第 5 章 HTTP与Servlet核心技术

第 2～4 章所学习的 HTML、CSS 和 JavaScript 语言主要用于展示静态页面，虽然 JavaScript 也提供了一些简单的动态交互功能，例如页面 DOM 元素的增删、页面显示效果的变化等，但这些交互均由浏览器端解释执行，本质上并不需要 Web 服务器端的处理。但是，如果用户想查看最近新增的题目或者最新的榜单排名这类资源，那么就必须去后台数据库中查询并返回相应的结果数据，因为这些数据都是实时动态变化的，而这种动态资源的开发正是本章要阐述的主题。

5.1 动态 Web 资源与服务器端程序

动态 Web 资源是指由 Web 应用程序根据用户请求、上下文环境或实时数据动态生成的资源，其内容会随请求参数、时间、用户身份等时刻变化，而非固定存储在服务器中的静态文件。与静态 Web 资源（如 HTML、CSS、图片等内容固定的文件）相比，动态 Web 资源的核心特征是“按需生成”。动态资源需要 Web 应用程序通过业务逻辑处理和数据交互来生成，是实现个性化交互服务的基础。下面通过编写一个显示服务器时间的程序来说明动态 Web 资源的生成与访问。

首先使用 Eclipse 软件创建一个名称为 chap05_servlet 的 Web 项目，具体步骤请参见 1.4 节（使用 IntelliJ IDEA 软件创建类似，具体步骤请参见 1.5 节）。

接着右击 Java Resources 下的 src 目录，在弹出的快捷菜单中选择 New→Servlet 新建一个 Servlet，如图 5-1 所示。

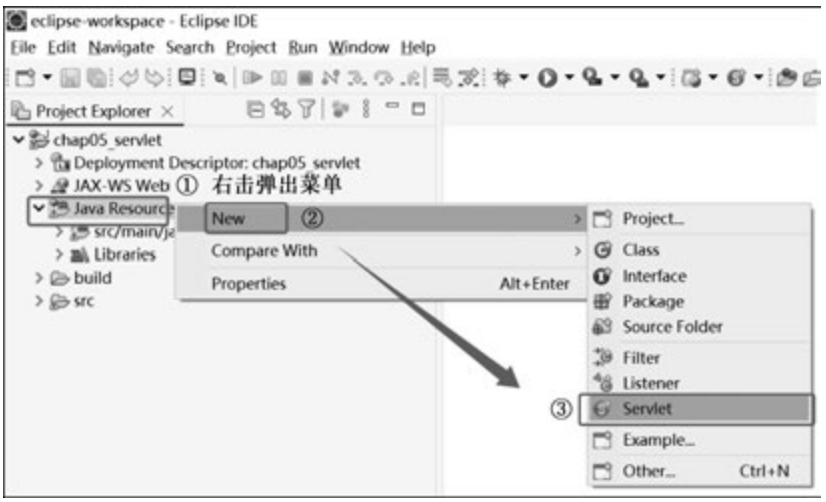

图 5-1 新建一个 Servlet

在弹出的对话框中输入 Java package 为 com.myoj.common，Class name 为 ShowServerTime，单击 Next 按钮，如图 5-2 所示。

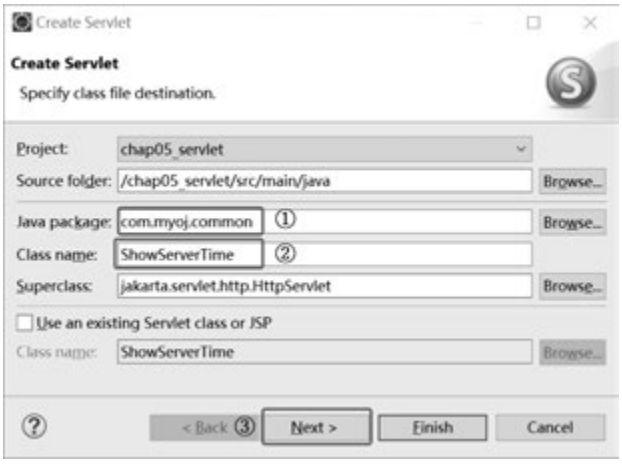

图 5-2　填写 Java package 及 Class name

进入下一步，保持默认配置，单击 Finish 按钮完成 Servlet 的创建，如图 5-3 所示。其中，Name 是这个 Servlet 的别名，URL mappings 则可通俗地认为是访问该 Servlet 所对应的映射网址。

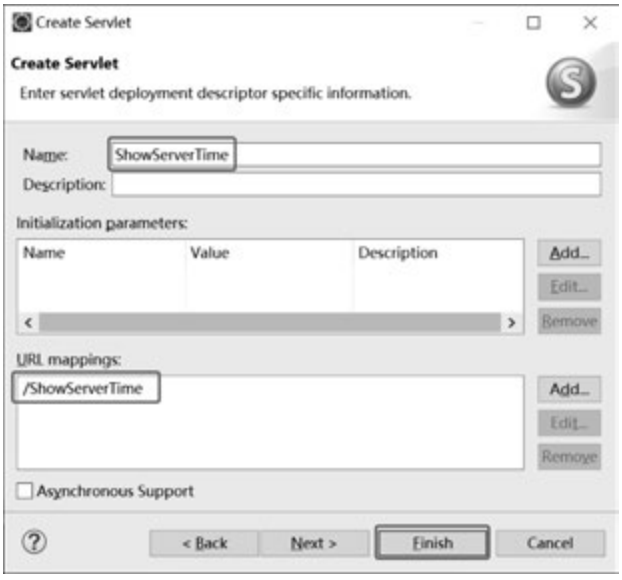

图 5-3　配置 Servlet Name 及 URL mappings

Eclipse 会自动生成一些代码，减少程序员的工作量，在此基础上可以直接修改 ShowServerTime.java 中的 doGet()方法以显示当前服务器的时间，如例 5-1 所示。

例 5-1　ShowServerTime.java

```
package com.myoj.common;

import java.io.PrintWriter;
import java.util.Calendar;
import java.io.IOException;
```

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
@WebServlet("/ShowServerTime")
public class ShowServerTime extends HttpServlet {
    …
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        PrintWriter out = response.getWriter();
        Calendar currentTime = Calendar.getInstance();
        out.print("Server time is: " +
                (currentTime.get((Calendar.MONTH)) + 1) + " - " +
                currentTime.get(Calendar.DAY_OF_MONTH) + " " +
                currentTime.get(Calendar.HOUR_OF_DAY) + ":" +
                currentTime.get(Calendar.MINUTE) + ":" +
                currentTime.get(Calendar.SECOND));
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        //TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

其中，引入了两个需要的类：PrintWriter 用于向浏览器客户端输出响应内容，Calendar 用于获取服务器当前的时间。注意，在获取月份时需要加 1，因为返回的月份值是从 0 开始计算的。

保存并尝试执行 ShowServerTime.java 程序，不过不同于以往所学过的 Java 程序，这其中没有可以直接执行的 main()方法，执行 ShowServerTime.java 程序首先要将编译后的 ShowServerTime.class 部署到 Tomcat 服务器上，然后通过浏览器输入网址来访问。

右击 ShowServerTime.java，在弹出的快捷菜单中选择 Run as→Run on Server，然后在弹出的对话框中选择一个已经部署在 Eclipse 中的 Tomcat 服务器，单击 Finish 按钮将 chap05_servlet 项目部署到 Tomcat 服务器上，如图 5-4 所示。

(a)

图 5-4　部署 chap05_servlet 项目到 Tomcat 服务器上

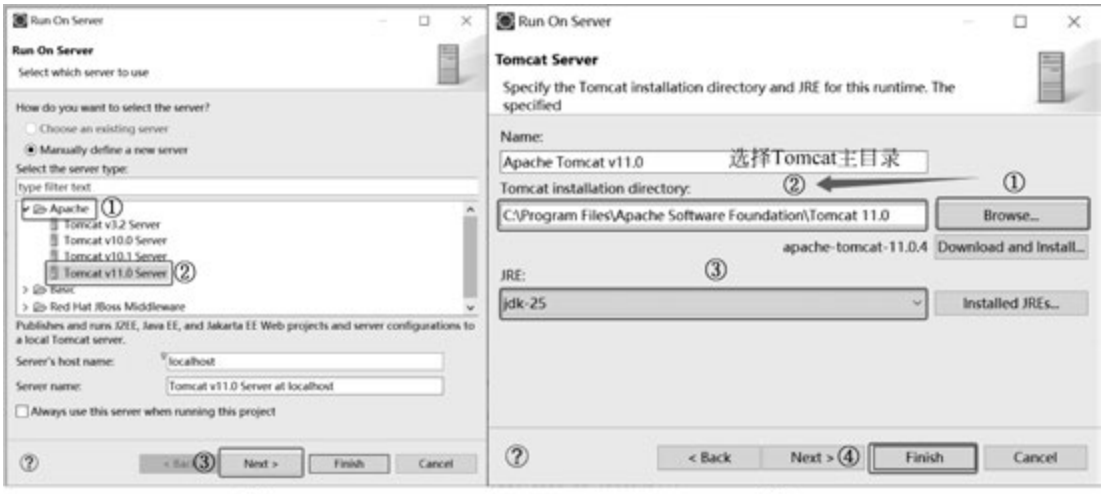

图 5-4　（续）

接着 Eclipse 会自动打开自带浏览器并访问 ShowServerTime.class 对应的 URL，如图 5-5 所示。

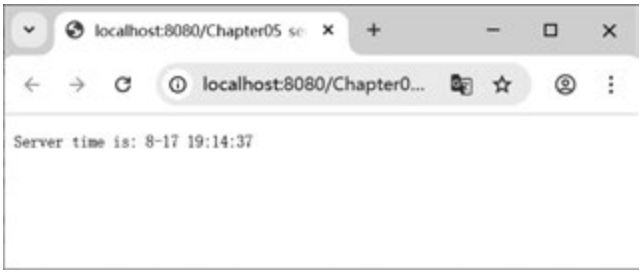

图 5-5　使用浏览器访问 ShowServerTime.class 对应 URL 的效果

可以看到，网页显示的内容是 out.print()所打印输出的内容，也可尝试修改打印内容并重新部署、查看修改后的效果。

5.2　HTTP 简介

结合 ShowServerTime.java 的源代码，接下来将深入探索服务器打印输出的内容是如何显示在浏览器上的。

收到访问某个 URL 的命令后，浏览器首先会从 URL 中解析出服务器的地址，例如 http://localhost:8080/chap05_servlet/ShowServerTime 中服务器地址就是 localhost，表示是本机地址，等价于 127.0.0.1；8080 则是服务器开放的端口号。

接着浏览器和服务器之间会采用 HTTP 来完成数据通信，整个过程包含 4 个主要步骤，如图 5-6 所示。

通信结束后，浏览器会解释收到的资源内容并将解释呈现在网页上，了解 HTTP 细节以及分解 HTTP 请求和响应消息包是构建服务器端动态 Web 资源的关键。

HTTP 是客户端与服务器端之间进行数据通信的无状态应用层协议，它定义了客户端请求与服务器端响应的格式、传输规则及交互流程，每次请求和响应都是一个独立的事务处理。

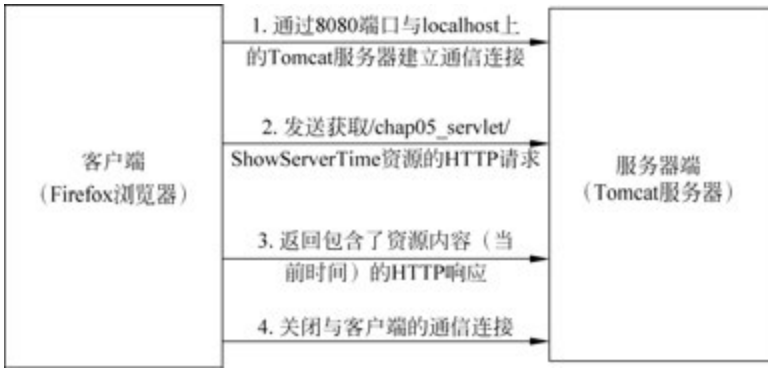

图 5-6　使用 HTTP 的数据通信过程示例

HTTP 具有良好的简洁性、扩展性和通用性。简洁性是指基于文本的报文格式，易于人类阅读和调试，同时降低了协议实现的复杂度；扩展性则通过请求方法、头部字段等可扩展机制实现，允许开发者根据需求自定义功能（如添加认证信息、缓存控制策略）；通用性表现为其支持多种数据类型的传输，不仅能传输超文本（HTML），还能传输图片、JSON 数据、音视频等，能够适配 Web 应用的多样化资源传输需求。这些特点使得 HTTP 成为互联网中应用最广泛的通信协议，支撑着简单网页浏览和复杂交互系统等所有应用场景。

HTTP 自诞生以来经历了多次版本迭代，各版本在性能、安全性和功能上不断优化。1996 年推出的 HTTP/1.0 是首个标准化版本，采用“一次连接一次请求”的模式，每次请求需重新建立 TCP 连接，效率较低。于是，在 1997 年推出的 HTTP/1.1 引入了持久连接（keep-alive）机制，允许同一 TCP 连接中处理多个请求，减少连接建立开销，同时新增了请求头缓存控制、范围请求等功能，成为长期主导 Web 通信的版本。

2015 年推出的 HTTP/2 和 2020 年推出的 HTTP/3 则进一步提升了协议性能。HTTP/2 采用二进制分帧技术，将请求与响应拆分为更小的帧进行并行传输，解决了 HTTP/1.1 的“队头阻塞”问题；同时引入服务器推送功能，可主动向客户端发送关联资源（例如 HTML 页面所需的 CSS 文件），减少请求次数。HTTP/3 则基于 QUIC（Quick UDP Internet Connection，快速 UDP 互联网连接）协议，通过 UDP 实现连接复用和更快的握手速度，进一步优化了弱网络环境下的传输稳定性。

可以看出，HTTP 版本持续演进的根本目的都是提高 Web 应用的访问速度、安全性和并发处理能力。

虽然 HTTP 在快速演变之中，但请求/响应的通信模式一直未发生变化。下面将结合服务器时间显示程序来分析 HTTP 请求和响应消息的格式。

▶ 5.2.1　HTTP 请求消息

一个 HTTP 请求消息由请求行（request line）、请求头（header）、空行（blank line）和请求体（request body）4 部分组成。

1. **请求行**

请求行包括三部分：请求方法、请求 URI 和 HTTP 版本号，这三部分之间用空格分隔。例如：GET /chap05_servlet/ShowServerTime HTTP/1.1，请求方法是 GET，/chap05_servlet/ShowServerTime 是请求 URI，协议版本号是 1.1。

除了 GET 请求，HTTP/1.1 定义了 8 种常用的请求方法来满足不同场景的需要，如表 5-1 所示。

表 5-1 常用 HTTP 请求方法

请求方法名称	描 述
GET	请求指定的资源，并返回实体主体
HEAD	类似于 GET 请求但返回的响应中没有具体的内容，用于获取消息头部
POST	向指定资源提交数据处理请求（例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/或已有资源的修改
PUT	从客户端向服务器端传送的数据取代指定的文档内容
DELETE	请求服务器端删除指定的页面
CONNECT	HTTP/1.1 中预留给能够将连接改为管道方式的代理服务器
OPTIONS	允许客户端查看服务器端的性能
TRACE	回显服务器端收到的请求，主要用于测试或诊断

2. 请求头和空行

请求头为请求消息添加了一些附加信息，由 key:value 对组成，每行一对，key 和 value 之间使用冒号分隔。请求头的最后会有一个空行，表示请求头结束，接下来是请求数据。常用的请求头如表 5-2 所示。

