第 5 章 Java Servlet基础

学习目的与要求

本章主要介绍 Java Servlet 的基础知识，包括部署 Servlet、Servlet 的生命周期、通过 JSP 页面访问 Servlet、重定向与转发等。通过本章的学习，要求读者熟练掌握 Servlet 对象的创建与运行，理解 Servlet 的生命周期与工作原理。

本章主要内容

- Servlet 对象的创建与运行
- Servlet 的生命周期
- 通过 JSP 页面访问 Servlet
- doGet()和 doPost()方法
- 重定向与转发
- 在 Java Servlet 中使用 session
- 基于 Servlet 的 MVC 模式

Java Servlet 的核心思想就是在 Web 服务器端创建用来响应客户端请求的对象，该对象被称为一个 Servlet 对象。JSP 技术以 Java Servlet 为基础，当客户端请求一个 JSP 页面时，Web 服务器(例如 Tomcat 服务器)会自动生成一个对应的 Java 文件，编译该 Java 文件，并用编译得到的字节码文件在服务器端创建一个 Servlet 对象。实际的 Web 应用需要 Servlet 对象具有特定的功能，这就需要 Web 开发人员编写创建 Servlet 对象的类。对于如何编写 Servlet 类以及如何使用 Servlet 类，将在本章中重点介绍。

按照 1.2.2 节的操作步骤创建 Web 项目 ch5，并为 ch5 添加 Tomcat 依赖。本章涉及的 Java 源文件保存在 ch5 项目的 src 目录中，涉及的 JSP 页面保存在 ch5 项目的 web 目录中。

5.1 Servlet 类与 Servlet 对象

编写一个 Servlet 类很简单，只要继承 jakarta.servlet.http 包中的 HttpServlet 类，并重写响应 HTTP 请求的方法即可。HttpServlet 类实现了 Servlet 接口，实现了响应用户请求的接口方法。HttpServlet 类的一个子类习惯地称为一个 Servlet 类，这样的子类创建的对象又习惯地称为 Servlet 对象。

【例 5-1】 一个简单的 Servlet 类。

例 5-1 的 FirstServlet.java 的代码如下：

```
package servlet;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
```

```
import jakarta.servlet.http.HttpServletResponse;
public class FirstServlet extends HttpServlet{
 private static final long serialVersionUID = 1L;
 public void init(ServletConfig config) throws ServletException{
       super.init(config);
 }
 public void service(HttpServletRequest request,HttpServletResponse response)
             throws IOException{
       //设置响应的内容类型
       response.setContentType("text/html;charset=UTF-8");
       //取得输出对象
       PrintWriter out = response.getWriter();
       out.println("<html><body>");
       //在浏览器中显示"第一个 Servlet 类"
       out.println("第一个 Servlet 类");
       out.println("</body></html>");
 }
}
```

在编写 Servlet 类时必须有包名，也就是说必须在包中编写 Servlet 类。在本章中新建一个 Web 项目 ch5，所有的 Servlet 类都放在 src 目录下的 servlet 包中。

编写完 Servlet 类的源文件，是不是就可以运行 Servlet 对象了呢？不可以，需要在部署 Servlet 以后才可以运行 Servlet 对象。

扫一扫

视频讲解

5.2 Servlet 对象的创建与运行

如果想让 Web 服务器使用 Servlet 类编译后的字节码文件创建 Servlet 对象处理用户请求，必须先为 Web 服务器部署 Servlet。部署 Servlet 目前有两种方式，一种是在 web.xml 中部署 Servlet，另一种是基于注解的方式部署 Servlet。

▶ 5.2.1 在 web.xml 中部署 Servlet

web.xml 文件由 Web 服务器负责管理，该文件是 Web 应用的部署描述文件，包含如何将用户请求 URL 映射到 Servlet。因此，用户可以在 Web 项目的 web\WEB-INF\web.xml 文件中部署自己的 Servlet。

❶ 部署 Servlet

为了在 web.xml 文件中部署 5.1 节中的 FirstServlet，需要在 web.xml 文件中找到 <web-app></web-app>标记，然后在<web-app></web-app>标记中添加如下内容：

```
<servlet>
   <servlet-name>firstServlet</servlet-name>
   <servlet-class>servlet.FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>firstServlet</servlet-name>
   <url-pattern>/firstServlet</url-pattern>
</servlet-mapping>
```

❷ 运行 Servlet

Servlet 第一次被访问时，需要将 Web 项目发布到 Tomcat Web 服务器。发布后，可以在浏览器的地址栏中输入“http://localhost:8080/ch5/firstServlet”来运行 Servlet，运行效果如

图 5.1 所示。

图 5.1 第一个 Servlet 的运行效果

❸ web.xml 文件中与 Servlet 部署有关的标记及其说明

1）根标记< web-app >

在 XML 文件中必须有一个根标记，web.xml 的根标记是< web-app >。

2）< servlet >标记及其子标记

在 web.xml 文件中可以有若干< servlet >标记，该标记的内容由 Web 服务器负责处理。在< servlet >标记中有两个子标记< servlet-name >和< servlet-class >，其中< servlet-name >子标记的内容是 Web 服务器创建的 Servlet 对象的名称。在 web.xml 文件中虽然可以有若干< servlet >标记，但是要求它们的< servlet-name >子标记的内容互不相同。< servlet-class >子标记的内容指定 Web 服务器用哪个类来创建 Servlet 对象，如果 Servlet 对象已经创建，那么 Web 服务器就不再使用指定的类创建。

