第 5 章　Servlet 技术基础知识

Servlet 是一种服务器端的编程语言，是 J2EE 中比较关键的组成部分，Servlet 技术的推出，扩展了 Java 语言在服务器端开发的功能，巩固了 Java 语言在服务器端开发中的地位，而且现在使用非常广泛的 JSP 技术也是基于 Servlet 的原理，JSP+JavaBeans+Servlet 成为实现 MVC 模式的一种有效的选择。在本章中将介绍 Servlet 的基础知识，并通过具体的示例介绍 Servlet 的强大功能。

5.1　Servlet 简介

Servlet 在本质上就是 Java 类，编写 Servlet 需要遵循 Java 的基本语法，但是与一般 Java 类所不同的是，Servlet 是只能运行在服务器端的 Java 类，而且必需遵循特殊的规范，在运行的过程中有自己的生命周期，这些特性都是 Servlet 所独有的。另外 Servlet 是和 HTTP 协议是紧密联系的，所以使用 Servlet 几乎可以处理 HTTP 协议各个方面的内容，这也正是 Servlet 收到开发人员青睐的最大原因。

5.1.1　Servlet 的工作原理

Servlet 需要在特定的容器中才能运行，在这里所说的容器即 Servlet 运行的时候所需的运行环境，一般情况下，市面上常见的 Java Web Server 都可以支持 Servlet，例如 Tomcat、Resin、Weblogic、WebSphere 等，在本书中采用 Tomcat 作为 Servlet 的容器，由 Tomcat 为 Servlet 提供基本的运行环境。

Servlet 容器环境在 HTTP 通信和 web 服务器平台之间实现了一个抽象层。Servlet 容器负责把请求传递给 Servlet，并把结果返回结客户。容器环境也提供了配置 Servlet 应用的简单方法，并且也提供用 XML 文件配置 Servlet 的方法。当 Servlet 容器收到对用户对 Servlet 请求的时候，Servlet 引擎就会判断这个 Servlet 是否是第一次被访问，如果是第一次访问，Servlet 引擎就会初始化这个 Servlet，即调用 Servlet 中的 init（）方法完成必要的初始化工作，当后续的客户请求 Servlet 服务的时候，就不再调用 init（）方法，而是直接调用 service（）方法，也就是说每个 Servlet 只被初始化一次，后续的请求只是新建一个线程，调用 Servlet 中的 service（）方法。

在使用 Servlet 的过程中，并发访问的问题由 Servlet 容器处理，当多个用户请求同一个 Servlet 的时候，Servlet 容器负责为每个用户启动一个线程，这些线程的运行和销毁由 Servlet 容器负责，而在传统的 CGI 程序中，是为每一个用户启动一个进程，因此 Servlet 的运行效率就要比 CGI 的高出很多。

5.1.2　Servlet 的生命周期

Servlet 是运行在服务器端的程序，所以 Servlet 的运行状态完全由 Servlet 容器维护，一个 Servlet 的生命周期一般有三个过程。

1．初始化

当一个 Servlet 被第一请求的时候，Servlet 引擎就初始化这个 Servlet，在这里是调用 init（）方法完

成必需的初始化工作。而且这个对象一致在内存中活动，Servlet 为后续的客户请求新建线程，直接调用 Servlet 中的 service（）方法提供服务，不再初始化 Servlet。

2．提供服务

当 Servlet 对象被创建以后，就可以调用具体的 service（）方法为用户提供服务。

3．销毁

Servlet 被初始化以后一直再内存中保存，后续的访问可以不再进行初始化工作，当服务器遇到问题需要重新启动的时候，这些对象就需要被销毁，这时候 Servlet 引擎就会调用 Servlet 的 destroy（）方法把内存中的 Servlet 对象销毁。

5.1.3 简单 Servlet 开发配置示例

Java Servlet API 包括两个基本的包，javax.servlet 和 javax.servlet.http，其中 javax.servlet 提供了用来控制 Servlet 生命周期所需的类和接口，是编写 Servlet 必需要实现的。javax.servlet.http 提供了处理与 HTTP 相关操作的类和接口，每个 Servlet 必需实现 Servlet 接口，但是在实际的开发中，一般情况都是通过继承 javax.servlet.http.HttpServlet 或者 javax.servlet.GenericServlet 来间接实现 Servlet 接口。

下面这个简单的示例程序就是继承了 javax.servlet.http.HttpServle，从而实现 Servlet 接口，具体的代码如下。

```
//--------文件名：HelloWorld.java--------------------
package servlets;

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

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

在上面这个简单的示例程序中，继承了 HttpServlet，而 HttpServlet 是一个实现了 Servlet 接口的类，所以这个 Servlet 就间接地实现了 Servlet 的接口，从而可以使用接口提供的服务。

在这个程序中，并没有具体的 init（）方法和 destroy（）方法，这里使用 Servlet 容器默认的方式对这个 Servlet 进行初始化和销毁动作，而在这里的 doGet（）方法就是具体的功能处理方法，这个方法可

以对以 get 方法发起的请求进行处理，在这里这个方法的功能就是打印出一个 HTML 页面。

上面这个程序编译以后会生成对应的 Servlet 类文件，把编译生成的类文件拷贝到当前应用项目的 WEB-INF/classes 文件夹中，Servlet 引擎会到这个目录下面寻找 Servlet 的类文件。这里需要注意，如果 Servlet 有包名，这时候把需要把整个包拷贝，而不是仅仅拷贝类文件。负责 Servlet 引擎就会找不到 Servlet 类文件

Servlet 引擎需要通过配置文件来找到具体的 Servlet，在一般情况下都需要在当前应用项目的 web.xml 配置文件中对各个 Servlet 进行配置，其中 web.xml 文件的位置在当前项目应用的 WEB-INF 文件夹下。对于上面这个简单的 Servlet 示例，可以在 web.xml 中进行如下的配置。

```
<servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>servlets.HelloWorld</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

对于每一个 Servlet 都需要像上面这样的配置信息，这时 Servlet 引擎初始化 Servlet 所必需的信息，这个配置信息可以分为两个部分，第一部分是配置 Servlet 的名称和类，第二部分是配置 Servlet 的访问路径，其中<servlet-name>是这个 Servlet 的名称，这个名字可以任意命名，但是要和<servlet-mapping>节点中的<servlet-name>保持一致，<servlet-class>是 Servlet 对应类的路径，在这里要注意，如果有 Servlet 带有包名，一定要把包路径写全，否则 Servlet 引擎就找不到对应的 Servlet 类。<servlet-mapping>节点是对 Servlet 的访问路径进行配置，<url-pattern>定义了 Servlet 的访问路径。

经过上面的步骤，就可以使用类似 http://localhost:8080/chapt5/HelloWorld 这样的网址访问这个 Servlet，其中 http://localhost:8080/是本机 Tomcat 服务器的访问地址，chapt5 是当前应用项目的名称，HelloWorld 是 Servlet 的访问路径，当 Servlet 引擎接收到这样的请求的时候，就会初始化 HelloWorld 这个 Servlet，然后调用其中的方法为用户提供服务。

