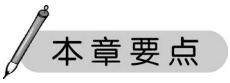
第5章

HTTP请求处理编程



- HTTP 请求内容
- HTTP 请求行
- HTTP 请求头
- HTTP 请求体
- Jakarta EE 请求对象类型
- Jakarta EE 请求对象功能和方法
- Jakarta EE 请求对象的生命周期
- 请求对象编程应用案例

Web 应用工作在请求/响应模式下，要访问 Web 文档，需要使用浏览器通过 URL 地址对该文档进行 HTTP 请求。当 Web 服务器接收并处理该请求后，向请求的客户端发送 HTTP 响应，客户端接收到 HTTP 响应后进行显示，用户即可看到请求文档的内容，主要是 HTML 网页及其他类型文档。

在动态 Web 应用中，用户需要将信息输入 Web 系统中，通常使用 HTML FORM 表单和表单元素，如文本框、单选按钮、复选框、文本域等。将客户端信息提交到 Web 服务器端，服务器接收客户提交的数据，按具体业务进行处理，完成业务功能，如验证用户是否合法、增加新员工等。

Jakarta EE 提供了 HTTP 请求对象，可以取得客户提交的数据。HTTP 请求对象保存客户在发送 HTTP 请求时传递给 Web 服务器的所有信息，并提供相应的方法取得不同的提交信息。

确定 HTTP 请求中包含的数据类型和内容，并使用 Jakarta EE 规范中的请求对象取得 HTTP 请求中包含的这些数据是开发动态 Web 的关键。

5.1 HTTP 请求内容

当客户端对 Web 文档进行 HTTP 请求时，在请求中不但包含请求协议（如 HTTP）、请

求 URL(如 localhost:8080/web01/login.jsp),还包含其他客户端提交的数据。因此,在 Web 应用编程中,开发人员需要了解客户端发送请求中包含的数据和类型。

5.1.1　HTTP 请求中包含信息

当在浏览器地址中输入 http://localhost:8080/web01/admin/login.jsp,对此 Web JSP 组件进行请求时,Web 服务器会收到请求中包含的如下信息:

```
GET /dumprequest HTTP/1.0
Host: djce.org.uk
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9.1.3) Gecko/20090824
Firefox/3.5.3 (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,en-US;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: GB2312,utf-8;q=0.7,*;q=0.7
Referer: http://www.google.cn/search?hl=zh-CN&source=hp&q=Http+request&btnG=
Google+%E6%90%9C%E7%B4%A2&aq=f&oq=Via: 1.1 cache3.dlut.edu.cn:3128 (squid/2.6.
STABLE18)
X-Forwarded-For: 210.30.108.201
Cache-Control: max-age=259200
Connection: keep-alive
```

以上请求信息按类别分为如下 3 部分。

(1) 请求行(Request Query)信息。请求行信息包括请求的协议 HTTP、请求的方式、请求的地址 URL、URI 等。

(2) 请求头(Request Header)信息。请求头信息主要包含请求指示信息,用于通知 Web 容器请求中信息的类型、请求方式、信息的大小、客户的 IP 地址等。根据这些信息,Web 组件可以采取不同的处理方式,实现对 HTTP 的请求处理。

(3) 请求体(Request body)信息。请求体信息中包含客户提交给服务器的数据,如表单提交中的数据、上传的文件等。

下面分别介绍每个组成部分的具体内容和意义。

5.1.2　请求行

请求行(Request Query)信息位于请求信息的第一行,包括请求的协议、请求的地址 URL、请求的方式等。请求行的案例代码如下:

```
GET https://www.baidu.com/content-search.xml HTTP/1.1
```

其中,GET 是请求方法,https://www.baidu com/是 URL 地址,HTTP/1.1 指定了协议版本。

不同的 HTTP 版本能够使用的请求方法也不同,具体介绍如下。

(1) HTTP 0.9:只有基本的文本 GET 功能。

(2) HTTP 1.0:具有完善的请求/响应模型,并将协议补充完整,定义了 GET、POST 和 HEAD 3 种请求方法。

(3) HTTP 1.1：在 HTTP 1.0 基础上进行更新，新增了 5 种请求方法：OPTIONS、PUT、DELETE、TRACE 和 CONNECT。

(4) HTTP 2.0：请求/响应首部的定义基本没有改变，只是所有首部键必须全部小写，而且请求行要独立为：method、:scheme、:host、:path 等键值对。

不同请求方式的意义如下。

(1) GET：请求指定的页面信息，并返回实体主体。

(2) POST：向指定资源提交数据请求处理(如提交表单或者上传文件)，数据被包含在请求体中。POST 请求可能会导致新的资源的建立和已有资源的修改。

(3) HEAD：类似于 GET 请求，只不过返回的响应中没有具体内容，用于获取报头。

(4) PUT：这种请求方式下，从客户端向服务器传送的数据取代指定的文档内容。

(5) DELETE：请求服务器删除指定的页面。

(6) OPTIONS：允许客户端查看服务器的性能。

(7) TRACE：回显服务器收到的请求，主要用于测试或诊断。

当 Web 客户向服务器发出 HTTP 请求时，请求头首先被发送到服务器端，服务器根据请求行的 URL 信息定位指定的文档，如果文档不存在，则服务器给客户端发送 404 错误信息，根据请求的方式，调用文档的指定的处理方法，如 Servlet 的 doGet、doPost、doPut、doDelete 等。

5.1.3 请求头

当 Web 客户向服务器发出 HTTP 请求时，发送请求行信息后，请求头信息被发送到服务器端，告知服务器此请求中包含的指示信息，服务器以便根据这些指示信息采取不同的处理。请求头中主要是客户端的一些基础信息，其中关键信息如下。

(1) accept：表示当前浏览器可以接受的文件类型。假设这里有 image/webp，表示当前浏览器可以支持 webp 格式的图片，那么当服务器给当前浏览器下发送 webp 类型图片时，可以更省流量。

(2) accept-encoding：表示当前浏览器可以接收的字符编码。如果服务器发送的响应数据不是浏览器可接收的字符编码，就会显示乱码。

(3) accept-language：表示当前客户端使用的语言，也包含客户所在的国家或地区。Web 服务器端应用要实现国际化，可根据此请求头信息给客户端发送对应的语言文本，如给中国大陆客户发送中文简体、给美国客户发送英语。

(4) Cookie：存储和用户相关的信息，每次用户在向服务器发送请求时会带上 Cookie。例如，用户在一个网站上登录之后，下次访问时就不用再登录，就是因为登录成功的 token 放在了 Cookie 中；另外，随着每次请求发送给服务器，服务器就会知道当前用户已登录。

(5) user-agent：表示浏览器的类型和版本信息。当服务器收到浏览器的请求后，通过该请求头知道浏览器的类型和版本，进而知道支持的语言版本(如 JavaScript、HTML、CSS 等)，开发者可以针对此类型浏览器编写对应版本的代码。

请求头中也可以包含客户端自定义的信息，通常前端框架(如 Vue、Angular 等)都可以在请求头中加入自己定义的 name 和 value 值。表 5-1 列出了 W3C 规范中规定的常用 HTTP 请求头标记和说明。

