第3章 Servlet会话跟踪

CHAPTER 3

本章思维导图

本章目标

- 掌握会话跟踪的相关技术。
- 理解 Cookie 的原理。
- 掌握 Cookie 的读写方法。
- 理解 Session 的原理。
- 理解 Session 的生命周期。
- 熟练掌握 Session 的使用方法。
- 掌握 ServletConfig 的使用方法。
- 掌握 ServletContext 的使用方法。

3.1 会话跟踪技术简介

Internet 通信协议可以分为两大类：有状态协议(stateful)和无状态协议(stateless)，两者最大的差别在于客户端与服务器之间维持联机上的不同。

HTTP 即是一种无状态协议。HTTP 采用“连接—请求—应答—关闭连接”模式。当客户端发出请求时，服务器才会建立连接，一旦客户端的请求结束，服务器便会中断连接，不会一直与客户端保持联机的状态。当下一次请求发起时，服务器会把这个请求看成一个新的连接，与之前的请求无关。

对于交互式的 Web 应用，保持状态是非常重要的。一个有状态协议可以用来帮助在多个请求和响应之间实现复杂的业务逻辑。例如：在购物网站中，服务器会为每个用户分配一个购物车，购物车会一直伴随该用户的整个购物过程并且互不混淆，此种情况下，就需要为客户端和服务器之间的交互存储状态。本章所要讲述的会话跟踪技术可以解决这些问题。

会话跟踪技术是一种在客户端与服务器间保持 HTTP 状态的解决方案。从开发角度考虑，就是使上一次请求所传递的数据能够维持状态到下一次请求，并可辨认出是否为同一客户端所发送出来的。会话跟踪技术的解决方案主要分为以下几种。

- Cookie 技术。
- Session 技术。
- URL 重写技术。
- 隐藏表单域技术。

3.2 Cookie 技术


视频讲解

Cookie 技术是一种在客户端保持会话跟踪的解决方案。Cookie 是指某些网站为了辨别用户身份而储存在用户终端上的文本信息(通常经过加密)。Cookie 在用户第一次访问服务器时，由服务器通过响应头的方式发送给客户端浏览器；当用户再次向服务器发送请求时会附带上这些文本信息。图 3-1 为服务器对第一次客户端请求所响应的含有“Set-Cookie”响应头的报文信息，图 3-2 为客户端再次请求时附带的含有“Cookie”请求头的报文信息。

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=144EFED6474EA40DFE7AE585EEC25D47; Path=/chapter04/; HttpOnly
Content-Type: text/html;charset=UTF-8
Content-Length: 317
Date: Tue, 18 Nov 2014 05:28:41 GMT
```

图 3-1 第一次请求时服务器响应的 Cookie 报头

```
GET /chapter04/CookieExampleServlet HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.114
Referer: http://localhost:8080/chapter04/commonPage.jsp
Accept-Encoding: gzip,deflate,sdch
Accept-Language: zh-CN,zh;q=0.8
Cookie: JSESSIONID=144EFED6474EA40DFE7AE585EEC25D47
```

图 3-2 再次请求时附带的 Cookie 报头

服务器在接收到来自客户端浏览器的请求时，通过 Cookie 能够分析请求头的内容而得到客户端特有的信息，从而动态生成与该客户端相对应的内容。例如，在很多登录界面中可以看到“记住我”这样的选项，如果勾选则下次再访问该网站时就会自动记住用户名和密码。另外，一些网站可以根据用户的使用喜好不同浏览器设置稍有不同，进行个性化的风格设置、广告投放等，这些功能都可以通过存储在客户端的 Cookie 实现。

注意 在使用 Cookie 时，要保证浏览器接受 Cookie。对 IE 浏览器来说，设置接受 Cookie 的方法：选择浏览器的右上角“工具”菜单→“Internet 选项”→“隐私”→“高级”→“接受”选项。

Cookie 可以通过 jakarta.servlet.http.Cookie 类的构造方法来创建，其示例代码如下所示。

【示例】 Cookie 对象的创建

```
Cookie unameCookie = new Cookie("username","zhaokeling");
```

其中，Cookie 的构造方法通常需要两个参数。

- 第一个 String 类型的参数用于指定 Cookie 的属性名。
- 第二个 String 类型的参数用于指定属性值。

创建完成的 Cookie 对象可以使用 HttpServletResponse 对象的 addCookie()方法，通过增加“Set-Cookie”响应头的方式(不是替换原有的)将其响应给客户端浏览器，存储在客户端机器上，示例代码如下所示。生成的 Cookie 仅在当前浏览器有效，不能跨浏览器。

【示例】 服务器向客户端响应 Cookie

```
response.addCookie(unameCookie);
```

其中，addCookie()方法中的参数为一个 Cookie 对象。存储在客户端的 Cookie，通过 HttpServletRequest 对象的 getCookies()方法获取，该方法返回所访问网站的所有 Cookie 的对象数组，遍历该数组可以获得各个 Cookie 对象，示例代码如下所示。

【示例】 获取并遍历客户端 Cookie

```
Cookie[] cookies = request.getCookie();
if(cookies != null)
for(Cookie c : cookies){
    out.println("属性名:" + c.getName());
    out.println("属性值" + c.getValue());
}
```

在默认情况下，Cookie 只能被创建它的应用获取。Cookie 的 setPath()方法可以重新指定其访问路径，例如将其设置为可被某个应用下的某个路径共享，或被同一服务器内的所有应用共享，如下述示例所示。

【示例】 设置 Cookie 在某个应用下的访问路径

```
unameCookie.setPath("/ch03/jsp/");
```

【示例】 设置 Cookie 在服务器中所有应用下的访问路径

```
unameCookie.setPath("/");
```

Cookie 有一定的存活时间，不会在客户端一直保存。默认情况下，Cookie 保存在浏览

器内存中，在浏览器关闭时失效，这种 Cookie 也称为临时 Cookie(或会话 Cookie)。若要让其长久地保存在磁盘上，可以通过 Cookie 对象的 setMaxAge()方法设置其存活时间(以秒为单位)，时间若为正整数，表示其存活的秒数；若为负数，表示其为临时 Cookie；若为 0，表示通知浏览器删除相应的 Cookie。保存在磁盘上的 Cookie 也称为持久 Cookie。下述示例描述存活时间为 1 周的持久 Cookie。

【示例】 设置 Cookie 的存活时间

```
unameCookie.setMaxAge(7 * 24 * 60 * 60);          //参数以秒为基本单位
```

下述代码演示使用 Cookie 保存用户名和密码，当用户再次登录时在相应的文本栏显示上次登录时输入的信息。

首先，编写用于接收用户输入的 HTML 表单文件，在该例子中，没有使用 HTML 文件而是用一个 Servlet 来完成此功能，这是因为需要通过 Servlet 去读取客户端的 Cookie，而 HTML 文件无法完成此功能。

【案例 3-1】 LoginServlet.java