3）< servlet-mapping >标记及其子标记

在 web.xml 文件中出现一个< servlet >标记就会对应地出现一个< servlet-mapping >标记。在< servlet-mapping >标记中有两个子标记< servlet-name >和< url-pattern >，其中< servlet-name >子标记的内容是 Web 服务器创建的 Servlet 对象的名称（该名称必须和< servlet >标记的子标记< servlet-name >的内容相同）；< url-pattern >子标记用来指定用户用怎样的模式请求 Servlet 对象，比如< url-pattern >子标记的内容是/firstServlet，用户需要请求服务器运行 Servlet 对象 firstServlet 为其服务，那么可以在浏览器的地址栏中输入“http://localhost:8080/ch5/firstServlet”。

一个 Web 项目的 web.xml 文件负责管理该 Web 项目的所有 Servlet 对象，当 Web 项目需要提供更多的 Servlet 对象时，只要在 web.xml 文件中添加< servlet >和< servlet-mapping >标记即可。

▶ 5.2.2 基于注解的方式部署 Servlet

从 5.2.1 节可知，每开发一个 Servlet，都要在 web.xml 文件中部署 Servlet 才能够使用，这样会给 Web 项目的维护带来非常大的麻烦。从 Servlet 3.0 开始提供了@WebServlet 注解，使得用户不再需要在 web.xml 文件中进行 Servlet 的部署描述，简化了开发流程。本书中后续的 Servlet 都是基于注解的方式进行部署。

注解虽然方便了开发人员，但是在后期会让维护和调试成本增加。为了方便后期维护，建议开发人员在部署 Servlet 时把@WebServlet 的 urlPatterns 属性的值设置为 Servlet 类的名称。例如：

```
@WebServlet(name = "secondServlet", urlPatterns = {"/secondServlet"})
public class SecondServlet extends HttpServlet {
}
```

❶ @WebServlet

@WebServlet 用于将一个类声明为 Servlet 对象，该注解将会在部署时被 Web 容器处理，Web 容器根据具体属性将相应的类部署为 Servlet 对象。该注解的常用属性如表 5.1 所示。

表 5.1 @WebServlet 注解的常用属性

属性名	类 型	描 述
name	String	指定 Servlet 的 name 属性，等价于< servlet-name >。如果没有显式指定，则该 Servlet 的取值即为类的全名
value	String[ ]	该属性等价于 urlPatterns 属性，这两个属性不能同时使用
urlPatterns	String[ ]	指定一组 Servlet 的 URL 匹配模式，等价于< url-pattern >标记
loadOnStartup	int	指定 Servlet 的加载顺序，等价于< load-on-startup >标记
initParams	WebInitParam[ ]	指定一组 Servlet 初始化参数，等价于< init-param >标记

以上所有属性都为可选属性，但 value 或 urlPatterns 通常是必需的，且二者不能共存，如果同时指定，通常忽略 value 的取值。用户可使用简化注解，例如@WebServlet("/secondServlet")，其中"/secondServlet"为请求的 URL，即 urlPatterns 属性的值。

【例 5-2】 基于注解的 Servlet 类——SecondServlet。

例 5-2 的 SecondServlet.java 的代码如下：

```
package servlet;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
//建议 urlPatterns 的值和类名一样，以方便维护，可使用@WebServlet("/secondServlet")简化注解
@WebServlet(name = "secondServlet", urlPatterns = {"/secondServlet"})
public class SecondServlet extends HttpServlet {
 private static final long serialVersionUID = 1L;
 public void init(ServletConfig config) throws ServletException {
 }
 protected void service ( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {
       //设置响应的内容类型
       response.setContentType("text/html;charset = UTF - 8");
       //取得输出对象
       PrintWriter out = response.getWriter();
       out.println("< html >< body >");
       //在浏览器中显示"第二个 Servlet 类"
       out.println("第二个 Servlet 类");
       out.println("</body ></html >");
 }
}
```

在 SecondServlet.java 的代码中使用"@WebServlet(name = "secondServlet", urlPatterns = {"/secondServlet"})"部署以后，就不必在 web.xml 中部署相应的< servlet >和< servlet-mapping >元素了，Web 容器会在部署时根据指定的属性将该类发布为 Servlet 对象。"@WebServlet(name = "secondServlet", urlPatterns = {"/secondServlet"})"等价的 web.xml 部署形式如下：

```
< servlet >
   < servlet - name > secondServlet </servlet - name >
   < servlet - class > servlet.SecondServlet </servlet - class >
</servlet >
< servlet - mapping >
```

```
    <servlet-name>secondServlet</servlet-name>
    <url-pattern>/secondServlet</url-pattern>
</servlet-mapping>
```

❷ @WebInitParam

@WebInitParam 注解通常不单独使用,而是配合@WebServlet 和@WebFilter(在第 6 章中讲解)使用。它的作用是为 Servlet 或 Filter 指定初始化参数,这等价于 web.xml 中<servlet>的<init-param>子标记。@WebInitParam 注解的常用属性如表 5.2 所示。

表 5.2 @WebInitParam 注解的常用属性

属性名	类 型	是否可选	描 述
name	String	否	指定参数的名称,等价于<param-name>
value	String	否	指定参数的值,等价于<param-value>