上面这个简单的示例程序的运行效果如图 5.1 所示。

图 5.1　HelloWorld 示例程序运行效果

总之，编写一个 Servlet 要经过以下几个步骤。

（1）编写 Servlet 的功能代码，即实现功能的代码类。

（2）把编译成功的 Servlet 功能代码类文件拷贝到当前应用项目的 WEB-INF/classes 目录下。

（3）在当前应用项目的 web.xml 文件中对 Servlet 进行配置，即在 web.xml 中添加配置信息。

经过这样三个步骤就可以通过浏览器访问这个 Servlet。

注意：Servlet 的配置信息需要添加在<web-app></web-app>标签之间。

5.1.4　使用 Servlet 实现 MVC 开发模式

Java 语言之所以受到开发人员支持，是因为 Java 语言实现科学方便的开发模式，在这些开发模式中，

最出色而且应用最广的就是 MVC 模式，对于 MVC 模式的研究由来已久，但是一直没有得到很好的推广和应用，随着 J2EE 技术的成熟，MVC 逐渐成为了一种常用而且重要的设计模式。

MVC（Model-View-Controller）把应用程序的开发分为三个层面：视图层、控制层、模型层。其中视图层负责从用户获取数据和向用户展示数据，在这层中不负责对业务逻辑的处理和数据流程的控制。而模型层负责处理业务逻辑和数据库的底层操作，其中视图层和模型层之间没有直接的联系。控制层主要负责处理视图层和模型层的交互，控制层从视图层接收请求，然后从模型层取出对请求的处理结果，并把结果返回给视图层。在控制层中只负责数据的流向，并不涉及具体的业务逻辑处理。

MVC 三层结构的内部关系如图 5.2 所示。

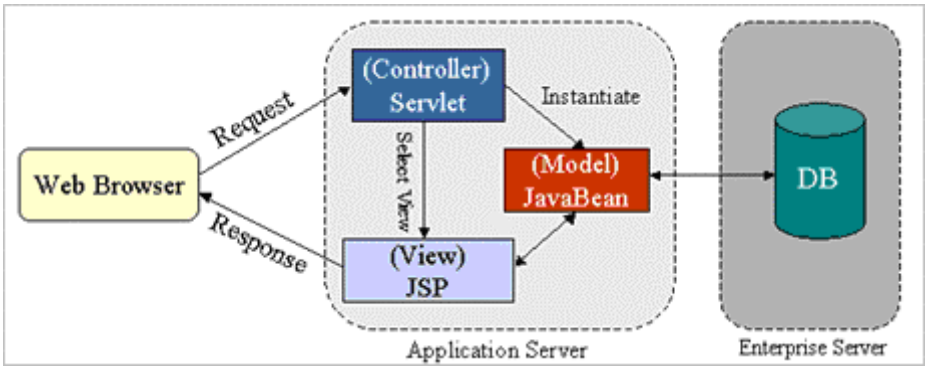

图 5.2　MVC 三层结构的内部关系

从图 5.2 中可以看出，Servlet 在 MVC 开发模式中承担着重要的角色，在 MVC 结构中，控制层就是依靠 Servlet 实现，Servlet 可以从浏览器端接收请求，然后从模型层取出处理结果，并且把处理结果返回给浏览器端的用户。在整个结构中，Servlet 负责数据流向控制的功能。

虽然现在用很多开源框架都很好的实现了 MVC 的开发模式，例如 Struts、WebWork 等，这些开源框架对 MVC 的实现都是非常出色的，在这些框架中，处理数据控制流向的时候，采用的还是 Servlet，例如在 Struts 中，对应每一个用户请求都有一个 Action，这个 Action 就是继承了 Servlet 的类，所以在 MVC 架构中，Servlet 是不可替代的。

5.2　JSP 页面调用 Servlet 的方法

在上面 HelloWorld 的示例程序中，我们直接在浏览器中输入具体的地址进行访问，在实际的应用中，不可能让用户在浏览器中直接输入 Servlet 的地址进行访问，一般情况下，可以通过调用 Servlet 进行访问，在这里介绍通过提交表单和超链接两种方式调用 Servlet。

5.2.1　通过表单提交调用 Servlet

在通过提交表单调用 Servlet 的时候，只需要把表单的 action 指向对应的 Servlet 即可，下面是一个简单的表单，通过这个表单可以调用指定的 Servlet。

```
//--------文件名：form.jsp--------------------
<%@ page language="java" import="java.util.*" contentType="text/html;charset=gb2312"%>
<html>
  <head>
```

```
    <title>Servlet 接收表单示例</title>
  </head>
  <body>
    <font size="2">
      <form action="AcceptForm" method="post">
        姓名：<input type="text" name="name"/><br>
        省份：<input type="text" name="province"><br>
        <input type="submit" value="提交">
      </form>
    </font>
  </body>
</html>
```

在上面这个表单中，指明表单的处理程序是 AcceptForm 这个 Servlet，下面是 AcceptForm 这个 Servlet 的具体代码。

```
//--------文件名：AcceptForm.java--------------------
package servlets;

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

public class AcceptForm extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");
        PrintWriter out = response.getWriter();
        String name = request.getParameter("name");
        String province = request.getParameter("province");

        out.println("<font size='2'>");
        out.print("提交的表单内容为:<br>");
        out.print("姓名:" + name + "<br>");
        out.print("省份:" + province + "<br>");
        out.print("</font>");   }

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

在上面这个 Servlet 中，只对 doGet（）方法进行重写，在 doPost（）方法中直接调用 doGet（）方法的具体内容。

下面来介绍这个 Servlet 的关键代码。

```
response.setContentType("text/html");
```

上面这行代码设置了服务器响应的内容格式为 HTML 文档格式。

```
response.setCharacterEncoding("gb2312");
```

这里设置服务器响应内容的字符编码格式为 gb2312。用来支持中文显示。

```
PrintWriter out = response.getWriter();
```

上面这行代码取出输出对象，用来在页面上输出要显示的内容。

```
String name = request.getParameter("name");
String province = request.getParameter("province");
```

上面这两行代码取出表单中的输出内容。

在前面 HelloWorld 的例子中已经介绍过，每个 Servlet 必需在 web.xml 中进行配置，然后 Servelt 引擎才能通过访问路径找到对应的 Servlet 类文件。上面这个 Servlet 的配置信息如下。

```
    <servlet>
        <servlet-name>AcceptForm</servlet-name>
        <servlet-class>servlets.AcceptForm</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>AcceptForm</servlet-name>
        <url-pattern>/AcceptForm</url-pattern>
    </servlet-mapping>
```

上面这个程序的执行效果如图 5.3 和图 5.4 所示。

图 5.3　提交表单内容

图 5.4　Servlet 处理表单结果

注意：在这里设置字符编码格式的操作要在取得输出对象之前，即在取出 PrintWriter 对象之前就要设置好字符的编码格式，如果在这之后设置，中文字符还是不能正常显示的。

5.2.2　通过超链接调用 Servlet

在上面这个例子中，用户有输入的内容需要提交给服务器，所以需要用表单来调用 Servlet，但是在没有输入的数据内容需要提交的情况下，使用表单就不是很合理了，在这里介绍 Servlet 的第二种调用方法，直接通过超链接的方式来调用 Servlet，在这种情况下还可以给 Servlet 传递参数。

下面的例子中就是使用超链接的方式调用 Servlet，并且向这个 Servlet 传递一个参数。这个调页面的代码如下。

```
//--------文件名：link.jsp--------------------
<%@ page language="java" import="java.util.*" contentType="text/html;charset=gb2312"%>
<html>
  <head>
    <title>Servlet 接收链接传递参数示例</title>
  </head>
  <body>
     <font size="2">
       单击下面的链接：<br>
       <a href="AcceptLink?name=Bill">调用 Servlet,并传递参数</a>
     </font>
  </body>
```