```
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throwsServletException, IOException {
        String cookieName = "userName";
        String cookiePwd = "pwd";
        // 获得所有 Cookie
        Cookie[] cookies = request.getCookies();
        String userName = "";
        String pwd = "";
        String isChecked = "";
        // 如果 Cookie 数组不为 null,说明曾经设置过
        // 也就是曾经登录过,那么取出上次登录的用户名和密码
        if (cookies != null) {
            // 如果曾经设置过 Cookie,checkbox 状态应该是 checked
            isChecked = "checked";
            for (inti = 0; i < cookies.length; i++) {
                // 取出登录名
                if (cookies[i].getName().equals(cookieName)) {
                    userName = cookies[i].getValue();
                }
                // 取出密码
                if (cookies[i].getName().equals(cookiePwd)) {
                    pwd = cookies[i].getValue();
                }
            }
        }
        response.setContentType("text/html;charset=GBK");
        PrintWriter out = response.getWriter();
        out.println("<html>\n");
        out.println("<head><title>登录</title></head>\n");
        out.println("<body>\n");
        out.println("<center>\n");
        out.println("<form action='CookieTest'" + " method='post'>\n");
        out.println("姓名:<input type='text'" + " name='UserName' value='"
```

```
                + userName + "'><br/>\n");
        out.println("密码:<input type='password' name='Pwd' value='" + pwd
                + "'><br/>\n");
        out.println("保存用户名和密码<input type='checkbox'"
                + "name='SaveCookie' value='Yes'" + isChecked + ">\n");
        out.println("       <br/>\n");
        out.println("       <input type=\"submit\">\n");
        out.println("    </form>\n");
        out.println("</center>\n");
        out.println("</body>\n");
        out.println("</html>\n");
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throwsServletException, IOException {
        doGet(request, response);
    }
}
```

上述代码先使用 request.getCookies()获取客户端 Cookie 数组；再遍历该数组，找到对应的 Cookie，取出用户名和密码；最后将信息显示在相应的表单控件中。

然后，编写 CookieTest.java 程序创建 Cookie 并保存到客户端。

【案例 3-2】 CookieTest.java

```
@WebServlet("/CookieTest")
public class CookieTest extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throwsServletException, IOException {
        Cookie userCookie = new Cookie("userName", request
                .getParameter("UserName"));
        Cookie pwdCookie = new Cookie("pwd", request.getParameter("Pwd"));
        if (request.getParameter("SaveCookie") != null
                &&request.getParameter("SaveCookie").equals("Yes")) {
            userCookie.setMaxAge(7 * 24 * 60 * 60);
            pwdCookie.setMaxAge(7 * 24 * 60 * 60);
        } else {
            //删除客户端对应的 Cookie
            userCookie.setMaxAge(0);
            pwdCookie.setMaxAge(0);
        }
        response.addCookie(userCookie);
        response.addCookie(pwdCookie);
        PrintWriter out = response.getWriter();
        out.println("Welcome," + request.getParameter("UserName"));
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throwsServletException, IOException {
        doGet(request, response);
    }
}
```

上述代码创建了两个 Cookie 对象，分别用来储存表单中传递过来的用户名和密码，然后根据客户端的“SaveCookie”元素的值，决定是否向客户端发送 Cookie，或者删除以前存储

的 Cookie。

启动 Tomcat 服务器，在浏览器中访问 http://localhost:8080/ch03/LoginServlet，第一次请求的访问登录页面，如图 3-3 所示。

图 3-3 第一次请求的响应报文

输入姓名和密码，选中保存复选框，单击提交按钮，显示结果如图 3-4 所示。

图 3-4 第二次请求的请求报文

当再次登录时，用户名和密码已显示，如图 3-5 所示。

图 3-5 第二次请求的响应报文

上述实例效果只限于使用同一浏览器且允许 Cookie 下访问，这是由 Cookie 本身的局限性决定的。Cookie 的缺点主要集中在其安全性和隐私保护上，主要包括以下几点。

- Cookie 可能被禁用，当用户非常注重个人隐私保护时，很可能会禁用浏览器的 Cookie 功能。
- Cookie 是与浏览器相关的，这意味着即使访问的是同一个页面，不同浏览器之间所保存的 Cookie 也是不能互相访问的。
- Cookie 可能被删除，因为每个 Cookie 都是硬盘上的一个文件，因此很有可能被用户删除。
- Cookie 的大小和个数受限，单个 Cookie 保存的数据不能超过 4KB，很多浏览器都限制一个站点最多保存 20 个 Cookie。
- Cookie 安全性不够高，所有的 Cookie 都是以纯文本的形式记录于文件中，因此如果要保存用户名和密码等信息，最好事先经过加密处理。


视频讲解

3.3 Session 技术

Session 技术是指使用 HttpSession 对象实现会话跟踪的技术，是一种在服务器端保持会话跟踪的解决方案。HttpSession 对象是 jakarta.servlet.http.HttpSession 接口的实例，也称为会话对象，该对象用来保存单个用户访问时的一些信息，是服务器在无状态的 HTTP 下用来识别和维护具体某个用户的主要方式。

3.3.1 Session 创建

HttpSession 对象会在用户第一次访问服务器时由容器创建(注意只有访问 JSP、Servlet 等程序时才会创建，只访问 HTML、IMAGE 等静态资源并不会创建)，当用户调用其失效方法(invalidate()方法)或超过其最大不活动时间时会失效。在此期间，用户与服务器之间的多次请求都属于同一个会话。

服务器在创建会话对象时，会为其分配一个唯一的会话标识——SessionId，在用户随后的请求中，服务器通过读取 SessionId 属性值来识别不同的用户，从而实现对每个用户的会话跟踪。

HttpServletRequest 接口提供了获取 HttpSession 对象的方法，如表 3-1 所示。

表 3-1 获取 HttpSession 对象的方法

方　法	描　述
getSession()	获取与客户端请求关联的当前的有效的 Session，若没有 Session 关联则新建一个
getSession(boolean create)	获取与客户端请求关联的当前的有效的 Session，若没有 Session 关联，当参数为真时，Session 被新建，为假时，返回空值

获取一个会话对象的示例代码如下所示。

【示例】 获取会话对象

