day11

JSP 入门

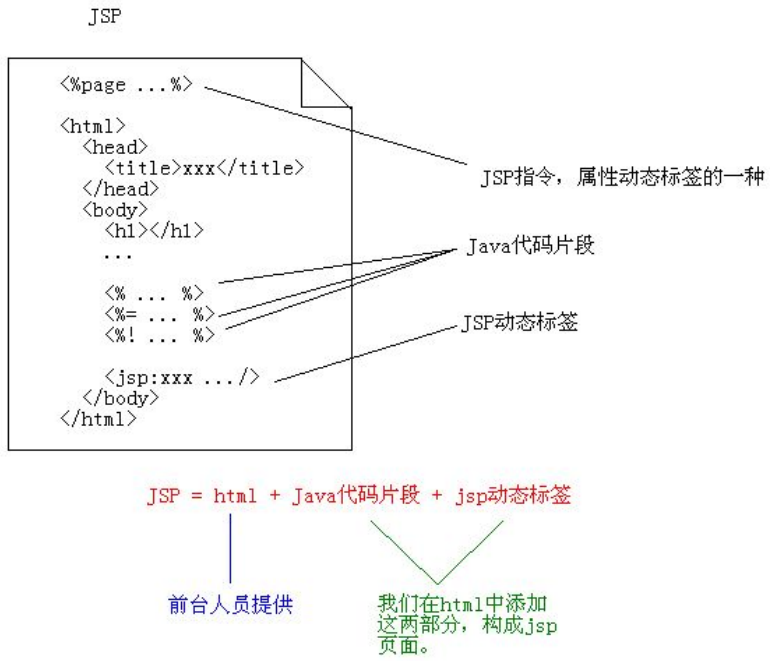
1 JSP 概述

1.1 什么是 JSP

JSP（Java Server Pages）是 JavaWeb 服务器端的**动态资源**。它与 html 页面的作用是相同的，**显示数据和获取数据**。

1.2 JSP 的组成

JSP = html + Java 脚本（代码片段） + JSP 动态标签



2 JSP 语法

2.1 JSP 脚本

JSP 脚本就是 Java 代码片段，它分为三种：

- <%...%>：Java 语句；
- <%=...%>：Java 表达式；
- <%!...%>：Java 定义类成员；

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>JSP演示</title>
  </head>

  <body>
    <h1>JSP演示</h1>
    <%
    // Java语句
    String s1 = "hello jsp";
    // 不会输出到客户端，而是在服务器端的控制台打印
    System.out.println(s1);
    %>
  <!-- 输出到客户端浏览器上 -->
    输出变量：<%=s1 %><br/>
    输出int类型常量：<%=100 %><br/>
    输出String类型常量：<%="你好" %><br/>
    <br/>
    使用表达式输出常量是很傻的一件事，因为可以直接使用html即可，下面是输出上面的常量：<br/>
    100<br/>
    你好
  </body>
</html>
```

2.2 内置对象 out

out 对象在 JSP 页面中无需创建就可以使用，它的作用是用来向客户端输出。

```
  <body>
    <h1>out.jsp</h1>
    <%
        //向客户端输出
        out.print("你好！");
    %>
  </body>
```

其中<%=...%>与 out.print()功能是相同的！它们都是向客户端输出，例如：

<%=s1%>等同于<% out.print(s1); %>

<%=”hello”%>等同于<% out.print(“hello”); %>，也等同于直接在页面中写 hello 一样。

2.3 多个<%...%>可以通用

在一个 JSP 中多个<%...%>是相通的。例如：

```
<body>
  <h1>out.jsp</h1>
  <%
      String s = "hello";
  %>
  <%
      out.print(s);
  %>
</body>
```

循环打印表格：

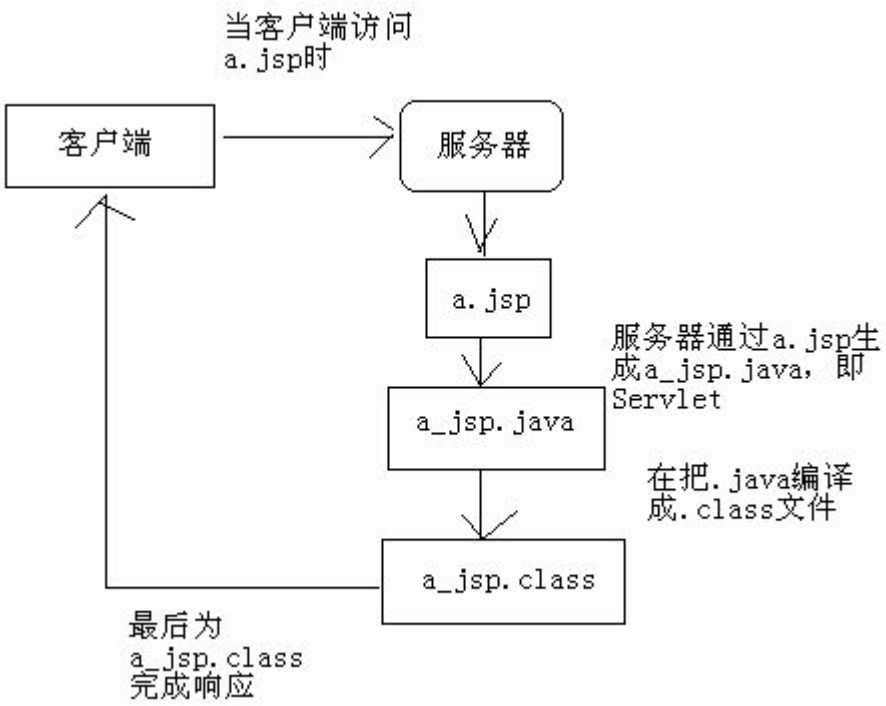
```
<body>
  <h1>表格</h1>

