先来看这句话的另外两个概念：（标准的 http 协议是无状态的，无连接的）

1、标准的 http 协议指的是不包括 cookies，session，application（是手机应用）的 http 协议，他们都不属于标准协议，虽然各种网络应用提供商，实现语言、web 容器等，都默认支持它

2、无连接指的是什么

1、每一个访问都是无连接，服务器挨个处理访问队列里的访问，处理完一个就关闭连接，这事儿就完了，然后处理下一个新的

2、无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接

对于【无状态】，我看到很多隔着一层磨砂玻璃一样的模糊说法（官方或者教程里的说法），看着非常难受（但其实算是对的）（后来我发现我为什么觉得它看着难受了，因为他们引入了很多新的，而且明显是一个可能用在很多地方的广义名词，这些词最大的作用就是，混淆概念，下面我标注了）

1、协议对于事务处理没有记忆能力【事物处理】【记忆能力】

2、对同一个 url 请求没有上下文关系【上下文关系】

3、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【无直接联系】【受直接影响】

4、服务器中没有保存客户端的状态，客户端必须每次带上自己的状态去请求服务器【状态】

我必须得到确切而具体的解释！

这几点给了我下一步思考的方向：

1、【服务器中没有保存客户端的状态，客户端必须每次带上自己的状态去请求服务器 】这里的客户端的状态是不是确切地指服务器没有保存客户的信息呢？但显然不是啊

2、【HTTP 无状态的特性严重阻碍了这些应用程序的实现，毕竟交互是需要承前启后的，简单的购物车程序也要知道用户到底在之前选择了什么商品】我对此质疑为什么无状态就不能实现购物车呢？服务器就不能存储东西了么？

3、【 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】我觉得这个说法比较靠谱，但是所谓的不同请求间的没有关系，是指的请求内容没有关系，还是只是指请求本身没有关系？

1、请求内容没有关系只可能是服务器上不存有用户数据才可能啊，但是显然是存有的啊

2、请求本身没有关系，这又有什么意义呢，每一次的请求有什么价值？

根据这个方向我做了一个模拟访问实验：假如没有 cookie 没有 session，只有 http 的时候，那当一个注册用户访问这个购物网站的时候，会发生这些事情：

1、前提情况：

1. 服务器肯定为每个注册用户建立了数据表，记录用户的数据

2. http 是无连接的

2、第一步需要登录

1. 用户通过 http 把用户的用户名和密码发送给服务器，服务器把他们跟自己存有的用户资料对比，如果一致，则返回信息登录成功

3、然后用户点击某一商品页

1. 这个动作相当于输入一个商品页的网址

2. 假如商品页比较机密不对外公开，需要是用户才能访问

3.而虽然 http 能传送用户名和密码，而且刚才也输入了，还验证成功了，但是因为服务器既不会记得你登录的状态，你的客户端也不会存储你刚才输入的用户名和密码

4.所以因为这一次访问因为无法确定你的身份，只能访问失败

1.这时候如果要解决这个问题，而且没有 cookie 没有 session，那就只能你在访问网址的同时继续带上你的用户名和密码（继续输入咯）

4、假设上一步的问题解决了，就是每次访问的时候都会手动输入用户名和密码，然后现在的情况是：你已经选了几件商品在你的购物车中，你想再添加一件商品，于是你点击某个商品旁边的加号

1.这个动作也相当于输入一个网址，网址的内容是发送一个请求，往你的购物车中加入这个商品

2.系统首先用你传来的用户名和密码验证你的身份，然后访问你的数据库，在其中的购物车属性下加一条数据，就是这个商品的数据

3.操作结束后，返回操作成功，并结束访问

5、OK，实验结束，看似没有 cookie 没有 session 也能凑合解决问题，其实两个操作都有很大的问题

1.你每访问一次需要权限的内容都需要在客户端输入用户名和密码，这一项的繁琐就不必赘述了

2.你的每一次操作都要与系统底层的数据库进行交互

1.多次少量的访问存在非常大的性能浪费。非常容易就能想到肯定是一次大量的操作更加有效率，于是就想到了缓存区

3.你的非重要琐碎数据也被写进数据库中，跟你的主要数据放在一起

1.一次次添加和删除购物车其实只是跟你这次浏览，或者叫这次会话有关，是临时的数据，跟用户的主要信息无关，它们没什么价值，纯粹的冗余数据（不排除现在有的公司觉得这种数据也有非常大的价值可以让它们巧妙的利用），用什么存放这些临时的数据，我们也很容易想到缓存区

