第5章 请求和响应

在客户端浏览器发送一个请求后，服务器执行一系列操作，并作出一个响应，发送给客户端，即完成一次 Web 访问过程。客户端浏览器的请求被封装成一个 HttpServletRequest 对象，简称 request 对象。服务器端对客户端浏览器作出的响应被封装成一个 HttpServletResponse 对象，简称 response 对象。request 对象和 response 对象起到了服务器端与客户端之间的信息传递作用。request 对象用于接收客户端浏览器提交的数据，而 response 对象的功能则是将服务器端的数据发送到客户端浏览器。

通过本章的学习，您可以：

(1) 掌握 HttpServletResponse 接口及其常用方法的使用；

(2) 掌握 HttpServletRequest 接口及其常用方法的使用；

(3) 掌握请求和响应过程中出现中文乱码问题的解决方法；

(4) 掌握请求转发和请求重定向的使用。

浏览器并不是直接与 Servlet 通信的，而是通过 Web 服务器和 Servlet 容器与 Servlet 通信。针对 Servlet 的每次请求，Servlet 容器在调用 service()方法之前，都会创建 request 对象和 response 对象；然后调用相应的 Servlet 程序。在 Servlet 程序运行时，它首先会从 request 对象中读取数据信息，再通过 service()方法处理请求消息，并将处理后的响应数据写入到 response 对象中；最后，Web 服务器会从 response 对象中读取到响应数据，并发送给浏览器。浏览器访问 Servlet 的过程示意如图 5-1 所示。

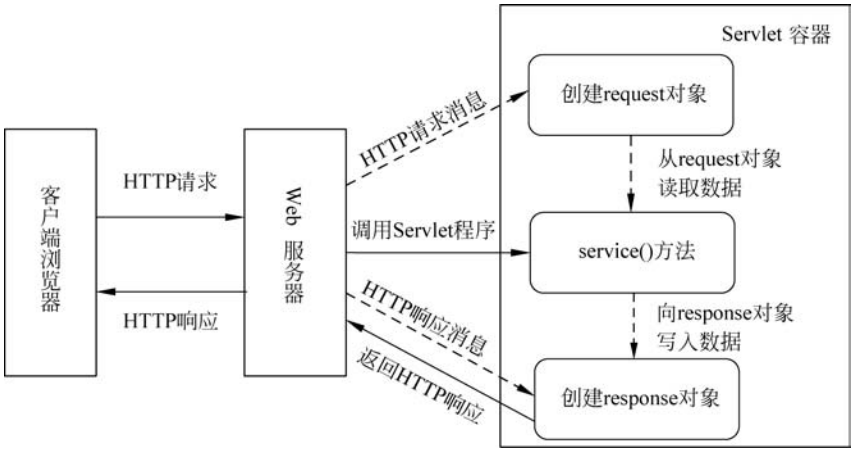

图 5-1 浏览器访问 Servlet 的过程示意

注意：在 Web 服务器运行阶段，每个 Servlet 都会创建一个实例对象，针对每次的 HTTP 请求，Web 服务器都会调用所请求 Servlet 实例的 service(HttpServletRequest request, HttpServletResponse response)方法，并重新创建一个 request 对象和一个 response 对象。

5.1 HttpServletResponse 接口及其应用


视频讲解

5.1.1 HttpServletResponse 接口

在 Servlet API 中定义了一个 HttpServletResponse 接口，它位于 javax.servlet.http 包中，继承自 javax.servlet.ServletResponse 接口，主要用于封装 HTTP 响应消息。由于 HTTP 响应消息分为响应状态行、响应消息头和响应消息体 3 部分，因此在 HttpServletResponse 接口中定义了向客户端发送响应状态码、响应消息头和响应消息体的方法。

1. 发送与状态码相关的方法

当 Servlet 向客户端回送响应消息时，需要在响应消息中设置响应状态行。响应状态行由 HTTP 版本、状态码和状态描述信息 3 部分构成。由于状态描述信息直接与状态码相关，而 HTTP 版本由服务器确定，因此，只要设置状态码即可。HttpServletResponse 接口定义了 setStatus()和 sendError()两种发送状态码的方法。

1) setStatus()方法

该方法用于设置 HTTP 响应消息的状态码，并生成响应状态行。需要注意的是，在正常情况下，Web 服务器会默认产生一个状态码为 200 的状态行。完整的语法格式如下：

```
void setStatus(int sc)
```

整型参数 sc 为状态码。

【例 5-1】 发送状态码 302。

创建项目 ch5_demo，创建包 com.yzpc.servlet，在包中创建 Servlet，命名为 TestSetvlet，设置虚拟路径为“/TestSetvlet”，重写 Servlet 的 doGet()方法，在 doGet()方法中调用 setStatus()方法。TestSetvlet.java 文件代码如文件 5-1 所示。

文件 5-1 TestSetvlet.java 文件代码

```
1   package com.yzpc.servlet;
2   import java.io.IOException;
3   import javax.servlet.ServletException;
4   import javax.servlet.annotation.WebServlet;
5   import javax.servlet.http.HttpServlet;
6   import javax.servlet.http.HttpServletRequest;
7   import javax.servlet.http.HttpServletResponse;
8   @WebServlet("/TestServlet")
9   public class TestServlet extends HttpServlet {
10     private static final long serialVersionUID = 1L;
11     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
       ServletException, IOException {
12        response.setStatus(302);                    //发送状态码 302
13     }
14  }
```

发布项目，启动服务器。打开 Chrome 浏览器，打开开发者工具。在浏览器地址栏中输入 http://localhost:8080/ch5_demo/TestServlet。观察开发者工具窗口，可见响应状态行信息为 HTTP/1.1 302。setStatus()方法运行结果如图 5-2 所示。

2) sendError()方法

该方法用于发送表示错误信息的状态码。response 对象提供了两个重载的发送错误信息

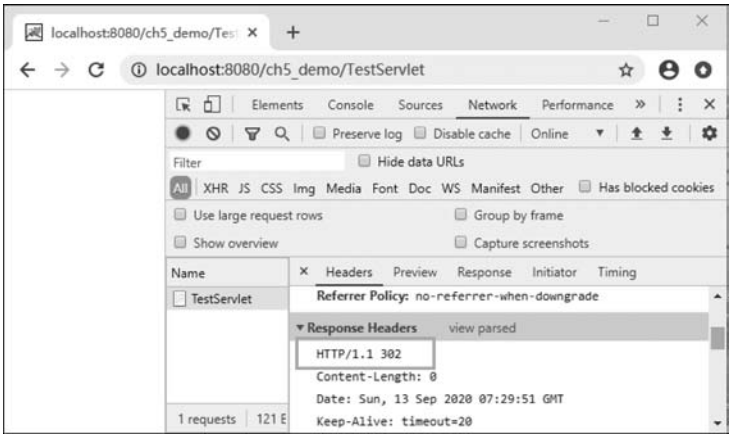

图 5-2　setStatus()方法运行结果

的状态码的方法,代码如下:

```
1    public void sendError(int code) throws java.io.IOException
2    public void sendError(int code,String message)throws java.io.IOException
```

在上面重载的两种方法中,第一种方法只发送错误信息的状态码,而第二种方法除了发送状态码以外,还可以增加一条用于提示说明的文本信息,该文本信息将出现在发送给客户端的正文内容中。

用如下代码替换文件 5-1 中第 12 行代码。

```
response.sendError(404);
```

运行后显示 404 错误提示页面,对应的响应状态行为 HTTP/1.1 404。sendError()方法运行结果如图 5-3 所示。

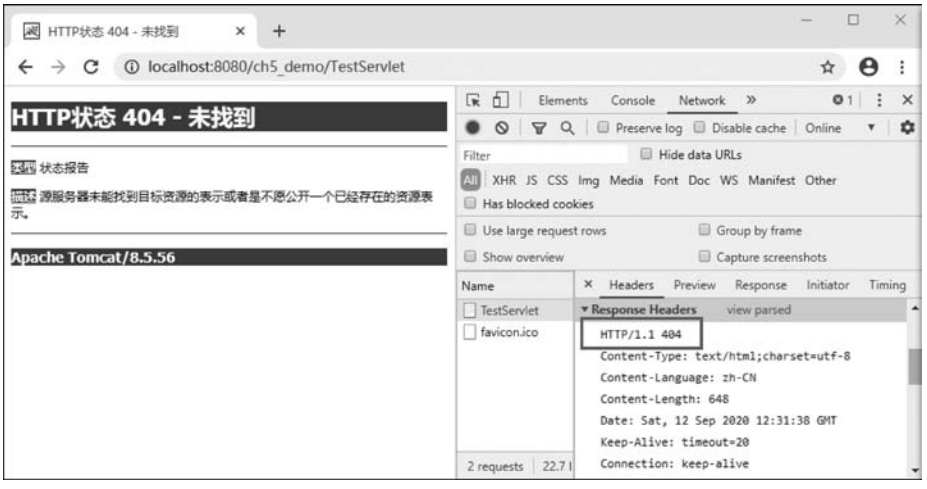

图 5-3　sendError()方法运行结果

sendError()方法适用于报错且存在对应的报错页面配置作为输出显示的情况,如 404、500 等错误页面的发送;而 setStatus()方法适用于正常响应的情况,仅仅可以改变响应状