  <table border="1" width="50%">
      <tr>
          <th>序号</th>
          <th>用户名</th>
          <th>密码</th>
      </tr>
  <%
      for(int i = 0; i < 10; i++) {
  %>
      <tr>
          <td><%=i+1 %></td>
          <td>user<%=i %></td>
          <td><%=100 + 1 %></td>
      </tr>
  <%
      }
  %>
  </table>
</body>
```

3 JSP 的原理

3.1 JSP 是特殊的 Servlet

JSP 是一种特殊的 Servlet，当 JSP 页面首次被访问时，容器（Tomcat）会先把 JSP 编译成 Servlet，然后再去执行 Servlet。所以 JSP 其实就是一个 Servlet！



3.2 JSP 真身存放目录

JSP 生成的 Servlet 存放在${CATALANA}/work 目录下，我经常开玩笑的说，它是 JSP 的“真身”。我们打开看看其中的内容，了解一下 JSP 的“真身”。

你会发现，在 JSP 中的静态信息（例如<html>等）在“真身”中都是使用 out.write()完成打印！这些静态信息都是作为字符串输出给了客户端。

JSP 的整篇内容都会放到名为_jspService 的方法中！你可能会说<@page>不在“真身”中，<%@page>我们明天再讲。

a_jsp.java 的_jspService()方法：

```
public void _jspService(final javax.servlet.http.HttpServletRequest request,
                        final javax.servlet.http.HttpServletResponse response)
      throws java.io.IOException, javax.servlet.ServletException {

    final javax.servlet.jsp.PageContext pageContext;
    javax.servlet.http.HttpSession session = null;
    final javax.servlet.ServletContext application;
    final javax.servlet.ServletConfig config;
    javax.servlet.jsp.JspWriter out = null;
```

```
    final java.lang.Object page = this;
    javax.servlet.jsp.JspWriter _jspx_out = null;
    javax.servlet.jsp.PageContext _jspx_page_context = null;


    try {
      response.setContentType("text/html;charset=UTF-8");
      pageContext = _jspxFactory.getPageContext(this, request, response,
                  null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

    ...
}
```

4 再论 JSP 脚本

JSP 脚本一共三种形式：

- <%...%>：内容会直接放到“真身”中；
- <%=...%>：内容会放到 out.print()中，作为 out.print()的参数；
- <%!...%>：内容会放到_jspService()方法之外，被类直接包含；

前面已经讲解了<%...%>和<%=...%>，但还没有讲解<%!...%>的作用！

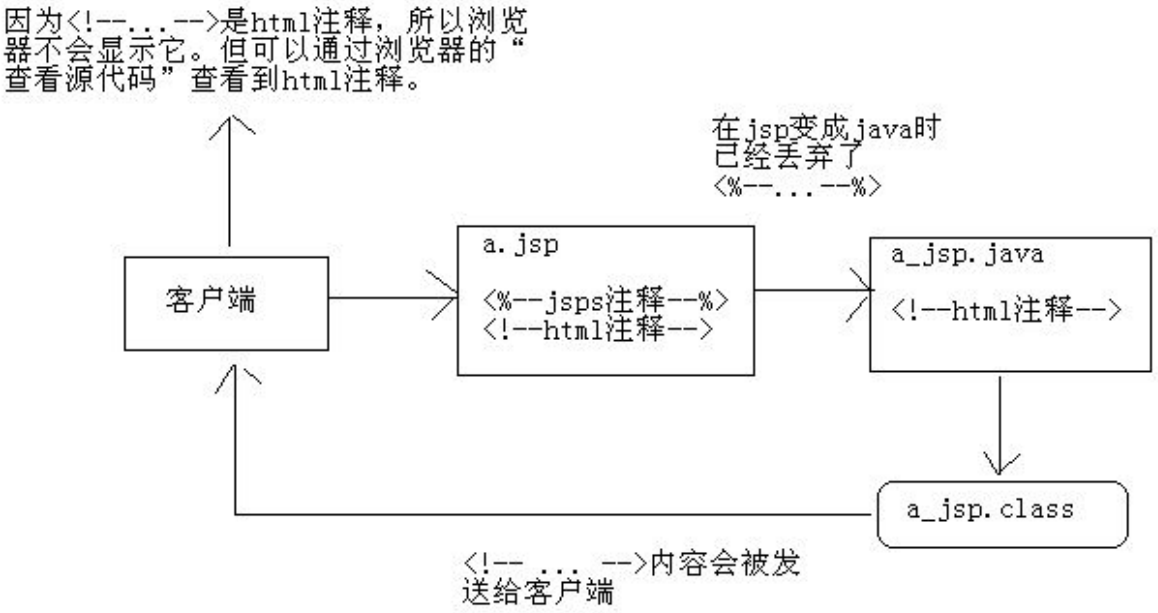
现在我们已经知道了，JSP 其实就是一个类，一个 Servlet 类。<%!...%>的作用是在类中添加方法或成员的，所以<%!...%>中的内容不会出现在_jspService()中。