表 5-1　W3C 规范中规定的 HTTP 请求头标记和说明

头　标　记	说　　明	包含的值示例
User-Agent	客户端的类型(包含浏览器名称)	LII-Cello/1.0 libwww/2.5
Accept	浏览器可接收的 MIME 类型	各种标准的 MIME 类型
Accept-Charset	浏览器支持的字符编码	字符编码,如 ISO-8859-1
Accept-Encoding	浏览器知道如何解码的数据编码类型	x-compress; x-zip
Accept-Language	浏览器指定的语言	如 en:English
Connection	是否使用持续连接	Keep-Alive: 持续连接
Content-Length	使用 POST 方法提交时,传递数据的字节数	2352
Cookie	保存的 Cookie 对象	userid=9001
Host	主机和端口	192.168.100.3:8080

在 Jakarta EE Web 组件的 Servlet 和 JSP 中,可以使用请求对象的方法读取这些请求头的信息,进而进行相应的处理。5.2.4 小节将讲述请求头信息的取得方法。

5.1.4　请求体

每次 HTTP 请求时,在请求头后面会有一个空行,之后是请求中包含的提交数据,即请求体。请求体通常是表单元素中输入的数据,所有 Web 应用都需要客户输入数据。登录淘宝、京东的账号和密码信息,增加新产品的信息数据等,这些数据通常包含在请求体中。

不是所有的请求都有请求体,当为 GET 请求时,则没有请求体,因为请求数据直接附加在请求文档的 URL 地址中,请求体作为 URL 的一部分发送到 Web 服务器。例如,http://localhost:8080/web01/login.do?id=9001&pass=9001,这时请求体为空,因为提交数据直接在 URL 中,作为请求行的一部分传输到 Web 服务器,通过解析 URL 的 QueryString 部分就可以得到提交的参数数据。这种方式对提交的数据大小有限制,不同浏览器会有所不同,如 IE 为 2083 字节。GET 请求时,数据会出现在 URL 中,保密性差,因此在实际项目编程中要尽量避免在 URL 地址栏中传递请求数据。

POST 请求时,请求体数据单独打包为数据块,通过 Socket 直接发送到 Web 服务器端,数据不会在地址栏中出现,因此可以提交数据的类型和大小基本没有限制,可以包括二进制文件,可以实现文件上传功能。原则上 POST 请求对提交的数据没有大小限制,但为了应用需要,一般在编程时会对文件的大小加以限制。

Jakarta EE Web 组件(如 Serlvet、JSP、Filter、Listener 等)规范中都定义了如何取得请求体数据的方法,在 5.2 节中会详细说明这些方法和编程应用。

5.2　Jakarta EE 请求对象

为取得客户 HTTP 请求中包含的信息,Jakarta EE 的 Web 组件规范定义了请求对象接口规范,通过实现该接口的请求对象可以取得请求中包含的所有信息,包括请求行、请求头和请求体。

5.2.1 请求对象接口类型与生命周期

1. 请求对象接口类型

Jakarta EE 规范中，通用请求对象（不依赖请求协议的情况）要实现接口：

```
jakarta.servlet.ServletRequest
```

而本书重点介绍的是 HTTP 下工作的请求对象，要实现接口：

```
jakarta.servlet.http.HttpServletRequest
```

这两个接口的所有方法和属性可参阅 Eclipse Jakarta EE API 文档。

2. 请求对象生命周期

在 Java Web 组件开发中，不需要开发者自己创建 Servlet 或 JSP 使用的请求对象，它们由 Web 容器自动创建，并传递给 Servlet 和 JSP 的服务方法 doGet、doPost、doPut、doDelete 及 doHead 等。在这些 HTTP 请求处理方法中可以直接使用请求对象，调用其方法，取得客户端提交的数据。

（1）创建请求对象。每次 Web 服务器接收到 HTTP 请求时，会自动创建实现 HttpServletRequest 接口的对象。具体的请求对象实现类由 Jakarta EE 服务器厂家实现，不同的服务器产品（如 Tomcat、GlassFish、WebLogic 等）实现请求对象接口的实现类不一定相同，但开发者不需要了解具体的请求对象的实现类型，只需掌握请求对象接口的方法即可。创建请求对象后，Web 服务器将请求行、请求头和请求体信息存入请求对象，并自动把请求对象传递给请求的 Web 组件，如 Servlet、JSP 等。这些 Web 组件可以通过请求对象的方法取得这些请求信息，即客户端用户提交的数据。

（2）销毁请求对象。当 Web 服务器处理 HTTP 请求，向客户端发送 HTTP 响应结束后，会自动销毁请求对象，保存在请求对象中的数据随即丢失。当下次请求时新的请求对象又会创建，重新开始请求对象新的生命周期。

5.2.2 请求对象的功能与方法

Jakarta EE 提供的 HttpServletRequest 请求对象用于取得 HTTP 请求中包含的请求行、请求头和请求体的数据信息。HttpServletRequest 接口定义的方法分类如下。

（1）取得请求行的数据。

（2）取得请求头信息。

（3）取得请求体中包含的提交参数数据，包含表单元素或地址栏 URL 的参数。

（4）取得服务器端的相关信息，如服务器的 IP 等。

（5）取得请求对象存储的属性信息。

请求对象除了可以取得客户端的各种信息外，还提供了作为传递数据的容器的方法，用于在 Web 组件间传递数据。该功能在 Web 开发中使用得非常多，在 7.2.3 小节中会详细讲解请求对象的此项功能和方法。

5.2.3 取得请求行方法

请求对象接口 HttpServletRequest 提供了如下方法，用于取得请求行中包含的数据。

(1) String getProtocol()：取得使用的请求协议。

(2) String getMethod()：取得请求的方式，返回字符串类型的 GET、POST、PUT、DELETE 等。

(3) StringBuffer getRequestURL()：取得请求的 URL 地址。需要注意的是，其返回类型不是 String，而是 StringBuffer，需要调用其 toString()方法将其转换为 String 类型再显示。

(4) String getRequestURI()：取得请求的 URI 地址。URI 地址是 Web 站点内的地址，从 Web 应用的起始站点名开始，假如 Web 的站点起始路径为/jakartaweb05，Web 文档 JSP 目录是/employee，文件名是 list.jsp，则 URI 地址是/jakartaweb05/employee/list.jsp。而 URL 地址是包含协议、IP 地址和端口的全地址，上面 JSP 文件的 URL 地址如下：

```
http://localhost:8080/jakartaweb05/employee/list.jsp
```

