[http无状态还是web应用无状态](http://blog.csdn.net/sun_dragon/article/details/8715747)

2013-03-25 10:24 2885人阅读 [评论](http://blog.csdn.net/sun_dragon/article/details/8715747#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/sun_dragon/article/details/8715747#report)

我们通常说的web应用程序的无状态性的含义是什么呢？

直观的说，“每次的请求都是**独立的**，它的执行情况和结果与前面的请求和之后的请求是**无直接**关系的，它不会受前面的请求应答情况**直接**影响，也不会**直接**影响后面的请求应答情况”

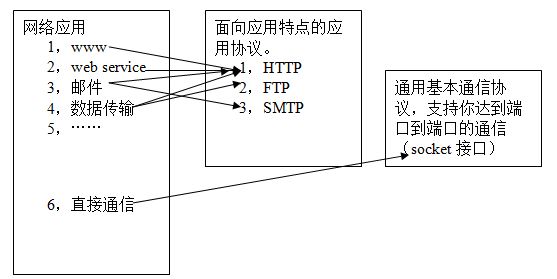
要明白，这句话的含义是指在说明，http协议作为技术背景的web应用程序**请求——应答**模式是无状态的，这个事实基本不会发生改变，也不会因为加入cookies、session机制而变成有状态的。要明白，这种前后因果关系：“我们要实现的是一种web应用，实现这种应用的协议我们选择了http这种本质上是无状态的通信协议。但是事实上，我们需要我们的web应用是有状态的。所以我们加入了cookies、session等机制去实现由状态的web应用”。所以我们可以这么理解：

***Web应用=http协议+session、cookies等状态机制+其他辅助的机制。***

其实，应用程序(软件通信)的状态与否是一个非常通用的概念。我们可知，在网络协议中，我们称TCP为一个有状态的传输层通信协议，而UDP则不是；IP是无状态的。要明白这种状态与否的判定，是面向你这一层次所指实现的功能——是否由上下文决定——来判定的（是否受之前的通信过程直接影响、是否直接影响之后的通信过程）

IP层以下的我们就不思考了。我们做一个网络应用，需要使用网络协议。其实按照原理上讲，标准的TCP/IP协议提供给我们的网络层协议(FTP,　HTTP)不能直接的被称为应用，因为在实现某种可用的、直接面向用户的应用的时候（如web应用，人们可以上网），只有http协议还是不够的。所以我们可以这么理解。网络标准协议分层中提供给我们的应用层协议，它更像是一种分类。自然界的应用可能是无穷尽的种类，但是根据他们的特点、传输的特色，标准的网络协议在传输层（通用网络协议）的基础上封装出若干种面向不同种类网络应用的协议。

某种角度上讲，我们想要实现某种可用的网络应用，直接使用网络协议的传输层给我们提供的接口就可以了（也就是socket接口），但是有时候，这种方式是有些麻烦的，所以我们还是根据你要实现的web应用，在已有的标准协议中提供的面向应用分类的协议中进行选择。这样可以免去那些繁琐的、通用的工作。

[](http://photo.blog.sina.com.cn/showpic.html#blogid=93b45b0f0101a4ix&url=http://s7.sinaimg.cn/orignal/93b45b0fgc9e2e0ae2236&690)

可以看到，我们实际生活中的有关网络的应用程序，与标准的网络通信协议提供给我们的应用层协议是没有绝对的对应关系。所以标准的网络通信协议给我们提供的应用层协议，只是提供给我的一种“建议的”分类。建议你：“如果你要实现***这样的***应用，你可以直接使用这个封装协议，而不是socket接口”。

我们再看看，前面一副图中，所给我们的关于网络应用层次中的各层次的有无状态情况。可以知道，支持协议（下层）的有无状态，消费协议（上层）的有无状态，没有直接的关系。还是那句话，每层协议的有无状态关系到它的本身功能执行的时候的有无状态地特点。

（1）IP是无状态的，它只负责将一个IP包发送到指定的IP地址上去。它不会考虑这个包与前面已经发送的包和后面的包的联系。（可能是重发包、可能是不连续包，它不管）。

（2）    TCP是**有状态**的，它通过包头中的一些控制字段（序列编码等）来表明各个包之间的关系（前后关系，重包与否等等）。所以，通过这个协议你可以做到一个**可靠**的传输。那么TCP是**面向连接**的协议是什么意思呢？其实这里的面向连接其实就是“三次握手”。三次握手，首先可以保证对方的存在，其次握手的所交换的内容是为将来进行有状态的传输做准备。

（3）    UDP是无状态的，它仅仅是在IP上加了Port，其他的事情什么也不干。这样它不可能做到可靠的传输，同样也不需要连接。

（4）    HTTP是无状态的，它的底层协议是由状态的TCP，但是HTTP的一次完整协议动作，里面是使用有状态的TCP协议来完成的。而每次协议动作之间没有任何关系。例如：第7次请求HTTP协议包，并不知道，这个包是为了什么？它或许是因为上次没有请求成功而重传，或许是上次的后续请求，或许是其他的，这些HTTP自身都不知道。