```
<%!
    private String name;
    public String hello() {
        return "hello JSP!";
    }
%>
```

5 JSP 注释

我们现在已经知道 JSP 是需要先编译成.java，再编译成.class 的。其中<%-- ... --%>中的内容在 JSP 编译成.java 时会被忽略的，即 JSP 注释。

也可以在 JSP 页面中使用 html 注释：<!-- ... -->，但这个注释在 JSP 编译成的.java 中是存在的，它不会被忽略，而且会被发送到客户端浏览器。但是在浏览器显示服务器发送过来的 html 时，因为<!-- ... -->是 html 的注释，所以浏览器是不会显示它的。



会话跟踪技术

1 什么是会话跟踪技术

我们需要先了解一下什么是会话！可以把会话理解为客户端与服务器之间的一次会晤，在一次会晤中可能会包含多次请求和响应。例如你给 10086 打个电话，你就是客户端，而 10086 服务人员就是服务器了。从双方接通电话那一刻起，会话就开始了，到某一方挂断电话表示会话结束。在通话过程中，你会向 10086 发出多个请求，那么这多个请求都在一个会话中。

在 JavaWeb 中，客户向某一服务器发出第一个请求开始，会话就开始了，直到客户关闭了浏览器会话结束。

在一个会话的多个请求中共享数据，这就是会话跟踪技术。例如在一个会话中的请求如下：

- 请求银行主页；
- 请求登录（请求参数是用户名和密码）；
- 请求转账（请求参数与转账相关的数据）；
- 请求信誉卡还款（请求参数与还款相关的数据）。

在这上会话中当前用户信息必须在这个会话中共享的，因为登录的是张三，那么在转账和还款时一定是相对张三的转账和还款！这就说明我们必须在一个会话过程中有共享数据的能力。

2 会话路径技术使用 Cookie 或 session 完成

我们知道 HTTP 协议是无状态协议，也就是说每个请求都是独立的!无法记录前一次请求的状态。但 HTTP 协议中可以使用 Cookie 来完成会话跟踪!

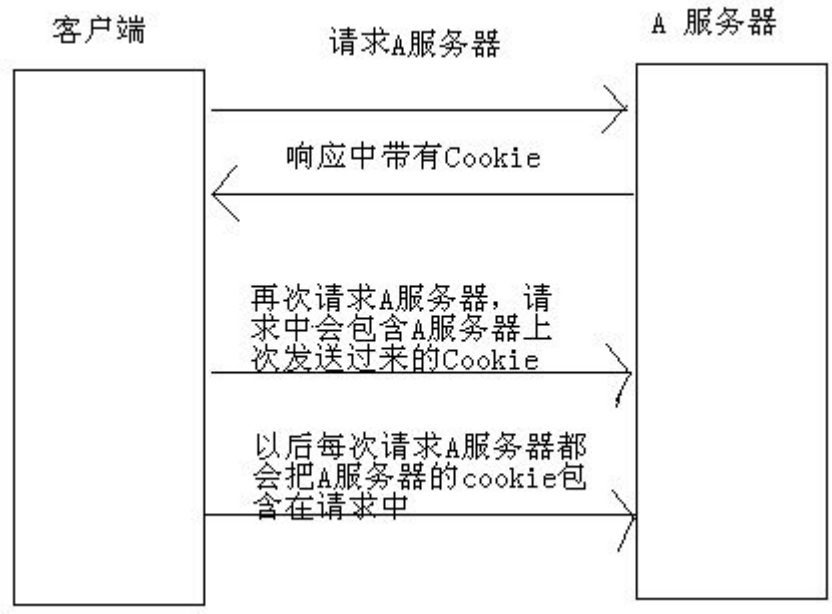
在 JavaWeb 中，使用 session 来完成会话跟踪，session 底层依赖 Cookie 技术。

Cookie

1 Cookie 概述

1.1 什么叫 Cookie

Cookie 翻译成中文是小甜点，小饼干的意思。在 HTTP 中它表示服务器送给客户端浏览器的小甜点。其实 Cookie 就是一个键和一个值构成的，随着服务器端的响应发送给客户端浏览器。然后客户端浏览器会把 Cookie 保存起来，当下一次再访问服务器时把 Cookie 再发送给服务器。



Cookie 是由服务器创建，然后通过响应发送给客户端的一个键值对。客户端会保存 Cookie，并会标注出 Cookie 的来源（哪个服务器的 Cookie）。当客户端向服务器发出请求时会把所有这个服务器 Cookie 包含在请求中发送给服务器，这样服务器就可以识别客户端了!

1.2 Cookie 规范

- Cookie 大小上限为 4KB；
- 一个服务器最多在客户端浏览器上保存 20 个 Cookie；
- 一个浏览器最多保存 300 个 Cookie；

上面的数据只是 HTTP 的 Cookie 规范，但在浏览器大战的今天，一些浏览器为了打败对手，为了展现自己的能力起见，可能对 Cookie 规范“扩展”了一些，例如每个 Cookie 的大小为 8KB，最多可保存 500 个 Cookie 等！但也不会出现把你硬盘占满的可能！

注意，不同浏览器之间是不共享 Cookie 的。也就是说在你使用 IE 访问服务器时，服务器会把 Cookie 发给 IE，然后由 IE 保存起来，当你在使用 FireFox 访问服务器时，不可能把 IE 保存的 Cookie 发送给服务器。

1.3 Cookie 与 HTTP 头

Cookie 是通过 HTTP 请求和响应头在客户端和服务器端传递的：

- Cookie：请求头，客户端发送给服务器端；
  - 格式：Cookie: a=A; b=B; c=C。即多个 Cookie 用分号离开；
- Set-Cookie：响应头，服务器端发送给客户端；
  - 一个 Cookie 对象一个 Set-Cookie：

    Set-Cookie: a=A

    Set-Cookie: b=B

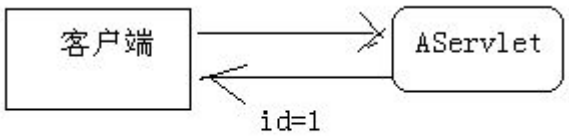
    Set-Cookie: c=C

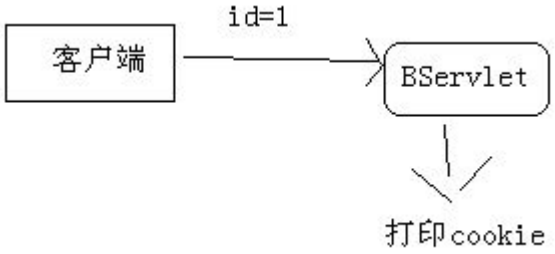
1.4 Cookie 的覆盖

如果服务器端发送重复的 Cookie 那么会覆盖原有的 Cookie，例如客户端的第一个请求服务器端发送的 Cookie 是：Set-Cookie: a=A；第二请求服务器端发送的是：Set-Cookie: a=AA，那么客户端只留下一个 Cookie，即：a=AA。

1.5 Cookie 第一例

我们这个案例是，客户端访问 AServlet，AServlet 在响应中添加 Cookie，浏览器会自动保存 Cookie。然后客户端访问 BServlet，这时浏览器会自动在请求中带上 Cookie，BServlet 获取请求中的 Cookie 打印出来。





AServlet.java

```
package cn.itcast.servlet;

import java.io.IOException;
import java.util.UUID;

import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * 给客户端发送Cookie
 * @author Administrator
 *
 */
public class AServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");

        String id = UUID.randomUUID().toString();//生成一个随机字符串
        Cookie cookie = new Cookie("id", id);//创建Cookie对象，指定名字和值
        response.addCookie(cookie);//在响应中添加Cookie对象
        response.getWriter().print("已经给你发送了ID");
    }
}
```

BServlet.java

```
package cn.itcast.servlet;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * 获取客户端请求中的Cookie
 * @author Administrator
 *
 */
public class BServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");

