第3章 Servlet


视频讲解

3.1 Servlet 概述

Servlet 是用 Java 语言编写的服务器端程序，在服务器端调用和执行。Servlet 可以处理客户端发来的 HTTP 请求，并返回一个响应。狭义的 Servlet 指用 Java 语言实现的一个 Servlet 接口，广义的 Servlet 指任何实现了这个接口的类。虽然是用 Java 语言编写的程序，Servlet 没有 public static void main(String[] args)方法，不能独立运行。它的运行需要服务器提供运行环境。能够为 Servlet 提供运行环境的软件称为 Web 容器或 Servlet 容器(如 Tomcat)。Web 容器在接收客户端请求后生成响应。一台物理服务器上可以布置多个 Web 容器。而 Servlet 需要由 Web 容器实例化并调用。Servlet 应用程序的体系结构如图 3-1 所示。

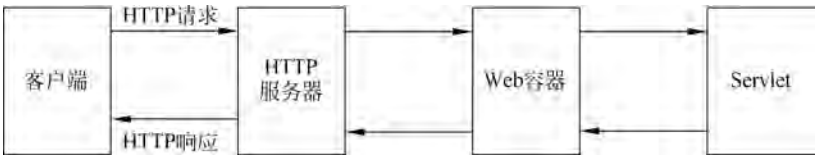

图 3-1 Servlet 应用程序的体系结构

如图 3-1 所示，客户端向 Servlet 发出请求，该请求首先被 HTTP 服务器(如 Nginx、Apache 等)接收，HTTP 服务器只负责静态页面(HTML 页面)的解析，对于 Servlet 请求则转交给 Web 容器。Web 容器会根据请求路径的映射关系调用相应的 Servlet 进行处理。Servlet 处理请求后，将响应返回给客户端。

与其他技术相比，Servlet 技术具有以下特点：

(1) Servlet 使用了与 CGI(Common Gateway Interface，通用网关接口)不同的处理模型，因此运行速度更快。

(2) Servlet 使用了很多 Web 容器都支持的标准 API(Application Programming Interface，应用程序编程接口)。

(3) Servlet 具有 Java 语言的全部优点，如开发简单、平台独立等。

(4) Servlet 可以使用 Java API。

3.2 Servlet 基础

针对 Servlet 开发，有一系列可用的接口和类。其中最重要的接口是 jakarta.servlet.Servlet。Servlet 接口定义了 5 个抽象方法，如表 3-1 所示。

表 3-1　Servlet 接口的抽象方法

方 法 声 明	说　　明
void init(ServletConfig config)	Web 容器在创建 Servlet 对象后，会调用此方法。该方法接收一个 ServletConfig 类型的参数，容器通过这个参数向 Servlet 传递初始化配置信息
ServletConfig getServletConfig()	用于获取 Servlet 对象的配置信息，返回 ServletConfig 对象
String getServletInfo()	返回一个字符串，其中包含 Servlet 信息，如作者、版本等
void service(ServletRequest request,ServletResponse response)	负责响应用户请求，当容器收到客户端访问 Servlet 的请求时就会调用此方法。容器会构造一个表示客户端请求信息的 ServletRequest 对象和一个用于响应请求的 ServletResponse 对象作为参数传递给 service()方法。该方法可以通过 ServletRequest 对象得到客户端的相关信息和请求信息，在对请求进行处理后，调用 ServletResponse 对象的方法设置响应信息
void destroy()	负责释放 Servlet 对象占用的资源。当服务器关闭或 Servlet 对象被移除时，容器会调用此方法销毁 Servlet 对象

Servlet 接口是 Jakarta Servlet API 的核心。所有的 Servlet 都可以实现这个接口。或者更直接地，用户自定义的 Servlet 可以继承 Servlet 接口的实现类。在 Jakarta Servlet API 中，有两个实现了 Servlet 接口的类：GenericServlet 和 HttpServlet。Servlet 接口、GenericServlet 类和 HttpServlet 类的关系如图 3-2 所示。对大多数开发者而言，可以通过直接继承 HttpServlet 类，创建一个 Servlet。

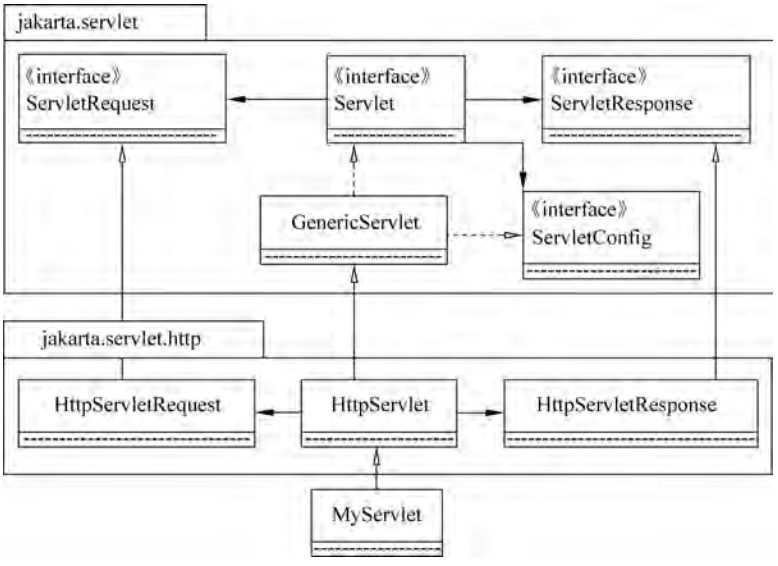

图 3-2　Servlet 接口、GenericServlet 类和 HttpServlet 类的关系

图 3-2 给出了 jakarta.servlet 包中与 Servlet 接口有关的 4 个接口。ServletRequest 和 ServletResponse 接口分别代表请求和响应对象；ServletConfig 代表 Servlet 初始化时用来传递 Servlet 配置信息的对象。此外，在 jakarta.servlet.http 包中还提供了 ServletRequest 接口的子接口 HttpServletRequest 和 ServletResponse 接口的子接口 HttpServletResponse，它们分别

代表 HTTP 请求对象和 HTTP 响应对象。关于 HttpServletRequest、HttpServletResponse 和 ServletConfig 接口的更多内容见 3.4 节。

表 3-1 列举的 5 个抽象方法中,init()、service()和 destroy()方法是与 Servlet 生命周期相关的 3 个方法。Servlet 的生命周期可以被定义为 Servlet 从创建到销毁的整个过程。Servlet 的生命周期可以分为初始化阶段、运行阶段和销毁阶段,如图 3-3 所示。

图 3-3 Servlet 的生命周期

1. 初始化阶段

当客户端向 Servlet 容器发出 HTTP 请求访问 Servlet 时,Servlet 容器会解析请求,检查内存中是否已经存在该 Servlet 对象。如果存在则直接使用该 Servlet 对象;如果没有则创建 Servlet 对象。然后调用 init()方法实现 Servlet 的初始化。初始化一般是完成一些一次性的工作,如读取持久化配置数据,执行一些耗时的操作[如基于 JDBC(Java Database Connetivity,Java 数据库连接)API 的数据库连接]。在 Servlet 的生命周期内,init()方法只被调用一次。

2. 运行阶段

这是 Servlet 生命周期中最重要的阶段。在这个阶段,Servlet 容器会创建代表客户端请求的 ServletRequest 对象和代表服务器响应的 ServletResponse 对象,然后将它们作为参数传递给 Servlet 的 service()方法。service()方法从 ServletRequest 对象中获得客户端请求信息并处理该请求,通过 ServletResponse 对象生成响应。在 Servlet 的整个生命周期内,对于每一个访问 Servlet 的请求,Servlet 容器都会创建新的 ServletRequest 和 ServletResponse 对象,并调用 service()方法处理该请求。即在 Servlet 的生命周期中,service()方法会被多次调用。

3. 销毁阶段

当 Servlet 容器关闭或 Servlet 对象被移除时,Servlet 容器会调用 destroy()方法。destroy()方法一般用于执行一些清理活动,如关闭数据库连接,停止后台线程,把 Cookie 数据写入磁盘等。在 Servlet 的生命周期中,destroy()方法只被调用一次。

【例 3-1】 创建一个 Servlet 并运行。

创建一个名为 myservlet 的动态 Web 项目,在 src/main/java 文件夹下创建一个名为

com. example. servlet. demo 的包。右击该包，在弹出的快捷菜单中选择 Servlet 的创建向导，如图 3-4 所示。

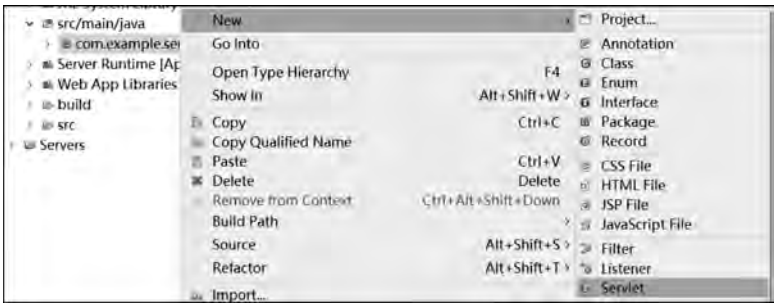

图 3-4　选择 Servlet 的创建向导

指定新建的 Servlet 的名字和所在包，如图 3-5 所示，本例中的 Servlet 继承了 jakarta. servlet. http. HttpServlet 类。单击 Next 按钮，指定要自动生成的 Servlet 方法，如图 3-6 所示，本案例中选择生成 init()、service()和 destroy()方法。单击 Finish 按钮即完成 Servlet 的创建。

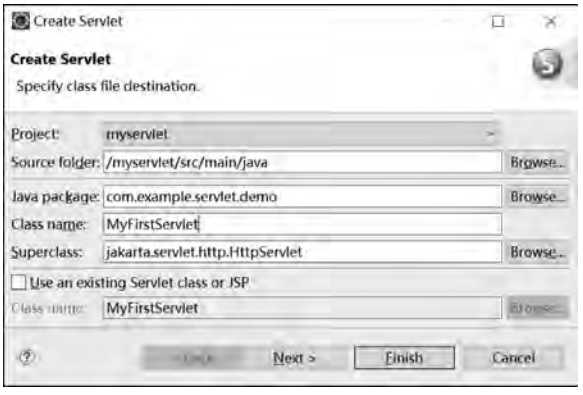

图 3-5　指定 Servlet 的包名和类名

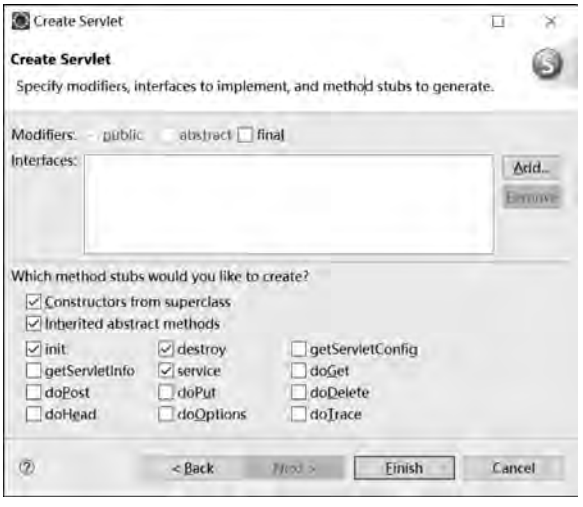

图 3-6　指定要自动生成的 Servlet 方法

修改生成的 Servlet 代码。在 init()方法和 service()方法中分别加入控制台输出。修改后的代码如文件 3-1 所示。

【文件 3-1】 MyFirstServlet.java

```
1  package com.example.servlet.demo;
2
3  import jakarta.servlet.ServletConfig;
4  import jakarta.servlet.ServletException;
5  import jakarta.servlet.annotation.WebServlet;
6  import jakarta.servlet.http.HttpServlet;
7  import jakarta.servlet.http.HttpServletRequest;
8  import jakarta.servlet.http.HttpServletResponse;
9  import java.io.IOException;
10
11 @WebServlet("/MyFirstServlet")
12 public class MyFirstServlet extends HttpServlet {
13     private static final long serialVersionUID = 1L;
14
15     public MyFirstServlet() {
16         super();
17     }
18     public void init(ServletConfig config) throws ServletException {
19         System.out.println("initialize");
20   }
21   public void destroy() {
22   }
23
24   protected void service(HttpServletRequest request,
25       HttpServletResponse response)
26       throws ServletException, IOException {
27         System.out.println("handle requests");
28   }
29 }
```