态码。

3）响应状态码对应的常量

HttpServletResponse 接口中定义了很多状态码的常量，当需要向客户端发送响应状态码时，可以使用这些常量，避免了直接写数字。常见响应状态码的常量如表 5-1 所示。

表 5-1　常见响应状态码的常量

静 态 常 量	状态码	表 示 意 义
static int SC_CONTINUE	100	指示客户端可以继续
static int SC_OK	200	指示请求正常
static int SC_MULTIPLE_CHOICES	300	指示所请求的资源对应于一组表示中的任何一个，每个表示具有其自己的特定位置
static int SC_FOUND	302	指示资源暂时驻留在不同的 URI 下
static int SC_NOT_MODIFIED	304	指示条件 GET 操作发现资源可用且未被修改
static int SC_NOT_FOUND	404	指示所请求资源不可用
static int SC_INTERNAL_SERVER_ERROR	500	指示 HTTP 服务器内部的错误，从而阻止其履行请求

2. 发送与响应消息头相关的方法

Servlet 向客户端发送的响应消息中包含响应头字段，由于 HTTP 的响应头字段有很多种，因此，HttpServletResponse 接口定义了一系列设置 HTTP 响应头字段的方法，既有通用的设置响应头字段的方法，也有设置特定响应头字段的方法。下面介绍几种常用的方法。

1）通用的设置响应头字段的方法

```
1   void setHeader(String name, String value);
2   void addHeader(String name, String value);
3   void setIntHeader(String name, int value);
4   void addIntHeader(String name, int value);
```

说明：

（1）setHeader()方法和 addHeader()方法用于设定 HTTP 的响应头字段，参数 name 用于指定响应头字段的名称，String 类型；参数 value 用于指定响应头字段的值，String 类型。

（2）setIntHeader()方法和 addIntHeader()方法与 setHeader()方法和 addHeader()方法的功能相似。不同的是，这两种方法只适用于响应头字段的值为 int 类型时的响应消息头的设置。

（3）addHeader()方法和 addIntHeader()方法允许增加同名的响应头字段的值；而 setHeader()方法和 setIntHeader()方法则会覆盖同名的响应头字段的值。

设置 refresh 头字段可以实现页面的自动刷新，也可以设置定时跳转网页。

【例 5-2】 refresh 头字段的应用。

（1）在项目 ch5_demo 的 com.yzpc.servlet 包中创建 RefreshServlet.java 类，设置虚拟路径“/RefreshServlet”，重写 doGet()方法。在 doGet()方法中，为了便于观察刷新功能，设计一个计数器，刷新一次，计数器值便增加 1。RefreshServlet.java 类的 doGet()方法如文件 5-2 所示。

文件 5-2　RefreshServlet.java 类的 doGet()方法

```
1   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
2       PrintWriter out = response.getWriter();
```

```
3       //利用 ServletContext 的域属性存储计数器 count
4       ServletContext context = this.getServletContext();        //获取 ServletContext 对象
5       Integer count = (Integer)context.getAttribute("count"); //读取域属性 count 的值
6       //计数,如果 count 为 null,设 count 初始值为 1,否则 count 增加 1
7       if( count == null) {count = 1;}else {count++;}
8       context.setAttribute("count", count);                     //存储当前计数次数到域属性中
9       out.print(count + "times. < br/>");                       //输出刷新次数
10      out.print("vocational skills competition");
11      //设置 refresh 头字段的值为 3,表示 3 秒后刷新当前页面
12      response.setHeader("refresh","3");
13  }
```

在文件 5-2 中,利用 ServletContext 的域属性存储计数器变量 count 的值,第 5 行读取域属性值,第一次读的时候,域属性 count 是不存在的,返回值为 null。第 7 行判断 count 变量的值是否为 null,若为 null,则说明是第一次访问页面,就设 count 变量初值为 1,否则将 count 变量增加 1。第 8 行将 count 变量的新值存储到域属性 count 中。第 12 行调用 setHeader()方法,设置 refresh 头字段的值为 3 秒,表示每隔 3 秒刷新当前页面一次。

注意:count 变量一定要使用包装类型,如 Integer,不能是 int。

每次刷新页面,都会发出一次新的 HTTP 请求,由于是当前页面刷新,因此会再次执行 doGet()方法。ServletContext 域属性的生命期是整个 Web 运行期间,因此在以后执行 doGet()方法时,ServletContext 域中已存储有 count 属性的有效值了。如此,通过 ServletContext 域属性实现了页面刷新计数的功能。

(2) 在 WebContent 根目录创建名为 refresh.html 的页面文件。其代码如下:

```
1   <!DOCTYPE html>
2   < html >
3       < head >
4           < meta charset = "GB18030">
5           < title >刷新并跳转</title >
6       </head >
7       < body >
8           < h2 align = "center">职业技能大赛 </h2 >
9       < center >第一届职业技能大赛 2020 年 12 月 10 日在广东省广州市开幕。</center >
10      </body >
11  </html >
```

(3) 启动服务器,在地址栏中输入地址 http://localhost:8080/ch5_demo/RefreshServlet 运行 RefreshServlet。页面每隔 3 秒就会刷新,页面上会显示计数刷新次数,且显示文本为文件 5-2 中发送的响应消息体。统计页面的刷新次数页面如图 5-4 所示。

图 5-4　统计页面的刷新次数页面

(4) 修改程序代码,实现定时跳转。修改文件 5-2 的代码。修改后的代码如下:

```
{protected void doGet(HttpServletRequest request, HttpServletResponse response)
  PrintWriter out = response.getWriter();
  out.print("vocational skills competition");
  //设置 refresh 头字段的值为"3,URI = refresh.html",表示 3 秒后刷新当前页面,并跳转到
  refresh.html 页面
  response.setHeader("refresh","3, URI = refresh.html");
}
```

(5) 重启服务器,在地址栏中输入地址 http://localhost: 8080/ch5_demo/RefreshServlet。首先显示 RefreshServlet 页面,3 秒后跳转到 refresh.html 页面。定时跳转页面如图 5-5 所示。

(a) RefreshServlet页面

(b) refresh.html页面

图 5-5 定时跳转页面

注意:当页面跳转时,浏览器地址栏中显示的是 refresh.html,而不是输入的 URL。

2) setContentType()方法

该方法用于设置发送到客户端的响应的内容类型,对应 HTTP 的 contentType 头字段。语法格式如下:

```
void setContentType(String type)
```

字符串类型参数 type 指定内容类型,若发送到客户端的内容是.jpeg 格式的图像数据,则使用"response.setContentType("image/jpeg");"语句;若发送的是文本数据,字符编码规范为 GB18030,则使用"response.setContentType("text/html; charset=GB18030");"语句。

3) setCharacterEncoding()方法

该方法用于设置输出内容使用的字符编码,对应 HTTP 的 contentType 头字段的字符集编码部分。如果没有设置 contentType 头字段,那么该方法设置的编码字符集不会出现在 HTTP 响应头中。语法格式如下:

```
void setCharacterEncoding(String charset)
```

String 类型参数 charset 指定编码字符集,如 GB18030、GB2312、UTF-8 等。

setCharacterEncoding( ) 方法和 setContentType( ) 方法都可用于设置字符编码,setCharacterEncoding()方法的优先权较高,它的设置结果将覆盖 setContentType()所设置的字符编码集。这两种设置字符编码的方法可以有效解决出现中文乱码的问题。

注意:这两种方法都必须在 getWriter()方法之前调用,输出的内容才会按照指定的内容类型设置。

3. 发送与响应消息体相关的方法

由于在 HTTP 响应消息中,大量的数据都是通过响应消息体传递的,因此,HttpServletResponse 接口的 ServletResponse 父接口中遵循以 I/O 流传递大量数据的设计理念。在发送响应消息体时,定义了两种与输出流相关的方法。

(1) getOutputStream() 方法。该方法所获取的字节输出流对象为 ServletOutputStream 类型。由于 ServletOutputStream 是 OutputStream 的子类,它可以直接输出字节数组中的二进制数据。因此,要想输出二进制格式的响应正文,就需要使用 getOutputStream() 方法。

(2) getWriter( ) 方法。该方法所获取的字符输出流对象为 PrintWriter 类型。由于 PrintWriter 类型的对象可以直接输出字符文本内容,因此,要想输出内容全部为字符文本的网页文档,就需要使用 getWriter()方法。

【例 5-3】 发送响应消息体。

在项目 ch5_demo 中创建 Servlet 并命名为 MsgServlet,设置虚拟路径为"/MsgServlet"。

① 使用 PrintWriter 字符流发送响应消息体。重写 MsgServlet 的 doGet( ) 方法。MsgServlet.java 的代码如文件 5-3 所示。