```
</html>
```

在上面这个 JSP 页面中，通过<a href="AcceptLink?name=Bill">这条超链接语句可以生成类似下面 http://localhost:8080/chapt5/AcceptLink?name=Bill 这样的访问地址，其中 AcceptLink 就是这个链接调用的 Servlet，并且这个链接还向 AcceptLink 这个 Servlet 传递了一个名为 name 的参数，这个参数的值为 Bill。AcceptLink 这个 Servlet 的具体代码如下。

```
//--------文件名：AcceptLink.java--------------------
package servlets;

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 AcceptLink extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");
        PrintWriter out = response.getWriter();
        String name = request.getParameter("name");

        out.println("<font size='2'>");
        out.println("链接传递过来的参数为：<br>");
        out.println("参数名称：name<br>");
        out.println("参数值：" + name + "<br>");
        out.print("</font>");
    }

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

在这个 Servlet 中，取出调用页面传递过来的参数，然后在页面打印，其中获取参数的方法仍然是 request.getParameter("name")。这个 Servlet 的配置信息如下。

```
    <servlet>
        <servlet-name>AcceptLink</servlet-name>
        <servlet-class>servlets.AcceptLink</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>AcceptLink</servlet-name>
        <url-pattern>/AcceptLink</url-pattern>
    </servlet-mapping>
```

上面这个示例程序的运行效果如图 5.5 和 5.6 所示。

图 5.5　超链接调用 Servlet 页面

图 5.6　Servlet 处理链接传递参数页面

5.3　Servlet 中的文件操作

在 JSP 的开发过程中，经常会遇到需要把相关内容存储为文件的情况，在 JSP 中是用输入输出流进行操作的，在 Servlet 中也可以使用输入输出流实现对文件的读写，同时，使用 Servlet 还可以很方便的实现文件的上传下载。接下来的内容将通过具体的示例展示 Servlet 文件操作的方法。

5.3.1　Servlet 读取文件

在这个例子中将要读取一个文本文件的内容，并且在页面上打印文件的内容。这个 Servlet 的代码如下。

```
//--------文件名：FileRead.java--------------------
package servlets;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FileRead extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");
        PrintWriter out = response.getWriter();
        String fileName =   "content.txt";
       String realPath = request.getRealPath(fileName);

        File file = new File(realPath);
     if(file.exists())
     {
```

```
            FileReader reader = new FileReader(file);
            BufferedReader bufferReader = new BufferedReader(reader);
            String line = null;
            while((line = bufferReader.readLine())!=null)
            {
                  out.print("<font size='2'>"line+"</font><br>");
            }
      }else
      {
            out.print("文件不存在！");
      }
     }

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

在上面这个 Servlet 中，首先从一个文本文件 content.txt 中读取内容，然后把读取的内容打印在页面。下面解释这个 Servlet 的关键代码。

```
response.setContentType("text/html");
response.setCharacterEncoding("gb2312");
```

上面两行代码设置服务器的响应内容类型是 HTML 格式，字符编码格式为 gb2312，用来支持中文的显示，如果没有在这里设置字符编码格式，页面上打印的中文内容就不能正常显示。

```
PrintWriter out = response.getWriter();
```

上面这行代码取得输出对象，用来在后面的操作中在页面上输出文件的内容。

```
String realPath = request.getRealPath(fileName);
```

上面这行代码把文件名转换为绝对路径，例如在这里文件名称为 content.txt，这个文本文件的位置在当前应用的根目录，那么通过上面这行代码就可以取得这个文件的绝对路径，转化以后的路径格式为 C:\Program Files\Apache Software Foundation\Tomcat 5.0\webapps\chapt5\content.txt,取得文件的绝对路径之后就可以使用 Java 的 API 尽心文件的读写操作。

```
File file = new File(realPath);
```

这行代码用文件的绝对路径构造出一个文件对象，后续的操作都可以在这个文件对象上面进行。

```
FileReader reader = new FileReader(file);
```

上面这行代码用文件对象构造了一个文件读取对象。

```
BufferedReader bufferReader = new BufferedReader(reader);
```

在 Java 文件操作中，为了提高文件读取的效率，提供了一个字符输入缓冲流类，这个类就是 BufferedReader，一般情况下 BufferedReader 和 FileReader 结合使用，在上面这行代码中，使用一个 FileReader 对象构造出一个 BufferedReader 对象。

```
while((line = bufferReader.readLine())!=null)
```

BufferedReader 类中有一个 readerLine（）方法，这个方法每次从 BufferedReader 对象中读取一行字符，在上面这行代码中循环读出 BufferedReader 中的每一行内容。

这个 Servlet 的配置信息如下。

```
    <servlet>
        <servlet-name>FileRead</servlet-name>
```

```
        <servlet-class>servlets.FileRead</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>FileRead</servlet-name>
        <url-pattern>/FileRead</url-pattern>
    </servlet-mapping>
```

上面这个 Servlet 可以用类似这样的地址进行访问 http://localhost:8080/chapt5/FileRead，其中 http://localhost:8080/是本机计算机的访问路径，chapt5 是当前应用项目的名称，FileRead 是文件读取 Servlet，这个示例程序的运行效果如图 5.7 所示。

图 5.7　文件读取 Servlet 运行效果

5.3.2　Servlet 写文件

Servlet 写文件的处理方法和读取文件的处理方法非常类似，只是把文件输入流换成文件输出流，在下面这个示例程序中，将在指定位置生成文件。这个 Servlet 的代码如下。

```
//--------文件名：FileWrite.java--------------------
package servlets;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FileWrite extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
        response.setCharacterEncoding("gb2312");
         PrintWriter out = response.getWriter();
         String fileName =   "new.txt";
         String realPath = request.getRealPath(fileName);

        File file = new File(realPath);
        FileWriter writer = new FileWriter(file);
```

```
            BufferedWriter bufferWriter = new BufferedWriter(writer);
            bufferWriter.write("计算机网络");
            bufferWriter.newLine();
            bufferWriter.write("高等数学");

            bufferWriter.flush();
            bufferWriter.close();
            writer.close();
            out.print("<font size='2'>文件写入成功，路径为:"+file.getAbsolutePath()+"</font>");
        }

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

上面这个 Servlet 可以在指定位置生成一个文件，并在这个文件中写入字符串，下面解释这个 Servlet 中的关键代码。

```
String fileName =   "new.txt";
String realPath = request.getRealPath(fileName);
```

上面这两行代码实现的功能还是把文件名转换成一个绝对路径。

```
File file = new File(realPath);
```

上面这行代码根据文件路径构造一个文件对象。需要注意的是，这个操作并没有在硬盘上生成指定的文件，这时候的文件对象仅仅存在内容中，并没有在硬盘上创建。

```
FileWriter writer = new FileWriter(file);
```

上面这行代码根据文件对象构造了一个 FileWriter 文件读取对象。

```
BufferedWriter bufferWriter = new BufferedWriter(writer);
```

同文件读取操作类似，为了提高文件写操作的效率，Java 语言提供了一个提供了一个字符输入缓冲流类，即 BufferedWriter，一般情况下 BufferedWriter 和 FileWriter 结合使用，在上面这行代码中，根据 FileWriter 对象构造一个 BufferedWriter 对象。

```
bufferWriter.write("计算机网络");
```

上面这行代码在 BufferedWriter 对象中写入一个字符串，这时候的操作还是在内存中进行，并不是在硬盘上的文件中写入内容，这个时候文件还没有生成。

```
bufferWriter.newLine();
```