@WebInitParam 注解的示例代码如下:

```
@WebServlet(name = "thirdServlet", urlPatterns = {"/thirdServlet"}, initParams = {@WebInitParam
(name = "firstParam", value = "one"),@WebInitParam(name = "secondParam", value = "two")})
```

5.2.3 实践环节——@WebServlet 的应用

首先将 web.xml 文件中有关 Servlet 部署的代码删除,然后使用注解的方式部署例 5-1 的 Servlet,并运行部署后的 Servlet。

扫一扫

视频讲解

5.3 Servlet 的生命周期

一个 Servlet 对象的生命周期主要由以下 3 个过程组成。

(1) 初始化 Servlet 对象:当 Servlet 对象第一次被请求加载时,服务器会创建一个 Servlet 对象,该 Servlet 对象调用 init()方法完成必要的初始化工作。

(2) service()方法响应请求:创建的 Servlet 对象调用 service()方法响应客户的请求。

(3) Servlet 对象死亡:当服务器关闭时,Servlet 对象调用 destroy()方法使自己消亡。

从上面 3 个过程来看,init()方法只能被调用一次,即在 Servlet 第一次被请求加载时调用。当客户端请求 Servlet 服务时,服务器将启动一个新的线程,在该线程中 Servlet 对象调用 service()方法响应客户端的请求。那么当多客户端请求 Servlet 服务时服务器会怎么处理?服务器会为每个客户端启动一个新的线程,在每个线程中 Servlet 对象调用 service()方法响应客户端的请求。也就是说,每个客户端请求都会导致 service()方法被调用执行,分别运行在不同的线程中。

【例 5-3】 Servlet 接口的 init()、service()和 destroy()方法。

例 5-3 的 ThirdServlet.java 的代码如下:

```
package servlet;
import java.io.IOException;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebInitParam;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
```

```
@WebServlet(name = "thirdServlet", urlPatterns = {"/thirdServlet"},
initParams = {@WebInitParam(name = "firstParam", value = "one"),
      @WebInitParam(name = "secondParam", value = "two")})
public class ThirdServlet extends HttpServlet {
 private static final long serialVersionUID = 1L;
 private String first = null;
 private String second = null;
 private static int count = 0;
 public void init(ServletConfig config) throws ServletException {
      //获取 firstParam 参数的值
      first = config.getInitParameter("firstParam");
      second = config.getInitParameter("secondParam");
      System.out.println("第一个参数值:" + first);
      System.out.println("第二个参数值:" + second);
 }
 protected void service(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
      count ++;
      System.out.println("您是第" + count + "个客户端请求该 Servlet!");
 }
 public void destroy() {
 }
}
```

在 ThirdServlet 的 init()方法中,通过 ServletConfig 的 config 对象调用 getInitParameter()方法来获取参数的值。当请求 3 次该 Servlet 以后,在 IntelliJ IDEA 的控制台中打印出如图 5.2 所示的结果。

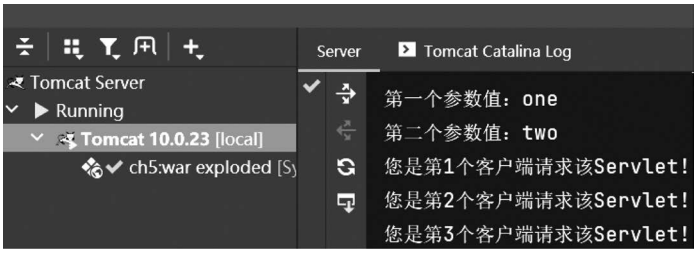

图 5.2 请求 3 次 thirdServlet 的结果

从图 5.2 可以看出,不管请求几次 thirdServlet,它的 init()方法只执行一次,而 service()方法每请求一次就执行一次。

扫一扫

视频讲解

5.4 通过 JSP 页面访问 Servlet

用户可以通过 JSP 页面的表单或超链接请求某个 Servlet。通过 JSP 页面访问 Servlet 的好处是,JSP 页面负责页面的静态信息处理,动态信息处理由 Servlet 完成。

❶ 通过表单访问 Servlet

假设在 JSP 页面中有如下表单:

```
<form action = "isLogin" method = "post">
      ...
</form>
```

那么该表单的处理程序(action)就是一个 Servlet,在为该 Servlet 部署时,@WebServlet 的 urlPatterns 属性值为“{"/isLogin"}”。

❷ 通过超链接访问 Servlet

在 JSP 页面中可以单击超链接访问 Servlet 对象，也可以通过超链接向 Servlet 提交信息。例如“< a href＝"loginServlet?user＝taipingle&&pwd＝zhenzuile">查看用户名和密码”，“查看用户名和密码”这个超链接将 user＝taipingle 和 pwd＝zhenzuile 两个信息提交给 Servlet 处理。

【例 5-4】 编写 JSP 页面 login.jsp，在该页面中通过表单向 urlPatterns 为“{"/loginServlet"}”的 Servlet(由 LoginServlet 类负责创建)提交用户名和密码，Servlet 负责判断输入的用户名和密码是否正确，并把判断结果返回。页面的运行效果如图 5.3 所示。