文件 5-3 MsgServlet.java 的代码

```
1  package com.yzpc.servlet;
2  import java.io.IOException;
3  import java.io.PrintWriter;                                    //导入 PrintWriter 类对应的包
4  import javax.servlet.ServletException;
5  import javax.servlet.annotation.WebServlet;
6  import javax.servlet.http.HttpServlet;
7  import javax.servlet.http.HttpServletRequest;
8  import javax.servlet.http.HttpServletResponse;
9  @WebServlet("/MsgServlet")                                     //配置虚拟路径
10 public class MsgServlet extends HttpServlet {
11    private static final long serialVersionUID = 1L;
12     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
       ServletException, IOException {
13        PrintWriter out = response.getWriter();                 //获取 PrintWriter 对象
14        out.print("Where there is a will, there is a way.");    //输出
15     }
16 }
```

在文件 5-3 中使用 response 对象的 getWriter()方法获得 PrintWriter 对象 out,需要导入对应的包 java.io.PrintWriter。

启动 Tomcat 服务器,在浏览器地址栏中输入 http://localhost:8080/ch5_demo/MsgServlet,在页面上显示"Where there is a will, there is a way."。发送消息体运行结果如图 5-6 所示。

图 5-6　发送消息体运行结果

② 使用 OutputStream 字节流发送响应消息体。重写文件 5-3 中的 doGet()方法，代码如下：

```
1   protected void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
2       String message = "Where there is a will, there is a way.";
3       OutputStream   out = response.getOutputStream();          //获取 OutputStream 对象
4       out.write(message.getBytes());
5   }
```

上述代码中第 3 行使用 response 对象的 getOutputStream()获取 OutputStream 对象。注意，需要导入包 java.io.OutputStream；第 4 行调用 write()方法输出响应消息体；由于参数 message 是字符串类型，第 4 行使用 getBytes()方法将字符串转换成字节数组。

重启 Tomcat 服务器后，刷新浏览器，输出结果与 PrintWriter 流输出响应消息体一样，如图 5-6 所示。

③ 同时使用 PrintWriter 字符流和 OutputStream 字节流发送响应消息体。将文件 5-3 的 doGet()方法修改为如下代码：

```
1   protected void doGet(HttpServletRequest request, HttpServletResponse response)
2   throws ServletException, IOException {
3       String message = "Where there is a will, there is a way.";
4       OutputStream   out = response.getOutputStream();          //获取 OutputStream 对象
5       out.write(message.getBytes());
6       PrintWriter out1 = response.getWriter();                  //获取 PrintWriter 对象
7       out1.print("Where there is a will, there is a way.");     //输出
8   }
```

上述代码中第 4 行调用了 response.getOutputStream()方法，第 6 行调用了 response.getWriter()方法。重启服务器，刷新浏览器后，在控制台上显示 IllegalStateException 异常。IllegalStateException 异常如图 5-7 所示。

图 5-7　IllegalStateException 异常

图 5-7 中发生异常的原因，是因为在 Servlet 调用 response.getWriter()方法之前已经调用了 response.getOutputStream()方法。虽然这两种方法都可以发送响应消息体，但它们之

间互相排斥,不可同时使用;如果同时使用,则会发生 IllegalStateException 异常。

5.1.2 HttpServletResponse 应用


视频讲解

1. 请求重定向

请求重定向是指 Web 服务器在接收到客户端的请求后,可能由于某些条件的限制,不能访问当前请求 URL 所指向的 Web 资源,而是指定了一个新的资源路径,须由客户端重新发送请求。在某些情况下,针对客户端的请求,一个 Servlet 类可能无法完成全部工作,可以使用请求重定向来继续完成这一工作。

为了实现请求重定向,HttpServletResponse 接口定义了一个 sendRedirect()方法,该方法用于生成 302 响应码和 Location 响应头,从而通知客户端重新访问 Location 响应头中指定的 URL。sendRedirect() 方法的完整语法如下:

```
public void sendRedirect(String location) throws IOException
```

在上述方法代码中,参数 location 可以使用相对 URL,Web 服务器会自动将相对 URL 翻译成绝对 URL,再生成 Location 头字段。sendRedirect()方法的工作原理如图 5-8 所示。

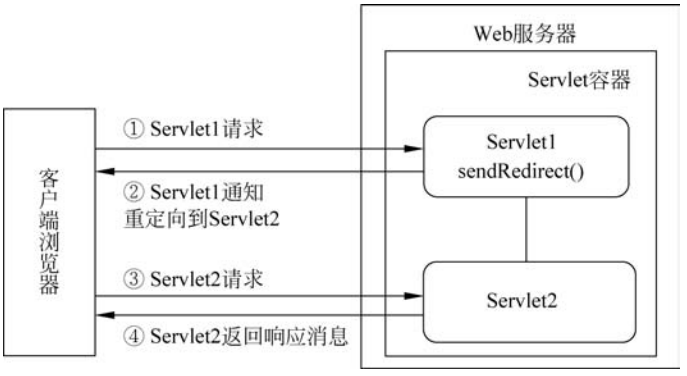

图 5-8 sendRedirect()方法的工作原理

在图 5-8 中,当客户端访问 Servlet1 时,由于在 Servlet1 中调用了 sendRedirect()方法将请求重定向到 Servlet2,因此,浏览器收到 Servlet1 的响应消息后,立刻向 Servlet2 发送请求,Servlet2 对请求处理完毕后,再将响应消息回送给客户端浏览器并显示。在这个过程中客户端与服务器端之间发生了两次请求,两次响应。

【例 5-4】 用户登录。

用户在登录页面输入用户名和密码,当用户名和密码都正确时,显示欢迎页面,否则返回到登录页面。

分析:本例中需要创建一个登录页面 login.html,一个欢迎页面 welcome.html,一个判断用户名和密码是否正确的 LoginServlet.java。步骤如下所述。

(1) 在项目 ch5_demo 的 WebContent 文件夹下,创建 login.html 文件。login.html 代码如文件 5-4 所示。

文件 5-4 login.html 代码

```
1   <!DOCTYPE html>
2   <html>
3       <head>
```

```
4            <meta charset = "GB18030">
5            <title>登录页面</title>
6        </head>
7        <body>
8        <p>登录</p>
9        <form action = "login" method = "POST">
10           <label>姓名:</label><input type = "text" name = "userName" /><br/>
11           <label>密码:</label><input type = "password" name = "password"/></br>
12           <input type = "submit" value = "提交"/>
13           <input type = "reset" value = "重置"/>
14       </form>
15       </body>
16   </html>
```

(2) 在 WebContent 文件夹下,创建 welcome.html 文件。<body>标签代码如下:

```
1    <body>
2    <h2 align = "center">全国职业院校技能大赛</h2>
3    <p>全国职业院校技能大赛(简称大赛)是教育部发起并牵头,联合国务院有关部门以及有关行业、
     人民团体、学术团体和地方共同举办的一项公益性、全国性职业院校学生综合技能竞赛活动。每年
     举办一届。</p>
4    </body>
```

(3) 在 com.yzpc.servlet 包中创建 LoginServlet.java 文件,设其虚拟路径为"/login",重写 doGet()方法。其代码如下:

```
1    public class LoginServlet extends HttpServlet {
2      private static final long serialVersionUID = 1L;
3      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
       ServletException, IOException {
4      //用 HttpServletRequest 对象的 getParameter() 方法获取用户名和密码
5      String username = request.getParameter("userName");
6      String password = request.getParameter("password");
7      //假设用户名和密码分别为 admin 和 123456
8      if ("admin".equals(username) && ("123456").equals(password)) {
9          //如果用户名和密码正确,重定向到 welcome.html
10         response.sendRedirect("welcome.html");
11     } else {
12         //如果用户名和密码错误,重定向到 login.html
13         response.sendRedirect("login.html");
14     }
15   }}
```

sendRedirect()方法的参数可以使用相对路径,也可以使用绝对路径。

使用相对路径重定向到当前目录下的资源 welcome.html。其代码如下:

```
response.sendRedirect("welcome.html")
```

如上述代码的第 10 行和第 13 行都使用了相对路径。

如果使用绝对路径,第 10 行代码可以改写为如下代码。

```
response.sendRedirect(this.getServletContext().getContextPath() + "/welcome.html");
```

其中 this.getServletContext().getContextPath()表示取得当前项目的上下文路径,即"/ch5_demo"。本例中当前目录就是根目录,因此,使用相对路径和绝对路径两种用法其效果是一样的。

(4) 发布项目并运行,在浏览器地址栏中输入访问路径 http://localhost:8080/ch5_demo/login.html。用户登录运行结果如图 5-9 所示。

图 5-9 用户登录运行结果

当用户名和密码正确时,会重定向到 welcome.html 页面,否则重定向到 login.html 页面。地址栏中会显示当前实际请求资源的地址。

注意:在 Eclipse 的 Web 项目中,WebContent 文件夹的文件在发布到服务器时,直接发布到根目录文件夹下。打开 Tomcat 安装目录下的 webapps 文件夹,找到 ch5_demo 项目,可以看到有 welcome.html 和 login.html 文件。ch5_demo 项目发布目录如图 5-10 所示。

