第 5 章　Java Servlet 编程范例

Servlet 是在 Java 基础上的一种技术、一种标准。为了使 Java 在服务器端的程序更具效率，就发展了这样一种标准。Servlet 程序实际上就是 Java 程序，它是由服务器端调用和执行的 Java 类，可以把它看做运行在面向请求的服务器上的模块，它的功能与传统的 CGI（Common Gateway Interface，公共网关接口）相同。从严格意义上来讲，Servlet、JSP，还有 ASP、PHP 等 Web 开发语言都是 CGI，而传统的 CGI 主要由 Perl、Shell、Script、C 语言编写，我们平时讲的 CGI 一般是指这些传统的 CGI。Servlet 和传统的 CGI 相比有编写简单、运行稳健等优点，最主要的还是运行效率。从片面的意义上来看，Java 编写的程序需要通过 Java 虚拟机才能运行，所以 Java 程序很难比 C 语言甚至 Perl 运行得更快，但是这里讲的运行效率并不是指一个程序片运行一次的效率。由于 Web 编程的特殊性，在高访问量、高并发时能够提供更高的效率才是问题的关键。传统的 CGI 是当接收到客户请求时建立一个进程处理请求，处理完毕后进程结束。而 Servlet 被客户端发出的第一次请求激活，处理完请求后继续在后台运行以等待新的请求，每个请求将生成一个新的线程，而不是一个完整的进程。Servlet 只有在 Web 服务器关闭时才会卸载，这就保证了 Servlet 在高访问量、高并发时比传统的 CGI 有更高的执行效率。也正是这个原因，如果 Servlet 已经被激活，在修改 Servlet 又重新编译后需要重新启动 Web 服务器，这样对 Servlet 的修改才会生效。传统的 CGI 现在也有如 modperl、fastCGI 等方法使程序驻留在内存，也会使程序有非常高的运行效率，但由于其编写复杂、维护困难等原因被新技术替代已经成为必然。

Servlet 是 JSP 的基础，按照常理应该先讲 Servlet 再讲 JSP，可是 JSP 比 Servlet 更加通俗易懂，先知其然再知其所以然也不失为一个好办法。先有了感性的认识再逐步地探究其原理往往更能事半功倍，所以我们把 Servlet 放到 JSP 的后边。

在这一章我们要学习使用 Servlet 进行服务器端编程。Servlet 是 Java 的 Web 应用程序开发的最基本的技术，JSP 也是首先被编译转换成 Servlet 然后执行的。Servlet 为服务器端的代码和基于 Web 的客户端之间提供了一种简便的通讯形式，因此成为 Java 服务器端编程的核心。为了避免 Servlet 程序员考虑过多的网络连接、获取请求、产生特定格式的响应等细节问题，这些问题都由称作 Servlet Container（容器）或 Servlet Engine（引擎）的部分来完成，而 Servlet 开发人员只需要通过 Servlet API 实现业务层次的逻辑即可。Servlet Engine 和 Servlet API 都包含在 Java Servlet 开发人员工具包（Java Servlet Development Kit，JSDK）中，它可以从 http://java.sun.com/products/servlet 下载。本章将主要介绍使用 Servlet API 开发服务器端应用程序的方法。

5.1　简单的 Servlet 例子

5.1.1　实例说明

这个例子首先演示 Servlet 接收一个表单请求并将该请求中的数据显示出来，它的目的在

于让读者了解 Servlet 的基本结构和方法。同时，通过此例来解释 Servlet 中的一些基本概念，以及熟悉 Servlet 涉及的一些内嵌对象。

Java Servlet是以.java为后缀的源文件，通过javac编译成.class文件。为了运行这些Servlet实例，需要把编译好的 Servlet 的 class 文件放进 WEB-INF/classes 中。注意 Tomcat 默认要在使用 Servlet 时在浏览器地址栏加入/servlet/，例如我们要把下面的例子 servletExample.java 编译成为 servletExample.class，请确保把 servlet.jar 加入了 CLASSPATH 中，否则编译器将报告找不到javax.servlet的错误。在Tomcat-4.x中，Servlet.jar在$CATALINA_HOME/common/lib目录下，Servlet API 就包含在该包里。要使用 Servlet，只需要把编译好的 Servlet 类文件放在 $CATALINA_HOME/webapps/test/WEB-INF/classes/ 下，访问这个 Servlet 就要访问 http://localhost/test/servlet/servlet 类名。