```
HttpSession session = request.getSession();
HttpSession session = request.getSession(true);
```

HttpSession 接口提供了存取会话域属性和管理会话生命周期的方法，如表 3-2 所示。

表 3-2 HttpSession 接口常用方法

方　法	描　述
void setAttribute(String key,Object value)	以 key/value 的形式将对象保存在 HttpSession 对象中
Object getAttribute(String key)	通过 key 获取对象值
void removeAttribute(String key)	从 HttpSession 对象中删除指定名称 key 所对应的对象
void invalidate()	设置 HttpSession 对象失效
void setMaxInactiveInterval(int interval)	设定 HttpSession 对象的最大不活动时间(以秒为单位)，若超过这个时间，HttpSession 对象将会失效
int getMaxInactiveInterval()	获取 HttpSession 对象的有效最大不活动时间(以秒为单位)
String getId()	获取 HttpSession 对象标识 SessionID
long getCreationTime()	获取 HttpSession 对象产生的时间，单位是毫秒
long getLastAccessedTime()	获取用户最后通过这个 HttpSession 对象送出请求的时间

其中，存取会话域属性数据的方法示例如下所示。

【示例】 存取会话域属性

```
//存储会话域属性"username",值为"zkl"
session.setAttribute("username","zkl");
//通过属性名"username"从会话域中获取属性值
String uname = (String)session.getAttribute("username");
//通过属性名将属性从会话域中移除
session.removeAttribute("username");
```

HttpSession 接口用于管理会话生命周期的方法示例如下所示。

【示例】 获取会话的最大不活动时间

```
int time = session.getMaxInactiveInterval();        //单位为秒
```

会话的最大不活动时间是指，会话超过此时间不进行任何操作则会话自动失效的时间。HttpSession 对象的最大不活动时间与容器配置有关，对于 Tomcat 容器，默认时间为 1800s。实际开发中，可以根据业务需求，通过 web.xml 重新设置该时间，设置方式如下所示。

【示例】 在 web.xml 中设置会话最大不活动时间

```
<session-config>
    <session-timeout>10</session-timeout><!--单位为分钟-->
</session-config>
```

其中设置时间的单位为分钟。

除了此种方式外，还可以通过会话对象的 setMaxInactiveInterval()方法进行设置，示例如下。

【示例】 使用代码设置会话最大不活动时间

```
session.setMaxInactiveInterval(600);                //单位为秒
```

会话对象除了在超过最大不活动时间自动失效外，也可以通过调用 invalidate()方法让其立即失效，示例代码如下所示。

【示例】 设置会话立即失效

```
session.invalidate();
```

服务器在执行会话失效代码后，会清除会话对象及其所有会话域属性，同时响应客户端浏览器 Session。在实际应用中，此方法多用来实现系统的“安全退出”，使客户端和服务器彻底结束此次会话，清除所有与会话相关的信息，防止会话劫持等黑客攻击。

3.3.2 Session 生命周期

Session 生命周期经过以下几个过程。

(1) 客户端向服务器第一次发送请求时，request 中并无 SessionID。

(2) 此时服务器会创建一个 Session 对象，并分配一个 SessionID。Serssion 对象保存在服务器端，此时为新建状态，调用 session.isNew()返回 true。

(3) 当服务器端处理完毕后，会将 SessionID 通过 response 对象传回到客户端，浏览器负责保存到当前进程中。

(4) 当客户端再次发送请求时,会同时将 SessionID 发送给服务器。

(5) 服务器根据传递过来的 SessionID 将这次请求(request)与保存在服务器端的 Session 对象联系起来。此时 Session 已不处于新建状态,调用 session.isNew()返回 false。

(6) 循环执行上面的过程(3)~(5),直到 Session 超时或销毁。

Session 的生命周期和访问范围如图 3-6 所示。每个客户(如 Client1)可以访问多个 Servlet,但是一个客户的多个请求将共享一个 Session,同一 Web 应用下的所有 Servlet 共享一个 ServletContext,即 Servlet 上下文。有关 ServletContext 在本章后续章节将详细介绍。

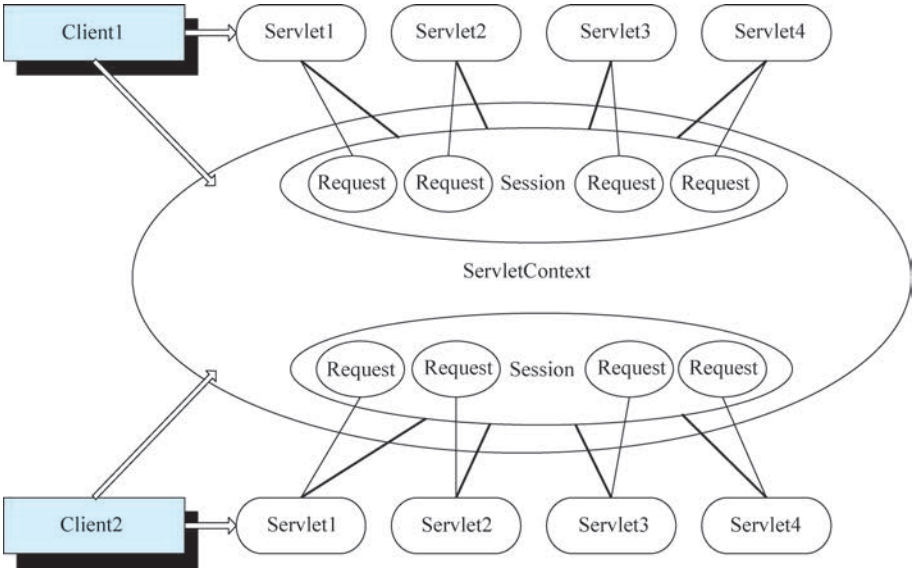

图 3-6 Session 的生命周期和访问范围

3.3.3 Session 应用

下述代码演示使用 Session 实现一个购物车。其中,案例 3-3 中的 bookChoose.jsp 页面用于让用户选择需要放入购物车的书籍,案例 3-4 中的 ShoppingCarServlet.java 用于将书籍存入购物车,案例 3-5 中的 ShoppingListServlet.java 用于从购物车中取出书籍进行显示。

【案例 3-3】 bookChoose.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>书籍选购</title>
</head>
<body>
<h2>请选择您要购买的书籍:</h2>
```

```
<form action="ShoppingCarServlet" method="post">
<p><input type="checkbox" name="book" value="JavaSE 应用与开发">JavaSE 应用与开发</p>
<p><input type="checkbox" name="book" value="JavaWeb 应用与开发">JavaWeb 应用与开发</p>
<p><input type="checkbox" name="book" value="JavaEE 应用与开发">JavaEE 应用与开发</p>
<p><input type="submit" value="提交"></p>
</form>
</body>
</html>
```

启动 Tomcat 服务器，在浏览器中访问“http://localhost:8080/ch03/bookChoose.jsp”，运行结果如图 3-7 所示。

图 3-7 bookChoose.jsp 运行结果

实现购物车功能的 ShoppingCarServlet 代码如下所示。

【案例 3-4】 ShoppingCarServlet.java

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

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        // 获取会话对象
        HttpSession session = request.getSession();