(a) 信息输入页面

(b) 信息正确页面

(c) 信息错误页面

图 5.3 通过 JSP 页面访问 Servlet

login.jsp 的代码如下：

```
<%@ page language = "java" contentType = "text/html; charset = UTF - 8" pageEncoding = "UTF - 8" %>
<! DOCTYPE html >
< html >
< head >
< meta charset = "UTF - 8">
< title > login.jsp </title >
</head >
  < body >
    < form action = "loginServlet" method = "post">
      < table >
            < tr >
                < td >用户名:</td >
                < td >< input type = "text" name = "user"/></td >
            </tr >
            < tr >
                < td >密　码:</td >
                < td >< input type = "password" name = "pwd"/></td >
            </tr >
            < tr >
                < td >< input type = "submit" value = "提交"/></td >
                < td >< input type = "reset" value = "重置"/></td >
            </tr >
      </table >
    </form >
  </body >
</html >
```

LoginServlet.java 的代码如下：

```
package servlet;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
```

```
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
@WebServlet(name = "loginServlet", urlPatterns = {"/loginServlet"})
public class LoginServlet extends HttpServlet {
 private static final long serialVersionUID = 1L;
 protected void service ( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      String name = request.getParameter("user");        //获取客户提交的信息
      String password = request.getParameter("pwd");     //获取客户提交的信息
      out.println("<html><body>");
      if(name == null || name.length() == 0){
           out.println("请输入用户名");
      }
      else if(password == null || password.length() == 0){
           out.println("请输入密码");
      }
      else if(name.length() > 0 && password.length() > 0){
           if(name.equals("zhangsan") && password.equals("lisi")){
                out.println("信息输入正确");
           }else{
                out.println("信息输入错误");
           }
      }
      out.println("</body></html>");
 }
}
```

扫一扫

视频讲解

5.5 doGet()和doPost()方法

当服务器接收到一个Servlet请求时会产生一个新线程，在这个线程中让Servlet对象调用service()方法为请求做出响应。service()方法首先检查HTTP请求类型(get或post)，并在service()方法中根据用户的请求方式对应地调用doGet()或doPost()方法。

当HTTP请求类型为get方式时，service()方法调用doGet()方法响应用户请求；当HTTP请求类型为post方式时，service()方法调用doPost()方法响应用户请求，因此在Servlet类中没有必要重写service()方法，直接继承即可。

在Servlet类中通过重写doGet()或doPost()方法来响应用户的请求，这样可以增加响应的灵活性，同时减轻服务器的负担。

在一般情况下，如果不论用户的请求类型是get还是post，服务器的处理过程完全相同，那么可以只在doPost()方法中编写处理过程，而在doGet()方法中调用doPost()方法；或只在doGet()方法中编写处理过程，而在doPost()方法中调用doGet()方法。

【例5-5】 编写JSP页面inputLader.jsp，在该页面中使用表单向urlPatterns为“{"/getLengthOrAreaServlet"}”的Servlet提交矩形的长与宽。Servlet(由GetLengthOrAreaServlet负责创建)的处理手段依赖表单提交数据的方式，当提交方式为get时，Servlet计算矩形的周长；当提交方式为post时，Servlet计算矩形的面积。页面的运行效果如图5.4所示。

(a) 信息输入页面

(b) 以post方式提交获取矩形的面积

(c) 以get方式提交获取矩形的周长

图 5.4　计算矩形的面积和周长

inputLader.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>inputLader.jsp</title>
</head>
<body>
    <h2>输入矩形的长和宽，提交给 Servlet(post 方式)求面积:</h2>
    <form action="getLengthOrAreaServlet" method="post">
      长:<input type="text" name="length"/><br/>
      宽:<input type="text" name="width"/><br/>
      <input type="submit" value="提交"/>
    </form>
     <br/>
    <h2>输入矩形的长和宽，提交给 Servlet(get 方式)求周长:</h2>
    <form action="getLengthOrAreaServlet" method="get">
      长:<input type="text" name="length"/><br/>
      宽:<input type="text" name="width"/><br/>
      <input type="submit" value="提交"/>
    </form>
  </body>
</html>
```

GetLengthOrAreaServlet.java 的代码如下：

```
package servlet;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
```

```
import jakarta.servlet.http.HttpServletResponse;
@WebServlet(name = "getLengthOrAreaServlet", urlPatterns = {"/getLengthOrAreaServlet"})
public class GetLengthOrAreaServlet extends HttpServlet {
 private static final long serialVersionUID = 1L;
 protected void doGet ( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String l = request.getParameter("length");
        String w = request.getParameter("width");
        out.println("<html><body>");
        double m = 0,n = 0;
        try{
            m = Double.parseDouble(l);
            n = Double.parseDouble(w);
            out.println("矩形的周长是:" + ( m + n ) * 2);
        }catch(NumberFormatException e){
            out.println("请输入数字字符!");
        }
        out.println("</body></html>");
 }
 protected void doPost ( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String l = request.getParameter("length");
        String w = request.getParameter("width");
        out.println("<html><body>");
        double m = 0, n = 0;
        try{
            m = Double.parseDouble(l);
            n = Double.parseDouble(w);
            out.println("矩形的面积是:" + m * n);
        }catch(NumberFormatException e){
            out.println("请输入数字字符!");
        }
        out.println("</body></html>");
 }
}
```

扫一扫

视频讲解

5.6 重定向与转发

重定向是将用户从当前 JSP 页面或 Servlet 定向到另一个 JSP 页面或 Servlet，以前的 request 中存放的信息全部失效，并进入一个新的 request 作用域；转发是将用户对当前 JSP 页面或 Servlet 的请求转发给另一个 JSP 页面或 Servlet，以前的 request 中存放的信息不会失效。

▶ 5.6.1 重定向

在 Servlet 中通过调用 HttpServletResponse 类中的 sendRedirect(String location)方法来实现重定向，重定向的目标页面或 Servlet(由参数 location 指定)无法从以前的 request 对象中获取用户提交的数据。

▶ 5.6.2 转发

使用 jakarta.servlet.RequestDispatcher 对象可以将用户对当前 JSP 页面或 Servlet 的请求转发给另一个 JSP 页面或 Servlet,实现转发需要以下两个步骤。

❶ 获取 RequestDispatcher 对象

在当前 JSP 页面或 Servlet 中使用 request 对象调用 public RequestDispatcher getRequestDispatcher(String url)方法返回一个 RequestDispatcher 对象,其中参数 url 就是要转发的 JSP 页面或 Servlet 的地址。例如:

```
RequestDispatcher dis = request.getRequestDispatcher("dologin");
```

❷ 使用 RequestDispatcher 对象调用 forward()方法实现转发

在获取 RequestDispatcher 对象之后,就可以使用该对象调用 public void forward(ServletRequest request,ServletResponse response)方法将用户对当前 JSP 页面或 Servlet 的请求转发给 RequestDispatcher 对象所指定的 JSP 页面或 Servlet。例如:

```
dis.forward(request,response);
```

转发是服务器行为,重定向是客户端行为。其具体工作流程如下。

转发过程:客户浏览器发送 HTTP 请求,Web 服务器接受此请求,调用内部的一个方法在容器内部完成请求处理和转发动作,将目标资源发送给客户。在这里转发的路径必须是同一个 Web 容器下的 URL,其不能转发到其他的 Web 路径上,中间传递的是自己容器内的 request。在客户浏览器的地址栏中显示的仍然是其第一次访问的路径,也就是说客户是感觉不到服务器做了转发的。转发行为是浏览器只做了一次访问请求。

重定向过程:客户浏览器发送 HTTP 请求,Web 服务器接受后发送 302 状态码响应及对应的新的 location 给客户浏览器,客户浏览器发现是 302 响应,则自动发送一个新的 HTTP 请求,请求 URL 是新的 location 地址,服务器根据此请求寻找资源并发送给客户。在这里 location 可以重定向到任意 URL,既然是浏览器重新发出了请求,则就没有什么 request 传递的概念了。在客户浏览器的地址栏中显示的是其重定向的路径,客户可以观察到地址的变化。重定向行为是浏览器至少做了两次访问请求。

【例 5-6】 编写 JSP 页面 redirectForward.jsp,在该 JSP 页面中通过表单向 urlPatterns 为"{"/redirectForwardServlet"}"的 Servlet(由 RedirectForwardServlet 负责创建)提交用户名和密码。如果用户输入的数据不完整,redirectForwardServlet 将用户重定向到 redirectForward.jsp 页面;如果用户输入的数据完整,redirectForwardServlet 将用户对 redirectForward.jsp 页面的请求转发给 urlPatterns 为"{"/showServlet"}"的 Servlet(由 ShowServlet 负责创建),showServlet 显示用户输入的信息。

redirectForward.jsp 的代码如下:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>redirectForward.jsp</title>
</head>
  <body>
    <form action="redirectForwardServlet" method="post">
      <table>
```

```
            <tr>
                <td>用户名:</td>
                <td><input type="text" name="user"/></td>
            </tr>
            <tr>
                <td>密　码:</td>
                <td><input type="password" name="pwd"/></td>
            </tr>
            <tr>
                <td><input type="submit" value="提交"/></td>
                <td><input type="reset" value="重置"/></td>
            </tr>
        </table>
    </form>
  </body>
</html>
```

RedirectForwardServlet.java 的代码如下：

```
package servlet;
import java.io.IOException;
import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
@WebServlet(name = "redirectForwardServlet", urlPatterns = {"/redirectForwardServlet"})
public class RedirectForwardServlet extends HttpServlet {
 private static final long serialVersionUID = 1L;
 protected void doGet(HttpServletRequest request,
            HttpServletResponse response)throws ServletException, IOException {
      doPost(request, response);
 }
 protected void doPost(HttpServletRequest request,
            HttpServletResponse response)throws ServletException, IOException {
      String name = request.getParameter("user");
      String password = request.getParameter("pwd");
      if (name == null || name.length() == 0) {
            //使用 response 调用 sendRedirect()方法重定向到 redirectForward.jsp
            response.sendRedirect("redirectForward.jsp");
      } else if (password == null || password.length() == 0) {
            response.sendRedirect("redirectForward.jsp");
      } else if (name.length() > 0 && password.length() > 0) {
            //转发
            RequestDispatcher dis = request.getRequestDispatcher("showServlet");
            dis.forward(request, response);
      }
 }
}
```

ShowServlet.java 的代码如下：

```
package servlet;
import java.io.IOException;
import java.io.PrintWriter;
```