运行 Servlet 的时候,需要先将项目部署到 Tomcat 服务器上(右击 Servlet 类文件,依次选择 Run as→Run on server 命令),启动 Tomcat,然后在浏览器的地址栏输入“http://localhost:8080/myservlet/MyFirstServlet”。此时,控制台的输出如图 3-7 所示。

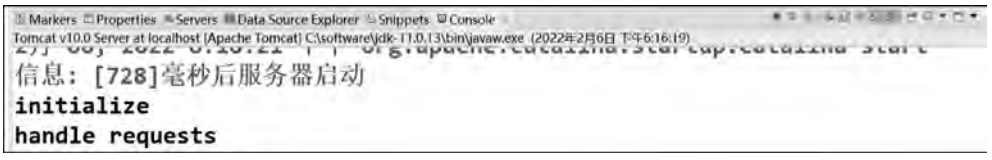

图 3-7 控制台的输出

可见,当 MyFirstServlet 接收到客户端请求时,首先进行初始化,由 Servlet 容器(Tomcat)调用 init()方法,控制台输出字符串“initialize”。随后,Tomcat 调用 Servlet 的 service()方法处理用户请求,控制台输出字符串“handle requests”。此时,如果刷新浏览器窗口,会看到控制台又一次输出字符串“handle requests”。由此可见,在 Servlet 的生命周

期中，init()方法只被调用一次，而用来处理请求的 service()方法会被多次调用。

提示：给 Servlet 发送请求的时候，请求的地址必须与@WebServlet 注解中的参数完全一致。如本例中，@WebServlet 标签中指定的参数是“/MyFirstServlet”，因此，该 Servlet 请求地址即为“http://localhost:8080/myservlet/MyFirstServlet”

事实上，HttpServlet 类不仅定义了 service()方法，也定义了 doGet()和 doPost()方法。这样，在 Servlet 的生命周期中，处理客户端请求就有两种方案。一种是用 service()方法处理请求，另一种是用 doGet()和 doPost()方法代替 service()方法处理请求。这两个方法与客户端发送请求的方式密切相关。doGet()方法处理以 GET 方式发送的请求，doPost()方法处理以 POST 方式发送的请求。由于大多数客户端发送请求的方式都是 GET 和 POST，因此，学习如何使用 doGet()方法和 doPost()方法处理请求就变得相当重要。下面通过一个案例来介绍 doGet()和 doPost()方法的使用。

【例 3-2】 分别以 GET 和 POST 方式向 Servlet 发送请求，并查看控制台输出，步骤如下。

1. 创建 Servlet 类

在 com.example.servlet.demo 包中创建一个名为 RequestMethodServlet 的类。

2. 重写 doGet()和 doPost()方法

重写 doGet()和 doPost()方法如文件 3-2 所示。

【文件 3-2】 RequestMethodServlet.java

```
1  package com.example.servlet.demo;
2
3  import jakarta.servlet.ServletException;
4  import jakarta.servlet.annotation.WebServlet;
5  import jakarta.servlet.http.HttpServlet;
6  import jakarta.servlet.http.HttpServletRequest;
7  import jakarta.servlet.http.HttpServletResponse;
8  import java.io.IOException;
9  import java.io.PrintWriter;
10
11 @WebServlet("/RequestMethodServlet")
12 public class RequestMethodServlet extends HttpServlet {
13
14 protected void doGet(HttpServletRequest request,
15   HttpServletResponse response) throws ServletException, IOException {
16       PrintWriter out = response.getWriter();
17       out.print("this is doGet() method.");
18   }
19
20 protected void doPost(HttpServletRequest request,
21   HttpServletResponse response) throws ServletException, IOException {
22       PrintWriter out = response.getWriter();
23       out.print("this is doPost() method.");
24   }
25 }
```

3. 提交 GET 请求

启动 Tomcat 服务器,在浏览器的地址栏输入“http://localhost:8080/myservlet/RequestMethodServlet”,以 GET 方式向 Servlet 发送请求,浏览器显示结果如图 3-8 所示。由此可见,当以 GET 方式向 Servlet 发送请求时,Servlet 会调用 doGet()方法处理请求。

图 3-8 提交 GET 请求后浏览器的显示结果

4. 提交 POST 请求

采用 POST 方式提交请求时,需要在 src/main/webapp 目录下创建一个名为 form.html 的文件,其中表单的 acion 属性要与@WebServlet 注解指定的参数一致。这里省略了代表项目根目录的正斜线(/),并指定请求提交方式为 POST,代码如文件 3-3 所示。

【文件 3-3】 form.html

```
1  <html>
2  <body>
3    <form action="RequestMethodServlet" method="post">
4        <label>用户名</label>
5        <input type="text" name="name" /><br>
6        <input type="submit" value="提交" />
7    </form>
8  </body>
9  </html>
```

启动 Tomcat 服务器后,在浏览器的地址栏输入“http://localhost:8080/myservlet/form.html”。填写相关内容后,单击“提交”按钮,以 POST 方式向 Servlet 发送请求,浏览器显示的结果如图 3-9 所示。由此可见,采用 POST 方式提交请求时,Servlet 会调用 doPost()方法处理请求。

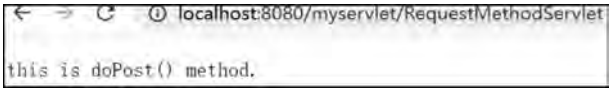

图 3-9 提交 POST 请求后浏览器的显示结果

3.3 Servlet 配置

不同于传统的 Java 应用程序,Servlet 在创建后需要在服务器端做好配置。当然,有些集成化开发环境在创建 Servlet 的同时即完成了配置,例如文件 3-1 的创建过程。配置 Servlet 有两种方式:部署描述符和注解。

1. 部署描述符

部署描述符可以在应用程序开发阶段、集成阶段和部署阶段传递 Web 应用程序的元素和配置信息。在 Servlet 5.0 规范中,部署描述符是根据 XML Schema 文档定义的。对于

例 3-1 中的 Servlet，可采用如下的部署描述符进行配置。

在项目的 WEB-INF 目录中创建(或修改)web.xml 文件，代码如文件 3-4 所示。

【文件 3-4】 web.xml

```
1  <servlet>
2    <servlet-name>MyFirstServlet</servlet-name>
3    <servlet-class>
4       com.example.servlet.demo.MyFirstServlet
5   </servlet-class>
6  <load-on-start-up>1</load-on-start-up>
7    <init-param>
8       <param-name>catalog</param-name>
9       <param-value>Spring</param-value>
10   </init-param>
11 </servlet>
12 <servlet-mapping>
13   <servlet-name>MyFirstServlet</servlet-name>
14   <url-pattern>/MyFirstServlet</url-pattern>
15 </servlet-mapping>
```

如文件 3-4 所示，元素<servlet>用于注册一个 Servlet(第 1～11 行)。它的两个子元素<servlet-name>和<servlet-class>分别用来指定 Servlet 的名字(第 2 行)和全限定名(第 3～5 行)。元素<servlet-mapping>用于映射外界对 Servlet 的访问路径(第 12～15 行)，它的子元素<servlet-name>的值(第 13 行)必须与<servlet>元素中<servlet-name>的值完全一致。子元素<url-pattern>则是用于指定访问该 Servlet 的虚拟路径(第 14 行)，该路径以正斜线(/)开始，代表当前 Web 应用程序的根目录。

元素<servlet>的子元素<load-on-start-up>是一个可选项(第 6 行)，它用于指定 Servlet 被加载的时机和顺序。在<load-on-start-up>元素中，必须设置一个整数。如果这个值是一个负数或者没有设定这个元素，Servlet 容器将在客户端首次请求这个 Servlet 的时候加载它；如果这个值是正整数或 0，Servlet 容器将在 Web 应用启动时加载并初始化 Servlet，<load-on-start-up>设置的值越小，对应的 Servlet 被加载的优先级越高。

元素<servlet>的子元素<init-param>是一个可选项(第 7～10 行)，用来配置 Servlet 的初始化参数。参数的名字和值分别由<param-name>和<param-value>子元素指定。

2. 注解

从 Servlet 3.0 规范开始，可以使用注解(Annotation)来告知 Servlet 容器哪些 Servlet 会提供服务。在创建 Servlet 后，可以用@WebServlet 注解来配置 Servlet。在文件 3-1 中，使用@WebServlet("/MyFirstServlet")来配置一个 Servlet(第 11 行)。因为设置了@WebServlet 注解，容器会自动读取注解中的内容，进而完成配置。该注解告知容器，如果请求的 URL 中包含 /MyFirstServlet ，则由当前的 MyFirstServlet 处理此请求。因此，访问这个 Servlet 的时候，只要在地址栏输入“http://localhost:8080/myservlet/MyFirstServlet”即可。@WebServlet 注解的属性如表 3-2 所示。

提示：在配置一个 Servlet 的时候，部署描述符和注解只能选择一种，两种方式不可混用。本书的后续案例均以注解方式进行配置。

表 3-2 @WebServlet 的属性

属 性 名	类 型	说 明
asyncSupported	boolean	声明 Servlet 是否支持异步操作模式
description	String	对 Servlet 的描述
displayName	String	Servlet 的显示名,通常配合工具使用
initParams	WebInitParam[]	指定一组 Servlet 的初始化参数,等价于< init-param >
largeIcon	String	指定 Servlet 的大图标
loadOnStartup	int	指定 Servlet 的加载顺序,等价于< load-on-start-up >
name	String	指定 Servlet 的名字,等价于< servlet-name >。如果没有显式指定,则该属性的取值即为类的全限定名
smallIcon	String	指定 Servlet 的小图标
urlPatterns	String[]	指定一组 Servlet 的 URL 匹配模式,等价于< url-pattern >
value	String[]	等价于 urlPatterns 属性。两个属性不能同时使用