表 5-2 常用 HTTP 请求头

请求头名称	描 述
Accept	可接受的响应内容类型
Accept-Charset	可接受的字符集
Accept-Encoding	可接受的响应内容的编码方式
Accept-Language	可接受的响应内容语言列表
Authorization	用于表示 HTTP 中需要认证资源的认证信息
Cache-Control	用来指定当前的请求/回复是否使用缓存机制
Connection	客户端（浏览器）想要优先使用的连接类型
Cookie	由之前服务器通过 Set-Cookie（见下文）设置的一个 HTTP Cookie
Content-Length	以八进制表示的请求体的长度
Content-Type	请求体的 MIME 类型（用于 POST 和 PUT 请求中）
Date	发送该消息的日期和时间（以 RFC 7231 中定义的“HTTP 日期”格式来发送）
Host	表示服务器的域名以及服务器所监听的端口号。如果所请求的端口是对应的服务的标准端口 80，则端口号可以省略
User-Agent	表示浏览器身份标识的字符串

3. 请求体

请求体一般用来存储请求的参数数据，这些参数数据将会一起发送到服务器端。

下面利用浏览器来观察例 5-1 中的 HTTP 请求消息，打开浏览器，按下 F12 键，选择“网络”选项，输入地址 http://localhost:8080/chap05_servlet/ShowServerTime 后可以看到 HTTP 请求消息的具体结构，如图 5-7 所示。

▶ 5.2.2 HTTP 响应消息

HTTP 响应报文由三部分组成：响应行（response line）、响应头（response header）、响应体（response body）。

1. 响应行

响应行由协议版本、状态码（status-code）和状态描述（reason-phrase）组成，这三部分也由空格分隔。例如：HTTP/1.1 200 OK，其中：HTTP/1.1 是协议版本号；200 是状态码，表示

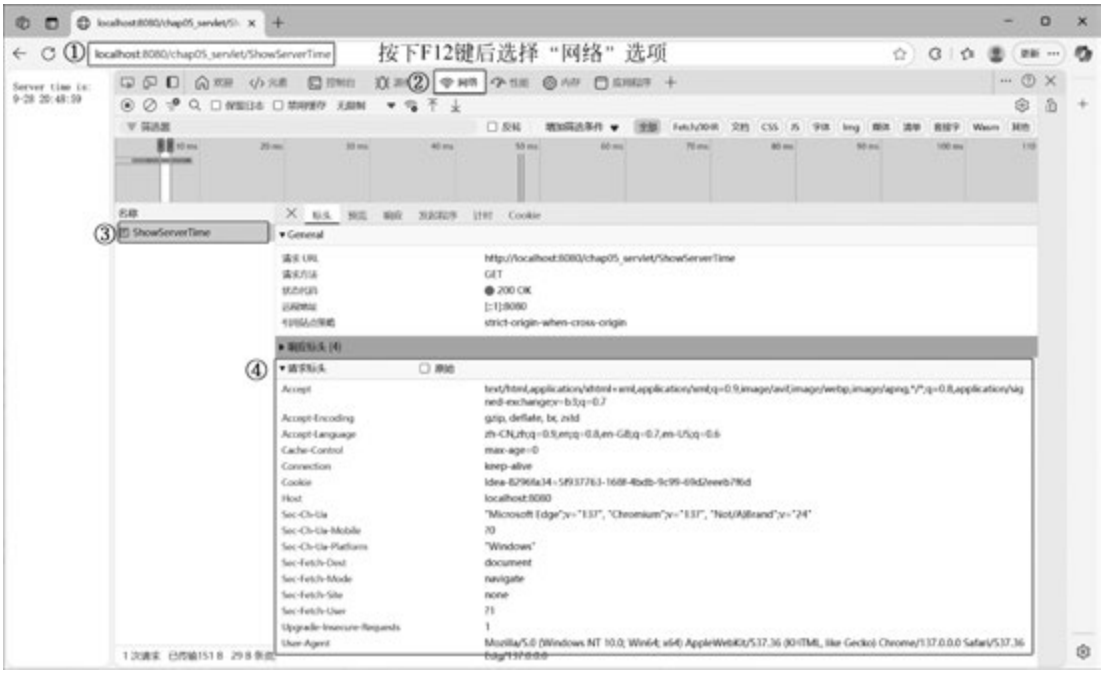

图 5-7　查看 HTTP 请求消息示例

处理请求成功；OK 是状态描述。HTTP 响应状态码由 3 位数字组成，分为 5 个类别，分别表示不同的处理结果。

-1xx(信息类)：表示请求已接收，继续处理(如 100 Continue)。

-2xx(成功类)：表示请求已成功处理(如 200 OK)。

-3xx(重定向类)：表示需要客户端进一步操作才能完成请求(如 302 Found)。

-4xx(客户端错误类)：表示请求存在错误(如 400 Bad Request，404 Not Found)。

-5xx(服务器端错误类)：表示服务器端处理请求时发生错误(如 500 Internal Server Error)。

常见状态码及其含义如表 5-3 所示。

表 5-3　常见状态码及其含义

状态码	状态码的英文描述和功能
100	Continue。这个临时响应是用来通知客户端它的部分请求已经被服务器端接收，且未被拒绝，客户端应当继续发送下一次请求才能完成
200	Ok。请求已成功，请求所希望的响应头或数据体将随此响应返回
302	Found。请求的资源临时从不同的 URI 响应请求。由于这样的重定向是临时的，因此客户端应当继续向原有地址发送以后的请求
401	Unauthorized。当前请求需要用户验证。客户端可以重复提交一个包含恰当的 Authorization 头信息的请求。如果当前请求已经包含了 Authorization 证书，那么 401 响应代表服务器端验证已经拒绝了那些证书
403	Forbidden。服务器端已经理解请求，但是拒绝执行它
404	Not Found。请求失败，请求所希望得到的资源未在服务器端发现
500	Internal Server Error。服务器端遇到了一个未曾预料的状况，导致了它无法完成对请求的处理。一般来说，这个问题都会在服务器端的源代码出现错误时出现
502	Bad Gateway。作为网关或者代理工作的服务器尝试执行请求时，从上游服务器接收到无效的响应

2．响应头

响应头中包含服务器端描述信息和数据描述信息，用于通知客户端浏览器如何处理将接收的回送数据。常见响应头及其含义如表 5-4 所示。

表 5-4　常见响应头及其含义

名　　称	含　　义
Allow	对于特定资源的有效动作
Cache-Control	通知从服务器到客户端内的所有缓存机制，表示它们是否可以缓存这个对象及缓存有效时间，单位为秒
Connection	针对该连接所预期的选项
Content-Disposition	描述对资源的处理，浏览器可以根据这个响应头决定是对返回资源的动作，如将其下载或是打开
Content-Encoding	响应资源所使用的编码类型
Content-Length	响应消息体的长度，用八进制字节表示
Content-Type	当前内容的 MIME 类型，并通过 charset 参数指定响应内容的字符编码
Refresh	用于重定向，或者当一个新的资源被创建时。默认会在 5s 后刷新重定向
Set-Cookie	设置 HTTP Cookie
WWW-Authenticate	表示在请求获取这个实体时应当使用的认证模式

3．响应体

响应体是真正发送给客户端的数据，如例 5-1 中的服务器当前时间。

下面利用浏览器来观察例 5-1 中的 HTTP 响应消息，打开浏览器，按下 F12 键，选择“网络”选项，输入地址 http://localhost:8080/chap05_servlet/ShowServerTime 后可以看到 HTTP 响应消息的具体结构，如图 5-8 所示。

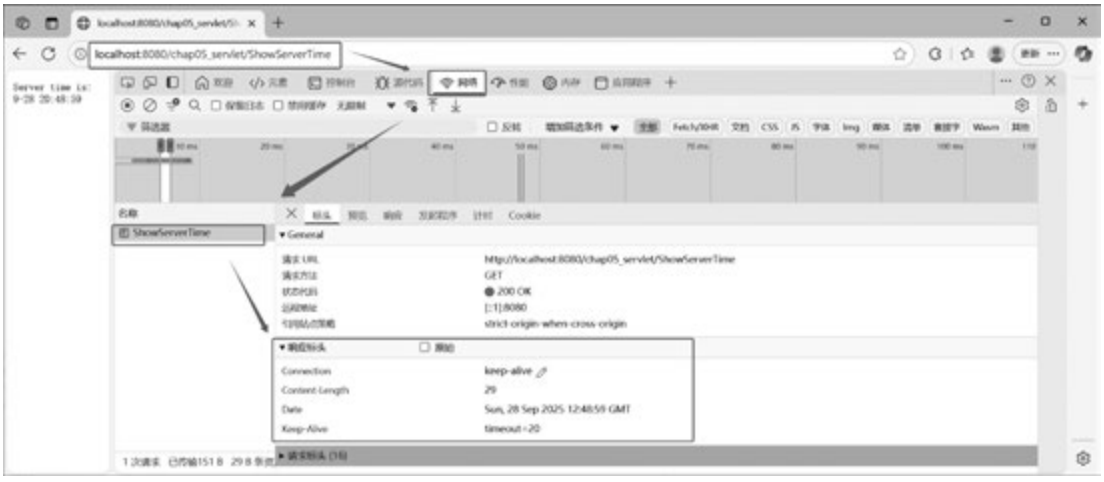

图 5-8　查看 HTTP 响应头及响应数据

▶ 5.2.3　HTTP 服务器的模拟实现

从 5.2.2 节可以看到，HTTP 细节众多，要完整地解析 HTTP 是一件非常烦琐的事情，下面利用 Java 语言提供的网络包来开发一个简易 HTTP 服务器，以帮助理解 Web 服务器工作的过程。

打开 Eclipse，在 com.myoj.common 下新建一个名称为 MyHTTPServer 的普通 Java 类

来发送服务器时间给浏览器，如图 5-9 所示。

图 5-9　新建 MyHTTPServer 类

MyHTTPServer.java 源代码如例 5-2 所示。

例 5-2　MyHTTPServer.java

```
package com.myoj.common;

import java.io.*;
import java.net.*;
import java.util.Calendar;

public class MyHTTPServer {
  public static void main(String[] args) {
     try {
        ServerSocket ss = new ServerSocket(8888);          //服务器监听端口号为 8888
        System.out.println("服务器启动成功,监听端口 8888...");
        while (true) {
            Socket socket = ss.accept();                   //接受客户端请求
            //1. 生成客户端输入流,读取 HTTP 请求消息:请求头、换行符、请求体等
                BufferedReader br = new BufferedReader(new InputStreamReader(socket.
getInputStream()));
            //2. 解析 HTTP 请求消息
            String requestInfo = "",url = "";
            while ((requestInfo = br.readLine()) != null && !requestInfo.isEmpty()) {

                //3. 输出请求头
                System.out.println(requestInfo);
                //4. 读取 GET 请求方法及 URL
                if (requestInfo.startsWith("GET")) {        //如果请求行显示是 GET 请求
                    int begin = requestInfo.indexOf("/") + 1;
                    int end = requestInfo.indexOf("HTTP/");
                    url = requestInfo.substring(begin, end - 1);
                    System.out.println("收到 GET 请求 URL 是:" + url);
                    System.out.println("正在发送响应...");
                    PrintWriter pw = new PrintWriter(socket.getOutputStream());
                    /*模拟发送 HTTP 响应*/
                    if(url.equals("ShowServerTime")) {
                       pw.println("HTTP/1.1 200 OK");      //处理请求成功
                       pw.println("Content-type:text/html");
                       pw.println("");
                       Calendar currentTime = Calendar.getInstance();
```

```
                        pw.println("欢迎访问 MyHTTPServer!当前服务器时间: " +
                                (currentTime.get((Calendar.MONTH)) + 1) + "-" +
                                currentTime.get(Calendar.DAY_OF_MONTH) + " " +
                                currentTime.get(Calendar.HOUR_OF_DAY) + ":" +
                                currentTime.get(Calendar.MINUTE) + ":" +
                                currentTime.get(Calendar.SECOND));
                    }else {
                        pw.println("HTTP/1.1 404 Not found");   //未找到资源,无法处理
                        pw.println("Content-type:text/html");
                        pw.println("");
                    }
                    pw.flush();                          //立即将输出缓冲区的内容发送到客户端
                }
            }
            socket.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
  }
}
```

运行例 5-2,启动 MyHTTPServer 服务器,如图 5-10 所示。

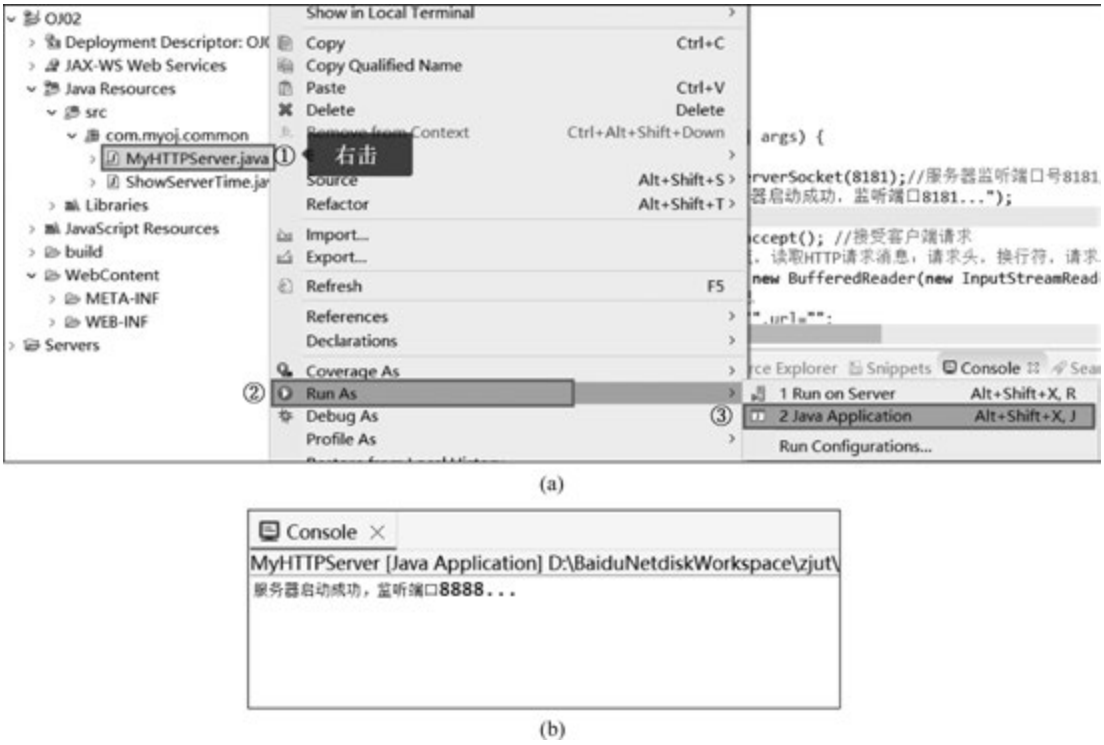

图 5-10　启动 MyHTTPServer 服务器

下面打开浏览器,输入 http://localhost:8888/ShowServerTime 来访问服务器的当前时间,如图 5-11 所示,这样就完成了对模拟 HTTP 服务器的访问。

试着修改 ShowServerTime 为其他任意 URI,会看到模拟服务器返回的 404 错误。

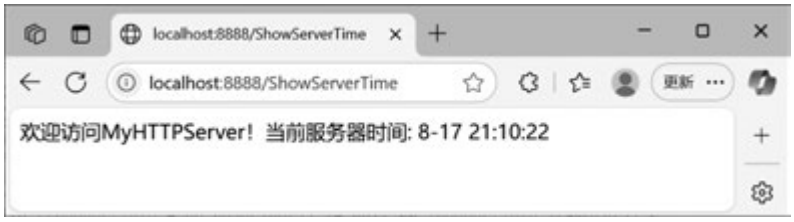

图 5-11　访问模拟 HTTP 服务器

从例 5-2 可以看出，HTTP 细节众多，如果从基础开始在服务器端进行 Web 编程会非常烦琐，所幸的是，现有的 Web 服务器（如 Tomcat 和 Apache 等）已经为开发者大大地简化了编程过程，它们不仅屏蔽了 HTTP 的通信细节，而且提供了各种功能的 API，让程序员能够集中精力关注主要的业务工作。Web 服务器工作的基本步骤和原理如图 5-12 所示（以 Tomcat 服务器为例）。