经过这个模拟访问实验，结合前面的思考方向，我们知道了三点：

1、服务器上肯定存有用户的数据，你提交的增删改查它也能够处理，所以这句话中【服务器中没有保存客户端的状态】的状态并不是指用户的数据，我们的猜测不对

2、我们的质疑对了，无状态能实现购物车，可以通过服务器上存有的用户数据来实现

3、但是，使用上面这种方式实现购物车，存在三个比较大的问题。由此，我们不禁会想，这三个问题的解决是不是跟我们不确切了解的【状态】一词有关？于是，接下来我们来通过解决这三个问题来把【状态】的意义探寻下去

由上所述，我们可以在 http 的基础上增加一些机制来解决上面出现的三个问题

1、在用户端增加一个记录本是非常有必要的，正好官方加入的 cookie 机制跟这个一样，它的用处也确实是上面讨论的那样，一般就是用来标识访问者的身份

2、在服务器增加一个缓存区能同时解决后两个问题

1.有了这个缓存区作为一个数据缓冲，就不用一次次地访问数据库，浪费大量计算机资源，而是在最后统一归入数据库

2.有了这个缓存区，你就不用把临时的数据放到数据库中了，只需要在你们交流告一段落之后，再把数据整理，把有用的数据归入数据库

3、这里就自然引申出了一个重要的概念：会话，它作为一个缓冲存储区被从数据库中分离出来，理由并不生硬，它有其独特的重要且不可替代的作用。这个东西恰好跟官方加入的 session 机制一样

1.另外说一个非常具有迷惑性的容易让人对 session 的主要作用产生偏离的理解：认为 session 存在的价值就是给访问者分配一个 sessionID 代替用户名和密码，

2.为什么非常具有迷惑性，因为 session 确实做了这件事，而且也起到了很大的作用，所以它是对的，但是只对一半，而且没有涉及问题的本质，这种情况是最危险的（看似很有说服力，把你说服了，所以你很难有动力继续找下去，但是真实情况跟它有偏差，但是偏差不大，所以又很难把你说服回来，只有隐隐的不对劲，这个时候你离真实最近，也离真实最远）

3.那就顺便说说它为什么是对的，也就是用 session 做的另一件有用的事：

1.给每个 session 一个 ID，一方面用来方便自己查询，另一方面把这个 ID 给用户，用户下一次访问的时候就可以不用用户名和密码，而是直接使用这个 ID 来表明自己的身份

2.首先，这个 ID 安全吗？这个 ID 比直接传用户名和密码安全吗？

1.你很容易会想到，本来用户名和密码的组合还特地设置地比较复杂，你这换一组数字就代替了，是不是太不安全了？

2.我们知道 http 协议本身是完全不加密的，如果使用用户名和密码，第一次访问是放在 http 头中，后边自动保存了密码就会放在 cookie 中，这些都完全没有加密，它的安全性基本为 0，就是裸奔了，只要被窃取，那就丢失了

3.所以，就这个意义来讲，sessionID 的安全性跟使用用户名和密码没什么区别

4.但是其实，虽然 http 本身不能加密，但是有些软件什么的，能在应用层面手动给你加密，比如 QQ 就会使用户名密码加临时验证码联合哈希，sessionID 加一个时间戳简单加密也是非常常用的方法

