**cookie、session及实现记住密码，自动登录思路**

在登录帐号、密码框下，有三种帐号登录模式可供选择，用户可根据自己的具体情况选择其中一种适合自己的模式。

**1、网吧模式：**勾选网吧模式后，登录的帐号会在歪歪注销/退出的时候自动清除，不会留在登录框中，可以保护自己的帐号不对外泄露，建议在外地上网时使用，比方网吧或者其他人机器上登录时。

**2、记住密码：**勾选记住密码后，系统会记住你所登录的帐号和密码，注销后帐号和密码会自动留存在登录框，退出后再登录，帐号、密码会自动出现在登录框。方便下次登录，建议在自己电脑上使用，以免造成不必要的损失。

**3、自动登录：**记住密码后，再勾选上“自动登录”，这样下次登录时，会默认上次记住的帐号和密码，自动连接登录。

**一、cookie机制和session机制的区别**  
　　具体来说cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。  
　　同时我们也看到，由于在服务器端保持状态的方案在客户端也需要保存一个标识，所以session机制可能需要借助于cookie机制来达到保存标识的目的，但实际上还有其他选择。  
**二、会话cookie和持久cookie的区别**  
**如果不设置过期时间，则表示这个cookie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。这种生命期为浏览会话期的cookie被称为**会话cookie**。会话cookie一般不保存在硬盘上而是保存在内存里。  
**如果设置了过期时间，浏览器就会把cookie保存到硬盘上，关闭后再次打开浏览器，这些cookie依然有效直到超过设定的过期时间**。  
　　存储在硬盘上的cookie可以在不同的浏览器进程间共享，比如两个IE窗口。而对于保存在内存的cookie，不同的浏览器有不同的处理方式。  
**三、如何利用实现自动登录**  
　　当用户在某个网站注册后，就会收到一个惟一用户ID的cookie。客户后来重新连接时，这个用户ID会自动返回，服务器对它进行检查，确定它是否为注册用户且选择了自动登录，从而使用户无需给出明确的用户名和密码，就可以访问服务器上的资源。  
**四、如何根据用户的爱好定制站点**  
　　网站可以使用cookie记录用户的意愿。对于简单的设置，网站可以直接将页面的设置存储在cookie中完成定制。然而对于更复杂的定制，网站只需仅将一个惟一的标识符发送给用户，由服务器端的数据库存储每个标识符对应的页面设置。  
**五、cookie的发送**  
1.创建Cookie对象  
2.设置最大时效  
3.将Cookie放入到HTTP响应报头  
　　如果你创建了一个cookie，并将他发送到浏览器，默认情况下它是一个会话级别的cookie:存储在浏览器的内存中，用户退出浏览器之后被删除。如果你希望浏览器将该cookie存储在磁盘上，则需要使用maxAge，并给出一个以秒为单位的时间。将最大时效设为0则是命令浏览器删除该 cookie。  
　　发送cookie需要使用HttpServletResponse的addCookie方法，将cookie插入到一个 Set-Cookie HTTP请求报头中。由于这个方法并不修改任何之前指定的Set-Cookie报头，而是创建新的报头，因此我们将这个方法称为是addCookie，而非setCookie。同样要记住响应报头必须在任何文档内容发送到客户端之前设置。  
**六、cookie的读取**  
1.调用request.getCookie  
　　要获取有浏览器发送来的cookie，需要调用HttpServletRequest的getCookies方法，这个调用返回Cookie对象的数组，对应由HTTP请求中Cookie报头输入的值。  
2.对数组进行循环，调用每个cookie的getName方法，直到找到感兴趣的cookie为止  
　　cookie与你的主机(域)相关，而非你的servlet或JSP页面。因而，尽管你的servlet可能只发送了单个cookie，你也可能会得到许多不相关的cookie。  
例如：  
　　String cookieName = “userID”;  
   Cookie cookies［］ = request.getCookies();  
   if (cookies!=null){  
       for(int i=0;i<cookies.length;i++){  
   Cookie cookie = cookies［i］;  
   if (cookieName.equals(cookie.getName())){  
       doSomethingWith(cookie.getValue());  
}  
}  
}  
