Web之状态管理

课前默写

```
1、写出HTTPServletResponse的常用方法
2、写出HTTPServletRequest的常用方法
```

课程回顾

```
1、HTTPServletResponse的使用
2、HTTPServletRequest的使用
```

今日内容

```
1、状态管理
2、Cookie的使用
3、Cookie的路径
4、Session的工作原理
5、Session的使用
6、URL重定向
```

教学目标

```
1、掌握状态管理
2、掌握Cookie的使用
3、掌握Cookie的路径
4、掌握Session的工作原理
5、掌握Session的使用
6、掌握URL重定向
```

第九章 状态管理-Cookie

9.1 状态管理概述

9.1.1 为什么需要状态管理

```
HTTP协议是无状态的，不能保存每次提交的信息，即当服务器返回与请求相对应的应答之后，这次事务的所有信息就丢掉了。
    如果用户发来一个新的请求，服务器无法知道它是否与上次的请求有联系。
    对于那些需要多次提交数据才能完成的Web操作，比如登录来说，就成问题了。
```

9.1.2 什么是状态管理

```
WEB应用中的会话是指一个客户端浏览器与WEB服务器之间连续发生的一系列请求和响应过程。
WEB应用的会话状态是指WEB服务器与浏览器在会话过程中产生的状态信息，借助会话状态，WEB服务器能够把属于同一会话中的一系列的请求和响应过程关联起来。
```

9.1.3 状态管理的两种常见模式

客户端状态管理技术：将状态保存在客户端。代表性的是Cookie技术。

服务器状态管理技术：将状态保存在服务器端。代表性的是session技术（服务器传递sessionID时需要使用Cookie的方式）。

9.2 Cookie应用

9.2.1 什么是Cookie

Cookie是在浏览器访问WEB服务器的某个资源时，由WEB服务器在HTTP响应消息头中附带传送给浏览器的一片数据，WEB服务器传送给各个客户端浏览器的数据是可以各不相同的。
一旦WEB浏览器保存了某个Cookie，那么它在以后每次访问该WEB服务器时，都应在HTTP请求头中将这个Cookie回传给WEB服务器。
WEB服务器通过在HTTP响应消息中增加Set-Cookie响应头字段将Cookie信息发送给浏览器，浏览器则通过在HTTP请求消息中增加Cookie请求头字段将Cookie回传给WEB服务器。
一个Cookie只能标识一种信息，它至少含有一个标识该信息的名称（NAME）和设置值（VALUE）。
一个WEB站点可以给一个WEB浏览器发送多个Cookie，一个WEB浏览器也可以存储多个WEB站点提供的Cookie。
浏览器一般只允许存放300个Cookie，每个站点最多存放20个Cookie，每个Cookie的大小限制为4KB

9.2.2 如何创建Cookie

```
        //创建Cookie
        Cookie ck=new Cookie("code", code);
        ck.setPath("/");//设置Cookie的路径
        ck.setMaxAge(-1);//内存存储，取值有三种：>0有效期，单位秒；=0失效；<0内存存储
        response.addCookie(ck);//让浏览器添加Cookie
```

9.2.3 如何查询Cookie

```
//获取所有的Cookie
Cookie[] cks=request.getCookies();
        //遍历Cookie
        for(Cookie ck:cks){
         //检索出自己的Cookie
            if(ck.getName().equals("code"))
            {
             //记录Cookie的值
                code=ck.getValue();
                break;
            }
        }
```

9.2.4 如何修改Cookie

只需要保证Cookie的名和路径一致即可修改

```
        //创建Cookie
        Cookie ck=new Cookie("code", code);
        ck.setPath("/");//设置Cookie的路径
        ck.setMaxAge(-1);//内存存储，取值有三种：>0有效期，单位秒；=0失效；<0内存存储
        response.addCookie(ck);//让浏览器添加Cookie
```