        Cookie[] cs = request.getCookies();//获取请求中的Cookie
        if(cs != null) {//如果请求中存在Cookie
            for(Cookie c : cs) {//遍历所有Cookie
                if(c.getName().equals("id")) {//获取Cookie名字，如果Cookie名字是
id
                    response.getWriter().print("您的ID是: " + c.getValue());//打
印Cookie值
                }
            }
        }
    }
}
```

2 Cookie 的生命

2.1 什么是 Cookie 的生命

Cookie 不只是有 name 和 value，Cookie 还是生命。所谓生命就是 Cookie 在客户端的有效时间，

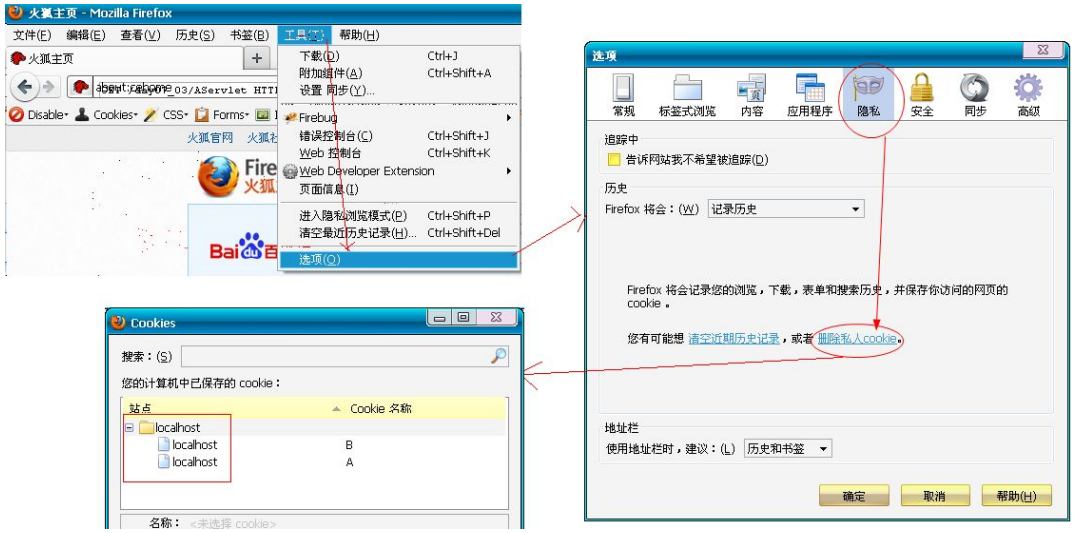
可以通过 setMaxAge(int)来设置 Cookie 的有效时间。

- cookie.setMaxAge(-1)：cookie 的 maxAge 属性的默认值就是-1，表示只在浏览器内存中存活。一旦关闭浏览器窗口，那么 cookie 就会消失。
- cookie.setMaxAge(60*60)：表示 cookie 对象可存活 1 小时。当生命大于 0 时，浏览器会把 Cookie 保存到硬盘上，就算关闭浏览器，就算重启客户端电脑，cookie 也会存活 1 小时；
- cookie.setMaxAge(0)：cookie 生命等于 0 是一个特殊的值，它表示 cookie 被作废！也就是说，如果原来浏览器已经保存了这个 Cookie，那么可以通过 Cookie 的 setMaxAge(0)来删除这个 Cookie。无论是在浏览器内存中，还是在客户端硬盘上都会删除这个 Cookie。

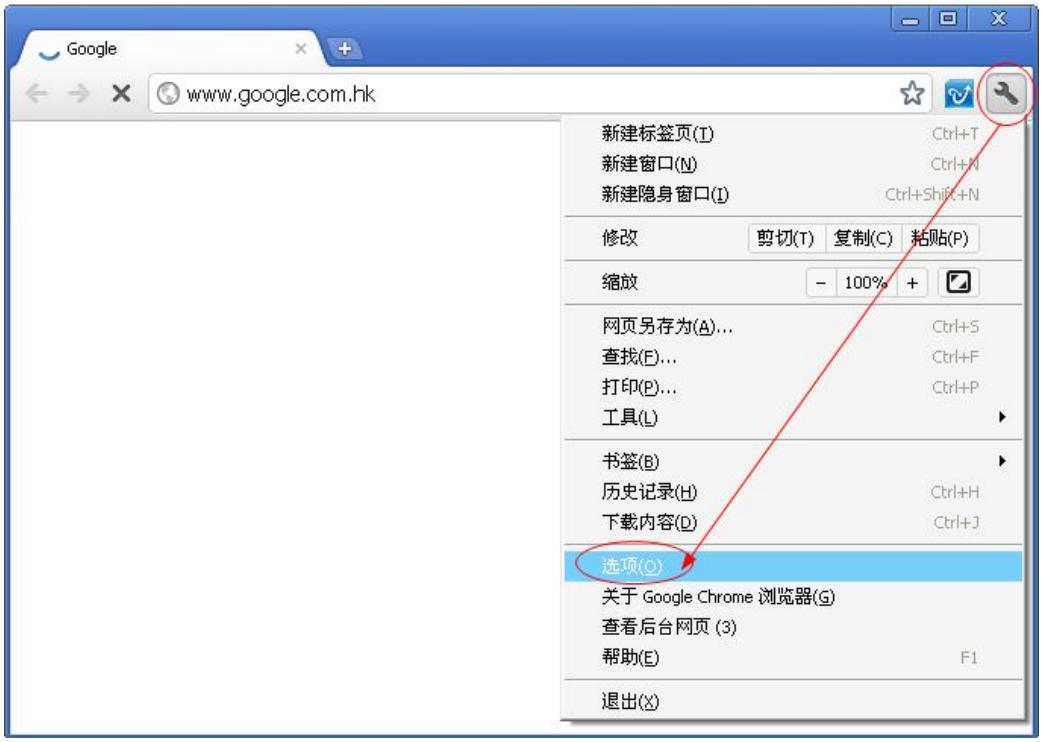
2.2 浏览器查看 Cookie

下面是浏览器查看 Cookie 的方式：

- IE 查看 Cookie 文件的路径：C:\Documents and Settings\Administrator\Cookies；
- FireFox 查看 Cooke：



- Google 查看 Cookie：


Google
www.google.com.hk
新建标签页(T) Ctrl+T
新建窗口(N) Ctrl+N
新建隐身窗口(I) Ctrl+Shift+N
修改 剪切(T) 复制(C) 粘贴(P)
缩放 - 100% +
网页另存为(A)... Ctrl+S
查找(F)... Ctrl+F
打印(P)... Ctrl+P
工具(L)
书签(B)
历史记录(H) Ctrl+H
下载内容(D) Ctrl+J
选项(O)
关于 Google Chrome 浏览器(G)
查看后台网页 (3)
帮助(E) F1
退出(X)


Google
选项 - 高级选项
chrome://settings/advanced
选项
搜索选项
基本设置
个人资料
高级选项
高级选项
隐私设置
内容设置...
清除浏览数据...
Google Chrome 浏览器可能会使用网络服务改善您的浏览
您可以视情况停用这些服务 了解详情
使用网络服务帮助解决导航错误
借助联想式建议服务，帮您在地址栏中自动填充未输完
预测网络操作，以提高网页载入速度
启用针对网上诱骗和恶意软件的防护功能
将使用情况统计信息和崩溃报告自动发送给 Google
网页内容
字体大小：
中
网页缩放：
100%

2.3 案例：显示上次访问时间

- 创建 Cookie，名为 lasttime，值为当前时间，添加到 response 中；
- 在 AServlet 中获取请求中名为 lasttime 的 Cookie；
- 如果不存在输出“您是第一次访问本站”，如果存在输出“您上一次访问本站的时间是 xxx”；

AServlet.java