（5）    www应用，但是很多时候，www应用是需要HTTP动作之间是有关联的，那就是使应用有状态。这样才能提供给用户最好的用户体验。

于是，问题就来了，为什么当初HTTP会设计成无状态的，既然现在我们所需要的www应用是有状态的，为什么给他提供的这样的底层协议是无状态的。我想这个问题，可以从历史的角度去思考。在www应用还很简单的时候，这个应用只是被用来浏览内容。如果只是浏览内容的话，无状态的协议已经够了，这样实现可以减轻实现的负担，因为有状态的协议实现起来代价相对来说是很高的（要维护状态，根据状态来处理情况，这就是为什么建议你可以不用session的时候就不用，因为服务器要给你负担起很多的东西，例如内存空间啊）。好，现在看来，似乎www 应用是大部分需要状态了，那么是否我们就应该改变这个协议来让他变成一个有状态的协议呢？从这个角度上讲，我认为是不应该的。

首先，web应用与文件传输是不同的，文件传输，从开始到结束是一个“尽可能做完”的动作，所以这类动作不会在资源占有上，浪费它不该浪费的东西。而web应用中，用户可能访问一个页面后，在那个页面上逗留很久才跳转到另外一个页面，如果你需要我们在这两个页面（两个http请求应答）之间维持状态，是非常代价高的。

其次，历史让http无状态，而应用需求对http提出有状态的要求，按照软件领域的通常做法是，保持历史遗留的经验（不再http协议本质上作太大的改动），兼容过去的软件。在http上再加上一层来实现我们的目的（ “再加上一层，你能做任何事”）。这一层，就是cookies，就是session等。

最后总结，http协议仍然保持无状态，其充分的理由，并且，想要基于http协议的web应用变得有状态，实现起来并不麻烦。

**web应用都有哪些方法来让应用有状态**

于是就引出了，在http协议的基础上，web应用引入cookies, session, application。这样的东西来保持web应用之间的状态。

可知，cookies, session，application都不是标准协议，但是各种网络应用提供商，实现语言、web容器等，都默认支持它。当然这种支持与对网络标准协议的支持是不同的，标准协议规定的接口，而这种机制，只是规定了思想。就告诉你，大的概念上，jsp和ASP的session机制所要实现的功能和实现的方法不会有太大的出入。

有人将web应用中有无状态的情况，比着顾客逛商店的情景。

顾客：浏览器访问方；

商店：web服务器；

一次购买：一次http访问

我们知道，上一次顾客购买，并不代表顾客下一个小时一定会买（当然也不能代表不会）。也就是说同一个顾客的不同购买之间的关系是不定的。所以说实在的，这种情况下，让商店保存所有的顾客购买的信息，等到下一次购买可以知道这个顾客以前购买的内容代价非常大的。所以商店为了避免这个代价，索性就认为每次的购买都是一次独立的新的购买。浅台词：商店不区分对待老顾客和新过客。这就是无状态的。

但是，商店为了提高收益。她是想鼓励顾客购买的。所以告诉你，只要你在一个月内购买了５瓶以上的啤酒，就送你一个酒杯。

我们看看这种情况我们怎么去实现呢？

A,给顾客发放一个磁卡，里面放有顾客过去的购买信息。

这样商店就可以知道了。这就是cookie.

B,给顾客发放一个唯一号码，号码制定的顾客的消费信息，存储在商店的服务器中。这就是session。

最后，商店可以全局的决定，是５瓶为送酒杯还是6瓶。这就是application。

其实，这些机制都是在无状态的传统购买过程中加入了一点东西，使整个过程变得有状态。Web应用就是这样的。

**注意，viewstate**

其实从我们上面的分析看来，application不应该被视为这种意义上出现的维护状态的机制。它是决定怎么应用程序的“配制文件”。但是如果你从这种状态维持机制所覆盖的范围来推导，你会发现，application好像也算得上。

Session所控制的范围是一个session。一个会话，会话从第一次访问服务器开始存在，到服务器调用session.invalidator()（可能是超时，可能是其它原因）。

Cookies所控制的范围有它自己的定义(与session没有直接的关系)，可以长可以短。只要服务器放在用户文件系统中的cookies没有被删除，至少服务器还识别它。它的控制范围就是还在的。

这个角度上讲，Session和Cookies都可以归为跨页面的状态。但是session跨不出一次会话，Cookies跨不出两端的限制。

Application，则是关联这个网络应用程序的。

除了这三种状态机制，有些网络应用的特别实现还各自特殊的东西。像ASP.net的viewstate。这个东西，其实就是使用input type=”hyde”的东西来实现的。再加上web服务器那边辅助，就可以实现，一个页面在“回传”的时候还能保持上一次的页面状态。

其实，这里要说明的就是“回传”的意思，事实上，无状态的http协议下，是没有所谓的回传的概念的。这里的回传，情景可以这么理解。

我在客户端填写一定的数据（在文本框中、密码框中、下拉列表中）。然后触发一个访问服务器的动作，例如：提交。这个时候服务器接受到我的请求然后又返回刚才这个页面给我们。这种情景我们很常见（输入用户名密码，点击“登陆”，服务器端判断密码不对，就返回登陆页面，但是之前我们填写的内容并没有被擦出。）这种情景就好像实现了一种回传（你传给我，我传还给你）。当然，如果你了解web服务的工作模式，这种回传只会是一种假象。前后两个页面根本不是一个页面。其实我们可以实现这种情景（不考虑Ajax）。页面上的数据全部通过request的方式传到服务器，服务器返回页面的时候，将数据由注入到页面中。

而viewstate就可以让用户很容易的实现这种功能。但是它有一个问题就是它附属于页面上的hide控件。虽然前后是两个不同的页面，但是来自一个ASPX模板，页面上的状态数据就在这些同模板页面的hide空间之间进行传播（由服务器控制），不可能，也不应该传递到别的ASP.new的页面上去。所以我们称他为页面级的状态维持机制。