第5章 会话管理

本章要点：

- 能理解会话的概念和原理。
- 能理解 Cookie 的生命周期，会熟练使用浏览器查看 Cookie 数据。
- 能理解 Cookie 的运行机制，会熟练编写 Cookie 相关代码。
- 能理解 Session 的生命周期，会熟练使用浏览器查看 Session 的 Id 数据。
- 能理解 Session 的运行机制，会熟练编写 Session 相关代码。

5.1 会话概述

在计算机术语中，会话是指一个终端用户与交互系统进行通信的过程，例如，在 ATM 机上，用户从输入账户密码进入操作系统到退出操作系统就是一个会话过程。

在 Java Web 应用中，通常客户端浏览器通过“请求响应”模式访问同一个网站的不同页面，从开始访问这个网站直到结束的整个过程称为一次会话。一次会话过程，包含了多次客户端与服务器端之间的请求响应过程。如图 5-1 所示，图中的三次请求响应可称为一次会话。

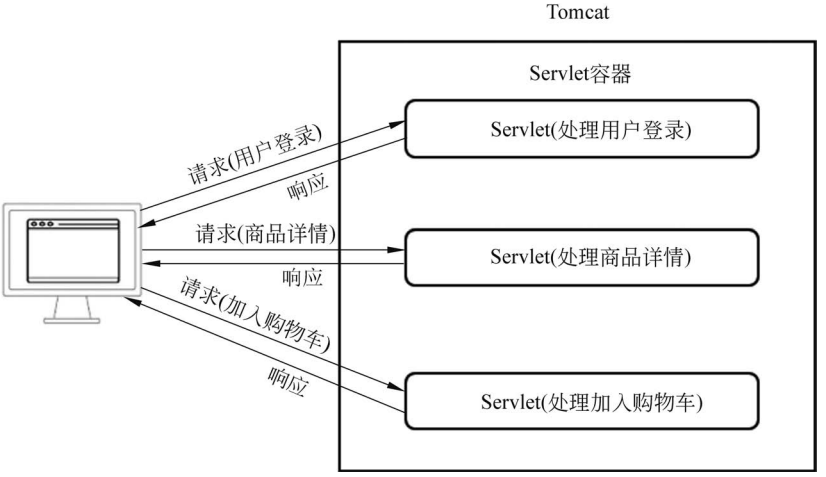

图 5-1　会话的概念

从理论上来说，每次客户端与服务器端之间的请求响应是独立的，与之前后的请求响应无关。主要原因是请求响应是基于 HTTP 通信协议，而在 HTTP 通信协议规范下每个请求都是完全独立的，每个请求包含了处理这个请求所需的完整数据，发送请求不涉及状态变更。因此，通常认为 HTTP 通信协议是无状态的。

在图 5-1 中，从基于 HTTP 通信协议的请求响应来分析。

第一次用户登录请求，浏览器会在请求数据包中封装登录表单中的用户名、密码等信息发送给服务器端。

第二次查看商品详情请求，浏览器会在请求数据包中封装用户单击的商品编号等信息发送给服务器端。

第三次添加商品至购物车请求，浏览器会在请求数据包中封装用户购买的商品编号、数量、金额等信息发送给服务器端。

因为每次的请求都是独立的，所以在第三次请求中，服务器端接收的请求数据中仅包含商品信息，不包含用户信息，无法把商品放入用户购物车。即服务器端不能为不同的客户端处理和响应不同的信息，不能支撑交互式动态网站的实现。

若从会话角度来说，上述三个请求显然属于同一个会话。对服务器端来说，在同一个会话中交互的是同一个用户，如果能“识别”这个用户，那么就能解决 HTTP 通信协议无状态的问题，即可实现交互式动态网站。

为了使服务器端具有“识别”不同客户端的能力，产生了 Cookie 和 Session 两种技术，它们都属于 Java Web 应用中的会话管理技术。

5.2 Cookie 技术

■ 5.2.1 Cookie 概述

HTTP 通信协议本身是无状态的，这与 HTTP 通信协议原本的设计目的是相符的。在早期的互联网中，客户端只需要简单地向服务器端发送请求，服务器端依据请求内容返回相应的响应内容。无论是客户端还是服务器端都没有必要记录彼此之间的交互行为，每次请求之间都是独立的，好比一个顾客和一个非会员制超市之间的关系一样。