上面这行代码实现的是换行操作。

```
bufferWriter.flush();
```

上面这行代码刷新字符输入缓冲区中的内容，这个时候才把内容中的文件对象输出到硬盘，即此时才在硬盘中生成指定的文件。

```
bufferWriter.close();
writer.close();
```

文件操作结束以后要对相关资源进行释放，上面两行代码分别关闭 FileWriter 对象和 BufferedWriter 对象，这两个对象释放以后文件操作就已经结束，如果在这两个对象关闭以后仍然在这两个对象上进行操作就会抛出异常。

上面这个 Servlet 的配置信息如下。

```
    <servlet>
```

```
        <servlet-name>FileWrite</servlet-name>
        <servlet-class>servlets.FileWrite</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>FileWrite</servlet-name>
        <url-pattern>/FileWrite</url-pattern>
    </servlet-mapping>
```

在配置工作完成以后就可以用类似 http://localhost:8080/chapt5/FileWrite 这样的地址进行访问，其中 http://localhost:8080/是本机计算机的访问路径，chapt5 是当前应用项目的名称，FileWrite 是写文件的 Servlet，这个示例程序的运行效果如图 5.8 所示。

图 5.8　Servlet 写文件示例运行效果

5.3.3　Servlet 上传文件

文件的上传下载在 Web 开发中会经常遇到，使用基本的 IO 输入输出流当然可以完成这项操作，但是出于对开发的效率和程序运行的效率方面的考虑，在实际的开发过程中一般采用第三方的组件来完成这个上传的功能。

在实际开发过程中用的比较多的是 commons-fileupload 组件和 jspSmartUpload 组件，这两个组件都可以很好地完成文件上传的功能，在本书中选择使用 commons-fileupload 组件，这个组件是 Apache 基金开发维护的，可以在 http://jakarta.apache.org/site/downloads 找到各种版本的下载，目前的版本为 1.2，下载下来的文件是一个压缩文件 commons-fileupload-1.2-bin.zip，解压这个文件可以得到如图 5.9 所示的目录结构。

图 5.9　commons-fileupload 组件的目录结构

在如图 5.9 所示的 lib 文件夹中有一个名为 commons-fileupload-1.2.jar 的文件，这个就是 commons-fileupload 组件的类库，把这个文件拷贝到当前应用项目的 WEB-INF\lib 文件夹中。

使用 commons-fileupload 组件的时候，需要 Apache 另外一个组件 commons-io 的支持，commons-io 组件也可以在 http://jakarta.apache.org/site/downloads 上找到，当前最新的版本为 1.3.2，下载下来的文件为 commons-io-1.3.2-bin.zip，把这个文件解压后可以得到如图 5.10 所示的目录结构。

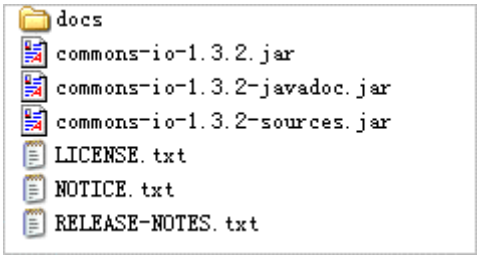

图 5.10　commons-io-1.3.2 组件的目录结构

把如图 5.10 所示的 commons-io-1.3.2.jar 拷贝到当前应用项目的 WEB-INF\lib 的文件夹下，这个时候 commons-fileupload 组件的配置工作就完成了，可以在项目中开始使用 commons-fileupload 组件提供的文件上传功能。

在下面这个例子中将详细介绍如何使用这个组件完成文件上传的功能。

下面这个页面是上传文件选择表单页面。

```
//--------文件名：upload.jsp--------------------
<%@ page contentType="text/html; charset=gb2312" %>
<html>
<head>
<title>commonfileupload 上传文件示例</title>
</head>
<body>
<font size="2">
commonfileupload 上传文件示例 <br>
<form method="post" action="FileUpload" ENCTYPE="multipart/form-data">
文件:<input type="file"    name="file">
<input type="submit" value="上传" name="submit">
</form>
</font>
</body>
</html>
```

在这里需要注意的是，上传文件的时候表单中需要添加 ENCTYPE="multipart/form-data"的属性，而且最好使用 POST 方法提交表单。

下面是接收表单实现文件上传的 Servlet。

```
//--------文件名：FileUpload.java-------------------
package servlets;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Iterator;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemFactory;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
```

```
import org.apache.commons.fileupload.servlet.ServletFileUpload;
public class FileUpload extends HttpServlet {
	public void doPost(HttpServletRequest request, HttpServletResponse response)
			throws IOException, ServletException {
		boolean isMultipart = ServletFileUpload.isMultipartContent(request);
		if (isMultipart) {
			FileItemFactory factory = new DiskFileItemFactory();
			ServletFileUpload upload = new ServletFileUpload(factory);
			Iterator items;
			try {
				items = upload.parseRequest(request).iterator();
				while (items.hasNext()) {
					FileItem item = (FileItem) items.next();
					if (!item.isFormField()) {
						//取出上传文件的文件名称
						String name = item.getName();
						String fileName = name.substring(name.lastIndexOf('\\')+1,name.length());
						String path = request.getRealPath("file")+File.separatorChar+fileName;
						//上传文件
						File uploadedFile = new File(path);
						item.write(uploadedFile);

						//打印上传成功信息
						response.setContentType("text/html");
						response.setCharacterEncoding("gb2312");
						PrintWriter out = response.getWriter();
						out.print("<font size='2'>上传的文件为："+name+"<br>");
						out.print("保存的地址为："+path+"</font>");
					}
				}
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	}
}
```

下面详细解释这个程序中的关键代码。

```
boolean isMultipart = ServletFileUpload.isMultipartContent(request);
```

上面这行代码可以判断出提交过来的表单是否为文件上传表单，如果不是文件上传表单，在后续的处理中就不再用文件上传功能来处理这个表单。

```
FileItemFactory factory = new DiskFileItemFactory();
ServletFileUpload upload = new ServletFileUpload(factory);
```

上面这两行代码构造了一个文件上传处理对象。

```
items = upload.parseRequest(request).iterator();
```

上面这行代码解析出表单中提交的所有文件内容。

```
String name = item.getName();
String fileName = name.substring(name.lastIndexOf('\\')+1,name.length());
String path = request.getRealPath("file")+File.separatorChar+fileName;
```

上面这段代码可以取出文件名，并且得出这个文件上传到服务器以后存储的路径。上传的文件将被

存储再当前应用项目的 file 文件夹中。

```
File uploadedFile = new File(path);
item.write(uploadedFile);
```

上面这两行代码把上传的文件存储到服务器中，完成上传操作。

这个文件上传 Servlet 的配置信息如下。

```
    <servlet>
        <servlet-name>FileUpload</servlet-name>
        <servlet-class>servlets.FileUpload</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>FileUpload</servlet-name>
        <url-pattern>/FileUpload</url-pattern>
    </servlet-mapping>
```