从 URL 地址可以看出，URL 包含 URI，URI 是 URL 的一部分，即：

```
URL = 协议://IP:端口/URI
```

测试取得请求行的 Servlet 代码如程序 5-1 所示。

程序 5-1　RequestQueryGetting.java 取得请求行的测试 Servlet 类代码。

```
package com.city.oa.servlet;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebInitParam;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
/**
 * 取得请求行的测试 Servlet
 */
@WebServlet(urlPatterns = { "/requestquery/get.do" })
public class RequestQueryGetting extends HttpServlet {
    private static final long serialVersionUID = 1L;
    /**
     * GET 请求处理
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        //取得请求协议
        String protocol = request.getProtocol();
        //取得请求方式
        String method = request.getMethod();
        //取得请求地址 URL
        String url = request.getRequestURL().toString();
        //取得请求地址 URI
        String uri = request.getRequestURI();
        //发送响应数据
        response.setContentType("text/html");
```

```
            response.setCharacterEncoding("UTF-8");
            PrintWriter out = response.getWriter();
            out.println("<h1>取得请求行信息</h1>");
            out.println("<hr/>");
            out.println("请求协议:" + protocol + "<br/>");
            out.println("请求方式:" + method + "<br/>");
            out.println("请求URL地址:" + url + "<br/>");
            out.println("请求URI地址:" + uri + "");
            out.println("<hr/>");
            out.flush();
            out.close();
        }
        /**
         * POST请求处理
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
            doGet(request, response);
        }
    }
```

运行此 Servlet 代码,结果如图 5-1 所示。

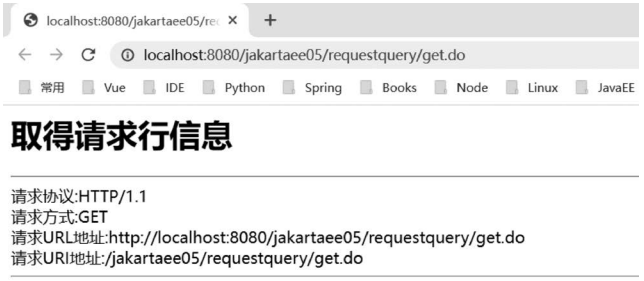


图 5-1 取得请求行的 Servlet 代码运行结果

5.2.4 取得请求头方法

5.1.3 小节介绍了请求中包含的主要请求头信息,HttpServletRequest 接口提供了如下方法用于取得不同类型的请求头中的数据。通常请求头中的数据类型主要有 String、int、Date 等。

(1) String getHeader(String name):取得指定请求头字符串类型的内容。例如,在 Servlet 的 doGet 或 doPost 方法中取得客户端浏览器类型的代码如下:

```
String browser = request.getHeader("User-Agent");
```

(2) int getIntHeader(String name):取得整数类型的指定请求头内容。如当 HTTP 请求中包含请求体数据时,通常会在请求中包含名称为 Content-Length 的请求头,表示请求体的长度,服务器端可以根据该请求头的值得知请求体数据的字节长度。取得请求体长度的示例代码如下:

```
int size = request.getIntHeader("Content-Length");
```

请求头 Content-Length 中包含的请求体长度为 int 类型。该方法在编程文件上传类型应用中特别有用。

(3) long getDateHeader(String name)：此方法取得日期类型的指定请求头的内容。其返回的类型不是 Date 型，而是 long 型(表示从 1970 年 1 月 1 日 0 点开始计时的毫秒数)，根据此 long 值计算出 Date 类型日期。如下代码为取得 If-Modified-Since 的请求头的值，表达请求文档的最近修改日期：

```
long datetime = request.getDataHeader("If - Modified - Since");
```

上述代码取得请求文档的最后修改日期的毫秒数。如果想要取得具体的日期，则使用 java.util.Date 的构造方法传递此 long 类型的值即可。其示例代码如下：

```
Date modifyDate = new Date(datetime);
```

(4) Enumeration getHeaderNames()：此方法取得所有请求头的 name 的列表，以枚举类型返回。可以使用遍历枚举类型的方法取得所有的请求头，包括 name 和 value。

程序 5-2 的代码展示了取得并输出所有请求头名称及每个请求头 name 对应的值的 Servlet 编程。

程序 5-2　RequestHeaderGetting.java 取得请求头的 Servlet 类代码。

```
package com.city.oa.servlet;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
/**
 * 取得请求头的 Servlet
 */
@WebServlet(urlPatterns = { "/requestheaders/get.do" })
public class RequestHeaderGetting extends HttpServlet {
    private static final long serialVersionUID = 1L;
    /**
     * GET 请求处理方法
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF - 8");
        PrintWriter out = response.getWriter();
        out.println("<h1>取得请求头信息</h1>");
        out.println("<hr/>");
        out.println("请求头 User - Agent:" + request.getHeader("User - Agent") + "<br/>");
        out.println("请求头 Host:" + request.getHeader("Host") + "<br/>");
        out.println("客户 IP 地址:" + request.getRemoteAddr() + "<br/>");
        out.println("<hr/>");
        out.println("<h1>所有请求头遍历</h1>");
        Enumeration<String> headers = request.getHeaderNames();
```

```
            for(;headers.hasMoreElements();) {
                String headerName = headers.nextElement();
                out.println(headerName + " = " + request.getHeader(headerName) + "<br/>");
            }
            out.println("<hr/>");
            out.flush();
            out.close();
        }
        /**
         * POST处理方法
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
            doGet(request, response);
        }
    }
```

上述代码首先使用 request 请求对象的方法取得了常用的请求头，然后使用遍历方法遍历所有请求头的名称和值。将上述 Servlet 代码部署到 Tomcat 上运行，使用浏览器请求该 Servlet 取得图 5-2 所示的请求头信息。

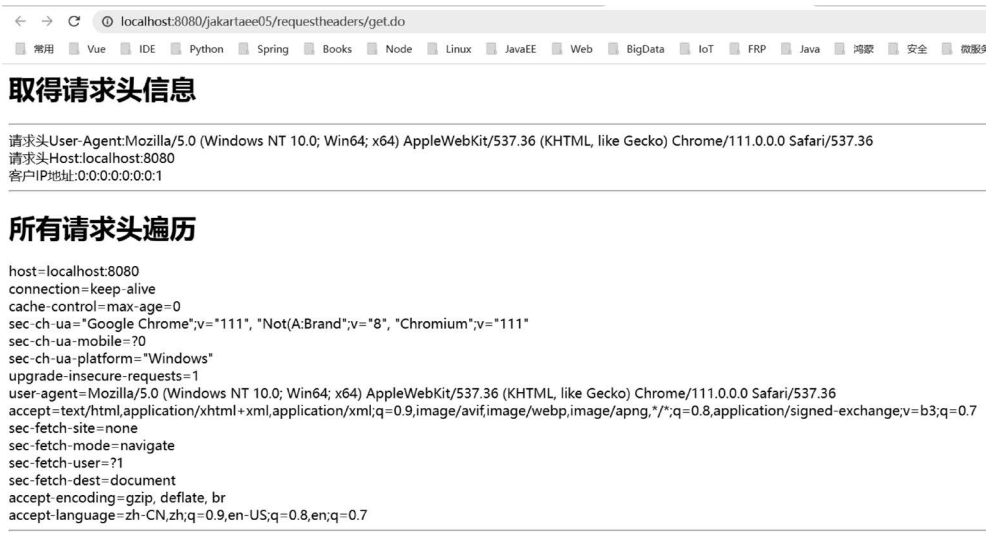


图 5-2　Servlet 使用请求对象取得的请求头信息

5.2.5　取得请求体方法

在 Web 开发中，用户通过表单输入将客户端数据提交到服务器端，这些数据被 Web 服务器自动保存到请求对象中，Web 组件 Servlet 和 JSP 可以通过请求对象取得提交的数据。HttpServletRequest 请求对象提供了如下方法，用于取得客户提交的数据。

