Cookie/Session 机制详解

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（后期发展的技术 token）。Cookie 通过在客户端（浏览器本地电脑）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。

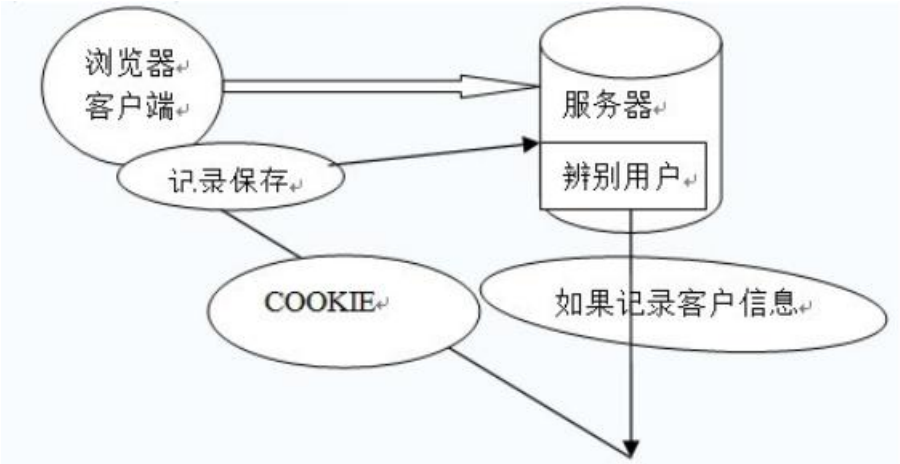
Cookie 就是这样的一种机制。它可以弥补 HTTP 协议无状态的不足。在 Session 出现之前，基本上所有的网站都采用 Cookie 来跟踪会话。

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。

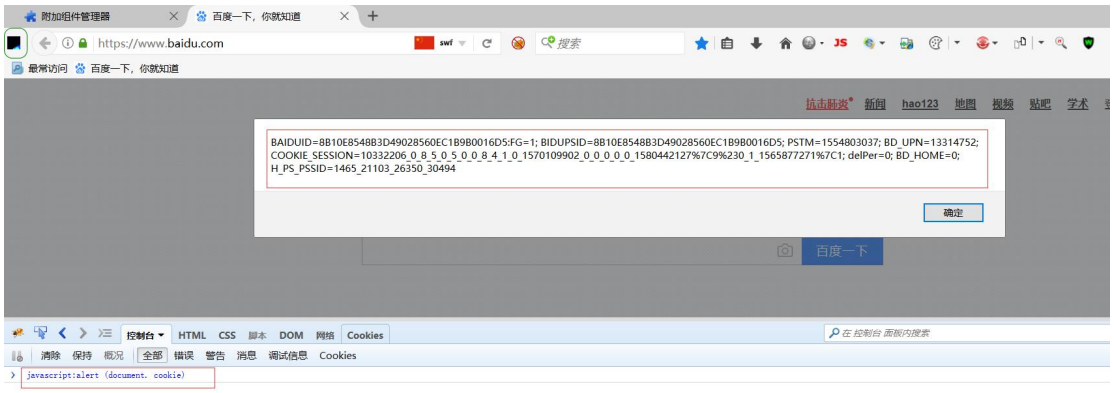


图 1.1　Baidu 网站颁发的 Cookie

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 记录的就是用户的身份，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。

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。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

1.2 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。

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。

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。**

1.2.5 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。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。

