**Cookie/Session机制、区别详解**

作者：   发布时间：2015-10-27 09:44:09   浏览次数：570

会话（Session）跟踪是Web程序中常用的技术，用来**跟踪用户的整个会话**。常用的会话跟踪技术是Cookie与Session。**Cookie通过在客户端记录信息确定用户身份**，**Session通过在服务器端记录信息确定用户身份**。

本章将系统地讲述Cookie与Session机制，并比较说明什么时候不能用Cookie，什么时候不能用Session。

**1.1  Cookie机制**

在程序中，会话跟踪是很重要的事情。理论上，**一个用户的所有请求操作都应该属于同一个会话**，而另一个用户的所有请求操作则应该属于另一个会话，二者不能混淆。例如，用户A在超市购买的任何商品都应该放在A的购物车内，不论是用户A什么时间购买的，这都是属于同一个会话的，不能放入用户B或用户C的购物车内，这不属于同一个会话。

而Web应用程序是使用HTTP协议传输数据的。**HTTP协议是无状态的协议。一旦数据交换完毕，客户端与服务器端的连接就会关闭，再次交换数据需要建立新的连接。这就意味着服务器无法从连接上跟踪会话**。即用户A购买了一件商品放入购物车内，当再次购买商品时服务器已经无法判断该购买行为是属于用户A的会话还是用户B的会话了。要跟踪该会话，必须引入一种机制。

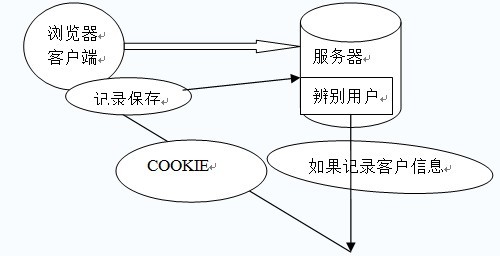
Cookie就是这样的一种机制。它可以弥补HTTP协议无状态的不足。在Session出现之前，基本上所有的网站都采用Cookie来跟踪会话。

**1.1.1  什么是Cookie**

Cookie意为“甜饼”，是**由W3C组织提出**，最早由Netscape社区发展的一种机制。目前Cookie已经成为标准，所有的主流浏览器如IE、Netscape、Firefox、Opera等都支持Cookie。

由于HTTP是一种无状态的协议，服务器单从网络连接上无从知道客户身份。怎么办呢？就**给客户端们颁发一个通行证吧，每人一个，无论谁访问都必须携带自己通行证。这样服务器就能从通行证上确认客户身份了。这就是Cookie的工作原理**。

Cookie实际上是一小段的文本信息。客户端请求服务器，如果服务器需要记录该用户状态，就使用response向客 户端浏览器颁发一个Cookie。客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务 器。服务器检查该Cookie，以此来辨认用户状态。服务器还可以根据需要修改Cookie的内容。



查看某个网站颁发的Cookie很简单。在浏览器地址栏输入**javascript:alert (document. cookie)**就可以了（需要有网才能查看）。JavaScript脚本会弹出一个对话框显示本网站颁发的所有Cookie的内容，如图1.1所示。



图1.1  Baidu网站颁发的Cookie

图1.1中弹出的对话框中显示的为Baidu网站的Cookie。其中第一行BAIDUID记录的就是笔者的身份helloweenvsfei，只是Baidu使用特殊的方法将Cookie信息加密了。

注意：Cookie功能需要浏览器的支持。

如果浏览器不支持Cookie（如大部分手机中的浏览器）或者把Cookie禁用了，Cookie功能就会失效。

不同的浏览器采用不同的方式保存Cookie。

IE浏览器会在“C:Documents and Settings你的用户名Cookies”文件夹下以文本文件形式保存，一个文本文件保存一个Cookie。

**1.1.2  记录用户访问次数**

Java中把Cookie封装成了javax.servlet.http.Cookie类。每个Cookie都是该Cookie类的对象。服务器通过操作Cookie类对象对客户端Cookie进行操作。通过**request.getCookie()获取客户端提交的所有Cookie**（以Cookie[]数组形式返回），**通过response.addCookie(Cookiecookie)向客户端设置Cookie。**

Cookie对象使用key-value属性对的形式保存用户状态，一个Cookie对象保存一个属性对，一个request或者response同时使用多个Cookie。因为Cookie类位于包javax.servlet.http.\*下面，所以JSP中不需要import该类。

**1.1.3  Cookie的不可跨域名性**

很多网站都会使用Cookie。例如，Google会向客户端颁发Cookie，Baidu也会向客户端颁发Cookie。那浏览器访问Google会不会也携带上Baidu颁发的Cookie呢？或者Google能不能修改Baidu颁发的Cookie呢？

答案是否定的。**Cookie具有不可跨域名性**。根据Cookie规范，浏览器访问Google只会携带Google的Cookie，而不会携带Baidu的Cookie。Google也只能操作Google的Cookie，而不能操作Baidu的Cookie。

