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。

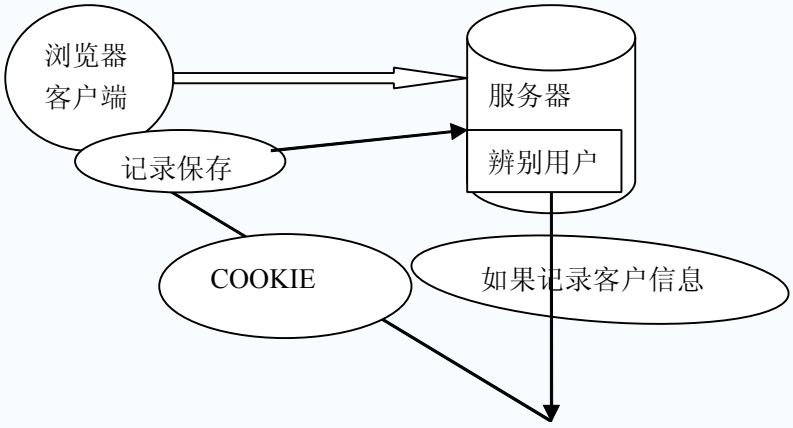
Cookie 就是这样的一种机制。它可以弥补 HTTP 协议无状态的不足。在 Session 出现之前，基本上所有的网站都采用 Cookie 来跟踪会话。

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。

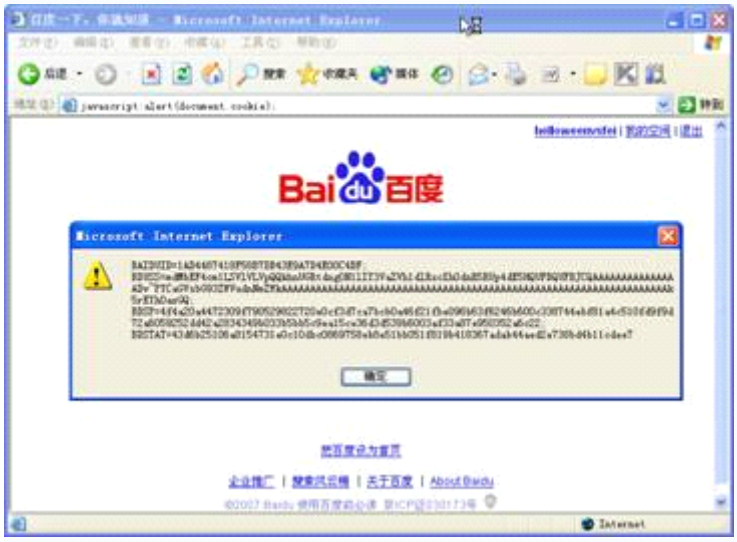


图 1.1　Baidu 网站颁发的 Cookie

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(Cookie cookie)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:\WINDOWS\system32\drivers\etc 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(String ss) { // 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
       MessageDigest mdTemp = 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[] = new char[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 new String(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 = new Integer(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

        Cookie accountCookie = 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;
    }
    else if("logout".equals(action)){                   // 如果为 logout 动作

        Cookie accountCookie = 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 4.01 Transitional//EN">
        <legend><%= login ? "欢迎您回来" : "请先登录" %></legend>
        <% if(login){ %>
            欢迎您, ${ cookie.account.value }.   
            <a href="${ pageContext.request.requestURI }?action=logout">
            注销</a>
        <% } else { %>
        <form action="${ 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><input type="password" name="password"></td>
                </tr>
                <tr>
                    <td>有效期：  </td>
                    <td><input type="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。

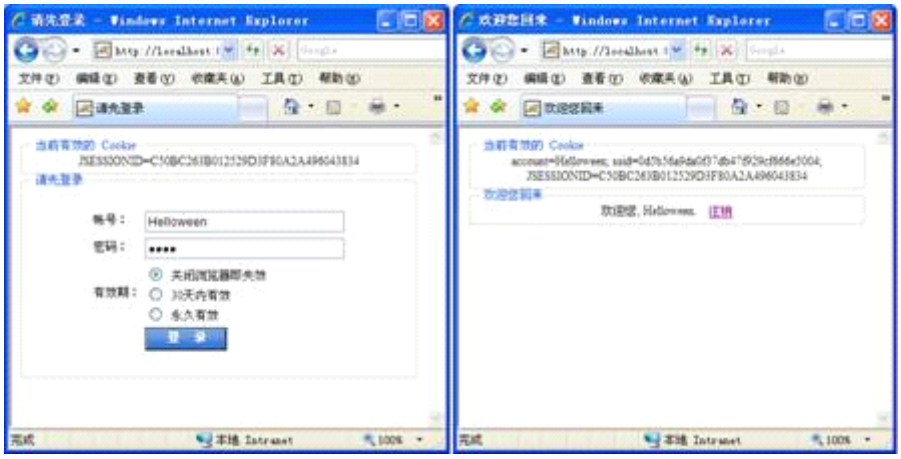


图 1.7　永久登录

%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(String key)和 setAttribute(String key，Object value)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 = new SimpleDateFormat("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 4.01 Transitional//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.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 = new SimpleDateFormat("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。

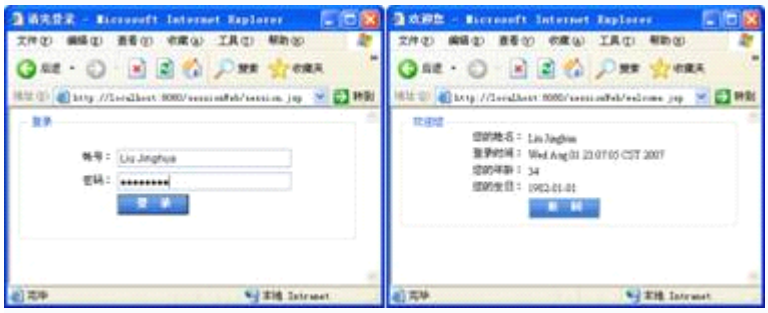


图 1.8　使用 Session 记录用户信息

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(long interval)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(String url)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>
  <a href="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 each web 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。