        // 从会话域中获取 shoppingCar 属性对象(即:购物车)
        // 对象定义为 Map 类型,key 为书名,value 为购买数量
        Map<String, Integer> car = (Map<String, Integer>) session
                .getAttribute("shoppingCar");
        // 若会话域中无 shoppingCar 属性对象,则实例化一个
        if (car == null) {
            car = new HashMap<String, Integer>();
        }
        // 获取用户选择的书籍
        String[] books = request.getParameterValues("book");
        if (books != null && books.length > 0) {
            for (String bookName : books) {
                // 判断此书籍是否已在购物车中
```

```
                if (car.get(bookName) != null) {
                    int num = car.get(bookName);
                    car.put(bookName, num + 1);
                } else {
                    car.put(bookName, 1);
                }
            }
        }
        // 将更新后的购物车存储在会话域中
        session.setAttribute("shoppingCar", car);
        response.sendRedirect("ShoppingListServlet");
    }
}
```

在 ShoppingListServlet 中，从会话域中取出购物车，对其存储的货物进行遍历显示，代码如下所示。

【案例 3-5】 ShoppingListServlet.java

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

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

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        HttpSession session = request.getSession();
        Map<String, Integer> car = (Map<String, Integer>) session
                .getAttribute("shoppingCar");

        if (car != null && car.size() > 0) {
            out.println("<p>您购买的书籍有:</p>");
            // 遍历显示购物车中的书籍名称和选择次数
            for (StringbookName : car.keySet()) {
                out.println("<p>" + bookName + " , " + car.get(bookName)
                        + "本</p>");
            }
        } else {
            out.println("<p>您还未购买任何书籍!</p>");
        }
        out.println("<p><a href='bookChoose.jsp'>继续购买</a></p>");
    }
}
```

用户提交书籍选择表单后的运行结果如图 3-8 所示。

单击“继续购买”链接，返回到 bookChoose.jsp 图书页面可以继续购买，结果如图 3-9 所示。

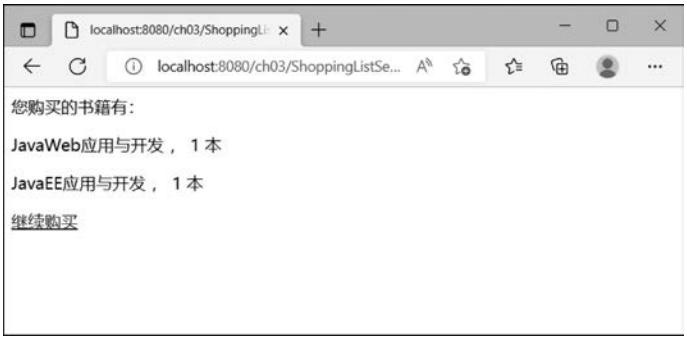

图 3-8 表单提交后的运行结果

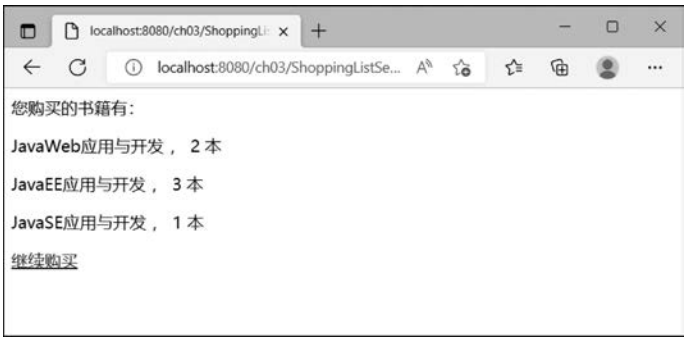

图 3-9 继续购买后的运行结果

3.4 URL 重写技术

URL 重写是指服务器程序对接收的 URL 请求重新写成网站可以处理的另一个 URL 的过程。URL 重写技术是实现动态网站会话跟踪的重要保障。在实际应用中，当不能确定客户端浏览器是否支持 Cookie 的情况下，使用 URL 重写技术可以对请求的 URL 地址追加会话标识，从而实现用户的会话跟踪功能。

例如，对于如下格式的请求地址：

```
http://localhost:8080/ch03/EncodeURLServlet
```

经过 URL 重写后，地址格式变为：

```
http://localhost:8080/ch03/EncodeURLServlet;jsessionid=24666BB458B4E0A68068CC49A97FC4A9
```

其中“jsessionid”即为追加的会话标识，服务器可以通过它来识别跟踪某个用户的访问。

URL 重写通过 HttpServletResponse 的 encodeURL()方法和 encodeRedirectURL()方法实现，其中 encodeRedirectURL()方法主要对使用 sendRedirect()方法的 URL 进行重写。URL 重写方法根据请求信息中是否包含“Set-Cookie”请求头来决定是否进行 URL 重写，若包含该请求头，会将 URL 原样输出；若不包含，则会将会话标识重写到 URL 中。

URL 重写的示例代码如下所示。

【示例】 encodeURL()方法的使用

```
out.print("<a href='" + response.encodeURL("EncodeURLServlet") + "'>链接请求</a>");
```

【示例】 encodeRedirectURL()方法的使用

```
response.sendRedirect(response.encodeRedirectURL("EncodeURLServlet"));
```

下述代码演示在浏览器 Cookie 禁用后，普通请求和重定向请求的 URL 重写方法，以及重写后会话标识“jsessionid”的跟踪情况。

图 3-10 演示对 IE 浏览器 Cookie 的禁用设置。

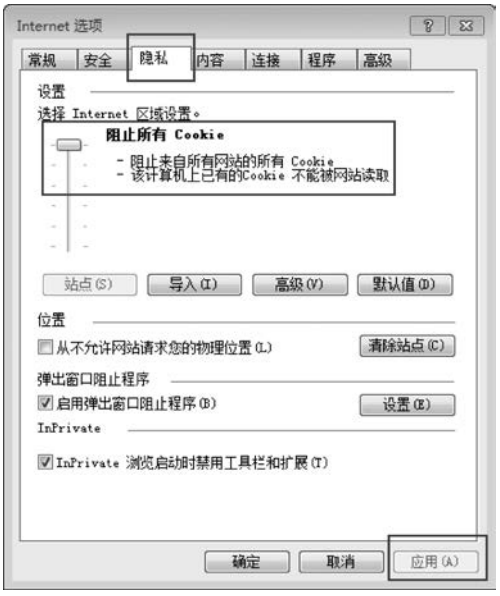

图 3-10 IE 浏览器 Cookie 的禁用设置

【案例 3-6】 UrlRewritingServlet.java

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

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        // 获取会话对象
        HttpSession session = request.getSession();

        // 对 CommonServlet 和 UseRedirectServlet 两个请求地址进行 URL 重写
        String link1 = response.encodeURL("CommonServlet");
        String link2 = response.encodeURL("UseRedirectServlet");
        // 使用超链接形式对 URL 重写地址进行请求
        out.println("<a href='" + link1 + "'>对一个普通 Servlet 的请求</a>");
        out.println("<a href='" + link2 + "'>对一个含有重定向代码的 Servlet 的请求</a>");
    }