视频讲解

3.4 Servlet 常用接口

3.4.1 HttpServletRequest 接口

在 Jakarta Servlet API 中定义了一个 HttpServletRequest 接口。它继承自 ServletRequest 接口,专门用来封装 HTTP 请求。由于 HTTP 请求消息分为请求行、请求头和请求消息体(实体主体)3 部分,因此,在 HttpServletRequest 接口中定义了获取请求行、请求头和请求消息体的相关方法。

1. 获取请求行的相关方法

请求行主要包括请求方法字段、URL 字段、协议名称和版本号字段等。相关方法如表 3-3 所示。

表 3-3 获取请求行的相关方法

方 法 声 明	说 明
String getMethod()	获取 HTTP 请求消息中的请求方法(如 GET、POST 等)
String getRequestURI()	获取请求行中资源名称的部分,即从协议名称到 HTTP 请求第一行中的查询字符串之前的部分(不含服务器名称、端口号)
StringBuffer getRequestURL()	重建客户端用于发出请求的 URL。返回的 URL 包含协议名、服务器名称、端口号和服务器路径,但不包括查询字符串参数
String getQueryString()	返回请求行中的参数部分,即请求 URL 中问号(?)以后的内容
String getProtocol()	返回请求行中的协议名和版本,如 HTTP/1.1
String getContextPath()	返回请求 URI 中指示请求上下文的部分。在请求 URI 中,上下文路径总是排在第一位。路径以“/”字符开头,但不以“/”字符结尾。对于默认(根)上下文中的 Servlet,此方法返回空字符串“”
String getServletPath()	获取 Servlet 名称或 Servlet 的映射路径
String getRemoteAddr()	获取客户端的 IP 地址
String getRemoteHost()	获取客户端的完整主机名,如 host.example.com。如果无法解析客户端的完整主机名,将返回客户端的 IP 地址

续表

方 法 声 明	说　　明
int getRemotePort()	获取客户端网络连接的端口号
String getLocalAddr()	获取 Web 服务器上用于接收请求的端口的 IP 地址
int getLocalPort()	获取 Web 服务器上用于接收请求的端口的端口号
String getLocalName()	获取 Web 服务器上接收请求的端口的主机名
String getServerName()	返回请求发送到的服务器的主机名
int getServerPort()	返回请求发送到的服务器的端口号
String getScheme()	获取请求的协议名，如 http，https 或 ftp

下面，通过一个案例来演示这些方法的使用。在 src/main/java 文件夹下新建一个名为 com. example. servlet. request 的包。在包中创建一个名为 RequestLineServlet 的类，在该类中编写用于获取请求行中相关信息的方法。

【例 3-3】 获取请求行信息。代码如文件 3-5 所示。

【文件 3-5】 RequestLineServlet. java

```
1  package com.example.servlet.request;
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5
6  import jakarta.servlet.ServletException;
7  import jakarta.servlet.annotation.WebServlet;
8  import jakarta.servlet.http.HttpServlet;
9  import jakarta.servlet.http.HttpServletRequest;
10 import jakarta.servlet.http.HttpServletResponse;
11
12 @WebServlet("/RequestLineServlet")
13 public class RequestLineServlet extends HttpServlet {
14
15   protected void doGet(HttpServletRequest request,
16   HttpServletResponse response) throws ServletException, IOException {
17     response.setContentType("text/html;charset=UTF-8");
18     PrintWriter out = response.getWriter();
19     // 获取请求行的相关信息
20     out.print("getMethod:" + request.getMethod() + "<br>");
21     out.print("getRequestURI:" + request.getRequestURI() + "<br>");
22     out.print("getQueryString:" + request.getQueryString() + "<br>");
23     out.print("getProtocol:" + request.getProtocol() + "<br>");
24     out.print("getContextPath:" + request.getContextPath() + "<br>");
25     out.print("getPathInfo:" + request.getPathInfo() + "<br>");
26     out.print("getServletPath:" + request.getServletPath() + "<br>");
27     out.print("getRemoteAddr:" + request.getRemoteAddr() + "<br>");
28     out.print("getRemoteHost:" + request.getRemoteHost() + "<br>");
29     out.print("getRemotePort:" + request.getRemotePort() + "<br>");
30     out.print("getLocalAddr:" + request.getLocalAddr() + "<br>");
31     out.print("getLocalName:" + request.getLocalName() + "<br>");
32     out.print("getLocalPort:" + request.getLocalPort() + "<br>");
33     out.print("getServerName:" + request.getServerName() + "<br>");
34     out.print("getServerPort:" + request.getServerPort() + "<br>");
```

```
35        out.print("getScheme:" + request.getScheme() + "<br>");
36        out.print("getRequestURL:" + request.getRequestURL() + "<br>");
37    }
38    protected void doPost(HttpServletRequest request,
39        HttpServletResponse response) throws ServletException, IOException {
40          doGet(request, response);
41    }
42  }
```

启动 Tomcat 服务器,在浏览器的地址栏输入“http://localhost:8080/myservlet/RequestLineServlet”,向 RequestLineServlet 发送请求,运行结果如图 3-10 所示。

图 3-10　RequestLineServlet 的运行结果

2. 获取请求头的相关方法

请求头可以用来向服务器传递附加的请求信息。如客户端可以接收的数据类型、压缩方式、语言等。为此,HttpServletRequest 接口中定义了一系列用于获取 HTTP 请求头字段的方法,如表 3-4 所示。

表 3-4　获取请求头的相关方法

方法声明	说　明
String getHeader(String name)	获取一个指定头字段的值,如果请求消息中没有包含指定的头字段,则返回 null;如果请求消息中包含多个指定名称的头字段,则返回其中第一个头字段的值
Enumeration getHeaders(String name)	获取指定名称的头字段的所有值
Enumeration getHeaderNames()	获取一个包含所有头字段名称的枚举对象
long getDateHeader(String name)	获取指定头字段的值,并将其按 GMT 时间格式转换成一个代表日期/时间的长整数,这个长整数是自 1970 年 1 月 1 日 0 时 0 分 0 秒算起的以毫秒为单位的值
int getIntHeader(String name)	获取指定的头字段的值,并将其值转换为 int 类型。如果指定的名称不存在,则返回-1;如果获取的字段值无法转换为 int 类型,则抛出 NumberFormatException 异常
String getContentType()	获取 Content-Type 头字段的值
int getContentLength()	获取 Content-Length 头字段的值
String getCharacterEncoding()	获取请求消息的实体部分的字符集编码,通常从 Content-Type 头字段中进行提取

下面，通过一个案例来演示这些方法的使用。在 com.example.servlet.request 包中创建一个名为 RequestHeaderServlet 的类，该类中编写了用于获取请求头中相关信息的方法。

【例 3-4】 获取请求头信息。代码如文件 3-6 所示。

【文件 3-6】 RequestHeaderServlet.java

```
1  package com.example.servlet.request;
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5  import java.util.ArrayList;
6  import java.util.Collections;
7  import java.util.Enumeration;
8
9  import jakarta.servlet.ServletException;
10 import jakarta.servlet.annotation.WebServlet;
11 import jakarta.servlet.http.HttpServlet;
12 import jakarta.servlet.http.HttpServletRequest;
13 import jakarta.servlet.http.HttpServletResponse;
14
15 @WebServlet("/RequestHeaderServlet")
16 public class RequestHeaderServlet extends HttpServlet {
17
18   public void doGet(HttpServletRequest request,
19     HttpServletResponse response)
20       throws ServletException, IOException {
21         response.setContentType("text/html;charset=utf-8");
22         PrintWriter out = response.getWriter();
23         //获取请求消息中的所有头字段
24         Enumeration headerNames = request.getHeaderNames();
25         //使用 Lambda 表达式遍历所有请求头，
26         //并通过 getHeader()方法获取一个指定名称的头字段
27         ArrayList<String> list = (ArrayList<String>)
28         Collections.list(headerNames);
29         list.forEach((name) -> out.write(name + " : "
30              + request.getHeader(name) + "<br>"));
31     }
32     public void doPost(HttpServletRequest request,
33       HttpServletResponse response)
34       throws ServletException, IOException {
35         doGet(request, response);
36     }
37 }
```

启动 Tomcat 服务器，打开浏览器的开发者工具窗口，在浏览器的地址栏输入“http://localhost:8080/myservlet/RequestHeaderServlet”向 RequestHeaderServlet 发送请求，使用浏览器的开发者工具查看请求头信息，并与程序运行结果比对，如图 3-11 所示。

3. 获取请求消息体的相关方法

HttpServletRequest 接口的父接口 ServletRequest 定义了一系列获取请求参数和请求属性的方法，如表 3-5 所示。

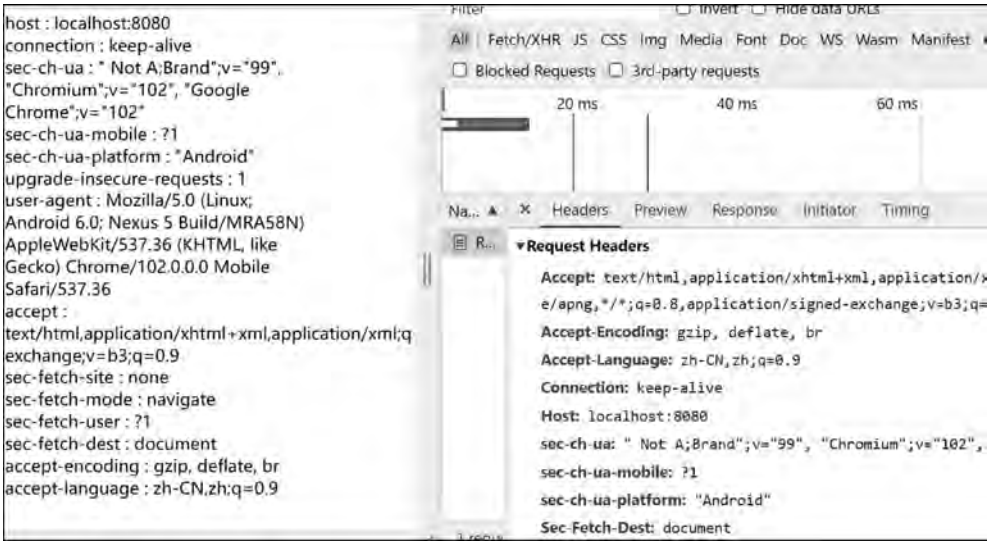

图 3-11　查看请求头信息

表 3-5　获取请求参数和请求属性的相关方法

方 法 声 明	说　　明
String getParameter(String name)	获取指定名称的请求参数的值,如果请求消息中没有指定名称的参数,则返回 null;如果指定名称的参数存在但没有设置值,则返回空串;如果请求消息中包含多个该名称指定的参数,则返回第一个出现的参数值
String[] getParameterValues(String name)	获取同一个参数名对应的所有参数值
Enumeration getParameterNames()	获取请求消息中所有参数的名字
Map getParameterMap()	将请求消息中的所有参数及其值封装为一个 Map 对象并返回该 Map 对象
void setAttribute(String name,Object obj)	将一个对象 obj 和一个名字 name 关联后存放在 ServletRequest 对象中
Object getAttribute(String name)	从 ServletRequest 对象中获取指定名称的属性对象
void removeAttribute(String name)	从 ServletRequest 对象中删除指定名称的属性对象
Enumeration getAttributeNames()	获取 ServletRequest 对象中所有的属性名