配置工作完成以后就可以浏览器中进行文件上传操作，这个示例程序的运行效果如图 5.11 和 5.12 所示。

图 5.11 文件选择界面

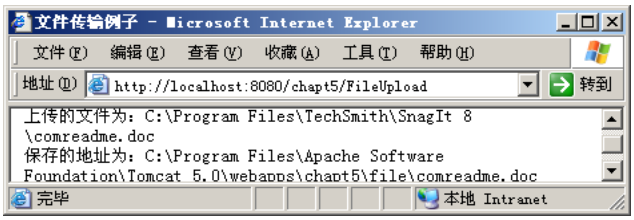

图 5.12 文件上传成功界面

在这个示例程序中，接收上传文件的 Servlet 可以分析出表单中所有的文件内容，所以如果要上传多个文件的，只需要表单中添加文件选择输入框<input type="file" name="newFile">即可，其中 name 属性可以任意命名。

5.3.4 Servlet 下载文件

用 Servlet 下载文件的时候，并不需要第三方组件的帮助，只需要对服务器的响应对象 response 进行简单的设置即可，下面的就是一个文件下载的示例程序，这个程序将从当前应用项目的根目录下载一个名为 test.xls 的 Excel 文档，具体代码如下。

```
//--------文件名：FileDownload.java--------------------
package servlets;

import java.io.FileInputStream;
import java.io.OutputStream;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class FileDownload extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        try {
            String fname = "test.xls";
            response.setCharacterEncoding("UTF-8");
            fname = java.net.URLEncoder.encode(fname, "UTF-8");
```

```
                response.setHeader("Content-Disposition", "attachment; filename="+fname);
                response.setContentType("application/msexcel");// 定义输出类型
            } catch (Exception e) {
                System.out.println(e);
            }
        }
}
```

下面来解释这个 Servlet 中的关键代码。

```
response.setCharacterEncoding("UTF-8");
fname = java.net.URLEncoder.encode(fname, "UTF-8");
```

上面这两行代码是对字符编码的设置，用来支持中文的文件名。

```
response.setHeader("Content-Disposition", "attachment; filename="+fname);
```

上面这行代码指明了这个 Servlet 的功能是输出文件，并且指明文件的位置。

```
response.setContentType("application/msexcel");// 定义输出类型
```

上面这行代码指明了要输入文件的类型，其中 application/msexcel 就是 Excel 文件的 MIME 类型描述。

这个 Servlet 的配置信息如下。

```
    <servlet>
        <servlet-name>FileDownload</servlet-name>
        <servlet-class>servlets.FileDownload</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>FileDownload</servlet-name>
        <url-pattern>/FileDownload</url-pattern>
    </servlet-mapping>
```

上面这个 Servlet 可以用类似这样的地址进行访问 http://localhost:8080/chapt5/FileDownload，其中 http://localhost:8080/是本机计算机的访问路径，chapt5 是当前应用项目的名称，FileDownload 是文件下载 Servlet，这个示例程序的运行效果如图 5.13 所示。

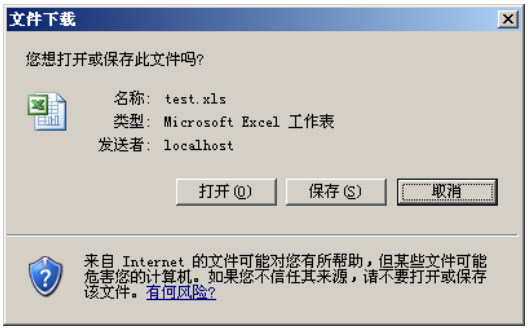

图 5.13　文件下载对话框

5.4　Servlet 过滤器

在 Web 应用中可以使用过滤器对所有的访问和请求进行统一的处理，IP 访问限制，用户发送请求的字符编码转换等，在进行具体的业务逻辑处理之前，首先要经过过滤器的统一处理，然后才开始进入真正的逻辑处理阶段。在本节内容中，将介绍过滤器的原理的实际应用。

5.4.1 过滤器的基本原理

过滤器的功能就是在服务器和客户中间增加了一个中间层，可以对两者之间的交互进行统一的处理，每一个从客户端提交的请求都需要通过过滤器的处理，然后再进行其他的操作。

在实际开发中，过滤器器可以用来对用户进行统一的身份判断、IP 访问限制，用户发送请求的字符编码转换、对请求和响应进行加密和解密、记录用户登录日志等。当然过滤器的用途不仅仅这些，读者可以根据过滤器的实现原理，思考过滤器更多的用途。

5.4.2 IP 访问 filter

在实际的应用中，可能会遇到这样的情况，需要对某些 IP 进行访问限制，不让非法的 IP 访问应用系统，这个时候就需要用到过滤器进行限制，当一个用户发出访问请求的时候，首先通过过滤器进行判断，如果用户的 IP 地址被限制，就禁止访问，只有合法的 IP 才可以继续访问。

这个 IP 访问限制过滤器的具体代码如下。

```
//--------文件名：IPFilter.java--------------------
package filters;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.UnavailableException;

public class IPFilter implements Filter {
    protected FilterConfig filterConfig;
    protected String ip;

    public void init(FilterConfig filterConfig) throws ServletException {
        this.filterConfig = filterConfig;
        this.ip = this.filterConfig.getInitParameter("ip");
    }

    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        String remoteIP = request.getRemoteAddr();
        if(remoteIP.equals(ip))
        {
            response.setCharacterEncoding("gb2312");
            PrintWriter out = response.getWriter();
            out.println("<b>你的 IP 地址被禁止访问。</b>");
        }else
        {
            chain.doFilter(request,response);
```

```
            }
        }
        public void destroy() {}
}
```

在上面这个过滤器中，主要的方法就是 init（）和 doFilter（）这两个方法，init（）方法是这个过滤器初始化的时候调用的，在这个过滤器中初始化的工作就是从配置文件中读取参数的内容，而 doFilter（）方法的功能就是这个过滤器真正要执行的处理功能，在这个示例程序中，对 IP 的限制就是在这个方法中实现的。

```
String remoteIP = request.getRemoteAddr();
```

这行代码可以取出访问系统的用户的 IP 地址。

```
if(remoteIP.equals(ip))
            {
                response.setCharacterEncoding("gb2312");
                PrintWriter out = response.getWriter();
                out.println("<b>你的 IP 地址被禁止访问。</b>");
            }
```

上面这段代码的功能是，如果用户的 IP 地址和配置文件中限制的 IP 相同，就拒绝用户的访问，并在页面上打印错误信息。

```
chain.doFilter(request,response);
```