Cookie在客户端是由浏览器来管理的。浏览器能够保证Google只会操作Google的Cookie而不会操作 Baidu的Cookie，从而保证用户的隐私安全。浏览器判断一个网站是否能操作另一个网站Cookie的依据是域名。Google与Baidu的域名 不一样，因此Google不能操作Baidu的Cookie。

需要注意的是，虽然网站images.google.com与网站www.google.com同属于Google，但是域名不一样，二者同样不能互相操作彼此的Cookie。

注意：用户登录网站www.google.com之后会发现访问images.google.com时登录信息仍然有效，而普通的Cookie是做不到的。这是因为Google做了特殊处理。本章后面也会对Cookie做类似的处理。

**1.1.4  Unicode编码：保存中文**

中文与英文字符不同，**中文属于Unicode字符，在内存中占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，否则会乱码。

提示：Cookie中保存中文只能编码。一般使用UTF-8编码即可。不推荐使用GBK等中文编码，因为浏览器不一定支持，而且JavaScript也不支持GBK编码。

**1.1.5  BASE64编码：保存二进制图片**

Cookie不仅可以使用ASCII字符与Unicode字符，还可以使用二进制数据。例如在Cookie中使用数字证书，提供安全度。使用二进制数据时也需要进行编码。

%注意：本程序仅用于展示Cookie中可以存储二进制内容，并不实用。由于浏览器每次请求服务器都会携带Cookie，因此Cookie内容不宜过多，否则影响速度。Cookie的内容应该少而精。

**1.1.6  设置Cookie的所有属性**

除了name与value之外，Cookie还具有其他几个常用的属性。每个属性对应一个getter方法与一个setter方法。Cookie类的所有属性如表1.1所示。

表1.1  Cookie常用属性

|  |  |
| --- | --- |
| 属  性  名 | 描    述 |
| String name | 该Cookie的名称。Cookie一旦创建，名称便不可更改 |
| Object value | 该Cookie的值。如果值为Unicode字符，需要为字符编码。如果值为二进制数据，则需要使用BASE64编码 |
| **int maxAge** | **该Cookie失效的时间，单位秒。如果为正数，则该Cookie在maxAge秒之后失效。如果为负数，该Cookie为临时Cookie，关闭浏览器即失效，浏览器也不会以任何形式保存该Cookie。如果为0，表示删除该Cookie。默认为–1** |
| boolean secure | 该Cookie是否仅被使用安全协议传输。安全协议。安全协议有HTTPS，SSL等，在网络上传输数据之前先将数据加密。默认为false |
| String path | 该Cookie的使用路径。如果设置为“/sessionWeb/”，则只有contextPath为“/sessionWeb”的程序可以访问该Cookie。如果设置为“/”，则本域名下contextPath都可以访问该Cookie。注意最后一个字符必须为“/” |
| String domain | 可以访问该Cookie的域名。如果设置为“.google.com”，则所有以“google.com”结尾的域名都可以访问该Cookie。注意第一个字符必须为“.” |
| String comment | 该Cookie的用处说明。浏览器显示Cookie信息的时候显示该说明 |
| int version | 该Cookie使用的版本号。0表示遵循Netscape的Cookie规范，1表示遵循W3C的RFC 2109规范 |

**1.1.7  Cookie的有效期**

Cookie的maxAge决定着Cookie的有效期，单位为秒（Second）。Cookie中通过getMaxAge()方法与setMaxAge(int maxAge)方法来读写maxAge属性。

如果maxAge属性为正数，则表示该Cookie会在maxAge秒之后自动失效。浏览器会将maxAge为正数的 Cookie持久化，即写到对应的Cookie文件中。无论客户关闭了浏览器还是电脑，只要还在maxAge秒之前，登录网站时该Cookie仍然有效。 下面代码中的Cookie信息将永远有效。

**Cookie cookie = new Cookie("username","helloweenvsfei");   // 新建Cookie**

**cookie.setMaxAge(Integer.MAX\_VALUE);           // 设置生命周期为MAX\_VALUE**

**response.addCookie(cookie);                    // 输出到客户端**

如果maxAge为负数，则表示该Cookie仅在本浏览器窗口以及本窗口打开的子窗口内有效，关闭窗口后该 Cookie即失效。maxAge为负数的Cookie，为临时性Cookie，不会被持久化，不会被写到Cookie文件中。Cookie信息保存在浏 览器内存中，因此关闭浏览器该Cookie就消失了。Cookie默认的maxAge值为–1。

如果maxAge为0，则表示删除该Cookie。Cookie机制没有提供删除Cookie的方法，因此通过设置该Cookie即时失效实现删除Cookie的效果。失效的Cookie会被浏览器从Cookie文件或者内存中删除，

