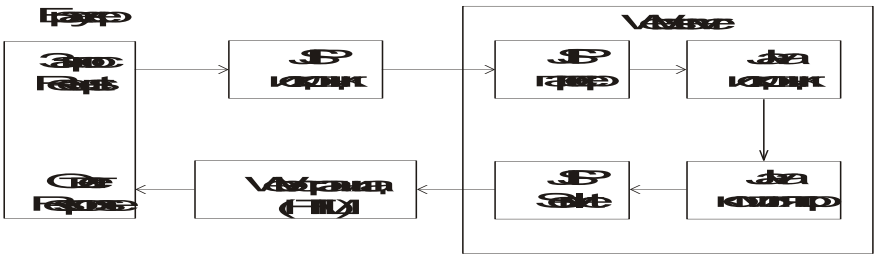

Рис. 2. Рабочий цикл JSP

Если рассмотреть преобразование в сервлет простейшей JSP, отправляющей

в браузер приветствие:

```
<!--пример # 2 : простейшая страница JSP : simple.jsp -->
<html><head>
<title>Simple</title>
</head>
<body>
<jsp:text>Hello, Bender</jsp:text>
</body></html>
```

то в результате запуска в браузер будет выведено:

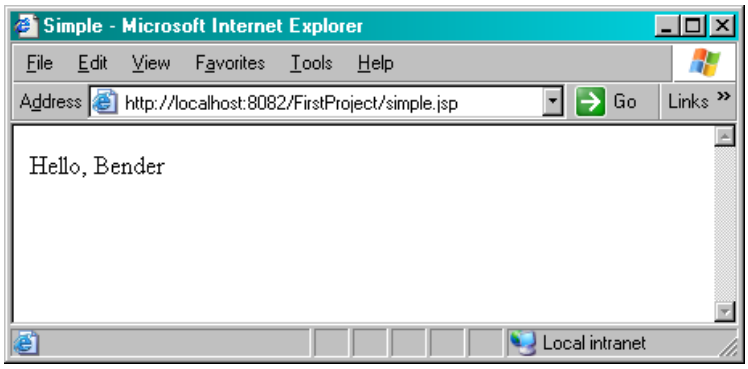

Рис. 3. Вывод после вызова index.jsp

А код сервлета будет получен следующий:

// пример # 3 : сгенерированный сервлет : simple_jsp.java

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class simple_jsp
   extends org.apache.jasper.runtime.HttpJspBase
   implements org.apache.jasper.runtime.JspSourceDependent{
  private static java.util.List _jspx_dependants;

  public Object getDependants() {
    return _jspx_dependants;
  }
  public void _jspService(HttpServletRequest request,
                          HttpServletResponse response)
        throws java.io.IOException, ServletException {

    JspFactory _jspxFactory = null;
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;
    try {
      _jspxFactory = JspFactory.getDefaultFactory();
      response.setContentType("text/html");
      pageContext = _jspxFactory.getPageContext(this,
            request, response, null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write("<html><head>\r\n");
      out.write("<title>Simple</title>\r\n");
      out.write("</head>\r\n");
      out.write("<body>\r\n");
      out.write("Hello, Bender\r\n");
      out.write("</body></html>");
    } catch (Throwable t) {
      if (!(t instanceof SkipPageException)){
        out = _jspx_out;
```

```
        if (out != null && out.getBufferSize() != 0)
          out.clearBuffer();
        if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
      }
    } finally {
      if (_jspxFactory != null)
_jspxFactory.releasePageContext(_jspx_page_context);
    }
  }
}
```

JSP-код заключается в специальные теги, которые указывают контейнеру, чтобы он использовал этот код для генерации сервлета или его части. Таким образом поддерживается документ, который одновременно содержит и статическую страницу, и теги Java, которые управляют этой страницей. Статические части HTML-страниц посылаются в виде строк в метод **write()**. Динамические части включаются прямо в код сервлета. С этого момента страница ведет себя как обычная HTML-страница с ассоциированным сервлетом.

Взаимодействие сервлета и JSP

Страницы JSP и сервлеты никогда не используются в информационных системах друг без друга. Причиной являются принципиально различные роли, которые играют данные компоненты в приложении. Страница JSP ответственна за формирование пользовательского интерфейса и отображение информации, переданной с сервера. Сервлет выполняет роль контроллера запросов и ответов, то есть принимает запросы от всех связанных с ним JSP-страниц, вызывает соответствующую бизнес-логику для их (запросов) обработки и в зависимости от результата выполнения решает, какую JSP поставить этому результату в соответствие.

Ниже приведен пример вызова сервлета из JSP с последующим вызовом другой JSP.