**七、如何使用cookie检测初访者**  
A.调用HttpServletRequest.getCookies()获取Cookie数组  
B.在循环中检索指定名字的cookie是否存在以及对应的值是否正确  
C.如果是则退出循环并设置区别标识  
D.根据区别标识判断用户是否为初访者从而进行不同的操作  
**八、使用cookie检测初访者的常见错误**  
　　不能仅仅因为cookie数组中不存在在特定的数据项就认为用户是个初访者。如果cookie数组为null，客户可能是一个初访者，也可能是由于用户将cookie删除或禁用造成的结果。  
　　但是，如果数组非null,也不过是显示客户曾经到过你的网站或域，并不能说明他们曾经访问过你的servlet。其它servlet、JSP页面以及非Java Web应用都可以设置cookie，依据路径的设置，其中的任何cookie都有可能返回给用户的浏览器。  
　　正确的做法是判断cookie数组是否为空且是否存在指定的Cookie对象且值正确。  
**九、使用cookie属性的注意问题**  
　　属性是从服务器发送到浏览器的报头的一部分；但它们不属于由浏览器返回给服务器的报头。　  
　　因此除了名称和值之外，cookie属性只适用于从服务器输出到客户端的cookie；服务器端来自于浏览器的cookie并没有设置这些属性。　  
　　因而不要期望通过request.getCookies得到的cookie中可以使用这个属性。这意味着，你不能仅仅通过设置cookie的最大时效，发出它，在随后的输入数组中查找适当的cookie,读取它的值，修改它并将它存回Cookie，从而实现不断改变的cookie值。  
**十、如何使用cookie记录各个用户的访问计数**  
1.获取cookie数组中专门用于统计用户访问次数的cookie的值  
2.将值转换成int型  
3.将值加1并用原来的名称重新创建一个Cookie对象  
4.重新设置最大时效  
5.将新的cookie输出  
**十一、session在不同环境下的不同含义**  
　　session，中文经常翻译为会话，其本来的含义是指有始有终的一系列动作/消息，比如打电话是从拿起电话拨号到挂断电话这中间的一系列过程可以称之为一个session。  
　　然而当session一词与网络协议相关联时，它又往往隐含了“面向连接”和/或“保持状态”这样两个含义。  
　　session在[web开发](http://bbs.it-home.org/forum-web-1.html" \t "_blank)环境下的语义又有了新的扩展，它的含义是指一类用来在客户端与服务器端之间保持状态的解决方案。有时候Session也用来指这种解决方案的存储结构。  
**十二、session的机制**  
　　session机制是一种服务器端的机制，服务器使用一种类似于散列表的结构(也可能就是使用散列表)来保存信息。  
　　但程序需要为某个客户端的请求创建一个session的时候，服务器首先检查这个客户端的请求里是否包含了一个session标识－称为session id,如果已经包含一个session id则说明以前已经为此客户创建过session，服务器就按照session id把这个session检索出来使用(如果检索不到，可能会新建一个，这种情况可能出现在服务端已经删除了该用户对应的session对象，但用户人为地在请求的URL后面附加上一个JSESSION的参数)。  
　　如果客户请求不包含session id，则为此客户创建一个session并且生成一个与此session相关联的session id，这个session id将在本次响应中返回给客户端保存。  
**十三、保存session id的几种方式**  
A．保存session id的方式可以采用cookie，这样在交互过程中浏览器可以自动的按照规则把这个标识发送给服务器。  
B．由于cookie可以被人为的禁止，必须有其它的机制以便在cookie被禁止时仍然能够把session id传递回服务器，经常采用的一种技术叫做URL重写，就是把session id附加在URL路径的后面，附加的方式也有两种，一种是作为URL路径的附加信息，另一种是作为查询字符串附加在URL后面。网络在整个交互过程中始终保持状态，就必须在每个客户端可能请求的路径后面都包含这个session id。  
C．另一种技术叫做表单隐藏字段。就是服务器会自动修改表单，添加一个隐藏字段，以便在表单提交时能够把session id传递回服务器。  
**十四、session什么时候被创建**  
　　一个常见的错误是以为session在有客户端访问时就被创建，然而事实是直到某server端程序(如Servlet)调用HttpServletRequest.getSession(true)这样的语句时才会被创建。  
**十五、session何时被删除**  
session在下列情况下被删除：  