下面,通过一个案例来演示这些方法的使用。创建一个 Servlet,用来获取用户填写的表单内容并显示。

【例 3-5】 获取请求参数信息。用户在表单中填写姓名、性别并选择爱好,并将上述信息提交给服务器(Servlet)。Servlet 处理后将上述信息输出。可以按以下步骤完成此任务。

1) 创建 JSP 文件

在 src/main/webapp 文件夹下创建一个 form.jsp 表单文件,要求用户填写姓名、性别并选择爱好。代码如文件 3-7 所示。

【文件 3-7】 form.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<html>
<body>
    <form action="RequestParamServlet" method="post">
        姓名<input type="text" name="name" /><br>
        性别
        <input type="radio" name="gender" value="m" checked/>男
        <input type="radio" name="gender" value="f"/>女<br>
        爱好
        <input type="checkbox" name="hobby" value="0" />篮球
        <input type="checkbox" name="hobby" value="1" />足球
        <input type="checkbox" name="hobby" value="2" />游泳<br>
        <input type="submit" value="提交"/>
    </form>
</body>
</html>
```

2）创建 Serlvet

在 com.example.servlet.request 包中创建一个名为 RequestParamServlet 的类，获取请求参数，代码如文件 3-8 所示。

【文件 3-8】 RequestParamServlet.java

```
package com.example.servlet.request;

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("/RequestParamServlet")
public class RequestParamServlet extends HttpServlet {

  protected void doGet(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {
      String name = request.getParameter("name");
      System.out.println("姓名:" + name);
      String gender = request.getParameter("gender");
      System.out.print("性别:");
      System.out.println(gender.equals("m")?"男":"女");
      // 获取参数名为"hobby"的值
      String[] hobbys = request.getParameterValues("hobby");
      System.out.print("爱好:");
      String[] hb = {"篮球","足球","游泳"};
      for (int i = 0; i < hobbys.length; i++) {
          System.out.print(hb[Integer.parseInt(hobbys[i])] + ",");
```

```
28          }
29      }
30      protected void doPost(HttpServletRequest request,
31        HttpServletResponse response)throws ServletException,
32      IOException {
33          doGet(request, response);
34      }
35  }
```

启动 Tomcat 服务器,在浏览器的地址栏输入“http://localhost:8080/myservlet/form.jsp”,填写表单相关信息,如图 3-12 所示。

单击“提交”按钮后,可在控制台看到 Servlet 的输出信息,如图 3-13 所示。

图 3-12　填写表单信息

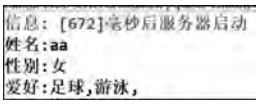

图 3-13　Servlet 在控制台输出的信息

对于文件 3-8 有以下几点说明:

(1) form.jsp 中使用了< form >标签封装表单数据。当表单提交时,< form >标签中封装的内容会被作为请求参数自动提交给 RequestParamServlet。这些请求参数的名字正是< form >标签中定义的控件的名字。

(2) 第 17 行和第 19 行分别用 request.getParameter()方法获取姓名和性别参数的值。

(3) 参数 hobby 的值可能有多个,因此第 23 行使用 getParameterValues()方法获取同名参数的多个值。通过遍历返回值数组,输出每个 hobby 参数对应的名称。

4. 请求转发器

一个 HTTP 请求可以被多个 Servlet 处理。例如,可以用一个 Servlet 实现请示文件的上传,用另一个 Servlet 实现文件批阅并生成最终的用户响应。要实现一个请求经由多个 Servlet 处理,需要用到请求转发器。Servlet 中的请求转发器由 RequestDispatcher (jakarta.servlet.RequestDispatcher)接口定义。可以通过 HttpServletRequest 接口提供的 getRequestDispatcher()方法获取 RequestDispatcher 对象。getRequestDispatcher()方法的原型如下:

```
RequestDispatcher getRequestDispatcher(String path)
```

该方法返回一个 RequestDispatcher 对象。其中参数 path 用于指定目标资源的路径,借助这个路径,请求转发器可以将当前请求转发给目标资源。如果使用相对路径,则指相对于当前 Servlet 的路径;也可以使用正斜线(/)开头的路径,表示相对于当前 Web 应用根目录的路径。

在获取到请求转发器 RequestDispatcher 对象以后,可以将当前请求通过请求转发器转发给目标资源继续处理。为此,RequestDispatcher 接口提供了一个 forward()方法,该方法可以将当前请求转发给其他 Web 资源。forward()方法的原型如下:

```
void forward(ServletRequest request, ServletResponse response)
      throws ServletException, IOException
```

forward()方法可以将当前请求转发给目标资源继续处理。需要注意的是,该方法必须在响应提交给客户端之前调用,否则会抛出 IllegalStateException 异常。请求转发的工作原理如图 3-14 所示。

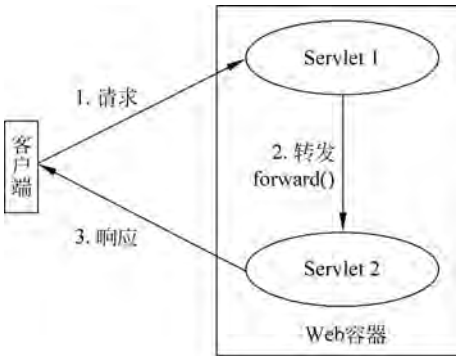

图 3-14　请求转发的工作原理

如图 3-14 所示,当 Servlet 1 处理请求后,并不生成响应,而是将该请求转发给其他的 Web 资源继续处理(图 3-14 中的 Servlet 2)。当 Servlet 2 处理请求后,生成响应并发送给客户端。注意,这里的请求和响应包含但不局限于 HTTP 请求和 HTTP 响应。了解到请求转发的工作原理后,下面通过一个案例演示请求转发器的应用。

【例 3-6】 请求转发器的应用。

分别创建两个名为 RequestForwardServlet 和 RequestDestServlet 的类。RequestForwardServelt 接收到请求后将请求转发给 RequestDestServlet。代码分别如文件 3-9 和文件 3-10 所示。

【文件 3-9】 RequestForwardServlet.java

```
1  package com.example.servlet.request;
2
3  import jakarta.servlet.ServletException;
4  import jakarta.servlet.annotation.WebServlet;
5  import jakarta.servlet.http.HttpServlet;
6  import jakarta.servlet.http.HttpServletRequest;
7  import jakarta.servlet.http.HttpServletResponse;
8  import java.io.IOException;
9
10 @WebServlet("/RequestForwardServlet")
11 public class RequestForwardServlet extends HttpServlet {
12
13   protected void doGet(HttpServletRequest request,
14       HttpServletResponse response)
15       throws ServletException, IOException {
16          System.out.println("this is servlet 1");
17          request.setAttribute("name","com.example");
18          request.getRequestDispatcher("RequestDestServlet")
19          .forward(request,response);
20   }
21
```

```
22    protected void doPost(HttpServletRequest request,
23      HttpServletResponse response)  throws ServletException,
24          IOException {
25        doGet(request, response);
26    }
27  }
```

【文件 3-10】 RequestDestServlet.java

```
1   package com.example.servlet.request;
2
3   import jakarta.servlet.ServletException;
4   import jakarta.servlet.annotation.WebServlet;
5   import jakarta.servlet.http.HttpServlet;
6   import jakarta.servlet.http.HttpServletRequest;
7   import jakarta.servlet.http.HttpServletResponse;
8   import java.io.IOException;
9   import java.io.PrintWriter;
10
11  @WebServlet("/RequestDestServlet")
12  public class RequestDestServlet extends HttpServlet {
13
14      protected void doGet(HttpServletRequest request,
15      HttpServletResponse response)
16              throws ServletException, IOException {
17          System.out.println("this is servlet 2");
18          String str = (String)request.getAttribute("name");
19          PrintWriter out = response.getWriter();
20          out.print("name is " + str);
21    }
22
23    protected void doPost(HttpServletRequest request,
24       HttpServletResponse response)  throws ServletException,
25         IOException {
26       doGet(request, response);
27    }
28  }
```

如文件 3-9 所示,请求首先到达 RequestForwardServlet。该 Servlet 处理请求后在控制台输出字符串“this is servlet 1”(第 16 行)。同时,RequestForwardServlet 在该请求域的范围内追加了 name 属性(第 17 行)。执行了上述处理后,RequestForwardServlet 并没有针对该请求生成响应,而是创建请求转发器对象(第 18 行),并调用请求转发器的 forward()方法将请求转发给 RequestDestServlet(第 19 行)。文件 3-10 描述了 RequestDestServlet 如何继续处理请求。为显示 ReqeustDestServlet 开始处理请求,首先在控制台输出字符串“this is servlet 2”(第 17 行)。在从请求域中取出 name 属性的值之后(第 18 行),RequestDestServlet 调用 out 对象的 print()方法生成对该请求的响应(第 19～20 行)。至此,请求被处理完毕。向 RequestForwardServlet 发送请求后,通过浏览器看到的响应内容如图 3-15 所示。

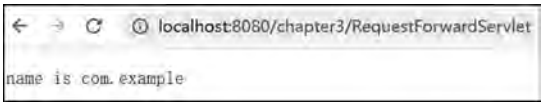

图 3-15　用浏览器查看响应内容

同时，控制台输出信息可以描述请求被两个 Servlet 处理的过程，如图 3-16 所示。

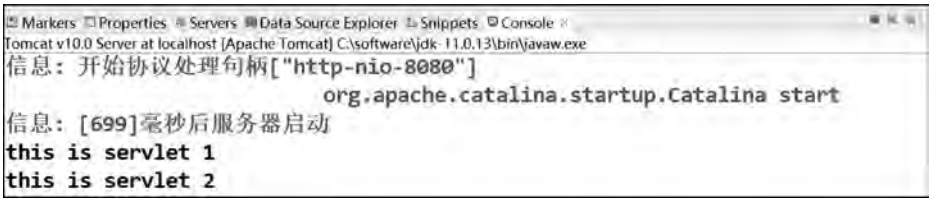

图 3-16　控制台输出的请求处理过程

3.4.2　HttpServletResponse 接口

在 Jakarta Servlet API 中定义了一个 HttpServletResponse 接口，它继承自 ServletResponse 接口，专门用来封装 HTTP 响应消息。由于 HTTP 响应消息分为状态行、响应消息头、消息体（实体主体）3 部分。因此，HttpServletResponse 接口定义了向客户端发送响应状态码、响应消息头、响应消息体的方法。

1. 发送响应状态码的方法

当 Servlet 向客户端发送响应消息时，需要在响应消息中设置状态码。为此，HttpServletResponse 接口定义了两个发送状态码的方法，具体如下所述。

1）void setStatus(int status)方法

void setStatus(int status)方法用于设置 HTTP 响应消息的状态码。由于响应状态行中的状态描述信息与状态码直接相关，而 HTTP 版本由服务器确定。因此，只要通过 setStatus(int status)方法设置状态码，即可发送状态行。正常情况下，服务器会默认产生一个状态码为 200 的状态行。合法的状态码范围为 2 ** ，3 ** ，4 ** 和 5 ** ，其中 * 表示一位非负整数。其他范围的状态码被视为属于特定容器的。

2）void sendError(int sc)throws IOException 方法

void sendError(int sc) throws IOException 方法用于发送表示错误信息的状态码。例如，404 状态码表示找不到客户端请求的资源。其中的 sc 参数表示错误信息的状态码。此外，还有一个重载的方法：