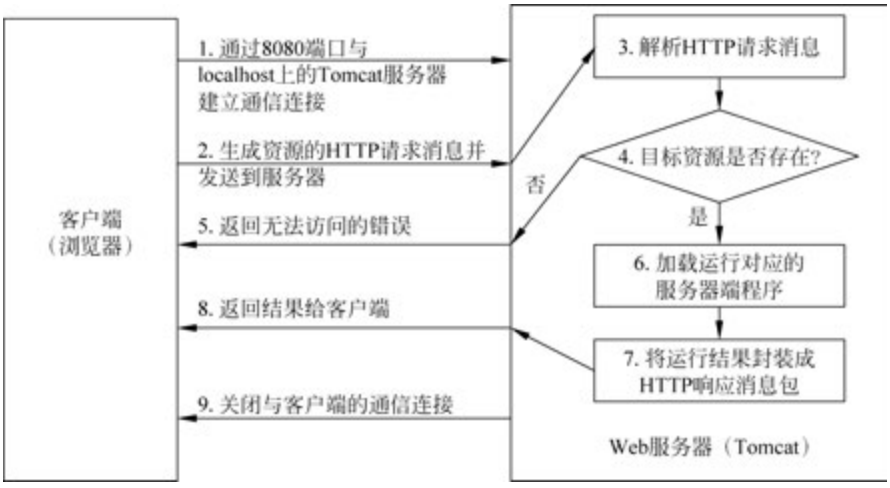

图 5-12　Web 服务器工作的基本步骤和原理

5.3　Servlet 技术简介及其 API 组成

Servlet 技术是 Jakarta EE 规范（原 Java EE 规范）中用于开发动态 Web 资源的核心。Servlet 本质上是运行在 Web 服务器中的 Java 类，通过处理 HTTP 请求、生成动态响应，实现客户端与服务器端的业务交互。与静态资源及简单的脚本语言不同，Servlet 依托 Java 语言的跨平台特性和面向对象能力，为 Web 应用开发提供了稳定、可扩展的动态资源生成方案。

Servlet 最初的设计目标是解决动态 Web 资源开发的共性问题：一是实现请求与响应的标准化处理，通过统一接口封装 HTTP 细节（如请求参数解析、响应头设置），降低开发者对底层通信的关注；二是支持组件化开发，Servlet 作为独立组件可被 Web 服务器管理（如自动实例化、生命周期控制），便于与其他 Java 技术（如数据库访问、第三方框架）的集成；三是保证跨容器兼容性，遵循 Servlet 规范的应用可在任何支持该规范的 Web 服务器（如 Tomcat、Jetty 和 Jboss 等）中运行，提升了代码的可移植性。

Servlet 技术通过规范与实现的分离，既保证了开发的标准化，又为 Web 服务器厂商提供了灵活的实现空间。在动态 Web 资源开发中，Servlet 技术为 Java 生态提供了坚实的基础，至

今仍是构建企业级 Web 应用的重要选择。

Servlet 技术历经多个版本迭代，Servlet 1.0 于 1997 年推出，是该技术的首个正式版本，定义了 Servlet 的基本接口和生命周期，奠定了 Servlet 技术的基础，但功能相对简单，仅支持基本的请求和响应处理。

Servlet 2.0 在 2000 年发布，引入了对 Web 应用的部署描述文件(web.xml)的支持支持，使得 Servlet 的配置更加灵活，开发者可以通过配置文件管理 Servlet 的映射、初始化参数等，无须硬编码在代码中，增强了应用的可维护性。

2003 年推出的 Servlet 2.4 进一步完善与 JSP 2.0 的协同工作，支持更多的 HTTP 方法和请求属性，同时对过滤器(filter)机制进行了增强，使得开发者可以更方便地对请求和响应进行拦截和处理，例如实现权限控制、日志记录等功能。

Servlet 3.0 于 2009 年发布，引入了注解(annotation)支持，开发者可以通过注解来配置 Servlet、过滤器和监听器等，减少了对 web.xml 的依赖，简化了开发流程；同时引入了异步处理机制，允许 Servlet 在处理耗时操作时释放容器线程，提高了服务器的并发处理能力。2013 年推出的 Servlet 3.1 新增了对非阻塞 I/O 的支持，进一步提升了 Servlet 处理高并发请求，尤其是大量异步请求时的性能。

2017 年发布的 Servlet 4.0 主要新增了对 HTTP/2 的支持，能够更好地利用 HTTP/2 的多路复用等特性，提升 Web 应用的性能和响应速度，适应现代 Web 应用对高性能的需求。

Servlet 5.0 于 2019 年发布，这一版本着重提升了对 Java EE 新特性的适配及安全性优化。在 Java EE 生态迁移至 Jakarta EE 的背景下，Servlet 5.0 完成了相关命名空间的切换，从 javax.servlet 迁移至 jakarta.servlet，为后续版本的持续演进奠定基础。安全层面，它更新了对 HTTP 安全头的支持，帮助开发者构建更安全的 Web 应用，抵御常见的 Web 攻击(如跨站脚本攻击、单击劫持等)。同时，Servlet 5.0 对 Servlet 容器的启动和部署性能进行了优化，减少了 Web 应用的启动时间，提升了开发和部署效率。

2023 年推出的 Servlet 6.0 进一步完善了规范细节，明确了 URI 路径的解码和规范化规则，避免因路径解析不一致导致的潜在问题，使 Web 应用在处理复杂请求路径时更加稳健。

Servlet API 是 Servlet 技术的核心，它为开发者提供了一套标准化的接口和类，便于处理 HTTP 请求与响应、Servlet 与 Web 容器的交互等主要功能。以 Servlet 5.0 及后续版本为例，Servlet API 主要包含两个核心包，分别对应通用 Servlet 规范和 HTTP 扩展。

- jakarta.servlet 包：定义了 Servlet 技术的基础规范，涵盖 Servlet 的生命周期管理、配置信息、上下文环境等核心概念，不依赖具体的通信协议，适用于各种基于请求-响应模型的服务。
- jakarta.servlet.http 包：基于 jakarta.servlet 包针对 HTTP 设计的扩展，提供了处理 HTTP 请求、响应、会话管理等功能的接口和类，是 Web 应用开发中最常用的包。

1. **主要接口**

Servlet 接口：所有 Servlet 类的根接口，定义了 Servlet 的 5 个核心方法：初始化方法 init()、核心服务方法 service()、销毁方法 destroy()、获取 Servlet 信息方法 getServletInfo()和获取 Servlet 配置信息方法 getServletConfig()等。Web 容器通过调用这些方法管理 Servlet 的生命周期，开发者需通过实现该接口或继承其实现类来开发自定义 Servlet。

ServletConfig 接口：用于获取 Servlet 的配置信息，如初始化参数方法 getInitParameter()、获取 Servlet 上下文方法 getServletContext()等。它使 Servlet 在部署时能够配置并传入参数，

增强了 Servlet 的灵活性。

ServletContext 接口：代表整个 Web 应用程序的上下文环境，是 Servlet 与 Web 容器之间以及同一应用内多个 Servlet 之间共享资源的桥梁。通过它可以获取 Servlet 所述应用程序的初始化参数、访问资源文件、管理会话等。

ServletRequest 与 HttpServletRequest 接口：ServletRequest 是通用请求接口，定义了请求参数、输入流、属性的获取方法。HttpServletRequest 则继承自 ServletRequest，增加了 HTTP 特有的功能，如获取请求方法 getMethod()、请求 URL 获取方法 getRequestURL 等，它是处理 HTTP 请求的核心接口。

ServletResponse 与 HttpServletResponse 接口：ServletResponse 是通用响应接口，提供了设置响应内容类型、获取输出流等方法；HttpServletResponse 则继承自 ServletResponse，增加了 HTTP 相关的设置响应状态码方法 setStatus()、添加 Cookie 方法 addCookie()、设置响应重定向方法 sendRedirect()等。

2. **主要类**

GenericServlet 类：该类实现了 Servlet、ServletConfig 和 Serializable 三个接口，是一个通用的 Servlet 基类，它提供了 ServletConfig 接口方法的默认实现。

HttpServlet 类：继承自 GenericServlet，专门用于处理 HTTP 请求，它是 Web 开发中最重要也最常用的 Servlet 基类。通过重写 service()方法，HttpServlet 能够根据 HTTP 请求方法的种类(如 GET、POST、PUT 等)将请求分发给对应的 doGet()、doPost()等方法，开发者只需重写这些方法即可处理特定类型的请求，而无须关注协议细节。

这些接口和类共同构成了 Servlet API 的核心，使开发者能够专注于业务逻辑实现，同时确保 Servlet 在不同 Web 容器中正常运行。

下面利用不限于任何协议的 GenericServlet 类来实现服务器时间显示功能，如例 5-3 所示。

例 5-3　GenericServerTime.java

```
package com.myoj.common;

import java.util.*;
import java.io.IOException;
import java.io.PrintWriter;
import jakarta.servlet.GenericServlet;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRequest;
import jakarta.servlet.ServletResponse;
import jakarta.servlet.annotation.WebServlet;

@WebServlet(name = "genericServerTime", urlPatterns = { "/ShowServerTime" })
public class GenericServerTime extends GenericServlet {
    private ServletConfig servletConfig;

    @Override
    public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
        servletConfig = getServletConfig();
        String servletName = servletConfig.getServletName();
```

```
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("Servlet name is: " + servletName + "<br>");
        Calendar currentTime = Calendar.getInstance();
        out.print("Server time is: " + (currentTime.get((Calendar.MONTH)) + 1)
                + "-" + currentTime.get(Calendar.DAY_OF_MONTH) + " " + currentTime.get
(Calendar.HOUR_OF_DAY) + ":"
                + currentTime.get(Calendar.MINUTE) + ":" + currentTime.get(Calendar.
SECOND));
    }
}
```

运行结果如图 5-13 所示。

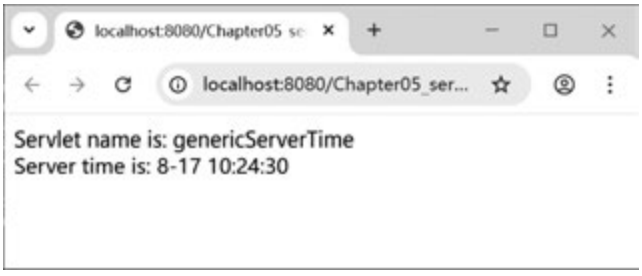

图 5-13　利用 GenericServlet 类显示服务器时间

当用户访问 http://localhost:8080/Chapter05_servlet/ShowServerTime 时，可以看到当前 URL 对应的 Servlet 名称和服务器的当前时间，下面对代码进行解释。

前面各行 import 语句导入了 GenericServlet 抽象类、ServletRequest 和 ServletResponse 抽象接口。GenericServlet 类实现了 Servlet 和 ServletConfig 接口，开发人员直接继承该类并编写核心业务方法 service()即可，而无须实现 Servlet 接口的所有方法。

“import jakarta.servlet.annotation.WebServlet;”语句导入了 WebServlet 注解，通过 WebServlet 注解可以直接配置 Servlet 的访问路径(即 URL)和名称等信息。例如：

```
@WebServlet(name = "genericServerTime", urlPatterns = { "/currentServerTime" })
```

其中，name = "genericServerTime" 指定 Servlet 的名称为 genericServerTime，可通过 ServletConfig 获取该名称；urlPatterns = { "/currentServerTime" }指定访问该 Servlet 的 URL 路径，客户端可通过该 URL 访问这个 Servlet，实现从 URL 路径映射到 Servlet 的执行。

public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException 是重写 GenericServlet 中的核心 service()方法，该方法是 Servlet 处理客户端请求的入口，其中：

参数 ServletRequest request 来自客户端的请求对象；参数 ServletResponse response：向客户端响应的对象；

这两个对象都是由 Web 服务器自动创建，可以直接使用。

“private ServletConfig servletConfig;”语句定义了 servletConfig 变量，通过该变量所指对象的 getServletName()方法可以获取的当前 Servlet 的名称。

“response.setContentType("text/html;charset=UTF-8");”语句用于通知客户端，服务器端将要发送的资源文件类型为 text/html，且该资源文件的字符集使用 UTF-8 编码。客户

端应使用 UTF-8 编码解释该 HTML 文件，否则将会出现乱码问题。

“PrintWriter out = response.getWriter();”语句通过 response 对象的 getWriter()方法，获取输出流对象 out，后续将通过该对象向客户端输出内容。

接下来使用 Calendar 类生成时间并通过 out 输出到客户端。

通过上述例子可以看出，开发者无须关注通信协议细节，只需要编写显示时间的业务代码即可。事实上，对于 HttpServlet 的使用也和 GenericServlet 类似，下面对上述代码稍作变更，以通过 HTTP 获取服务器时间，如例 5-4 所示。

例 5-4　HttpServerTime.java

```
package com.myoj.common;

import java.util.*;
import java.io.IOException;
import java.io.PrintWriter;

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

@WebServlet("/HttpServerTime")
public class HttpServerTime extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        Calendar currentTime = Calendar.getInstance();
        out.print("Server time is: " + (currentTime.get((Calendar.MONTH)) + 1)
                + "-" + currentTime.get(Calendar.DAY_OF_MONTH) + " " + currentTime.get
(Calendar.HOUR_OF_DAY) + ":"
                + currentTime.get(Calendar.MINUTE) + ":" + currentTime.get(Calendar.
SECOND));
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request, response);
    }
}
```

上述代码中，首先导入 HttpServlet 类、HttpServletRequest 和 HttpServletResponse 接口，然后使用@WebServlet("/HttpServerTime")注解 URL 路径。注意，这里使用了更加简洁的注解语法。

接下来通过继承 HttpServlet 类并实现了 doGet()成员方法来输出服务器时间，此处重写的 HttpServlet 类 doGet()方法中也有一组请求和响应的参数 request 和 response，这两个参数也是由服务器根据 HTTP 自动创建的，也就是说，服务器封装了 HTTP 的请求和响应细节，开发人员同样只需要关注核心业务本身(显示服务器时间)。

值得注意的是，doPost()方法直接调用了 doGet()方法，这种相互调用的做法在 Web 应用开发中非常常见，能够使单个 Servlet 在面对多种请求(例如 get 和 post 请求)时采用同样的

业务方法。

5.4　Servlet 的生命周期

前面的例子展示了服务器时间的显示，但这些例子中并没有常见 Java 类中的 main()方法主入口，那么 Servlet 是如何被加载到内存中并运行的呢？Web 容器主导完成了这些工作，图 5-14 显示了 Servlet 的执行过程和 Servlet 生命周期各个阶段的状态切换。