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
@WebServlet(name = "showServlet", urlPatterns = {"/showServlet"})
public class ShowServlet extends HttpServlet {
 private static final long serialVersionUID = 1L;
 protected void doGet(HttpServletRequest request,
             HttpServletResponse response)throws ServletException, IOException {
       doPost(request, response);
 }
 protected void doPost(HttpServletRequest request,
             HttpServletResponse response)throws ServletException, IOException {
       response.setContentType("text/html;charset=UTF-8");
       PrintWriter out = response.getWriter();
       String name = request.getParameter("user");
       String password = request.getParameter("pwd");
       out.println("您的用户名是:" + name);
       out.println("<br>您的密码是:" + password);
 }
}
```

▶ 5.6.3 实践环节——登录验证

编写登录页面 login_1.jsp,在该 JSP 页面中通过表单向 urlPatterns 为“{"/loginServlet_1"}”的 Servlet(由 LoginServlet_1 类负责创建)提交用户名和密码。如果用户输入的数据不完整,loginServlet_1 将用户重定向到 login_1.jsp 页面;如果用户输入的数据完整并正确(用户名为“zhangsan”,密码为“123”),loginServlet_1 将用户的请求转发给 loginSuccess_1.jsp 页面,loginSuccess_1.jsp 页面显示用户输入的信息。

扫一扫

视频讲解

5.7 在 Java Servlet 中使用 session

在 Servlet 中获取当前请求的会话对象可以通过调用 HttpServletRequest 的 getSession()方法实现。例如:

```
HttpSession session = request.getSession(true);   //若存在会话返回该会话,否则新建一个会话
```

或

```
HttpSession session = request.getSession(false);   //若存在会话返回该会话,否则返回 null
```

在通常情况下,通过第一种方式获取 session,即指定 getSession()的参数为 true,true 是默认值,也就是 request.getSession(true)等同于 request.getSession()。

【例 5-7】 编写 JSP 页面 useSession.jsp,在该页面中通过表单向名为 useSession 的 Servlet 对象(由 UseSessionServlet 类负责创建)提交用户名,useSession 将用户名存入用户的 session 对象中,然后用户请求另一个 Servlet 对象 showName(由 ShowNameServlet 类负责创建),showName 从用户的 session 对象中取出存储的用户名,并显示在浏览器中。页面的运行效果如图 5.5 所示。

(a) 信息输入页面

(b) 获取会话并存储数据

(c) 获取会话中的数据并显示

图 5.5 在 Servlet 中使用 session

useSession.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>useSession.jsp</title>
   </head>
   <body>
     <form action="sendMyName" method="post">
        <table>
              <tr>
                    <td>用户名:</td>
                    <td><input type="text" name="user"/></td>
              </tr>
              <tr>
                    <td><input type="submit" value="提交"/></td>
              </tr>
        </table>
     </form>
   </body>
</html>
```

UseSessionServlet.java 的代码如下：

```
package servlet;
import java.io.*;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
@WebServlet(name = "useSession", urlPatterns = {"/sendMyName"})
public class UseSessionServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;
  public void init(ServletConfig config) throws ServletException {
       super.init(config);
  }
  public void doPost(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException {
       response.setContentType("text/html;charset=UTF-8");
       PrintWriter out = response.getWriter();
```

```
        String name = request.getParameter("user");
        if (null == name || name.trim().length() == 0) {
              response.sendRedirect("useSession.jsp");
        } else {
              HttpSession session = request.getSession(true);
              session.setAttribute("myName", name);
              out.println("<htm><body>");
              out.println("您请求的 Servlet 对象是:" + getServletName());
              out.println("<br>您的会话 ID 是:" + session.getId());
              out.println("<br>请单击请求另一个 Servlet:");
              out.println("<br><a href='showMyName'>请求另一个 Servlet</a>");
              out.println("</body></htm>");
        }
    }
    public void doGet(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException {
        doPost(request, response);
    }
}
```

ShowNameServlet.java 的代码如下：

```
package servlet;
import java.io.*;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
@WebServlet(name = "showName", urlPatterns = {"/showMyName"})
public class ShowNameServlet extends HttpServlet {
 private static final long serialVersionUID = 1L;
 public void init(ServletConfig config) throws ServletException {
        super.init(config);
 }
 public void doPost(HttpServletRequest request, HttpServletResponse response)
             throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        HttpSession session = request.getSession(true);
        String name = (String) session.getAttribute("myName");
        out.println("<htm><body>");
        out.println("您请求的 Servlet 对象是:" + getServletName());
        out.println("<br>您的会话 ID 是:" + session.getId());
        out.println("<br>您的会话中存储的用户名是:" + name);
        out.println("</body></htm>");

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

扫一扫

视频讲解

5.8 基于 Servlet 的 MVC 模式

本节将重点介绍基于 Servlet 的 MVC 模式。

5.8.1 MVC 模式

❶ MVC 的概念

MVC 是 Model、View、Controller 的缩写，分别代表 Web 应用程序中的 3 种职责。

（1）模型：用于存储数据以及处理用户请求的业务逻辑。

（2）视图：向控制器提交数据，显示模型中的数据。

（3）控制器：根据视图提出的请求判断将请求和数据交给哪个模型处理以及将处理后的有关结果交给哪个视图更新显示。

❷ 基于 Servlet 的 MVC 模式的具体实现

基于 Servlet 的 MVC 模式的具体实现如下。

（1）模型：一个或多个 JavaBean 对象，用于存储数据（实体模型，由 JavaBean 类创建）和处理业务逻辑（业务模型，由一般的 Java 类创建）。

（2）视图：一个或多个 JSP 页面，向控制器提交数据和为模型提供数据显示，JSP 页面主要使用 HTML 标记和 JavaBean 标记来显示数据。

（3）控制器：一个或多个 Servlet 对象，根据视图提交的请求进行控制，即将请求转发给处理业务逻辑的 JavaBean，并将处理结果存放到实体模型 JavaBean 中，输出给视图显示。

基于 Servlet 的 MVC 模式的工作流程如图 5.6 所示。

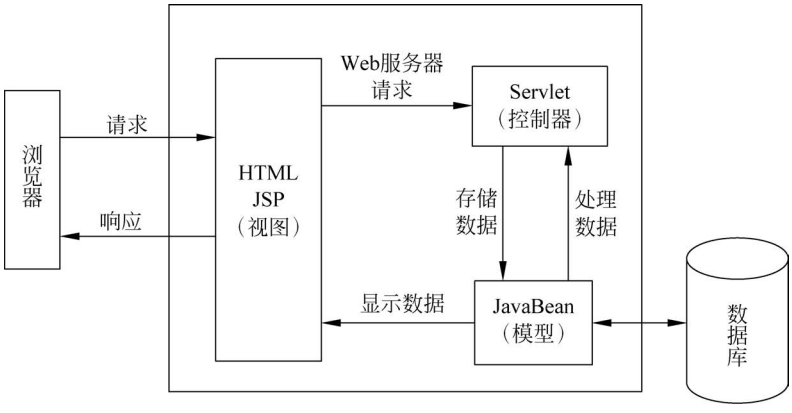

图 5.6 JSP 中的 MVC 模式

5.8.2 使用 JSP、Servlet 和 JavaBean 实现 MVC

【例 5-8】 使用 MVC 模式实现简单的用户登录验证程序，其中包括实体模型 User、业务模型 UserCheck、控制器 LoginCheckServlet 和两个视图页面，即登录页面和登录成功页面。

❶ 定义实体模型来表示数据

User 类用于创建实体模型存储用户信息，代码如下：