5.1.2 代码分析

servletExample.java 的代码如下所示：

```
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
//继承 HttpServlet 来实现自己的 Servlet
public class servletExample extends HttpServlet {
//重载 HttpServlet 的 doGet 方法。首先设置内容类型和页面编码为
//text/html;charset=gb2312。然后输出 HTML 文件的头部信息
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws IOException, ServletException{
        response.setContentType("text/html;charset=gb2312");
//使用 response 的 out 对象输出 HTML 文件头
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<head>");
        out.println("<title>Servlet 示例</title>");
        out.println("</head>");
        out.println("<body bgcolor=\"white\">");
        out.println("<h3>Servlet 示例</h3>");
//接收客户端请求中传来的数据
        String username = request.getParameter("username");
        String password = request.getParameter("password");
         if (username!=null){//转换为中文字符集
             username=new String(username.getBytes("8859_1"),"gb2312");
         }
         out.println("<br>");
//将取到的数据输出到页面中
        if (username != null || password != null) {
           out.println("你的姓名是：");
           out.println(username + "<br>");
           out.println("你的密码是：");
```

```
            out.println(password);
        }
    //再在输出页面中产生一个表单，接受用户再输入姓名和密码信息
        out.print("<form action=\");
        out.print("servletExample\");
        out.println("method=POST>");
        out.println("姓名");
        out.println("<input type=text size=20 name=username>");
        out.println("<br>");
        out.println("密码");
        out.println("<input type=password size=20 name=password>");
        out.println("<br>");
        out.println("<input type=submit value='提交'>");
        out.println("</form>");
        out.println("</body>");
        out.println("</html>");
    }
    //doPost 方法的处理与 doGet 一致，即对 post 请求的处理与 get 请求一致
    public void doPost(HttpServletRequest request,
    HttpServletResponse response)throws IOException, ServletException{
        doGet(request, response);
    }
    }
```

这个 Servlet 继承了 HttpServlet 类。在 Servlet API 中定义了 GenericServlet 基类，它提供了 Servlet 接口的基本实现部分。HttpServlet 扩展了 GenericServlet 类并且支持 HTTP 协议的实现。HttpServlet 将是学习的重点，因为虽然 Servlet 可以支持多种协议，HTTP 仍然是 Servlet 使用最广泛的协议。我们的 Servlet 继承了 HttpServlet 类，它就获得了执行 HTTP 协议的能力。

方法 doGet 是这个 Servlet 的实现体，该方法是 Servlet 实际处理 HTTP 请求的方法。HTTP 协议中的请求类型有 GET、POST、HEAD 等，我们通常需要处理的是 GET 和 POST 类型。当收到 GET 类型的请求时，HttpServlet 中的实现会将请求分派给 doGet 方法。因此，只要将处理 GET 请求的代码放在 doGet 方法中，它就可用来处理 GET 请求。

doGet 方法有两个参数，分别是 HttpServletRequest 和 HttpServletResponse 类型。这两种类型都是 Servlet 规范中定义的接口，分别代表请求对象和应答对象。在实现 doGet 方法时，从 request 对象中获取请求的详细信息，并将处理结果输出到 response 中。结果的输出通过从 response 对象获得的 PrintWriter 对象完成。在 out 对象中写入 HTML 格式的文本，这个 HTML 就是实际的输出结果。也就是说，请求该 Servlet 返回的结果与下面的 HTML 页面的效果是一样的。

```
<html>
<head>
    <title>Servlet 示例</title>
</head>
<body bgcolor="white">
        <h3>Servlet 示例</h3>
<form action=\servletExample\ method=POST>
    姓名 <input type=text size=20 name=username>
    <br>
```

```
    密码 <input type=password size=20 name=password>
    <br>
    <input type=submit value='提交'>
</form>
</body>
</html>
```