```
    public void doGet(HttpServletRequest request, HttpServletResponse
response)
```

```
        throws ServletException, IOException {
    response.setContentType("text/html;charset=utf-8");

    Cookie cookie = new Cookie("lasttime", new Date().toString());
    cookie.setMaxAge(60 * 60);
    response.addCookie(cookie);

    Cookie[] cs = request.getCookies();
    String s = "您是首次访问本站！";
    if(cs != null) {
        for(Cookie c : cs) {
            if(c.getName().equals("lasttime")) {
                s = "您上次的访问时间是：" + c.getValue();
            }
        }
    }

    response.getWriter().print(s);
}
```

3 Cookie 的 path

3.1 什么是 Cookie 的路径

现在有 WEB 应用 A，向客户端发送了 10 个 Cookie，这就说明客户端无论访问应用 A 的哪个 Servlet 都会把这 10 个 Cookie 包含在请求中！但是也许只有 AServlet 需要读取请求中的 Cookie，而其他 Servlet 根本就不会获取请求中的 Cookie。这说明客户端浏览器有时发送这些 Cookie 是多余的！

可以通过设置 Cookie 的 path 来指定浏览器，在访问什么样的路径时，包含什么样的 Cookie。

3.2 Cookie 路径与请求路径的关系

下面我们来看看 Cookie 路径的作用：

下面是客户端浏览器保存的 3 个 Cookie 的路径：

a: /cookietest；

b: /cookietest/servlet；

c: /cookietest/jsp；

下面是浏览器请求的 URL：

A: http://localhost:8080/cookietest/AServlet；

B: http://localhost:8080/cookietest/servlet/BServlet；

C: http://localhost:8080/cookietest/servlet/CServlet；

- 请求 A 时，会在请求中包含 a；

- 请求 B 时，会在请求中包含 a、b；
- 请求 C 时，会在请求中包含 a、c；

也就是说，请求路径如果包含了 Cookie 路径，那么会在请求中包含这个 Cookie，否则不会请求中不会包含这个 Cookie。

- A 请求的 URL 包含了“/cookietest”，所以会在请求中包含路径为“/cookietest”的 Cookie；
- B 请求的 URL 包含了“/cookietest”，以及“/cookietest/servlet”，所以请求中包含路径为“/cookietest”和“/cookietest/servlet”两个 Cookie；
- B 请求的 URL 包含了“/cookietest”，以及“/cookietest/jsp”，所以请求中包含路径为“/cookietest”和“/cookietest/jsp”两个 Cookie；

3.3 设置 Cookie 的路径

设置 Cookie 的路径需要使用 setPath()方法，例如：
cookie.setPath(“/cookietest/servlet”);

如果没有设置 Cookie 的路径，那么 Cookie 路径的默认值当前访问资源所在路径，例如：

- 访问 http://localhost:8080/cookietest/AServlet 时添加的 Cookie 默认路径为/cookietest；
- 访问 http://localhost:8080/cookietest/servlet/BServlet 时添加的 Cookie 默认路径为/cookietest/servlet；
- 访问 http://localhost:8080/cookietest/jsp/BServlet 时添加的 Cookie 默认路径为/cookietest/jsp；

4 Cookie 的 domain

Cookie 的 domain 属性可以让网站中二级域共享 Cookie，次要！

百度你是了解的对吧！
http://www.baidu.com
http://zhidao.baidu.com
http://news.baidu.com
http://tieba.baidu.com

现在我希望在这些主机之间共享 Cookie（例如在 www.baidu.com 中响应的 cookie，可以在 news.baidu.com 请求中包含）。很明显，现在不是路径的问题了，而是主机的问题，即域名的问题。处理这一问题其实很简单，只需要下面两步：

- 设置 Cookie 的 path 为“/”：c.setPath(“/”);
- 设置 Cookie 的 domain 为“.baidu.com”：c.setDomain(“.baidu.com”)。

当 domain 为“.baidu.com”时，无论前缀是什么，都会共享 Cookie 的。但是现在我们需要设置两个虚拟主机：www.baidu.com 和 news.baidu.com。

第一步：设置 windows 的 DNS 路径解析
找到 C:\WINDOWS\system32\drivers\etc\hosts 文件，添加如下内容

```
127.0.0.1       localhost
```

```
127.0.0.1        www.baidu.com
127.0.0.1        news.baidu.com
```

第二步：设置 Tomcat 虚拟主机

找到 server.xml 文件，添加<Host>元素，内容如下：

```
<Host name="www.baidu.com"    appBase="F:\webapps\www"
        unpackWARs="true" autoDeploy="true"
        xmlValidation="false" xmlNamespaceAware="false"/>
<Host name="news.baidu.com"    appBase="F:\webapps\news"
        unpackWARs="true" autoDeploy="true"
        xmlValidation="false" xmlNamespaceAware="false"/>
```

第三步：创建 A 项目，创建 AServlet，设置 Cookie。

```
Cookie c = new Cookie("id", "baidu");
c.setPath("/");
c.setDomain(".baidu.com");
c.setMaxAge(60*60);
response.addCookie(c);
response.getWriter().print("OK");
```

把 A 项目的 WebRoot 目录复制到 F:\webapps\www 目录下，并把 WebRoot 目录的名字修改为 ROOT。

第四步：创建 B 项目，创建 BServlet，获取 Cookie，并打印出来。

