第 3 章 请求与响应

C/S 架构的程序，其数据请求和数据处理都在客户端完成，只在数据存取时访问服务器后台，设计过程相对简单；B/S 架构的程序，其数据请求在客户端的浏览器实现，该请求要发送到后台 Web 服务器，数据处理与响应由后台服务器完成，最终的处理结果再发送回客户端浏览器。因此，B/S 架构的 Web 程序设计存在两大技术难点，一个是如何把客户端的请求数据发送到服务器，另一个是服务器将请求所需数据处理完成后如何把结果返回给客户端浏览器。

3.1 请求响应模型


视频讲解

客户端的 JSP 页面一般是将页面数据和请求信息放置在< form >标签内部向 Web 服务器完成表单提交，利用< form >标签的 action 属性指定请求提交的目的处理程序，该程序一般是某个 Servlet 或另外的 JSP 页面。

Servlet 最主要的功能就是处理客户端请求，并将处理结果返回给客户端，该过程称为响应。客户端浏览器访问 Servlet 的具体过程如图 3-1 所示，在该过程中 Java Web 程序处理请求响应的具体流程描述如下。

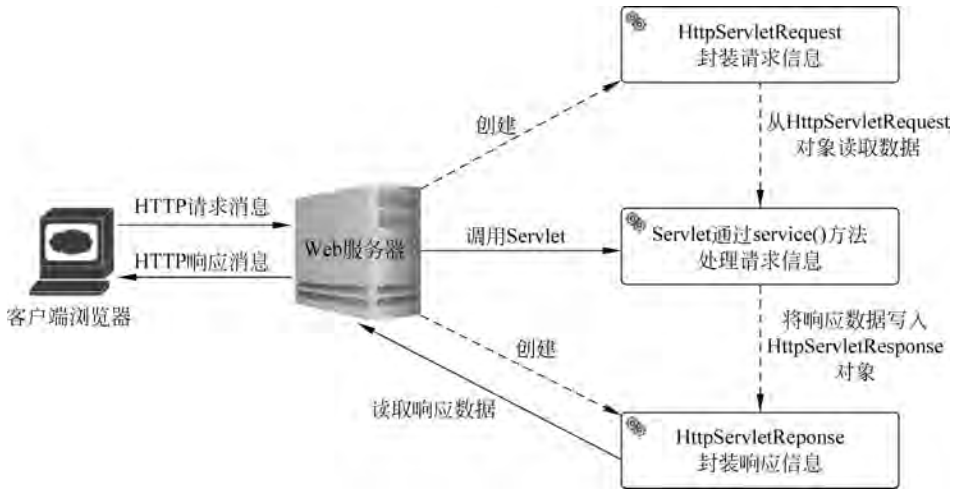

图 3-1 浏览器访问 Servlet 对应的请求响应模型

(1) 客户端的浏览器通过 JSP 页面中的 form 表单将数据进行封装，然后向 Web 服务器发送 HTTP 请求。

(2) Web 服务器接收到客户端发来的 HTTP 请求，创建 HttpServletRequest 对象，将 form 表单中的数据和请求信息拆解并重新封装到 HttpServletRequest 对象中。

(3) Web 服务器在创建 HttpServletRequest 对象的同时,一并创建 HttpServletResponse 对象,用来接收即将返回的响应信息。

(4) Web 服务器根据 form 表单中 action 属性设定的 Servlet 映射调用对应 Servlet 的 service(HttpServletRequest request, HttpServletResponse response)方法进行请求处理,该方法中的两个参数 request 和 response 对应步骤(2)中的 HttpServletRequest 对象和步骤(3)中的 HttpServletResponse 对象。

(5) service()方法将请求处理完毕后将所需的响应信息封装到 response 参数中,该参数负责将响应信息写入到步骤(3)中创建的 HttpServletResponse 对象。

(6) Web 服务器读取 HttpServletResponse 对象中的响应数据,将响应数据封装到特定的页面中返回给客户端浏览器。

上述过程中产生的 HttpServletRequest 对象用于封装 HTTP 请求信息,简称 request 对象; HttpServletResponse 对象用于封装 HTTP 响应信息,简称 response 对象。这两个对象作为 JSP 九大内置对象在后续章节还会深入讨论。

需要注意的是,在 Web 服务器运行期间,每个 Servlet 类只会创建一个实例对象,该对象负责处理多个 HTTP 请求并做出响应。但对于每次 HTTP 请求,Web 服务器都会调用一次 Servlet 实例对象的 service(HttpServletRequest request, HttpServletResponse response)方法,每调用一次 service 方法就会重新创建一个 request 对象和一个 response 对象。

视频讲解

3.2 HttpServletRequest 对象

HttpServletRequest 是一个继承自 ServletRequest 接口的子接口,主要用于封装 HTTP 的请求信息。HTTP 请求消息分为请求行、请求消息头和请求消息体三部分,如表 3-1 所示。

表 3-1 HTTP 消息结构

消息部分	说明
请求行或状态行	指定请求或响应消息的目的
请求头或响应头	指定元信息,如关于消息内容的大小、类型、编码方式等
空行	
消息体	请求或响应消息的主要内容,可以为空