(1) String getParameter(String name)：取得指定名称的请求体数据。此方法用于单个数据的参数，所有取得的参数值都是 String 类型，开发者需要根据业务需求，将其转换为对应的数据类型，如 int、double、Date 等。

参数 name 为 FORM 表单元素的 name 属性或 URL 参数名称，如：

```
产品名称:< input type = "text" name = "productName" />
productSearch.do?productName = Acer
```

如下代码可取得以上参数名为 productName 的数据：

```
String productName = request.getParameter("productName");
```

(2) String[] getParameterValues(String name)：取得指定参数名称的数据数组，用于多值参数的情况，如复选框、复选列表等。如下示例代码中展示复选框形式的爱好选择数据：

```
爱好:< input type = "checkbox" name = "behave" value = "旅游" />旅游
     < input type = "checkbox" name = "behave" value = "读书" />读书
     < input type = "checkbox" name = "behave" value = "体育" />体育
```

取得上面复选框选中爱好数据的示例代码如下：

```
String[] behaves = request.getParameterValues("behave");
for(int i = 0;i < behaves.length;i++){
    out.println(behaves[i]);
}
```

注意：此数组不需要事先确定大小，由 Web 容器自动根据参数名对应值的个数确定数组的容量大小。

(3) Enumeration getParameterNames()：取得所有请求参数的名称，返回遍历器类型。使用遍历器的方法可以取得所有请求的参数名。如下示例代码为遍历所有请求参数名：

```
for (Enumeration enum = request. getParameterNames(); enum.hasMoreElements();)
{   String paramName = (String)enum.nextElement();
    System.out.println("Name = " + paramName);
}
```

(4) Map getParameterMap()：取得所有请求参数名和值，包装在一个 Map 对象中，可以使用该对象同时取得所有参数名和参数值。如下示例代码取得所有请求参数名和参数值：

```
Map params = request.getParameterMap();
Set names = params.keySet();
for(Object o:names) {
    String paramName = (String)o;
    out.print(paramName + " = " + params.get(paramName) + "< br/>");
}
```

(5) ServletInputStream getInputStream() throws IOException：取得客户提交数据的输入流。当使用 getParamerer 方法后，就无法使用 getInputStream 方法，反之亦然，二者只能使用其一。当用户使用 POST 方式提交包含文件上传的数据时，请求数据以二进制编码方式提交到服务器，此时 Servlet 无法使用之前的 getParameter 方法取得请求数据，只能取得纯文本数据。要取得包含文本和二进制编码的请求数据，只能使用 getInputStream 方法以二进制方式取得请求数据，再对此数据进行解析，从而分离出文本数据和上传的文件。如果开发者自己编程解析将非常复杂，因此经常使用成熟的框架技术来处理这种有文件上传的请求。目前市场上已经存在多种第三方框架来实现上传文件处理，如 Apache 的

Common upload 组件、JSP Smartupload 等。如下代码示例为有文件上传的表单 HTML：

```
<form action="addEmp.do" method="POST" enctype="multipart/form-data"/>
    姓名:<input type="text" name="name" /><br/>
    照片:<input type="file" name="photo" /><br/>
<input type="submit" value="提交"/>
</form>
```

使用第三方框架技术，如 Common Upload，可以非常方便地取得表单中包含的姓名和照片文件，具体可参阅相应的框架文档资料。

(6) Part getPart(String name)：从 Servlet 3.0 开始，请求对象提供了取得上传文件的方法 getPart。在此之前要取得上传文件，必须使用 getInputStream 方法取得所有请求数据的输入流，开发者自己解析此输入流取得上传的文件，或者使用第三方框架。现在直接使用 getPart 方法就可以取得表单中的上传文件，非常方便。其中 getPart 方法的参数是提交数据项的 name，返回 Part 类型的数据，其接口类型为 jakarta.servlet.http.Part，该接口的具体实现类由使用的服务器(如 Tomcat)实现。

Part 接口提供了如下方法用于处理取得的上传文件。

① InputStream getInputStream()：取得其输入流对象，进而可以通过流的编程取得上传文件。

② long getSize()：取得上传文件的大小(字节数)。

③ String getContentType()：取得上传文件的类型，为 MIME 类型，将在第 6 章中详细介绍。

④ String getSubmittedFileName()：取得文件的名称。

⑤ void write(String fileName)：将上传的文件写入指定的文件中。当需要保存上传文件到指定目录时，此方法特别有用。

⑥ void delete()：将此上传文件删除。当服务器接收到上传文件时，会在服务器指定的临时目录中保存此文件。使用此方法可以立即删除此文件。

5.2.6 请求对象取得常用请求头数据的便捷方法

对于取得请求行和请求头数据，HttpServletRequest 请求对象除提供通用的方法以外，还提供了专门的便捷方法来取得请求行和请求头信息，如客户端信息、请求方式、客户端 IP 地址等。下面是请求对象提供的取得专门信息的便捷方法。

(1) String getRemoteHost()：取得请求客户的主机名。其与通用方法 getHeader("Host")等价。

(2) String getRemoteAddr()：取得请求客户端的 IP 地址。其没有专门的等价通用方法，需要先取得 Host 请求头，再解析客户的 IP 地址。

(3) int getRemotePort()：取得请求客户的端口号。其没有等价的通用方法，也需要先取得 Host 请求头，再解析客户的端口。

(4) String getProtocol()：直接取得请求行中的协议。

(5) String getContentType()：取得请求体的内容类型，以 MIME 表达。其与通用的取得请求头方法 getHeader("Content-Type")等价。

(6) int getContentLength()：取得请求体的长度(字节数)，当处理有文件上传请求时特别有用。其与通用的取得请求头的方法 getIntHeader("Content-Length")等价。

(7) String getMethod()：取得请求的方式，返回 GET、POST、PUT、DELETE 等信息。

5.2.7　取得服务器端信息

通过 HttpServletRequest 请求对象还可以取得服务器的信息，如服务器名称、接收端口等。如下方法用于服务器端信息的取得。

(1) String getServerName()：取得服务器的 HOST，一般为 IP 地址。

(2) int getServerPort()：取得服务器接收端口。实际编程中很少使用此方法，因为服务器的地址和端口是开发人员已知的，而客户端的地址和端口是未知的。因此，取得客户端的 IP 是必要的，而且经常使用。例如，所有的聊天类、BBS 公告板、贴吧等应用都需要取得客户的 IP 地址，以便追踪客户的上网地址。

如下代码演示了取得服务器的 IP 地址和端口号：

```
out.println("服务器名称:" + request.getServerName() + "<br/>");
out.println("服务器端口:" + request.getServerPort() + "<br/>");
```

上述代码将显示图 5-3 所示的内容。

服务器名称:localhost

服务器端口:8080

图 5-3　取得服务器名称和端口的显示信息

5.3　取得客户端 HTML 表单提交数据案例

本案例使用 HttpServeltRequest 请求对象取得客户端表单提交的业务数据编程。在实际项目开发中，经常需要向服务器提交数据，如用户注册、产品增加等类似应用都非常普遍。