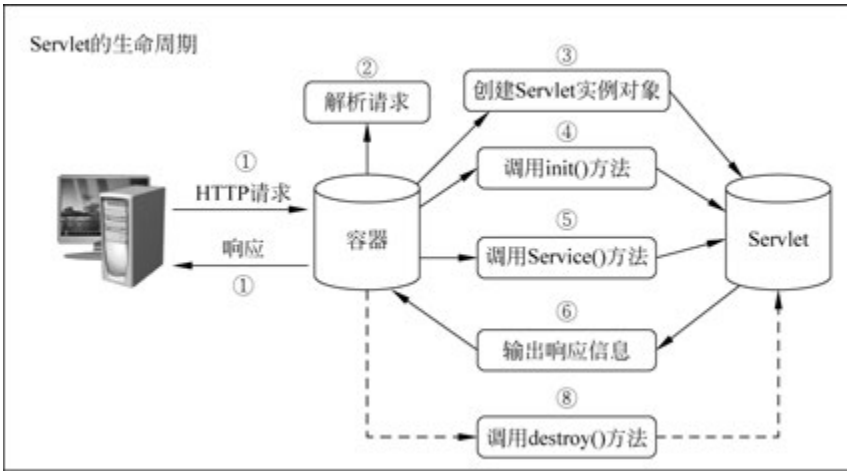

图 5-14　Servlet 的执行过程和 Servlet 生命周期各个阶段的状态切换

Servlet 的执行过程遵循“加载初始化-服务客户-销毁卸载”的生命周期，该过程由 Web 容器(如 Tomcat)全程管理，下面结合 HttpServerTime 类详细解析 Servlet 的完整生命周期。

1. **加载初始化阶段：从类到可用实例(仅执行一次)**

当 Web 应用启动或客户端**首次**请求访问 HttpServerTime 时，Web 容器会首先完成 Servlet 的初始化，具体步骤如下。

(1) 加载类文件：容器通过 class.forName()等类加载器加载 Servlet 类的 class 字节码文件到内存中，并验证类的合法性(如是否继承 HttpServlet)。

(2) 实例化对象：通过 new 关键字创建该 Servlet 的实例对象。

(3) 调用 init()方法：容器调用 HttpServerTime 实例对象的 init(ServletConfig config)方法，完成初始化工作。

2. **服务客户阶段：处理客户请求并响应客户(可多次反复执行)**

(1) 创建请求与响应对象：Web 容器封装客户端请求信息为 HttpServletRequest 对象(如请求方法、参数)，同时创建 HttpServletResponse 对象用于构建响应。

(2) 启动一个线程调用执行 Servlet 的 service()方法：容器将两个对象传入 HttpServerTime 实例的 service(HttpServletRequest request, HttpServletResponse response)方法。该方法会根据 HTTP 请求方法(GET/POST 等)，自动分发到对应的 doGet()或 doPost()方法。每次请求都会触发 service()或其分发的方法，容器将会通过多线程机制支持多个并发请求。

3. **销毁卸载阶段：释放资源(仅执行一次)**

当 Web 应用停止(如服务器关闭)或 Servlet 被移除时，容器启动销毁流程并调用执行

HttpServerTime 实例的 destroy()方法，释放该 Servlet 占用的内存、网络或数据库连接资源。此时，实例不再接收新请求，等待 JVM 垃圾回收机制回收内存。

总体上，通过 Servlet 生命周期可以观察到它不但具有“一次初始化，多次复用，最终销毁”的高效模式，还具备单实例性和容器主导两个特点。

(1) 单实例性：一个 Servlet 类在容器中仅存在一个实例(仅初始化一次)，所有请求共享该实例对象，因此要尽量避免使用成员变量存储数据，以防引发多线程的读写安全问题。

(2) 容器主导：生命周期各阶段的方法均由 Web 容器触发执行，开发人员无权手动调用 init()、service()或 destroy()方法。

5.5 分析处理 HTTP 请求

本节将进一步分析 HTTP 的请求以完成用户注册功能，具体要求如下。

(1) 用户注册页面 signup.html：用户填写用户名、密码和语言偏好(C 语言、C++语言，Java 语言、Python 语言)；

(2) 注册处理功能 RegisterServlet.java：验证输入的用户名和密码合法性，如果不合法(用户名长度小于 6 位或密码长度小于 8 位)，则提示用户重新输入，否则提示“注册成功”并输出个人偏好信息。

首先完成用户注册页面 signup.html，效果图如图 5-15 所示，相关代码如例 5-5 所示。

图 5-15 signup.html 页面效果

例 5-5 signup.html

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <title>用户注册</title>
</head>
<body>
        <h2>欢迎使用在线评判系统，请注册账号</h2>
        <!-- 表单提交到 RegisterCheckServlet，使用 POST 方法 -->
        <form action="RegisterCheckServlet" method="post">
                <label for="username">用户名：</label>
                <input type="text" id="username" name="username" required>
                <label for="password">密码：</label>
                <input type="password" id="password" name="password" required>
```

```
                <br/>
                <label>语言偏好(可多选):</label>
                <div class="checkbox-group">
                    <input type="checkbox" id="lang1" name="language" value="C语言">
                    <label for="lang1">C语言</label>
                    <input type="checkbox" id="lang2" name="language" value="C++语言">
                    <label for="lang2">C++语言</label>
                    <input type="checkbox" id="lang3" name="language" value="Java语言">
                    <label for="lang3">Java语言</label>
                    <input type="checkbox" id="lang4" name="language" value="Python语言">
                    <label for="lang4">Python语言</label>
                </div>
            <button type="submit">注册</button>
        </form>
</body>
</html>
```

由于需要把用户填写和勾选的信息发送到服务器程序进行判定处理，因此“用户名”编辑框 username、“密码”框 password、复选框 language 这些数据都放置在< form >表单中，当用户单击“注册”按钮时，表单中的数据就会发送给对应的服务器程序，也就是< form >表单中 action 属性所指定 URL 路径 RegisterCheckServlet 所对应的 RegisterCheckServlet. class，数据提交的方式是常用的 post 请求，这种方式相比 get 请求更为隐蔽和安全一些。

▶ 5.5.1　读取请求参数

接下来将读取用户提交的数据并进行响应处理，因为所有来自客户端的请求信息都被 Web 容器封装在 HttpServletRequest 对象中，所以开发者只需利用该对象的参数读取方法即可提取到用户填写的信息，参数读取方法主要包括 getParameter()、getParameterValues()和 getParameterNames()等，它们的功能与返回值如表 5-5 所示。

表 5-5　常见用户参数读取方法的功能与返回值

方法原型	功　能	返回值类型
getParameter(String name)	根据参数名 name 获取对应的参数值，适用于单值参数(如文本框、单选按钮等)	String
getParameterValues(String name)	根据参数名 name 获取对应的所有值，返回一个字符串数组，适用于多值参数(如复选框、多选列表等)	String[]
getParameterNames( )	返回请求中所有参数名的枚举 Enumeration 对象，可用于遍历请求中的所有参数，适用于动态处理未知参数名的场景	Enumeration < String >

示例如例 5-6 所示。

例 5-6　RegisterCheckServlet. java

```
package com.myoj.service;

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;

@WebServlet(name = "RegisterCheckServlet", urlPatterns = {"/RegisterCheckServlet"})
public class RegisterCheckServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        //设置请求和响应的编码格式(避免中文乱码)
        request.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        //获取表单提交的参数
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        //复选框参数通过 getParameterValues()获取,返回字符串数组
        String[] languages = request.getParameterValues("language");

        //验证用户名和密码合法性
        boolean isUsernameValid = username != null && username.length() >= 6;
        boolean isPasswordValid = password != null && password.length() >= 8;

        //生成响应内容
        if (isUsernameValid && isPasswordValid) {
            //注册成功:显示成功信息和用户偏好
            out.println("<div class='success'>注册成功!</div>");
            out.println("<p>用户名:" + username + "</p>");
            out.println("<p>语言偏好:");
            if (languages != null && languages.length > 0) {
                //将数组转换为逗号分隔的字符串
                out.println(String.join("、", languages));
            } else {
                out.println("未选择");
            }
            out.println("</p>");
        } else {
            //注册失败:提示错误原因并提供返回链接
            out.println("<div class='error'>注册失败,请重新输入!</div>");
            if (!isUsernameValid) {
                out.println("<p>用户名长度必须至少 6 位</p>");
            }
            if (!isPasswordValid) {
                out.println("<p>密码长度必须至少 8 位</p>");
            }
            out.println("<p><a href='signup.html'>返回注册页面</a></p>");
        }

        out.println("</body>");
        out.println("</html>");
    }
}
```

上述代码中，getParameter()和 getPameterValues()方法中的参数值一定要和 signup.html 页面的用户名、密码和语言偏好等表单域的 name 属性值一致(包括大小写)，如不一致会导致无法正确读取用户填写和勾选的信息。此外，还应注意多选时的复选框 name 属性也要保持一致，否则会被看作多组数据。

下面尝试将表单< form >的 method 属性值修改为 get(即把表单数据的提交方式改为 get 请求)：

```
< form action = "RegisterCheckServlet" method = "get">
```

再填写如下信息后单击“提交”按钮。

用户名：administrator。

密码：administrator。

勾选：“C 语言”“Java 语言”复选框。

可以发现地址栏中直接以查询字符串(query string)的形式显示了用户填写和勾选的信息：

```
http://localhost:8080/Chapter05_servlet/RegisterCheckServlet?username = administrator&password = administrator&language = C % 2B % 2B % E8 % AF % AD % E8 % A8 % 80&language = Java % E8 % AF % AD % E8 % A8 % 80
```

甚至连用户密码也被直接泄露，因此对于 get 请求要慎用。当然，对于一些可公开的信息，采用 get 请求和查询字符串来向服务器提交发送数据是比较方便的，因为不用专门去设计表单来存储数据。

查询字符串是 URL 中用于向服务器传递附加数据的一部分，位于问号(?)之后，由一系列键值对组成，常用于筛选和分页等应用场景(如 https://example.com/search?keyword=book&page=2 中，keyword=book&page=2 即为查询字符串)。其基本语法规则为：以“键=值”(key=value)表示单个参数，多个参数用 & 连接；若参数值包含特殊字符(如空格、?、&等)，需进行 URL 编码(如空格编码为%20)。

▶ 5.5.2　读取请求头

除了用户填写、勾选的信息之外，如果还想了解客户端浏览器类型、可接受的数据传输格式和语言等信息用于兼容适配不同浏览器和语言，此时需要读取 HTTP 请求头中的数据信息，常见读取方法如表 5-6 所示。

表 5-6　常见读取方法

方法原型	功　能	返回值类型
getHeader(String name)	根据参数名 name 获取对应的头字段值	String
getHeaders(String name)	根据参数名 name 获取对应的所有值，返回一个字符串数组，适用于多值参数(如复选框、多选列表等)	Enumeration < String >
getHeaderNames( )	获取所有头字段名称	Enumeration < String >

例 5-7 展示了请求头的读取应用，运行效果如图 5-16 所示。

例 5-7　HeaderReaderServlet.java

```
package com.myoj.service;

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.*;
import java.util.Enumeration;

@WebServlet("/HeaderReaderServlet")
public class HeaderReaderServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        //获取所有请求头名称并遍历输出
        Enumeration<String> headerNames = request.getHeaderNames();
        out.println("<h3>所有请求头信息:</h3>");
        while (headerNames.hasMoreElements()) {
            String headerName = headerNames.nextElement();
            String headerValue = request.getHeader(headerName);
            out.println(headerName + " : " + headerValue + "<br>");
        }

        //单独获取指定请求头
        String userAgent = request.getHeader("User-Agent");
        out.println("<h3>User-Agent:</h3>" + userAgent);
        //获取 Accept 头的所有值
        Enumeration<String> acceptHeaders = request.getHeaders("Accept");
        out.println("<h3>Accept 头的所有值:</h3>");
        while (acceptHeaders.hasMoreElements()) {
            String acceptValue = acceptHeaders.nextElement();
            out.println(acceptValue + "<br>");
        }
    }
}
```

其中,User-Agent 即为浏览器兼容的类型,Accept 头的所有值既可以通过 getHeader(String name)来读取,也可以通过 getHeaders(String name)来读取。

▶ 5.5.3 读取客户端信息

除了请求头信息的读取方法,HttpServletRequest 还内置了一些方法读取客户端标识、网络连接、请求路径、协议类型等参数信息,以下是主要方法说明。

1. **获取客户端网络信息**

- StringgetRemoteAddr():返回发送请求的客户端 IP 地址。
- int getRemotePort():返回发送请求的客户端端口号。
- StringgetRemoteHost():返回发送请求的客户端主机名(也有可能是 IP 地址的反向解析结果)。

2. **获取服务器网络信息。**

- StringgetLocalAddr():返回接收请求的服务器 IP 地址。
- int LocalPort():返回接收请求的服务器端口号。

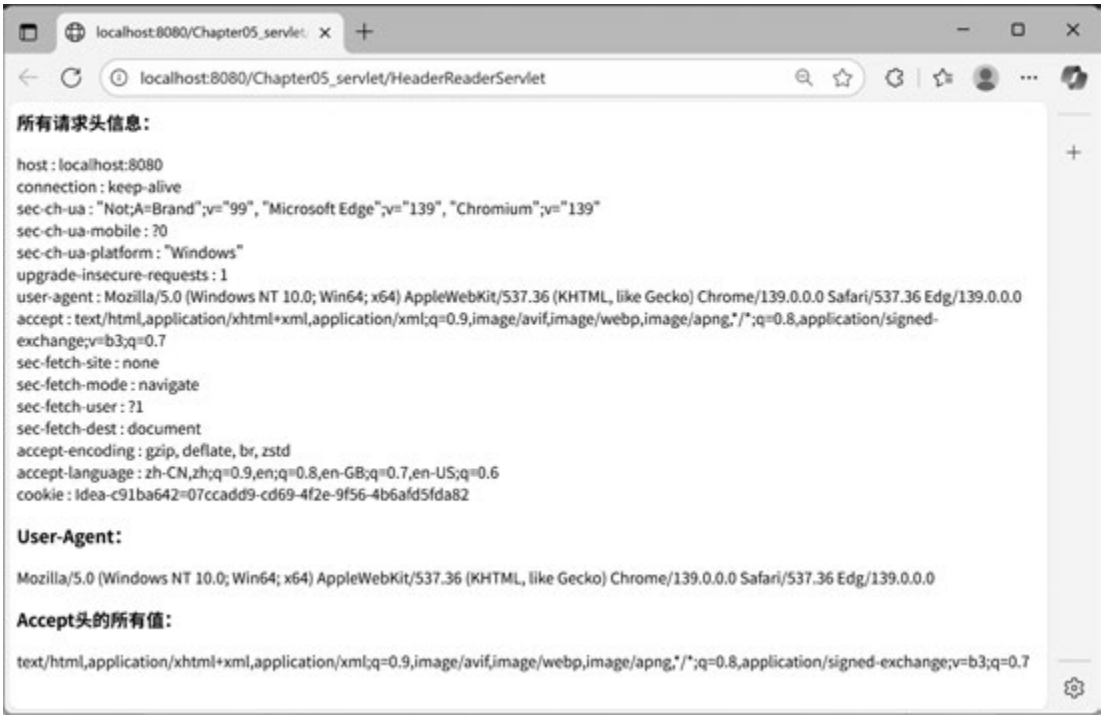

图 5-16　请求头读取示例

- StringgetLocalName()：返回接收请求的服务器主机名。

3. **获取请求方法与路径信息**

- StringgetMethod()：返回请求使用的 HTTP 方法(如 GET、POST、PUT 等)。
- StringgetRequestURI()：返回请求的 URI(包含上下文路径和资源路径，如/myoj/problems)。
- StringgetRequestURL()：返回请求的完整 URL 字符串(如 http://example.com/app/user)。
- StringgetContextPath()：返回当前 Web 应用的上下文路径(如/myoj)。
- StringgetServletPath()：返回 Servlet 的路径(如/contests)。
- StringgetPathInfo()：返回请求 URI 中 Servlet 路径之后的额外路径信息，若不存在则返回 null。

4. **获取协议信息**

- StringgetProtocol()：返回请求使用的协议及版本(如 HTTP/1.1)。
- StringgetScheme()：返回请求使用的协议名称(如 http 或 https)。

例 5-8 展示了这些方法的使用。

例 5-8　ClientServerInfo.java

```
package com.myoj.service;

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.*;