表 3-2 给出了一个典型的 POST 请求中的详细请求信息。

表 3-2 典型的 POST 请求信息

组成部分	说明
请求行	POST/WebJDBC/CheckLoginServlet HTTP/1.1
请求头	Accept=*/* Accept-Language=zh-ch Accept-Encoding=gzip,deflate User-Agent=Mozilla/4.0(compatible; MSIE 9.0; Windows NT 5.1; SV1;.NET CLR 1.1.4322;.NET CLR 2.0.50727) Host=localhost:8080 Connection=Keep-Alive
空行	
消息体	可以为空

HttpServletRequest 接口中主要定义了获取这三部分数据信息的相关函数。

3.2.1 获取请求行

客户端浏览器访问 Servlet 时，会发送相应的请求消息给服务器，该请求消息分为请求行、请求消息头和请求消息体三部分。其中，在请求行部分包含请求方法名、请求资源的 URL 和 HTTP 版本等信息，这三部分由空格隔开。如表 3-2 中第一行数据所示，请求方法为 POST，请求资源的 URI 为/WebJDBC/CheckLoginServlet，使用的协议及其版本为 HTTP/1.1。

HttpServletRequest 接口中定义了获取请求行信息的相关方法，如表 3-3 所示。

表 3-3 获取请求行信息的方法

方 法	说 明
String getMethod()	获取 HTTP 请求的请求方式(如 GET、POST 等)
String getRequestURI()	获取请求行中资源名称部分，即位于 URL 的主机和端口之后、参数之前的部分
String getQueryString()	获取请求行中的参数部分，即资源路径后面问号以后的所有内容
String getProtocol()	获取请求行中的协议名及版本，如 HTTP/1.0 或 HTTP/1.1
String getContextPath()	获取 URL 中 Web 应用程序的路径
String getServletPath()	获取 Servlet 的名称或 Servlet 所映射的路径
String getRemoteAddr(0)	获取客户端的 IP 地址
String getRemoteHost()	获取客户端的完整主机名
int getRemotePort()	获取客户端的访问端口号
String getLocalAddr()	获取服务器 IP 地址
String getLocalName()	获取服务器主机名
int getLocalPort()	获取服务器端口号
String getServerName()	获取当前请求所指向的主机名，即 HTTP 请求消息中 Host 头字段所对应的主机名部分
int getServerPort()	获取当前请求所连接的服务器端口号
String getScheme()	获取请求的协议名，如 http、https 或 ftp 等
StringBuffer getRequestURL()	获取客户端发送请求的完整 URL(不含参数部分)

【例 3-1】 测试 HttpServletRequest 对象获取请求行相关函数。

(1) 在 Eclipse 中新建一个名为“0301-RequestLineInfo”的项目。

(2) 新建一个名为“GetRequestLineInfoServlet”的 Servlet，放置在 cn.pju.servlets 包下，详细代码如下。

```
package cn.pju.servlets;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
@WebServlet("/GetRequestLineInfoServlet")
public class GetRequestLineInfoServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;
   public GetRequestLineInfoServlet() {
        super();
    }
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
     System.out.println("getMethod(): " + request.getMethod());
     System.out.println("getRequestURI(): " + request.getRequestURI());
     System.out.println("getRequestURL(): " + request.getRequestURL());
     System.out.println("getQueryString(): " + request.getQueryString());
     System.out.println("getProtocol(): " + request.getProtocol());
     System.out.println("getContextPath(): " + request.getContextPath());
     System.out.println("getServletPath(): " + request.getServletPath());
     System.out.println("getRemoteAddr(): " + request.getRemoteAddr());
     System.out.println("getRemoteHost():" + request.getRemoteHost());
     System.out.println("getRemotePort():" + request.getRemotePort());
     System.out.println("getLocalAddr():" + request.getLocalAddr());
     System.out.println("getLocalName():" + request.getLocalName());
     System.out.println("getLocalPort():" + request.getLocalPort());
     System.out.println("getServerName():" + request.getServerName());
     System.out.println("getServerPort():" + request.getServerPort());
     System.out.println("getScheme():" + request.getScheme());
     System.out.println("getRequestURL():" + request.getRequestURL());
  }

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

}
```

(3) 将该项目配置到 Tomcat 服务器,然后运行 GetRequestLineInfoServlet,在浏览器中输入 http://localhost:8080/0301-RequestLineInfo/GetRequestLineInfoServlet? info=justatest,查看 Console 控制台的输出信息,如图 3-2 所示。

由于在访问过程中使用的是本机主机名 localhost,所以 getRemoteAddr()、getRemoteHost()、getLocalAddr()和 getLocalName()函数返回的结果都是“0:0:0:0:0:0:0:1”,即 IPv6 中的本机地址[::1]。用户可以将 localhost 修改为本机的外部访问 IP 地址或局域网 IP 地址(在命令提示符下输入“ipconfig/all”进行查看),如作者当前计算机的局域网 IP 地址为 192.168.0.111,所以在浏览器中输入如下 URL 地址 http://192.168.0.111:8080/0301-RequestLineInfo/GetRequestLineInfoServlet? info=justatest,则对应的测试结果如图 3-3 所示。