例如：

Cookie cookie = new Cookie("username","helloweenvsfei");   // 新建Cookie

cookie.setMaxAge(0);                          // 设置生命周期为0，不能为负数

response.addCookie(cookie);                    // 必须执行这一句

response对象提供的Cookie操作方法只有一个添加操作add(Cookie cookie)。

要想修改Cookie只能使用一个同名的Cookie来覆盖原来的Cookie，达到修改的目的。删除时只需要把maxAge修改为0即可。

注意：从客户端读取Cookie时，包括maxAge在内的其他属性都是不可读的，也不会被提交。浏览器提交Cookie时只会提交name与value属性。maxAge属性只被浏览器用来判断Cookie是否过期。

**1.1.8  Cookie的修改、删除**

Cookie并不提供修改、删除操作。如果要修改某个Cookie，只需要新建一个同名的Cookie，添加到response中覆盖原来的Cookie。

如果要删除某个Cookie，只需要新建一个同名的Cookie，并将maxAge设置为0，并添加到response中覆盖原来的Cookie。注意是0而不是负数。负数代表其他的意义。读者可以通过上例的程序进行验证，设置不同的属性。

注意：修改、删除Cookie时，新建的Cookie除value、maxAge之外的所有属性，例如name、path、domain等，都要与原Cookie完全一样。否则，浏览器将视为两个不同的Cookie不予覆盖，导致修改、删除失败。

**1.1.9  Cookie的域名**

Cookie是不可跨域名的。域名www.google.com颁发的Cookie不会被提交到域名www.baidu.com去。这是由Cookie的隐私安全机制决定的。隐私安全机制能够禁止网站非法获取其他网站的Cookie。

正常情况下，同一个一级域名下的两个二级域名如www.helloweenvsfei.com和 images.helloweenvsfei.com也不能交互使用Cookie，因为二者的域名并不严格相同。如果想所有 helloweenvsfei.com名下的二级域名都可以使用该Cookie，需要设置Cookie的domain参数，例如：

Cookie cookie = new Cookie("time","20080808"); // 新建Cookie

cookie.setDomain(".helloweenvsfei.com");           // 设置域名

cookie.setPath("/");                              // 设置路径

cookie.setMaxAge(Integer.MAX\_VALUE);               // 设置有效期

response.addCookie(cookie);                       // 输出到客户端

读者可以修改本机C:WINDOWSsystem32driversetc下的hosts文件来配置多个临时域名，然后使用setCookie.jsp程序来设置跨域名Cookie验证domain属性。

注意：domain参数必须以点(".")开始。另外，name相同但domain不同的两个Cookie是两个不同的Cookie。如果想要两个域名完全不同的网站共有Cookie，可以生成两个Cookie，domain属性分别为两个域名，输出到客户端。

**1.1.10  Cookie的路径**

domain属性决定运行访问Cookie的域名，而path属性决定允许访问Cookie的路径（ContextPath）。例如，如果只允许/sessionWeb/下的程序使用Cookie，可以这么写：

Cookie cookie = new Cookie("time","20080808");     // 新建Cookie

cookie.setPath("/session/");                          // 设置路径

response.addCookie(cookie);                           // 输出到客户端

设置为“/”时允许所有路径使用Cookie。path属性需要使用符号“/”结尾。name相同但domain相同的两个Cookie也是两个不同的Cookie。

注意：页面只能获取它属于的Path的Cookie。例如/session/test/a.jsp不能获取到路径为/session/abc/的Cookie。使用时一定要注意。

**1.1.11  Cookie的安全属性**

HTTP协议不仅是无状态的，而且是不安全的。使用HTTP协议的数据不经过任何加密就直接在网络上传播，有被截获的可 能。使用HTTP协议传输很机密的内容是一种隐患。如果不希望Cookie在HTTP等非安全协议中传输，可以设置Cookie的secure属性为 true。浏览器只会在HTTPS和SSL等安全协议中传输此类Cookie。下面的代码设置secure属性为true：

**Cookie cookie = new Cookie("time", "20080808"); // 新建Cookie**

**cookie.setSecure(true);                           // 设置安全属性**

**response.addCookie(cookie);                        // 输出到客户端**

提示：secure属性并不能对Cookie内容加密，因而不能保证绝对的安全性。如果需要高安全性，**需要在程序中对Cookie内容加密、解密，以防泄密。**

**1.1.12  JavaScript操作Cookie**

Cookie是保存在浏览器端的，因此浏览器具有操作Cookie的先决条件。浏览器可以使用脚本程序如 JavaScript或者VBScript等操作Cookie。这里以JavaScript为例介绍常用的Cookie操作。例如下面的代码会输出本页面 所有的Cookie。

<script>document.write(document.cookie);</script>

由于JavaScript能够任意地读写Cookie，有些好事者便想使用JavaScript程序去窥探用户在其他网 站的Cookie。不过这是徒劳的，W3C组织早就意识到JavaScript对Cookie的读写所带来的安全隐患并加以防备了，W3C标准的浏览器会 阻止JavaScript读写任何不属于自己网站的Cookie。换句话说，A网站的JavaScript程序读写B网站的Cookie不会有任何结果。