然而随着互联网的普及，出现了个性化的交互式动态网站服务的需求。这种需求一方面促使 HTML 逐步添加了表单、脚本、DOM 等客户端行为；另一方面在服务器端则出现了 CGI 规范以响应客户端的动态请求，作为传输载体的 HTTP 通信协议也添加了文件上传、Cookie 等特性。其中 Cookie 就是为了解决 HTTP 通信协议无状态的缺陷所做出的努力。

Cookie 是设计交互式动态网页的一项重要技术，它可以将一些简短的数据存储在客户端，这些数据变量称为 Cookie。当客户端浏览器向服务器端发送请求时，浏览器会自动在请求中携带属于当前服务器端的 Cookie 数据，服务器端在接收请求时可以读取 Cookie 数据，从而获得了“识别”不同客户端的能力。

■ 5.2.2　Cookie 的运行机制

超市为了吸引顾客往往会采用会员制，给会员提供折扣、积分、节日赠品等多项福利。通常顾客首次来超市时，超市会制作并发放会员卡给顾客。后续顾客来超市购物时带上会员卡，就可以享受超市提供的会员福利了。

Cookie 运行机制的基本原理与上述场景相似，客户端对应顾客，服务器端对应超市，Cookie 对应会员卡。

Cookie 的创建和分发过程与超市会员卡场景中的会员卡发放相似。Cookie 是通过扩展 HTTP 通信协议来实现的。Cookie 是服务器端在响应客户端请求时，在 HTTP 的响应头中加入的特定数据信息。客户端浏览器在接收服务器端响应内容后，读取特定数据信息并存储。其中“在 HTTP 的响应头中加入特定数据信息”需要开发人员编写代码实现，“客户端浏览器读取特定数据信息并存储”与开发人员无关，浏览器具备这个功能。

Cookie 的使用过程与顾客去超市时携带并出示会员卡相似。客户端浏览器在访问网站时，会按照一定的规则自动把 Cookie 添加到请求中发送给服务器端。具体过程为：浏览器检查所有存储的 Cookie，如果某个 Cookie 符合所请求网站的范围，则把该 Cookie 添加到 HTTP 请求头中发送给服务器端。类似于顾客有各种不同的会员卡，去“A 超市”时要带上该超市能使用的会员卡。其中“按照一定的规则自动把 Cookie 添加到请求中发送给服务器端”与开发人员无关，浏览器具备这个功能。

Java Web 开发中，Cookie 的工作原理如图 5-2 所示。服务器端 Servlet 程序创建 Cookie 并在响应中写入 Cookie，客户端浏览器读取响应头中的 Cookie 并自动存储。在后续的请求中，浏览器自动在请求头中包含 Cookie，服务器端 Servlet 可读取 Cookie 并进行相应的个性化业务处理和响应。

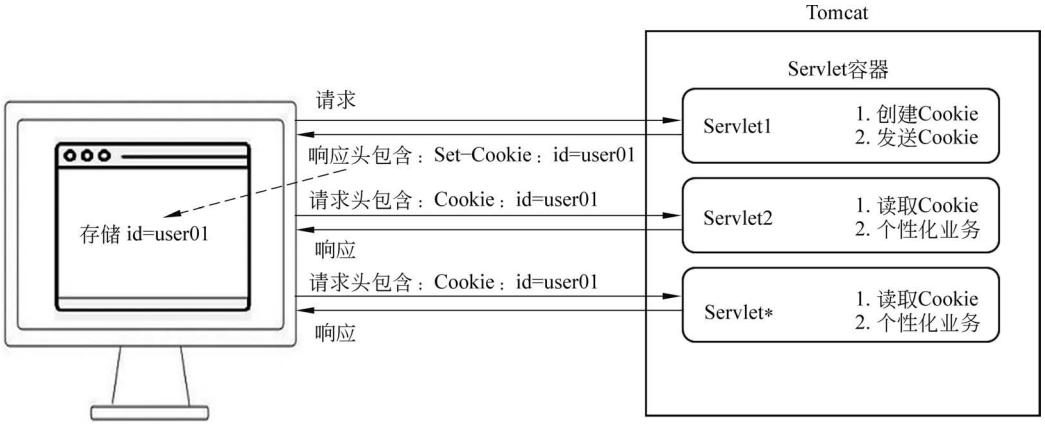

图 5-2　Cookie 的工作原理

Cookie 的内容主要包括：名字、值、到期时间、路径和域。域用于指定某一个特定域名，如 baidu.com，也可以用于指定域名下的二级域名，如 tieba.baidu.com 或者 fanyi.baidu.com。路径就是域名后面的 URL 路径，如“/”或者“/admin”等。域与路径合在一起就构成了 Cookie 的作用范围。