5.1.3 运行结果

第一次访问时的页面如图 5-1 所示，在输入框中输入相应的信息后，单击“提交“按钮。该表单提交的结果页面如图 5-2 所示。

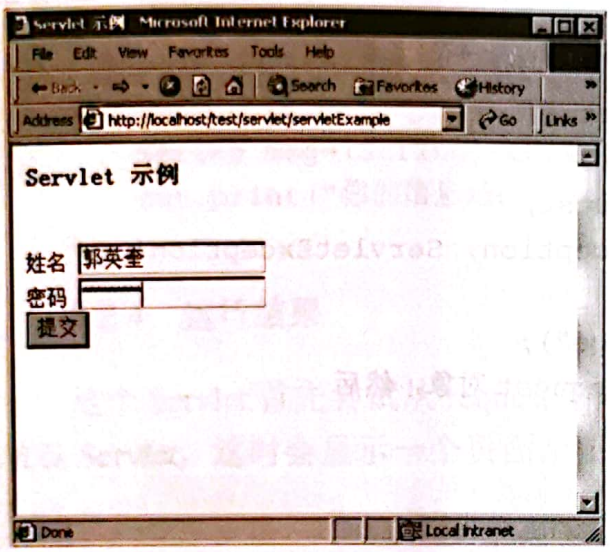

图 5-1 请求页面

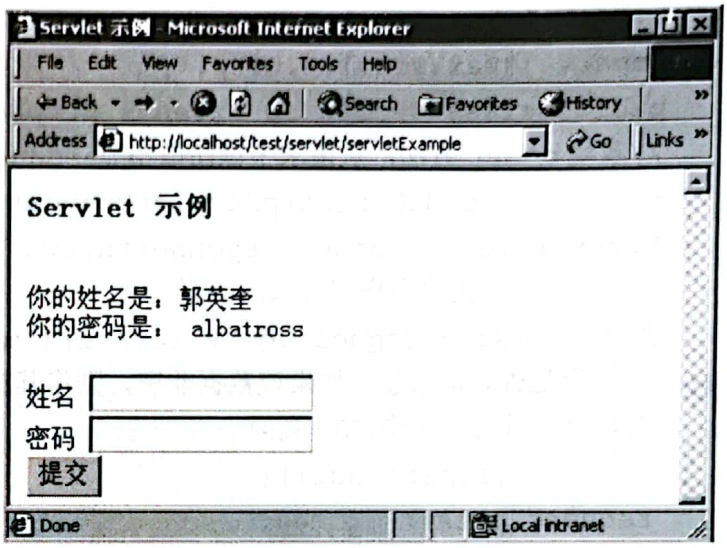

图 5-2 Servlet 输出页面

5.2 Servlet 与 JSP 之间的通信

5.2.1 实例说明

由于 JSP 是先被转换成 Servlet 然后编译的，所以在服务器端直接使用 Servlet 的效率要稍高于使用 JSP 的效率。但 JSP 是将 Servlet 以类似于 HTML 的 tag 方式表现出来，在易读性和输出页面方面要比 Servlet 有更大的方便性。基于它们各自的特点，Servlet 通常用来处理业务逻辑，而 JSP 用来处理页面的表现部分。因此它们之间的通讯是很必需的，下面通过一个例子来演示 Servlet 如何将结果返回给 JSP。

5.2.2 编程思路

程序之间的值的互相传递是编程时经常要用到的，其中最简单的办法是类似浏览器的 GET 方法，即直接使用 URL 传递值。这种方法简便直接，需要注意的是进行 URL 编码后才能保证值会被正确地传递。而且 URL 的长度是有限制的，如果需要传递的数据太大，即 URL 太长，就不能保证值会被正确地传递了。当然还可以通过 Session 来传递值，这种方法在第 4 章已经讲过。在 Servlet 中使用 Session 与在 JSP 中基本相同，这里就不介绍了。但是这种方法浪费系统资源而且效率不高。在 JSP 中还可以使用类似浏览器端的 POST 方法，这是在 ASP