CHAPTER 3

第3章

Servlet 编程

本章要点

- Web 基础回顾；
- 什么是 Servlet；
- Servlet 功能；
- Servlet 编程；
- Servlet 配置；
- Servlet 生命周期；
- Servlet 应用案例。

在 Java EE 应用编程中 Servlet 是基础，JSP 和 JSF 都是建立在 Servlet 基础之上的，其他 Web 框架如 Struts，WebWork 和 Spring MVC 都是基于 Servlet 的，因此理解 Servlet 的工作过程、生命周期、部署和调用是掌握 Java Web 开发的基础。

3.1 Web 基础回顾

在系统学习 Servlet 开发之前，了解一下 Web 基础是很有必要的，对学习 Servlet 开发有非常重要的引导作用，Servlet 本身就是 Web 组件，完全遵循 Web 工作过程。

3.1.1 Web 基本概念

Web(World Wide Web，万维网)本质上就是 Internet 上的所有文档的集合。Web 文档的主要类型有 HTML 网页、CSS、JavaScript、各种动态网页、图片、声音和视频等。

Web 文档保存在 Web 站点(Site)上，Web 站点驻留在 Web 服务器上。Web 服务器是一种软件系统，提供 Web 文档的管理和请求服务。常见的 Web 服务器软件有 Apache，IIS，WebLogic，GlassFish，JBoss 和 Tomcat 等。Web 服务器运行在连接 Internet 的计算机上，我们一般称之为服务器。每个服务器都有一个唯一的 IP 地址。Web 服务器对外都有一个服务端口，默认是 80，也可以设定为其他端口。

Web 文档都有一个唯一的地址，通过 URL(Uniform Resource Locator)格式来进行定位，

其格式为：协议://IP地址:端口/站点名/目录/文件名。其中协议主要有HTTP,HTTPS和FTP。根据不同的协议,默认端口号可以省略,HTTP和HTTPS为80,FTP为21。

Web文档请求例子：

http://210.30.108.30:8080/crm2009/admin/login.jsp。以HTTP协议请求Web文档。

ftp://210.30.108.30/software/sun/jsk6.zip。以FTP协议请求Web文档。

Web服务器接收到请求后,根据URL定位到相应的文档,根据文档的类型进行对应的处理,将文档通过网络发送到客户端,一般是浏览器,用户即可查看或下载请求的文档。

3.1.2 Web工作模式

Web使用请求/响应模式进行工作,即由客户(一般是浏览器)使用URL对Web文档进行请求,Web服务器接收并处理请求,处理结束后将响应内容发送到客户。Web服务器不会主动将Web文档发送到客户端,这种方式也称为拉(PULL)模式。

3.1.3 Web请求方式

Web的请求方式主要有GET,POST,PUT,DELETE和HEAD。其中：

(1) GET请求：直接返回请求的文档,同时可以在请求时传递参数数据,参数数据在URL地址上直接传递。如：http://localhost:8080/web01/main.do?id=lhd&password=9002。

Web请求基本上使用GET方式,如在浏览器地址栏直接输入URL地址和超链接等都使用GET方式进行工作。

(2) POST请求：将传递到Web服务器的数据保存到数据流中,可以发送大的请求数据,例如上传文件到Web服务器。POST方式只有使用表单提交才能实现。如下为POST请求实例：

```
<form action="add.do" method="post">
   <input type="text" name="username" />
   <input type="submit" value="提交"/>
</form>
```

Web请求方式只有GET和POST请求使用最为广泛,本书编程只涉及这两种方式。

3.1.4 Web响应类型

当Web服务器接收客户端请求并处理完毕后,向客户端发送HTTP响应(Response),当客户端接收完服务器的响应后,显示在浏览器上,完成一次请求/响应过程。

HTTP响应一般情况下是HTML文档,也可以是其他类型格式。Web使用MIME(Multipurpose Internet Mail Extensions)标准来确定具体的响应类型,在www.w3c.org互联网组织官方网站上包含所有的MIME类型。