```
void sendError(int sc, String msg) throws IOException
```

这个方法除了发送错误信息的状态码外，还可以增加一条用于提示说明的文本信息，该文本信息将出现在发送给客户端的正文内容中。

2. 发送响应消息头的方法

HTTP 有很多响应头字段，HttpServletResponse 接口定义了一系列设置 HTTP 响应头的方法，如表 3-6 所示。

表 3-6 设置响应头字段的方法

方法声明	说明
void addHeader(String name, String value)	用于设置 HTTP 响应头字段。参数 name 用于指定响应头字段的名称,参数 value 用于指定响应头字段的值。addHeader()方法用于增加同名的响应头字段,setHeader()方法用于覆盖同名的响应头字段
void setHeader(String name, String value)	
void addIntHeader(String name, int value)	用于设置包含整数值的响应头。这样可以避免在使用 addHeader()和 setHeader()方法时,需要将 int 类型的值转换为 String 类型的麻烦
void setIntHeader(String name, int value)	
void setContentLength(int len)	用于设置响应消息的实体主体的大小,单位:字节。对于 HTTP 来说,这个方法就是设置 Content-Length 响应头字段的值
void setContentType(String type)	设置 Servlet 输出内容的 MIME 类型。对于 HTTP 来说,就是设置 Content-Type 响应头字段的值。例如,如果发送到客户端的响应内容是 jpeg 格式的图像,则响应头字段类型设置为"image/jpeg"。如果响应的内容是文本,则设置响应类型并指定字符编码,如"text/html;charset=UTF-8"
void setLocale(Locale loc)	设置响应消息的本地化信息。对于 HTTP 来说,就是设置 Content-Language 响应头字段和 Content-Type 头字段的字符集编码部分
void setCharacterEncoding(String charset)	设置输出内容使用的字符编码。对于 HTTP 来说,就是设置 Content-Type 响应头字段的字符集编码部分。如果没有设置 Content-Type 响应头字段,setCharacterEncoding()方法这时的字符集编码不会出现在 HTTP 响应消息中。setCharacterEncoding()方法比 setContentType()和 setLocale()方法的优先级高。它的设置结果将覆盖 setContentType()和 setLocale()方法所设置的字符编码

3. 发送响应消息体的方法

由于在 HTTP 响应消息中,大量的数据都是通过响应消息体传递的。因此,ServletResponse 接口遵循以 I/O 流形式传递数据的理念来传送响应消息体。接口中定义了两个与输出流相关的方法。

1) ServletOutputStream getOutputStream() throws IOException

ServletOutputStream getOutputStream() throws IOException 方法可返回一个能够在响应中写入二进制数据的 ServletOutputStream(字节流)对象。由于 Servlet 容器不编码二进制数据,要想输出二进制格式的响应正文,就需要使用 getOutputStream()方法。

2) PrintWriter getWriter() throws IOException

PrintWriter getWriter() throws IOException 方法返回可以向客户端发送字符文本的 PrintWriter(字符流)对象。PrintWriter 使用 getCharacterEncoding()方法返回的字符编码。如果 getCharacterEncoding()方法返回默认值 ISO-8859-1,则 getWriter() 方法会将其更新为 ISO-8859-1。

【例 3-7】 向客户端发送响应消息体。

在 src/main/java 文件夹下创建一个名为 com.example.servlet.response 的包；在包中创建一个名为 ResponseMsgServlet 的类。该类使用上述两个方法发送响应消息体，代码如文件 3-11 所示。

【文件 3-11】 ResponseMsgServlet.java

```
1  package com.example.servlet.response;
2
3  import jakarta.servlet.ServletException;
4  import jakarta.servlet.annotation.WebServlet;
5  import jakarta.servlet.http.HttpServlet;
6  import jakarta.servlet.http.HttpServletRequest;
7  import jakarta.servlet.http.HttpServletResponse;
8  import java.io.IOException;
9  import java.io.OutputStream;
10
11 @WebServlet("/ResponseMsgServlet")
12 public class ResponseMsgServlet extends HttpServlet {
13
14     protected void doGet(HttpServletRequest request,
15         HttpServletResponse response)
16         throws ServletException, IOException {
17         String msg = "this is response";
18         OutputStream output = response.getOutputStream();
19         output.write(msg.getBytes());
20     }
21     protected void doPost(HttpServletRequest request,
22         HttpServletResponse response) throws ServletException, IOException {
23         doGet(request, response);
24     }
25 }
```

启动 Tomcat 服务器，在浏览器的地址栏输入“http://localhost:8080/myservlet/ResponseMsgServlet”，浏览器显示的结果如图 3-17 所示。

图 3-17 输出响应消息

对于上述案例，可改用字符流形式输出响应消息，将第 18 行和第 19 行分别修改为：

```
PrintWriter output = response.getWriter();
output.write(msg);
```

可以得到同样的运行结果。

提示：虽然使用字符流和字节流都可以输出响应消息。但是，这两种方式不能同时使用。

4. HttpServletResponse 接口的应用

1) 解决中文乱码

计算机中的数据都是以二进制形式存储的。当传输文本时,就会发生字符和字节之间的转换。字符和字节之间的转换是通过查编码表完成的,将字符转换成字节的过程叫编码,将字节转换成字符的过程叫解码。如果编码和解码使用的码表不一致,就会产生乱码。解决中文乱码问题的思路就是在 Servlet 生成响应消息前,通知浏览器使用与 Servlet 一致的码表来对收到的响应内容进行解码。

【例 3-8】 显示中文响应消息。

在 com.example.servlet.response 包中创建一个名为 ResponseChineseCharServlet 的类,在浏览器页面显示中文响应消息。代码如文件 3-12 所示。

【文件 3-12】 ResponseChineseCharServlet.java

```
1  package com.example.servlet.response;
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5
6  import jakarta.servlet.ServletException;
7  import jakarta.servlet.annotation.WebServlet;
8  import jakarta.servlet.http.HttpServlet;
9  import jakarta.servlet.http.HttpServletRequest;
10 import jakarta.servlet.http.HttpServletResponse;
11
12 @WebServlet("/ResponseChineseCharServlet")
13 public class ResponseChineseCharServlet extends HttpServlet {
14
15     protected void doGet(HttpServletRequest request,
16     HttpServletResponse response)
17             throws ServletException, IOException {
18         response.setContentType("text/html; charset=UTF-8");
19         String msg = "这是响应消息";
20         PrintWriter out = response.getWriter();
21         out.write(msg);
22     }
23     protected void doPost(HttpServletRequest request,
24       HttpServletResponse response) throws ServletException, IOException {
25         doGet(request, response);
26   }
27 }
```

如文件 3-12 所示,在 Servlet 发送响应数据前(第 18 行),用 setContentType()方法设置 Servlet 响应的编码,通知浏览器使用同样的编码。启动 Tomcat 服务器,在浏览器的地址栏输入地址“http://localhost:8080/chapter3/ResponseChineseCharServlet”,浏览器可显示出正确的中文字符,如图 3-18 所示。

2) 页面自动跳转

在 Web 开发中,经常会遇到定时跳转页面的需求。这个功能可以利用 HTTP 响应头

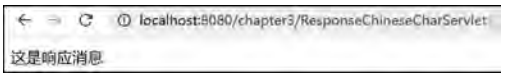

图 3-18　显示正确中文字符

中的 refresh 字段实现。它可以通知浏览器在指定的时间段内自动刷新并跳转到指定页面。

【例 3-9】 页面的定时刷新与自动跳转。

在 com. example. servlet. response 包中创建一个名为 ResponseAutoJumpServlet 的类，其功能是在收到请求 3s 后跳转到百度主页。代码如文件 3-13 所示。

【文件 3-13】 ResponseAutoJumpServlet. java

```
1  package com.example.servlet.response;
2
3  import java.io.IOException;
4
5  import jakarta.servlet.ServletException;
6  import jakarta.servlet.annotation.WebServlet;
7  import jakarta.servlet.http.HttpServlet;
8  import jakarta.servlet.http.HttpServletRequest;
9  import jakarta.servlet.http.HttpServletResponse;
10
11 @WebServlet("/ResponseAutoJumpServlet")
12 public class ResponseAutoJumpServlet extends HttpServlet {
13
14     protected void doGet(HttpServletRequest request,
15     HttpServletResponse response)
16         throws ServletException, IOException {
17         response.setHeader("Refresh","3;url = http://www.baidu.com");
18     }
19     protected void doPost(HttpServletRequest request,
20         HttpServletResponse response) throws ServletException, IOException {
21         doGet(request, response);
22     }
23 }
```

如文件 3-13 所示，调用 response 对象的 setHeader()方法来设置 refresh 响应头字段的值(第 17 行)。其中的 3 表示 3s 后发出下一个请求；url 用于指定请求的目标资源。启动 Tomcat 服务器，在浏览器的地址栏输入“http://localhost: 8080/myservlet/ResponseAutoJumpServlet”，给 Servlet 发送请求，等待 3s 后页面自动跳转到“百度”主页，如图 3-19 所示。

3) 重定向

重定向是指 Web 服务器接收到客户端的请求后，由于某些限制，无法访问请求 URL 所指向的 Web 资源，而是指定一个新的资源，指示客户端给新的资源发送请求。这一过程发生在客户端和服务器之间，用户只知道发出了请求并收到响应，重定向的整个过程对用户透明。

为实现重定向，HttpServletResponse 接口定义了一个 sendRedirect()方法。该方法用于通知客户端重新访问指定的 URL。sendRedirect()方法的原型如下。

图 3-19　页面自动跳转到“百度”主页

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

其中,参数 location 指定重定向资源的 URL。这个 URL 既可以用相对地址,也可以用绝对地址。图 3-20 说明了 sendRedirect()方法实现重定向的过程。当客户端访问资源 1 时,资源 1 响应此请求,发出重定向响应,指示客户端向资源 2 发送请求。客户端在收到重定向响应后,根据响应给出的 URL 向资源 2 发送请求,资源 2 收到请求后将响应发送给客户端。对用户来讲,整个过程只有步骤 1 和步骤 4 可见,即发出一个请求,收到一个响应。这个过程与一般的请求-响应过程相比好像毫无差异。实际上,这个过程是两次请求-响应过程。

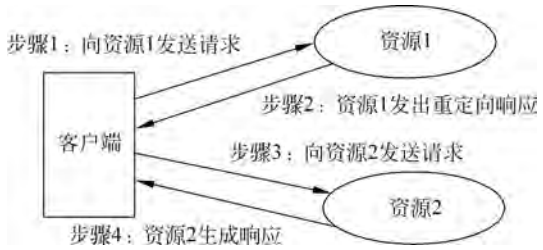

图 3-20　重定向的过程

【例 3-10】 利用重定向实现一个简单的登录验证程序。要求如下：如果用户名和密码正确,则跳转到欢迎页面；如果用户名或密码错误,则跳转回登录页面。实现步骤如下。

(1) 在 src/main/webapp 文件夹下创建两个 jsp 文件,一个用户登录页面 login.jsp,一个欢迎页面 welcome.jsp。login.jsp 代码如文件 3-14。welcome.jsp 用于输出欢迎消息,代码略。

【文件 3-14】　login.jsp