上面这行代码的意思是，如果 IP 合法，就允许访问，用户可以进入应用系统进行其他的操作。

过滤器和 Servlet 很类似，都需要在 web.xml 文件中进行配置，但是过滤器的配置明显要比 Servlet 的复杂很多，下面是这个 IP 限制过滤器的配置信息，这段配置内容需要添加在 web.xml 文件的<web-app></web-app>标签之间。

```
    <filter>
      <filter-name>IPFilter</filter-name>
      <filter-class>filters.IPFilter</filter-class>
      <init-param>
        <param-name>ip</param-name>
        <param-value>127.0.0.1</param-value>
      </init-param>
    </filter>
    <filter-mapping>
        <filter-name>IPFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

在上面这个配置文件中，<filter>节点是对过滤器自身的属性进行配置，<filter-mapping>节点是对过滤器访问路径的配置，其中<filter-name>是过滤器的名称，这个属性可以自己命名，但是要和<filter-mapping>中的<filter-name>保持一致，<filter-class>指的是过滤器所对应的 Java 类文件，这里一定注意要把包名和类名写全。<init-param>定义了这个过滤器在初始化的时候加载的参数，<param-name>是初始化参数的名称，<param-value>是初始化参数的值，在一个过滤器中可以没有参数，也可以有一个到多个参数。<url-pattern>指明了这个过滤器对那些路径的访问有效，在这里“/*”的意思对于所有请求，都必需经过这个过滤器的的处理。

同时，在一个应用系统中，可以没有过滤器，也可以定义一个或者多个过滤器。

总之，开发一个过滤器需要通过下面三个基本步骤。

（1）编写过滤器的功能代码，即实现功能的代码类。

（2）把编译成功的过滤器功能代码类文件拷贝到当前应用项目的 WEB-INF/classes 目录下。

（3）在当前应用项目的 web.xml 文件中对过滤器进行配置，即在 web.xml 中添加配置信息。

经过这样三个步骤这个过滤器就可以发挥作用。

上面这个 IP 访问限制的示例程序中，设置禁止访问的 IP 地址为 127.0.0.1，这个 IP 地址就是本机 IP，所以在上面这个过滤器起作用以后，再访问上面任何一个页面或者是 Servlet，都会得到如图 5.14 所示的界面。

图 5.14　IP 访问过滤器运行效果

注意：在这个 IP 访问限制的过滤器中，配置文件中<init-param>的值可以设置成任意要限制的 IP，此处仅仅是为了展示方便，才设置为本机默认的 IP。

5.4.3　转换字符编码 filter

在 Java 语言中，默认的编码方式是 ISO-8859-1，这种编码格式不支持中文的显示，我们可以用类似<%@ page contentType="text/html;charset=gb2312"%>这样的方式来规定页面字符编码格式，但是如果要显示的内容是表单提交、或者是经过 Servlet 处理，这时候字符内容本身的编码格式就是 ISO-8859-1，所以尽管页面指定的字符编码方案为 gb2312，在这种情况下中文内容仍然不能正常显示。在第四章中已经对中文处理的问题做了详细的介绍，所以在本章仅仅对其中使用过滤器解决中文乱码问题进行详细的分析。

这个转换中文字符编码过滤器的代码如下。

```
//--------文件名：SetCharacterEncodingFilter.java--------------------
package filters;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.UnavailableException;
public class SetCharacterEncodingFilter implements Filter {

   protected FilterConfig filterConfig;
   protected String encodingName;
   protected boolean enable;

   public SetCharacterEncodingFilter() {
      this.encodingName = "gb2312";
      this.enable = false;
```

```
  }
  public void init(FilterConfig filterConfig) throws ServletException {
    this.filterConfig = filterConfig;
    loadConfigParams();
  }
  private void loadConfigParams() {
    this.encodingName = this.filterConfig.getInitParameter("encoding");
    String strIgnoreFlag = this.filterConfig.getInitParameter("enable");
    if (strIgnoreFlag.equalsIgnoreCase("true")) {
      this.enable = true;
    } else {
      this.enable = false;
    }
  }

  public void doFilter(ServletRequest request, ServletResponse response,
                       FilterChain chain) throws IOException, ServletException {
    if(this.enable) {
      request.setCharacterEncoding(this.encodingName);
    }
    chain.doFilter(request, response);
  }

  public void destroy() {
  }

}
```

在上面这个过滤器中，init（）方法从配置文件中取出字符编码格式的参数，在 doFilter（）方法中使用 request 对象对所有的请求统一编码格式。

这个字符编码设置过滤器的配置信息如下。

```
<filter>
      <filter-name>SetCharacterEncodingFilter</filter-name>
      <filter-class>filters.SetCharacterEncodingFilter</filter-class>
      <init-param>
        <param-name>encoding</param-name>
        <param-value>gb2312</param-value>
      </init-param>
      <init-param>
        <param-name>enable</param-name>
        <param-value>true</param-value>
      </init-param>
  </filter>
  <filter-mapping>
      <filter-name>SetCharacterEncodingFilter</filter-name>
      <url-pattern>/*</url-pattern>
  </filter-mapping>
```

在上面这个过滤器中，初始化的时候提供了两个参数，参数 encoding 指明了编码格式为 gb2312，参数 enable 指明是否使用这个过滤器处理字符编码，在这里设置为 true，即启动这个字符编码过滤器，需要注意的是，这里的 enable 参数并不是所有过滤器必需的，只是在这个字符处理的程序中自己设定的

一个参数，没有通用的含义。<url-pattern>/*</url-pattern>指明了对所有的请求都使用字符编码过滤器进行转码处理。这样就避免了对每个请求都进行字符编码设置，从而大大减少了工作量。

经过这个字符编码格式转换过滤器的处理，所有来自客户的请求数据都被转换成 gb2312 的格式，这样就可以解决中文编码格式不同带来的乱码问题，这里需要注意，在 Tomcat 中，对 URL 和 GET 方法提交的表单是按照 ISO-8859-1 的格式进行编码的，过滤器对这种情况并不起作用，这时候就需要修改 Tomcat 的配置文件来解决，这在第四章中的中文乱码解决方案中已经提供详细的解决方法。

5.5 Servlet 应用示例

在上面的内容中已经提到，Servlet 是与 HTTP 协议紧密结合的，使用 Servlet 几乎可以处理 HTTP 协议各个方面的内容，在本节的几个示例程序中，将集中展示 Servlet 在 HTTP 方面的具体应用。

5.5.1 获取请求信息头部内容

当客户访问一个页面的时候，会提交一个 HTTP 请求给服务器的 Servlet 引擎，在这个请求中有 HTTP 的文件头信息，其中包含这个请求的详细属性信息，在下面这个示例 Servlet 中将取出 HTTP 头部内容，并在页面打印，这个 Servlet 的具体代码如下。

```
//--------文件名：RequestHeader.java--------------------
package servlets;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class RequestHeader extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Enumeration e = request.getHeaderNames();
        while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            String value = request.getHeader(name);
            out.println(name + " = " + value+"<br>");
        }
    }
}
```

在上面这个 Servlet 中，首先使用 request.getHeaderNames()取出所有的 HTTP 头信息的名称，然后循环使用 request.getHeader(name)取出对应的头信息的值，并且在页面打印。

这个 Servlet 的配置信息如下。

```
    <servlet>
        <servlet-name>RequestHeader</servlet-name>
        <servlet-class>servlets.RequestHeader</servlet-class>
```

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

这个 Servlet 的配置信息和上面其他的 Servlet 的配置基本一样，在配置工作完成以后就可以用类似 http://localhost:8080/chapt5/ RequestHeader 这样的地址进行访问，其中 http://localhost:8080/是本机计算机的访问路径，chapt5 是当前应用项目的名称，RequestHeader 是取 HTTP 头信息的 Servlet，上面这个示例 Servlet 的运行效果如图 5.15 所示。

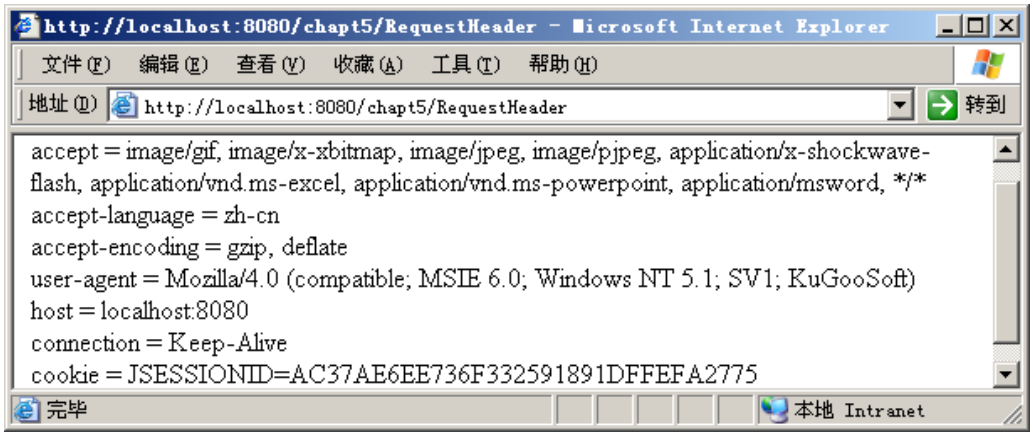

图 5.15　获取 HTTP 头部信息示例程序运行效果

如图 5.15 所示，等号左边的是 HTTP 头信息项的名称，等号右边是这项的值。而且具体 HTTP 头文件信息的内容不尽相同，在不同的浏览器中会有所不同。

5.5.2　获取请求信息

在上面这个 Servlet 示例中，我们取出所有的 HTTP 文件头信息，在 Servlet 中还可以很方便取出客户发出请求对象自身的信息。这些信息是和客户的请求密切相关的，例如客户提交请求所使用的协议，客户提交表单的方法是 POST 还是 GET 等，在下面这个示例程序中将介绍集中常见属性的取值方法。这个示例程序的具体代码如下。