HTTP响应类型总体上分为两类：

(1) 文本类型：纯文本类型响应,包括纯文本字符、HTML和XML等。

(2) 二进制原始类型：包括图片、声音和视频等。

3.2 Servlet 概述

在 Sun 公司制定 Java EE 规范初期，为实现动态 Web 而引入了 Servlet，用于替代笨拙的 CGI(通用网关接口)，实现了 Java 语言编程的动态 Web 技术，奠定了 Java EE 的基础，使动态 Web 开发达到了一个新的境界。后来为进一步简化动态 Web 网页的生成，并且在微软公司推出了 ASP(Active X 服务系统页面)技术的竞争情况下，Sun 推出了 JSP 规范，进一步简化了 Web 网页的编程。但 JSP 在进行 HTTP 请求处理方面不如 Servlet 方便和规范，Servlet 在当今的 MVC 模式 Web 开发中牢牢占据着一席之地。并且现在流行的 Web 框架基本基于 Servlet 技术，如 Struts，WebWork 和 Spring MVC 等。只有掌握了 Servlet，才能真正掌握 Java Web 编程的核心和精髓。

3.2.1 什么是 Servlet

按照 Java EE 规范定义，Servlet 是运行在 Web 容器的 Java 类，它能处理 Web 客户的 HTTP 请求，并产生 HTTP 响应。

Servlet 是 Java EE 规范定义的 Web 组件，运行在 Web 容器中。由 Web 容器负责管理 Servlet 的生命周期，包括创建和销毁 Servlet 对象。

客户不能直接创建 Servlet 对象和调用 Servlet 的方法，只能通过向 Web 服务器发出 HTTP 请求，间接调用 Servlet 的方法。这是 Servlet 与普通 Java 类的重要区别。

3.2.2 Servlet 体系结构

Sun 在如下两个包中提供了 Servlet 的全部接口和类：

(1) javax.servlet 包含支持所有协议的通用的 Web 组件接口和类。

(2) javax.servlet.http 包含支持 HTTP 协议的接口和类。

Servlet API 的主要接口和类结构如图 3-1 所示。

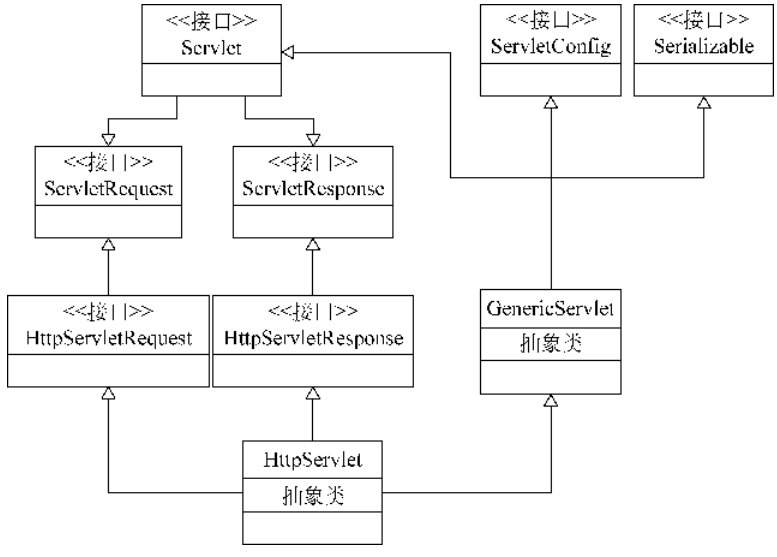

图 3-1 Servlet API 主要接口和类结构

3.2.3 Servlet 功能

Servlet 可以完成 Web 组件具有的所有功能，具体功能如下：

(1) 接收 HTTP 请求。

(2) 取得请求信息，包括请求头和请求参数数据。

(3) 调用其他 Java 类方法，完成具体的业务功能。

(4) 生成 HTTP 响应，包括 HTML 和非 HTML 响应。

(5) 实现到其他 Web 组件的跳转，包括重定向和转发，后续章节将详细介绍这些跳转方式。

3.3 Servlet 编程

Servlet 的编程方式与普通 Java 类的编写相同。与一般的 Java 类相比特别之处在于 Servlet 的类编写要严格按照 Java EE 的规范进行，包括需要实现的接口、继承的类、方法和方法的参数都要符合规范，否则无法在 Web 容器内部署和运行。Servlet 编程遵循以下步骤。