9.2.5 Cookie的生存时间

```
ck.setMaxAge(-1)；设置生成时间
取值说明：
>0有效期，单位秒
=0失效
<0内存存储
```

9.2.6 Cookie的编码与解码

```
中文和英文字符不同，中文属于Unicode字符，在内存中占用4个字符，而英文属于ASCII字符，内存中只占2个字节。Cookie中使用Unicode字符时需要对Unicode字符进行编码，否则会出现乱码。编码可以使用java.net.URLEncoder类的encode(String str,String encoding)方法，解码使用java.net.URLDecoder类的decode(String str,String encoding)方法
```

代码如下：

保存：Servlet类

```
// 使用中文的 Cookie. name 与 value 都使用 UTF-8 编码.
    Cookie cookie = new Cookie(
        URLEncoder.encode("姓名", "UTF-8"),
        URLEncoder.encode("老邢", "UTF-8"));
    // 发送到客户端
    response.addCookie(cookie);
```

读取：jsp页面

```
<%
    if(request.getCookies() != null){
        for(Cookie cc : request.getCookies()){
            String cookieName = URLDecoder.decode(cc.getName(), "UTF-8");
            String cookieValue = URLDecoder.decode(cc.getValue(), "UTF-8");
            out.println(cookieName + "=");
            out.println(cookieValue + "; <br/>");
        }
    }
    else{
        out.println("Cookie 已经写入客户端. 请刷新页面. ");
    }
%>
```

9.3 Cookie的路径问题

9.3.1 什么是Cookie的路径问题

cookie 一般都是由于用户访问页面而被创建的，可是并不是只有在创建 cookie 的页面才可以访问这个cookie。在默认情况下，出于安全方面的考虑，只有与创建 cookie 的页面处于同一个目录或在创建cookie页面的子目录下的网页才可以访问。那么此时如果希望其父级或者整个网页都能够使用cookie，就需要进行路径的设置

9.3.2 发送Cookie的条件

浏览器在发送请求之前，首先会根据请求url中的域名在cookie列表中找所有与当前域名一样的cookie，然后再根据指定的路径进行匹配，如果当前请求在域匹配的基础上还与路径匹配那么就会将所有匹配的cookie发送给服务器，这里要注意的是最大匹配和最小匹配问题，有些cookie服务器在发送之前会有意扩大当前页面cookie的匹配范围，此时这些被扩大范围的cookie也会一起发送给服务器

9.3.3 如何设置Cookie的路径

通过Cookie的setPath方法设置路径

9.4 Cookie的特点

优点：

可配置到期规则 Cookie 可以在浏览器会话结束时到期，或者可以在客户端计算机上无限期存在，这取决于客户端的到期规则。
不需要任何服务器资源 Cookie 存储在客户端并在发送后由服务器读取。
简单性 Cookie 是一种基于文本的轻量结构，包含简单的键值对。
数据持久性 虽然客户端计算机上 Cookie 的持续时间取决于客户端上的 Cookie 过期处理和用户干预，Cookie 通常是客户端上持续时间最长的数据保留形式

缺点：

大小受到限制 大多数浏览器对 Cookie 的大小有 4096 字节的限制，尽管在当今新的浏览器和客户端设备版本中，支持 8192 字节的 Cookie 大小已愈发常见。
用户配置为禁用 有些用户禁用了浏览器或客户端设备接收 Cookie 的能力，因此限制了这一功能。
潜在的安全风险 Cookie 可能会被篡改。用户可能会操纵其计算机上的 Cookie，这意味着会对安全性造成潜在风险或者导致依赖于 Cookie 的应用程序失败。
另外，虽然 Cookie 只能被将它们发送到客户端的域访问，历史上黑客已经发现从用户计算机上的其他域访问 Cookie 的方法。您可以手动加密和解密 Cookie，但这需要额外的编码，并且因为加密和解密需要耗费一定的时间而影响应用程序的性能。

第十章 状态管理-Session

10.1 Session概述

10.1.1 什么是Session

Session用于跟踪客户的状态。
Session指的是在一段时间内，单个客户与Web服务器的一连串相关的交互过程。
在一个Session中，客户可能会多次请求访问同一个网页，也有可能请求访问各种不同的服务器资源。