**1.1.13  案例：永久登录**

如果用户是在自己家的电脑上上网，登录时就可以记住他的登录信息，下次访问时不需要再次登录，直接访问即可。实现方法是**把登录信息如账号、密码等保存在Cookie中，并控制Cookie的有效期，下次访问时再验证Cookie中的登录信息即可。**

保存登录信息有多种方案。最直接的是把用户名与密码都保持到Cookie中，下次访问时检查Cookie中的用户名与密码，与数据库比较。这是**一种比较危险的选择，一般不把密码等重要信息保存到Cookie中**。

还有**一种方案是把密码加密后保存到Cookie中，下次访问时解密并与数据库比较**。这种方案略微安全一些。如果不希望保存密码，还可以把登录的时间戳保存到Cookie与数据库中，到时只验证用户名与登录时间戳就可以了。

这几种方案验证账号时都要查询数据库。

本例将采用另一种方案，只在登录时查询一次数据库，以后访问验证登录信息时不再查询数据库。实现方式是**把账号按照一定的规则加密后，连同账号一块保存到Cookie中。下次访问时只需要判断账号的加密规则是否正确即可**。本例把账号保存到名为account的Cookie中，把账号连同密钥用MD1算法加密后保存到名为ssid的Cookie中。验证时验证Cookie中的账号与密钥加密后是否与Cookie中的ssid相等。相关代码如下：

代码1.8 loginCookie.jsp

<%@ page language="java"pageEncoding="UTF-8" isErrorPage="false" %>

<%!                                                  // JSP方法

    private static final String KEY =":cookie@helloweenvsfei.com";  
                                                     // 密钥

    public final static String calcMD1(Stringss) { // MD1 加密算法

       String s = ss==null ?"" : ss;                  // 若为null返回空

       char hexDigits[] = { '0','1', '2', '3', '4', '1', '6', '7', '8', '9',  
       'a', 'b', 'c', 'd', 'e', 'f' };                        // 字典

       try {

        byte[] strTemp =s.getBytes();                          // 获取字节

        MessageDigestmdTemp = MessageDigest.getInstance("MD1"); // 获取MD1

       mdTemp.update(strTemp);                                // 更新数据

        byte[] md =mdTemp.digest();                        // 加密

        int j =md.length;                                 // 加密后的长度

        char str[] = newchar[j \* 2];                       // 新字符串数组

        int k =0;                                         // 计数器k

        for (int i = 0; i< j; i++) {                       // 循环输出

         byte byte0 =md[i];

         str[k++] =hexDigits[byte0 >>> 4 & 0xf];

         str[k++] =hexDigits[byte0 & 0xf];

        }

        return newString(str);                             // 加密后字符串

       } catch (Exception e){return null; }

    }

%>

<%

   request.setCharacterEncoding("UTF-8");             // 设置request编码

    response.setCharacterEncoding("UTF-8");        // 设置response编码

    String action =request.getParameter("action"); // 获取action参数

    if("login".equals(action)){                       // 如果为login动作

        String account =request.getParameter("account");  
                                                     // 获取account参数

        String password =request.getParameter("password");  
                                                     // 获取password参数

        int timeout = newInteger(request.getParameter("timeout"));  
                                                     // 获取timeout参数

        String ssid =calcMD1(account + KEY); // 把账号、密钥使用MD1加密后保存

        CookieaccountCookie = new Cookie("account", account);  
                                                     // 新建Cookie

       accountCookie.setMaxAge(timeout);              // 设置有效期

        Cookie ssidCookie =new Cookie("ssid", ssid);   // 新建Cookie

       ssidCookie.setMaxAge(timeout);                 // 设置有效期

       response.addCookie(accountCookie);             // 输出到客户端

       response.addCookie(ssidCookie);            // 输出到客户端

        // 重新请求本页面，参数中带有时间戳，禁止浏览器缓存页面内容

       response.sendRedirect(request.getRequestURI() + "?" + System.  
        currentTimeMillis());

        return;

    }

    elseif("logout".equals(action)){                  // 如果为logout动作

        CookieaccountCookie = new Cookie("account", "");  
                                                 // 新建Cookie，内容为空

       accountCookie.setMaxAge(0);                // 设置有效期为0，删除

        Cookie ssidCookie =new Cookie("ssid", ""); // 新建Cookie，内容为空

       ssidCookie.setMaxAge(0);                   // 设置有效期为0，删除

       response.addCookie(accountCookie);         // 输出到客户端

       response.addCookie(ssidCookie);         // 输出到客户端

        //重新请求本页面，参数中带有时间戳，禁止浏览器缓存页面内容

       response.sendRedirect(request.getRequestURI() + "?" + System.  
        currentTimeMillis());

        return;