3.3.1 引入包

编写 Servlet 要引入 Servlet 的两个包和 Java I/O 包。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

3.3.2 类定义

编写接收 HTTP 请求并进行 HTTP 响应的 Servlet 要继承 javax.servlet.http.HttpServlet。Servlet 类的定义代码如下：

```
public class LoginAction extends HttpServlet
{  }
```

一般情况下不需要编写构造方法，即使用默认的无参数的构造方法。

3.3.3 重写 doGet 方法

每个 Servlet 一般都需要重写 doGet 方法，因为父类 HttpServlet 的 doGet 方法是空的，没有实现任何代码，子类需要重写此方法。doGet 方法的定义代码如下：

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException,IOException
{  }
```

当客户使用 GET 方式请求 Servlet 时，Web 容器调用 doGet 方法处理请求。

3.3.4 重写 doPost 方法

同样编写 Servlet 需要重写父类的 doPost 方法。此方法的定义代码如下：

```
public void doPost ( HttpServletRequest request, HttpServletResponse response ) throws
ServletException,IOException
{     }
```

当客户使用 POST 方式请求 Servlet 时，Web 容器调用 doPost 方法。

doGet 和 doPost 方法都接收 Web 容器自动创建的请求对象和响应对象，使得 Servlet 能够解析请求数据和发送响应给客户端。

3.3.5 重写 init 方法

当 Web 容器创建 Servlet 对象后，会自动调用 init 方法完成初始化功能，一般需要将耗时的连接数据库和打开外部资源文件的操作放在 init 方法中。init 方法在 Web 容器创建 Servlet 类对象后立即执行，且只执行一次，每次 Servlet 处理 HTTP 协议的 GET 或 POST 请求时，就不再运行 init 方法，只执行 doGet 或 doPost 方法。init 方法的定义代码如下：

```
public void init(ServletConfig config) throws ServletException
{
    super.init(config);
    //这里放置进行初始化工作的代码
}
```

在 init 方法中可以使用 Web 容器传递的 config 对象取得 Servlet 的各种配置初始参数，进而使用这些参数完成读取数据库或其他外部资源。

3.3.6 重写 destroy 方法

当 Web 容器需要销毁 Servlet 对象时，一般是 Web 容器停止运行或 Servlet 源代码修改而重新部署的时候。Web 容器自动运行 destroy 方法完成清理工作，如关闭数据库连接和关闭 I/O 流等。如下为关闭数据库连接的 destroy 方法，可以在 init 方法中取得数据库连接对象，每次处理 HTTP 请求时可以使用此连接对象，最后在 Servlet 销毁之前将其关闭并销毁，这将极大地提高 Servlet 的运行性能和系统的响应速度。