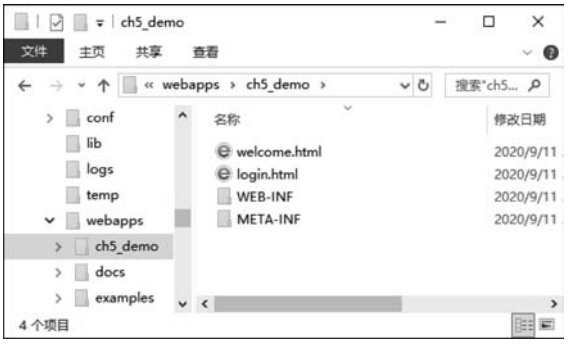

图 5-10 ch5_demo 项目发布目录

2. 输出中文乱码

计算机中的数据都是以二进制形式存储的,因此,当传输文本数据时,会发生字符和字节之间的转换。字符与字节之间的转换是通过查码表完成的,将字符转换成字节的过程称为编码,将字节转换成字符的过程称为解码。如果编码和解码使用的码表不一致,就会出现乱码问题。

1) 输出中文乱码的显示

在项目 ch5_demo 的 com.yzpc.servlet 包中创建 ChineseServlet 类,在 doGet()方法中添加如下代码。

```
response.getWriter.print("生态文明");
```

编译运行后，浏览器的页面显示乱码——四个“?”，而不是显示“生态文明”四个字。输出中文乱码问题界面如图 5-11 所示。

图 5-11　输出中文乱码问题界面

2）输出中文乱码原因及解决方案

HttpServletResponse 对象在输出时默认使用 ISO-8859-1 码表，不支持中文，而客户端浏览器默认使用中文编码，因此出现中文乱码。输出中文乱码示意如图 5-12 所示。

图 5-12　输出中文乱码示意

出现中文乱码问题的主要原因是服务器端和客户端使用的编码方式不一致造成的。下面分别介绍使用 OutputStream 流和 PrintWriter 流对于输出中文乱码问题的解决方案。

（1）使用 OutputStream 流向客户端浏览器输出中文数据。

在服务器端，响应消息体的数据是以哪个码表编码的，就要控制客户端浏览器以相应的码表解码。比如：outputStream.write("中国".getBytes("GB18030"))；使用 OutputStream 流向客户端浏览器输出中文，以 GB18030 码表编码输出，此时就要控制客户端浏览器以 GB18030 码表解码，否则显示的时候就会出现中文乱码。在服务器端控制客户端浏览器的编码字符集，可以通过设置 HTTP 响应头字段 content-Type 来实现。其代码如下：

```
response.setHeader("Content-Type", "text/html;charset= GB18030");
```

通过设置 Content-Type 响应头控制浏览器以 GB18030 的编码显示数据。

使用 OutputStream 流向客户端浏览器输出中文数据的步骤如下所述。

第 1 步，用 response.setHeader("Content-Type", "text/html; charset= GB18030")方法指定客户端的编码字符集，如不指定，将使用当前浏览器默认的字符集，可能会出现乱码。

第 2 步，使用 response.getOutputStream()方法获取输出流对象。

第 3 步，使用 getBytes()方法将字符串转换成字节数组。

第 4 步，调用 outputStream.write()方法发送响应消息体。

注意：第 1 步指定的字符集与第 3 步中编码使用的字符集要一致。

【例 5-5】 使用 OutputStream 字节流输出中文。

① 重写 ChineseServlet.java 类的 doGet()方法，定义一个中文字符串，调用 OutputStream 流输出。其代码如下：

```
1   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
2       String data = "生态文明";
```

```
3       //1.通过设置响应头控制浏览器以 GB18030 的编码显示数据,如果不加这语句,那么浏览器显
        示的将是乱码
4       response.setHeader("content-Type", "text/html;charset=GB18030");
5       //2.获取 OutputStream 输出流
6       OutputStream out = response.getOutputStream();
7       //3.将字符转换成字节数组,指定以 GB18030 编码进行转换与第 1 步中指定的编码一致
8       byte[] dataByteArr = data.getBytes("GB18030");
9       //4.使用 OutputStream 流向客户端输出字节数组
10      out.write(dataByteArr);
11  }
```

② 启动服务器。运行结果能正确显示中文数据,如图 5-13 所示。

图 5-13　运行结果

(2) 使用 PrintWriter 流向客户端浏览器输出中文数据。

PrintWriter 流是字符流,使用 PrintWriter 流向客户端浏览器输出中文数据,其步骤如下。

第 1 步,使用 response.setCharacterEncoding("GB18030")设置字符以什么样的编码输出到浏览器,若不指定,默认使用 ISO-8859-1 编码,该编码不兼容中文。

第 2 步,使用 response.setHeader("Content-Type", "text/html; charset=GB18030")通知浏览器使用的解码字符集。

第 3 步,使用 response.getWriter()方法; 获取 PrintWriter 输出流。

第 4 步,调用 PrintWriter 对象的 write()方法或 print()方法发送响应消息体。

注意:

(1) 第 1 步与第 2 步中指定的字符集需一致;

(2) 第 3 步一定要在前两步执行后再执行,否则不能解决乱码问题;

(3) 通过方法 response.setContent-Type("text/html; charset=GB18030")可以同时实现第 1 步与第 2 步的功能。

【例 5-6】 使用 PrintWriter 字符流输出中文。

重写例 5-5 中 ChineseServlet 的 doGet()方法,用字符流输出中文字符。其代码如下:

```
1   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
2       //使用 PrintWriter 流输出中文
3       response.setCharacterEncoding("GB18030");
4       response.setHeader("Content-Type", "text/html;charset=GB18030");
5       String data = "网络空间";
6       PrintWriter out = response.getWriter();
7       out.print(data);
8   }
```

启动服务器,刷新浏览器,运行结果能正确显示中文数据,如图 5-14 所示。

图 5-14　PrintWriter 字符流输出中文

5.2　HttpServletRequest 接口及其应用


视频讲解

5.2.1　HttpServletRequest 接口

在 Servlet API 中定义了一个 HttpServletRequest 接口，它位于 javax.servlet.http 包中，继承自 javax.servlet.ServletRequest 接口，其主要作用是封装 HTTP 请求消息。由于 HTTP 请求消息分为请求行、请求消息头和请求消息体 3 部分，因此，在 HttpServletRequest 接口中定义了获取请求行、请求头和请求消息体的相关方法。

1. 获取请求行信息的相关方法

当访问 Servlet 时，所有 HTTP 请求消息将被封装到 HttpServletRequest 对象中，HTTP 请求消息的请求行中包含请求方式、请求资源名和请求路径等信息。为了获取这些信息，HttpServletRequest 接口定义了一系列方法，如表 5-2 所示。

表 5-2　获取请求行信息的方法

方法声明	功能描述
String getMethod()	用于获取 HTTP 请求消息中的请求方法
String getRequestURI()	用于获取客户端发出请求时的 URI
String getQueryString()	用于获取请求行中的参数部分，也就是资源路径后面问号(?)以后的所有内容
String getContextPath()	用于获取请求 URL 中属于 Web 应用程序的路径，这个路径以“/”开头，表示相对于整个 Web 站点的根目录，路径结尾不含“/”。如果请求 URL 属于 Web 站点的根目录，那么返回结果为空字符串("")
StringBuffer getRequestURL()	用于获取客户端发出请求时的完整的 URL，包括协议、服务器名、端口号、资源路径等信息，但不包括后面的查找参数部分。注意，该方法返回的结果是 StringBuffer 类型而不是 string 类型，这样更便于对结果进行修改
String getPathInfo()	方法返回请求 URL 中的额外路径信息。额外路径信息是请求 URL 中的位于 Servlet 的路径之后和查询参数之前的内容，它以“/”开头
String getRemoteAddr()	方法返回发出请求的客户端的 IP 地址
String getRemoteHost()	方法返回发出请求的客户端的完整主机名
String getRemotePort()	方法返回客户端所使用的网络端口号
String getLocalAddr()	方法返回 Web 服务器的 IP 地址
String getLocalName()	方法返回 Web 服务器的主机名

常用的是前 5 种方法，特别是 getRequestURL()方法、getRequestURI()方法、getContextPath()方法，它们常用于文件路径的设置。下面举例说明这 5 种方法的使用。