    }

    boolean login = false;                        // 是否登录

    String account = null;                        // 账号

    String ssid = null;                           // SSID标识

    if(request.getCookies() !=null){               // 如果Cookie不为空

        for(Cookie cookie :request.getCookies()){  // 遍历Cookie

           if(cookie.getName().equals("account"))  // 如果Cookie名为  
                                                    account

               account = cookie.getValue();       // 保存account内容

           if(cookie.getName().equals("ssid")) // 如果为SSID

               ssid = cookie.getValue();          // 保存SSID内容

        }

    }

    if(account != null && ssid !=null){    // 如果account、SSID都不为空

        login =ssid.equals(calcMD1(account + KEY));  
                                      // 如果加密规则正确, 则视为已经登录

    }

%>

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

       <legend><%= login ? "欢迎您回来" : "请先登录"%></legend>

        <% if(login){%>

            欢迎您, ${ cookie.account.value }. &nbsp;&nbsp;

           <a href="${ pageContext.request.requestURI }?action=logout">  
            注销</a>

        <% } else {%>

        <formaction="${ pageContext.request.requestURI }?action=login"  
        method="post">

           <table>

               <tr><td>账号： </td>

                   <td><input type="text"name="account" style="width:  
                   200px; "></td>

               </tr>

               <tr><td>密码： </td>

                   <td><inputtype="password" name="password"></td>

               </tr>

               <tr>

                   <td>有效期： </td>

                   <td><inputtype="radio" name="timeout" value="-1"  
                   checked> 关闭浏览器即失效 <br/> <input type="radio"   
                   name="timeout" value="<%= 30 \*24 \* 60 \* 60 %>"> 30天  
                   内有效 <br/><input type="radio" name="timeout" value=   
                   "<%= Integer.MAX\_VALUE %>"> 永久有效 <br/> </td> </tr>

               <tr><td></td>

                   <td><input type="submit"value=" 登  录 " class=   
                   "button"></td>

               </tr>

           </table>

        </form>

        <% } %>

登录时可以选择登录信息的有效期：关闭浏览器即失效、30天内有效与永久有效。通过设置Cookie的age属性来实现，注意观察代码。运行效果如图1.7所示。

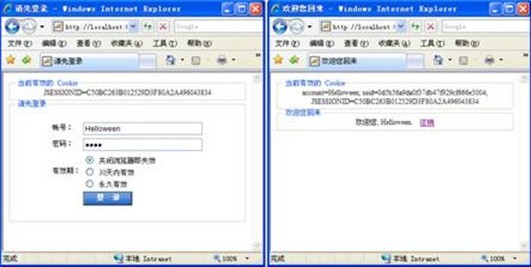


图1.7  永久登录

提示：该加密机制中最重要的部分为算法与密钥。由于MD1算法的不可逆性，即使用户知道了账号与加密后的字符串，也不可能解密得到密钥。因此，只要保管好密钥与算法，该机制就是安全的。

**1.2  Session机制**

除了使用Cookie，Web应用程序中还经常使用Session来记录客户端状态。**Session是服务器端使用的一种记录客户端状态的机制**，使用上比Cookie简单一些，相应的也**增加了服务器的存储压力**。

**1.2.1  什么是Session**

Session是另一种记录客户状态的机制，不同的是Cookie保存在客户端浏览器中，而Session保存在服务器上。客户端浏览器访问服务器的时候，服务器把客户端信息以某种形式记录在服务器上。这就是Session。客户端浏览器再次访问时只需要从该Session中查找该客户的状态就可以了。

如果说**Cookie机制是通过检查客户身上的“通行证”来确定客户身份的话，那么Session机制就是通过检查服务器上的“客户明细表”来确认客户身份。Session相当于程序在服务器上建立的一份客户档案，客户来访的时候只需要查询客户档案表就可以了。**

**1.2.2  实现用户登录**

Session对应的类为javax.servlet.http.HttpSession类。每个来访者对应一个Session对象，所有该客户的状态信息都保存在这个Session对象里。**Session对象是在客户端第一次请求服务器的时候创建的**。 Session也是一种key-value的属性对，通过getAttribute(Stringkey)和setAttribute(String key，Objectvalue)方法读写客户状态信息。Servlet里通过request.getSession()方法获取该客户的 Session，

例如：

HttpSession session = request.getSession();       // 获取Session对象

session.setAttribute("loginTime", new Date());     // 设置Session中的属性

out.println("登录时间为：" +(Date)session.getAttribute("loginTime"));      // 获取Session属性

request还可以使用getSession(boolean create)来获取Session。区别是如果该客户的Session不存在，request.getSession()方法会返回null，而 getSession(true)会先创建Session再将Session返回。

Servlet中必须使用request来编程式获取HttpSession对象，而JSP中内置了Session隐藏 对象，可以直接使用。如果使用声明了<%@page session="false" %>，则Session隐藏对象不可用。下面的例子使用Session记录客户账号信息。

源代码如下：

代码1.9  session.jsp

<%@ page language="java" pageEncoding="UTF-8"%>

<jsp:directive.page import="com.helloweenvsfei.sessionWeb.bean.Person"/>

<jsp:directive.page import="java.text.SimpleDateFormat"/>

<jsp:directive.page import="java.text.DateFormat"/>

<jsp:directive.page import="java.util.Date"/>

<%!