```
//--------文件名：RequestInfo.java--------------------
package servlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class RequestInfo extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
```

```
            out.println("<head>");
            out.println("<title>请求信息示例</title>");
            out.println("</head>");
            out.println("<body><font size='2'");
            out.println("<b>请求信息示例</b><br>");
            out.println("Method: " + request.getMethod()+"<br>");
            out.println("Request URI: " + request.getRequestURI()+"<br>");
            out.println("Protocol: " + request.getProtocol()+"<br>");
            out.println("Remote Address: " + request.getRemoteAddr()+"<br>");
            out.println("</font></body>");
            out.println("</html>");
        }
        public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException
        {
            doGet(request, response);
        }
}
```

下面解释上面这个 Servlet 的关键代码。

```
response.setContentType("text/html");
response.setCharacterEncoding("gb2312");
```

上面这两行代码这时服务器响应的文件格式和字符编码格式。其中 gb2312 的编码格式可以用来支持中文字符的显示。

```
out.println("Method: " + request.getMethod()+"<br>");
```

上面这行代码取出表单提交的方法，可以是 POST 或者是 GET。

```
out.println("Request URI: " + request.getRequestURI()+"<br>");
```

上面这行代码取出这个客户请求的 URI，即相对的访问路径，在本例中的值为/chapt5/RequestInfo。

```
out.println("Protocol: " + request.getProtocol()+"<br>");
```

上面这行代码取出客户发出请求的时候使用的协议，在本例中的值为 HTTP/1.1。

```
out.println("Remote Address: " + request.getRemoteAddr()+"<br>");
```

上面这行代码取出客户的 IP 地址。

上面这个 Serlet 示例程序的配置信息如下。

```
    <servlet>
        <servlet-name>RequestInfo</servlet-name>
        <servlet-class>servlets.RequestInfo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>RequestInfo</servlet-name>
        <url-pattern>/RequestInfo</url-pattern>
    </servlet-mapping>
```

在配置工作完成以后就可以用类似 http://localhost:8080/chapt5/ RequestInfo 这样的地址进行访问，其中 http://localhost:8080/是本机计算机的访问路径，chapt5 是当前应用项目的名称，RequestInfo 是获取请求信息的 Servlet，上面这个示例 Servlet 的运行效果如图 5.16 所示。

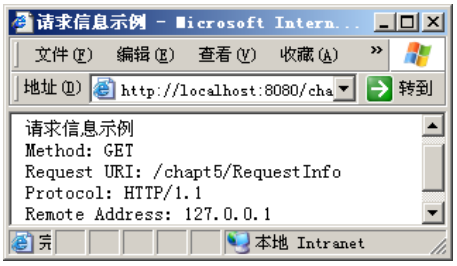

图 5.16　获取请求信息示例运行效果

在上面这个示例程序中，仅仅展示了有限的几个请求信息的获取方法，限于篇幅，其他请求信息的获取方法不在展示，它们的使用方法都是基本类似的，读者可以尝试自己进行思考、摸索。

5.5.3　获取参数信息

在 Servlet 中，同样可以很方便的取出用户请求中的参数信息，这种参数包括以 POST 方法或者是 GET 方法提交的表单，也包括直接使用超链接传递的参数，Servlet 都可以取出这些信息并且加以处理，在下面的例子中将具体展示 Servlet 获取各种参数的方法。

首先来看这样一个表单，这个表单就是在 5.3.1 中那个简单的用户信息表单，在这个例子中我们会分别使用 POST 方法和 GET 方法提交用户的请求信息。

这个表单的具体内容如下。

```
//--------文件名：paramForm.jsp--------------------
<%@ page language="java" import="java.util.*" contentType="text/html;charset=gb2312"%>
<html>
  <head>
    <title>Servlet 接收表单示例</title>
  </head>
  <body>
     <font size="2">
        <form action="RequestParam" method="post">
             姓名：<input type="text" name="name"/><br>
             省份：<input type="text" name="province"><br>
            <input type="submit" value="提交">
        </form>
     </font>
  </body>
</html>
```

接收这个表单请求的 Servlet 的具体代码如下。

```
//--------文件名：RequestParam.java--------------------
package servlets;

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

public class RequestParam extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
    throws IOException, ServletException
    {
            response.setContentType("text/html");
            response.setCharacterEncoding("gb2312");
            Enumeration e = request.getParameterNames();
            PrintWriter out = response.getWriter ();
            out.print("<font size='2'>");
            out.print("下面是用 GET 方法传递过来的参数:<br>");

            while (e.hasMoreElements()) {
                String name = (String)e.nextElement();
                String value = request.getParameter(name);
                out.println(name + " = " + value+"<br>");
            }
            out.print("</font>");

    }

    public void doPost(HttpServletRequest request, HttpServletResponse res)
    throws IOException, ServletException
    {
        res.setContentType("text/html");
        res.setCharacterEncoding("gb2312");
        Enumeration e = request.getParameterNames();
        PrintWriter out = res.getWriter ();
        out.print("<font size='2'>");
        out.print("下面是用 POST 方法传递过来的参数:<br>");
        while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            String value = request.getParameter(name);
            out.println(name + " = " + value+"<br>");
        }
        out.print("</font>");
    }
}
```

在这个 Servlet 中可以看到，我们分别处理 doGet 和 doPost 方法，而不是像前面的示例程序中只实现 doGet 方法，然后在 doPost 中简单调用，在这里因为要处理不同方法提交的参数，需要有不同的处理方法，所以在这里分别实现 doGet 和 doPost 方法。

这个 Servlet 的配置信息如下。