```
package dto;
public class User {
 private String name;
 private String pwd;
 public String getName() {
       return name;
 }
 public void setName(String name) {
       this.name = name;
 }
 public String getPwd() {
       return pwd;
 }
 public void setPwd(String pwd) {
       this.pwd = pwd;
 }
}
```

❷ 定义业务模型来处理业务

UserCheck 类用于判断用户名和密码是否正确，代码如下：

```
package service;
import dto.User;
public class UserCheck {
 //验证登录
 public boolean validate(User user) {
       if (user != null && user.getName().equals("JSPMVC")) {
             if (user.getPwd().equals("MVC")) {
                   return true;
             }
             return false;
       }
       return false;
 }
}
```

❸ 编写 Servlet 处理请求

LoginCheckServlet 用于完成请求控制，代码如下：

```
package servlet;
import java.io.IOException;
import jakarta.servlet.RequestDispatcher;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import service.UserCheck;
import dto.User;
@WebServlet(name = "loginCheckServlet", urlPatterns = {"/loginCheckServlet"})
public class LoginCheckServlet extends HttpServlet {
 private static final long serialVersionUID = 1L;
 protected void doGet(HttpServletRequest request,
             HttpServletResponse response)throws ServletException, IOException {
       doPost(request, response);
 }
 protected void doPost(HttpServletRequest request,
             HttpServletResponse response)throws ServletException, IOException {
       String name = request.getParameter("name");
       String pwd = request.getParameter("pwd");
```

```
        User user = new User();          //实例化实体模型 user
        user.setName(name);              //把数据存储在模型 user 中
        user.setPwd(pwd);                //把数据存储在模型 user 中
        UserCheck uc = new UserCheck(); //实例化业务模型 userCheck
        if (uc.validate(user)) {
              //把装有数据的实体模型 user 存储在 request 范围内
              request.setAttribute("user", user);
              RequestDispatcher dis = request
                        .getRequestDispatcher("loginSuccess.jsp");
              dis.forward(request, response);
        } else {
              response.sendRedirect("loginCheck.jsp");
        }
  }
}
```

❹ 编写视图

登录页面 loginCheck.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>loginCheck.jsp</title>
  </head>
  <body>
     <form action="loginCheckServlet" method="post">
       <table>
             <tr>
                  <td>用户名:</td>
                  <td><input type="text" name="name"/></td>
             </tr>
             <tr>
                  <td>密　码:</td>
                  <td><input type="password" name="pwd"/></td>
             </tr>
             <tr>
                  <td><input type="submit" value="提交"/></td>
                  <td><input type="reset" value="重置"/></td>
             </tr>
       </table>
     </form>
  </body>
</html>
```

登录成功页面 loginSuccess.jsp 的代码如下：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>loginSuccess.jsp</title>
</head>
<body>
    <jsp:useBean id="user" type="dto.User" scope="request"/>
    恭喜<jsp:getProperty property="name" name="user"/>登录成功!
</body>
</html>
```

▶ 5.8.3 模型周期

在基于 Servlet 的 MVC 模式中，控制器 Servlet 创建的实体模型 JavaBean 也涉及生命周期，生命周期分别为 request、session 和 application。下面以例 5-8 中的实体模型 user 来讨论这 3 种生命周期的用法。

❶ request 周期的模型

使用 request 周期的模型一般需要以下几个步骤：

1）创建模型并把数据保存到模型中

在 Servlet 中需要如下代码：

```
User user = new User();                    //实例化模型 user
user.setName(name);                        //把数据存储在模型 user 中
user.setPwd(pwd);                          //把数据存储在模型 user 中
```

2）将模型保存到 request 对象中并转发给 JSP 视图

在 Servlet 中需要如下代码：