10.1.2 Session工作原理

session被用于表示一个持续的连接状态，在网站访问中一般指代客户端浏览器的进程从开启到结束的过程。session其实就是网站分析的访问（visits）度量，表示一个访问的过程。
session的常见实现形式是会话cookie（session cookie），即未设置过期时间的cookie，这个cookie的默认生命周期为浏览器会话期间，只要关闭浏览器窗口，cookie就消失了。实现机制是当用户发起一个请求的时候，服务器会检查该请求中是否包含sessionid，如果未包含，则系统会创造一个名为JSESSIONID的输出 cookie返回给浏览器(只放入内存，并不存在硬盘中)，并将其以HashTable的形式写到服务器的内存里面；当已经包含sessionid是，服务端会检查找到与该session相匹配的信息，如果存在则直接使用该sessionid，若不存在则重新生成新的 session。这里需要注意的是session始终是有服务端创建的，并非浏览器自己生成的。 但是浏览器的cookie被禁止后session就需要用get方法的URL重写的机制或使用POST方法提交隐藏表单的形式来实现

10.1.3 如何获得Session

```
//获取Session对象
        HttpSession session=request.getSession();
        System.out.println("Id: "+session.getId());//唯一标记,
        System.out.println("getLastAccessedTime:"+session.getLastAccessedTime());//最后一次访问时间，毫秒
        System.out.println("getMaxInactiveInterval:"+session.getMaxInactiveInterval());//获取最大的空闲时间，单位秒
        System.out.println("getCreationTime:"+session.getCreationTime());//获取Session的创建，单位毫秒
```

10.1.4 如何使用Session绑定对象

使用HttpSession的setAttrobute(属性名,Object)方法

10.1.5 如何删除Session

使用HttpSession的invalidate方法

10.2 Session超时

10.2.1 什么是Session超时

HttpSession的最后一程访问时间和当前时间的差距大于了指定的最大空闲时间，这时服务器就会销毁Session对象。默认的空闲时间为30分钟。

10.2.2 如何修改Session的缺省时间限制

使用HttpSession的setMaxAge设置，单位秒

10.2.3 Session失效的几种情况

1、超过了设置的超时时间
2、主动调用了invalidate方法
3、服务器主动或异常关闭
注意：浏览器关闭并不会让Session失效

10.3 浏览器禁用Cookie的解决方案

10.3.1 浏览器禁用Cookie的后果

如果浏览器禁用Cookie，session还能用吗?
答：不能，但又其他的解决方案
服务器在默认情况下，会使用Cookie的方式将sessionID发送给浏览器，如果用户禁止Cookie，则sessionID不会被浏览器保存，此时，服务器可以使用如URL重写这样的方式来发送sessionID.

使用Session区分每个用户的方式：
1、作为Cookie
2、作为隐藏域嵌入HTML表单中，附加在主体的URL中，通常作为指向其他应用程序页面的链接，即URL重写。
3、更加高效的方法：会话跟踪（Session）

10.3.2 什么是URL重写

浏览器在访问服务器上的某个地址时，不再使用原来的那个地址，而是使用经过改写的地址（即，在原来的地址后面加上了sessionID）

10.3.3 如何实现URL重写

如果是链接地址和表单提交，使用
response.encodeURL(String url)生成重写后的URL
如果是重定向，使用
response.encodeRedirectURL(String url)生成重写的URL

10.4 Session的典型应用

10.4.1 验证登录

登录的jsp页面：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>登录页面</title>
<style type="text/css">
.dv_error{
color:red;
}
</style>
</head>
<body>
<div class="dv_error"><h1>${msg }</h1></div>
<form action="login" method="post">
<label>邮箱：</label><input name="email"><br/>
<label>密码：</label><input type="password" name="psw"><br/>
<input type="submit" value="登录">
</form>

</body>
```

```
</html>
```

登录的成功页面：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" import="java.util.*"%>
<!DOCTYPE html >
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>登录成功</title>
</head>
<body>
<div>
<h1>欢迎${username }</h1>
<a href="exit">注销</a><br/>
<a href="login.jsp">登录</a><br/>
</div>
</body>
</html>
```