}
```

启动服务器，在 IE 中访问“http://localhost:8080/ch03/UrlRewritingServlet”，运行结果和页面源码如图 3-11 所示。

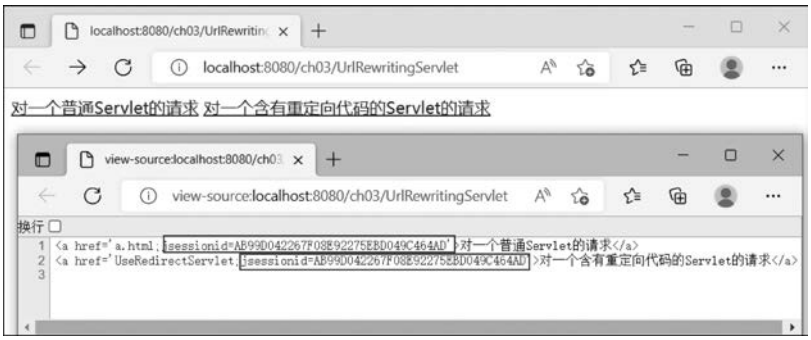

图 3-11 UrlRewritingServlet.java 运行结果和页面源码

从运行结果可以看出，两个 Servlet 请求地址经 URL 重写后，都被附加了 jsessionid 标识。下述代码演示第一个超链接 CommonSevlet 对会话标识的获取。

【案例 3-7】 CommonServlet.java

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

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        PrintWriter out = response.getWriter();
        // 获取经 URL 重写传递来的会话标识值
        String sessionId = request.getSession().getId();
        out.println(sessionId);
    }

}
```

单击第一个超链接，运行结果如图 3-12 所示。

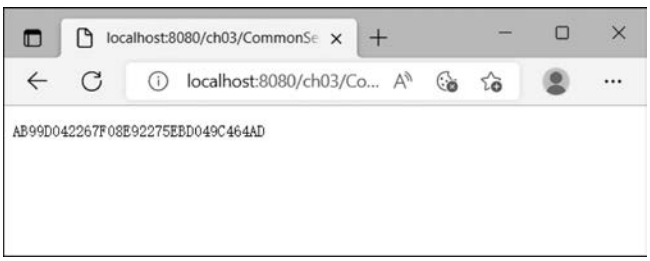

图 3-12 CommonServlet.java 运行结果

下述代码演示第二个超链接 UseRedirectServlet 对重定向 URL 的重写方法。

【案例 3-8】 UseRedirectServlet.java

```
@WebServlet("/UseRedirectServlet")
public class UseRedirectServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
```

```
            // 对重定向的 URL 进行重写
            String encodeURL = response.encodeRedirectURL("CommonServlet");
            // 进行重定向
            response.sendRedirect(encodeURL);
        }
    }
```

单击第二个超链接,可以发现运行结果与图 3-12 完全相同。

由此实例可以看出,在客户端浏览器完全禁用了 Cookie 后,通过在请求地址后附加会话标识的 URL 重写技术仍可实现会话的跟踪。但使用此种方式,有以下几个方面需要注意。

- 如果应用需要使用 URL 重写,那么必须对应用的所有请求(包括所有的超链接、表单的 action 属性值和重定向地址)都进行重写,从而将 jsessionid 维持下来。
- 浏览器对 URL 地址长度有限制,所以在对含有查询参数的 GET 请求进行 URL 重写时,需要注意其总长度。
- 由于静态页面不能进行会话标识的传递,因此所有的 URL 地址都必须为动态请求地址。

3.5 隐藏表单域技术

利用 Form 表单的隐藏表单域技术,可以在完全脱离浏览器对 Cookie 的使用限制以及在用户无法从页面显示看到隐藏标识的情况下,将标识随请求一起传送给服务器处理,从而实现会话的跟踪。

设置隐藏表单域的示例代码如下所示。

【示例】 在 Form 表单中定义隐藏域

```
<form action = "xx" method = "post">
    <input type = "hidden" name = "userID" value = "10010">
    <input type = "submit" value = "提交">
</form>
```

在服务器端通过 HttpServletRequest 对象获取隐藏域的值,示例代码如下所示。

【示例】 隐藏域的获取

```
String flag = request.getParameter("userID");
```

由于使用隐藏表单域技术进行会话跟踪的基本前提是只能通过 Form 表单来传递标识信息,因此此技术在实际应用中并不常用。

3.6 ServletConfig 接口

jakarta.servlet.ServletConfig 接口的定义为:

```
public abstract interface jakarta.servlet.ServletConfig
```

容器在初始化一个 Servlet 时,将为该 Servlet 创建一个唯一的 ServletConfig 对象,并将这个 ServletConfig 对象通过 init(ServletConfig config)方法传递并保存在此 Servlet 对象中。

ServletConfig 接口的主要方法如表 3-3 所示。

表 3-3 ServletConfig 接口的主要方法

方 法	方法描述
getInitParameter(String param)	根据给定的初始化参数名称，返回参数值，若参数不存在，返回 null
getInitParameterNames()	返回一个 Enumeration 对象，里面包含了所有的初始化参数名称
getServletContext()	返回当前 ServletContext()对象
getServletName()	返回当前 Servlet 的名字，即@WebServlet 的 name 属性值。如果没有配置这个属性，则返回 Servlet 类的全限定名

使用 ServletConfig 接口中的方法主要可以访问两项内容：Servlet 初始化参数和 ServletContext 对象。前者通常由容器从 Servlet 的配置属性中读取(如 initParams 或<init-param>所指定的参数)；后者为 Servlet 提供有关容器的信息。

在实际应用中经常会遇到一些随需求不断变更的信息，例如数据库的链接地址、账号、密码等，若将这些信息硬编码到 Servlet 类中，则信息的每次修改都将使 Servlet 重新编译，这将大大降低系统的可维护性。这时可以采用 Servlet 的初始参数配置来解决这类问题。

下述示例演示通过 web.xml 文件配置初始化参数和使用 ServletConfig 对象获取初始化参数。

【示例】 Servlet 初始化参数在 web.xml 文件中的配置

```
<servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>com.zkl.ch03.servlet.HelloServlet</servlet-class>
        <init-param>
            <param-name>url</param-name>
            <param-value>jdbc:oracle:thin:@localhost:1521:orcl</param-value>
        </init-param>
        <init-param>
            <param-name>user</param-name>
            <param-value>zkl</param-value>
        </init-param>
        <init-param>
            <param-name>password</param-name>
            <param-value>123456</param-value>
        </init-param>
</servlet>
```

在上述代码中，配置 Servlet 时使用<init-param>元素设定初始化参数信息，该元素有两个子元素：<param-name>子元素设置初始化参数名，<param-value>子元素设置初始化参数值。