```
request.setAttribute("user", user);  //把装有数据的模型 user 输出给视图页面 loginSuccess.jsp
RequestDispatcher dis = request.getRequestDispatcher("loginSuccess.jsp");
dis.forward(request, response);
```

request.setAttribute("user", user)这句代码指定了查找 JavaBean 的关键字，并决定了 JavaBean 的生命周期为 request。

3）视图的更新

Servlet 所转发的页面，比如 loginSuccess.jsp 页面，必须使用 useBean 标记获取 Servlet 所创建的 JavaBean 对象(视图不负责创建 JavaBean)。在 JSP 页面中需要使用如下代码：

```
<jsp:useBean id = "user" type = "dto.User" scope = "request"/>
<jsp:getProperty property = "name" name = "user"/>
```

标记中的 id 就是 Servlet 所创建的模型 JavaBean，它和 request 对象中的关键字对应。因为在视图中不创建 JavaBean 对象，所以在 useBean 标记中使用 type 属性而不使用 class 属性。useBean 标记中的 scope 必须和存储模型时的范围(request)一致。

❷ session 周期的模型

使用 session 周期的模型一般需要以下几个步骤：

1）创建模型并把数据保存到模型中

在 Servlet 中需要如下代码：

```
User user = new User();                //实例化模型 user
user.setName(name);                    //把数据存储在模型 user 中
user.setPwd(pwd);                      //把数据存储在模型 user 中
```

2）将模型保存到 session 对象中并转发给 JSP 视图

在 Servlet 中需要如下代码：

```
session.setAttribute("user", user);  //把装有数据的模型 user 输出给视图页面 loginSuccess.jsp
RequestDispatcher dis = request.getRequestDispatcher("loginSuccess.jsp");
dis.forward(request, response);
```

session.setAttribute("user", user)这句代码指定了查找 JavaBean 的关键字，并决定了 JavaBean 的生命周期为 session。

3）视图的更新

Servlet 所转发的页面，比如 loginSuccess.jsp 页面，必须使用 useBean 标记获取 Servlet 所创建的 JavaBean 对象（视图不负责创建 JavaBean）。在 JSP 页面中需要使用如下代码：

```
<jsp:useBean id = "user" type = "dto.User" scope = "session"/>
<jsp:getProperty property = "name" name = "user"/>
```

标记中的 id 就是 Servlet 所创建的模型 JavaBean，它和 session 对象中的关键字对应。因为在视图中不创建 JavaBean 对象，所以在 useBean 标记中使用 type 属性而不使用 class 属性。useBean 标记中的 scope 必须和存储模型时的范围（session）一致。

注意：对于生命周期为 session 的模型，Servlet 不仅可以使用 RequestDispatcher 对象转发给 JSP 页面，还可以使用 response 的重定向方法（sendRedirect()）定向到 JSP 页面。

❸ application 周期的模型

使用 application 周期的模型一般需要以下几个步骤：

1）创建模型并把数据保存到模型中

在 Servlet 中需要如下代码：

```
User user = new User ();                    //实例化模型 user
user.setName(name);                         //把数据存储在模型 user 中
user.setPwd(pwd);                           //把数据存储在模型 user 中
```

2）将模型保存到 application 对象中并转发给 JSP 视图

在 Servlet 中需要如下代码：

```
application.setAttribute("user", user); //把装有数据的模型 user 输出给视图页面 loginSuccess.jsp
RequestDispatcher dis = request.getRequestDispatcher("loginSuccess.jsp");
dis.forward(request, response);
```

application.setAttribute("user", user)这句代码指定了查找 JavaBean 的关键字，并决定了 JavaBean 的生命周期为 application。

3）视图的更新

Servlet 所转发的页面，比如 loginSuccess.jsp 页面，必须使用 useBean 标记获取 Servlet 所创建的 JavaBean 对象（视图不负责创建 JavaBean）。在 JSP 页面中需要使用如下代码：

```
<jsp:useBean id = "user" type = "dto.User" scope = "application"/>
<jsp:getProperty property = "name" name = "user"/>
```

标记中的 id 就是 Servlet 所创建的模型 JavaBean，它和 application 对象中的关键字对应。因为在视图中不创建 JavaBean 对象，所以在 useBean 标记中使用 type 属性而不使用 class 属性。useBean 标记中的 scope 必须和存储模型时的范围（application）一致。

注意：对于生命周期为 session 或 application 的模型，Servlet 不仅可以使用 RequestDispatcher 对象转发给 JSP 页面，还可以使用 response 的重定向方法（sendRedirect()）定向到 JSP 页面。

▶ 5.8.4 实践环节——四则运算

模仿例 5-8，使用基于 Servlet 的 MVC 模式设计一个 Web 应用，要求如下：

用户通过 JSP 页面 inputNumber.jsp 输入两个操作数，并选择一种运算符，单击“提交”按钮后，调用 HandleComputer.java 这个 Servlet。在 HandleComputer 中首先获取用户输入的数字和运算符并将这些内容存入实体模型（由 Computer.java 创建）中，然后调用业务模型（由 CalculateBean.java 创建）进行计算并把结果存入实体模型中，在 showResult.jsp 中调用

JavaBean 显示计算的结果。

5.9 本章小结

本章使用了 Servlet 的注解机制部署 Servlet，简化了 Servlet 的开发流程，使得 web.xml 部署描述文件从 Servlet 3.0 开始不再是必选的。

习 题 5

扫一扫

习题

扫一扫

自测题