图 3-2　HttpServletRequest 函数运行结果

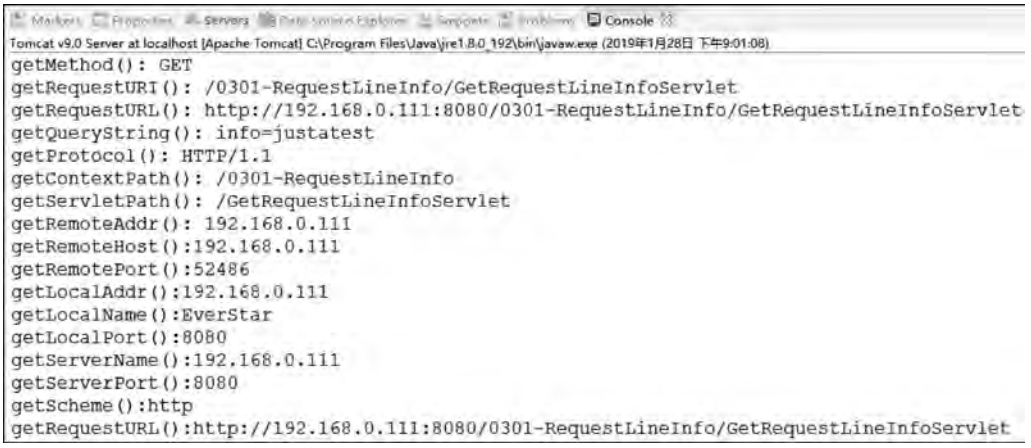

图 3-3　使用局域网 IP 地址测试 HttpServletRequest 函数运行结果

3.2.2　获取请求消息头

HTTP 请求消息中的请求消息头主要用来向服务器传递附加信息，例如，字符集编码、语言、压缩类型等。HttpServletRequest 接口定义了常用的获取 HTTP 请求头字段信息的方法，见表 3-4。

表 3-4　获取请求头信息的方法

方　法	说　明
String getHeader(String name)	获取 HTTP 请求中指定头字段的值，若不存在返回 NULL；存在多个时返回首个
Enumeration getHeaders(String name)	返回名为 name 的所有头字段值所组成的 Enumeration 集合对象
Enumeration getHeaderNames()	获取所有请求头字段组成的 Enumeration 对象
int getIntHeader(String name)	获取 HTTP 请求中名为 name 的头字段的值，并转换为 int 类型

续表

方　法	说　明
long getDateHeader(String name)	获取 HTTP 请求中名为 name 的头字段的值，并转换为 GMT 时间格式的长整型
String getContentType()	获取头字段 Content-Type(内容类型，即文件的类型和网页的编码)的值
int getContentLength()	该方法用于获取 Content-Length 头字段的值
String getCharacterEncoding()	返回请求消息实体部分的字符集编码

【例 3-2】 测试 HttpServletRequest 对象获取请求头相关函数。

(1) 在项目 0301-RequestLineInfo 中新建一个名为 GetRequestHeaderInfoServlet 的 Servlet，放置在 cn.pju.servlets 包下，详细代码如下。

```
package cn.pju.servlets;

import java.io.IOException;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/GetRequestHeaderInfoServlet")
public class GetRequestHeaderInfoServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

    public GetRequestHeaderInfoServlet() {
        super();
    }

  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    request.setCharacterEncoding("UTF-8");
    //循环遍历所有的请求头，并输出请求信息
    Enumeration hs = request.getHeaderNames();
    while(hs.hasMoreElements()) {
        String hName = (String) hs.nextElement();
        System.out.println(hName + " : " + request.getHeader(hName));
    }
    System.out.println(request.getCharacterEncoding());
  }

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

(2) 在浏览器中输入网址 http://localhost:8080/0301-RequestLineInfo/GetRequestLineInfoServlet? info=justatest,运行 GetRequestHeaderInfoServlet,查看 Console 控制台的输出信息,如图 3-4 所示。

图 3-4　HttpServletRequest 请求头信息结果

3.2.3　相关应用

1. 获取请求参数

在 Web 项目开发中,通常需要通过前端页面中的 form 表单向后台服务器传递数据,例如登录界面中的用户名、密码、随机验证码等。这些参数传递到后台之后,可以被 HttpServletRequest 接口及其父类 ServletRequest 对应的一系列方法进行获取。获取请求参数的方法如表 3-5 所示。

表 3-5　获取请求参数的方法

方　法	说　明
String getParameter(String name)	获取 HTTP 请求中指定名称为 name 的参数值,若不存在返回 NULL;如存在但未设置值,则返回空串;如果存在多个时返回首个
String[] getParameterValues(String name)	若传递的 HTTP 请求中存在多个名为 name 的参数值,使用该函数读取所有的参数值,放入一个字符串数组后返回
Enumeration getParameterNames()	返回 HTTP 请求中由所有参数名所组成的 Enumeration 集合对象,利用本函数及以上两个函数可以实现对请求消息中所有参数的遍历操作
Map getParameterMap()	将请求消息中所有的参数名和对应的参数值封装进一个 Map 对象并返回

【例 3-3】 利用 HttpServletRequest 对象获取客户端传入的指定名称参数值。

(1) 新建名为 0303-RequestApp 的动态 Web 项目。

(2) 在 WebContent 目录下新建 regist.jsp 页面,代码如下。

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>用户注册</title>
</head>
<body>
    <form action="/0303-RequestApp/RequestParamsServlet" method="post">
        账号:<input type="text" name="username"><br>
        密码:<input type="password" name="password"><br>
        研究方向:
        <input type="checkbox" name="researcharea" value="hardware">计算机硬件
        <input type="checkbox" name="researcharea" value="software">软件工程
        <input type="checkbox" name="researcharea" value="bigdata">大数据<br>
        <input type="submit" value="提交">
    </form>
</body>
</html>
```