【例 5-7】 获取请求行信息。

(1) 在项目 ch5_demo 中，创建 ReqServlet 类，重写 doGet()方法。其代码如下：

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter pw = response.getWriter();
    //获得请求行的相关信息
    pw.println("请求的方法:" + request.getMethod() + "<br/>");      //获取请求的方法
    pw.println("请求 URI:" + request.getRequestURI() + "<br/>");    //获取 URI
    pw.println("请求行中的参数:" + request.getQueryString() + "<br/>");
                                                                    //获取请求行中的参数
    pw.println("Web 应用程序的路径:" + request.getContextPath() + "<br/>");
                                                                    //Web 应用程序的路径
    pw.println("完整的 URL:" +   request.getRequestURL() + "<br/>");//完整的 URL
}
```

(2) 测试运行结果。启动服务器,在浏览器地址栏中输入 http://localhost:8080/ch5_demo/ReqServlet。运行结果如图 5-15(a)所示。

图 5-15　获取请求行信息运行结果

在图 5-15(a)中,请求行参数为 null,这是因为输入的地址中没有带参数。重新输入如下地址 http://localhost:8080/ch5_demo/ReqServlet?a=123。运行结果如图 5-15(b)所示。

2. 获取请求消息头的相关方法

当浏览器发送 Servlet 请求时,需要通过请求消息头向服务器端传递附加信息,例如客户端可以接收的数据类型、压缩方式、语言等。为此,在 HttpServletRequest 接口中定义了一系列用于获取 HTTP 请求头字段的方法,如表 5-3 所示。

表 5-3　获取请求头字段的方法

方 法 声 明	功 能 描 述
String getHeader(String name)	根据请求头字段的名称获取对应的请求头字段的值,只返回满足条件的第 1 个值,如果请求消息中不包含指定的头字段,就返回 null
Enumeration getHeaders(String name)	根据请求头字段的名称获取对应的请求头字段的所有值,存储到一个 Enumeration 对象中
Enumeration getHeaderNames()	用于获取一个包含所有请求头字段名称的 Enumeration 对象
int getIntHeader(String name)	根据请求头字段的名称获取对应的请求头字段的值,并转换为 int 类型。如果指定头字段不存在,返回-1;如果获取到的头字段不能转换为 int,将发生 NumberFormatException 异常
long getDateHeader(String name)	根据请求头字段的名称获取对应的请求头字段的值,并将其按 GMT 时间格式转换为一个代表日期/时间的长整数,这个长整数是自 1970 年 1 月 1 日 0 时 0 分 0 秒算起的以毫秒为单位的时间值
String getContentType()	获取 Content-Type 头字段的值
String getCharacteEncoding()	用于返回请求消息的实体部分的字符集编码,通常从 Content-Type 头字段中进行提取

下面通过一个案例，介绍使用 getHeaderNames()方法和 getHeader()方法获取头字段及其值的实现。

【例 5-8】 通过 request 对象获取客户端所有请求头信息。

(1) 在项目 ch5_demo 中，创建 HeadServlet 类，重写 doGet()方法。其代码如下：

```
1   public void doGet(HttpServletRequest request, HttpServletResponse response)
2   throws ServletException, IOException {
3       //通过设置响应头控制浏览器以 UTF-8 的编码显示数据
4       response.setContentType( "text/html;charset=UTF-8");
5       PrintWriter out = response.getWriter();
6       Enumeration<String> headNames = request.getHeaderNames();  //获取所有的请求头
7       out.write("获取到的客户端所有的请求头信息如下:");
8       out.write("<hr/>");
9       while (headNames.hasMoreElements()) {
10        String headName = (String) headNames.nextElement();
11        //根据请求头的名字获取对应的请求头的值
12        String headValue = request.getHeader(headName);
13        out.write(headName + ":" + headValue);
14        out.write("<br/>");
15      }
16      out.write("<br/>");
17  }
```

(2) 测试运行结果。启动服务器，在浏览器的地址栏中输入 http://localhost:8080/ch5_demo/headServlet。运行结果如图 5-16 所示。

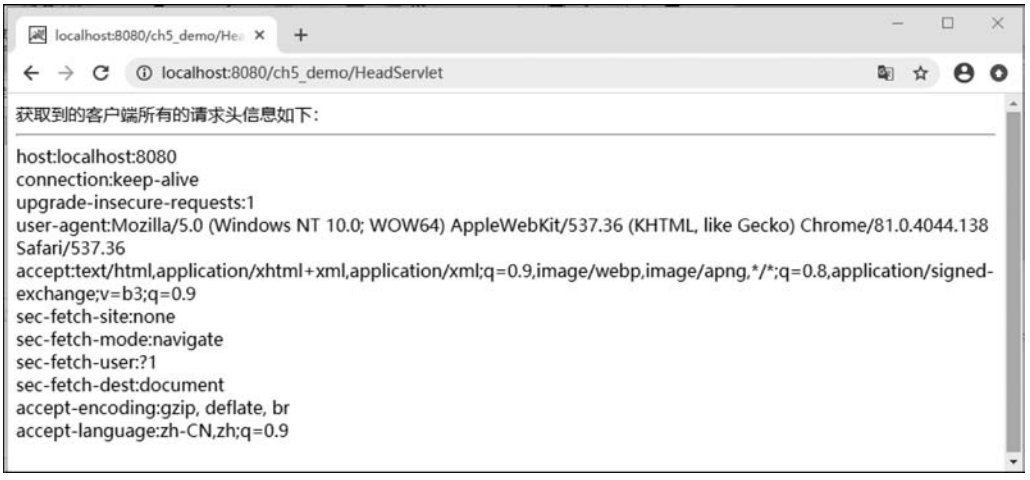

图 5-16 获取请求头信息的运行结果

3. 获取请求消息体的相关方法

当使用 POST 请求方式时，请求消息包含请求消息体，在请求消息体中封装了 POST 请求的参数。

在实际开发过程中，经常需要从 HttpServletRequest 中读取 HTTP 请求的消息体的内容，在 HttpServletRequest 接口的 ServletRequest 父接口中定义了一系列读取消息体的方法，如表 5-4 所示。

表 5-4 获取请求参数消息体的相关方法

方法声明	功能描述
String getParameter(String name)	该方法用于获取某个指定名称的参数值,如果请求消息中没有包含指定名称的参数,getParameter()方法返回 null;如果指定名称的参数存在但没有设置值,返回一个空串;如果请求消息中包含有多个该指定名称的参数,getParameter()方法返回第一个出现的参数值
String[] getParameterValues(String name)	HTTP 请求消息中可以有多个相同名称的参数(通常由一个包含有多个同名的字段元素的 FORM 表单生成),如果要获得 HTTP 请求消息中的同一个参数名所对应的所有参数值,那么就应该使用 getParameterValues()方法,该方法用于返回一个 String 类型的数组
Enumeration getParameterNames()	该方法用于返回一个包含请求消息中所有参数名的 Enumeration 对象,在此基础上,可以对请求消息中的所有参数进行遍历处理
Map getParameterMap()	个体 Parameter Map()方法用于将请求消息中的所有参数名和值装入一个 Map 对象中并返回
BufferedReader getReader()	获取字符输入流,只能操作字符数据
ServletInputStream getInputStream()	获取字节输入流,可以操作所有类型的数据

getParameter()方法、getInputStream()方法和 getReader()方法都是从 Servlet 中的 request 对象得到提交的数据,但是用途不同,因此要根据表单提交数据的编码方式选择不同的方法,这里不再赘述。在默认情况下使用 getParameter()方法获取表单字段及其值。

4. 通过 Request 对象传递数据

Request 对象不仅可以获取一系列数据,还可以通过属性传递数据。ServletRequest 接口中定义了一系列操作属性的方法。

(1) setAttribute()方法。该方法用于将一个对象与一个名称关联后存储到 ServletRequest 对象中,其完整语法代码如下:

```
public void setAttribute(String name,Object o);
```

注意:如果 ServletRequest 对象中已经存在指定名称的属性,那么 setAttribute() 方法将会先删除原来的属性,然后再添加新的属性。如果传递给 setAttribute()方法的属性值对象为 null,就删除指定名称的属性,这时的效果等同于 removeAttribute()方法。

(2) getAttribute()方法。该方法用于从 ServletRequest 对象中返回指定名称的属性对象,其完整的语法代码如下:

```
public Object getAttribute(String name);
```

(3) removeAttribute()方法。该方法用于从 ServletRequest 对象中删除指定名称的属性,其完整的语法代码如下:

```
public void removeAttribute(String name);
```

(4) getAttributeNames()方法。该方法用于返回一个包含 ServletRequest 对象中的所有属性名的 Enumeration 对象,在此基础上,可以对 ServletRequest 对象中的所有属性进行

遍历处理。getAttributeNames()方法的完整语法代码如下：

```
public Enumeration getAttributeNames();
```

注意：只有属于同一个请求中的数据才可以通过 ServletRequest 对象传递数据。

5.2.2 HttpServletRequest 应用


视频讲解

1. 获取请求参数

在实际开发中，经常需要获取用户提交的表单数据，例如用户名和密码等，为了方便获取表单中的请求参数，在 HttpServletRequest 接口的 ServletRequest 父接口中定义了一系列获取请求参数的方法，如表 5-4 所示。下面举例说明如何使用这些方法来获取用户提交的表单数据。

【例 5-9】 学生选课。

创建图 5-17 所示的“学生选课”对话框，输入学号、姓名、性别和课程信息，单击“提交”按钮，显示选课成功页面，并在页面上显示学生的相关信息。本例中分别介绍 getParameter()方法、getParameterValues()方法、getParameterNames()方法及 getParameterMap()方法的使用。具体步骤如下所述。

图 5-17 “学生选课”对话框

(1) 在项目 ch5_demo 的 WebContent 文件夹下新建 selCourse.html 页面。该页面中的表单代码如下：