@WebServlet("/ClientServerInfo")
public class ClientServerInfo extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        //1. 客户端网络信息
        out.println("<h3>客户端网络信息</h3>");
        out.println("客户端 IP 地址:" + request.getRemoteAddr() + "<br>");
        out.println("客户端端口:" + request.getRemotePort() + "<br>");
        out.println("客户端主机名:" + request.getRemoteHost() + "<br>");

        //2. 服务器网络信息
        out.println("<h3>服务器网络信息</h3>");
        out.println("服务器 IP 地址:" + request.getLocalAddr() + "<br>");
        out.println("服务器端口:" + request.getLocalPort() + "<br>");
        out.println("服务器主机名:" + request.getLocalName() + "<br>");

        //3. 请求方法与路径信息
        out.println("<h3>请求方法与路径信息</h3>");
        out.println("HTTP 请求方法:" + request.getMethod() + "<br>");
        out.println("请求 URI:" + request.getRequestURI() + "<br>");
        out.println("完整请求 URL:" + request.getRequestURL() + "<br>");
        out.println("上下文路径:" + request.getContextPath() + "<br>");
        out.println("Servlet 路径:" + request.getServletPath() + "<br>");
        out.println("额外路径信息:" + (request.getPathInfo() != null ? request.getPathInfo
() : "无") + "<br>");

        //4. 协议与会话信息
        out.println("<h3>协议与会话信息</h3>");
        out.println("协议及版本:" + request.getProtocol() + "<br>");
        out.println("协议名称:" + request.getScheme() + "<br>");
    }
}
```

运行效果如图 5-17 所示。

上述例子有两点需要注意：第一点是客户端端口为 63807，这个端口是与服务器通信的实际端口，服务器端口 8080 只是用来接收客户端连接请求的入口；第二点是 HTTP 的版本是 1.1，虽然 HTTP/2 和 HTTP/3 在性能和安全性上有显著提升，但受限于新版本的普及速度和现有网络环境多重因素的制约，HTTP/1.1 仍是被广泛默认使用的版本，全面替代 HTTP/1.1 仍需等待硬件、软件、运维体系的协同进化。

▶ 5.5.4 请求转发

考虑实际场景：当用户注册成功后，须直接跳转到用户中心并显示欢迎信息："XXX，您有 Y 种喜欢的编程语言，欢迎您加入程序员大家庭"。由于模块化松耦合的要求，开发者不能把所有功能都放在一个 RegisterCheckServlet 中，须另外设计一个 UserCenterServlet 来完成该欢迎功能，此时就需要在这两个 Servlet 之间传递转发 HTTP 请求和响应对象，以便共享用

图 5-17　客户端、服务器和网络连接等信息读取示例

户名信息 XXX，并且这种功能转移是在服务器内部完成的，不应当让客户端感知到这种变化，这样一个过程即可看作请求转发(request forwarding)。

一般地，请求转发是指当客户端发送请求到 ServletA 后，ServletA 不直接生成响应，而是通过服务器将请求“转发”给另一个资源(如 ServletB 或 JSP 页面)，由后者完成最终的响应生成。整个过程中，客户端只发送一次请求，服务器内部完成资源跳转且只做一次响应，客户端地址栏的 URL 保持不变，且转发前后的 Servlet 之间共享同一个请求对象(HttpServletRequest)和响应对象(HttpServletResponse)。

请求转发的实现步骤如下。

1. **获取 RequestDispatcher 对象**

ServletA 通过 HttpServletRequest 的 getRequestDispatcher(String path)方法获取转发器对象，参数 path 为目标资源的路径。

2. **存储转发所需数据(可选步骤)**

若需向目标资源 ServletB 传递处理后的数据，可在 ServletA 通过 request.setAttribute(String name, Object value)方法将数据存入请求作用域(这类数据仅在请求转发的多个 Servlet 之间有效)。

3. **执行转发**

调用 RequestDispatcher 转发器对象的 forward(ServletRequest request, ServletResponse response)方法，将请求和响应对象传递给目标资源 ServletB，完成跳转。

例 5-9～例 5-11 展示了用户注册成功后并转发到用户中心显示欢迎信息的功能，由于大部分代码和前面注册功能是一致的，因此本例中只展示代码的不同之处。

例 5-9　signup1. html

```
…
< form action = "RegisterCheckServlet1"method = "post">
…
```

例 5-10　RegisterCheckServlet1. java

```
…
@WebServlet(name = "RegisterCheckServlet1", urlPatterns = {"/RegisterCheckServlet1"})
public class RegisterCheckServlet1 extends HttpServlet {
    …
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
…
      //验证用户名和密码合法性
      boolean isUsernameValid = username != null && username.length() >= 6;
      boolean isPasswordValid = password != null && password.length() >= 8;
       if(isUsernameValid&&isPasswordValid) {
           String dest = "UserCenterInfo";
           request.setAttribute("count",languages.length);
           RequestDispatcher rd = request.getRequestDispatcher(dest);
           rd.forward(request, response);
        }
}
```

例 5-11　UserCenterServlet. java

```
package com.myoj.service;

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;

@WebServlet("/UserCenterInfo")
public class UserCenterServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        int count = (int)request.getAttribute("count");
        String username = (String)request.getParameter("username");
        response.setContentType("text/html;charset = UTF - 8");
        PrintWriter out = response.getWriter();
        out.println(username + ",您有" + count + "种喜欢的编程语言,欢迎您加入程序员大
家庭!");
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request, response);
    }
}
```

上述例子中，用户在 signup1.html 页面填写信息，然后提交到 RegisterCheckServlet1 进行用户名和密码校验，并统计喜欢的语言数量，然后将请求转发到 UserCenterServlet 显示欢迎信息。整个过程中，用户只能看到 RegisterCheckServlet1 对应的 URL，并不能感知到 UserCenterServlet 参与了工作，具体运行效果如图 5-18 所示。

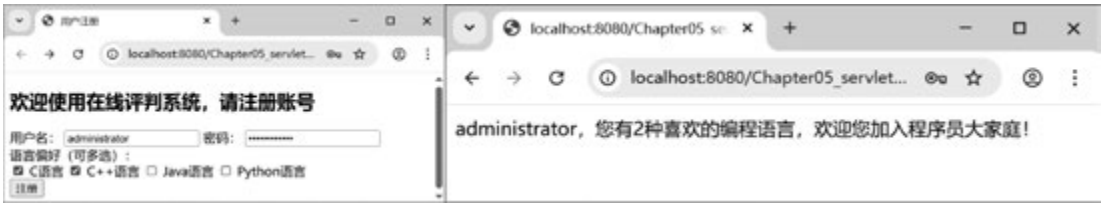

图 5-18　请发转发示例

5.6　设置处理 HTTP 响应

迄今为止我们接触到的所有功能和例子都是向客户端输出的内容是 HTML 文档，但在实际应用中常常还会出现其他功能要求，例如向客户端输出 Excel、Word、PDF 和压缩包等其他类型的资源文件、提供文件下载以及通知客户端所请求的资源不存在或被转移到其他地方等，这些工作都需要通过 HttpServletResponse 接口对象来完成。

▶ 5.6.1　设置响应内容类型

首先来看一个新功能：生成当前参赛人员的动态排名，并以 Excel 文档形式显示。要实现这个功能，须通过 HttpServletResponse 接口对象的 setContentType()方法设置响应内容的类型为 Excel 类型，即通知客户端浏览准备好接收 Excel 文档并显示，setContentType()方法的原型及说明如下：

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

该方法用于指定 Servlet 返回给客户端的响应数据的 MIME 类型，同时可指定字符编码格式。客户端浏览器会根据 MIME 类型决定如何解析或下载数据（例如解析为 HTML、Excel、PDF 或下载为文件或播放音频）。

参数 type：类型格式，通常为“主类型/子类型；charset＝编码格式”，其中主类型/子类型表示 MIME 类型，charset 用于指定字符编码（可选）。

必须注意的是，要让此方法生效，需在获取响应输出流 response.getWriter()之前调用，否则会导致设置失败。

例 5-12 展示了利用 setContentType()方法来实现动态排名的 Excel 输出。

例 5-12　ShowExcelRank.java

```
package com.myoj.service;

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;

@WebServlet(name = "ShowExcelRank", urlPatterns = {"/showExcelRank"})
public class ShowExcelRank extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        //1. 设置响应内容类型为 Excel,指定 UTF-8 编码
        response.setContentType("application/vnd.ms-excel;charset=UTF-8");

        //2. 设置响应头,指定内联显示并处理中文
        String fileName = "contest_rank.xls";
        response.setHeader("Content-Disposition", "inline; filename=\"" + fileName + "\"");

        //3. 获取输出流
        PrintWriter out = response.getWriter();

        Object[][] contestants = {
            {1, "上海交通大学", 8, 300},
            {2, "北京大学", 7, 280},
            {3, "浙江大学", 7, 320},
        };

        //4. 生成 Excel 表格内容
        out.println("<html>");
        out.println("<head>");
        //HTML 元标签指定编码,增强兼容性
        out.println("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=
UTF-8\">");
        out.println("</head>");
        out.println("<body>");

        //表格开始
        out.println("<table border='1'>");
        out.println("<caption>实时竞赛排名</caption>");
        //表头行
        out.println("<tr>");
        out.println("<th>排名</th>");
        out.println("<th>姓名</th>");
        out.println("<th>解题数</th>");
        out.println("<th>罚时(分钟)</th>");
        out.println("</tr>");

        //循环输出参赛人员数据
        for (Object[] contestant : contestants) {
            out.println("<tr>");
            out.println("<td>" + contestant[0] + "</td>");
            out.println("<td>" + contestant[1] + "</td>");
            out.println("<td>" + contestant[2] + "</td>");
            out.println("<td>" + contestant[3] + "</td>");
            out.println("</tr>");
        }
```

```
            //表格结束
            out.println("</table>");
            out.println("</body>");
            out.println("</html>");

            //刷新并关闭输出流
            out.flush();
            out.close();
        }

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

打开浏览器，访问 http://localhost:8080/Chapter05_servlet/showExcelRank，可以看到会直接下载 Excel 文件、直接打开 Excel 文件或弹出下载保存 Excel 文件的提示框（根据浏览器设置不同），下载完成后打开该 Excel 文件如图 5-19 所示。

图 5-19　利用 Servlet 输出的排名 Excel 文件

下面分析下上述代码中的关键语句。

“response.setContentType("application/vnd.ms-excel;charset=UTF-8");”表示设置 MIME 类型为 Excel，字符集使用 UTF-8 编码，即通知客户端准备接收编码格式为 UTF-8 的 Excel 文件。

“response.setHeader("Content-Disposition","inline;filename=\""+fileName+"\"");”中出现了一个新方法 setHeader()，顾名思义，该方法的作用是设置响应头：响应头类型为 Content-Disposition，即通知客户端浏览器处理响应内容的方法；类型值"inline;filename=\""+fileName+"\""中的 inline 表示直接打开该文件（如浏览器设置了可直接打开），filename 表示打开或下载的默认文件名。

接下来的多条语句是以 HTML 表格的形式输出 Excel 文档内容。

除了目前已经见到的 text/html、application/vnd.ms-excel 的 MIME 类型外，其他常见的 MIME 类型如表 5-7 所示。

表 5-7　常见的 MIME 类型

类别	MIME 类型	对应扩展名	用 途 说 明
文本类	text/plain	.txt	纯文本文件
	text/xml	.xml	可扩展标记语言文件，用于数据存储和交换（如配置文件、接口数据）
图像类	image/jpeg	.jpg、.jpeg	JPEG 图像格式
	image/png	.png	PNG 图像格式，支持透明通道和无损压缩，适用于图标、Logo、截图（质量更高）
	image/gif	.gif	GIF 图像格式，支持简单动画和透明色，适用于表情包、简单动态图标
音频类	audio/mpeg	.mp3	MP3 音频格式，主流音频压缩格式，兼容性强（如音乐、语音）
	audio/wav	.wav	WAV 音频格式，无损音质，文件体积大（如原始录音、音效）
	audio/ogg	.ogg	OGG 音频格式，开源压缩格式，支持多声道（适合流媒体）
视频类	video/mp4	.mp4	MP4 视频格式，主流视频压缩格式，兼容性强（如电影、短视频）
	video/webm	.webm	WebM 视频格式，开源格式，适合网页端播放（体积小，加载快）
	video/quicktime	.mov	QuickTime 视频格式，苹果公司开发，适合专业视频编辑
文档类	application/pdf	.pdf	PDF 文档格式，跨平台固定版式文档（如合同、说明书、电子书）
	application/vnd.openxmlformats-officedocument.wordprocessingml.document	.docx	新版 Word 文档格式（Office 2007 及以后，基于 XML）；早期 Word 文档格式（Office 2003 及以前）为 application/msword(.doc)
	application/vnd.openxmlformats-officedocument.spreadsheetml.sheet	.xlsx	新版 Excel 表格格式（Office 2007 及以后，基于 XML）；早期 Excel 表格格式（Office 2003 及以前）为 application/vnd.ms-excel(.xls)
压缩包类	application/zip	.zip	ZIP 压缩包格式，最常用的跨平台压缩格式（可包含多个文件/文件夹）
	application/rar	.rar	RAR 压缩包格式，压缩率较高（私有格式，需专用软件解压）
数据交换类	application/json	.json	JSON 格式文件，轻量级数据交换格式（前后端接口、配置文件常用）
	multipart/form-data	无	表单上传文件时的格式，支持二进制数据（如图片、文档上传）
通用二进制类	application/octet-stream	无	未知类型的二进制文件，浏览器默认触发下载（如.exe、.dll、.class）

▶ 5.6.2　设置响应头

例 5-12 中使用的响应头设置方法 setHeader()还有更多的灵活用途，下面结合 setHeader()方法原型说明来阐述它的功能。

```
public void setHeader(String name, String value);
```

其中，参数 name 为响应头的名称（大小写不敏感）；参数 value 为响应头的值，根据不同的头名称，值的格式和含义不同。该方法用于向 HTTP 响应中添加一个指定名称和值的响应头。如果同名响应头已存在，则会覆盖原有值。

1. **设置内容响应类型**

```
response.setHeader("Content-Type", "text/html;charset=UTF-8");
```

该语句功能完全等价于 response.setContentType("text/html;charset=UTF-8")，实际上 setContentType()方法是设置响应头的便捷封装。

2. **控制缓存行为**

```
response.setHeader("Cache-Control", "no-store, no-cache, must-revalidate");
```

该语句禁止浏览器缓存当前响应，用于须保证数据新鲜度的验证码、实时行情等情形。no-store 表示禁止浏览器存储任何响应内容；no-cache 表示浏览器需向服务器验证后才能使用缓存；must-revalidate 用于控制浏览器或中间缓存（如代理服务器）对缓存内容的使用规则，核心作用是强制缓存必须先向服务器验证内容有效性，只有在服务器确认内容未过期后，才能使用缓存。这三者组合起来用于确保数据的实时更新。

3. **控制页面刷新**

```
response.setHeader("Refresh", "3;url=HttpServerTime");
```

该语句表示每隔 3s 自动刷新跳转到 HttpServerTime 的 URL。URL 可使用相对路径或绝对路径。

▶ 5.6.3　响应重定向

在 Web 应用开发中，页面跳转是常见的功能需求，例如用户注册失败后重新跳转到注册页面、使用第三方账号登录、到第三方网站浏览竞赛题目、到第三方支付进行订单支付等。这类需求常采用响应重定向（redirect）来实现。

响应重定向是指服务器在接收到客户端请求后，不直接处理请求，而是通过 HTTP 响应头告知客户端“需要重新向另一个 URL 发起请求”，由客户端（浏览器）自动跳转到新的 URL。响应重定向通过 HttpServletResponse 接口的 sendRedirect()方法实现，方法原型如下：