(3) 新建 cn.pju.requestapp.svlt 包,在该包下新建名为 RequestParamsServlet 的 Servlet 类,代码如下。

```
package cn.pju.requestapp.svlt;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/RequestParamsServlet")
public class RequestParamsServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        String[] researcharea = request.getParameterValues("researcharea");
        System.out.println("用户名:" + username);
        System.out.println("密码:" + password);
        System.out.println("研究领域:");
        for (String ra : researcharea) {
            System.out.println(ra);
        }
    }

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

(4) 将项目配置到 Tomcat 容器上，启动服务器，在浏览器中输入访问地址 http://localhost:8080/0303-RequestApp/regist.jsp，打开如图 3-5 所示的用户注册界面。

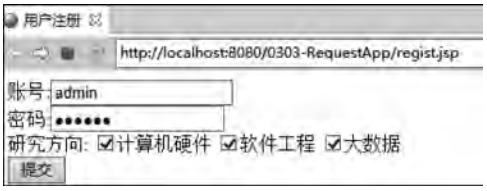

图 3-5　用户注册界面

(5) 在账号栏和密码栏中分别输入 admin 和 sysman，全选三个研究方向复选框，单击“提交”按钮，观察后台 Console 窗口数据显示，如图 3-6 所示。

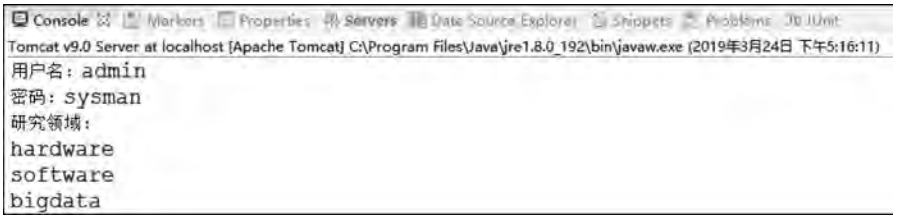

图 3-6　运行结果

2. 解决请求参数的中文乱码问题

如果在如图 3-5 所示的用户注册界面输入中文字符的账号(比如“张俊”)，再次单击“提交”按钮发送至后台的 Servlet 程序处理，则解析结果为乱码，如图 3-7 所示。

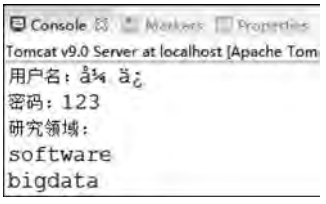

图 3-7　中文参数值解析出现乱码

在此分析一下，客户端浏览器向后台服务器传递中文参数值，出现中文乱码现象的原因。

首先，当客户端浏览器表单中输入了中英文参数值并通过表单提交按钮向后台服务器提交时，浏览器会根据默认设置的编码格式对参数值进行封装，不同浏览器的默认字符编码不尽相同，而且用户可以根据自身需要进行调整设置。以 IE 浏览器为例，在客户端输入中文账号，在浏览器空白区域右击，展开“编码”菜单，可见浏览器当前使用的默认编码方式为 Unicode(UTF-8)，如图 3-8 所示，也可根据需要切换简体中文(GB2312)字符编码格式。浏览器设置的字符编码不同，输入框中的数据传递给 form 表单的实际二进制编码内容也不同。

其次，客户端封装后的数据传递到后台服务器，由 HttpServletRequest 对象进行数据解析，解码时也会用到字符编码，如果解码时所使用的字符编码与封装时使用的字符编码不一致，就会出现中文乱码现象。在 Servlet 中可以使用函数 setCharacterEncoding(String encoding) 来设置 Request 对象编码格式，如 request.setCharacterEncoding("UTF-8")，需要注意的是该语句必须写在 doGet()函数的首行，这是因为 doGet()函数在执行函数代码之前首先检测是否有字符编码设置函数语句进行了编码设置，如果未发现则默认采用 ISO-8859-1 编码格式作为首选，然后继续执行函数体内其他语句，当再次遇到 setCharacterEncoding()函数

图 3-8　浏览器编码格式

设置不同代码时,将忽略该语句。

最后,客户端参数的编码封装格式还与 form 表单的提交方式有关,这是因为 setCharacterEncoding()函数只对 post 方式提交的表单有效,对 get 方式提交的表单无效。

有的读者可能会产生疑问,为什么英文字母和阿拉伯数字不存在乱码问题,而唯独中文会出现乱码? 这是因为英文字母和阿拉伯数字数量相对较少,所以在 ISO-8859-1、Unicode (UTF-8)和简体中文(GB2312)等常用编码中,这些符号的编码都与其 ASCII 码值保持一致,也就是说在这些编码机制中,其编码都是相同的,所以无论使用何种编码进行封装和解析,都不会出现乱码现象。而中文具有其特殊性,在不同的编码格式中,同一个汉字对应的编码是不同的。比如在 A 编码格式中,汉字"中"存储在 x 行 y 列,我们将其编码设定为 xy,而在 B 编码格式中,xy 编码对应的 x 行 y 列存储的可能是另一个汉字或者是某个汉字的一部分,此时就会出现乱码或前后解析不符的现象。其实不光中文数据在客户端与服务器端编码不一致时会产生乱码现象,常见的 CJK 编码都会出现类似现象,这是由这些语言的特殊编码格式所决定的。

总结以上分析可见,客户端浏览器将表单数据进行封装然后向后台服务器传递中文参数值,服务器端由 Request 对象再进行数据解析,出现乱码现象与客户端浏览器编码格式、表单提交方式和服务器端 Request 对象进行解析时使用的 CharacterEncoding 编码格式三个因素有关。在实际使用过程中,只要三者设置保持同一种编码或相互兼容的编码格式,就可避免中文解析出现乱码现象。修改后的源程序文件 RequestParamsServlet.java 代码如下。