若不设置到期时间，则表示这个 Cookie 的生命周期为浏览器会话期间，只要关闭浏览

器，Cookie 就消失了。这种生命周期为浏览器会话期的 Cookie 被称为会话 Cookie，会话 Cookie 不存储在客户端硬盘而是保存在浏览器运行时的内存中。

若设置了到期时间，浏览器就会把 Cookie 存储在客户端硬盘上，关闭后再次打开浏览器，这些 Cookie 仍然有效，直到超过设定的到期时间。

存储在客户端硬盘上的 Cookie 可以在同一个浏览器的不同进程间共享，如同一个客户端上开启的两个 Google Chrome 浏览器应用，但是不同浏览器存储的 Cookie 不能共享。

而对于保存在浏览器运行时内存中的 Cookie，当前浏览器所有进程和标签页都可以共享 Cookie。

使用浏览器内置的开发者工具，可以查看当前正在浏览网站的 Cookie 信息。以 Google Chrome 浏览器为例，访问百度网站，单击键盘上的 F12 快捷键，在弹出的界面上选择“网络”或者“Network”选项卡，然后单击左侧的 www.baidu.com 文件，在右侧查看响应标头中的 Cookie 信息，如图 5-3 所示。

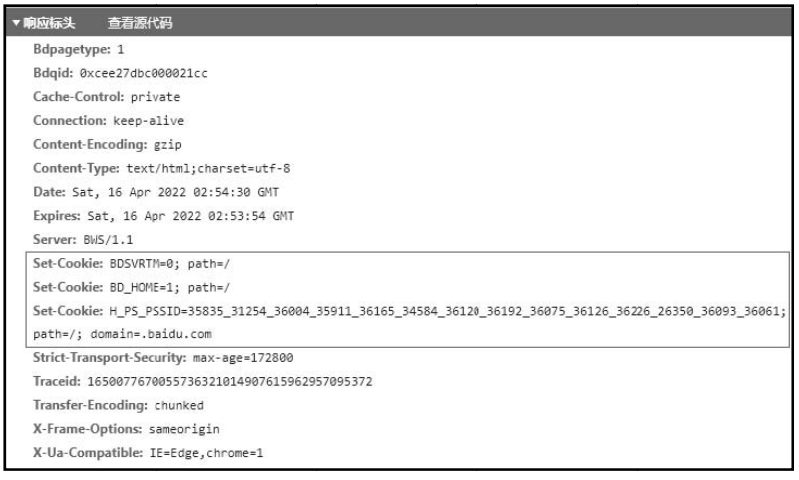

图 5-3　HTTP 通信协议响应头中的 Cookie 信息

在浏览器中通过“设置”菜单，可以查看浏览器存储在硬盘的所有网站的 Cookie 信息。以 Google Chrome 浏览器为例，单击“设置”菜单，在弹出的界面上单击“隐私设置和安全性”选项，继续选择“Cookie 及其他网站数据”→“查看所有 Cookie 和网站数据”命令，在弹出的界面上单击“baidu.com”，即可以查看百度网站相关的 Cookie 信息，如图 5-4 所示。

图 5-4　浏览器存储在硬盘的 Cookie 信息

■ 5.2.3　Cookie 的应用

在 Java Web 开发中，开发人员在服务器端进行 Cookie 的创建、设置和发送等相关代码的编写。Cookie 的常用方法如表 5-1 所示。

表 5-1　Cookie 的常用方法

方　法　名	说　　明
setDomain	设置 Cookie 适用的域名
setMaxAge	设置 Cookie 过期时间，以秒为单位
setPath	设置能够访问 Cookie 的路径
setSecure	设置浏览器使用的安全协议，如 HTTPS
setValue	设置 Cookie 对象的值

1. Cookie 的创建与发送

通常在服务器端 Servlet 中编写代码创建 Cookie，服务器端的一次响应过程中可以创建与发送多个 Cookie 对象，示例代码如下：

```
String id = request.getParameter("id");          //假设 id 参数值为 user01
String name = request.getParameter("name");      //假设 name 参数值为 name01
Cookie cookieId = new Cookie("id",id);           //以键值对的方式存放内容
Cookie cookieName = new Cookie("name",name);     //以键值对的方式存放内容
```

Cookie 对象创建后，不能增加其他的键值对，但是可以修改 Cookie 对象值的内容，示例代码如下：

```
cookieName.setValue("userNameStr");
```