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

其中，参数 location 为跳转的目标 URL，可以是相对路径（如/registerCheckServlet）或绝对路径（如 https://myoj.com/home）等。

下面利用 sendRedirect()方法对 signup.html 页面和 RegisterCheckServlet1 的注册功能进行简单修改：当注册失败时自动重定向到注册页面。

例 5-13　signup2.html

```
…
<form action="RegisterCheckServlet2" method="post">
…
```

例 5-14　RegisterCheckServlet2.java

```
…
@WebServlet(name = "RegisterCheckServlet2", urlPatterns = {"/RegisterCheckServlet2"})
```

```
public class RegisterCheckServlet2 extends HttpServlet {
    …
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    …
      //验证用户名和密码合法性
      boolean isUsernameValid = username != null && username.length() >= 6;
      boolean isPasswordValid = password != null && password.length() >= 8;
      if(!isUsernameValid || !isPasswordValid) {
         System.out.println("注册失败!用户名和密码不合法");
         String location = "signup2.html";
         response.sendRedirect(location);
       }
}
```

其中,“System.out.println("注册失败! 用户名和密码不合法");”语句用于向编程工具的控制台输出调试信息(例如 Eclipse),在注册页面上并不会显示。

而“response.sendRedirect(location);”语句实际上是如下两个响应头按序设置的便捷封装:

```
response.setStatus(302);                          //设置临时重定向的状态码
response.setHeader("Location", "signup2.html ");  //设置目标 URL
```

这两条语句表示服务器在处理请求时,会首先发送响应状态码 302 给客户端浏览器,说明“本次响应是一个重定向指令”,然后通过 setHeader("Location", "signup2.html")告知浏览器“重定向的目标地址是 signup2.html”。浏览器收到响应后,会自动发起第二次请求,访问 signup2.html 页面,最终显示该页面内容,同时浏览器的地址栏内容也会变为 signup2.html。

注意,虽然请求转发(request forwarding)和响应重定向(response redirect)都可以用于实现对目标 URL 的跳转,但本质上它们是完全不同的,具体对比如表 5-8 所示。

表 5-8 请求转发和响应重定向的对比

对比项目	请 求 转 发	响应重定向
本质	服务器内部的请求资源跳转(客户端无感知)	服务器告知客户端重新请求新资源(客户端主动跳转)
请求响应次数	一次请求和一次响应(客户端仅发起一次请求,服务器内部转发)	两次请求和两次响应(客户端先请求原始资源,再根据服务器指令请求新资源)
浏览器地址栏	显示原始请求 URL(地址不变)	显示目标资源 URL(地址会更新)
数据共享	可通过 request.setAttribute()方法共享数据	不能通过 request.setAttribute()方法共享数据,只能通过查询字符串传递参数
跳转范围	仅能跳转到当前 Web 应用内的资源(同一上下文)	可跳转到任意 URL(包括当前应用、其他应用、外部网站)
状态码	无特殊状态码(默认返回 200 OK)	需返回重定向状态码(302 为临时重定向、301 为永久重定向)
适用场景	同一应用内的页面跳转,需共享 request 对象的相关数据	表单提交后跳转(避免重复提交)、跨应用跳转、权限验证失败跳转

▶ 5.6.4 设置响应码和自定义错误消息

5.6.3 节我们接触到了用于设定响应状态码的方法 response.setStatus()。响应状态码

是服务器端对客户端请求的处理结果的数字标识。通过状态码，客户端可以快速判断请求是否成功、失败原因以及需要采取的后续操作。

response.setStatus()的语法原型如下：

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

其中，参数 sc 为表示 HTTP 状态码的整数(取值范围为 100～599)。

常见状态码的常量与应用场景如表 5-9 所示。

表 5-9　常见状态码的常量与应用场景

状态码	HttpServletResponse 常量	含义	应用场景
200	SC_OK	请求成功	正常处理并返回数据(如查询用户信息、提交成功)
302	SC_FOUND	临时重定向	页面跳转(如登录后跳转到首页、权限不足跳转到登录页)
400	SC_BAD_REQUEST	客户端请求错误	表单参数缺失或格式错误(如提交空数据)
401	SC_UNAUTHORIZED	未授权	未登录用户访问需要权限的资源(如未登录查看个人信息)
403	SC_FORBIDDEN	禁止访问	已登录但无权限访问(如普通用户访问管理员后台)
404	SC_NOT_FOUND	资源不存在	请求的 URL 对应的资源不存在(如访问不存在的资源)
500	SC_INTERNAL_SERVER_ERROR	服务器内部错误	服务器处理时发生异常(如代码判题时出现程序错误)

例 5-15 展示了状态码 400 和 404 的应用：检查题目是否存在和用户提交的代码是否为空，并给出错误提示。

例 5-15　ProblemHandlerServlet.java

```
@WebServlet("/ProblemHandlerServlet")
public class ProblemHandlerServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        //设置响应内容类型为 HTML(支持中文)
        response.setContentType("text/html;charset=UTF-8");
        //1. 获取请求参数
        int problemId = Integer.parseInt(request.getParameter("problemId"));
        String code = request.getParameter("code");
        PrintWriter out = response.getWriter();
        //2. 检查题目是否存在(404 错误场景)
        if (problemId<1000 || problemId>2000) { //假设 ID 为 1001～2000 之外的题目都不存在
            //使用 sendError()方法发送 404 错误及提示信息
            response.sendError(
                HttpServletResponse.SC_NOT_FOUND,
                "错误:ID 为【" + problemId + "】的题目不存在,请检查题目 ID 是否正确"
            );
            return;        //终止后续处理
        }
        //3. 检查提交的代码是否为空(400 错误场景)
        if (code == null || code.trim().isEmpty()) {
```

```
                //先使用 setStatus()设置 400 状态码
                response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
                //手动输出错误提示内容
                out.println("<html>");
                out.println("<head><title>提交错误</title></head>");
                out.println("<body>");
                out.println("<h2 style='color:red'>400 Bad Request</h2>");
                out.println("<p>错误提示:提交的代码不能为空,请输入有效的代码后重新提交</p>");
                out.println("</body></html>");
                return; //终止后续处理
            }
            //4. 验证通过,正常处理提交,等待判题
            out.println("<html>");
            out.println("<body>");
            out.println("<h3>提交成功!</h3>");
            out.println("<p>题目 ID:" + problemId + "</p>");
            out.println("<p>代码已接收,正在判题中...</p>");
            out.println("</body></html>");
        }
        @Override
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            doPost(request,response);
        }
    }
```

在地址栏分别输入如下 3 种 URL,可以看到相应的结果分别如图 5-20～图 5-22 所示。

(1) http://localhost:8080/Chapter05_servlet/ProblemHandlerServlet?problemId=1999&code=main()。

图 5-20　提交正常题目和代码的结果

(2) http://localhost:8080/Chapter05_servlet/ProblemHandlerServlet? problemId=1999&code=。

此处的自定义错误消息采用了 response 的新方法 sendError()实现,其原型和使用如下:

```
public void sendError(int sc[, String msg])
```

其中,参数 sc 为状态码,msg 为错误提示信息(可省略)。该方法会自动生成一个默认的错误页面(包含状态码和信息)。若不指定 msg,则使用状态码的默认描述(如 404 对应 "Not Found")。

(3) http://localhost:8080/Chapter05_servlet/ProblemHandlerServlet?problemId=2999&code=main()。

图 5-21　提交正常题目但代码为空的结果

图 5-22　提交题目不存在的结果

5.7　web.xml、注解和部署 Web 应用程序

随着学习的深入，所编写的 Servlet 数量已经很多，有点杂乱了，仅使用@webservlet 注解来配置 Servlet 的映射 URL 也有不便之处，当需要改变映射 URL 时，需要修改 Servlet 的源代码文件并重新编译部署上线才能生效，维护代价很高。那么如何高效地让开发人员和 Web 容器(如 Tomcat)统一配置、管理和维护众多 Servlet 组件？这就需要引入 Web 应用程序的核心配置文件 web.xml，也称为部署描述文件(deployment descriptor)。

▶ 5.7.1　部署描述文件 web.xml

web.xml 文件是以 XML 格式存储，用于定义 Web 应用程序的初始化参数、资源映射和组件关系的文件。它是 Web 容器识别和管理各类应用组件的依据，必须放置在 Web 应用程序的 WEB-INF 目录下(路径：WEB-INF/web.xml)。web.xml 文件可以看作 Web 应用程序与 Web 容器之间建立的标准化“合同”，规定了如何解决组件管理、解耦组件配置与组件代码以及跨环境部署等关键问题，其主要功能如下。

(1) 配置 Servlet、监听器、过滤器(后续将会学习)等组件的基本信息、初始化参数和 URL 映射。

(2) 配置 Web 应用程序的全局初始化参数。

(3) 配置欢迎页、错误页等全局应用特性。

(4) 配置引用外部资源(如数据库等)。

web.xml 文件的常规配置如例 5-16 所示，包括了 Servlet 初始化参数和 URL 映射配置、Web 应用程序全局参数配置、欢迎页和错误页配置。

例 5-16　web.xml

```
<?xml version = "1.0"encoding = "UTF - 8"?>
< web - app xmlns = "http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema - instance"
    xsi:schemaLocation = "http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web - app_4_0.xsd"
    version = "4.0"
    metadata - complete = "false">
    <!-- Servlet 配置 -->
    < servlet >
        < servlet - name > JudgeServlet </servlet - name > <!-- Servlet 唯一标识 -->
        < servlet - class > com.myoj.service.JudgeServlet </servlet - class > <!-- 完全类名 -->
        <!-- 当前 Servlet 的初始化参数 -->
        < init - param >
            < param - name > maxCodeSize </param - name >
            < param - value > 10 240 </param - value > <!-- 代码最大长度:10 240 字节 -->
        </init - param >
    </servlet >
    < servlet - mapping > <!-- 配置 Servlet 映射的 URL -->
        < servlet - name > JudgeServlet </servlet - name > <!-- 关联的 Servlet 名称 -->
        < url - pattern >/judge </url - pattern >
    </servlet - mapping >
    <!-- Web 应用程序全局参数配置 -->
    < context - param >
        < param - name > ojName </param - name >
        < param - value >在线判题系统</param - value > <!-- 系统名称 -->
    </context - param >
    <!-- 欢迎页 -->
    < welcome - file - list >
        < welcome - file > index.html </welcome - file >
        < welcome - file > home.html </welcome - file >
    </welcome - file - list >
    <!-- 错误页 -->
    < error - page >
        < error - code > 404 </error - code > <!-- 404:资源不存在 -->
        < location >/404.html </location >
    </error - page >
</web - app >
```

下面围绕这这份 web.xml 来讲解其中的元素配置和语法。

1. <? xml version = "1.0" encoding = "UTF-8"? >

它是 XML 文件的声明语句,用于通知解析器该文件格式为 XML 文件,字符编码格式为 UTF-8。该声明必须位于 XML 文件的第一行,且前面不能有任何空格、换行或其他字符。

2. < web-app >元素及其子元素

< web-app >是 Web 应用部署描述文件(web.xml)的根元素,用于定义整个 Web 应用程序的配置信息,是 Web 容器解析应用配置的入口,其属性说明如下。

xmlns="http://xmlns.jcp.org/xml/ns/javaee"声明了 XML 命名空间(namespace),指定当前配置遵循 Java EE 规范,避免标签命名冲突。

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"引入了 XML Schema 实例命名空间,用于指定 Schema 约束文件的位置。

xsi:schemaLocation 定义 XML Schema 文件的位置,确保配置符合 Servlet 规范的语法规则(如 web-app_4_0.xsd 对应 Servlet4.0 规范)。

version="4.0"指定当前 Web 应用遵循 Servlet4.0 规范版本。

metadata-complete="false"标识是否忽略注解配置。默认值为 false，表示允许注解（如@WebServlet 等）与 web.xml 共同生效，但发生冲突时 web.xml 中的配置优先；如果为 true 则表示仅使用 web.xml 的配置。

<web-app>元素的子元素如表 5-10 所示。

表 5-10　<web-app>元素的子元素

子　元　素	功 能 描 述
<display-name>	定义 Web 应用的名称
<servlet>	定义 Servlet 组件的基本信息，包括 Servlet 名称、完全类名、初始化参数等
<servlet-mapping>	将<servlet>中定义的 Servlet 映射到指定 URL 路径，使容器能根据请求路径找到对应的 Servlet
<filter>	定义过滤器组件（拦截请求/响应），包括过滤器名称、全类名、初始化参数，常用于权限验证、字符编码处理
<filter-mapping>	将<filter>中定义的过滤器映射到指定 URL 路径或 Servlet，指定拦截规则（例如/* 表示拦截所有请求）
<listener>	注册 Web 监听器，只需指定监听器的全类名
<context-param>	定义 Web 应用的全局初始化参数（如系统名称、数据库地址），所有组件可通过 ServletContext 获取
<welcome-file-list>	定义应用的默认首页列表（欢迎页），当访问应用根路径（如 http://localhost:8080/myoj）时，容器会自动查找列表中的页面（如 index.html）
<error-page>	配置错误状态码（如 404、500）或异常类型与对应错误页面的映射，当发生指定错误时自动跳转至配置的页面
<session-config>	配置会话（session）的全局属性，如会话超时时间（session-timeout，单位为分钟）、Cookie 配置等
<mime-mapping>	定义文件扩展名与 MIME 类型的映射（如.html 对应 text/html），确保浏览器正确解析响应内容
<security-constraint>	配置 Web 应用的安全约束，如限制特定 URL 的访问权限（需结合<auth-constraint>等子元素）
<login-config>	配置身份验证方式（如表单登录、基本认证）及登录页面、错误页面的路径
<resource-ref>	声明对外部资源的引用（如数据库连接池），指定资源类型、名称等信息

3. <servlet>和<servlet-mapping>元素及其子元素

在 Web 应用部署描述文件 web.xml 中，<servlet>和<servlet-mapping>是配置 Servlet 组件的核心元素，二者配合实现 Servlet 的定义与 URL 映射，使 Web 容器能根据请求路径找到对应的 Servlet 进行处理。通俗地说，<servlet>负责通知 Web 容器“Servlet 是谁以及如何初始化”；<servlet-mapping>负责通知容器“Servlet 处理哪些 URL 请求”，两者必须通过相同的<servlet-name>子元素关联，一起完成 Servlet 从定义到访问的配置闭环。

<servlet>的子元素如表 5-11 所示。

表 5-11　<servlet>的子元素

子　元　素	语 法 描 述
<servlet-name>	必选子元素。定义 Servlet 的唯一名称（字符串，如 JudgeServlet），用于与<servlet-mapping>关联。 示例：<servlet-name>JudgeServlet</servlet-name>

续表

子 元 素	语 法 描 述
< servlet-class >	必选子元素。指定 Servlet 的完全类名(包名＋类名),Web 容器通过该类名反射创建 Servlet 实例。 示例: < servlet-class > com. myoj. service. JudgeServlet </servlet-class >
< init-param >	可选子元素。定义当前 Servlet 的初始化参数(键值对),通过当前 Servlet 的 ServletConfig 获取。< init-param >包含如下子元素。 －< param－name >:参数名(如 maxCodeSize); －< param－value >:参数值(如 10240). 示例: < init－param > < param－name > maxCodeSize </param－name > < param－value > 10240 </param－value > </init－param >
< load-on-startup >	可选子元素。指定 Servlet 的加载时机(预装载或延迟装载)。 • 取值为非负整数: Web 应用启动时按照该值依次顺序加载 Servlet,值越小优先级越高,此时称为 servlet 的预装载。 • 省略或为负值: 首次被请求时才加载,此时称为 Servlet 的延迟装载。 示例: < load-on-startup > 2 </load-on-startup >
< async-supported >	可选子元素。声明 Servlet 是否支持异步处理(true/false),默认为 false。 示例: < async-supported > true </async-supported >