```
public void destroy()
{
      try {
           cn.close();
      } catch(Exception e) {
          application.Log("登录处理关闭数据库错误" + e.getMessage());
      }
}
```

代码中的 application 为 Web 应用的上下文环境对象，详见第 7 章。

3.4 Servlet 生命周期

编写 Servlet 必须真正了解 Servlet 的生命周期，理解 Servlet 生命周期中每个阶段的状态，进而开发出性能好的 Servlet 组件。

Servlet 的生命周期完全由 Web 容器掌管，客户必须通过 Web 容器发送对 Servlet 的请求，不能直接使用 new Servlet 对象，也不能像调用普通 Java 类那样直接调用 Servlet 的方法。Servlet 的所有方法都由 Web 容器调用。Servlet 要经过加载实例化、初始化、服务和销毁 4 个阶段，如图 3-2 所示是 Servlet 生命周期时序图。

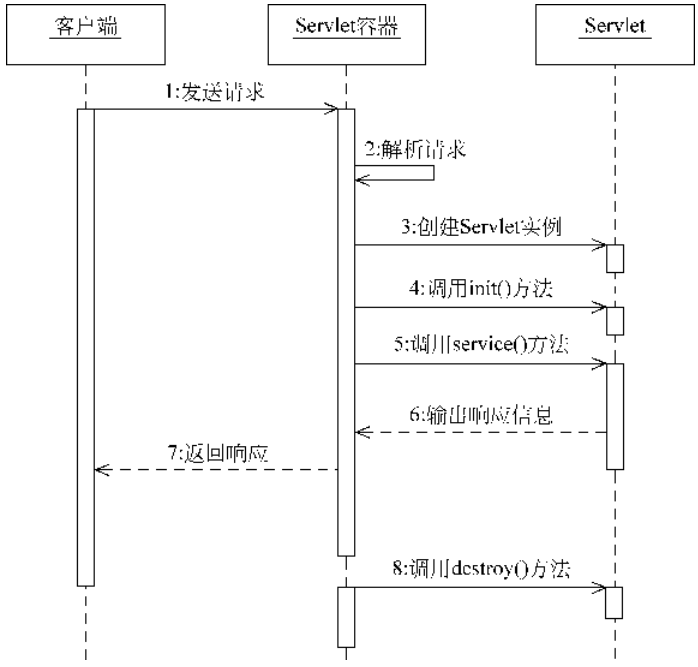

图 3-2 Servlet 生命周期时序图

3.4.1 加载类和实例化阶段

Servlet 由 Web 容器进行加载，当 Web 容器检测到客户首次请求 Servlet 时，根据 web.xml 文件的配置包名和类名，在/WEB-INF/classes 目录下查找 Servlet 类文件并加载到内存中。类加载结束后，使用反射机制调用默认的无参数的构造方法创建 Servlet 类对象，并保存在 Web 容器的 JVM 内存中。

3.4.2 初始化阶段

在创建 Servlet 对象后，Web 容器会调用 Servlet 的 init 方法，完成初始化工作，如连接数据和打开 I/O 流等。对每个 Servlet 对象 init 方法只执行一次，适合完成耗时较长的对象创建操作。当对象建立以后，每次请求的服务处理方法就可以直接使用 init 中创建的对象，

从而极大地提高系统的性能。可以重写两种形式的 init 方法中的一种：

(1) public void init()throws ServletException;

(2) public void init(ServletConfig config)throws ServletException;

推荐重写第二种形式的 init 方法，通过 ServletConfig 对象可以取得 Servlet 的很多配置信息，而第一种方法无法得到 ServletConfig 对象。

3.4.3 处理请求阶段

Web 容器每次接收到对 Servlet 的 HTTP 请求时，自动调用 Servlet 的 service 方法，一般不需要重写 service 方法。而父类 HttpServlet 的 service 方法非空，在此方法中会取得请求的方式，如果是 GET 请求就会调用子类的 doGet 方法，如果是 POST 请求就调用 doPost 方法。

Web 容器在调用 service 之前，将创建 HttpServletRequest 请求对象和 HttpServletResponse 响应对象。将客户端发送的请求头(Header)和请求体进行解析，写入到请求对象中，将请求对象和响应对象作为参数传递到 service 方法中，进而传递到 doGet 或 doPost 方法中。

Servlet 在请求处理的方法 doGet 或 doPost 中通过请求对象取得客户提交的请求信息，如表单数据和地址栏参数等，并对这些数据进行处理，如写入到数据库中等，从而完成业务处理。

Servlet 使用响应对象完成对客户的响应，设置响应头和响应体，由 Web 容器将响应对象的内容通过 HTTP 和 TCP/IP 协议以流方式发送到客户端的浏览器中。

Servlet 的请求/响应处理主要在服务阶段进行，编程任务集中在 doGet 和 doPost 方法中。

3.4.4 销毁阶段

当以下情况发生时，Web 容器就会销毁 Servlet 组件，在销毁 Servlet 对象之前，就会调用 Servlet 的 destroy 方法，完成资源清理工作。

(1) Web 容器停止。

(2) Servlet 类更新。

(3) Web 应用重新部署。

destroy 方法完成在 init 方法中取得的各种资源对象的关闭和销毁工作，如关闭数据库连接和关闭 I/O 流等操作并释放这些对象所占的内存。

3.5 Servlet 配置

Servlet 作为 Web 组件可以处理 HTTP 请求/响应，因而对外要求一个唯一的 URL 地址。但由于 Servlet 是一个 Java 类文件，不像 JSP 那样直接存放在 Web 目录下就能获得 URL 请求访问地址。Servlet 必须在 Web 的配置文件/WEB-INF/web.xml 中进行配置和映射才能响应 HTTP 请求。Servlet 的配置分为声明和映射两个步骤。

3.5.1 Servlet 声明

Servlet 声明的任务是通知 Web 容器 Servlet 的存在，声明的语法如下：