A．程序调用HttpSession.invalidate()  
B．距离上一次收到客户端发送的session id时间间隔超过了session的最大有效时间  
C．服务器进程被停止  
　　再次注意关闭浏览器只会使存储在客户端浏览器内存中的session cookie失效，不会使服务器端的session对象失效。  
**十六、URL重写有什么缺点**  
　　对所有的URL使用URL重写，包括超链接，form的action，和重定向的URL。每个引用你的站点的URL，以及那些返回给用户的URL(即使通过间接手段，比如服务器重定向中的Location字段)都要添加额外的信息。  
　　这意味着在你的站点上不能有任何静态的HTML页面(至少静态页面中不能有任何链接到站点动态页面的链接)。因此，每个页面都必须使用servlet或 JSP动态生成。即使所有的页面都动态生成，如果用户离开了会话并通过书签或链接再次回来，会话的信息都会丢失，因为存储下来的链接含有错误的标识信息－该URL后面的SESSION ID已经过期了。　　  
**十七、使用隐藏的表单域有什么缺点**  
   仅当每个页面都是有表单提交而动态生成时，才能使用这种方法。单击常规的<A HREF..>超文本链接并不产生表单提交，因此隐藏的表单域不能支持通常的会话跟踪，只能用于一系列特定的操作中，比如在线商店的结账过程  
**十八、会话跟踪的基本步骤**  
1．访问与当前请求相关的会话对象  
2．查找与会话相关的信息  
3．存储会话信息  
4．废弃会话数据  
**十九、getSession()/getSession(true)、getSession(false)的区别**  
getSession()/getSession(true)：当session存在时返回该session，否则新建一个session并返回该对象  
getSession(false)：当session存在时返回该session，否则不会新建session，返回null  
**二十、如何将信息与会话关联起来**  
　　setAttribute会替换任何之前设定的值；如果想要在不提供任何代替的情况下移除某个值，则应使用removeAttribute。这个方法会触发所有实现了HttpSessionBindingListener接口的值的valueUnbound方法。  
**二十一、会话属性的类型有什么限制吗**  
　　通常会话属性的类型只要是Object就可以了。除了null或基本类型，如int,double,boolean。  
　　如果要使用基本类型的值作为属性，必须将其转换为相应的封装类对象  
**二十二、如何废弃会话数据**  
A．只移除自己编写的servlet创建的数据：  
  调用removeAttribute(“key”)将指定键关联的值废弃  
B．删除整个会话(在当前Web应用中)：  
  调用invalidate，将整个会话废弃掉。这样做会丢失该用户的所有会话数据，而非仅仅由我们servlet或JSP页面创建的会话数据  
C．将用户从系统中注销并删除所有属于他(或她)的会话  
  调用logOut，将客户从Web服务器中注销，同时废弃所有与该用户相关联的会话(每个Web应用至多一个)。这个操作有可能影响到服务器上多个不同的Web应用。  
**二十三、使用isNew来判断用户是否为新旧用户的错误做法**  
　　public boolean isNew()方法如果会话尚未和客户程序(浏览器)发生任何联系，则这个方法返回true，这一般是因为会话是新建的，不是由输入的客户请求所引起的。  
　　但如果isNew返回false，只不过是说明他之前曾经访问该Web应用，并不代表他们曾访问过我们的servlet或JSP页面。  
　　因为session是与用户相关的，在用户之前访问的每一个页面都有可能创建了会话。因此isNew为false只能说用户之前访问过该Web应用，session可以是当前页面创建，也可能是由用户之前访问过的页面创建的。  
　　正确的做法是判断某个session中是否存在某个特定的key且其value是否正确  
**二十四、Cookie的过期和Session的超时有什么区别**  
　　会话的超时由服务器来维护，它不同于Cookie的失效日期。首先，会话一般基于驻留内存的cookie不是持续性的cookie，因而也就没有截至日期。即使截取到JSESSIONID cookie，并为它设定一个失效日期发送出去。浏览器会话和服务器会话也会截然不同。  
**二十五、session cookie和session对象的生命周期是一样的吗**  
　　当用户关闭了浏览器虽然session cookie已经消失，但session对象仍然保存在服务器端  