Cookie 对象添加到 response 对象返回给客户端，示例代码如下：

```
response.addCookie(cookieId);
response.addCookie(cookieName);
```

上述代码执行后，在浏览器内置的开发者工具中可以查看响应头中的 Cookie 信息，如图 5-5 所示。

2. Cookie 过期时间的设置

默认情况下，Cookie 为会话 Cookie，即 Cookie 保存在浏览器运行时的内存中，浏览器关闭后，Cookie 就失效了。通过设置过期时间，告知浏览器把 Cookie 数据保存在客户端硬盘，即使关闭浏览器，Cookie 仍然有效，示例代码如下：

```
cookieName.setMaxAge(-1);       //参数为负整数，关闭浏览器，Cookie 就失效
cookieId.setMaxAge(7*24*60*60);//参数为正整数，单位为秒，Cookie 存储到客户端硬盘
                                //本语句表示 Cookie 在 7 天内有效
cookieId.setMaxAge(0);          //参数为 0，立即删除 Cookie
```

图 5-5　在浏览器中查看响应头中的 **Cookie** 信息

3. Cookie 作用范围的设置

Cookie 默认的路径为当前访问的 Servlet 的父路径，默认作用范围是当前域名的默认路径。

例如，在问卷调查网站中，创建和发送 Cookie 的是 SendCookieServlet，路径为 http://localhost:8080/question/SendCookieServlet。由此创建的 Cookie 默认路径是/question，默认作用范围是 http://localhost:8080/question。

在单独网站的开发过程中，通常使用 Cookie 默认作用范围即可。若要在 Tomcat 中部署的所有网站都能使用同一个 Cookie，需要设置该 Cookie 的路径为“/”，示例代码如下：

```
cookie.setPath("/");                    //当前 Tomcat 下所有网站都能获取该 Cookie
```

Cookie 的作用范围支持跨域。例如，在百度（www.baidu.com）网站中创建的 Cookie，需要在百度贴吧（teiba.baidu.com）、百度翻译（fanyi.baidu.com）等网站中有效，可通过设置 Cookie 的域实现，示例代码如下：

```
cookie.setDomain(".baidu.com");    //参数以“.”开始,在一级域名下实现 Cookie 共享
```

在 Java Web 项目开发中，通常包含用户登录功能模块，用户登录时会提供“记住我”选项，让登录页面记住用户上一次访问时的用户名，使得网站的用户体验更佳。这项功能可以使用 Cookie 技术实现。

实例 5-1

实例 5-1　利用 Cookie 实现记住用户名的功能

本实例为在网站登录页面 login.html 中使用 JavaScript 代码读取 Cookie 信息，实现“记住我”功能，用户在 7 天内访问该页面时，页面自动填写用户名，浏览效果如图 5-6 所示。

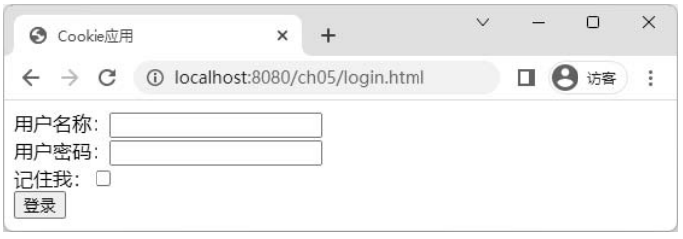

图 5-6　利用 **Cookie** 实现记住用户名的浏览效果

源程序：**index.jsp** 文件

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
    <title>Cookie 应用</title>
</head>
<body>
<h1>欢迎来到 Java Web 开发的网站</h1>
<a href="<%=request.getContextPath()%>/login.html">重新登录</a>
</body>
</html>
```

源程序：**login.html** 文件

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Cookie 应用</title>
</head>
<body>
<form action="DoLoginServlet" method="post">
  用户名称：<input type="text"  name="username" id="username" /><br>
  用户密码：<input type="password" name="password" /><br>
  记住我：<input type="checkbox" name="remember" value="ok" /><br>
  <input type="submit" value="登录"/>
</form>

<script>
  //调用函数 getCookieValueByKey 获取 Cookie 中保存的用户名称
  //注意：调用函数中参数的值，与 Servlet 中创建的 Cookie 的 key 要一致
  let cookieUserName = getCookieValueByKey("name");
  //根据 HTML 标记的 id 属性，获取用户名称文本框，并写入 Cookie 中读取的用户名
  document.getElementById("username").value = cookieUserName;

  //函数：根据 Cookie 的 key 获取对应的值
  function getCookieValueByKey(key){
    let val = "";
    let cookies = document.cookie;
    cookies = cookies.replace(/\s/,"");
    let cookieArray = cookies.split(";");
    for(let i=0; i<cookieArray.length; i++){
      let cookie = cookieArray[i];
      let kvArray = cookie.split("=");
      if(kvArray[0] == key){
        val = kvArray[1];
        break;
```

```
        }
      }
      return val;
    }
</script>
</body>
</html>
```