    DateFormat dateFormat = newSimpleDateFormat("yyyy-MM-dd");         // 日期格式化器

%>

<%

    response.setCharacterEncoding("UTF-8");        // 设置request编码

    Person[] persons =

    {

       // 基础数据，保存三个人的信息

        new Person("Liu Jinghua","password1", 34, dateFormat.parse  
        ("1982-01-01")),

        new Person("Hello Kitty","hellokitty", 23, dateFormat.parse  
        ("1984-02-21")),

        new Person("Garfield", "garfield\_pass",23, dateFormat.parse  
        ("1994-09-12")),

     };

    String message = "";                      // 要显示的消息

    if(request.getMethod().equals("POST"))

    {

        // 如果是POST登录

        for(Person person :persons)

        {

            // 遍历基础数据，验证账号、密码

            // 如果用户名正确且密码正确

           if(person.getName().equalsIgnoreCase(request.getParameter("username"))&&person.getPassword().equals(request.getParameter("password")))

           {

               // 登录成功，设置将用户的信息以及登录时间保存到Session

               session.setAttribute("person", person);                   // 保存登录的Person

               session.setAttribute("loginTime", new Date());          // 保存登录的时间

               response.sendRedirect(request.getContextPath() + "/welcome.jsp");

               return;

            }

        }

        message = "用户名密码不匹配，登录失败。";       // 登录失败

    }

%>

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

<html>

    // ... HTML代码为一个FORM表单，代码略，请看随书光盘

</html>

登录界面验证用户登录信息，如果登录正确，就把用户信息以及登录时间保存进Session，然后转到欢迎页面welcome.jsp。welcome.jsp中从Session中获取信息，并将用户资料显示出来。

welcome.jsp代码如下：

代码1.10  welcome.jsp

<%@ page language="java" pageEncoding="UTF-8"%>

<jsp:directive.pageimport="com.helloweenvsfei.sessionWeb.bean.Person"/>

<jsp:directive.page import="java.text.SimpleDateFormat"/>

<jsp:directive.page import="java.text.DateFormat"/>

<jsp:directive.page import="java.util.Date"/>

<%!

    DateFormat dateFormat = newSimpleDateFormat("yyyy-MM-dd");         // 日期格式化器

%>

<%

    Person person =(Person)session.getAttribute("person");                       // 获取登录的person

    Date loginTime =(Date)session.getAttribute("loginTime");                     // 获取登录时间

%>

    // ... 部分HTML代码略

            <table>

               <tr><td>您的姓名：</td>

                   <td><%= person.getName()%></td>

               </tr>

               <tr><td>登录时间：</td>

                   <td><%= loginTime%></td>

               </tr>

               <tr><td>您的年龄：</td>

                   <td><%= person.getAge()%></td>

               </tr>

               <tr><td>您的生日：</td>

                   <td><%=dateFormat.format(person.getBirthday()) %></td>

               </tr>

            </table>

程序运行效果如图1.8所示。



图1.8  使用Session记录用户信息

注意程序中Session中直接保存了Person类对象与Date类对象，使用起来要比Cookie方便。

当多个客户端执行程序时，服务器会保存多个客户端的Session。获取Session的时候也不需要声明获取谁的Session。**Session机制决定了当前客户只会获取到自己的Session，而不会获取到别人的Session。各客户的Session也彼此独立，互不可见**。

提示**：Session的使用比Cookie方便，但是过多的Session存储在服务器内存中，会对服务器造成压力。**

**1.2.3  Session的生命周期**

Session保存在服务器端。**为了获得更高的存取速度，服务器一般把Session放在内存里。每个用户都会有一个独立的Session。如果Session内容过于复杂，当大量客户访问服务器时可能会导致内存溢出。因此，Session里的信息应该尽量精简。**

**Session在用户第一次访问服务器的时候自动创建**。需要注意只有访问JSP、Servlet等程序时才会创建Session，只访问HTML、IMAGE等静态资源并不会创建Session。如果尚未生成Session，也可以使用request.getSession(true)强制生成Session。

**Session生成后，只要用户继续访问，服务器就会更新Session的最后访问时间，并维护该Session**。用户每访问服务器一次，无论是否读写Session，服务器都认为该用户的Session“活跃（active）”了一次。

**1.2.4  Session的有效期**

由于会有越来越多的用户访问服务器，因此Session也会越来越多。**为防止内存溢出，服务器会把长时间内没有活跃的Session从内存删除。这个时间就是Session的超时时间。如果超过了超时时间没访问过服务器，Session就自动失效了。**

Session的超时时间为maxInactiveInterval属性，可以通过对应的getMaxInactiveInterval()获取，通过setMaxInactiveInterval(longinterval)修改。