**二十六、是否只要关闭浏览器，session就消失了**  
　　程序一般都是在用户做log off的时候发个指令去删除session，然而浏览器从来不会主动在关闭之前通知服务器它将要被关闭，因此服务器根本不会有机会知道浏览器已经关闭。服务器会一直保留这个会话对象直到它处于非活动状态超过设定的间隔为止。  
　　之所以会有这种错误的认识，是因为大部分session机制都使用会话cookie来保存session id，而关闭浏览器后这个session id就消失了，再次连接到服务器时也就无法找到原来的session。  
　　如果服务器设置的cookie被保存到硬盘上，或者使用某种手段改写浏览器发出的HTTP请求报头，把原来的session id发送到服务器，则再次打开浏览器仍然能够找到原来的session。  
　　恰恰是由于关闭浏览器不会导致session被删除，迫使服务器为session设置了一个失效时间，当距离客户上一次使用session的时间超过了这个失效时间时，服务器就可以认为客户端已经停止了活动，才会把session删除以节省存储空间。  
　　由此我们可以得出如下结论：  
　　关闭浏览器，只会是浏览器端内存里的session cookie消失，但不会使保存在服务器端的session对象消失，同样也不会使已经保存到硬盘上的持久化cookie消失。  
**二十七、打开两个浏览器窗口访问应用程序会使用同一个session还是不同的session**  
　　通常session cookie是不能跨窗口使用的，当你新开了一个浏览器窗口进入相同页面时，系统会赋予你一个新的session id，这样我们信息共享的目的就达不到了。  
　　此时我们可以先把session id保存在persistent cookie中(通过设置session的最大有效时间)，然后在新窗口中读出来，就可以得到上一个窗口的session id了，这样通过session cookie和persistent cookie的结合我们就可以实现了跨窗口的会话跟踪。  
**二十八、如何使用会话显示每个客户的访问次数**  
　　由于客户的访问次数是一个整型的变量，但session的属性类型中不能使用int，double，boolean等基本类型的变量，所以我们要用到这些基本类型的封装类型对象作为session对象中属性的值  
　　但像Integer是一种不可修改(Immutable)的数据结构：构建后就不能更改。这意味着每个请求都必须创建新的Integer对象，之后使用setAttribute来代替之前存在的老的属性的值。例如：  
HttpSession session = request.getSession();  
SomeImmutalbeClass value = (SomeImmutableClass)session.getAttribute(“SomeIdentifier”);  
if (value= =null){  
   value = new SomeImmutableClass(…);　// 新创建一个不可更改对象  
}else{  
   value = new SomeImmutableClass(calculatedFrom(value)); // 对value重新计算后创建新的对象  
}  
session.setAttribute(“someIdentifier”,value); // 使用新创建的对象覆盖原来的老的对象  
**二十九、如何使用会话累计用户的数据**  
　　使用可变的数据结构，比如数组、List、Map或含有可写字段的应用程序专有的数据结构。通过这种方式，除非首次分配对象，否则不需要调用setAttribute。例如  
HttpSession session = request.getSession();  
SomeMutableClass value = (SomeMutableClass)session.getAttribute(“someIdentifier”);  
if(value = = null){  
   value = new SomeMutableClass(…);  
   session.setAttribute(“someIdentifier”,value);  
}else{  
   value.updateInternalAttribute(…);     // 如果已经存在该对象则更新其属性而不需重新设置属性  
}  
**三十、不可更改对象和可更改对象在会话数据更新时的不同处理**  
　　不可更改对象因为一旦创建之后就不能更改，所以每次要修改会话中属性的值的时候，都需要调用 setAttribute(“someIdentifier”,newValue)来代替原有的属性的值，否则属性的值不会被更新可更改对象因为其自身一般提供了修改自身属性的方法，所以每次要修改会话中属性的值的时候，只要调用该可更改对象的相关修改自身属性的方法就可以了。这意味着我们就不需要调用 setAttribute方法了。