```
<servlet>
  <servlet-name>loginaction</servlet-name>
  <servlet-class>com.city.oa.action.LoginAction</servlet-class>
</servlet>
```

其中：

<servlet-name>声明 Servlet 的名字，可以为任何字符串，一般与 Servlet 的类名相同即可，要求在一个 web.xml 文件内名字唯一。

<servlet-class>指定 Servlet 的全名，即包名.类名。Web 容器会根据此定义载入类文件到内容中，进而调用默认构造方法创建 Servlet 对象。

在 Servlet 声明中还可以有以下内容：

1. Servlet 初始参数

在 Servlet 声明配置中可以配置 Servlet 初始参数，如数据库的 Driver，URL，账号和密码等信息，在 Servlet 中可以读取这些信息，从而避免在 Servlet 代码中定义这些信息。当这些信息要修改时，不需要重新编译 Servlet，直接修改配置文件即可，提高系统的可维护性。

Servlet 初始参数的配置语法如下所示，其中定义了 1 个初始参数，即数据库的 JDBC 驱动。

```
<servlet>
   <init-param>
     <param-name>driver</param-name>
     <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
   </init-param>
</servlet>
```

在 Servlet 中可以通过 ServletConfig 取得定义的初始化参数，取得以上定义的参数的示例代码如下：

```
//取得 Servlet 定义的初始参数
String driver = config.getInitParameter("driver");
//根据 Servlet 初始参数连接数据库
Class.forName(driver);
Connection cn = DriverManager.getConnection("jdbc:odbc:cityoa");
```

其中 config 是在 Servlet 中定义的 ServletConfig 类型的属性变量，由 init 方法取得它的实例。由此可见要连接不同的数据库，直接修改配置文件即可，不需要代码的修改和重新编译。

2. Servlet 启动时机

在配置 Servlet 时，可以指示 Servlet 跟随 Web 容器一起自动启动，这时 Servlet 就可以

在没有请求的情形下，进行实例化和初始化，完成特定的任务。许多 Web 框架如 Struts 就是用这种方法在 Web 容器启动后，使用自启动 Servlet 完成框架的导入和对象创建工作。自启动 Servlet 的配置语法是：

```
<load-on-startup>2</load-on-startup>
```

其中：数字表示启动的顺序，数字越小越先启动，最小为 0，表示紧跟 Web 容器启动后，第一个启动。原则上不同的 Servlet 应该使用不同的启动顺序数字。

如下示例代码是 Struts 1 的核心 Servlet 的配置案例：

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>
```

在以上 Servlet 配置中，配置启动顺序为 2，并且配置了 Servlet 初始参数，指定了 Struts 的配置文件位置。

3.5.2 Servlet 映射

任何 Web 文档在 Internet 上都要有一个 URL 地址才能被请求访问，Servlet 不能像 JSP 一样直接放在 Web 的发布目录下，因此 Servlet 需要单独映射 URL 地址。在/WEB-INF/web.xml 中进行 Servlet 的 URL 映射。

(1) 映射语法：

```
<servlet-mapping>
    <servlet-name>servlet 名称</servlet-name>
    <url-pattern>URL</url-pattern>
</servlet-mapping>
```

其中：servlet 名称与 Servlet 声明中的名称要一致。

(2) 映射地址方式：

Servlet 映射地址可以是绝对地址，也可以是匹配式地址对应多个请求地址。

① 绝对地址方式映射：绝对地址只能映射到 1 个地址，URL 的格式如下：/目录/目录/文件名.扩展名。

例子：

```
<servlet-mapping>
    <servlet-name>LoginAction</servlet-name>
    <url-pattern>/login.action</url-pattern>
</servlet-mapping>
```