1.2.6 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。

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。

先谈 cookie

网络传输基于的 Http 协议，是无状态的协议，即每次连接断开后再去连接，服务器是无法判断此次连接的客户端是谁。

如果每次数据传输都需要进行连接和断开，那造成的开销是很巨大的。

为了解决这个问题，cookie 就应运而生，当用户登陆成功，服务器会在返回响应数据的同时也携带着 cookie 给到客户端，

之后客户端每次发起请求只要携带着这个 cookie，那就免去登录的步骤。cookie 是保存在客户端的数据。

这确实极大改善了网络传输的效率。当时由于 cookie 是保存在浏览器客户端的，所以也很容易被提取，这在安全方面存在隐患。

再谈 session

session 的实现需要依赖于 cookie，其本质也是通过以 cookie 的方式向客户端发送随机字符串，每次客户端发起请求时只要携带

该随机字符串便可很容易进行验证。下面大概写下整个 session 过程的原理。

1. 用户发出登录请求
2. 判断账户密码是否正确
3. 如正确，则返回数据并在cookie中写随机字符串（sessionID），并且在服务端以{随机字符串：{'k':'v'}}的形式存储用户相关数据
4. 下次同个客户发送请求，携带cookie（包含sessionID）
5. 服务端会判断cookie是否包含sessionID，如有再去服务器内存中查询该sessionID 是否有对应的数据，如果有，则说明是登录过的用户，返回数据。

session 是保存在服务端的数据。

由此可见，session 是不会向客户端发送敏感信息的，随机字符串即使被人窃取，也无法简单在客户端被恶意行为伪造请求。

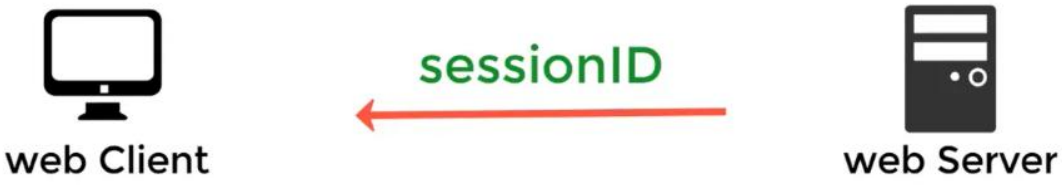
注意：

1、cookie 存储的数量和字符数量都有限制，只能存储几十个，不超 4096 左右个字符（4KB）。

Session 实现过程：

客户端浏览器访问网站的时候

服务器会向客户浏览器发送一个每个用户特有的会话编号 sessionID，让他进入到 cookie 里。



服务器同时也把 sessionID 和对应的用户信息、用户操作记录在服务器上，这些记录就是 session。

客户端浏览器再次访问时，会发送 cookie 给服务器，其中就包含 sessionID。

服务器从 cookie 里找到 sessionID，再根据 sessionID 找到以前记录的用户信息就可以知道他之前操控些、访问过哪里。

session 保存在服务器端比较安全，但是可能需要记录千百万用户的信息，对服务器的存储压力很大，所以我们应该有选择的合理使用 cookie 和 session。

问题一、关于 cookie 跨域使用

domain 表示的是 cookie 所在的域，默认为请求的地址，如网址为 www.jb51.net/test/test.aspx，那么 domain 默认为 www.jb51.net。而跨域访问，如域 A 为 t1.test.com,域 B 为 t2.test.com,那么在域 A 生产一个令域 A 和域 B 都能访问的 cookie 就要将该 cookie 的 domain 设置为.test.com；如果要在域 A 生产一个令域 A 不能访问而域

B 能访问的 cookie 就要将该 cookie 的 domain 设置为 t2.test.com。

问题二、PHPSESSID 和 Jsessionid

服务器的 php 端使用 session_start()后，$_COOKIE[session_name()]就可以取到 session_id 的具体值了。因此，页面返回前端后，session_id 的值就自动存在 cookie 中了，下次浏览器再往服务端发请求时就会在 cookie 头中自动携带该值。php 设置的默认 cookie 的名字是：PHPSESSID，可以在 php.ini 修改此名字。

Jsessionid 只是 tomcat 的对 sessionid 的叫法，其实就是 sessionid；在其它的容器也许就不叫 jsessionid 了。

所谓 session 可以这样理解：当与服务器端进行会话时，比如说登陆成功后，服务器端会为用户开壁一块内存区间，用以存放用户这次会话的一些内容，比如说用户名之类的。那么就需要一个东西来标志这个内存区间是你的而不是别人的，这个东西就是 session id(jsessionid 只是 tomcat 中对 session id 的叫法，在其它容器里面，不一定就是叫 jsessionid 了。),而这个内存区间你可以理解为 session。