具体方法代码推荐： [java web项目用cookie记住用户名、密码](http://bbs.it-home.org/thread-3585-1-1.html)

**1.jsp页面：**

01

<% String flag = (String)session.getAttribute("flag")==null?"":(String)session.getAttribute("flag");

02

03

String name = "";

04

String password = "";

05

try{

06

Cookie[] cookies=request.getCookies();

07

if(cookies!=null){

08

for(int i=0;i<cookies.length;i++){

09

if(cookies[i].getName().equals("cookie\_user")){

10

String value = cookies[i].getValue();

11

if(value!=null&&!"".equals(value)){

12

name=cookies[i].getValue().split("-")[0];

13

if(cookies[i].getValue().split("-")[1]!=null && !cookies[i].getValue().split("-")[1].equals("null")){

01

password=cookies[i].getValue().split("-")[1];

02

}

03

04

}

05

}

06

request.setAttribute("name",name);

07

request.setAttribute("passward",password);

08

}

09

}

10

}catch(Exception e){

11

e.printStackTrace();

12

}

13

%>

14

<body>

15

<div id="logo"><img src="<%=request.getContextPath() %>/frontsite/Images/Logo.jpg" width="244" height="44" alt="" /></div>

16

<div id="loginfrm">

17

<form action="<%=request.getContextPath() %>/frontsite/login.do?method=login" method="post" id="forms" onsubmit="checkForm();return false;">

18

<div class="frm">

19

<ul>

20

<li class="l1">用户名：</li>

21

<li class="l2"><input name="login\_id" type="text" id="login\_id" class="input" value="<%=name %>"/></li>

22

<li class="l3">\* 最大限度为20字节</li>

23

<li class="l1">密码：</li>

24

<li class="l2"><input name="login\_pwd" type="password" id="login\_pwd" class="input" value="<%=password %>"/></li>

25

<li class="l3">\* 最大限度为20字节</li>

26

27

<li class="l1">验证码：

28

<li class="l2" style="width: 180px;"><input name="checkImg" id="checkImg" type="text" class="input" style="width: 60px;" size="10" />

29

<img src="<%=request.getContextPath() %>/CheckImg\_FT" width="49" height="22" /></li>

30

31

<li class="l4">

32

<label><input type="checkbox" name="flag" id="flag" value="1" <%if(flag!=null && flag.equals("1")){%> checked ; value ="1"; <%}else {%> value="0" <%;}%> />记住密码</label>

33

<label><span style="margin-left:10px; color: #F00;" ><html:errors /></span></label>

34

</li>

35

36

<li class="l4"><input type="image" name="imageField" id="imageField" src="<%=request.getContextPath() %>/frontsite/Images/login\_bnt.jpg" /></li>

37

</ul>

38

</div>

39

</form>

40

</div>

41

</body>

**2.java 类：**

01

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

02

//set cookie

03

if(flag!=null && flag.equals("1")){

04

Cookie cookie = new Cookie("cookie\_user", po.getLogin\_id()+"-"+form.getLogin\_pwd());

05

cookie.setMaxAge(60\*60\*24\*30); //cookie 保存30天

06

response.addCookie(cookie);

07

}else{

08

Cookie cookie = new Cookie("cookie\_user",po.getLogin\_id()+"-"+null);

09

cookie.setMaxAge(60\*60\*24\*30); //cookie 保存30天

10

response.addCookie(cookie);

11

}

if (checkBox.isSelected()) {

setCookie("exusername", userName);

} else {

removeCookie("exusername");

}

\*\*

\*给cookie赋值

\* @param cookieName

\* @param cookieValue

\*

public void setCookie(String cookieName, String cookieValue) {

String values = null;

System.out.println("setCookie");

try {

values = URLEncoder.encode(cookieValue, "UTF-8");

System.out.println("setCookie1" + values);

} catch (Exception e) {

e.printStackTrace();

}

Cookie cookie = new Cookie(cookieName, values);

HttpServletResponse response =

(HttpServletResponse)FacesContext.getCurrentInstance().getExternalContext().getResponse();

cookie.setMaxAge(9999999);

cookie.setPath("/");

response.addCookie(cookie);

}

\*\*

\*删除cookie

\* @param cookieName

\*/

public void removeCookie(String cookieName) {

Cookie cookie = new Cookie(cookieName, null);

HttpServletResponse response =

(HttpServletResponse)FacesContext.getCurrentInstance().getExternalContext().getResponse();

cookie.setMaxAge(0);

cookie.setPath("/");

response.addCookie(cookie);

}

\*\*

\*获取cookie值

\* @param cookieName

\* @return

\*

public String getCookieValue(String cookieName) {

String CookieValue = null;

HttpServletRequest request =

(HttpServletRequest)FacesContext.getCurrentInstance().getExternalContext().getRequest();

Cookie[] cookies = request.getCookies();

for (Cookie cookie : cookies) {

if (cookie.getName().toString().equals(cookieName)) {

try {

CookieValue =

URLDecoder.decode(cookie.getValue(), "UTF-8");

System.out.println("11111111111111" + CookieValue);

} catch (Exception e) {

e.printStackTrace();

}

}

}

return CookieValue;

}

\*\*

\*初始化

\* @param phaseEvent

\*

public void HomePagebeforePhase(PhaseEvent phaseEvent) {

// Add event code here...

System.out.println(name.getValue());

if (name.getValue() == null) {

if (getCookieValue("exusername") != null) {

checkBox.setSelected(true);

name.setValue(getCookieValue("exusername"));

}

}

}