< servlet-mapping >元素的功能是建立 Servlet 与 URL 路径的映射关系,其子元素如表 5-12 所示。

表 5-12 < servlet-mapping >的子元素

子 元 素	语 法 描 述
< servlet-name >	必选子元素。关联< servlet >中定义的 Servlet 名称(必须与< servlet >的< servlet-name >完全一致),实现 Servlet 与 URL 的绑定。 示例: < servlet-name > JudgeServlet </servlet-name >
< url-pattern >	必选子元素。定义 Servlet 匹配的 URL 路径,支持多种匹配规则。 (1) 精确匹配: 如/judge(仅匹配 http://域名/应用程序名/judge)。 (2) 目录匹配: 如/user/ *(匹配/user 下的所有路径,如/user/login)。 (3) 扩展名匹配: 如 *. do(匹配所有以. do 结尾的路径,如/submit. do)。 如果出现多个 servlet 都匹配某 URL,则按上述顺序选择匹配最长路径的 servlet。 示例: < url-pattern >/judge </url-pattern >

4. < context-param >元素及其子元素

< context-param >用于定义 Web 应用级别的全局初始化参数,这些参数对整个 Web 应用的所有组件(包括所有 Servlet、过滤器、监听器等)均可见,主要作用包括:

(1) 存储应用的全局配置信息(如系统名称、版本号、数据库连接地址等)。

(2) 实现跨组件的参数共享,避免在每个组件中重复定义相同参数。

(3) 便于集中管理和修改全局参数(无须修改代码,仅更新 web. xml 即可)。

< context-param >的子元素如表 5-13 所示。

表 5-13　<context-param>的子元素

子　元　素	语 法 描 述
<param-name>	必选子元素。定义全局参数的名称(唯一),使用易于理解的字符串。 示例:<param-name>ojName</param-name>
<param-value>	必选子元素。定义对应参数的值,可根据需求设置为字符串、数字、路径等(如在线判题系统)。 示例:<param-value>在线判题系统</param-value>

注意,<context-param>定义的是全局参数,这与 Servlet 定义的私有参数(<servlet>中的<init-param>)是不同的。全局参数须通过 ServletContext 接口对象的 getInitParameter(String name)方法获取,Servlet 私有参数仅当前 Servlet 可见,须通过 ServletConfig 接口对象的 getInitParameter(String name)方法获取。

例 5-17 展示了读取这两种参数的方法与区别。

例 5-17　JudgeServlet.java

```
package com.myoj.service;

import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletContext;
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;

public class JudgeServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        //设置响应内容类型及编码
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        //判题逻辑处理,待完成…

        //1. 获取 ServletContext 对象(全局上下文)
        ServletContext context = getServletContext();
        //读取<context-param>全局参数,参数名称要与配置文件一致(包括大小写)
        String ojName = context.getInitParameter("ojName");

        //2. 获取 ServletConfig 对象(当前 Servlet 配置)
        ServletConfig config = getServletConfig();
        // 读取<servlet>中的<init-param>参数,参数名称要与配置文件一致(包括大小写)
        String maxCodeSize = config.getInitParameter("maxCodeSize");

        //输出参数信息到页面
        out.println("<html>");
        out.println("<head><title>参数读取示例</title></head>");
        out.println("<body>");
        out.println("<h4>web.xml 参数读取结果</h3>");
```

```
            out.println("<h4>1. 全局参数(context-param)</h4>");
            out.println("<p>系统名称(ojName):" + (ojName != null ? ojName : "未配置") + "</p>");
            out.println("<h4>2. Servlet 私有参数(init-param)</h4>");
            out.println("<p>最大代码长度(maxCodeSize):" +
                    (maxCodeSize != null ? maxCodeSize + "字节" : "未配置") + "</p>");
            out.println("</body></html>");
        }
    }
```

参数读取结果如图 5-23 所示。

图 5-23　全局参数和 Servlet 初始化参数读取结果

5. <welcome-file-list>元素及其子元素

<welcome-file-list>是用于配置 Web 应用的默认首页(欢迎页)。当用户访问 Web 应用的根路径(如 http://localhost:8080/myoj/)而未指定具体页面时,Web 容器会自动查找该列表中的页面并返回第一个存在的页面。它仅有一个必选子元素<welcome-file>,注意事项如下。

(1) <welcome-file>中无须添加/前缀(如直接写 index.html,而非/index.html)。

(2) 列表中靠前的页面优先级更高,容器找到第一个存在的页面后会立即返回。

6. <error-page>元素及其子元素

<error-page>元素用于定义错误处理规则,即当 Web 应用发生特定错误(如 404 页面不存在、500 服务器内部错误)时,自动跳转到相应的错误页面,以提供统一错误处理风格和更友好的用户体验,而非直接显示系统默认的生硬错误提示页面,其主要功能包括以下两点。

(1) 捕获 Web 应用运行过程中出现的错误(包括 HTTP 状态码错误和运行异常 exception)。

(2) 将错误定向到自定义页面,避免向用户展示技术细节(如堆栈跟踪信息)。

<error-page>通过子元素的不同组合支持两种错误匹配方式:按 HTTP 状态码匹配和 exception 异常类型匹配,其子元素定义如表 5-14 所示。

表 5-14　<error-page>的子元素

子　元　素	语 法 描 述
<error-code>	它与<exception-type>只能选一个,用于指定 HTTP 状态码(如 404、500),当发生对应状态码的错误时,跳转到<location>定义的页面。 示例:<error-code>404</error-code>

续表

子　元　素	语 法 描 述
<exception-type>	它与<error-code>只能选一个，用于指定 Java 异常类型（完全类名），当应用抛出该类型（或其子类型）的未捕获异常时，跳转到<location>定义的页面。 示例：<exception-type>java.lang.NullPointerException</exception-type>
<location>	定义错误页面的路径（相对于 Web 应用根目录），当触发匹配的错误时，容器会转发请求到该页面，注意路径必须以“/”开头。 示例：<location>/404.html</location>

▶ 5.7.2　Annotation 注解机制

使用 web.xml 虽然能够统一松耦合地管理、配置和维护 Servlet 等组件，但对于开发人员却并不友好。

（1）配置冗余：一个简单的 Servlet 需要在 web.xml 中同时配置<servlet>和<servlet-mapping>标签，代码量庞大。

（2）分离性差：配置信息与 Java 代码完全分离，开发者需在 web.xml 和 Java 类之间频繁切换才能理解组件的功能和映射关系。

于是，为了便于开发人员快速编写代码，Annotation 注解机制在 Servlet3.0 中被引入用作 web.xml 配置的补充，允许开发人员直接在 Java 代码中嵌入配置信息，替代传统 web.xml 中的 XML 配置。注解本质是代码级别的标记，通过“@注解名”的形式声明，由 Web 容器在运行时解析并生效。目前使用的注解主要有@WebServlet、@MultipartConfig、@WebFilter 和@WebListener。@MultipartConfig、@WebFilter 和@WebListener 将在后续章节讲述，下面重点介绍@WebServlet 注解。

@WebServlet 用于直接替代传统 web.xml 中<servlet>和<servlet-mapping>的 XML 配置，可直接写在代码当中。它包含多个可选属性，均可通过键值对形式配置，如表 5-15 所示。

表 5-15　@WebServlet 注解的属性说明

属性名	类　　型	说　　明
name	String	Servlet 的唯一标识（对应 web.xml 的<servlet-name>，默认值为类名）
urlPatterns	String[]	Servlet 映射的 URL 路径，支持多个路径（如{"/judge", "/submit"}）
value	String[]	urlPatterns 的简化写法。如果只需要配置路径（无其他属性要配置），可直接写作：value = "/judge"或"/judge"或 value = {"/judge", "/submit"}或{"/judge", "/submit"}
initParams	WebInitParam[]	初始化参数数组，每个参数通过@WebInitParam(name = "...", value = "...")注解进行定义
loadOnStartup	int	应用启动时加载 Servlet：非负整数（值越小优先级越高）；默认为负数（首次请求时加载）
asyncSupported	boolean	是否支持异步处理（默认为 false）

注意，urlPatterns 和 value 属性只能必选一项，二者是互斥的。

下面利用@WebServlet 注解来完成 JudgeServlet1 的参数配置和读取，如例 5-18 所示。

例 5-18　JudgeServlet1.java

```
package com.myoj.service;

import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.annotation.WebInitParam;
import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletContext;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

//使用注解配置 Servlet
@WebServlet(
        urlPatterns = {"/judge1", "/submit1"},     //显式声明路径属性
        initParams = {@WebInitParam(name = "timeout", value = "5"),@WebInitParam(name =
"maxCodeSize", value = "512")},
        loadOnStartup = 1
    )
//@WebServlet({"/judge1", "/submit1"})

public class JudgeServlet1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        //设置响应内容类型及编码
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        //1. 获取 ServletConfig 对象(当前 Servlet 配置)
        ServletConfig config = getServletConfig();
        //2. 读取<servlet>中的<init-param>参数
        String timeout = config.getInitParameter("timeout");   //参数名称要与注解一致
        String maxCodeSize = config.getInitParameter("maxCodeSize");

        //输出参数信息到页面
        out.println("<html>");
        out.println("<head><title>参数读取示例</title></head>");
        out.println("<body>");
        out.println("<h4>Servlet 注解的初始化参数读取结果</h3>");
        out.println("<p>超时(timeout):" +
                (timeout != null ? timeout + "秒" : "未配置") + "</p>");
        out.println("<p>最大代码长度(maxCodeSize):" +
                    (maxCodeSize != null ? maxCodeSize + "字节" : "未配置") + "</p>");
        out.println("</body></html>");
    }
}
```

上述代码中,JudgeServlet1 的映射 URL 有两个,分别是/judge1 和/submit1,即这两个 URL 都会被 Web 容器解释为执行 JudgeServlet1。此外,还为 JudgeServlet1 配置了两项初始化参数 timeout 和 maxCodeSize,这些参数采用了 getInitParameter(String name)方法来读取,读取结果如图 5-24 所示。

图 5-24　Servlet 注解的初始化参数读取结果

▶ 5.7.3　Web 应用程序的部署及其目录结构

Web 应用程序在开发完毕后都需要部署上线，按照 Servlet 规范正确打包 Web 应用程序是成功发布应用的前提，也是 Web 容器正确识别加载各种组件和资源的保障。

Web 应用程序的目录结构是由 Servlet 规范定义的标准化布局，它规定了各类文件（如 Java 类、配置文件、第三方库文件和资源）的存放位置，这种标准化的目录结构不仅便于开发人员统一管理代码，还能保证应用在不同容器中的兼容性和可移植性。典型 Web 应用程序的目录结构如下：

```
myoj/                                  //应用根目录(部署时通常作为 WAR 包或文件夹)
├── META-INF/                          //元数据目录(存放应用描述、配置清单等)
│   ├── MANIFEST.MF                    //应用清单文件(记录应用版本、主类等信息)
│   ├── context.xml                    //容器特定的上下文配置(如 Tomcat 的数据源配置)
│   └── resources/                     //应用级共享资源(可选,如国际化配置文件)
├── WEB-INF/                           //核心配置目录(客户端无法直接访问,安全目录)
│   ├── classes/                       //编译后的 Java 类文件(Servlet、工具类等)
│   │   └── com/
│   │       └── myoj/
│   │           ├── common/            //公共类
│   │           ├── service/           //业务逻辑类(如 JudgeServlet.class)
│   │           └── util/              //工具类
│   ├── lib/                           //应用依赖的 JAR 包(如数据库驱动、第三方框架)
│   │   ├── mysql-connector-java.jar
│   │   └── jackson-databind.jar
│   ├── web.xml                        //部署描述文件(配置 Servlet、过滤器、全局参数等)
│   └── tags/                          //自定义 JSP 标签库(可选)
├── index.html                         //欢迎页(默认首页)
├── static/                            //静态资源目录
│   ├── css/                           //样式表文件(如 style.css)
│   ├── js/                            //JavaScript 文件(如 judge.js)
│   └── images/                        //图片资源(如 logo.png)
├── pages/                             //动态页面目录(JSP 等)
│   ├── login.jsp
│   └── problem.jsp
└── upload/                            //文件上传目录(如用户提交的代码文件)
```

核心目录的功能说明如下。

(1) 应用根目录(如 myoj/)。

① 整个 Web 应用的顶层目录，名称通常为应用的上下文路径(如部署后通过 http://localhost:8080/myoj 访问)。

② 存放可直接被客户端访问的静态资源(如 HTML、CSS、JavaScript)和欢迎页(如

index.html)。

(2) META-INF/目录：存放元数据信息的标准化目录，不允许客户端通过 URL 直接访问。

(3) WEB-INF/目录。

① 核心作用：存放应用的私有配置和代码，客户端无法通过 URL 直接访问(需通过 Servlet 转发)，保证资源安全。

② 包含以下子目录和文件。

- classes/：存放编译后的 Java 字节码文件(.class)，目录结构与 Java 包结构一致(如 com/myoj/servlet/)，Web 容器会自动加载该目录下的类。
- lib/：存放应用依赖的 JAR 包(如数据库驱动、JSON 处理库)，容器会自动将这些 JAR 包加入类路径，无须手动配置环境变量。
- web.xml：部署描述文件，配置 Servlet 映射、过滤器、全局参数、欢迎页等核心信息(Servlet3.0+可部分通过注解替代，但复杂配置仍需依赖此文件)。

(4) 静态资源目录(如 static/)。

① 存放 HTML、CSS、JavaScript、图片等静态文件，客户端可直接通过 URL 访问(如 http://localhost:8080/myoj/static/css/style.css)。

② 按资源类型细分(css/、js/、images/)便于管理，尤其适合前端资源较多的应用。

(5) 动态页面目录(如 pages/)。

① 存放 JSP 等动态页面文件，这些页面通常包含 Java 代码片段或标签库，需要容器解析后生成 HTML 响应。

② 若需限制直接访问，可将其放在 WEB-INF/pages/目录下，通过 Servlet 转发访问(如 request.getRequestDispatcher("/WEB-INF/pages/login.jsp").forward(...))。

(6) 文件上传目录(如 upload/)。

存放用户上传的文件(如 OJ 中用户提交的代码文件、图片等)，通常需要在代码中通过 ServletContext.getRealPath("/upload")获取其物理路径。

5.8 ServletConfig 接口与 ServletContext 接口

ServletConfig 接口和 ServletContext 接口在前面的例子已经被用作来读取 Servlet 的初始化参数和 Web 应用的全局初始化参数。实际上，从它们的名称可以看出，这二者是为读取 Servlet 的私有配置和整个 Web 应用的全局配置而准备的，下面来看下它们的接口原型及其他功能。

1. ServletConfig 接口

ServletConfig 接口定义在 jakarta.servlet 包中，包含以下主要方法。

```
public interface ServletConfig {
    //1. 获取指定名称的初始化参数值
    String getInitParameter(String name);
    //2. 获取所有初始化参数的名称集合
    Enumeration<String> getInitParameterNames();
    //3. 获取 ServletContext 对象
    ServletContext getServletContext();
    //4. 获取当前 Servlet 的名称(对应<servlet-name>或@WebServlet 的 name 属性)
```

```
    String getServletName();
}
```

除了 getServletContext()方法用于获取 ServletContext 接口对象外，其他几个方法的功能都对应了 web.xml 的 Servlet 配置或 Servlet 注解配置。每个 Servlet 对象实例都有一个相应的 ServletConfig 接口对象实例。

2. ServletContext 接口

除了读取 Web 应用的全局初始化参数之外，还有许多实际应用功能须用到 ServletContext 接口。实际上，ServletContext 接口对象作为 Web 应用的“全局管家”，是沟通 Web 应用与 Web 容器的桥梁。Web 容器在加载某项 Web 应用时，都会给它创建唯一的 ServletContext 实例，该实例被应用内的所有组件共享，能够适用丰富的应用场景。