```
package cn.pju.requestapp.svlt;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/RequestParamsServlet")
public class RequestParamsServlet extends HttpServlet {

  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    request.setCharacterEncoding("UTF-8");
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    String[] researcharea = request.getParameterValues("researcharea");
    System.out.println("用户名: " + username);
    System.out.println("密码: " + password);
    System.out.println("研究领域: ");
    for (String ra : researcharea) {
        System.out.println(ra);
    }
  }

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

3. 通过 Request 对象传递数据

在客户端的浏览器通过表单中的控件向后台 Servlet 传递的数据一般被当作 Parameter 参数保存,如果想向 Request 对象中插入其他非参数变量,则可使用 setAttibute()方法设置参数,使用 getAttribute()函数进行参数值的读取。Request 对象常用的属性操作函数如表 3-6 所示。

表 3-6 Request 对象常用的属性操作函数

方 法	说 明
void setAttribute(String name, Object obj)	向 ServletRequest 对象中写入名为 name 的属性并设置其值为 obj;若已存在 name 属性,则覆盖;若 obj 值为 null,则删除指定属性,效果等同于 removeAttribute()
Object getAttribute(String name)	获取 ServletRequest 对象中名为 name 的属性值
removeAttribute(String name)	在 ServletRequest 对象中删除名为 name 的属性
Enumeration getAttributeNames()	返回一个包含 ServletRequest 对象中所有属性的 Enumeration 对象

修改源程序文件 RequestParamsServlet.java，将注册结果写入 ServletRequest 对象的 registOK 属性，代码如下。

```
package cn.pju.requestapp.svlt;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.sun.org.apache.xpath.internal.operations.And;
@WebServlet("/RequestParamsServlet")
public class RequestParamsServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    response.setCharacterEncoding("UTF-8");
    request.setCharacterEncoding("UTF-8");
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    String[] researcharea = request.getParameterValues("researcharea");
    System.out.println("用户名: " + username);
    System.out.println("密码: " + password);
    System.out.println("研究领域: ");
    for (String ra : researcharea) {
        System.out.println(ra);
    }
    //注册结果判断
    if(username != null && !username.isEmpty() && password != null && !password.isEmpty() &&
researcharea != null && researcharea.length > 0 ) {
        request.setAttribute("registOk", true);
    } else {
        request.setAttribute("registOk", false);
    }
    if((Boolean)request.getAttribute("registOk")) {
        response.getWriter().println("注册成功");
    } else {
        response.getWriter().println("注册失败");
    }
  }
  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    doGet(request, response);
  }
}
```

输入合法的用户名、密码，选择对应的研究领域选项，单击“提交”按钮，客户端浏览器反馈结果如图 3-9 所示。

(a) 用户注册页面　　(b) 注册成功提示页面

图 3-9　注册成功提示

3.3　HttpServletResponse 对象

视频讲解

HTTP 应答与 HTTP 请求相似，也由 3 个部分构成，分别是状态行、响应头(Response Header)和响应正文。在 Servlet API 中，一般通过 HttpServletResponse 接口将服务器端的信息传递给客户端，HttpServletResponse 接口专门用来封装 HTTP 响应消息，它是 ServletResponse 接口的子接口。与 HTTP 响应消息分为状态行、响应消息头、消息体三部分相对应，HttpServletResponse 接口中定义了向客户端发送响应状态码、响应消息头、响应消息体的三类方法。

3.3.1　常用方法

1. 发送状态码相关的方法

Servlet 向客户端浏览器回送 Response 响应消息时，需要在消息中设置一个状态码。HttpServlet Response 接口定义了两个常用发送状态码的方法。

1) setStatus(int status)方法

设置 HTTP 响应消息状态码的值。Response 响应状态行中的状态描述信息直接与状态码相关，如 404 状态码表示找不到客户端所请求的资源。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。状态行由协议版本、数字形式的状态代码，及相应的状态描述组成，各元素之间以空格分隔。而 HTTP 版本由服务器确定，因此只要通过 setStatus(int status)方法设置了状态码，即可实现状态行的发送。