源程序：**DoLoginServlet.java** 文件的部分代码

```
@WebServlet("/DoLoginServlet")
public class DoLoginServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException{
        //获取表单提交的三个参数
        //注意：request.getParameter 方法参数值与表单中各个标记的 name 值应当一致
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        String remember = request.getParameter("remember");
        //当用户选中“记住我”选项提交时，remember 变量值为 ok
        if("ok".equals(remember)) {
            Cookie cookie = new Cookie("name", username); //Cookie 的 key 为 name
            cookie.setMaxAge(7 * 24 * 60 * 60);       //Cookie 有效期为 7 天
            response.addCookie(cookie);
        }
        //这里假设正确的用户名称和用户密码分别是 admin 和 pwd
        //实际项目中，通常与数据库中的用户名称和用户密码进行比较
        if ("admin".equals(username) && "pwd".equals(password)){
            //验证通过，重定向至首页
            response.sendRedirect(request.getContextPath() + "/index.jsp");
        }else{
            //验证失败，重定向至登录页面
            response.sendRedirect(request.getContextPath() + "/login.html");
        }
    }
}
```

操作步骤：

（1）在 Book 项目中创建 ch05 模块。右击 Book 项目名称，在弹出的快捷菜单中选择 New→Module 命令，在弹出的对话框中选择模块类型为 Java Enterprise，输入模块名称为 ch05、位置为 D:\ideaProj\Book\ch05，选择模块结构为 Web application，选择应用服务器为 Tomcat 9.0.29，然后单击 Next 按钮，在弹出的对话框中单击 Finish 按钮，ch05 模块建立完成。

（2）右击 ch05 模块中的 webapp 文件夹，在弹出的快捷菜单中选择 New→JSP/JSPX 命令，输入文件名 index.jsp，按 Enter 键确认建立文件。

（3）右击 ch05 模块中的 webapp 文件夹，在弹出的快捷菜单中选择 New→HTML 命

令，输入文件名 login.html，按 Enter 键确认建立文件。

（4）右击 com.example.ch05 包，在弹出的快捷菜单中选择 New→Servlet 命令，然后输入名称 DoLoginServlet，按 Enter 键确认建立文件。

（5）分别在三个新建的文件中输入源代码。

（6）在 IDEA 运行工具栏中选择配置 Tomcat，在 Deployment 选项卡内点击“+”按钮，选择 Artifact 选项，在弹出的界面中选择 ch05:war exploded，最后修改 Application context 为“/ch05”。单击 OK 按钮，完成网站部署。

（7）单击运行工具栏中的“运行”按钮 ▶，启动 Tomcat。

（8）在 login.html 代码视图中单击浏览器图标，或者在浏览器地址栏中输入网址 http://localhost:8080/ch05/login.html，查看浏览效果。

程序说明：

- 第一次登录时输入正确的用户名称 admin 和密码 pwd，并且选中“记住我”，登录成功后，关闭浏览器。然后再打开浏览器，输入网址 http://localhost:8080/ch05/login.html，查看页面是否已经自动填写用户名称 admin。
- 浏览本实例的过程中，通过浏览器内置的开发者工具，可以查看请求头中的 Cookie 信息。
- 本实例的 Cookie 有效期为 7 天，可以通过浏览器“设置”→“隐私设置和安全性”查看保存在硬盘 localhost 站点的 Cookie 详细信息。
- 操作步骤中的第（6）步，仅需要在 ch05 模块网站第一次运行时配置。

5.3 Session 技术

■ 5.3.1　Session 概述

HTTP 通信协议是一种无状态协议，即每次服务器端接收到客户端请求时，都是一个全新的请求，服务器端并不知道客户端的历史请求记录。

Session 是设计交互式动态网页的一项重要技术，它可以将客户端的用户数据存储在服务器端。通过在服务器端存储与客户端在同一个会话期间的交互信息，从而实现交互式动态网站。

Cookie 技术采用的是在客户端保持会话状态的方案，而 Session 技术采用的是在服务器端保持会话状态的方案。Session 的主要目的就是为了弥补 HTTP 通信协议的无状态特性。