【示例】 Servlet 初始化参数的获取

```
public class HelloServlet extends HttpServlet {

    public void init(ServletConfig config) throws ServletException {
        String url = config.getInitParameter("url");
        String user = config.getInitParameter("user");
        String password = config.getInitParameter("password");
        try {
            Connection conn = DriverManager.getConnection(url, user, password);
```

```
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    ...
```

通过上述示例可以看出，在项目开发和应用过程中若要对数据库连接信息进行变更，只需修改 web.xml 中的 Servlet 配置属性即可，而不需要修改代码和重新编译代码。

3.7 ServletContext 接口

jakarta.servlet.ServletContext 接口的定义为：

```
public abstract interface jakarta.servlet.ServletContext
```

ServletContext 也称为 Servlet 上下文，代表当前 Servlet 运行环境，是 Servlet 与 Servlet 容器之间直接通信的接口。Servlet 容器在启动一个 Web 应用时，会为该应用创建一个唯一的 ServletContext 对象供该应用中的所有 Servlet 对象共享，Servlet 对象可以通过 ServletContext 对象来访问容器中的各种资源。

获得 ServletContext 对象可以通过以下两种方式。

(1) 通过 ServletConfig 接口的 getServletContext()方法获得 ServletContext 对象。

(2) 通过 GenericServlet 抽象类的 getServletContext()方法获得 ServletContext 对象，实质上该方法也是调用了 ServletConfig 接口的 getServletContext()方法。

ServletContext 接口提供了以下几种类型的方法。

- 获取应用范围的初始化参数的方法。
- 存取应用域属性的方法。
- 获取当前 Web 应用信息的方法。
- 获取当前容器信息和向容器输出日志的方法。
- 获取服务器文件资源的方法。

下述各小节将依次对其进行详细介绍。


视频讲解

3.7.1 获取应用范围的初始化参数

在 Web 应用开发中可以通过 web.xml 配置应用范围的初始化参数，容器在应用程序加载时会读取这些配置参数并存入 ServletContext 对象中。ServletContext 接口提供了这些初始化参数的获取方法，如表 3-4 所示。

表 3-4 ServletContext 接口获取应用范围的初始化参数的方法

方 法	方法描述
getInitParameter(String name)	返回 Web 应用范围内指定的初始化参数值。在 web.xml 中使用<context-param>元素表示应用范围内的初始化参数
getInitParameterNames()	返回一个包含所有初始化参数名称的 Enumeration 对象

下述代码演示 Web 应用范围的初始化参数的配置及获取。

首先，在 web.xml 配置文件中配置 Web 应用范围的初始化参数，该参数通过<content-

param >元素来指定,代码如下所示。

【案例 3-9】 web. xml

```
<?xml version = "1.0" encoding = "UTF - 8"?>
< web - app xmlns:xsi = "http://www.w3.org/2001/XMLSchema - instance" ...
    version = "5.0">
    < display - name > ch03 </display - name >
    < context - param >
        < param - name > webSite </param - name >
        < param - value > www.baidu.com </param - value >
    </context - param >
    < context - param >
        < param - name > adminEmail </param - name >
        < param - value > zkl@qq.com </param - value >
    </context - param >
    < welcome - file - list >
        < welcome - file > index.html </welcome - file >
        < welcome - file > index.jsp </welcome - file >
    </welcome - file - list >
...
```

然后,通过使用 ServletContext 对象获取初始化参数的值,其代码如下所示。

【案例 3-10】 ContextInitParamServlet. java

```
@WebServlet("/ContextInitParamServlet")
public class ContextInitParamServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    public ContextInitParamServlet() {
        super();
    }
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        // 设置响应到客户端的 MIME 类型及编码方式
        response.setContentType("text/html;charset = UTF - 8");
        // 使用 ServletContext 对象获取所有初始化参数
        Enumeration < String > paramNames = super.getServletContext()
                .getInitParameterNames();
        // 使用 ServletContext 对象获取某个初始化参数
        String webSite = super.getServletContext().getInitParameter("webSite");
        StringadminEmail = super.getServletContext().getInitParameter(
                "adminEmail");
        // 获取输出流
        PrintWriter out = response.getWriter();
        // 输出响应结果
        out.print("< p >当前 Web 应用的所有初始化参数:");
        while (paramNames.hasMoreElements()) {
            String name = paramNames.nextElement();
            out.print(name + " ");
        }
        out.println("</p >< p > webSite 参数的值:" + webSite);
        out.println("</p >< p > adminEmail 参数的值:" + adminEmail + "</p >");
    }

}
```

启动服务器，在 IE 中访问“http://localhost:8080/cha03/ContextInitParamServlet”，运行结果如图 3-13 所示。

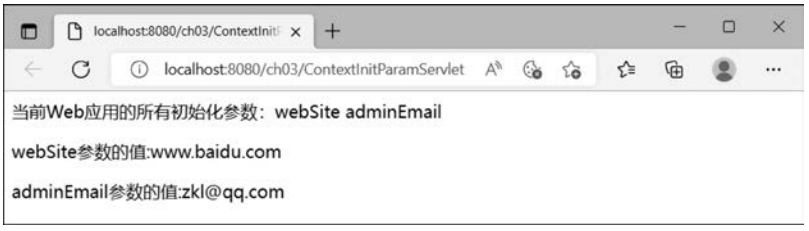

图 3-13 ContextInitParamServlet 运行结果


视频讲解

3.7.2 存取应用域属性

ServletContext 对象可以理解为容器内的一个共享空间，可以存放具有应用级别作用域的数据，Web 应用中的各个组件都可以共享这些数据。这些共享数据以 key/value 的形式存放在 ServletContext 对象中，并以 key 作为其属性名被访问。具体的应用域属性的存取方法如表 3-5 所示。

表 3-5 ServletContext 对象应用域属性的存取方法

方　法	方法描述
setAttribute(String name,Object object)	把一个对象和一个属性名绑定并存放到 ServletContext 中，参数 name 指定属性名，参数 Object 表示共享数据
getAttribute(String name)	根据参数给定的属性名，返回一个 Object 类型的对象
getAttributeNames()	返回一个 Enumeration 对象，该对象包含了所有存放在 ServletContext 中的属性名
removeAttribute(String name)	根据参数指定的属性名，从 ServletContext 对象中删除匹配的属性

注意 应用域具有以下两层含义：一是表示由 Web 应用的生命周期构成的时间段；二是表示在 Web 应用范围内的可访问性。

下述代码通过一个网站访问计数的例子演示应用域属性的存取方法。

【案例 3-11】 ContextAttributeServlet.java

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

    public ContextAttributeServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        //设置响应到客户端的文本类型
        response.setContentType("text/html;charset=UTF-8");
        //获取 ServletContext 对象
        ServletContext context = super.getServletContext();
        //从 ServletContext 对象获取 count 属性存放的计数值
```