```
1  <%@ page contentType="text/html; charset=UTF-8" %>
2  <html>
3  <body>
4      <form action="LoginServlet" method="post">
5          用户名：<input type="text" name="username" /><br>
6          密　   码：
7          <input type="password" name="password" /><br>
8          <input type="submit" value="登录" />
9      </form>
10 </body>
11 </html>
```

(2) 在 com.example.servlet.response 包下创建一个名为 LoginServlet 的类,用于处理用户的登录请求,如文件 3-15 所示。

【文件 3-15】 LoginServlet.java

```
1  package com.example.servlet.response;
2
3  import jakarta.servlet.ServletException;
4  import jakarta.servlet.annotation.WebServlet;
5  import jakarta.servlet.http.HttpServlet;
6  import jakarta.servlet.http.HttpServletRequest;
7  import jakarta.servlet.http.HttpServletResponse;
8  import java.io.IOException;
9
10 @WebServlet("/LoginServlet")
11 public class LoginServlet extends HttpServlet {
12
13     protected void doGet(HttpServletRequest request,
14     HttpServletResponse response)
15         throws ServletException, IOException {
16         response.setContentType("text/html;charset=utf-8");
17         //用 HttpServletRequest 对象的 getParameter()方法获取用户名和密码
18         String username = request.getParameter("username");
19         String password = request.getParameter("password");
20         //假设用户名和密码分别为 admin 和 123
21         if (("admin").equals(username) && ("123").equals(password)) {
22             //如果用户名和密码正确,重定向到 welcome.jsp
23             response.sendRedirect("welcome.jsp");
24         } else {
25             //如果用户名和密码错误,重定向到 login.jsp
26             response.sendRedirect("login.jsp");
27         }
28     }
29     protected void doPost(HttpServletRequest request,
30     HttpServletResponse response)   throws ServletException, IOException {
31         doGet(request, response);
32     }
33 }
```

如文件 3-15 所示,第 23 行和第 26 行调用 sendRedirect()方法将请求分别重定向到 welcome.jsp 和 login.jsp。此处,这两个 JSP 页面的 URL 都使用了相对路径。

(3) 启动 Tomcat 服务器,在浏览器的地址栏输入地址“http://localhost:8080/myservlet/login.jsp”。当输入的用户名(假定为 admin)和密码(假定为 123)都正确时,浏览器显示结果如图 3-21 所示。

图 3-21 登录成功时显示的结果

如果输入的用户名或密码错误,浏览器显示结果如图 3-22 所示。

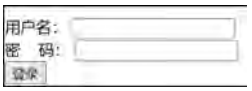

图 3-22 登录失败时显示的结果

3.4.3 ServletConfig 接口和 ServletContext 接口

1. ServletConfig 接口

在 Servlet 运行期间,经常需要一些配置信息。如文件的编码、使用 Servlet 程序的共享数据等。在初始化一个 Servlet 时,Servlet 容器会将 Servlet 的配置信息封装到一个 ServletConfig(jakarta.servlet.ServletConfig)对象中,并通过调用 Servlet 的 init(ServletConfig config)方法将 ServletConfig 对象传递给 Servlet 以完成初始化。ServletConfig 接口定义了一系列用于获取 Servlet 配置信息的方法,如表 3-7 所示。

表 3-7 ServletConfig 接口用于获取 Servlet 配置信息的方法

方法声明	说明
String getInitParameter(String name)	根据指定的名字获取初始化参数的值
Enumeration<String> getInitParameterNames()	返回一个 Enumeration 对象,其中包含所有的初始化参数的名字
ServletContext getServletContext()	返回一个代表当前 Web 应用的 ServletContext 对象
String getServletName()	返回 Servlet 的名字

【例 3-11】 创建一个 Servlet,利用 ServletConfig 接口获取 Servlet 的初始化参数。

在 src/main/java 目录下创建一个名为 com.example.servlet.sc 的包,并在该包中创建一个名为 ServletConfigDemo 的类,代码如文件 3-16 所示。

【文件 3-16】 ServletConfigDemo.java

```
1  package com.example.servlet.sc;
2
3  import java.io.IOException;
4  import java.io.PrintWriter;
5
6  import jakarta.servlet.ServletConfig;
7  import jakarta.servlet.ServletException;
8  import jakarta.servlet.annotation.WebInitParam;
9  import jakarta.servlet.annotation.WebServlet;
10 import jakarta.servlet.http.HttpServlet;
11 import jakarta.servlet.http.HttpServletRequest;
12 import jakarta.servlet.http.HttpServletResponse;
13
14 @WebServlet(urlPatterns = "/ServletConfigDemo",
15 initParams = {@WebInitParam(name = "param", value = "Hello")})
16 public class ServletConfigDemo extends HttpServlet {
17
18     protected void doGet(HttpServletRequest request,
```

```
19      HttpServletResponse response)
20          throws ServletException, IOException {
21          PrintWriter out = response.getWriter();
22          //获取 ServletConfig 对象
23          ServletConfig sc = this.getServletConfig();
24          String param = sc.getInitParameter("param");
25          out.println("param is " + param);
26      }
27      //此处省略了 doPost()方法
28  }
```

如文件 3-16 所示，第 14～15 行在@WebServlet 注解中使用 initParams 属性配置了一个名为 param 的初始化参数，并设置其值为 Hello。当前 Servlet 类的父类 GenericServlet 已定义了 getServletConfig()方法用于获取与当前 Servlet 对应的 ServletConfig 对象（第 23 行）。第 24 行调用 getInitParameter()方法获取初始化参数的值。启动 Tomcat 服务器，在浏览器的地址栏输入“http://localhost:8080/myservlet/ServletConfigDemo”，显示效果如图 3-23 所示。

图 3-23　获取初始化参数的显示效果

2. ServletContext 接口

当 Servlet 容器启动时，会为每个 Web 应用创建一个唯一的 ServletContext（jakarta.servlet.ServletContext）对象用以代表当前的 Web 应用。ServletContext 对象不仅封装了当前 Web 应用的所有信息，而且可以实现多个 Servlet 之间的数据共享。ServletContext 对象就是 JSP 的内置对象 application。下面介绍 ServletContext 在 3 个方面的应用。

1）实现多个 Servlet 之间的数据共享

一个 Web 应用程序对应一个 ServletContext 对象。而一个 Web 应用中可以包含多个 Servlet。所以，ServletContext 域的属性可以被该 Web 应用中所有的 Servlet 共享。ServletContext 接口定义了用于增加、删除和设置 ServletContext 域属性的方法，如表 3-8 所示。

表 3-8　ServletContext 接口的域属性操作方法

方法声明	说　明
Object getAttribute(String name)	根据指定的域属性的名字，返回对应的属性值
Enumeration<String> getAttributeNames()	返回一个 Enumeration 对象，其中包含了存放在 ServletContext 中的所有属性名
void setAttribute(String name, Object object)	将对象 object 与名字 name 绑定后添加到 ServletContext 域中
void removeAttribute(String name)	根据参数指定的域属性的名字，从 ServletContext 中删除对应的属性

【例 3-12】 利用 ServletContext 对象实现两个 Servlet 之间的数据共享。

创建两个 Servlet 类,ServletContextParamSetter 和 ServletContextParamGetter。这两个 Servlet 分别调用 ServletContext 接口中的方法设置和获取域属性。代码分别如文件 3-17 和文件 3-18 所示。

【文件 3-17】 ServletContextParamSetter.java

```
1  package com.example.servlet.sc;
2
3  import jakarta.servlet.ServletContext;
4  import jakarta.servlet.ServletException;
5  import jakarta.servlet.annotation.WebServlet;
6  import jakarta.servlet.http.HttpServlet;
7  import jakarta.servlet.http.HttpServletRequest;
8  import jakarta.servlet.http.HttpServletResponse;
9  import java.io.IOException;
10
11 @WebServlet("/ServletContextParamSetter")
12 public class ServletContextParamSetter extends HttpServlet {
13
14     protected void doGet(HttpServletRequest request,
15     HttpServletResponse response)
16             throws ServletException, IOException {
17         ServletContext sc = this.getServletContext();
18         sc.setAttribute("data", "this is shared data");
19   }
20   //此处省略了 doPost()方法
21 }
```

【文件 3-18】 ServletContextParamGetter.java

```
1  package com.example.servlet.sc;
2
3  //import 部分与文件 3-17 相同,此处省略
4
5  @WebServlet("/ServletContextParamGetter")
6  public class ServletContextParamGetter extends HttpServlet {
7
8      protected void doGet(HttpServletRequest request,
9      HttpServletResponse response)
10           throws ServletException, IOException {
11         ServletContext sc = this.getServletContext();
12         String param = (String)sc.getAttribute("data");
13         System.out.println(param);
14     }
15     //此处省略了 doPost()方法
16 }
```

文件 3-17 中,第 18 行调用 ServletContext 接口的 setAttribute()方法用于在 ServletContext 域中设置属性 data。文件 3-18 中,第 12 行调用 ServletContext 接口的 getAttribute()方法用于获取 ServletContext 对象的属性值。第 13 行在控制台输出获取到的

共享数据。为了验证 ServletContext 对象能否实现 Servlet 之间的数据共享，启动 Tomcat 服务器，在浏览器的地址栏输入“http://localhost:8080/myservlet/ServletContextParamSetter”，首先将共享数据存入 ServletContext 域中，然后在浏览器的地址栏输入“http://localhost:8080/myservlet/ServletContextParamGetter”，控制台的输出结果如图 3-24 所示。从控制台的输出可以确认，利用 ServletContext 能够实现 Servlet 间的数据共享。

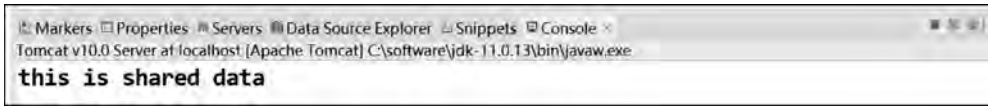

图 3-24　控制台的输出结果

2）获取 Web 应用程序的初始化参数

在 web.xml 文件中，除了可以配置 Servlet 的初始化信息，还可以配置整个 Web 应用程序的初始化信息。利用 web.xml 文件配置 Web 应用程序的初始化参数的格式如下：