■ 5.3.2　Session 的运行机制

Session 在客户端第一次访问服务器（Tomcat）时自动创建，但是只有访问 Servlet、JSP 等资源时才会创建 Session，而访问 HTML、CSS、图片等静态资源时并不会创建 Session。

当多个客户端访问服务器端时，服务器端会为每个客户端分别创建不同的 Session，这

些 Session 通过 Id 属性进行区分。服务器端依据客户端提供的 Id 来“识别”不同客户，读取相应 Session 的内容，保持与客户的会话状态。

在 Java Web 开发中，Tomcat 管理服务器端 Session，在创建 Session 时，会自动创建名称为 JSESSIONID 的 Cookie，并将 Session 的 Id 属性值赋值该 Cookie，返回给客户端；客户端浏览器会在后续请求中，自动包含该 Cookie。因此，服务器端在与客户端交互过程中，可依据 Cookie 中的 JSESSIONID 读取保存在服务器端的 Session 数据。

由图 5-7 可以看出，服务器端存储了多个客户端的 Session 数据，识别当前客户端的 Session 依据的是客户端请求中包含的 Cookie 中 JSESSIONID 的值。JSESSIONID 的值与 Session 的 Id 属性值是一一对应的。

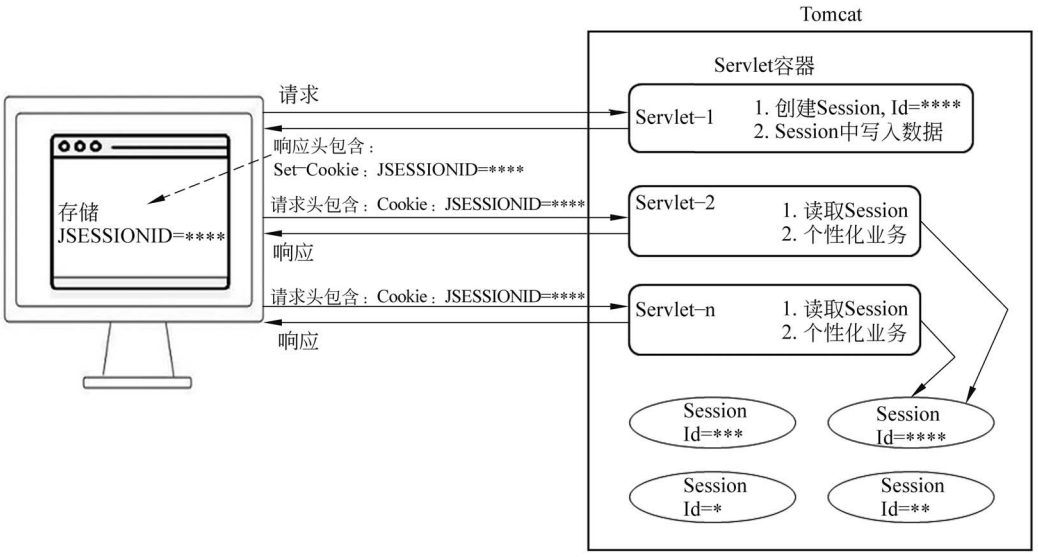

图 5-7　Session 的工作原理

由图 5-2 所示的 Cookie 的工作原理可知，普通的 Cookie 创建并添加到 response 响应中的工作，是开发人员在 Servlet 中编写代码实现的。但是 JSESSIONID 这个 Cookie 不需要开发人员编写代码实现，而是由 Tomcat 自动管理。

通过 Session 技术维护服务器端与客户端的会话状态，需要 Cookie 技术的支持，当客户端浏览器禁用 Cookie 时，Tomcat 会启用 URL 重写机制。这个机制是客户端 Cookie 被禁用时的一个兜底策略，通过在 URL 后面加上“;jsessionid=xxx”来传递 Session 的 Id，这样即使 Cookie 被禁用时，也可以保证 Session 的可用性，但是 Session 的 Id 暴露在 URL 里，本身是不安全的。

默认情况下，用户可以参考图 5-3，查看 Cookie 中的 JSESSIONID 的信息；浏览器 Cookie 被禁用时，用户可以在浏览器地址栏中查看 JSESSIONID 的信息。

■ 5.3.3　Session 的生命周期

Session 数据是保存在服务器端的，为了获得更快的存取速度，服务器端一般把 Session 存储在内存里。每个客户端都有一个独立的 Session，如果 Session 内容过于复杂，当大量