登录的Servlet:

```
package com.qf.web.servlet;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

/**
 * Servlet implementation class LoginServlet
 */
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public LoginServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
```

```
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        response.getWriter().append("Served at: ").append(request.getContextPath());
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        //doGet(request, response);
        request.setCharacterEncoding("utf-8");
        String username=request.getParameter("email");
        String pass=request.getParameter("psw");
        System.out.println(request.getRemoteAddr()+": 登录: 账号: "+username+"----->密
码: "+pass);
        //将数据存储在Session中
        HttpSession session=request.getSession();
        System.out.println("Id: "+session.getId());//唯一标记,
        System.out.println("getLastAccessedTime:"+session.getLastAccessedTime());//最后一次访问
时间, 毫秒
        System.out.println("getMaxInactiveInterval:"+session.getMaxInactiveInterval());//获取最
大的空闲时间, 单位秒
        System.out.println("getCreationTime:"+session.getCreationTime());//获取Session的创建, 单
位毫秒
        //设置数据
        session.setAttribute("username", username);
        Random rm=new Random();
        if(rm.nextInt(2)==0){
            //重定向
            response.sendRedirect(response.encodeURL("main.jsp"));
        }else{
            request.setAttribute("msg", "运气欠佳, 重新再来");
            //转发
            request.getRequestDispatcher("login.jsp").forward(request, response);
        }
    }
}
```

注销登录的Servlet:

```
/**
 * Servlet implementation class ExitLogin
 */
@WebServlet("/exit")
public class ExitLogin extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
```

```
     * @see HttpServlet#HttpServlet()
     */
    public ExitLogin() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
      //移除登录信息
        request.getSession().removeAttribute("username");
        //request.getSession().invalidate();//将当前的Sessio失效
        response.sendRedirect("main.jsp");
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        //doGet(request, response);
    }
```

10.4.2 验证码的实现

生成验证码：

```
/**
 * Servlet implementation class CreateCode
 * 验证码的生成
 */
@WebServlet("/createcode")
public class CreateCode extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public CreateCode() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */


    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
```

```
ServletException, IOException {
        // TODO Auto-generated method stub
        //response.getWriter().append("Served at: ").append(request.getContextPath());
        ValidateCode vc=new ValidateCode(200, 30, 4, 10);
        String code=vc.getCode();
        System.out.println(request.getRemoteAddr()+":生成:"+code);
        //使用Session存储生成的验证码
        HttpSession session=request.getSession();
       session.setAttribute("code",code);
        vc.write(response.getOutputStream());
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }

}
```

jsp页面输入验证码：

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html >
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>验证码的校验</title>
<script type="text/javascript">
function changeCode(obj) {
    obj.src="createcode?r="+Math.random();
}

</script>
</head>
<body>
<h1>${msg }</h1>
<form action="validatecode">
<label>验证码：</label><input name="nc"/><img alt="验证码" onclick="changeCode(this)"
src="createcode" width="200" height="30">
<br/>
<input type="submit" value="验证输入的验证码">
</form>
</body>
</html>
```

验证验证码的Servlet

```java
/**
 * Servlet implementation class ValidateCode
 * 验证用户输入的验证码
 */
@WebServlet("/validatecode")
public class ValidateCode extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ValidateCode() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        //response.getWriter().append("Served at: ").append(request.getContextPath());


        String c=request.getParameter("nc");
        String code=(String)request.getSession().getAttribute("code");

        //比较验证码
        if(code.equalsIgnoreCase(c)){
            request.setAttribute("msg", "验证码正确");
        }else{
            request.setAttribute("msg", "验证码输入失败");
        }
        request.getRequestDispatcher("code.jsp").forward(request, response);
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

作业题

1、描述Cookie和Session的区别
2、使用Cookie实现浏览记录
3、使用Session实现登录

面试题

1、session产生的session_id放在cookie里面，如果用户把cookie禁止掉，是不是session也不能用了呢?
2、为什么说session 比cookie更安全?
3、cookie和session原理?