```
    <servlet>
        <servlet-name>RequestParam</servlet-name>
        <servlet-class>servlets.RequestParam</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>RequestParam</servlet-name>
        <url-pattern>/RequestParam</url-pattern>
    </servlet-mapping>
```

在配置工作完成以后就可以用类似 http://localhost:8080/chapt5/ paramForm.jsp 这样的地址进行访问，其中 http://localhost:8080/是本机计算机的访问路径，chapt5 是当前应用项目的名称，paramForm.jsp 是填

写表单的页面，这个 Servlet 的运行效果如图 5.17 所示。

图 5.17　Servlet 接收 POST 方法提交的参数效果

当把上面这个表单的提交方法改为 GET 的时候，这个 Servlet 的运行效果如图 5.18 所示。

图 5.18　Servlet 接收 GET 方法提交参数的效果

在上面这两个效果图中可以看出，浏览器地址栏中的内容是不同的，用 POST 方法提交的时候参数的内容不在地址栏中显示，而使用 GET 方法提交的时候，所有的参数都在地址栏中显示，在这里因为 Tomcat 对 URL 的字符编码格式为 ISO-8859-1，所以在地址栏中的中文字符不能正常显示，不过我们已经按照第四章中处理中文乱码的方式修改了 Tomcat 的配置，所以在页面上还是可以正常显示出中文参数的内容。

超链接传递参数的处理方法和处理 GET 方法提交表单的参数手段是一样的，因为单击超链接以后，参数的内容也是在地址栏中显示的，其道理是和 GET 方法提交表单中的参数是相同的，所以在这里就不在赘述。

5.5.4　Cookies 操作

Cookies 是指在 Web 应用中，为了辨别用户身份而存储在用户本地计算机上的数据。Servlet API 提供了 Cookie 操作类，封装了操作 Cookie 常用的方法，在下面的例子中，将展示 Servlet 操作 Cookie 的方法。这个 Servlet 的具体代码如下。

```
//--------文件名：Cookies.java--------------------
package servlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Cookies extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");
```

```
        PrintWriter out = response.getWriter();

        //设置一个 Cookie
        Cookie cookie = new Cookie("name", "Gates");
        response.addCookie(cookie);

        //打印 Cookie 的内容
        Cookie[] cookies = request.getCookies();
        out.print("<font size='2'>");
         out.print("下面是 Cookie 的内容:<br>");
        for (int i = 0; i < cookies.length; i++) {
            Cookie c = cookies[i];
            String name = c.getName();
            String value = c.getValue();
            out.println(name + " = " + value+"<br>");
        }
        out.print("</font>");
    }
}
```

下面介绍 Servlet 中操作 Cookie 的方法。

```
Cookie cookie = new Cookie("name", "Gates");
```

上面这行代码创建了一个 Cookie 对象，这个 Cookie 的名字为 name，存储的信息为 Gates。

```
response.addCookie(cookie);
```

上面这行代码把这个 Cookie 写入到用户本地的计算机中。

```
Cookie[] cookies = request.getCookies();
```

上面这行代码取出用户机器中的 Cookie，用于接下来的打印操作。

```
Cookie c = cookies[i];
String name = c.getName();
String value = c.getValue();
```

上面这段代码可以取出一个 Cookie 的名称和值。

这个示例程序的配置信息如下。

```
    <servlet>
        <servlet-name>Cookies</servlet-name>
        <servlet-class>servlets.Cookies</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Cookies</servlet-name>
        <url-pattern>/Cookies</url-pattern>
    </servlet-mapping>
```

在配置工作完成以后就可以用类似 http://localhost:8080/chapt5/ Cookies 这样的地址进行访问，其中 http://localhost:8080/是本机计算机的访问路径，chapt5 是当前应用项目的名称，Cookies 是操作 Cookie 的 Servlet，这个示例程序的运行效果如图 5.19 所示。

图 5.19　Cookie 操作示例程序运行效果

注意：在运行 Cookie 操作这个示例程序的时候，如果程序不能正常运行，需要检查浏览器是否禁用 Cookie。

5.5.5　Session 操作

在 JSP 中有内置的 Session 对象，可以用来保持服务器与用户之间的会话状态，在 Servlet 中间，同样可以对 Session 进行方便的操作，在现面的例子中，将详细介绍 Servlet 中处理 Session 的具体方法。这个 Servlet 的具体代码如下。

```
//--------文件名：RequestInfo.java--------------------
package Sessions;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Sessions extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        response.setCharacterEncoding("gb2312");
        PrintWriter out = response.getWriter();

        HttpSession session = request.getSession(true);
        // 打印当前 Session 的具体属性信息
        out.println("<font size='2'>");
        Date created = new Date(session.getCreationTime());
        Date accessed = new Date(session.getLastAccessedTime());
        out.println("Session 的 ID 为：  " + session.getId()+"<br>");
        out.println("Session 创建的时间为: " + created+"<br>");
        out.println("Session 上次访问时间为: " + accessed+"<br>");
        // 在这里可以设置一个 Session
        session.setAttribute("msg", "Hello");
        //打印 Session 的具体内容
        Enumeration e = session.getAttributeNames();
        out.println("Session 的内容如下：");
        while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            String value = session.getAttribute(name).toString();
```

```
                out.println(name + " = " + value+"<br>");
            }
            out.println("</font>");
        }
}
```

接下来详细介绍这个 Servlet 的关键代码。

```
HttpSession session = request.getSession(true);
```

上面这行代码从 request 对象中取出当前的 session 对象。

```
Date created = new Date(session.getCreationTime());
```

上面这行代码取出 session 的创建时间。

```
Date accessed = new Date(session.getLastAccessedTime());
```

上面这行代码取出 session 上次被访问的时间。

```
out.println("Session 的 ID 为：  " + session.getId()+"<br>");
```

上面这行代码取出 session 的 id，并且在页面上打印。

```
session.setAttribute("msg", "Hello");
```

上面这行代码在 session 中设置了一个名为 msg 的属性变量，这个变量的值为 Hello。

```
Enumeration e = session.getAttributeNames();
```

上面这句话从 session 中取出所有的属性，并放在一个集合中间。

```
while (e.hasMoreElements()) {
        String name = (String)e.nextElement();
        String value = session.getAttribute(name).toString();
        out.println(name + " = " + value+"<br>");
 }
```

上面这段程序中，循环取出 session 中的内容，并在页面打印。

上面这个 Servlet 的配置信息如下。

```
    <servlet>
        <servlet-name>Sessions</servlet-name>
        <servlet-class>servlets.Sessions</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Sessions</servlet-name>
        <url-pattern>/Sessions</url-pattern>
    </servlet-mapping>
```

在配置工作完成以后就可以用类似 http://localhost:8080/chapt5/ Sessions 这样的地址进行访问，其中 http://localhost:8080/是本机计算机的访问路径，chapt5 是当前应用项目的名称，Sessions 是操作 session 的 Servlet，上面这个示例 Servlet 的运行效果如图 5.20 所示。

图 5.20　Session 操作示例运行效果

在上面这个示例程序中，只要不关闭浏览器，在整个用户与服务器的交互期间，这个 session 对象

是始终存在的，这一点可以从 session 的 id 和创建时间看出。一旦关闭浏览器，这个 session 对象就会被销毁，再次访问的时候服务器重新生成一个 session 用来维护与用户之间的状态。

5.6 小结

在本章的内容中，详细讲解了 Servlet 的工作原理，并且通过实际的示例程序详细介绍了 Servlet 的调用方法，对 Servlet 常见的文件操作也做了比较详细的介绍，Servlet 是和 HTTP 协议密切联系的，所以在本章最后的部分对 Servlet 的 HTTP 操作方法做了细致的讲解。

通过本章内容的讲解，读者已经可以对 Servlet 有一个总体上的把握，Servlet 在本质上就是 Java 类，在了解这 Servlet 的基本原理和基本使用方法以后，如果要想在 Servlet 领域有更大的提高，还是需要回头巩固 Java 的基础，这才是学习 Servlet 的最根本的途径。