```
<!--пример # 4 : страница JSP с вызовом сервлета : index.jsp -->
<%@ page language="java" contentType="text/html; charset=ISO-8859-5" pageEncoding="ISO-8859-5"%>
<html><body>
<jsp:useBean id="gc" class="java.util.GregorianCalendar"/>
<jsp:getProperty name="gc" property="time"/>
<FORM action="serv" method="POST">
            <INPUT type="submit" value="Вызвать сервлет">
</FORM>
</body></html>
```

В результате запуска проекта в браузер будет выведено:

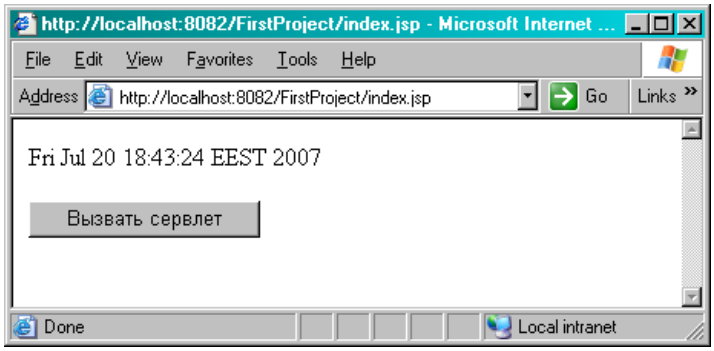

Рис. 4. Запуск index.jsp

Кодировка для символов кириллицы задана с помощью директивы **page**. Action-теги **useBean** и **getProperty** используются для создания объекта класса **GregorianCalendar** в области видимости JSP и вывода его значения. Сервлет **ContServlet** вызывается методом **POST**.

```
// пример # 5 : простой контроллер : ContServlet.java
package chapt17;
import java.io.IOException;
import java.util.Calendar;
import java.util.Locale;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ContServlet
            extends javax.servlet.http.HttpServlet {
```

```
        protected void doPost(HttpServletRequest request,
                              HttpServletResponse response)
                          throws ServletException, IOException {
        //добавление атрибута к запросу
        request.setAttribute("loc", Locale.getDefault());
        //добавление атрибута к сессии
        request.getSession().setAttribute("calend",
                                    Calendar.getInstance());
        //получение объекта RequestDispatcher и вызов JSP
request.getRequestDispatcher("/main.jsp").forward(request,
                                                   response);
        }
}
```

Передачу информации между JSP и сервлетом можно осуществлять, в частности, с помощью добавления атрибутов к объектам **HttpServletRequest**, **HttpSession**, **HttpServletContext**. Вызов **main.jsp** из сервлета

в данном случае производится методом **forward()** интерфейса **RequestDispatcher**.

```
<!--пример # 6 : страница, вызванная сервлетом : main.jsp -->
<%@ page language="java"
contentType="text/html; charset=utf-8"
pageEncoding="utf-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c"%>
<html><body>
<h3>Региональные установки и Время</h3>
<c:out value="Locale from request: ${loc}"/><br>
<c:out value="Time from Servlet: ${calend.time}"/>
</body></html>
```

После вызова сервлета и последующего вызова **main.jsp** будет выведено:

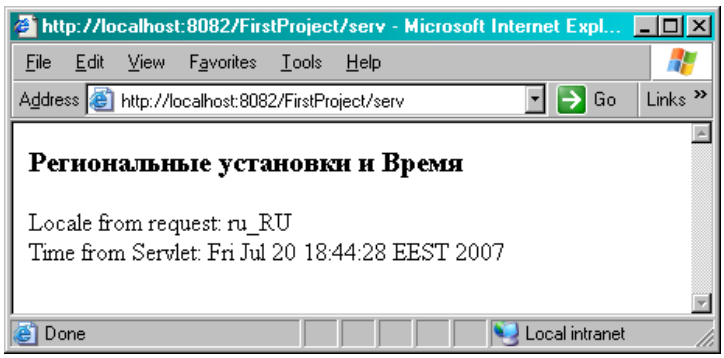

Рис. 5. Вывод информации страницей main.jsp

В данном коде директива **taglib** подключает JSP Standard Tag Library (JSTL), и становится возможным вызов тега **<c:out>**, а также использование Expression Language (EL) в виде **${loc}**.

Конфигурационный файл **web.xml** для данной задачи должен содержать следующую информацию:

```
<servlet>
  <display-name>Controller</display-name>
  <servlet-name>controller</servlet-name>
  <servlet-class>chapt17.ContServlet</servlet-class>
 </servlet>
  <servlet-mapping>
  <servlet-name>controller</servlet-name>
  <url-pattern>/serv</url-pattern>
 </servlet-mapping>
```

В этой главе была дана общая информация о взаимодействии различных компонентов Web-приложения.