**HTTP Session**

一、淺析

HTTP協議(http://www.w3.org/Protocols/)是“一次性單向”協議。服務端不能主動連接客戶端，只能被動等待並答复客戶端請求。客戶端連接服務端，發出一個HTTP Request，服務端處理請求，並且返回一個HTTP Response給客戶端，本次HTTP Request-Response Cycle結束。HTTP協議並不支持服務端保存客戶端的狀態信息。於是Web Server中引入了session的概念，用來保存客戶端的狀態信息。

假設Web Server是一個商場的存包處，HTTP Request是一個顧客，第一次來到存包處，管理員把顧客的物品存放在某一個櫃子裡面(這個櫃子就相當於Session)，然後把一個號碼牌交給這個顧客，作為取包憑證(這個號碼牌就是Session ID)。顧客(HTTP Request)下一次來的時候，就要把號碼牌(Session ID)交給存包處(Web Server)的管理員。管理員根據號碼牌(Session ID)找到相應的櫃子(Session)，根據顧客(HTTP Request)的請求，Web Server可以取出、更換、添加櫃子(Session)中的物品，Web Server也可以讓顧客(HTTP Request)的號碼牌和號碼牌對應的櫃子(Session)失效。顧客(HTTP Request)的忘性很大，管理員在顧客回去的時候(HTTP Response)都要重新提醒顧客記住自己的號碼牌(Session ID)。這樣，顧客(HTTP Request)下次來的時候，就又帶著號碼牌回來了。Session ID實際上是在客戶端和服務端間通過HTTP Request和HTTP Response傳來傳去的。號碼牌(Session ID)必須包含在HTTP Request裡面。關於HTTP Request的具體格式，請參見HTTP協議（http://www.w3.org/Protocols/）。這裡只做一個簡單的介紹。

在Java Web Server(即Servlet/JSP Server)中，Session ID用jsessionid表示(請參見Servlet規範)。

HTTP Request一般由3部分組成：

(1) Request Line。這一行由HTTP Method(如GET或POST)、URL、和HTTP版本號組成。例如，GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1

GET http://www.google.com/search?q=Tomcat HTTP/1.1

POST http://www.google.com/search HTTP/1.1

GET http://www.somsite.com/menu.do;jsessionid=1001 HTTP/1.1

(2) Request Headers。定義了重要的頭部信息，如瀏覽器的種類，語言，類型。Request Headers中還可包括Cookie的定義。例如：

User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)

Accept-Language: en-us

Cookie: jsessionid=1001

(3) Message Body。如果HTTP Method是GET，那麼Message Body為空。如果HTTP Method是POST，說明這個HTTP Request是submit一個HTML Form的結果，那麼Message Body為HTML Form裡面定義的Input屬性。例如，

user=guest

password=guest

jsessionid=1001

如果把HTML Form元素的Method屬性改為GET。那麼，Message Body為空，所有的Input屬性都會加在URL的後面。你在瀏覽器的URL地址欄中會看到這些屬性，類似於

http://www.somesite/login.do?user=guest&password=guest&jsessionid=1001

從理論上來說，這3個部分(Request URL，Cookie Header, Message Body)都可以用來存放Session ID。由於Message Body方法必須需要一個包含Session ID的HTML Form，所以這種方法不通用。

一般用來實現Session的方法有兩種：

1. URL重寫。Web Server返回Response時，檢查頁面中所有的URL，包括所有的連接，和HTML Form的Action屬性，在URL後面加上“;jsessionid=XXX”。下一次，用戶訪問這個頁面中的URL。jsessionid就會傳回到Web Server。
2. Cookie。如果客戶端支持Cookie，Web Server在返回Response的時候，在Response的Header部分，加入一個“set-cookie: jsessionid=XXXX”header屬性，把jsessionid放在Cookie里傳到客戶端。客戶端會把Cookie存放在本地文件裡，下一次訪問Web Server的時候，再把Cookie的信息放到HTTP Request的“Cookie”header屬性裡面，這樣jsessionid就隨著HTTP Request返回給Web Server。