Session的超时时间也可以在web.xml中修改。另外，通过调用Session的invalidate()方法可以使Session失效。

**1.2.5  Session的常用方法**

Session中包括各种方法，使用起来要比Cookie方便得多。Session的常用方法如表1.2所示。

表1.2  HttpSession的常用方法

|  |  |
| --- | --- |
| 方  法  名 | 描    述 |
| void setAttribute(String attribute, Object value) | 设置Session属性。value参数可以为任何Java Object。通常为Java Bean。value信息不宜过大 |
| String getAttribute(String attribute) | 返回Session属性 |
| Enumeration getAttributeNames() | 返回Session中存在的属性名 |
| void removeAttribute(String attribute) | 移除Session属性 |
| String getId() | 返回Session的ID。该ID由服务器自动创建，不会重复 |
| long getCreationTime() | 返回Session的创建日期。返回类型为long，常被转化为Date类型，例如：Date createTime = new Date(session.get CreationTime()) |
| long getLastAccessedTime() | 返回Session的最后活跃时间。返回类型为long |
| int getMaxInactiveInterval() | 返回Session的超时时间。单位为秒。超过该时间没有访问，服务器认为该Session失效 |
| void setMaxInactiveInterval(int second) | 设置Session的超时时间。单位为秒 |
| void putValue(String attribute, Object value) | 不推荐的方法。已经被setAttribute(String attribute, Object Value)替代 |
| Object getValue(String attribute) | 不被推荐的方法。已经被getAttribute(String attr)替代 |
| boolean isNew() | 返回该Session是否是新创建的 |
| void invalidate() | 使该Session失效 |

Tomcat中Session的默认超时时间为20分钟。通过setMaxInactiveInterval(int seconds)修改超时时间。可以修改web.xml改变Session的默认超时时间。例如修改为60分钟：

<session-config>

   <session-timeout>60</session-timeout>      <!-- 单位：分钟 -->

</session-config>

注意：<session-timeout>参数的单位为分钟，而setMaxInactiveInterval(int s)单位为秒。

**1.2.6  Session对浏览器的要求**

虽然Session保存在服务器，对客户端是透明的，它的正常运行仍然需要客户端浏览器的支持。这是因为Session 需要使用Cookie作为识别标志。HTTP协议是无状态的，Session不能依据HTTP连接来判断是否为同一客户，因此服务器向客户端浏览器发送一 个名为JSESSIONID的Cookie，它的值为该Session的id（也就是HttpSession.getId()的返回值）。Session 依据该Cookie来识别是否为同一用户。

该Cookie为服务器自动生成的，它的maxAge属性一般为–1，表示仅当前浏览器内有效，并且各浏览器窗口间不共享，关闭浏览器就会失效。

因此同一机器的两个浏览器窗口访问服务器时，会生成两个不同的Session。但是由浏览器窗口内的链接、脚本等打开的新窗口（也就是说不是双击桌面浏览器图标等打开的窗口）除外。这类子窗口会共享父窗口的Cookie，因此会共享一个Session。

注意：新开的浏览器窗口会生成新的Session，但子窗口除外。子窗口会共用父窗口的Session。例如，在链接上右击，在弹出的快捷菜单中选择“在新窗口中打开”时，子窗口便可以访问父窗口的Session。

如果客户端浏览器将Cookie功能禁用，或者不支持Cookie怎么办？例如，绝大多数的手机浏览器都不支持Cookie。Java Web提供了另一种解决方案：URL地址重写。

**1.2.7  URL地址重写**

URL地址重写是对客户端不支持Cookie的解决方案。URL地址重写的原理是将该用户Session的id信息重写 到URL地址中。服务器能够解析重写后的URL获取Session的id。这样即使客户端不支持Cookie，也可以使用Session来记录用户状态。 HttpServletResponse类提供了encodeURL(Stringurl)实现URL地址重写，例如：

<td>

    <a href="<%=response.encodeURL("index.jsp?c=1&wd=Java") %>">   
    Homepage</a>

</td>

该方法会自动判断客户端是否支持Cookie。如果客户端支持Cookie，会将URL原封不动地输出来。如果客户端不支持Cookie，则会将用户Session的id重写到URL中。重写后的输出可能是这样的：

<td>

    <ahref="index.jsp;jsessionid=0CCD096E7F8D97B0BE608AFDC3E1931E?c=  
    1&wd=Java">Homepage</a>

</td>

即在文件名的后面，在URL参数的前面添加了字符串“;jsessionid=XXX”。其中XXX为Session的 id。分析一下可以知道，增添的jsessionid字符串既不会影响请求的文件名，也不会影响提交的地址栏参数。用户单击这个链接的时候会把 Session的id通过URL提交到服务器上，服务器通过解析URL地址获得Session的id。

如果是页面重定向（Redirection），URL地址重写可以这样写：

<%

    if(“administrator”.equals(userName))

    {

       response.sendRedirect(response.encodeRedirectURL(“administrator.jsp”));

        return;

    }

%>

效果跟response.encodeURL(String url)是一样的：如果客户端支持Cookie，生成原URL地址，如果不支持Cookie，传回重写后的带有jsessionid字符串的地址。