2) sendError(int code)方法

当需要向客户端发送错误状态码时可使用此方法，比如 sendError(404) 表示找不到客户端请求的资源。该方法提供了以下两个重载格式。

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

其中，sendError(int code)方法只是发送错误信息的状态码，sendError(int code, String message)方法在发送错误状态码 code 的同时还增加了一条用于提示说明的文本信息 message，该文本信息将会显示在发送给客户端的正文内容中。

2. 发送响应消息头相关的方法

由于 HTTP 有多种响应头字段，所以在 HttpServletResponse 接口中对应定义了一系列设置 HTTP 响应头字段的方法，如表 3-7 所示。

表 3-7　设置响应消息头字段的相关方法

方　法	说　明
void addHeader(String name, String value)	添加响应头的名字，以及与响应头名字对应的值。如果 name 对应的响应头不存在，则新增一个；如果响应头已存在，则增加一个同名的 name 响应头，HTTP 响应消息中允许同一名称的头字段出现多次
void setHeader(String name, String value)	设置与响应头名字对应的值。如果 name 对应的响应头不存在，则新增一个；如果响应头已存在，则将用新的设置值取代原来的设置值
void addIntHeader(String name, int value)	添加响应头的名字，以及与响应头名字对应的整型值。函数用法与 addHeader(String name, String value)类似，仅用于设置响应头对应值为整型时的情况
void setIntHeader(String name, int value)	设置与响应头名字对应的整型值。函数用法与 setHeader(String name,String value)类似，仅用于设置响应头对应值为整型时的情况
void setContentLength(int len)	设置响应消息中实体内容的长度，单位是字节。对于 HTTP 来说，该方法等价于设置 Content-Length 响应头字段的值
void setContentType(String type)	设置 Servlet 输出内容的 MIME 类型，对于 HTTP 而言，就是设置 Content-Type 响应头字段的值。例如，如果发送到客户端的内容是 jpeg 图像数据时，就需要将响应头字段的类型设置为“image/jpeg”；如果响应的内容为文本，需要使用 setContentType()方法设置字符编码，如 text/html;charset=UTF-8
void setLocale(Locale loc)	设置响应消息的本地化信息。即设置 Content-Language 响应头字段和 Content-Type 头字段中的字符集编码。如果 HTTP 消息没有设置 Content-Type 头字段，setLocale()方法设置的字符集编码就不会出现在任何 HTTP 消息的响应头中，如果调用 setCharacterEncoding)或 setContentType()方法指定了响应内容的字符集编码，setLocale()方法将不再具有指定字符集编码的功能
void setCharacterEncoding(String charset)	设置输出内容使用的字符编码，即设置 Content-Type 头字段中的字符集编码部分。如果没有设置 Content-Type 头字段，setCharacterEncoding 方法设置的字符集编码不会出现在 HTTP 消息的响应头中。SetCharacterEncoding()方法比 setContentType()和 setLocale()方法的优先权高，它的设置结果将覆盖 setContentType()和 setLocale()方法所设置的字符码表

3. 发送响应消息体的相关方法

设置响应消息体的相关方法如表 3-8 所示。

表 3-8　设置响应消息体的相关方法

方　法	说　明
ServletOutputStream getOutputStream()	获取 ServletOutputStream 类型的字节输出流。该函数可以直接输出字节数组中的二进制数据，常用于输出二进制格式的响应正文
PrintWriter getWriter()	获取 PrintWriter 类型的字符输出流对象，该对象可以直接输出字符文本内容

【例 3-4】 测试 HttpServletResponse 对象发送消息体相关函数。

(1) 在 Eclipse 中新建一个名为 0304-MyResponse 的项目,并将整个项目的编码格式设置为 UTF-8。

(2) 新建一个名为 MyPrintServlet 的 Servlet,放置在 cn. pju. response 包下,详细代码如下。

```
package cn.pju.response;

import java.io.IOException;
import java.io.OutputStream;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/MyPrintServlet")
public class MyPrintServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

    public MyPrintServlet() {
    }

  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    response.setCharacterEncoding("UTF-8");                    //第一句,设置服务器端编码
    response.setHeader("Content-Type","text/html;charset=UTF-8"); //第二句,设置浏览器端解码
    String data = "Java Web 程序设计";
    OutputStream out = response.getOutputStream();
    out.write(data.getBytes());
  }

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

}
```

(3) 将项目配置到 Tomcat 容器上,启动服务器,在浏览器中输入访问地址 http://localhost:8080/0304-MyResponse/MyPrintServlet,运行结果如图 3-10 所示。