```
<context-param>
    <param-name>参数名</param-name>
    <param-value>参数值</param-value>
</context-param>
```

其中，<context-param>元素是根元素<web-app>的直接子元素，并且<context-param>元素可以出现多次。<context-param>的子元素<param-name>和<param-value>分别用于指定初始化参数的名字和值。可以通过调用 ServletContext 接口的 getInitParameterNames()和 getInitParameter()方法获取初始化参数的名字和值。

【例 3-13】 利用 ServletContext 接口获取 Web 应用程序的初始化参数 name 和 address 的值。其中，web.xml 文件的内容如文件 3-19 所示。

【文件 3-19】 web.xml

```
1    <context-param>
2        <param-name>name</param-name>
3        <param-value>com.example</param-value>
4    </context-param>
5    <context-param>
6        <param-name>address</param-name>
7        <param-value>Beijing China</param-value>
8    </context-param>
```

在包 com.example.servlet.sc 中创建名为 ServletContextInitParam 的类，用于读取 Web 应用程序的初始化参数，代码如文件 3-20 所示。

【文件 3-20】 ServletContextInitParam.java

```
1  package com.example.servlet.sc;
2  //import 部分略
3
4  @WebServlet("/ServletContextInitParam")
5  public class ServletContextInitParam extends HttpServlet {
6      protected void doGet(HttpServletRequest request,
```

```
7       HttpServletResponse response)
8             throws ServletException, IOException {
9         PrintWriter out = response.getWriter();
10        ServletContext sc = this.getServletContext();
11        Enumeration paramNames = sc.getInitParameterNames();
12        ArrayList<String> names = (ArrayList<String>)
13            Collections.list(paramNames);
14        names.stream().forEach((name) -> out.println(name + " : "
15             + sc.getInitParameter(name)));
16    }
17    //此处省略了 doPost()方法
18  }
```

文件 3-20 的第 11 行调用 ServletContext 接口的 getInitParameterNames()方法获取 Web 应用程序的所有初始化参数的名字,并在第 14～15 行调用 getInitParameter()方法根据名字获取对应参数的值。启动 Tomcat 服务器,在浏览器的地址栏输入“http://localhost:8080/myservlet/ServletContextInitParam”,可在浏览器界面看到获取的初始化参数的名字和值,其运行结果如图 3-25 所示。

图 3-25　运行结果

3) 获取 Web 应用程序的资源文件

在项目开发中,有时需要读取 Web 应用程序的资源文件,如图片、配置文件等。为此,ServletContext 接口定义了一些获取 Web 资源的方法。这些方法依靠 Web 容器来实现。Web 容器根据资源文件的相对路径,返回资源文件的 IO 流、资源文件在文件系统中的绝对路径等。ServletContext 接口用于获取资源路径的方法如表 3-9 所示。

表 3-9　ServletContext 用于获取资源路径的方法

方 法 声 明	说　　明
Set<String> getResourcePaths (String path)	返回的集合对象中包含资源目录中子目录和文件的路径名称。参数 path 必须以正斜线(/)开始,指定匹配资源的部分路径
String getRealPath(String path)	返回资源文件服务器的文件系统上的真实路径(绝对路径)。参数 path 代表资源文件的虚拟路径,以正斜线(/)开始,(/)表示当前 Web 应用的根目录。如果 Web 容器不能将虚拟路径转换为文件系统的真实路径,则返回 null
URL getResource(String path) throws MalformedURLException	返回映射到某个资源文件的 URL 对象,参数 path 必须以正斜线(/)开始
InputStream getResourceAsStream (String path)	返回映射到某个资源文件的 InputStream 输入流,参数 path 的传递规则与 getResource()方法完全一致

【例 3-14】 利用 ServletContext 接口读取 Web 应用程序的资源文件。具体步骤如下：

(1) 在 src/main/java 文件夹下创建一个名为 sc.properties 的文件，其配置信息如下：

```
name = com.example
address = Beijing China
```

(2) 创建一个名为 ServletContextResourceFile 的类，代码如文件 3-21 所示。

【文件 3-21】 ServletContextResourceFile.java

```
1  package com.example.servlet.sc;
2
3  //import 部分略
4  @WebServlet("/ServletContextResource")
5  public class ServletContextResourceFile extends HttpServlet {
6
7      protected void doGet(HttpServletRequest request,
8      HttpServletResponse response)
9              throws ServletException, IOException {
10         PrintWriter out = response.getWriter();
11         ServletContext sc = this.getServletContext();
12         InputStream is = sc.getResourceAsStream("/WEB-INF
13             /classes/sc.properties");
14         Properties pros = new Properties();
15         pros.load(is);
16         out.println("name = " + pros.getProperty("name"));
17         out.println("address = " + pros.getProperty("address"));
18     }
19     //此处省略了 doPost()方法
20 }
```

项目的资源文件会最终存放在 WEB-INF/classes 目录下。在文件 3-21 的第 12、13 行指定资源文件最终的存放路径，并调用 getResourceAsStream()方法获得资源文件的输入流对象。

(3) 启动 Tomcat 服务器，在浏览器的地址栏输入“http://localhost:8080/myservlet/ServletContextResource”，可以在浏览器页面查看资源文件中配置的参数，运行结果如图 3-26 所示。

图 3-26 运行结果

3.5 会话跟踪技术

3.5.1 会话概述

日常生活中，从电话接通到电话挂断之间的通话过程就是会话。Web 应用中的会话过

程类似于打电话。它是指一个客户端和 Web 服务器之间连续发生的一系列请求和响应过程。例如,一个客户在某电商网站上购物的过程就是会话。一般来讲,客户端和服务器间的会话会产生一些数据,特定用户会话产生的数据应从属于该用户。即 A 用户在会话过程中产生的数据应从属于 A 的会话,B 用户在会话过程中产生的数据应从属于 B 的会话。

会话跟踪是 Web 应用程序开发中常用的技术。它是一种维护用户状态的方法。Web 应用程序是使用 HTTP 传输数据的。HTTP 是无状态协议,客户端每次向服务器发出的请求都会被服务器认为是新的请求。因此,服务器无法通过请求来维护用户状态以及识别特定用户。这就需要采用会话跟踪技术。常用的会话跟踪技术有 Cookie 和 session。

3.5.2 Cookie

Cookie 是由 W3C(World Wide Web Consortium,万维网联盟)提出的一种会话跟踪技术,是一小段文本信息。它可以将会话过程中产生的数据保存到客户端(如浏览器)。当用户通过浏览器第一次给服务器发送请求时,服务器会给客户端发送一些信息,如用户标识等,这些信息会保存在 Cookie 中。当再次向服务器发送请求时,浏览器会将请求和 Cookie 一同提交给服务器。服务器检查 Cookie,以此来辨别用户的状态。

服务器向客户端发送 Cookie 时,会在 HTTP 响应中增加 Set-Cookie 响应头字段。在 Set-Cookie 响应头字段中设置 Cookie 的案例如下:

```
Set-Cookie: user=admin; Path=/;
```

在上述示例中,user 表示 Cookie 的名称,admin 表示 Cookie 的值,Path 表示 Cookie 的属性。Cookie 必须以键值对的形式存在,Cookie 属性可以有多个,属性之间用分号";"和空格分隔。一般来讲,Cookie 会以文件的形式存放在客户端硬盘上;也有一种形式的 Cookie 是存放在客户端内存的,当用户关闭浏览器时即失效,这种 Cookie 称为会话 Cookie。Cookie 在浏览器和服务器间的传输过程如图 3-27 所示。

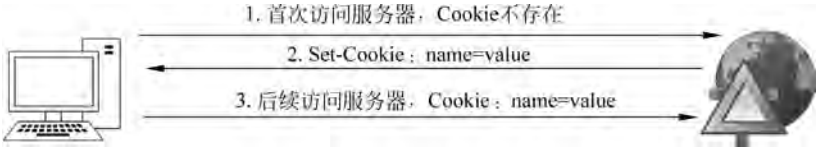

图 3-27 Cookie 在浏览器和服务器间的传输过程

为了封装 Cookie 信息,Servlet API 提供了 jakarta.servlet.http.Cookie 类。该类包含创建 Cookie 和提取 Cookie 信息的一系列方法。Cookie 类的常用方法如表 3-10 所示。

表 3-10 Cookie 类的常用方法

方法声明	说明
public Cookie(String name,String value)	构造方法。参数 name 用于指定 Cookie 的名字,参数 value 用于指定 Cookie 的值
public void setComment(String purpose)	用于设置 Cookie 的注释部分
public String getComment()	用于返回 Cookie 的注释部分

续表

方法声明	说　明
public void setMaxAge(int expiry)	设置 Cookie 在浏览器客户机上的生存时间，单位：s。参数 expiry 取大于 0 的整数，表示 Cookie 的最大生存时间；取 0 表示删除该 Cookie；取负数表示不存储该 Cookie，关闭浏览器时即删除 Cookie(会话 Cookie)
public int getMaxAge()	返回 Cookie 在浏览器客户机上的生存时间，单位：s
public void setPath(String uri)	设置 Cookie 的有效路径
public String getPath()	返回 Cookie 的有效路径
public void setSecure(boolean flag)	设置该 Cookie 是否只能使用安全协议(如 HTTPS 或 SSL)传送
public boolean getSecure()	返回该 Cookie 是否只能使用安全协议传送
public String getName()	返回 Cookie 的名称
public void setValue(String newValue)	为 Cookie 设置新的值
public String getValue()	返回 Cookie 的值
public int getVersion()	返回此 Cookie 遵守的协议版本
public void setVersion(int v)	设置该 Cookie 采用的协议版本

【例 3-15】 利用 Cookie 记录用户的登录 IP。如果用户是第一次登录，显示欢迎消息，并将其 IP 地址写入 Cookie 中。如果用户在 Cookie 有效期内再次登录，显示欢迎消息和上次登录时的 IP 地址。

实现思路：在客户端给 Servlet 发送请求时，Servlet 记录客户端的 IP 地址，并将 IP 地址以 Cookie 的形式发送给客户端。当客户端再次访问 Servlet 时，Servlet 在获取客户端的全部 Cookie 后，查找自己需要的 Cookie 并从该 Cookie 中取出 IP 地址。

在 src/main/java 目录下创建名为 com.example.servlet.cookie 的包，并在该包中创建名为 CookieDemoServlet 的类，代码如文件 3-22 所示。

【文件 3-22】 CookieDemoServlet.java

```
1  papackage com.example.servlet.cookie;
2  //import 部分略
3
4  @WebServlet("/CookieDemoServlet")
5  public class CookieDemoServlet extends HttpServlet {
6
7      protected void doGet(HttpServletRequest request,
8          HttpServletResponse response)
9          throws ServletException, IOException {
10         PrintWriter out = response.getWriter();
11         Cookie[] cookies = request.getCookies();
12         response.setContentType("text/html;charset=UTF-8");
13         boolean flag = false;
14         //判断能否获取到 Cookie
15         if(cookies!=null) {
16             for(Cookie c:cookies) {
17                 if("last".equals(c.getName())) {
18                     out.write("欢迎回来，上次访问的 IP 为：" + c.getValue());
```