上述 Servlet LoginAction 只能响应地址 /login.action 的请求。

② 匹配目录模式映射方式：URL 格式如下：/目录/目录/*。这类映射重点匹配目录，只要目录符合映射模式，不考虑文件名，这个 Servlet 可以响应多个请求 URL。

例子：

```
<servlet-mapping>
    <servlet-name>MainAction</servlet-name>
    <url-pattern>/main/*</url-pattern>
</servlet-mapping>
```

在这个映射地址配置中，只要是以/main/为开头的任何 URL 都能请求此 Servlet。

如下请求均被此 Servlet 响应：

```
http://localhost:8080/web01/main/login.jsp
http://localhost:8080/web01/main/info/add.do
```

③ 匹配扩展名模式映射方式：以匹配扩展名的方式进行 URL 映射，不考虑文件的目录信息，也可以响应多地址的请求。URL 格式：*.扩展名。

例子：

```
<servlet-mapping>
    <servlet-name>MainAction</servlet-name>
    <url-pattern>*.action</url-pattern>
</servlet-mapping>
```

以上配置中扩展名为 action 的任何请求均被此 Servlet 响应，如：

```
http://localhost:8080/web01/login.action
http://localhost:8080/web01/main/info/add.action
```

注意：不能混合使用以上两种匹配模式，否则会在 Web 项目部署并运行时产生运行时错误。如下映射地址是错误的：

```
<servlet-mapping>
    <servlet-name>MainAction</servlet-name>
    <url-pattern>/main/*.action</url-pattern>
</servlet-mapping>
Caused by: java.lang.IllegalArgumentException: Invalid <url-pattern> /main/*.do in servlet
mapping.
```

3.6 Servlet 部署

编译好的 Servlet class 文件要放置到指定的 Web 应用目录下，才能被 Web 容器找到，这个目录就是/WEB-INF/classes 目录。在此目录下根据 Servlet 类的包名创建对应的目录。即：

/WEB-INF/classes/包名目录/类名.class

例如 Servlet 类为：com.city.oa.action.MainAction，则其部署为：

/WEB-INF/classes/com/city/oa/action/MainAction.class

如果使用 IDE(集成开发环境)工具如 Eclipse,MyEclipse 和 NetBean 等,会自动进行 Servlet 的部署,不需要手工编译和复制。

3.7 Servlet 应用案例:取得数据表记录并显示

3.7.1 案例功能简述

编写一个 Servlet,连接 Oracle 数据库,使用 Oracle 的内置练习账号 scott,显示所有员工的记录列表,包括数据表 EMP 的姓名(ENAME)、职位(JOB)、工资(SAL)和加入公司日期(HIREDATE)。

3.7.2 案例分析设计

为演示直接使用 Servlet 编程,本案例没有按照 MVC 模式进行设计,而是直接在 Servlet 中连接数据库、执行查询、遍历取得的记录和显示所有的员工信息。

数据库的连接信息都在 Servlet 初始参数中进行配置,避免硬编码方式,提高了系统的可维护性。在 init 方法中取得初始参数,并取得数据库连接,供每次请求时使用,而不是每次请求时都连接数据库,提高了系统的性能。

在 destroy 方法中关闭数据连接和释放连接对象,节省了内存占用。

3.7.3 案例编程实现

(1) 案例 Servlet 编码:完成案例功能的 Servlet 如程序 3-1 代码所示:

程序 3-1 ShowEmployeeList.java

```
package com.city.j2ee.ch02;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
//显示 Oracle 数据库 scott 员工表 EMP 的所有记录
public class ShowEmployeeList extends HttpServlet
{
    //定义数据库连接对象属性,供 Servlet 其他方法使用
    private Connection cn = null;
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        //取得 Servlet 配置的数据库连接初始参数
        String driver = config.getInitParameter("driver");
        String url = config.getInitParameter("url");
        String user = config.getInitParameter("user");
        String password = config.getInitParameter("password");
```