5.3.1　业务描述

在线购物网站中要求有客户注册功能，只有已经注册且登录的用户才能进行购物结算和发送订单。本案例即实现用户注册和处理功能，并且在 Servlet 中直接完成数据库的处理。这样做的目的是演示 Servlet 能完成的功能和编程，企业实际应用开发中并不会使用 Servlet 直接进行数据库的操作，而是使用 MVC 模式，通过持久化 DAO 层进行数据库的操作。

本案例使用 JSP 页面完成用户注册界面的编程，用户注册的处理使用 Servlet 完成。其中，用户注册页面如图 5-4 所示。

用户在注册页面(/customer/register.jsp)输入注册信息，提交给注册处理 Servlet (CustomerRegisterServlet)，该 Servlet 将取得的注册信息写入数据库表中。Servlet 处理成功后，跳转到注册处理成功显示页面(/customer/registersuccess.jsp)，显示注册成功消息，如图 5-5 所示。

图 5-4　用户注册页面

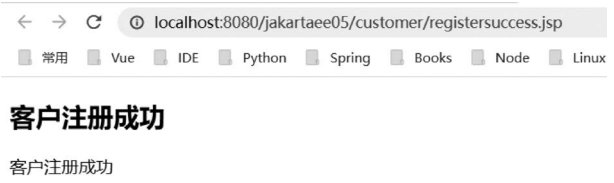


图 5-5　客户注册成功显示页面

5.3.2　案例编程

本案例使用一个 JSP 注册页面和一个处理 Servlet，使用 STS 和 Tomcat 10.1.17 进行开发和部署。

1. 创建 Maven Web 项目：oaweb2024

创建步骤参见第 3 章的 Eclipse 创建 Maven 项目的流程，创建的项目目录结构如图 5-6 所示。

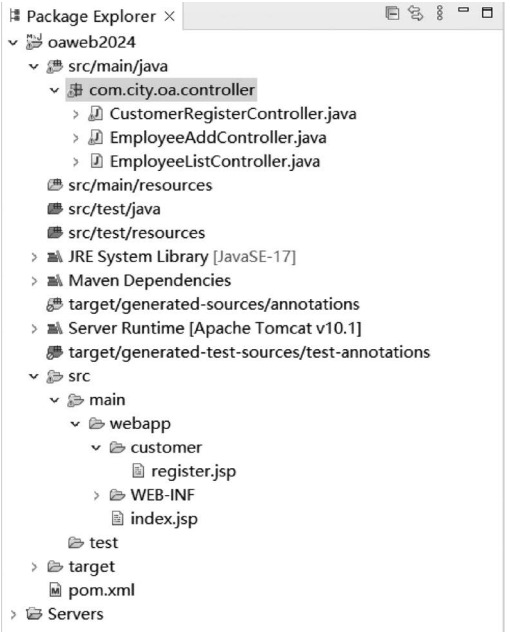


图 5-6　案例 Maven Web 项目的目录结构

2. 客户注册页面的 JSP 编程

客户注册页面采用 JSP 实现，显示一个简单的客户注册表单。客户输入表单中对应的注册信息，单击“提交”按钮，会请求注册处理 Servlet。该 JSP 页面的代码如程序 5-3 所示。

程序 5-3　register.jsp 客户注册页面的 JSP 代码。

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

```
<head>
<meta charset="UTF-8">
<title>网上商城系统</title>
</head>
<body>
<h1>客户注册</h1>
<form method="post" action="add.do">
登录账号:<input type="text" name="id" /><br/>
登录密码:<input type="password" name="password"/><br/>
确认密码:<input type="password" name="repassword"/><br/>
用户名称:<input type="text" name="name"/><br/>
<input type="submit" value="提交" />
</form>
</body>
</html>
```

使用表单和表单元素提交数据推荐使用 POST 方式，即设置<form>标记的属性 method="post"。

3. 客户注册处理 Servlet 编程

客户注册处理 Servlet 取得注册信息，并将注册信息写入数据库中。如果出现异常，将自动重定向到用户注册页面。注册处理 Sevlet 代码如程序 5-4 所示。

程序 5-4　CustomerRegisterController.java 客户注册处理 Servlet 代码。

```
package com.city.oa.controller;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
//用户注册处理 Servlet
@WebServlet(
        urlPatterns = { "/customer/register.do" },
        initParams = {
            @WebInitParam(name = "driver", value = "com.mysql.cj.jdbc.Driver"),
            @WebInitParam(name = "url", value = "jdbc:mysql://localhost:3319/cityoa"),
            @WebInitParam(name = "user", value = "root"),
            @WebInitParam(name = "password", value = "root1234")
})
public class CustomerRegisterController extends HttpServlet
{
    //定义数据库连接对象
    private Connection cn = null;
     //数据库驱动器
    private String driverName = null;
     //数据库地址 URL
    private String url = null;
    //初始化方法,取得数据库连接对象
    public void init(ServletConfig config) throws ServletException
    {
```

```
        super.init(config);
        driverName = config.getInitParameter("driverName");
        url = config.getInitParameter("url");
        try{
            Class.forName(driverName);
            cn = DriverManager.getConnection(url);
        } catch(Exception e){
            System.out.println("取得数据库连接错误:" + e.getMessage());
        }
    }
    //处理 GET 请求方法
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        //取得用户注册表单提交的数据
        String userid = request.getParameter("userid");
        String password = request.getParameter("password");
        String repassword = request.getParameter("repassword");
        String name = request.getParameter("name");
        //判断登录账号为空,自动跳转到注册页面
        if(userid == null||userid.trim().length() == 0) {
            response.sendRedirect("register.jsp");
        }
        //如果登录密码为空,则自动跳转到注册页面
        if(password == null||password.trim().length() == 0){
            response.sendRedirect("register.jsp");
        }
        //如果确认登录密码为空,则自动跳转到注册页面
        if(repassword == null||repassword.trim().length() == 0) {
            response.sendRedirect("register.jsp");
        }
        //如果密码和确认密码不符,则自动跳转到注册页面
        if(!password.equals(repassword)) {
            response.sendRedirect("register.jsp");
        }
        //将姓名进行汉字乱码处理
        if(name!= null&&name.trim().length()> 0){
      name = new String(name.getBytes("ISO - 8859 - 1")); }
         //增加新用户处理
         String sql = "insert into USERINFO (USERID,PASSWORD,NAME) values (?,?,?)";
         try{
            PreparedStatement ps = cn.prepareStatement(sql);
            ps.setString(1, userid);
            ps.setString(2, password);
            ps.setString(3, name);
            ps.executeUpdate();
            ps.close();
            //处理结束后,跳转到注册成功提示页面
            response.sendRedirect("registersuccess.jsp");
         } catch(Exception e){
            System.out.println("错误:" + e.getMessage());
            response.sendRedirect("register.jsp");
        }
    }
```

```
    //处理 POST 请求方法
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request,response);
    }
    //销毁方法
    public void destroy() {
        super.destroy();
        try{
            cn.close();
        } catch(Exception e) {
            System.out.println("关闭数据库错误:" + e.getMessage());
        }
    }
}
```

