# Часть 3.
# ТЕХНОЛОГИИ РАЗРАБОТКИ WEB-ПРИЛОЖЕНИЙ

*В третьей части даны основы программирования распределенных информационных систем с применением сервлетов, JSP и баз данных, а также основные принципы создания собственных библиотек тегов.*

## Глава 17
## ВВЕДЕНИЕ В СЕРВЛЕТЫ 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 ката-