```
Cookie[] cs = request.getCookies();
if(cs != null) {
    for(Cookie c : cs) {
        String s = c.getName() + ": " + c.getValue() + "<br/>";
        response.getWriter().print(s);
    }
}
```

把 B 项目的 WebRoot 目录复制到 F:\webapps\news 目录下，并把 WebRoot 目录的名字修改为 ROOT。

第五步：访问 www.baidu.com\AServlet，然后再访问 news.baidu.com\BServlet。

5 Cookie 中保存中文

Cookie 的 name 和 value 都不能使用中文，如果希望在 Cookie 中使用中文，那么需要先对中文进行 URL 编码，然后把编码后的字符串放到 Cookie 中。

向客户端响应中添加 Cookie

```
String name = URLEncoder.encode("姓名", "UTF-8");
```

```
        String value = URLEncoder.encode("张三", "UTF-8");
        Cookie c = new Cookie(name, value);
        c.setMaxAge(3600);
        response.addCookie(c);
```

从客户端请求中获取 Cookie

```
        response.setContentType("text/html;charset=utf-8");
        Cookie[] cs = request.getCookies();
        if(cs != null) {
            for(Cookie c : cs) {
                String name = URLDecoder.decode(c.getName(), "UTF-8");
                String value = URLDecoder.decode(c.getValue(), "UTF-8");
                String s = name + ": " + value + "<br/>";
                response.getWriter().print(s);
            }
        }
```

6 显示曾经浏览过的商品

index.jsp

```
<body>
  <h1>商品列表</h1>
  <a href="/day06_3/GoodServlet?name=ThinkPad">ThinkPad</a><br/>
  <a href="/day06_3/GoodServlet?name=Lenovo">Lenovo</a><br/>
  <a href="/day06_3/GoodServlet?name=Apple">Apple</a><br/>
  <a href="/day06_3/GoodServlet?name=HP">HP</a><br/>
  <a href="/day06_3/GoodServlet?name=SONY">SONY</a><br/>
  <a href="/day06_3/GoodServlet?name=ACER">ACER</a><br/>
  <a href="/day06_3/GoodServlet?name=DELL">DELL</a><br/>

  <hr/>
  您浏览过的商品:
  <%
  Cookie[] cs = request.getCookies();
  if(cs != null) {
      for(Cookie c : cs) {
          if(c.getName().equals("goods")) {
              out.print(c.getValue());
          }
      }
  }
  %>
```

```
  </body>
```

GoodServlet

```
public class GoodServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        String goodName = request.getParameter("name");
        String goods = CookieUtils.getCookValue(request, "goods");

        if(goods != null) {
            String[] arr = goods.split(", ");
            Set<String> goodSet = new LinkedHashSet(Arrays.asList(arr));
            goodSet.add(goodName);
            goods = goodSet.toString();
            goods = goods.substring(1, goods.length() - 1);
        } else {
            goods = goodName;
        }
        Cookie cookie = new Cookie("goods", goods);
        cookie.setMaxAge(1 * 60 * 60 * 24);
        response.addCookie(cookie);

        response.sendRedirect("/day06_3/index.jsp");
    }
}
```

CookieUtils

```
public class CookieUtils {
    public static String getCookValue(HttpServletRequest request, String name)
{
        Cookie[] cs = request.getCookies();
        if(cs == null) {
            return null;
        }
        for(Cookie c : cs) {
            if(c.getName().equals(name)) {
                return c.getValue();
            }
        }
        return null;
    }
}
```

HttpSession

1 HttpSession 概述

1.1 什么是 HttpSesssion

javax.servlet.http.HttpSession 接口表示一个会话，我们可以把一个会话内需要共享的数据保存到 HttSession 对象中！

1.2 获取 HttpSession 对象

- HttpSession request.getSesssion()：如果当前会话已经有了 session 对象那么直接返回，如果当前会话还不存在会话，那么创建 session 并返回；
- HttpSession request.getSession(boolean)：当参数为 true 时，与 requeset.getSession()相同。如果参数为 false，那么如果当前会话中存在 session 则返回，不存在返回 null；

1.3 HttpSession 是域对象

我们已经学习过 HttpServletRequest、ServletContext，它们都是域对象，现在我们又学习了一个 HttpSession，它也是域对象。它们三个是 Servlet 中可以使用的域对象，而 JSP 中可以多使用一个域对象，明天我们再讲解 JSP 的第四个域对象。

- HttpServletRequest：一个请求创建一个 request 对象，所以在同一个请求中可以共享 request，例如一个请求从 AServlet 转发到 BServlet，那么 AServlet 和 BServlet 可以共享 request 域中的数据；
- ServletContext：一个应用只创建一个 ServletContext 对象，所以在 ServletContext 中的数据可以在整个应用中共享，只要不启动服务器，那么 ServletContext 中的数据就可以共享；
- HttpSession：一个会话创建一个 HttpSession 对象，同一会话中的多个请求中可以共享 session 中的数据；

下载是 session 的域方法：

- void setAttribute(String name, Object value)：用来存储一个对象，也可以称之为存储一个域属性，例如：session.setAttribute(“xxx”, “XXX”)，在 session 中保存了一个域属性，域属性名称为 xxx，域属性的值为 XXX。请注意，如果多次调用该方法，并且使用相同的 name，那么会覆盖上一次的值，这一特性与 Map 相同；
- Object getAttribute(String name)：用来获取 session 中的数据，当前在获取之前需要先去存储才行，例如：String value = (String) session.getAttribute(“xxx”);，获取名为 xxx 的域属性；
- void removeAttribute(String name)：用来移除 HttpSession 中的域属性，如果参数 name 指定的域属性不存在，那么本方法什么都不做；
- Enumeration getAttributeNames()：获取所有域属性的名称；

2 登录案例

需要的页面：

- login.jsp：登录页面，提供登录表单；
- index1.jsp：主页，显示当前用户名称，如果没有登录，显示您还没登录；
- index2.jsp：主页，显示当前用户名称，如果没有登录，显示您还没登录；

Servlet：

- LoginServlet：在 login.jsp 页面提交表单时，请求本 Servlet。在本 Servlet 中获取用户名、密码进行校验，如果用户名、密码错误，显示“用户名或密码错误”，如果正确保存用户名 session 中，然后重定向到 index1.jsp；

当用户没有登录时访问 index1.jsp 或 index2.jsp，显示“您还没有登录”。如果用户在 login.jsp 登录成功后到达 index1.jsp 页面会显示当前用户名，而且不用再次登录去访问 index2.jsp 也会显示用户名。因为多次请求在一个会话范围，index1.jsp 和 index2.jsp 都会到 session 中获取用户名，session 对象在一个会话中是相同的，所以都可以获取到用户名！

login.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>login.jsp</title>
  </head>

  <body>
    <h1>login.jsp</h1>
    <hr/>
    <form action="/day06_4/LoginServlet" method="post">
    用户名：<input type="text" name="username" /><br/>
        <input type="submit" value="Submit"/>
    </form>
  </body>
</html>
```