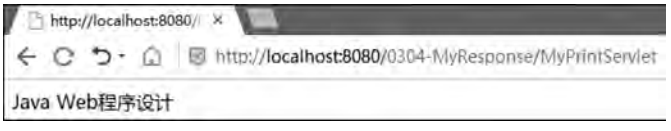

图 3-10 运行结果

(4) 修改程序代码,改用 response 对象的 getWriter()方法获取 PrintWriter 对象,然后调用该对象的 write(String data)方法直接输出字符型数据,代码如下。

```
package cn.pju.response;

import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/MyPrintServlet")
public class MyPrintServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

    public MyPrintServlet() {
    }

  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    String data = "Java Web 程序设计";
    PrintWriter writer = response.getWriter();
    writer.write(data);
  }

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

程序运行效果同图 3-10。

需要注意的是,response 对象的 getOutputStream()、getWriter()函数不可同时使用,否则会发生 IllegalStateException 异常。

3.3.2 相关应用

1. 解决中文输出乱码问题

在 Web 项目开发过程中,经常会用到几种常见的字符编码,有 ISO-8859-1、UTF-8、GBK、GBK2312、Big5 等。英文字母和阿拉伯数字在这些编码表中的数值是一致的,所以采用不同的编码方式进行英文字母输出,不会出现乱码问题。而对于 CJK 字符集,在不同的字符编码集合中其对应的编码是不相同的,甚至有的字符编码集合不支持 CJK 字符,所以 Java Web 项目输出中文字符时,有可能出现乱码,其主要原因就是后台程序与前端浏览器

所使用的字符编码不一致。

1）Java Web 项目整个项目的编码格式

在 Eclipse 中选中当前项目，选择右击弹出菜单中的 Properties，打开项目属性对话框，弹出如图 3-11 所示的属性设置窗口，选择左侧列表中的 Resource，在右侧的 Text file encoding 属性中选择对应的编码格式。

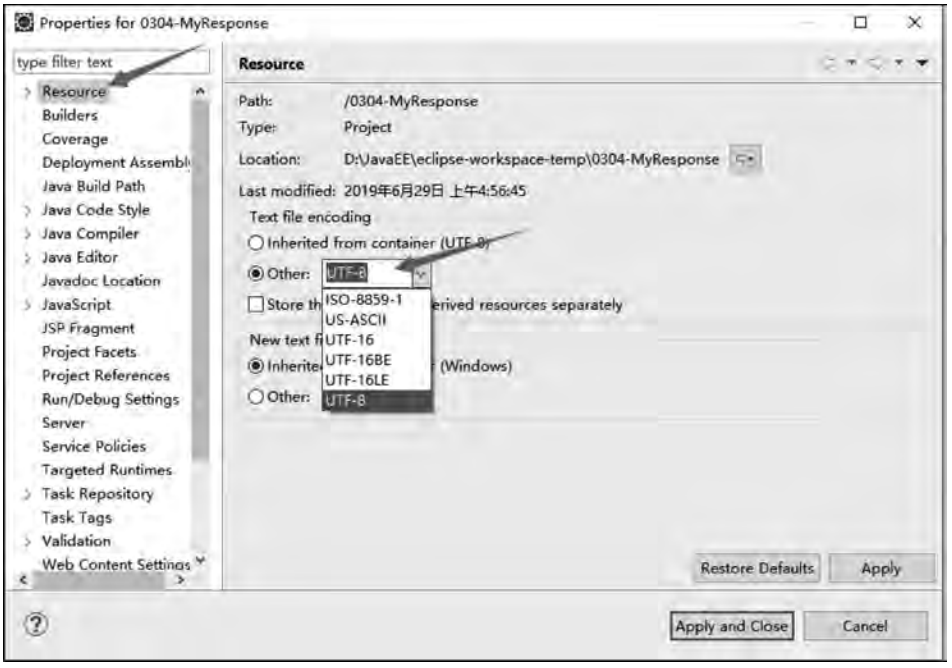

图 3-11　项目编码设置

2）后台源程序文件的编码格式

如果在项目开发过程中先设置了整个项目的编码格式，则后续新建的源代码文件都将继承项目的编码格式以保持项目本身的整体一致性。如新建项目之后立刻设置了项目的编码格式为 UTF-8，然后再创建源代码文件，则源代码文件的默认编码格式也会是 UTF-8。但也有读者是先创建了项目，然后创建了几个源程序文件之后才想起要设置项目的编码格式，但此时设置了项目编码格式之后，源程序文件的编码格式不会随着项目编码格式的改变而自动改变，这就需要手动设置源程序文件的编码格式，以保持与项目编码格式的一致。

在 Project Explorer 窗口选中需要设置的源代码文件，在右击弹出菜单中选择 Properties，在如图 3-12 所示的源程序文件属性设置窗口中设置其编码格式。

3）程序运行时后台服务器对应的编码格式

前端用户浏览器接收到的数据都是由后台服务器发送而来的，Web 服务器需要将后台的数据封装打包发送，所以后台程序也有自己的编码格式，该编码格式使用 response.setCharacterEncoding("UTF-8")语句进行设置。

4）前端页面输出时的编码格式

前端客户浏览器接收到数据之后，应按照服务器预期的编码格式进行解码，然后输出显示。在程序中通过 response.setHeader("Content-Type","text/html;charset=UTF-8")设

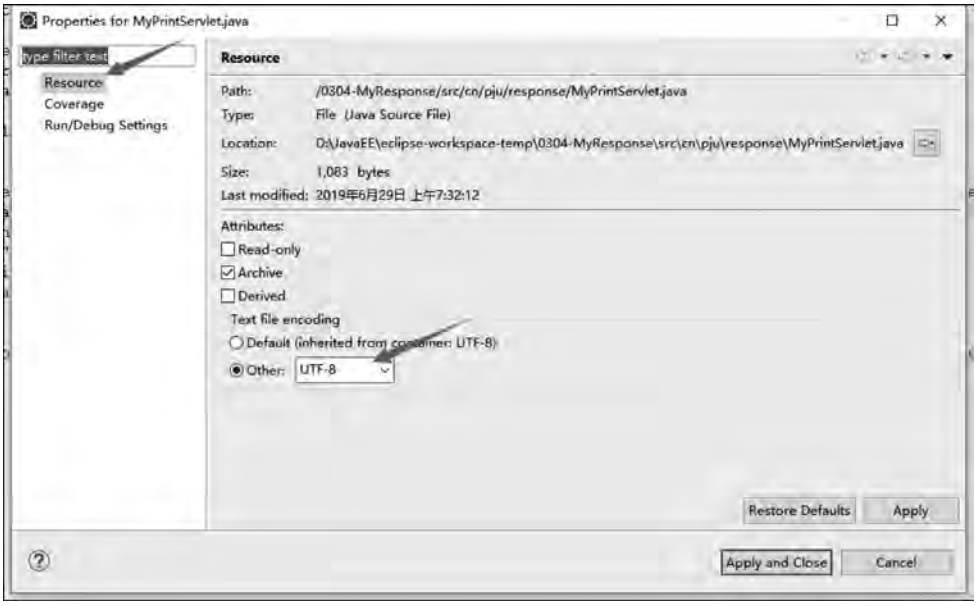

图 3-12　源程序文件属性设置

置浏览器的期望编码格式，有些浏览器设置了自动编码解析，只要在程序中设置了期望编码，浏览器就会自动切换相应的编码进行输出显示。

5）用户端浏览器实际使用的编码格式

客户浏览器端的实际使用编码不受源程序的限制，用户可以通过如图 3-13 所示方法自行强制调整，但通常情况下，客户端浏览器是启用编码自动检测功能的。

图 3-13　浏览器端的编码设置

如果想避免中文字符编码乱码现象出现，建议以上五种编码方式尽量全部统一。在Servlet 源程序代码中，一般使用以下两种方法进行编码设置，以确保避免乱码现象出现。

（1）分别设置服务器端和客户端的编码方式。