**cookie和session機制之間的區別與聯繫**

具體來說cookie機制採用的是在客戶端保持狀態的方案。它是在用戶端的會話狀態的存貯機制，需要用戶打開客戶端的cookie支持。cookie的作用就是為了解決HTTP協議無狀態的缺陷所作的努力。而session機制採用的是一種在客戶端與服務器之間保持狀態的解決方案。由於採用服務器端保持狀態的方案在客戶端也需要保存一個標識，所以session機制可能需要藉助於cookie機制來達到保存標識的目的。而session提供了方便管理全局變量的方式。

session是針對每一個用戶的，變量的值保存在服務器上，用一個sessionID來區分是哪個用戶session變量，這個值是通過用戶的瀏覽器在訪問的時候返回給服務器，當客戶禁用cookie時，這個值也可能設置為由get來返回給服務器。

就安全性來說：當你訪問一個使用session的站點，同時在自己機子上建立一個cookie，建議在服務器端的SESSION機制更安全些。因為它不會任意讀取客戶存儲的信息。正統的cookie分發是通過擴展HTTP協議來實現的，服務器通過在HTTP的響應頭中加上一行特殊的指示以提示瀏覽器按照指示生成相應的cookie。從網絡服務器觀點看所有HTTP請求都獨立於先前請求。就是說每一個HTTP響應完全依賴於相應請求中包含的信息狀態管理機制克服了HTTP的一些限制並允許網絡客戶端及服務器端維護請求間的關係。在這種關係維持的期間叫做會話(session)。Cookies是服務器在本地機器上存儲的小段文本並隨每一個請求發送至同一個服務器。網絡服務器用HTTP頭向客戶端發送cookies，在客戶終端，瀏覽器解析這些cookies並將它們保存為一個本地文件，它會自動將同一服務器的任何請求縛上這些cookies。

**理解session機制**

session機制是一種服務器端的機制，服務器使用一種類似於散列表的結構(也可能就是使用散列表)來保存信息。當程序需要為某個客戶端的請求創建一個session的時候，服務器首先檢查這個客戶端的請求裡是否已包含了一個session標識-稱為session id，如果已包含一個session id，則此客戶已經端創建過session，服務器就按照session id把這個session檢索出來使用(如果檢索不到，可能會新建一個)，如果客戶端請求不包含session id，則為此客戶端創建一個session並生成一個與此session相關聯的session id，session id的值應該是一個不重複，又不易被找到規律仿造的字符串，這個session id將被在本次響應中返回給客戶端保存。

保存這個session id的方式可以採用cookie，這樣在交互過程中瀏覽器可以自動的按照規則把這個標識發揮給服務器。一般這個cookie的名字都是類似於SEEESIONID，而。比如weblogic對於web應用程序生成的cookie，JSESSIONID= ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ99zWpBng!-145788764，它的名字就是JSESSIONID。由於cookie可被人為禁止，必須有其他機制以便在cookie被禁止時仍能夠把session id傳遞回服務器。常被用的技術叫做URL重寫，把session id直接附加在URL路徑後面，附加方式是作為URL路徑的附加​​信息，表現形式為http://... ../xxx;jsessionid= ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ99zWpBng!-145788764 另一種是作為查詢字符串附加在URL後面，表現形式為http://...../xxx?jsessionid=ByOK3vjFD75aPnrF7C2HmdnV6QZcEbzWoWiBYEnLerjQ99zWpBng!-145788764 這兩種方式對於用戶來說是沒有區別的，只是服務器在解析時處理的方式不同，第一種方式有利於把session id的信息和正常程序參數區分開來。為了在整個交互過程中始終保持狀態，在每個客戶端可能請求的路徑後面都包含這個session id。