(1) 全局属性管理(跨组件数据共享)：参考应用场景，统计整个在线评判系统的在线人数。

因为不清楚用户会访问使用哪项功能，所以需要设置一个全局变量来统计整个应用的在线人数，此时可使用 ServletContext 接口全局属性管理方法来实现，如表 5-16 所示。

表 5-16　ServletContext 接口的属性管理方法

方法原型	功能描述
Object getAttribute(String name)	根据属性名 name 获取全局属性值，返回 Object 类型(需强制转换)。示例：Integeronline＝(Integer) context.getAttribute("onlineUsers")
void setAttribute(String name, Object value)	设置全局属性(name/value 键值对)，值可为任意 Java 对象。示例：context.setAttribute("onlineUsers",100)
void removeAttribute(String name)	根据属性名 name 移除全局属性，释放资源
Enumeration<String> getAttributeNames()	返回所有全局属性的名称枚举，用于遍历所有属性

(2) 资源访问管理：参考应用场景，获取文件上传的绝对物理路径、读取待发送的邮件模板文件、遍历代码测试的数据用例文件等。

这些功能都需要和服务器的资源文件进行交互，可使用 ServletContext 接口的资源访问管理方法来实现，具体如表 5-17 所示。

表 5-17　ServletContext 接口的资源管理方法

方法原型	功能描述
String getRealPath(String path)	将 Web 应用的相对路径(以/开头)转换为服务器上的物理路径(绝对路径)。 示例：context.getRealPath("/upload") 返回 D:/tomcat/webapps/oj/upload
InputStream getResourceAsStream(String path)	以输入流形式读取指定资源，路径为 Web 应用的相对路径。 示例：context.getResourceAsStream("/config/db.properties")
Set<String> getResourcePaths(String path)	返回指定目录下所有资源文件的路径集合(包含子目录)，路径以/开头
URL getResource(String path)	返回指定资源的 URL 对象，用于获取资源的统一资源定位符

(3) 整个 Web 应用的信息查询：参考应用场景，显示应用名称、提取应用根路径、显示应用的 Servlet 容器版本信息等。相关方法如表 5-18 所示。

表 5-18　ServletContext 接口的应用信息查询方法

方 法 原 型	功 能 描 述
String getServletContextName()	获取 Web 应用的名称，对应 web.xml 中< display-name >的值（若未配置则返回 null）
String getContextPath()	返回 Web 应用的上下文路径（即部署后的访问路径前缀）
String getMimeType(String file)	根据文件名或扩展名获取对应的 MIME 类型。 示例：context.getMimeType("test.jpg") 返回 image/jpeg
int getMajorVersion()	返回当前 Servlet 规范的主版本号（如 Servlet5.0 返回 5）
int getMinorVersion()	返回当前 Servlet 规范的次版本号（如 Servlet5.0 返回 0）

（4）应用级日志记录：参考应用场景，系统运维信息管理。

日志是所有系统应用不可或缺的功能，它记录了系统运行过程中的关键事件、状态变化、错误信息及用户操作等内容，既是系统问题排查与故障定位的重要依据，也是系统运行状态监控的有效手段。通过分析日志的趋势变化可提前预警潜在风险，为保障系统稳定安全高效运行提供了数据基础。相关方法如下。

void log(String message)：用于记录普通日志信息，日志内容会输出到日志文件，例如 Tomcat 的 catalina.out。

void log(String message, Throwable throwable)：用于记录包含异常堆栈信息的错误日志，便于调试。

除了上述常用功能方法，ServletContext 接口还有事件管理、动态组件注册等功能，有兴趣的读者可以进一步深入学习。

5.9　特殊的 HTTP 请求与响应：文件上传与下载

文件上传与下载是每个 Web 应用程序必备的功能，例如个人头像的上传、通知文件的下载等。本节将结合相关的知识点完成这两项功能。

▶ 5.9.1　文件上传

文件上传本质上是客户端（浏览器）将本地文件通过 HTTP 发送到服务器端，服务器端接收并存储文件的过程文件的上传，具体步骤如下。

（1）前端页面：设计表单并设置两项属性值 enctype＝"multipart/form-data"和 method＝"post"。

（2）前端页面：在该表单中添加 file 表单域，让用户选择待上传的文件（可支持多选）。

（3）后台 Servlet：添加@MultipartConfig 注解支持文件上传。

（4）后台 Servlet：通过 request.getPart()方法获取文件对象，并验证类型和大小。

（5）后台 Servlet：生成保存目录并保存文件。

下面围绕个人头像上传的功能展开详细设计。

1. **前端页面 user-center.html 的设计**

例 5-19　user-center.html

```
<!DOCTYPE html>
<html>
```

```
<head>
    <meta charset="UTF-8">
    <title>用户中心</title>
</head>
<body>
    <h1>用户中心</h1>
    <h3>当前头像</h3>
    <img src="images/avatar.png" width="100" alt="用户头像">
    <h3>更换头像</h3>
    <form action="uploadAvatar" method="post" enctype="multipart/form-data">
        <input type="file" name="avatar" accept="image/jpeg" required>
        <button type="submit">立即上传头像</button>
        <p>说明:仅支持 JPG 格式,最大为 2MB</p>
    </form>
    <!-- 显示上传结果提示 -->
    <p id="message" style="color:red"></p>
    <script>
        const params = new URLSearchParams(window.location.search);
        const message = params.get('message');
        if (message) {
            document.getElementById('message').textContent = message;
        }
    </script>
</body>
</html>
```

上述代码中,表单的 enctype 属性用于指定表单数据提交到服务器时的编码类型,其默认值为 application/x-www-form-urlencoded,此时表单提交的所有数据会被编码为键值对字符串再传输到服务器端(如 username=test&code=main),特殊字符会被转换为相应的 URL 编码(如空格转换为%20);当 enctype="multipart/form-data"时表示该表单中提交的数据中包含文件,此时表单提交的数据会分成多个 Part 对象,每个 Part 对象都分别对应表单的一项数据。

文件表单域 avatar 的 accept 属性表示限制选择的文件类型,这里只允许选择 JPG 图片。如果要支持一次性选择多个文件,则需要加上 multiple 属性。

语句"const params=new URLSearchParams(window.location.search);"的作用是构造 URLSearchParams 对象,该对象用于解析查询字符串中的参数;参数 window.location.search 用于获取 URL 中的查询字符串(即?及其后面的查询参数部分)。语句"const message = params.get('message');"的作用从查询字符串中读取参数名称为 message 的参数值。两条语句共同用于显示上传头像操作后的相关消息。

2. 后台 AvatarUploadServlet.java 的设计

例 5-20　AvatarUploadServlet.java

```
package com.myoj.service;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.MultipartConfig;
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.File;
import java.io.InputStream;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.UUID;

//配置文件上传限制:单个文件最大为 2MB
@MultipartConfig(maxFileSize = 2 * 1024 * 1024)
@WebServlet("/uploadAvatar")
public class AvatarUploadServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        //1. 获取 Web 应用根目录的真实路径
        String webRootPath = getServletContext().getRealPath("/");
        //2. 定义文件存储子目录(根目录下的 upload/avatars)
        String avatarDir = webRootPath + "upload/avatars/";
        new File(avatarDir).mkdirs();       //确保存储目录存在
        try {
            //获取上传的文件
            Part avatarPart = request.getPart("avatar");
            if (avatarPart == null || avatarPart.getSize() == 0) {
                redirectWithMessage(response, "user-center.html", "请选择图片文件");
                return;
            }
            //验证文件类型
            String fileName = avatarPart.getSubmittedFileName();
            if (!fileName.toLowerCase().endsWith(".jpg")) {
                redirectWithMessage(response, "user-center.html", "仅支持 JPG 格式");
                return;
            }
            //生成唯一文件名并保存
            String suffix = fileName.substring(fileName.lastIndexOf("."));
            String filesavePath = avatarDir + UUID.randomUUID().toString() + suffix;
            //保存文件到服务器
            InputStream in = avatarPart.getInputStream();
            Files.copy(in, Paths.get(filesavePath));
            //模拟更新用户头像信息(实际项目中需更新数据库)
            System.out.println("头像保存成功:" + filesavePath);
            redirectWithMessage(response, "user-center.html", "头像上传成功");
        } catch (IllegalStateException e) {
            redirectWithMessage(response, "user-center.html", "文件大小不能超过 2MB");
        } catch (Exception e) {
            redirectWithMessage(response, "user-center.html", "上传失败:" + e.getMessage());
        }
    }
    //重定向并携带提示消息
    private void redirectWithMessage(HttpServletResponse response, String url, String message)
            throws IOException {
        response.sendRedirect(url + "?message=" + java.net.URLEncoder.encode(message,
"UTF-8"));
    }
}
```

上述代码中，注解@MultipartConfig用于通知Web容器此Servlet支持文件上传功能，可接收处理multipart/form-data类型的请求。Web容器会将multipart/form-data格式的请求体分离为普通数据和文件数据。注解@MultipartConfig包含多个可选属性，用于配置文件上传的细节（如临时存储路径、文件大小限制等），所有属性均为可选，具体如表5-19所示。

表5-19　@MultipartConfig的属性

属　性	说　明
int fileSizeThreshold	当文件大小超过该阈值（单位：字节）时，容器会将文件临时写入磁盘；否则保存在内存中。默认值为0（所有文件直接写入磁盘）
String location	指定文件上传时的临时存储目录（绝对路径），若未指定则使用容器默认的临时目录。示例：location = "D:/temp/upload"
long maxFileSize	单个上传文件的最大允许大小（单位：字节），超过此值会抛出IllegalStateException。默认值为-1L（无限制）
long maxRequestSize	整个请求（包含所有文件和普通字段）的最大允许大小（单位：字节），超过此值会抛出异常。默认值为-1L（无限制）

语句“String webRootPath = getServletContext().getRealPath("/");”中先通过getServletContext()方法取得了ServletContext接口对象引用，再借助该对象的getRealPath("/")方法读取了本Web应用根路径“/”所对应的绝对物理路径，该绝对物理路径用于辅助生成头像的保存目录。

语句“Part avatarPart = request.getPart("avatar");”中使用了HttpServletRequest的getPart(String name)方法来读取上传文件对应的Part对象；如果表单支持并提交了多个文件，则需先使用getParts()方法获取Part对象集合，再使用循环逐一处理。Part接口定义了对某项表单域数据的操作方法（表单请求的编码类型须为multipart/form-data），具体如表5-20所示。

表5-20　Part接口的方法

方　法	功能描述
String getName()	返回表单域的name属性值（如<input name="username">，返回"username"）
String getSubmittedFileName()	返回文件域的客户端原始本地文件名；文本域返回null
long getSize()	返回域内容的大小（单位为字节），包括文件域和文本域
InputStream getInputStream()	返回表单域内容的输入流，可读取文本或文件的原始数据
void write(String fileName)	将文件域的内容写入服务器指定路径（fileName为保存路径+文件名）
void delete()	删除上传过程中产生的临时文件（若文件曾被写入临时目录）
String getContentType()	返回文件域的MIME类型（如image/jpeg）；普通文本域返回null

语句“Files.copy(in, Paths.get(filesavePath));”中借助了Files类的静态方法copy()完成文件从内存到磁盘的复制。Files类是java.nio包内（非阻塞式New IO）的工具类，提供了文件的全部操作。

方法java.net.URLEncoder.encode(message,"UTF-8")的功能是对message消息内的中文进行转码。URLEncoder是java.net包中处理URL编码的工具类，用于将字符串转换为符合URL规范的编码格式（即application/x-www-form-urlencoded格式），可解决URL中特殊字符（如空格、中文、标点符号）的传输问题。它只有一个方法encode(String s, String enc)，其中，s为待编码的字符串，enc为编码格式。

▶ 5.9.2 文件下载

在 Web 应用中实现文件下载的方式有两种：直接通过 URL 访问静态文件和通过 Servlet 动态处理下载。本节将围绕“赛事通知下载”介绍每种方法的步骤。

1. **直接访问静态文件(最简便但不安全)**

首先将赛事通知文件(如 PDF、Word)放在 Web 应用的静态资源目录(如/downloads/)中,然后通过设置超链接让用户直接访问下载文件。

```
<a href="/downloads/match-notice.pdf" download="赛事通知.pdf">下载赛事通知(PDF)</a>
```

其中,download 为<a>标签的可选属性,用于指定浏览器下载时的默认文件名。

该方法适合无权限控制的公开文件下载,其缺点是容易受到攻击。

2. **通过 Servlet 动态处理下载(最常用)**

本方法是本节的重点,具体步骤如下。

(1) 准备文件：将文件存放在服务器的安全目录(如/WEB-INF/downloads/赛事通知.pdf,避免通过 URL 直接访问)下。

(2) 编写 Servlet 控制下载：包括验证用户权限、读取目标文件的输入流、设置响应头、设置文件默认处理方式和默认文件名、将文件流写入响应输出流等步骤。

(3) 访问 Servlet 对应的 URL 下载文件。

例 5-21 展示了如何利用 Servlet 控制文件下载。

例 5-21 NoticeDownloadServlet.java

```
package com.myoj.service;

import jakarta.servlet.annotation.*;
import jakarta.servlet.http.*;
import java.nio.file.*;
import java.io.*;

@WebServlet("/noticeDownloadServlet")
public class NoticeDownloadServlet extends HttpServlet {
    @Override
     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException {
        //1. 权限校验(示例:仅登录用户可下载)
        boolean loggedIn=true; //默认值为 true,具体校验可根据实际情况确定
        if (!loggedIn) {     response.sendRedirect("/login.html");  return;}
        //2. 定义文件路径和名称(赛事通知文件)
        String filePath=getServletContext().getRealPath("/WEB-INF/downloads/");
        String fileName = "赛事通知.pdf";      //下载时显示的文件名
        //3. 读取文件
        File file = new File(filePath+fileName);
        if (!file.exists()) {
            response.setContentType("text/html;charset=UTF-8");
            response.getWriter().println("文件不存在!");
            return;
        }
        //4. 设置响应头(通知浏览器该文件的默认打开方式)
        String type=Files.probeContentType(Paths.get(filePath+fileName));
```

```
            //自动探测文件类型
            response.setContentType(type);      //指定文件的 MIME 类型
            String encodedFileName = java.net.URLEncoder.encode(fileName, "UTF-8");
            //防止文件名乱码
            //告诉浏览器以附件形式下载(而非直接打开)
               response.setHeader("Content-Disposition", "attachment; filename=" +
encodedFileName);
            response.setContentLengthLong(file.length());
            //设置文件大小,支持并发传输、断点续传和校验

            //5. 写入文件流到响应
            InputStream in = new FileInputStream(file);
            OutputStream out = response.getOutputStream();
            byte[] buffer = new byte[1024];
            int len;
            while ((len = in.read(buffer)) != -1) {
                out.write(buffer, 0, len);
            }
            in.close();
        }
    }
```

上述代码中,除了第 5 部分的文件流写入,最关键的就是第 4 部分设置响应头。

probeContentType()方法也是 java.nio 中 Files 类的静态方法,用于探测文件的 MIME 类型,该方法的原型如下:

```
public static String probeContentType(Path path) throws IOException
```

其中,参数 path 表示需要探测 MIME 类型的文件路径对象,其类型为 java.nio.file.Path,可通过内置工具类 Paths 的 get(String filePath)方法将字符串路径转换为 Path 对象。

语句"response.setHeader("Content-Disposition", "attachment; filename="+encodedFileName);"用于通知浏览器发送给它的文件直接以 attachment(附件)形式下载。

语句"response.setContentLengthLong(file.length());"用于通知浏览器待发送文件的长度,这不是必需的操作,但通知文件长度可以让浏览器并发下载,并支持断点续传和文件校验。