在注册处理 Servlet 中取得注册页面提交的用户信息，若这些注册信息不为空，则将其插入用户表中。此 Servlet 使用注解类配置方式，不需要在 web.xml 文件中编写配置代码。

4. 注册成功显示页面编程

当客户注册处理 Servlet 完成客户注册的功能后，自动跳转到注册成功显示页面，提醒客户注册成功。其 JSP 页面代码如程序 5-5 所示。

程序 5-5　/customer/registersuccess.jsp 客户注册成功显示页面。

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>网上商城系统</title>
</head>
<body>
<h2>客户注册成功</h2>
</body>
</html>
```

启动项目，部署到 Tomcat 服务器，使用浏览器请求客户注册页面，填写注册信息，单击“提交”按钮，完成注册处理后，即显示注册成功页面。

5.4　取得客户端信息并验证案例

有的 Web 应用需要限制客户的访问，只允许部分 IP 地址的客户访问指定的页面或控制组件；有时封杀某些 IP 的客户访问，因为这些 IP 已经被记录在黑名单中。本案例使用请求对象取得客户的 IP 地址，检查 IP 是否在被封杀之列，从而决定此客户是否可以继续访问 Web 应用。

5.4.1　业务描述

编写 Servlet，取得客户端的 IP 地址，并将此 IP 与数据库表中保存的封杀 IP 进行比较。

如果此 IP 在封杀之列，则跳转到错误信息显示页面，阻止客户进一步的访问；如果 IP 不在封杀之列，则允许跳转到主页。

5.4.2 案例编程

根据案例的功能要求，设计如下数据库表和 Web 组件。

1. 设计 IP 封杀数据表

IP 封杀数据表保存被封杀的 IP 列表，表结构设计如表 5-2 所示。本案例使用 MySQL 数据库，在数据库 cityoa 下创建此 IP 封杀记录表。

表 5-2 IP 封杀数据表（表名称 LimitIP）结构设计

字 段 名	类 型	约 束	说 明
IPNO	Int	主键	编号
IP	Varchar(50)	非空	IP 地址

2. 监测 IP 地址是否被封杀的 Servlet 编程

此 Servlet 首先取得客户的 IP 地址，再连接数据库，判断 IP 是否在封杀列表中。如果 IP 在封杀之列，则自动跳转到错误信息显示页面；否则自动跳转到系统的主页面。监测客户 IP 是否被封杀的 Servlet 代码如程序 5-6 所示。

程序 5-6 IPCheckController.java IP 检查 Servlet 类代码。

```
package com.city.oa.controller;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
//客户 IP 检查 Servlet
@WebServlet(
        urlPatterns = { "/client/ipcheck.do" },
        initParams = {
            @WebInitParam(name = "driver", value = "com.mysql.cj.jdbc.Driver"),
            @WebInitParam(name = "url", value = "jdbc:mysql://localhost:3319/cityoa"),
            @WebInitParam(name = "user", value = "root"),
            @WebInitParam(name = "password", value = "root1234")
})
public class ClientIPCheckAction extends HttpServlet
{
    //定义数据库连接对象
    private Connection cn = null;
     //数据库驱动器
    private String driverName = null;
     //数据库地址 URL
```

```
    private String url = null;
    public void init(ServletConfig config) throws ServletException
    {
        super.init(config);
        driverName = config.getInitParameter("driverName");
        url = config.getInitParameter("url");
        try{
            Class.forName(driverName);
            cn = DriverManager.getConnection(url);
        } catch(Exception e){
            System.out.println("取得数据库连接错误:" + e.getMessage());
        }
    }
    //GET 请求
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        boolean isLocked = false;
        String ip = request.getRemoteAddr();
        String sql = "select * from LimitIP where IP = ?";
        try{
            PreparedStatement ps = cn.prepareStatement(sql);
            ps.setString(1, ip);
            ResultSet rs = ps.executeQuery();
            if(rs.next())
            {
                isLocked = true;            //如果 IP 在数据表中,则表示被封杀
            }
            rs.close();
            ps.close();
            if(isLocked)
            {
                //如果 IP 被封杀,自动跳转到封杀信息页面
                response.sendRedirect("ipLock.jsp");
            }
            else
            {
                //如果 IP 允许访问,则可以跳转到主页面
                response.sendRedirect("main.jsp");
            }
        } catch(Exception e) {
            System.out.println("检查 IP 是否封杀错误:" + e.getMessage());
            response.sendRedirect("errorInfo.jsp");
        }
    }
    //POST 请求处理
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        doGet(request,response);
    }
    //销毁方法
    public void destroy()
    {
```

```
            super.destroy();
            try{
                cn.close();
            } catch(Exception e){
                System.out.println("关闭数据库错误:" + e.getMessage());
            }
        }
    }
```

3. 错误信息显示页面编程

当客户 IP 在被封杀之列时,此页面将被显示。该页面 JSP 代码如程序 5-7 所示。

程序 5-7 errorInfo.jsp 客户 IP 被封杀信息显示页面。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <title>网上商城</title>
  </head>
  <body>
    <h1>错误信息</h1>
    <hr/>
    对不起,您无法访问网上商城.<br/><br/>
    因为您的 IP 已经被封杀!
    <hr/>
  </body>
</html>
```