在談論session機制的時候，常常聽到這樣一種誤解“只要關閉瀏覽器，session就消失了”。其實可以想像一下會員卡的例子，除非顧客主動對店家提出銷卡，否則店家絕對不會輕易刪除顧客的資料。對session來說也是一樣的，除非程序通知服務器刪除一個session，否則服務器會一直保留，程序一般都是在用戶做log off的時候發個指令去刪除session。然而瀏覽器從來不會主動在關閉之前通知服務器它將要關閉，因此服務器根本不會有機會知道瀏覽器已經關閉，之所以會有這種錯覺，是大部分session機制都使用會話cookie來保存session id，而關閉瀏覽器後這個session id就消失了，再次連接服務器時也就無法找到原來的session。如果服務器設置的cookie被保存到硬盤上，或者使用某種手段改寫瀏覽器發出的HTTP請求頭，把原來的session id發送給服務器，則再次打開瀏覽器仍然能夠找到原來的session。恰恰是由於關閉瀏覽器不會導致session被刪除，迫使服務器為seesion設置了一個失效時間，當距離客戶端上一次使用session的時間超過這個失效時間時，服務器就可以認為客戶端已經停止了活動，才會把session刪除以節省存儲空間。

**由JSESSIONID談cookie與SESSION的區別和聯繫**

在一些投票之類的場合，我們往往因為公平的原則要求每人只能投一票，在一些WEB開發中也有類似的情況，這時候我們通常會使用COOKIE來實現，例如如下的代碼：

< % cookie[]cookies = request.getCookies();

*if (cookies.lenght == 0 || cookies == null) doStuffForNewbie();//沒有訪問過*

*else doStuffForReturnVisitor(); //已經訪問過了*

% >

這是很淺顯易懂的道理，檢測COOKIE的存在，如果存在說明已經運行過寫入COOKIE的代碼了，然而運行以上的代碼後，無論何時結果都是執行doStuffForReturnVisitor()，通過控制面板-Internet選項-設置-察看文件卻始終看不到生成的cookie文件，奇怪，代碼明明沒有問題，不過既然有cookie，那就顯示出來看看。

*cookie[]cookies = request.getCookies();*

*if (cookies.lenght == 0 || cookies == null) out.println("Has not visited this website");*

*else {*

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

*out.println("cookie name:" + cookies[i].getName() + "cookie value:" +*

*cookie[i].getValue( ));*

*}*

*}*

運行結果:

cookie name:JSESSIONID cookie value:KWJHUG6JJM65HS2K6， http是無狀態的協議，客戶每次讀取web頁面時，服務器都打開新的會話，而且服務器也不會自動維護客戶的上下文信息，那麼要怎麼才能實現網上商店中的購物車呢，session就是一種保存上下文信息的機制，它是針對每一個用戶的，變量的值保存在服務器端，通過SessionID來區分不同的客戶，session是以cookie或URL重寫為基礎的，默認使用cookie來實現，系統會創造一個名為JSESSIONID的輸出cookie，叫做session cookie，以區別persistent cookies，也就是通常所說的cookie，session cookie存儲於瀏覽器內存中，不是寫到硬盤上，這也就是剛才看到的JSESSIONID，我們通常情是看不到JSESSIONID的，但是當我們把瀏覽器的cookie禁止後，web服務器會採用URL重寫的方式傳遞Sessionid，就可以在地址欄看到sessionid=KWJHUG6JJM65HS2K6之類的字符串。

明白了原理，我們就可以很容易的分辨出persistent cookies和session cookie的區別了，網上那些關於兩者安全性的討論也就一目了然了，session cookie針對某一次會話而言，會話結束session cookie也就隨著消失了，而persistent cookie只是存在於客戶端硬盤上的一段文本（通常是加密的），而且可能會遭到cookie欺騙以及針對cookie的跨站腳本攻擊，自然不如session cookie安全了。

通常session cookie是不能跨窗口使用的，當你新開了一個瀏覽器窗口進入相同頁面時，系統會賦予你一個新的sessionid，這樣我們信息共享的目的就達不到了，此時我們可以先把sessionid保存在persistent cookie中，然後在新窗口中讀出來，就可以得到上一個窗口SessionID了，這樣通過session cookie和persistent cookie的結合我們就實現了跨窗口的session tracking（會話跟踪）。