5.而且因为 sessionID 本身有有效期，即使丢了，也可能很快失效，造成的损失可能没那么大，而用户名跟密码丢了，那就大了

6.所以总结就是：

1.不严格加密的 sessionID 和用户名和密码一样，都不太安全

2.但是相比较来说，sessionID 要安全一些

3.而使用 https 是完全安全的

3.然后，使用 sessionID 有哪些好处

1.方便直接根据 ID 查询用户对应的 session

2.加密的时候计算量小

3.安全性不会降低，甚至还更高一些

OK，通过独立地解决纯 http 机制会产生的问题，我们探讨了 cookie 和 session 机制的本质。而且想到：【使用 http 协议，服务器中不会保存客户端的状态】所产生的问题通过增加 cookie 和 session 机制解决了，是不是就意味着这个【状态】跟 cookie 和 session 的关系非常紧密？所以这个无状态指的是【没有对 本次会话 设置一个缓存区，记录这次会话的状态，缓存区包括服务器端和用户端】但好像还是没有点破关键（主要是觉得跟前面那些官方对状态的说法不太吻合，甚至没有对应关系）忽然我想到一个问题：一个有状态的 http 是什么样的？

1、很难直接想象有状态的 http 是什么样，因为 http 这种机制是天然无状态的

2、那就类比一下吧，另一个天然有状态的机制叫 TCP

1.如果有状态的意思是它的每次请求是有联系的，那么有状态的 TCP 的样子是：假如一份数据分了三份 TCP 包发送，那这个包上面会标明这是第几个包，会标明这个包跟那几个包是有联系的，有什么联系

3、但好像这个有状态的 TCP 跟我们想要的有状态的 HTTP 没有关系，因为即使每次 http 请求之间互相有联系，它也不能解决上面提到的 http 无状态的问题

4、诶，等等，好像能类比：

1.假如每个 http 连接都有一个签名，于是第一次登陆成功之后，服务器就知道了这个签名是允许登陆的，于是之后所有同样签名的 http 连接都能登陆，这里利用了同一个用户发出的 http 连接之间的同主人关系，这里解决了一个保持登录状态的问题

2.同样，来尝试利用这个【每次 http 请求之间互相有联系】来解决上面碰到的那个问题【每一次操作都要与系统底层的数据库进行交互】，但想了半天确实无法进行下去

3.不过我灵机一动，从另一个角度来想，好像解决了这个问题：

1.只有【每次 http 请求之间互相有联系】这个条件，无法解决【每一次操作都要与系统底层的数据库进行交互】

2.因为很明显，要解决【每一次操作都要与系统底层的数据库进行交互】就必须在服务器端开辟一块缓存区

3.不过如果你思考一下如何实现【每次 http 请求之间互相有联系】，你就会发现，它也需要在服务器端开辟一块缓存区

4.所以【在服务器端开辟一块缓存区】才是真正的条件，也就是说，它确实等价于【有状态】

5.而且我也找到了这个【在服务器端开辟一块缓存区】的条件跟前面那些官方对状态的说法对应的点，那就是：

1.通过在服务器端开辟一块缓存区，存储、记忆、共享一些临时数据，你就可以：

1.协议对于事务处理有记忆能力【事物处理】【记忆能力】

2.对同一个 url 请求有上下文关系【上下文关系】

3.每次的请求都是不独立的，它的执行情况和结果与前面的请求和之后的请求是直接关系的【不独立】【直接关系】

4.服务器中保存客户端的状态【状态】

6.所以，这个状态，加上前面说的客户端也有 cookie，就是指，客户端和服务器在临时会话中产生的数据！而前面也说道了，使用缓存区保存临时会话中的数据是多么重要

1.所以状态不仅包括不同 URL 访问之间的关系，还有对其他 URL 访问的数据记录，还有一些其他的东西，所以更确切地说，状态应该是【实现了这些东西所凭借的后面的缓存空间】中的客户的临时数据

2.cookie 和 session 应该是完全实现了有状态这个功能

一种常见的对状态的误解：

1、有人在解释 HTTP 的无状态时，把它跟有连接对立，说是两种方式，也就是如果想不无状态，就必须有连接，但其实不然

2、有连接和无连接以及之后的 Keep-Alive 都是指 TCP 连接

3、有状态和无状态可以指 TCP 也可以指 HTTP

4、TCP 一直有状态，HTTP 一直无状态，但是应用为了有状态，就给 HTTP 加了 cookie 和 session 机制，让使用 http 的应用也能有状态，但 http 还是无状态