index1.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
```

```
    <title>index1.jsp</title>
  </head>

  <body>
<h1>index1.jsp</h1>
<%
    String username = (String)session.getAttribute("username");
    if(username == null) {
        out.print("您还没有登录！");
    } else {
        out.print("用户名：" + username);
    }
%>
<hr/>
<a href="/day06_4/index2.jsp">index2</a>
  </body>
</html>
```

index2.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>index2.jsp</title>
  </head>

  <body>
<h1>index2.jsp</h1>
<%
    String username = (String)session.getAttribute("username");
    if(username == null) {
        out.print("您还没有登录！");
    } else {
        out.print("用户名：" + username);
    }
%>
<hr/>
<a href="/day06_4/index1.jsp">index1</a>
  </body>
</html>
```

LoginServlet

```
public class LoginServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html;charset=utf-8");

        String username = request.getParameter("username");

        if(username.equalsIgnoreCase("itcast")) {
            response.getWriter().print("用户名或密码错误！");
        } else {
            HttpSession session = request.getSession();
            session.setAttribute("username", username);
            response.sendRedirect("/day06_4/index1.jsp");
        }
    }
}
```

3 session 的实现原理

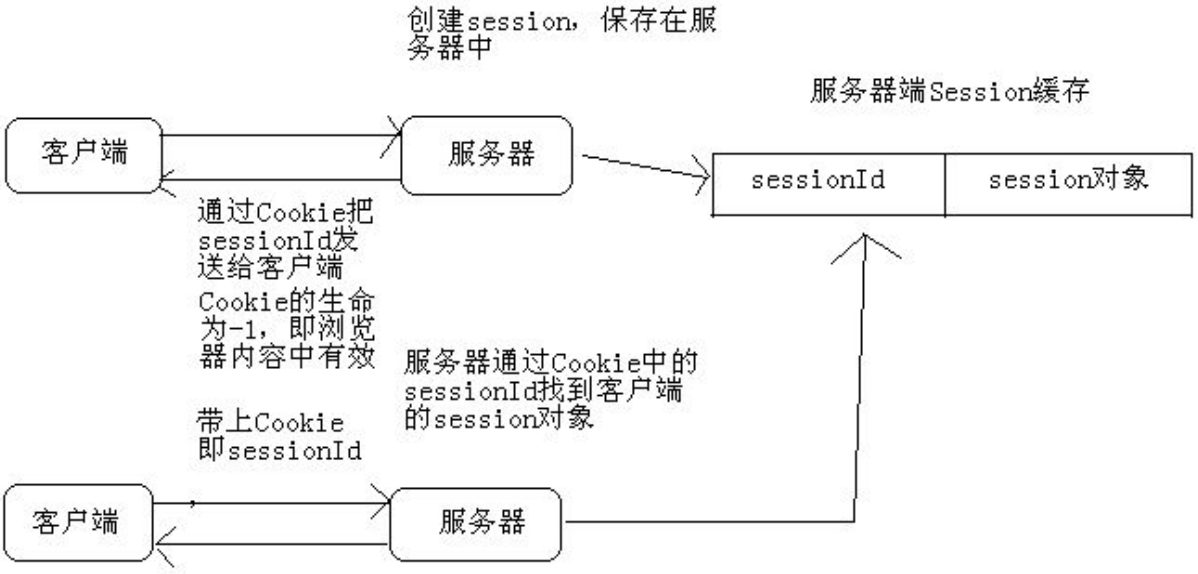
session 底层是依赖 Cookie 的！我们来理解一下 session 的原理吧！

当我首次去银行时，因为还没有账号，所以需要开一个账号，我获得的是银行卡，而银行这边的数据库中留下了我的账号，我的钱是保存在银行的账号中，而我带走的是我的卡号。

当我再次去银行时，只需要带上我的卡，而无需再次开一个账号了。只要带上我的卡，那么我在银行操作的一定是我的账号！

当首次使用 session 时，服务器端要创建 session，session 是保存在服务器端，而给客户端的 session 的 id（一个 cookie 中保存了 sessionId）。客户端带走的是 sessionId，而数据是保存在 session 中。

当客户端再次访问服务器时，在请求中会带上 sessionId，而服务器会通过 sessionId 找到对应的 session，而无需再创建新的 session。



4 session 与浏览器

session 保存在服务器，而 sessionId 通过 Cookie 发送给客户端，但这个 Cookie 的生命不-1，即只在浏览器内存中存在，也就是说如果用户关闭了浏览器，那么这个 Cookie 就丢失了。

当用户再次打开浏览器访问服务器时，就不会有 sessionId 发送给服务器，那么服务器会认为你没有 session，所以服务器会创建一个 session，并在响应中把 sessionId 中到 Cookie 中发送给客户端。

你可能会说，那原来的 session 对象会怎样？当一个 session 长时间没人使用的话，服务器会把 session 删除了！这个时长在 Tomcat 中配置是 30 分钟，可以在${CATALANA}/conf/web.xml 找到这个配置，当然你也可以在自己的 web.xml 中覆盖这个配置！

web.xml

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

session 失效时间也说明一个问题！如果你打开网站的一个页面开始长时间不动，超出了 30 分钟后，再去点击链接或提交表单时你会发现，你的 session 已经丢失了！

5 session 其他常用 API

- String getId()：获取 sessionId；
- int getMaxInactiveInterval()：获取 session 可以的最大不活动时间（秒），默认为 30 分钟。当 session 在 30 分钟内没有使用，那么 Tomcat 会在 session 池中移除这个 session；
- void setMaxInactiveInterval(int interval)：设置 session 允许的最大不活动时间（秒），如果设置为 1 秒，那么只要 session 在 1 秒内不被使用，那么 session 就会被移除；
- long getCreationTime()：返回 session 的创建时间，返回值为当前时间的毫秒值；
- long getLastAccessedTime()：返回 session 的最后活动时间，返回值为当前时间的毫秒值；
- void invalidate()：让 session 失效！调用这个方法会被 session 失效，当 session 失效后，客