此错误页面只显示简单的错误信息,关键用于配合 Servlet 进行演示。实际应用项目中,错误信息页面应设计得与应用页面总体布局相符。

4. 系统主页面编程

当 IP 通过检查之后,跳转到系统的主页面。该页面通常用于功能导航,其代码如程序 5-8 所示。

程序 5-8 main.jsp 系统主页面代码。

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-
8"%>
<!DOCTYPE html>
<html>
  <head>
    <title>网上商城</title>
  </head>
  <body>
    <h1>网上商城</h1>
    <hr/>
       欢迎您访问网上商城.<br/>
     <a href="product/main.jsp">产品检索</a>
          <a href="purchase/main.jsp"购物车</a>
     <a href="order/main.jsp">订单管理</a>
    <hr/>
  </body>
</html>
```

目前的主页只是一个简单页面，并没有实质功能，仅用于演示 Servlet 和请求对象的功能，以及简单的功能导航。

5.4.3　案例部署和测试

将开发完毕的 Web 项目部署到 Tomcat 服务器上，如果客户端 IP 地址与数据表的 IP 地址相同，则 IP 封杀检查 Servlet 将自动跳转到 IP 封杀信息页面，如图 5-7 所示。

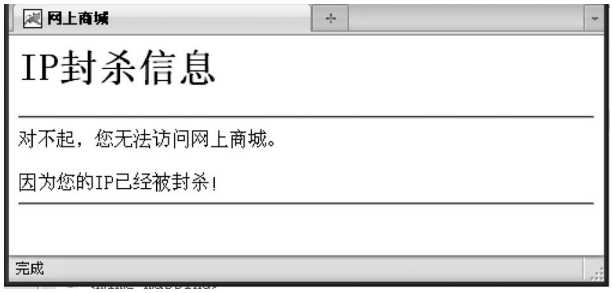


图 5-7　IP 封杀信息页面

修改数据表中的 IP 地址，使之与客户端的 IP 不同，再次请求此 Servlet，则自动跳转到网上商城的主页 main.jsp。

通过此案例，读者可以知道如何通过请求对象取得客户端的信息，并使用这些客户端信息进行特定的业务处理。

5.5　文件上传请求处理案例

任何项目都需要提交上传文件给服务器，如增加员工时上传员工的照片、网上商城增加产品时上传产品图片、新闻网站增加新闻时上传新闻图片等，所有这些都涉及文件的上传处理。有时需要将上传文件写入数据库表，有时则需要将图片保存到服务器端的指定目录中。

本节将详细介绍使用 Servlet 的请求对象如何取得表单中上传的图片文件，以及在取得上传文件后，如何将其写入数据库或保存到服务器的文件系统的指定目录中。

5.5.1　业务描述

在编写员工增加的 JSP 页面时，需要上传员工的照片。表单中包含文件域表单元素，文件域专门用于上传文件。输入员工信息后，单击“提交”按钮，请求员工增加处理的 Servlet，实现增加员工的处理功能。

Servlet 使用请求对象的方法，取得员工增加页面提交的数据，包括上传的图片文件，并将员工数据增加到员工表 oa_employee 中。员工表包含一个保存图片的二进制字段，其类型是 longblob，此类型专门用于保存文件的原始格式，最大能存储 2GB 的文件。员工表 oa_employee 的字段结构如图 5-8 所示。

5.5.2　案例编程

本案例包含一个员工增加的 JSP 页面，以及员工增加处理的 Servlet。其中，员工增加

图 5-8　员工表 oa_employee 的字段结构

JSP 页面负责增加界面的显示，接收用户输入的新员工数据；Servlet 负责取得员工增加 JSP 页面提交的数据，连接数据库，执行增加 SQL 语句，将新员工数据写入员工表 oa_employee，同时将取得的员工照片保存到 d:/temp 目录下。

1. 员工增加 JSP 页面编程

员工增加 JSP 页面主要负责增加表单的显示，使用 HTML 的表单和表单元素即可。为简化案例的编程，该 JSP 页面并没有使用使其美观的框架（如 Bootstrap 等）对其进行美化，而实际项目一定会使用特定的 UI 框架对操作界面进行美观处理。员工增加 JSP 页面的代码如程序 5-9 所示。

程序 5-9　/employee/add.jsp 员工增加 JSP 页面。

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h1>员工增加</h1>
<form method="post" action="add.do" enctype="multipart/form-data">
账号:<input type="text" name="id" /><br/>
密码:<input type="password" name="password"/><br/>
姓名:<input type="text" name="name"/><br/>
年龄:<input type="text" name="age"/><br/>
照片:<input type="file" name="photo"/><br/>
<input type="submit" value="提交" />
</form>
</body>
</html>
```

需要注意的是，由于该JSP页面需要文件上传功能，因此对表单< form >的属性有特殊要求。首先，必须使用POST请求，GET请求无法提交上传文件数据，因此设置method＝"post"。其次，由于表单中有文件域元素，请求时需要传输文件，因此必须使用混合表单数据模式，不能使用默认的纯文本模式数据传输，因此要增加属性enctype＝"multipart/form-data"，其中multipart/form-data表示请求体的数据是文本和二进制混合的数据。如果不指定enctype属性，则其默认值是x-www-form-urlencoded格式，即HTML文本格式，只能传输纯文本数据给服务器。

2. 员工增加处理Servlet编程

为处理有文件上传的Servlet，必须进行特殊的配置，即启用Servlet引擎的文件上传功能。在Servlet 3.0之前无法直接处理文件上传，只能由开发者自己编程，或者使用第三方文件上传框架(如Apache Common Upload、JSP Smart Upload等)取得和处理上传的文件。从Servlet 3.0开始，Web组件内置了文件上传处理机制，增加了取得上传文件的类型jakarta.servlet.http.Part；请求对象增加了getPart方法，用于取得Part类型的上传文件。5.2.5小节已经介绍了Part的常用方法，在此不再赘述。

在Servlet配置中，除了在类级别上使用常规的@WebServlet注解类对Servlet的请求地址和初始参数进行配置外，还需要使用注解类@MultipartConfig对Servlet类进行配置，该注解类用于启用Servlet 3.0内置的文件上传处理机制。如果不使用@MultipartConfig，则Servlet不能处理文件上传，只能处理提交的文本数据。员工增加处理的Servlet实现代码如程序5-10所示。

程序5-10　EmployeeAddController.java员工增加处理Servlet。

```
package com.city.oa.controller;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.MultipartConfig;
import jakarta.servlet.annotation.WebInitParam;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.Part;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
/**
 * 员工增加处理Servlet
 */
@WebServlet(
        urlPatterns = { "/employee/add.do" },
        initParams = {
            @WebInitParam(name = "driver", value = "com.mysql.cj.jdbc.Driver"),
            @WebInitParam(name = "url", value = "jdbc:mysql://localhost:3319/cityoa"),
            @WebInitParam(name = "user", value = "root"),
            @WebInitParam(name = "password", value = "root1234")
```

```
})
@MultipartConfig
public class EmployeeAddController extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private Connection cn = null;
    /**
     * 初始化方法,取得配置的初始化参数
     */
    public void init(ServletConfig config) throws ServletException {
        String drvier = config.getInitParameter("driver");
        String url = config.getInitParameter("url");
        String user = config.getInitParameter("user");
        String password = config.getInitParameter("password");
        try {
            Class.forName(drvier);
            cn = DriverManager.getConnection(url,user,password);
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    /**
     * Servlet 销毁方法,关闭数据库连接
     */
    public void destroy() {
        try {
            cn.close();
        }catch(Exception e) {
            e.printStackTrace();
        }
    }
    //处理 GET 请求的方法,取得客户端提交的员工数据,包括上传的员工照片
    //将数据增加到数据库表中,并保存员工照片到 d:/temp 目录
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

        //取得员工增加表单提交的数据
        String id = request.getParameter("id");
        String password = request.getParameter("password");
        String name = request.getParameter("name");
        String sage = request.getParameter("age");
     //取得员工照片
        Part photo = request.getPart("photo");
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<h1>员工增加处理 Servlet</h1>");
         String sql = "insert into oa_employee (EMPID, EMPPASSWORD, EMPNAME, AGE, PHOTO,
PHOTOCONTENTTYPE) values (?,?,?,?,?,?)";
        try {
            int age = Integer.parseInt(sage); //转换年龄类型为 int
            //将上传的图片保存到 d:/temp 目录下
            if(photo!= null&&photo.getSize()>0) {
                PreparedStatement ps = cn.prepareStatement(sql);
```

```
                    ps.setString(1, id);
                    ps.setString(2,password);
                    ps.setString(3,name);
                    ps.setInt(4, age);
                    //取得上传文件的输入流,写入 SQL 语句
                    ps.setBinaryStream(5, photo.getInputStream(),photo.getInputStream().available());
                    ps.setString(6, photo.getContentType());
                    ps.executeUpdate(); //执行 SQL 语句
                    ps.close();
                    //上传文件保存到指定目录
                    photo.write("d:/temp/" + photo.getSubmittedFileName());
                }
            }
            catch(Exception e) {
                out.println("增加员工异常:" + e.getLocalizedMessage());
            }
            out.flush();
            out.close();
        }
        // POST 请求处理,直接调用 GET 方法处理,让 doGet 方法处理请求数据
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
            doGet(request, response);
        }
    }
```