```
            Integer count = (Integer) context.getAttribute("count");
            if (count == null) {
                count = 1;
            } else {
                count = count + 1;
            }
            //将更新后的数值存放到 ServletContext 对象的 count 属性中
            context.setAttribute("count", count);
            //获取输出流
            PrintWriter out = response.getWriter();
            //输出计数信息
            out.println("<p>本网站目前访问人数是: " + count + "</p>");
        }
    }
```

再新建一个 Servlet 命名为 ContextAttributeOtherServlet，代码内容与 ContextAttributeServlet 完全相同，启动服务器，在 IE 中先后访问“http://localhost:8080/ch03/ContextAttributeServlet”与“http://localhost:8080/ch03/ContextAttributeOtherServlet”，运行结果如图 3-14 所示。

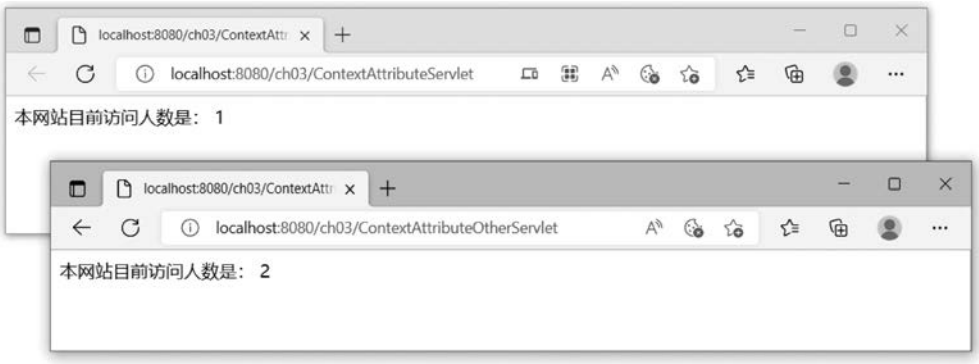

图 3-14 ContextAttributeServlet 与 ContextAttributeOtherServlet 运行结果

由上述代码可以看出，对于存放在 ServletContext 对象中的属性 count，不同的 Servlet 都可以通过 ServletContext 对象对其进行访问和修改，并且一方的修改会影响另一方获取的数据值，因此在多线程访问情况下，需要注意数据的同步问题。

3.7.3 获取当前应用信息

ServletContext 对象还包含有关 Web 应用的信息，例如：当前 Web 应用的根路径、应用的名称、应用组件间的转发，以及容器下其他 Web 应用的 ServletContext 对象等。具体信息的获取如表 3-6 所示。

表 3-6 ServletContext 接口访问当前应用信息的方法

方 法	方 法 描 述
getContextPath()	返回当前 Web 应用的根路径
getServletContextName()	返回 Web 应用的名称，即< web-app >元素中< display-name >元素的值
getRequestDispatcher(String path)	返回一个用于向其他 Web 组件转发请求的 RequestDispatcher 对象
getContext(String uripath)	根据参数指定的 URL 返回当前 Servlet 容器中其他 Web 应用的 ServletContext()对象，URL 必须是以“/”开头的绝对路径

下述代码演示获取当前应用信息的方法的使用。

【案例 3-12】 ContextAppInfoServlet.java

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

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        // 设置响应到客户端的文本类型为 HTML
        response.setContentType("text/html;charset=UTF-8");
        // 获取当前 ServletContext 对象
        ServletContext context = super.getServletContext();
        // 获取当前 Web 应用的上下文根路径
        String contextPath = context.getContextPath();
        // 获取当前 Web 应用的名称
        String contextName = context.getServletContextName();
        // 获取输出流
        PrintWriter out = response.getWriter();
        out.println("<P>当前 Web 应用的上下文根路径是:" + contextPath + "</p>");
        out.println("<p>当前 Web 应用的名称是:" + contextName + "</p>");
    }
}
```

启动服务器,在 IE 中访问“http://localhost:8080/ch03/ContextAppInfoServlet”,运行结果如图 3-15 所示。

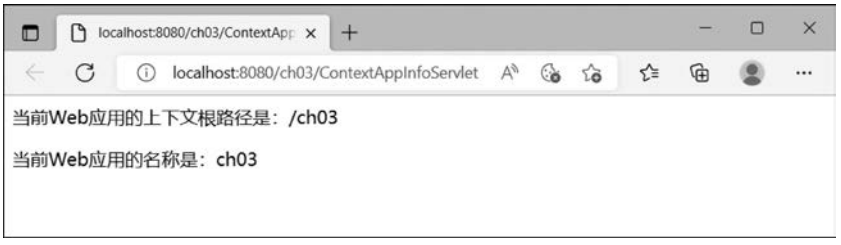

图 3-15 ContextAppInfoServlet.java 运行结果

注意 Tomcat 服务器默认不能跨应用访问,因此若要使用当前应用的 ServletContext 对象的 getContext(String uripath)方法访问同一容器下的其他应用,需要将"%TOMCAT_HOME%/conf/context.xml"文件中的“< Context >”的属性"crossContext"设为"true",例如:“< Context crossContext="true">”。

3.7.4 获取容器信息

ServletContext 接口还提供了获取有关容器信息和向容器输出日志的方法,如表 3-7 所示。

表 3-7 ServletContext 接口获取容器信息和向容器输出日志的方法

方 法	方法描述
getServerInfo()	返回 Web 容器的名字和版本
getMajorVersion()	返回 Web 容器支持的 Servlet API 的主版本号
getMinorVersion()	返回 Web 容器支持的 Servlet API 的次版本号
log(String msg)	用于记录一般的日志
log(String message,Throwable throw)	用于记录异常的堆栈日志

ServletContext 接口中常用方法的具体使用如下述代码所示。

【案例 3-13】 ContextLogInfoServlet.java

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

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // 设置响应到客户端 MIME 类型和字符编码方式
        response.setContentType("text/html;charset=UTF-8");
        // 获取 ServletContext 对象
        ServletContext context = super.getServletContext();
        // 获取 Web 容器的名字和版本
        String serverInfo = context.getServerInfo();
        // 获取 Web 容器支持的 Servlet API 的主版本号
        int majorVersion = context.getMajorVersion();
        // 获取 Web 容器支持的 Servlet API 的次版本号
        int minoVersion = context.getMinorVersion();
        // 记录一般的日志
        context.log("自定义日志信息");
        // 记录异常的堆栈日志
        context.log("自定义错误日志信息",new Exception("异常堆栈信息"));
        // 获取输出流
        PrintWriter out = response.getWriter();
        out.println("<p>Web 容器的名字和版本为:" + serverInfo + "</p>");
        out.println("<p>Web 容器支持的 Servlet API 的主版本号为:" + majorVersion + "</p>");
        out.println("<p>Web 容器支持的 Servlet API 的次版本号为:" + minoVersion + "</p>");