```
1  <form action="SelCourseServlet" method="POST">
2      <label>学号:</label><input type="text"  name="id"/><br>
3      <label>姓名:</label>  <input type="text"  name="name"/><br>
4      <label>性别:</label><input type="radio" name="sex" value="男" checked="checked">男
5      <input type="radio" name="sex" value="女">女<br>
6      <label>课程:</label><input type="checkbox" name="courses" value=" C++"> C++
7      <input type="checkbox" name="courses" value=" Java Web "> Java Web
8      <input type="checkbox" name="courses" value="英语">英语<br>
9      <input type="submit" value="提交"><input type="reset" value="重置">
10 </form>
```

第 4 和第 5 行的两个 radio 输入框的 name 属性取相同值，表示是同一组单选按钮，一次只能取一个值，value 属性值唯一。同样，第 6～8 行的三个 checkbox 输入框的 name 属性值相同，表示一组复选框，value 属性值有多个，系统将以字符串数组存储。

(2) 在项目 ch5_demo 的 com.yzpc.servlet 包中创建 SelCourseServlet.java 类文件。

(3) 使用 getParameter()方法和 getParameterValues()方法接收表单参数。SelCourseServlet.java 类的 doGet()方法代码如文件 5-5 所示。

文件 5-5　SelCourseServlet.java 类的 doGet()方法代码

```
1   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
2       request.setCharacterEncoding("GB18030");
3       //设置客户端浏览器以 GB18030 编码解析数据
4       response.setContentType("text/html;charset=GB18030");
5       PrintWriter out=response.getWriter();
6       /*
7       使用 getParameter()方法 getParameterValues()方法接收表单参数并输出到客户端
8       */
9       //获取填写的编号,id 是文本框的名字,<input type="text" name="id">
10      String id = request.getParameter("id");
11      String name = request.getParameter("name");              //获取填写的姓名
12      String sex = request.getParameter("sex");                //获取选中的性别
13      //获取选中的课程,因为可以选中多个值,所以获取到的值是一个字符串数组,因此需要使用
        getParameterValues()方法来获取
14      String[] courses = request.getParameterValues("courses");
15      String courseStr="";
16      //获取数组数据的技巧,可以避免 courses 数组为 null 时引发的空指针异常错误
17      for (int i = 0; courses!=null && i<courses.length; i++) {
18          if (i == courses.length-1) {courseStr+=courses[i];}else {courseStr+=courses
            [i]+",";}
19      }
20      String htmlStr = "<table>" +
21          "<tr><td>学号:</td><td>{0}</td></tr>" +
22          "<tr><td>姓名:</td><td>{1}</td></tr>" +
23          "<tr><td>性别:</td><td>{2}</td></tr>" +
24          "<tr><td>选中的课程:</td><td>{3}</td></tr>" +
25          "</table>";
26      htmlStr = MessageFormat.format(htmlStr, id,name,sex,courseStr);
27      out.write(htmlStr);                                      //输出 htmlStr 里面的内容到
客户端浏览器显示
28  }
```

第 10～12 行分别读取学号、姓名和性别,这几个参数的值都是唯一的,使用 getParameter()方法读取。第 14 行读取选中的课程名称,由于课程的值会有多个,使用 getParameterValues()方法读取,结果存放在字符串数组 courses 中;如果没有选中的课程,getParameterValues()方法返回 null。为了便于输出,第 17～19 行将字符串数组 courses 转换为一个字符串 courseStr。

注意:getParameter()方法的参数要与提交表单中对应输入框的 name 属性保持一致,否则读取错误。

启动服务器,在浏览器地址栏中输入 http://localhost:8080/ch5_demo/selCoursse.html,显示学生选课页面,如图 5-17 所示。在页面上显示的学生选课表单中填写数据,然后提交到 SelCourseServlet 由 Servlet 进行处理,运行结果如图 5-18 所示。

(4) 若在服务器端使用 getParameterNames()方法接收表单参数,则改写文件 5-5 中的 6～28 行。其代码如下:

```
1   /*
2   使用 getParameterNames() 方法接收表单参数并输出到客户端
3   */
```

```
4   Enumeration<String> paramNames = request.getParameterNames();  //获取所有的参数名
5   while (paramNames.hasMoreElements()) {
6       String name = paramNames.nextElement();                     //得到参数名
7       String value = request.getParameter(name);                  //通过参数名获取对应的值
8       out.println(MessageFormat.format("{0}:{1}<br>", name,value));
9   }
```

图 5-18　getParameter()方法和 getParameterValues()方法的运行结果

说明　在代码中通过 getParameterNames()方法获取所有请求参数并存储到枚举对象 paramNames 中，通过对 paramNames 迭代取得参数名，再通过 getParameter()方法根据参数名获得参数值，最后调用 MessageFormat.format()方法按格式输出参数和参数值。MessageFormat 类在 java.text 包中。使用 getParameterNames()方法的运行结果如图 5-19 所示。

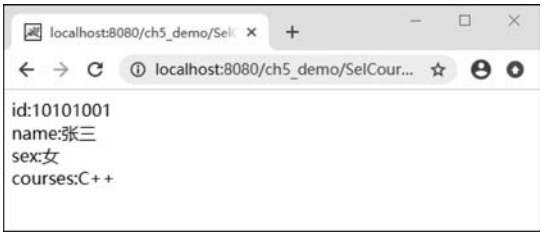

图 5-19　使用 getParameterNames()方法的运行结果

在图 5-19 所示中，只显示了一门课程名称。这是由于第 7 行代码在读取参数值时，使用 getParameter()方法，因此只能显示第一门选中的课程名称。

(5) 在服务器端使用 getParameterMap()方法接收表单参数，改写文件 5-5 中的 6～28 行。其代码如下：

```
1   /*
2   使用 getParameterMap()方法接收表单参数并输出到客户端
3    */
4   Map<String, String[]> paramMap = request.getParameterMap();
5   for(Map.Entry<String, String[]> entry :paramMap.entrySet()){
6       String paramName = entry.getKey();
7       String paramValue = "";
8       String[] paramValueArr = entry.getValue();
9       for (int i = 0; paramValueArr!= null && i < paramValueArr.length; i++) {
10          if (i == paramValueArr.length - 1) {
11          paramValue += paramValueArr[i];
12          }else {
13          paramValue += paramValueArr[i] + ",";
```

```
14            }
15        }
16        out.println(MessageFormat.format("{0}:{1}<br>", paramName,paramValue));
17    }
```

使用 getParameterMap()方法得到了每个参数及其所有值,存储到 Map<String,String[]>类型对象 paramMap 中。通过对 paramMap.entrySet()迭代,从每个 Map.Entry 对象中取得参数的名称 paramName(第 6 行)及该参数的所有值 paramValueArr(第 8 行);第 9～15 行通过循环读出 paramValueArr 的所有值并放到变量 paramValue 中;第 16 行输出该参数的名称和值。使用 getParameterMap()方法的运行结果如图 5-20 所示。

2. HTTP 请求的中文乱码问题

1) 以 POST 方式提交表单中中文参数出现的乱码问题

在文件 5-5 中,将第 2 行"request.setCharacterEncoding("GB18030");"删除。重新启动服务器,根据图 5-17 所示输入表单内容,表单数据提交后,运行结果如图 5-21 所示。

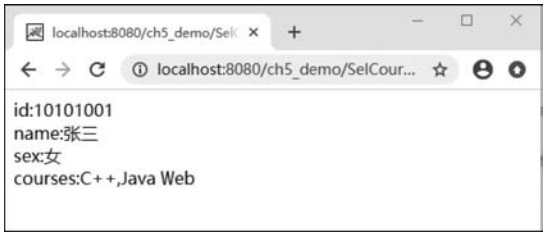

图 5-20　使用 getParameterMap()方法的运行结果

图 5-21　删除文件 5-5 中第 2 行内容后的运行结果

2) POST 方式提交中文数据乱码产生的原因和解决办法

在图 5-21 所示中,第一列中文正常显示,第二列内容中的中文则显示为乱码。这是因为当用户提交表单数据时,其中的数据是以 GB18030 编码的,在服务器端 request 对象读取参数时默认使用 ISO-8859-1 解码,由于编码与解码字符集不一致,所以读取的参数值就是乱码了。

HttpServletRequest 与 HttpServletResponse 在默认情况下都是使用 ISO-8859-1 码表,在本章的 5.1.2 节中分析了在中文输出乱码问题后,在 Servlet 中可以使用 setContentType()方法解决响应输出的中文乱码问题,因此,在输出第一列时能够正常输出;而第二列是 request 对象获取的请求参数的值,由于 request 对象没有能正确解码,得到的是乱码,因此发送到响应消息体的也是乱码。中文乱码问题示意如图 5-22 所示。

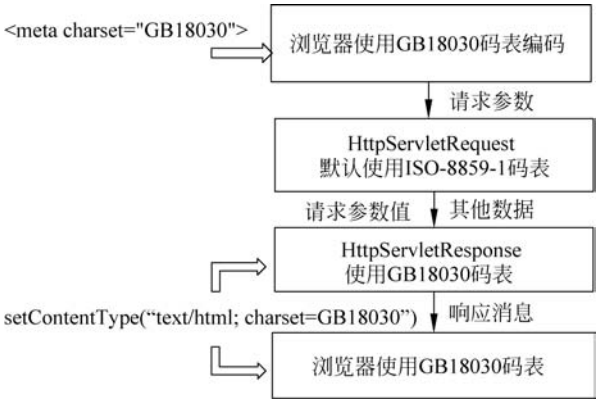

图 5-22　中文乱码问题示意图

由图 5-22 所示可以看到，之所以会产生乱码，就是因为服务器端 HttpServletRequest 对象和客户端浏览器使用的编码不一致造成的。因此，解决的办法是在客户端和 HttpServletRequest 对象之间设置一个统一的编码，之后就按照此编码进行数据的传输和接收。

由于客户端浏览器是以 GB18030 字符编码将表单数据传输到服务器端的，因此服务器端也需要设置以 GB18030 字符编码进行接收，要想完成此操作，服务器端可以直接使用从 ServletRequest 接口继承而来的 setCharacterEncoding(charset)方法进行统一的编码设置。

因此，文件 5-5 中的第 2 行代码请勿删除，否则会产生乱码。

3）以 GET 方式提交表单中文参数的乱码问题

在例 5-9 中，将 selCourse.html 文件中表单的 method 属性改为 GET，其他不变。

启动服务器运行，根据图 5-17 输入表单内容，表单数据提交后，页面依然产生乱码，显然，setCharacterEncoding(charset)方法不能解决 GET 请求的中文乱码问题。运行结果如图 5-23 所示。

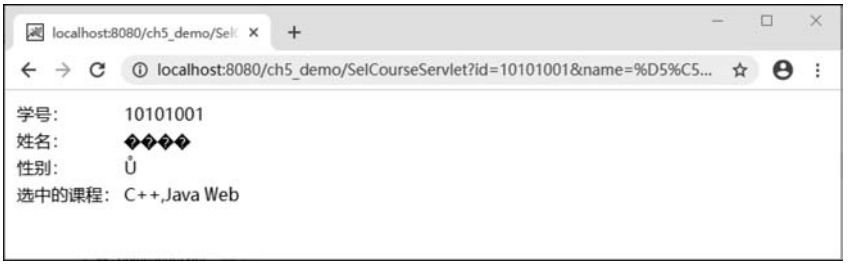

图 5-23　以 GET 方式的运行结果

4）以 GET 方式提交中文数据乱码产生的原因和解决办法

（1）产生的原因。对于以 GET 方式传输的数据，request 即使设置了以指定的编码接收数据也是无效的，默认的还是使用 ISO-8859-1 这个字符编码来接收数据。客户端以 GB18030 的编码传输数据到服务器端，而服务器端的 request 对象使用的是 ISO-8859-1 这个字符编码来接收数据，服务器和客户端沟通的编码不一致，因此才会产生中文乱码。

（2）解决办法。在接收到数据后，先获取 request 对象以 ISO-8859-1 字符编码接收到的原始数据的字节数组，然后通过字节数组以指定的编码构建字符串，即可解决乱码问题。其代码如下：