然后，服务器会将这个 session id 发回给你的浏览器，放入你的浏览器的 cookies 中（这个 cookies 是内存 cookies，跟一般的不一样，它会随着浏览器的关闭而消失）。

之后，只有你浏览器没有关闭，你每向服务器发请求，服务器就会从你发送过来的 cookies 中拿出这个 session id,然后根据这个 session id 到相应的内存中取你之前存放的数据。

但是，如果你退出登陆了，服务器会清掉属于你的内存区域，所以你再登的话，会产生一个新的 session 了。

这是一个保险措施 因为 Session 默认是需要 Cookie 支持的,但有些客户浏览器是关闭 Cookie 的【而 jsessionid 是存储在 Cookie 中的，如果禁用 Cookie 的话，也就是说服务器那边得不到 jsessionid，这样也就没法根据 jsessionid 获得对应的 session 了，获得不了 session 就得不到 session 中存储的数据了。】这个时候就需要在 URL 中指定服务器上的 session 标识,也就是类似于“jsessionid=5F4771183629C9834F8382E23BE13C4C” 这种格式。

问题三、关于 cookie 被禁用的 url 重写问题（java EE servlet 环境）

我们都知道 session 的实现主要两种方式：cookie 与 url 重写，而 cookie 是首选(默认)的方式，因为各种现代浏览器都默认开通 cookie 功能，但是每种浏览器也都有允许 cookie 失效的设置。 由于浏览器默认启动 cookie 功能，而且普通客户一般都不会取消 cookie 功能。久而久之，我们写代码的时候，也就不会在意 session 的具体实现，其实这里面还是有很多值得注意的地方，尤其在用户取消 cookie 功能的情况下。

servlet session 实现与接口简要介绍： servlet 规范规定实现 session 的 cookie 名称强制为 jsessionid(在 servlet 3.0 可以自定义了)，在浏览器第一次请求的时候，服务器产生一个唯一的 id，并把这个 id 设置给一个名为 jsessionid 的 cookie，然后再通过 response 的 addcookie，输出到浏览器端。

下面基于 http 协议解释一下：

浏览器第一次请求：

GET /cxt/index.do HTTP/1.1 ...

由于是第一次请求，所以没有 cookie 要推给服务器

服务器返回：

HTTP/1.1 200 OK

Set-Cookie: JSESSIONID=3EF0AEC40F2C6C30A0580844C0E6B2E8;

Path=/cxt

...

由于服务器发现浏览器没提供任何 cookie,服务器不知道浏览器未提供 cookie 的原因：可能是 cookie 功能取消了，也可能是第一次访问。所以服务器生成一个名为 jsessionid 的 cookie，用 Set-Cookie 来把 cookie 推给浏览器；并且，服务器的 servlet 在生成 html 页面的时候需要用 response.encodeURL 方法来编码 url，该方法其实就是用来实现 url 重写功能的，这是因为浏览器可能是因为取消 cookie 功能，而未提供 cookie 的。服务器为了确保下次提交成功，必须确保生成给浏览器端的 url 带有 jsessionid。

若 cookie 功能没取消，则浏览器发起第二次请求时：

POST /cxt/login.do HTTP/1.1

Cookie: JSESSIONID=3EF0AEC40F2C6C30A0580844C0E6B2E8;

...

浏览器在下一次请求的时候用 http 的 Cookie 属性把当前 domain 的 Cookie 都推给服务器，来表明自己的身份。这次，服务器知道浏览器支持 cookie 功能，servlet 不需要再使用 reponse.encodeURL 来编码 url 了。若浏览器 cookie 功能取消，则浏览器请求内容为

POST /cxt/login.do?jsessionid=3EF0AEC40F2C6C30A0580844C0E6B2E8 HTTP/1.1

...

服务器在接受到上述内容时，通过 url 后面的 jsessionid 参数知道这个请求与上一次请求是同一个 session（会话）。往后的操作也需要在 url 后面保持 jsessionid 的拼接。