    }

}
```

启动服务器,在浏览器中访问 http://localhost:8080/ch03/ContextLogInfoServlet,运行结果如图 3-16 所示。

图 3-16 ContextLogInfoServlet.java 运行结果

ContextLogInfoServlet 中记录的日志信息在 Tomcat 服务器控制台显示效果如图 3-17 所示。

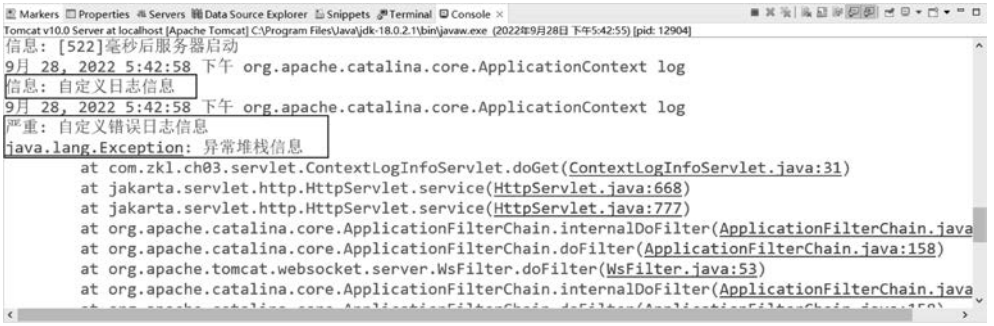

图 3-17　日志信息在 Tomcat 服务器控制台显示效果

3.7.5　获取服务器文件资源

使用 ServletContext 接口可以直接访问 Web 应用中的静态内容文件，例如 HTML、GIF、Properties 文件等，同时还可以获取文件资源的 MIME 类型以及其在服务器中的真实存放路径，具体方法如表 3-8 所示。

表 3-8　ServletContext 接口访问服务器端文件系统资源的方法

方　　法	方 法 描 述
getResourceAsStream(String path)	返回一个读取参数指定的文件的输入流，参数路径必须以"/"开头
getResource(String path)	返回由 path 指定的资源路径对应的一个 URL 对象，参数路径必须以"/"开头
getRealPath(String path)	根据参数指定的虚拟路径，返回文件系统中的一个真实的路径
getMimeType(String file)	返回参数指定的文件的 MIME 类型

下述代码演示使用 ServletContext 接口访问当前 ch03 应用中 images 目录下的 mypic.jpg 文件。

【案例 3-14】　ContextFileResourceServlet.java

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

    public ContextFileResourceServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        // 设置响应到客户端 MIME 类型和字符编码方式
        response.setContentType("text/html;charset=UTF-8");
        // 获取 ServletContext 对象
        ServletContext context = super.getServletContext();

        // 获取用于读取指定静态文件的输入流
```

```
            InputStream is = context.getResourceAsStream("/images/mypic.jpg");
            // 获取一个映射到指定静态文件路径的 URL
            URL url = context.getResource("/images/mypic.jpg");
            // 从 URL 对象中获取文件的输入流
            InputStream in = url.openStream();
            // 比较使用上述两种方法获取同一文件输入流的大小
            boolean isEqual = is.available() == in.available();

            // 根据指定的文件虚拟路径获取真实路径
            String fileRealPath = context.getRealPath("/images/mypic.jpg");
            // 获取指定文件的 MIME 类型
            String mimeType = context.getMimeType("/images/mypic.jpg");

            // 获取输出流
            PrintWriter out = response.getWriter();
            out.println("<p>两种方式获取同一文件输入流的大小是否相等:" + isEqual + "</p>");
            out.println("<p>虚拟路径"/images/mypic.jpg"的真实路径为:" + fileRealPath + "</p>");
            out.println("<p>mypic.jpg 的 MIME 类型为:" + mimeType + "</p>");
            out.close();
        }

    }
```

启动服务器，在浏览器中访问 http://localhost:8080/ch03/ContextFileResourceServlet，运行结果如图 3-18 所示。

图 3-18 ContextFileResourceServlet 运行结果

本章总结

- Cookie 是保存在客户端的小段文本。
- 通过请求可以获得 Cookie，通过响应可以写入 Cookie。
- Session 是浏览器与服务器之间的一次通话，它包含浏览器与服务器之间的多次请求、响应过程。
- Session 可以在用户访问一个 Web 站点的多个页面时共享信息。
- 在 Servlet 中通过 request.getSession()获取当前 Session 对象。
- 关闭浏览器、调用 Session 的 invalidate()方法或者等待 Session 超时都可以使 Session 失效。
- HttpSession 使用 getAttribute()和 setAttribute()方法读写数据。

- ServletContext 是运行 Servlet 的容器。
- 在 Servlet 中可以通过 getServletContext()方法获取 ServletContext 实例。
- ServletContext 使用 getAttribute()和 setAttribute()方法读写数据。

本章习题

1. 下列关于 Cookie 的说法正确的是_____。(多选)

A. Cookie 保存在客户端　　B. Cookie 可以被服务器端程序修改

C. Cookie 中可以保存任意长度的文本　　D. 浏览器可以关闭 Cookie 功能

2. 写入和读取 Cookie 的代码分别是_____。

A. request. addCookies()和 response. getCookies()

B. response. addCookie()和 request. getCookie()

C. response. addCookies()和 request. getCookies()

D. response. addCookie()和 request. getCookies()

3. HttpServletRequest 的_____方法可以得到会话。(多选)

A. getSession()　　B. getSession(boolean)

C. getRequestSession()　　D. getHttpSession()

4. 下列选项可以关闭会话的是_____。(多选)

A. 调用 HttpSession 的 close()方法

B. 调用 HttpSession 的 invalidate()方法

C. 等待 HttpSession 超时

D. 调用 HttpServletRequest 的 getSession(false)方法

5. 在 HttpSession 中写入和读取数据的方法是_____。

A. setParameter()和 getParameter()　　B. setAttribute()和 getAttribute()

C. addAttribute()和 getAttribute()　　D. set()和 get()

6. 关于 HttpSession 的 getAttribute()方法和 setAttribute()方法,说法正确的是_____。(多选)

A. getAttribute()方法返回类型是 String

B. getAttribute()方法返回类型是 Object

C. setAttribute()方法保存数据时如果名字重复会抛出异常

D. setAttribute()方法保存数据时如果名字重复会覆盖以前的数据

7. 设置 session 的有效时间(也叫超时时间)的方法是_____。

A. setMaxinactivelnterval(int interval)

B. getAttributeName()

C. setAttrlbuteName(Strlng name,java. lang. Object value)

D. getLastAccessedTime()

8. 使 HttpSession 失效的三种方式是_____、_____、_____。

9. 测试在其他浏览器下 session 的生命周期,如 Firefox、chrome 等。