```
1   String name = request.getParameter("name");                          //接收数据
2   name  = new String(name.getBytes("ISO-8859-1"), "GB18030") ;         //重新编码
```

5）以超链接形式传递中文参数的乱码问题

客户端想传输数据到服务器端，可以通过表单提交的形式，也可以通过超链接后面加参数的形式，例如：

```
<a href = "SelCourseServlet?id = 1101&name = 李磊">单击</a>
```

单击超链接，数据是以 GET 的方式传输到服务器的，所以接收中文数据时也会产生中文乱码问题，而解决中文乱码问题的方式与上述的以 GET 方式提交表单中文数据乱码处理问题的方式一致。

综上所述，HTTP 请求参数的提交方式有 GET 和 POST 两种，对应地，处理中文乱码问

题也有以下两种不同的方式。

(1) 如果提交方式为 POST,那么只需要在服务器端使用 setCharacterEncoding()方法设置 request 对象的编码即可,客户端以哪种编码提交的,服务器端的 request 对象就以对应的编码接收。

(2) 如果提交方式为 GET,那么只能在接收到数据后再手工转换。其代码如下:

```
1   String data = request.getParameter("paramName");     //获取客户端提交上来的数据
2   byte[] source = data.getBytes("ISO-8859-1");         //查找 ISO-8859-1 码表,得到客户端
                                                         //提交的原始数据的字节数组
3   data = new String(source, "GB18030");                //通过字节数组以指定的编码构建字
                                                         //符串,解决乱码
```

通过字节数组以指定的编码构建字符串,这里指定的编码是根据客户端提交数据时使用的字符编码来定的,如果是 GB2312,那么就设置成 data=new String(source, "GB2312"),如果是 UTF-8,那么就设置成 data=new String(source, "UTF-8")

无论使用哪种方式处理中文乱码,最终都要使服务器端与客户端中文编码方式保持一致。

视频讲解

5.3 RequestDispatcher 接口及其应用

5.3.1 RequestDispatcher 接口

当一个 Web 资源收到客户端的请求后,如果希望服务器通知另一个资源处理请求,那么这时可以通过 RequestDispatcher 接口的实例对象实现。ServletRequest 接口中定义了一个获取 RequestDispatcher 对象的方法。其代码如下:

```
public RequestDispatcher getRequestDispatcher(String path);
```

该方法返回一个 RequestDispatcher 对象,该对象充当给定路径的资源,资源可以是动态的也可以是静态的。参数 path 是指定资源路径名的字符串,可以是相对路径。如果路径名以"/"开头,用于表示当前 Web 应用的根目录。

注意:WEB-INF 目录中的内容对 RequestDispatcher 也是可见的。因此,传递给 RequestDispatcher()方法的资源可以是 WEB-INF 目录中的内容。

ServletContext 接口中也定义了同样的方法,和 ServletRequest 接口定义的 getRequestDispatcher()方法唯一的区别就是,ServletContext 接口的 getRequestDispatcher()方法的参数 path 必须以"/"开头。以下三条语句都可以获取资源 welcome.html 的 RequestDispatcher 对象。

```
1   ServletContext context = this.getServletContext();
2   RequestDispatcher rsd = context.getRequestDispatcher("/welcome.html");     //参数必须以"/"开头
3   RequestDispatcher rsd = request.getRequestDispatcher("welcome.html"); //参数不以"/"开头
4   RequestDispatcher rsd = request.getRequestDispatcher("/welcome.html");   //参数以"/"开头
```

获取到 RequestDispatcher 对象后,最重要的工作就是通知其他 Web 资源处理当前的 Servlet 请求,为此,RequestDispatcher 接口定义了两个相关方法。其代码如下:

```
1   public void forward(ServletRequest request, ServletResponse response)
2   public void include(ServletRequest request, ServletResponse response)
```

其中，forward()方法可以实现请求转发，include()方法可以实现请求包含。

注意：当 Servlet 源组件调用 RequestDispatcher 的 forward()方法或 include()方法时，都要把当前的 ServletRequest 对象和 ServletResponse 对象作为参数传给 forward 方法()或 include()方法，这就使得源组件和目标组件共享同一个 ServletRequest 对象和 ServletResponse 对象，就实现了多个 Servlet 协同处理同一个请求。

5.3.2 RequestDispatcher 应用

1. 请求转发

1) 请求转发的基本概念

在 Servlet 中，请求转发是指一个 Web 资源在接收到客户端请求后，通知服务器去调用另一个 Web 资源进行处理，即将原页面的 request 对象和 response 对象传入新的页面，这就使新旧页面拥有相同的 request 对象和 response 对象。请求转发的工作原理如图 5-24 所示。

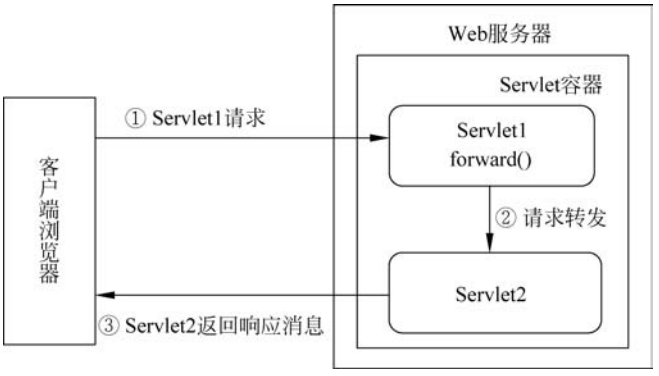

图 5-24　请求转发的工作原理

从图 5-24 所示中可以看出，当客户端访问 Servlet1 时，可以通过 forward()方法将请求转发给其他 Web 资源，其他 Web 资源处理完请求后，直接将响应结果返回到客户端。在这个过程中，客户端和服务器端发生一次请求和一次响应。

【例 5-10】 演示 forward()方法的使用。

(1) 在项目 ch5_Demo 的 com.yzpc.servlet 包中创建一个名为 ForwardServlet.java 的 Servlet 类，该类使用 forword()方法将请求转发到一个新的 welcome.html 页面。ForwardServlet.java 代码如文件 5-6 所示。

文件 5-6　ForwardServlet.java 代码

```
1   package com.yzpc.servlet;
2   import java.io.IOException;
3   import javax.servlet.RequestDispatcher;
4   import javax.servlet.ServletException;
5   import javax.servlet.annotation.WebServlet;
6   import javax.servlet.http.HttpServlet;
7   import javax.servlet.http.HttpServletRequest;
8   import javax.servlet.http.HttpServletResponse;
9   @WebServlet("/ForwardServlet")
10  public class ForwardServlet extends HttpServlet {
11      private static final long serialVersionUID = 1L;
12      protected void doGet(HttpServletRequest request, HttpServletResponse response)
13      throws ServletException, IOException {
```