```
response.setCharacterEncoding("UTF-8");  //设置服务器端 HttpServletResponse 使用 UTF-8 编码
response.setHeader("Content-Type","text/html;charset=UTF-8"); //通知浏览器使用 UTF-8 编码
```

（2）使用 setContentType()函数一次性设置，该函数包含以上两行代码的相同功能而且简洁明了，所以一般经常使用。

```
response.setContentType("text/html;charset=UTF-8");
```

在例 3-4 的两种写法中，其实就分别使用了上述两种方法进行编码设置，读者可仔细阅读比较一下。

2. 解决中文输出乱码问题

在 HTTP 中定义了一个 Refresh 头字段，用来设置指定时间内自动刷新和到期后的页面跳转。

【例 3-5】 测试 HttpServletResponse 对象的定时刷新功能。

在项目 0304-MyResponse 的源程序文件夹中新建一个名为 RefreshServlet 的 Servlet 类，代码如下。

```
package cn.pju.response;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/RefreshServlet")
public class RefreshServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

    public RefreshServlet() {
        super();
    }

  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    response.setHeader("Refresh", "5;URL=http://www.163.com");
  }

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

}
```

其中，response.setHeader("Refresh", "5;URL=http://www.163.com")语句表示设置页面 5s 之后跳转到 http://www.163.com 页面。分号前的 5 代表 5s 后进行刷新，如果在分号后添加了 URL 属性，则跳转到对应的页面。如果不设置 URL 属性，则只对当前页面进行刷新，不进行页面跳转。

例如，将以上程序修改为如下源代码，即可实现系统时间的每秒输出。

```
package cn.pju.response;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/RefreshServlet")
public class RefreshServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;
    public RefreshServlet() {
        super();
    }

  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    //设置每隔 2s 定期刷新
    response.setHeader("Refresh", "2");
    //输出服务器当前时间
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    response.getWriter().println(sdf.format(new java.util.Date()));
  }

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

小　　结

本章主要介绍了 HttpServletResponse 对象和 HttpServletRequest 对象的使用，其中，HttpServletResponse 对象封装了 HTTP 响应消息，并且提供了发送状态码、发送响应消息头、发送响应消息体的方法。使用这些方法可以解决中文输出乱码问题，实现网页的定时刷新跳转、请求重定向等。HttpServletRequest 对象封装了 HTTP 请求消息，也提供了获取请求行、获取请求消息头、获取请求参数的方法。使用这些方法可以解决请求参数的中文乱

码问题，并且使用 request 域对象传递数据的方法，还可以实现请求转发和请求包含。HttpServletResponse 和 HttpServletRequest 在 Web 开发中至关重要，要认真学习，牢固掌握。

习　　题

1. 简述请求转发与重定向的异同（至少写 3 点）。

2. 请编写一个类，该类能够实现访问完 App 应用下的 Servlet 后，5s 后跳转到 index.jsp 页面。

3. 编写一个系统登录界面，登录成功跳转到 welcome.jsp，否则提示错误并跳转到 login.jsp 页面。