```
19                          flag = true;
20                          break;
21                      }
22                  }
23          }
24          if(!flag) {
25              out.write("第一次访问,欢迎");
26              String ipAddress = request.getRemoteAddr();
27              Cookie cookie = new Cookie("last",ipAddress);
28              cookie.setMaxAge(3 * 60);
29              //将 Cookie 发送到客户端
30              response.addCookie(cookie);
31          }
32      }
33      //此处省略了 doPost()方法
34  }
```

如文件 3-23 所示,第 11 行通过 request 对象获取客户端所有的 Cookie,注意返回类型是一个 Cookie 数组。如果客户端已来访,Servlet 从客户端提交的 Cookie 中查找名为 last 的 Cookie(第 17 行)。在找到需要的 Cookie 后,Servlet 将 Cookie 中存储的 IP 地址取出并生成响应(第 18 行)。如果客户端第一次访问服务器,服务器无法获取到需要的 Cookie。因此,会输出欢迎消息(第 25 行),获取客户端 IP 地址(第 26 行),生成 Cookie 并发送到客户端(第 27～30 行)。本例中设置的 Cookie 的生存时间是 3min。

启动 Tomcat 服务器,在浏览器的地址栏输入“http://127.0.0.1:8080/myservlet/CookieDemoServlet”。由于是第一次访问,可以在浏览器中看到“第一次访问,欢迎”字样,如图 3-28 所示。刷新浏览器页面后,是第二次访问,可以看到图 3-29 所示的显示结果。由于本例设置的 Cookie 的生存时间是 3min。因此,在发送第二次请求后等待 3min,再次刷新浏览器页面,此时 Cookie 已经被移除,会看到图 3-28 所示的显示效果。

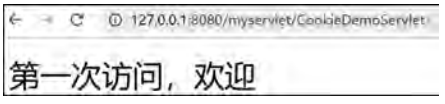

图 3-28　第一次运行的结果

图 3-29　第二次运行的结果

3.5.3　session

Cookie 技术可以将用户的信息保存在各自的客户端,并且可以在请求域内实现数据共享。但是,随着传输信息量的增加,服务器端的负载也会增加。为此,Servlet 提供了另一种会话跟踪技术——session。session 是服务器维护的一个数据结构,可以来跟踪用户的状态。

当浏览器访问 Web 应用时，Web 容器会创建一个 session 对象，其中包括了 session 对象的标识——id 属性。当客户端后续访问服务器时，只要将 id 传递给服务器，服务器就能判断出该请求是哪个客户端发送的，从而选择与之对应的 session 对象为其服务。session 对象的工作原理如图 3-30 所示。

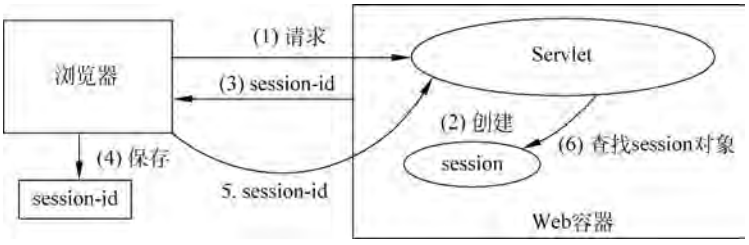

图 3-30　session 对象的工作原理

如图 3-30 所示，浏览器第一次向 Servlet 发送请求时，Web 容器会创建一个与当前请求相关的 session 对象，用以保存客户端信息。同时，服务器会将 session 对象的 id 属性以 Cookie 的形式(Set-Cookie：JSESSIONID= ***)发送给客户端，其中 *** 表示 id 的值。客户端保存此 id。当再次向 Servlet 发送请求时，浏览器会自动在请求消息头中将 Cookie (Cookie:JSESSIONID= ***)信息发送给服务器，服务器根据 id 属性查找对应的 session 对象。如果无法找到 session，则创建一个新的 session 对象。

关于 session 的其他说明如下。

1. session 的创建

session 对象是在客户端第一次访问服务器时创建的。准确来讲，只有当客户端访问 JSP、Servlet 等动态资源时才会创建 session。此外，也可以调用 HttpServletRequest 接口的 getSession()方法创建 session 对象。只访问 HTML、图片等静态资源时不会创建 session。

2. session 的生存时间

为避免 session 过多，大量占用服务器的内存空间，需要设置 session 的生存时间。超过该时间，session 对象会自动从 Web 容器的内存中删除。Tomcat 默认的 session 超时时间是 30min，即从用户最后一次发出请求开始计时，30min 内没有再次发出请求，则判断 session 超时。超时的 session 会成为垃圾对象，等待垃圾回收器将其从内存中彻底清除。设置 session 对象的超时时间共有 3 种方式。

(1) 在部署描述文件 web.xml 中设置，这种方式只对当前 Web 应用程序有效。例如，设置 session 的超时时间为 2min，代码如下：

```
<session-config>
  <session-timeout>2</session-timeout>
</session-config>
```

(2) 修改 Web 容器的默认设置，这种方式对部署在 Web 容器上的所有 Web 应用程序都有效。以 Tomcat 为例，修改 Tomcat 的 conf 目录下的 web.xml 文件中 session-config 标签的值，默认值为 30min。

```
<session-config>
  <session-timeout>30</session-timeout>
```

```
</session-config>
```

(3) 调用 HttpSession 接口的 setMaxInactiveInterval()方法。

3. session 的销毁

只要满足以下两个条件之一,session 对象就会被销毁:

① session 超时。

② 程序中调用了 HttpSession 接口的 invalidate()方法。

此外,如果采用会话 Cookie 来保存 session-id,当用户关闭浏览器时,Cookie 消失,session-id 也随之消失。这样浏览器再次连接服务器时无法找到原来的 session 对象,但并不意味着原来的 session 对象消失。

session 是与每个请求消息密切相关的。为此,HttpServletRequest 接口定义了用于获取 session 对象的 getSession()方法。具体如下:

① `public HttpSession getSession(boolean create)`

② `public HttpSession getSession()`

上面两个重载方法都可用于返回与当前请求相关的 HttpSession 对象。不同的是,第一个 getSession()方法根据传递的参数判断是否创建新的 HttpSession 对象。如果参数为 true,则在相关的 HttpSession 对象不存在时创建并返回新的 HttpSession 对象。如果参数为 false,不创建新的 HttpSession 对象,而是返回 null。第二个 getSession()方法相当于第一个方法的参数为 true 时的情况,在相关的 HttpSession 对象不存在时总是创建新的 HttpSession 对象。需要注意的是,由于 getSession()方法可能会产生发送会话标识号(session-id)的 Cookie 头字段,所以,必须在发送任何响应内容之前调用 getSession()方法。

要使用 HttpSession 对象管理会话,还需要了解 HttpSession 接口的相关方法。HttpSession 接口中的常用方法如表 3-11 所示。

表 3-11 HttpSession 接口的常用方法

方法声明	说明
void setAttribute(String name, Object value)	将一个对象 value 与名称 name 绑定,并存放到 HttpSession 对象中
Object getAttribute(String name)	从当前 HttpSession 对象中返回指定名称的属性值
void removeAttribute(String name)	从当前 HttpSession 对象中删除指定名称的属性
String getId()	返回与当前 HttpSession 对象关联的会话标识号
long getLastAccessedTime()	返回客户端最后一次发送请求的时间,这个时间是发送请求的时间与 1970 年 1 月 1 日 00:00:00 之间的差,单位: ms
void setMaxInactiveInterval(int interval)	设置当前 HttpSession 对象可空闲的最长时间间隔,即修改当前会话的默认超时时间,单位: s
long getCreationTime()	返回 HttpSession 对象的创建时间,这个时间是与 1970 年 1 月 1 日 00:00:00 之间的时间差,单位: ms
boolean isNew()	判断当前 HttpSession 对象是否是新创建的
void invalidate()	强制使 HttpSession 对象无效
ServletContext getServletContext()	返回当前 HttpSession 对象所属的 Web 应用程序对象,即代表当前 Web 应用程序的 ServletContext 对象

【例 3-16】 完善例 3-10 的用户登录程序，用 HttpSession 对象保存用户的登录信息。如果可以成功登录，则将用户名存放到 HttpSession 对象中；如果登录失败，在给出错误提示信息 3s 后，跳转到登录页。

其中，登录页面 login.jsp 内容与例 3-10 的内容相同。文件 welcome.jsp 的< body >部分改为：

```
欢迎<% = session.getAttribute("name") %>,登录成功!
<a href = "LogoutServlet">退出</a>
```

对于 LoginServlet，改名为 SessionDemoServlet，修改后的代码如文件 3-23 所示。

【文件 3-23】 SessionDemoServlet.java

```
1  package com.example.servlet.cookie;
2
3  //import 部分此处略
4  @WebServlet("/SessionDemoServlet")
5  public class SessionDemoServlet extends HttpServlet {
6
7    protected void doPost(HttpServletRequest request,
8          HttpServletResponse response)
9          throws ServletException, IOException {
10     response.setContentType("text/html;charset = utf - 8");
11     //用 HttpServletRequest 对象的 getParameter()方法
12     //获取用户名和密码
13     String username = request.getParameter("username");
14     String password = request.getParameter("password");
15     //假设用户名和密码分别为: admin 和 123
16     if (("admin").equals(username)&&("123").equals(password)) {
17         //获取 HttpSession 对象
18         HttpSession session = request.getSession();
19         //将用户名追加到 session 中
20         session.setAttribute("name", username);
21         //如果用户名和密码正确,重定向到 welcome.jsp
22         response.sendRedirect("welcome.jsp");
23     } else {
24           PrintWriter out = response.getWriter();
25           out.print("用户名或密码错误,返回< a href = \'login.jsp\'>登录</a >");
26           //如果用户名和密码错误,重定向到 login.jsp
27           response.setHeader("refresh", "3;url = login.jsp");
28     }
29   }
30   //此处省略了 doGet()方法
31 }
```

同时，编写 LogoutServlet，用于实现用户的注销功能。代码如文件 3-24 所示。

【文件 3-24】 LogoutServlet.java

```
1  package com.example.servlet.cookie;
2
3  //import 部分略
```

```
@WebServlet("/LogoutServlet")
public class LogoutServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request,
    HttpServletResponse response)
        throws ServletException, IOException {
        HttpSession session = request.getSession();
        //移除 session 中保存的属性 name
        session.removeAttribute("name");
        //强制作废 session 对象
        session.invalidate();
        //重定向到 login.jsp
        response.sendRedirect("login.jsp");
    }
    //此处省略了 doPost()方法
}
```

启动 Tomcat 服务器,在浏览器的地址栏输入“http://localhost:8080/myservlet/login.jsp”访问 login.jsp,填写用户名(admin)和密码(123)后,可以看到登录后的结果如图 3-31 所示。单击“退出”超链接,页面会返回登录页。

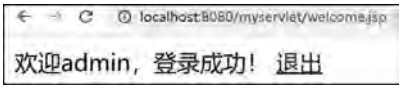

图 3-31　登录成功的页面显示

如果输入的用户名或密码错误,登录失败,浏览器显示结果如图 3-32 所示。在此消息显示 3s 后,页面会自动返回登录页。

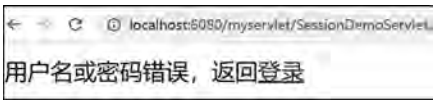

图 3-32　登录失败的页面显示