```
14          RequestDispatcher rsd = this.getServletContext().getRequestDispatcher("/welcome.html");
15          rsd.forward(request, response);
16      }
17      protected void doPost(HttpServletRequest request, HttpServletResponse response)
18      throws ServletException, IOException {
19          doGet(request,response);
20      }
21  }
```

(2) 在 ForwardServlet 中,通过调用 ServletContext 的 getRequestDispatcher(String path)方法,返回一个 RequestDispatcher 对象 rsd,再调用 rsd 的 forward()方法将当前 Servlet 的请求转发到 welcome.html 页面。

启动服务器后,在浏览器地址栏中输入 http://localhost:8080/ch5_demo/ForwardServlet,运行结果如图 5-25 所示。显示 welcome.html 页面的内容,但地址栏没有变化,这是因为,对于客户端来说,它只发出了一次请求,所以请求地址不变。

图 5-25 请求转发的运行结果

如果将文件 5-6 第 14 行代码中 getRequestDispatcher()方法的参数的"/"去掉,重新运行后,发生 IllegalArgumentException 异常,要求参数必须以"/"开头。IllegalArgumentException 异常如图 5-26 所示。

图 5-26 IllegalArgumentException 异常

如果通过 request 对象提供的 getRequestDispatche(String path)方法,获取 RequestDispatcher 对象,比较它们的使用有何不同。

将第 14 行代码改为"RequestDispatcher rsd=request.getRequestDispatcher("welcome.html");"

或者“RequestDispatcher rsd＝request. getRequestDispatcher("/welcome. html");”，重新编译后运行结果都如图 5-25 所示。

说明：request 对象的 getRequestDispatcher()可以使用绝对路径，也可以使用相对路径。其他效果与 ServletContext 对象的 getRequestDispatcher()方法一样。

2）请求转发可以传递数据

request 对象同时也是一个域对象(Map 容器)，开发人员通过 request 对象在实现转发时，把数据通过 request 对象带给其他 Web 资源处理。

【例 5-11】 使用请求转发传递数据。

(1) 在 com. yzpc. servlet 包中创建 Servlet 类 SendDataServlet，设置虚拟路径为“/send”。重写 doGet()方法，在 doGet()方法中调用 request. setAttribute()方法存储域数据，然后将请求转发给 ResultServlet 类。重写 doGet()方法的代码如下：

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset = GB18030");
    request.setAttribute("country", "中国");
    request.setAttribute("city", "扬州");
    RequestDispatcher rsd = request.getRequestDispatcher("ResultServlet");
    rsd.forward(request, response);
}
```

(2) 在 com. yzpc. servlet 包中创建一个名为 ResultServlet 的 Servlet 类，设置虚拟路径为“/ ResultServlet”，该类用于获取 SendDataServlet 类中存储在 request 对象中的数据并输出。ResultServlet 类的代码如下：

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    response.setContentType("text/html;charset = GB18030");
    String country = (String)request.getAttribute("country");
    String city = (String)request.getAttribute("city");
    out.print(country + "< br >" + city);
}
```

启动 Tomcat 服务器，在浏览器的地址栏中输入地址 http://localhost:8080/ch5_demo/send 访问 SendDataServlet，浏览器的显示结果如图 5-27 所示。

图 5-27　request 对象传递数据的运行结果

从图 5-27 所示中可以看出，地址栏中显示的仍然是 ForwardServlet 的请求路径，但是浏览器却显示出了 ResultServlet 中要输出的内容。这是因为，请求转发是发生在服务器内部的

行为,从 RequestForwardServlet 到 ResultServlet 属于一次请求,在一次请求中是可以使用 request 属性进行数据共享的。

3) 请求重定向和请求转发的区别

(1) RequestDispatcher.forward()方法只能将请求转发给同一个 Web 应用中的组件;而 HttpServletResponse.sendRedirect()方法还可以重定向到同一个站点上的其他应用程序中的资源,甚至是使用绝对 URL 重定向到其他站点的资源。

(2) 如果传递给 HttpServletResponse.sendRedirect()方法的相对 URL 以"/"开头,它是相对于整个 Web 站点的根目录;如果创建 RequestDispatcher 对象时指定的相对 URL 以"/"开头,它是相对于当前 Web 应用程序的根目录。

(3) 调用 HttpServletResponse.sendRedirect()方法重定向的访问过程结束后,浏览器地址栏中显示的 URL 会发生改变,由初始的 URL 地址变成重定向的目标 URL;调用 RequestDispatcher.forward()方法的请求转发过程结束后,浏览器地址栏保持初始的 URL 地址不变。

(4) RequestDispatcher.forward()方法的调用者与被调用者之间共享相同的 request 对象和 response 对象,它们属于同一个访问请求和响应过程;而 HttpServletResponse.sendRedirect()方法调用者与被调用者使用各自的 request 对象和 response 对象,它们属于两个独立的访问请求和响应过程。

2. 请求包含

1) 请求包含的基本概念

请求包含指的是使用 include()方法将 Servlet(源组件)请求转发给其他资源(目标组件)进行处理,并将生成的响应结果包含到源组件的响应结果中。

包含与转发相比,源组件与被包含的目标组件的输出数据都会被添加到响应结果中,在目标组件中对响应状态代码或者响应头所做的修改都会被忽略。

【例 5-12】 include()方法的使用。

(1) 在项目 ch5_demo 的 com.yapc.servlet 包中创建 IncludeServlet.java 文件。重写 doGet()方法代码如下:

```
1  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
   ServletException, IOException {
2      response.setContentType("text/html;charset=GB18030");
3      PrintWriter out=response.getWriter();
4      out.print("IncludeServlet: 故人西辞黄鹤楼<br>");
5      //调用 include()方法包含 IncludedServlet
6      request.getRequestDispatcher("IncludedServlet").include(request, response);
7      out.print("IncludeServlet: 烟花三月下扬州<br>");
8  }
```

上述代码中,第 2 行指定响应消息体的编码方式,第 4 行代码输出一行文本,第 6 行执行请求包含方法,将 IncludedServlet 包含到 IncludeServlet 中。

(2) 在 com.yapc.servlet 包中创建 IncludedServlet.java 文件。重写 doGet()方法代码如下:

```
1  protected void doGet(HttpServletRequest request, HttpServletResponse response)
2  throws ServletException, IOException {
3      response.getWriter().print("IncludedServlet: 孤帆远影碧空尽,唯见长江天际流。");
4  }
```

(3) 启动服务器。在浏览器地址栏中输入地址：http://localhost:8080/ch5_demo/IncludeServlet。请求包含运行结果如图 5-28 所示。

图 5-28　请求包含运行结果

由图 5-28 所示看出，运行结果既包含了 IncludeServlet 中的输出文本，又包含了 IncludedServlet 中的输出文本。并且 IncludedServlet 中的输出文本插在 IncludeServlet 中的输出文本的两句中间输出，输出次序与代码书写次序一致。

在 IncludedServlet 中没有指定响应消息体的编码方式，结果也没出现乱码。这是因为浏览器在请求 IncludeServlet 时已经创建了 response 对象，并且指定了编码方式，当客户端对接收到的数据进行解码时，Web 服务器会继续保持调用 response 对象中的信息，从而使 IncludedServlet 中的内容不会发生乱码。

即使 IncludedServlet 中指定响应消息体的编码方式，也会被服务器忽略。如将 IncludeServlet 中 doGet()方法的第 2 行代码删除，在 IncludedServlet 中 doGet()方法的第 1 和第 3 行代码之间添加“response. setContentType("text/html; charset=GB18030");”语句，运行结果输出中文乱码，如图 5-29 所示。

图 5-29　修改后的请求包含运行结果

2) 请求转发和请求包含的区别

(1) 相同点。

请求转发和请求包含都是在处理一个相同的请求，多个 Servlet 之间使用同一个 request 对象和 response 对象。

(2) 不同点。

① 如果在 AServlet 中请求转发到 BServlet，那么在 AServlet 中不允许再输出响应体，即不能使用 response. getWriter() 和 response. getOutputStream() 向客户端输出，这一工作交由 BServlet 来完成；如果是由 AServlet 请求包含 BServlet，就没有这个限制。

② 请求转发不能设置响应消息体，但是可以设置响应消息头，简单来说，就是“留头不留体”。例如：response. setContentType("text/html; charset=GB18030") 是可以留下来的；请求包含不仅可以设置响应消息头，还可以设置响应消息体，简单来说就是“留头又留体”。

③ 请求转发大多应用在 Servlet 中，转发目标大多是 JSP 页面；请求包含大多应用在 JSP 页面中，完成多页面的合并。

5.4 本章小结

本章介绍了 HttpServletRequest 和 HttpServletResponse 接口及其常用方法。分析了 HTTP 请求参数和响应消息体中出现中文乱码问题的原因及解决方案,几种常见的页面重定向的技术包括刷新并跳转、请求重定向、请求转发及这几种方法之间的异同及其应用。