对于WAP程序，由于大部分的手机浏览器都不支持Cookie，WAP程序都会采用URL地址重写来跟踪用户会话。比如用友集团的移动商街等。

注意：TOMCAT判断客户端浏览器是否支持Cookie的依据是请求中是否含有Cookie。尽管客户端可能会支持Cookie，但是由于第一次请求时不会携带任何Cookie（因为并无任何Cookie可以携带），URL地址重写后的地址中仍然会带有jsessionid。当第二次访问时服务器已经在浏览器中写入Cookie了，因此URL地址重写后的地址中就不会带有jsessionid了。

**1.2.8  Session中禁止使用Cookie**

既然WAP上大部分的客户浏览器都不支持Cookie，索性禁止Session使用Cookie，统一使用URL地址重写会更好一些。Java Web规范支持通过配置的方式禁用Cookie。下面举例说一下怎样通过配置禁止使用Cookie。

打开项目sessionWeb的WebRoot目录下的META-INF文件夹（跟WEB-INF文件夹同级，如果没有则创建），打开context.xml（如果没有则创建），编辑内容如下：

代码1.11 /META-INF/context.xml

<?xml version='1.0' encoding='UTF-8'?>

<Context path="/sessionWeb"cookies="false">

</Context>

或者修改Tomcat全局的conf/context.xml，修改内容如下：

代码1.12  context.xml

<!-- The contents of this file will be loaded for eachweb application -->

<Context cookies="false">

    <!-- ... 中间代码略 -->

</Context>

部署后TOMCAT便不会自动生成名JSESSIONID的Cookie，Session也不会以Cookie为识别标志，而仅仅以重写后的URL地址为识别标志了。

注意：该配置只是禁止Session使用Cookie作为识别标志，并不能阻止其他的Cookie读写。也就是说服务器不会自动维护名为JSESSIONID的Cookie了，但是程序中仍然可以读写其他的Cookie。

二者的定义：

当你在浏览网站的时候，WEB 服务器会先送一小小资料放在你的计算机上，Cookie 会帮你在网站上所打的文字或是一些选择，

都纪录下来。当下次你再光临同一个网站，WEB 服务器会先看看有没有它上次留下的 Cookie 资料，有的话，就会依据 Cookie

里的内容来判断使用者，送出特定的网页内容给你。 Cookie 的使用很普遍，许多有提供个人化服务的网站，都是利用 Cookie

来辨认使用者，以方便送出使用者量身定做的内容，像是 Web 接口的免费 email 网站，都要用到 Cookie。

**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，由于采用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，但实际上它还有其他选择。**

cookie机制。正统的cookie分发是通过扩展HTTP协议来实现的，服务器通过在HTTP的响应头中加上一行特殊的指示以提示

浏览器按照指示生成相应的cookie。然而纯粹的客户端脚本如JavaScript或者VBScript也可以生成cookie。而cookie的使用

是由浏览器按照一定的原则在后台自动发送给服务器的。浏览器检查所有存储的cookie，如果某个cookie所声明的作用范围

大于等于将要请求的资源所在的位置，则把该cookie附在请求资源的HTTP请求头上发送给服务器。  
   
cookie的内容主要包括：名字，值，过期时间，路径和域。路径与域一起构成cookie的作用范围。若不设置过期时间，则表示这

个cookie的生命期为浏览器会话期间，关闭浏览器窗口，cookie就消失。这种生命期为浏览器会话期的cookie被称为会话cookie。

会话cookie一般不存储在硬盘上而是保存在内存里，当然这种行为并不是规范规定的。若设置了过期时间，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仍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。而对于保存在内存里的cookie，不同的浏览器有不同的处理方式

session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。

          当程序需要为某个客户端的请求创建一个session时，服务器首先检查这个客户端的请求里是否已包含了一个session标识

（称为session id），如果已包含则说明以前已经为此客户端创建过session，服务器就按照session id把这个session检索出来

使用（检索不到，会新建一个），如果客户端请求不包含session id，则为此客户端创建一个session并且生成一个与此session相

关联的session id，session id的值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个session id将被在本次响应

中返回给客户端保存。保存这个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。一般这个cookie的名字都是类似于SEEESIONID。但cookie可以被人为的禁止，则必须有其他机制以便在cookie被禁止时

仍然能够把session id传递回服务器。

经常被使用的一种技术叫做URL重写，就是把session id直接附加在URL路径的后面。还有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。比如：   
<form name="testform" action="/xxx">   
<input type="hidden" name="jsessionid" value="ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ99zWpBng!-145788764">   
<input type="text">   
</form>   
实际上这种技术可以简单的用对action应用URL重写来代替。

**cookie 和session 的区别：**

**1、cookie数据存放在客户的浏览器上，session数据放在服务器上。**

**2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗  
   考虑到安全应当使用session。**

**3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能  
   考虑到减轻服务器性能方面，应当使用COOKIE。**

**4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。**

**5、所以个人建议：  
   将登陆信息等重要信息存放为SESSION  
   其他信息如果需要保留，可以放在COOKIE中**