Servlet 代码在类级别上使用了@MultipartConfig,这是需要特别注意的。之前的 Servlet 都没有此注解类,今后在开发有文件上传功能的 Servlet 时一定要增加此注解类。

在处理文件上传的代码中,使用 Part 的 getInputStream 方法取得上传文件的字节输入流,将其 set 到数据库表的 PHOTO 字段。JDBC 在执行 insert 语句时,自动读取字节流中的数据,写入该字段中。

Servlet 处理代码最后使用"photo. write("d:/temp/"+photo. getSubmittedFileName())"将上传的图片写入指定目录中。上传文件通常保存到数据库或服务器的目录中,具体选择哪种方式要根据项目的实际需求决定。如果项目需要快速读取上传的文件,则推荐保存到目录中,因为从数据库中读取文件速度太慢;如果安全性要求较高,则对上传文件的访问有权限限制,推荐将其保存到数据库。

5.5.3　案例部署和测试

将编写完成的案例项目部署到 Tomcat 服务器,使用浏览器请求增加员工的 JSP 页面,如图 5-9 所示。

输入新员工对应的数据,尤其是要选择员工的照片文件。单击"提交"按钮,请求到增加处理 Servlet。Servlet 取得请求数据后,完成处理,在数据库员工表 oa_employee 中增加一条新记录。增加的员工记录数据如图 5-10 所示。

如果图片上传成功,则 PHOTO 字段显示"(Binary/Image)",表明文件已经存储到数据库中;如果没有文件上

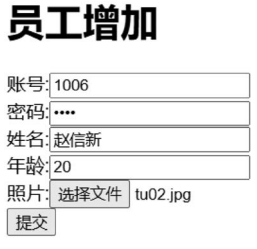


图 5-9　员工增加页面显示

1 信息 | 2 表数据 | 3 信息

限制行 第一行:

EMPID	DEPTNO	EMPPassword	EMPNAME	EMPSEX	AGE	BirthDAY	JOINDATE	SALARY	PHOTO
1001	1	1001	王明	男	20	1989-10-01	2013-10-10	3000.00	(NULL)
1002	1	1002	刘明	男	21	1988-05-01	2012-10-10	4000.00	(NULL)
1003	3	1003	赵明	男	22	1987-10-01	2011-11-10	5000.00	(NULL)
1004	4	1004	赵志刚	男	22	2022-10-31	2022-10-31	9899.00	(Binary/Image)
1006	(NULL)	1006	赵信新	(NULL)	20	(NULL)	(NULL)	0.00	(Binary/Image)
105	(NULL)	105	刘欣欣	男	20	2020-10-09	2023-10-09	5555.00	(Binary/Image)
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	18	(NULL)	(NULL)	0.00	(NULL)

图 5-10　增加的员工记录数据

传,则显示“(NULL)”。单击 PHOTO 的“(Binary/Image)”,MySQL 的客户端工具 SQLYog 弹出显示存储的图片对话框,如图 5-11 所示。

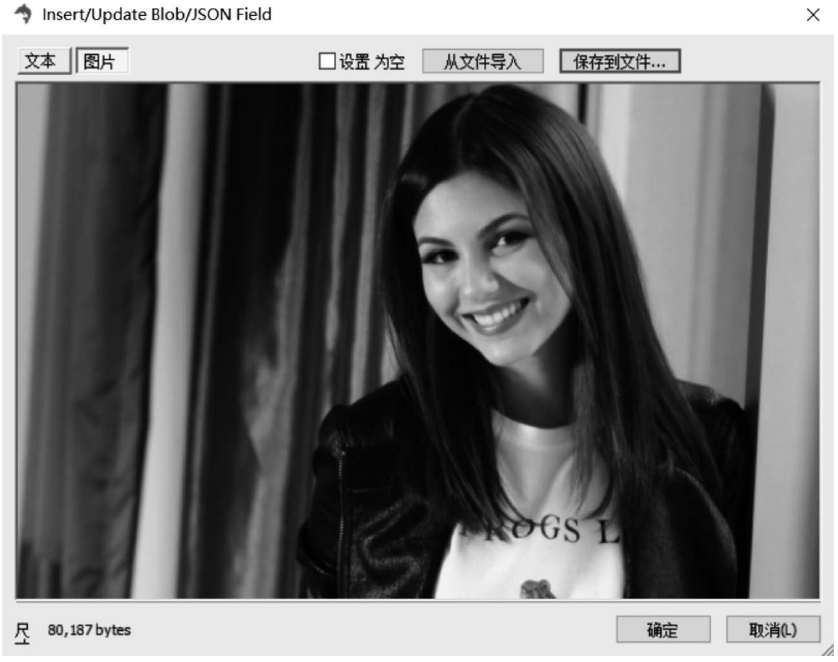


图 5-11　表中存储的图片

Servlet 处理完成后,直接在 Servlet 中显示处理信息,如图 5-12 所示。

此 Servlet 在保存上传图片到数据库中的同时,也将文件保存到 D:/temp 目录中。使用 Windows 的资源管理器,可以看到此目录下有上传的文件,如图 5-13 所示。

图 5-12　员工增加处理后的显示信息

图 5-13　将上传图片保存到 D:/temp 目录中

通过 Servlet 的编程可见,新版 Servlet 由于内置了文件上传机制,使得处理文件上传的编程非常简单,不再需要引入并使用第三方文件上传框架。

简答题

1. 简述请求对象的生命周期。
2. 描述请求对象的主要方法。

实验题

1. 创建 Web 项目：项目名：erpweb；项目使用规范：Jakarta EE 6.0。

2. 创建增加客户表单页面/customer/add.jsp，显示增加客户表单：

编号：文本框

登录密码：密码框

公司名称：文本框

是否上市：是 否 单选按钮。

购买产品：复选框(至少有 4 个产品名称)

公司人数：文本框

年销售额：文本框

提交按钮

提交后请求 Servelt 进行处理。

3. 编写客户增加处理 Servlet。

(1)包名：com.city.erp.servlet。(2)类名：CustomerAddAction。(3)映射地址为：/customer/add.do。(4)功能：取得表单提交的数据，根据需要进行相应的数据类型转换，在创建的 Customer 表中增加一个新客户。处理成功后显示“处理完毕”和返回超链接，否则显示异常信息。

4. 创建数据库和客户表

使用本机的 MySQL，创建数据库。

数据库名称：cityerp。

用户：cityerp。

密码：cityerp。

表：Customer，其结构如表 5-3 所示。

表 5-3　Customer 表字段结构

字　段　名	类　　型	说　　明
CompanyID	Varchar(20)	公司 ID
Password	Varchar(20)	密码
CompanyName	Varchar(50)	公司名称
staffnum	Int	公司人数
Income	Decimal(18,2)	年销售额
CompanyType	Char(4)	是否上市
Products	Varchar(200)	购买产品列表，使用空格分开