户端再次请求，服务器会给客户端创建一个新的 session，并在响应中给客户端新 session 的 sessionId；

- boolean isNew()：查看 session 是否为新。当客户端第一次请求时，服务器为客户端创建 session，但这时服务器还没有响应客户端，也就是还没有把 sessionId 响应给客户端时，这时 session 的状态为新。

6 URL 重写

我们知道 session 依赖 Cookie，那么 session 为什么依赖 Cookie 呢？因为服务器需要在每次请求中获取 sessionId，然后找到客户端的 session 对象。那么如果客户端浏览器关闭了 Cookie 呢？那么 session 是不是就会不存在了呢？

其实还有一种方法让服务器收到的每个请求中都带有 sessioinId，那就是 URL 重写！在每个页面中的每个链接和表单中都添加名为 jSessionId 的参数，值为当前 sessionid。当用户点击链接或提交表单时也服务器可以通过获取 jSessionId 这个参数来得到客户端的 sessionId，找到 sessoin 对象。

index.jsp

```
  <body>
<h1>URL重写</h1>
<a href='/day06_5/index.jsp;jsessionid=<%=session.getId() %>' >主页</a>

<form action='/day06_5/index.jsp;jsessionid=<%=session.getId() %>'
method="post">
    <input type="submit" value="提交"/>
</form>
  </body>
```

也可以使用 response.encodeURL()对每个请求的 URL 处理，这个方法会自动追加 jsessionid 参数，与上面我们手动添加是一样的效果。

```
<a href='<%=response.encodeURL("/day06_5/index.jsp") %>' >主页</a>

<form action='<%=response.encodeURL("/day06_5/index.jsp") %>' method="post">
    <input type="submit" value="提交"/>
</form>
```

使用 response.encodeURL()更加“智能”，它会判断客户端浏览器是否禁用了 Cookie，如果禁用了，那么这个方法在 URL 后面追加 jsessionid，否则不会追加。

案例：一次性图片验证码

1 验证码有啥用

在我们注册时，如果没有验证码的话，我们可以使用 URLConnection 来写一段代码发出注册请求。甚至可以使用 while(true)来注册！那么服务器就废了！

验证码可以去识别发出请求的是人还是程序！当然，如果聪明的程序可以去分析验证码图片！但分析图片也不是一件容易的事，因为一般验证码图片都会带有干扰线，人都看不清，那么程序一定分析不出来。

2 VerifyCode 类

现在我们已经有了 cn.itcast.utils.VerifyCode 类，这个类可以生成验证码图片！下面来看一个小例子。

```
    public void fun1() throws IOException {
        // 创建验证码类
        VerifyCode vc = new VerifyCode();
        // 获取随机图片
        BufferedImage image = vc.getImage();
        // 获取刚刚生成的随机图片上的文本
        String text = vc.getText();
        System.out.println(text);
        // 保存图片
        FileOutputStream out = new FileOutputStream("F:/xxx.jpg");
        VerifyCode.output(image, out);
    }
```

3 在页面中显示动态图片

我们需要写一个 VerifyCodeServlet，在这个 Servlet 中我们生成动态图片，然后它图片写入到 response.getOutputStream()流中！然后让页面的<img>元素指定这个 VerifyCodServlet 即可。

VerifyCodeServlet

```
public class VerifyCodeServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        VerifyCode vc = new VerifyCode();
        BufferedImage image = vc.getImage();
        String text = vc.getText();
        System.out.println("text:" + text);
        VerifyCode.output(image, response.getOutputStream());
    }
}
```

index.jsp

```
    <script type="text/javascript">
    function _change() {
        var imgEle = document.getElementById("vCode");
        imgEle.src = "/day06_6/VerifyCodeServlet?" + new Date().getTime();
    }
    </script>
...
  <body>
    <h1>验证码</h1>
    <img id="vCode" src="/day06_6/VerifyCodeServlet"/>
    <a href="javascript:_change()">看不清，换一张</a>
  </body>
```

4 在注册页面中使用验证码

```
    <form action="/day06_6/RegistServlet" method="post">
    用户名：<input type="text" name="username"/><br/>
    验证码：<input type="text" name="code" size="3"/>
    <img id="vCode" src="/day06_6/VerifyCodeServlet"/>
    <a href="javascript:_change()">看不清，换一张</a>
    <br/>
       <input type="submit" value="Submit"/>
    </form>
```

5 RegistServlet

修改 VerifyCodeServlet

```
public class VerifyCodeServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        VerifyCode vc = new VerifyCode();
        BufferedImage image = vc.getImage();
        request.getSession().setAttribute("vCode", vc.getText());
        VerifyCode.output(image, response.getOutputStream());
    }
}
```

RegistServlet

```
public class RegistServlet extends HttpServlet {
```

```
    public void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html;charset=utf-8");

        String username = request.getParameter("username");
        String vCode = request.getParameter("code");

        String sessionVerifyCode =
(String)request.getSession().getAttribute("vCode");

        if(vCode.equalsIgnoreCase(sessionVerifyCode)) {
            response.getWriter().print(username + ", 恭喜！注册成功！");
        } else {
            response.getWriter().print("验证码错误！");
        }
    }
}
```

6 总结验证码案例

- VerifyCodeServlet：
  - 生成验证码：VerifyCode vc = new VerifyCode(); BufferedImage image = vc.getImage();
  - 在 session 中保存验证码文本：request.getSession.getAttribute(“vCode”, vc.getText());
  - 把验证码输出到页面：VerifyCode.output(image, response.getOutputStream);
- regist.jsp：
  - 表单中包含 username 和 code 字段；
  - 在表单中给出<img>指向 VerifyCodeServlet，用来在页面中显示验证码图片；
  - 提供“看不清，换一张”链接，指向_change()函数；
  - 提交到 RegistServlet；
- RegistServlet：
  - 获取表单中的 username 和 code；
  - 获取 session 中的 vCode；
  - 比较 code 和 vCode 是否相同；
  - 相同说明用户输入的验证码正确，否则输入验证码错误。