开始 TCP 是有连接，后来 TCP 无连接，再后来也就是现在 TCP 是 Keep-Alive，有点像有连接

参考：https://www.cnblogs.com/bellkosmos/p/5237146.html

对 Http 协议无状态的理解

Http 是一个无状态协议，同一个会话的连续两个请求互相不了解，他们由最新实例化的环境进行解析，除了应用本身可能已经存储在全局对象中的所有信息外，该环境不保存与会话有关的任何信息。

自己的理解，在 asp.net 里：每次提交服务器的页面没有任何关系，每次记录在页面的信息下次提交是记不住的，（除了应用本身可能已经存储在全局对象中的所有信息外）在.net 里实际就是 ViewState，ViewState 是 asp.net 中保存页面信息的基本单位，应用时就是保存在控件隐藏域等中的数据

协议的状态是指下一次传输可以“记住”这次传输信息的能力.

http 是不会为了下一次连接而维护这次连接所传输的信息的.

无状态是指，当浏览器发送请求给服务器的时候，服务器响应，但是同一个浏览器再发送请求给服务器的时候，他会响应，但是他不知道你就是刚才那个浏览器，简单地说，就是服务器不会去记得你，所以是无状态协议。而 DNS 是有状态协议 。

HTTP 是一个属于应用层的面向对象的协议，HTTP 协议一共有五大特点，1、支持客户/服务器模式；2、简单快速；3、灵活；4、无连接；5、无状态；“无状态”是 HTTP 协议的主要特点之一，以下为“无状态”的解释。无状态：是指

协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

HTTP 是 Hyper Text Transfer Protocol 的缩写，顾名思义，这个协议支持着超文本的传输。那么什么是超文本呢？说白了就是使用 HTML 编写的页面。通常，我们使用客户端浏览器访问服务器的资源，最常见的 URL 也是以 html 为后缀的文件。因此，我们可以说超文本是网络上最主要的资源。

既然 HTTP 协议的目的在于支持超文本的传输，更加广义一些就是支持资源的传输，那么在客户端浏览器向 HTTP 服务器发送请求，继而 HTTP 服务器将相应的资源发回给客户端这样一个过程中，无论对于客户端还是服务器，都没有必要记录这个过程，因为每一次请求和响应都是相对独立的，就好像你在自动售货机前投下硬币购买商品一样，谁都不会也不需要记住这样一个交易过程。一般而言，一个 URL 对应着唯一的超文本，而 HTTP 服务器也绝对公平公正，不管你是 Michael，还是 Jordon，它都会根据接收到的 URL 请求返回相同的超文本。正是因为这样的唯一性，使得记录用户的行为状态变得毫无意义，所以，HTTP 协议被设计为无状态的连接协议符合它本身的需求。

然而，随着时间的推移，人们发现静态的 HTML 着实无聊而乏味，增加动态生成的内容才会令 Web 应用程序变得更加有用。于是乎，HTML 的语法在不断膨胀，其中最重要的是增加了表单（Form）；客户端也增加了诸如脚本处理、DOM 处理等功能；对于服务器，则相应的出现了 CGI（Common Gateway Interface）以处理包含表单提交在内的动态请求。在这种客户端与服务器进行动态交互的 Web 应用程序出现之后，HTTP 无状态的特性严重阻碍了这些应用程序的实现，毕竟交互是需要承前启后的，简单的购物车程序也要知道用户到底在之前选择了什么商品。于是，两种用于保持 HTTP 连接状态的技术就应运而生了，一个是 Cookie，而另一个则是 Session。

Cookie 是通过客户端保持状态的解决方案。从定义上来说，Cookie 就是由服务器发给客户端的特殊信息，而这些信息以文本文件的方式存放在客户端，然后客户端每次向服务器发送请求的时候都会带上这些特殊的信息。让我们说得更具体一些：当用户使用浏览器访问一个支持 Cookie 的网站的时候，用户会提供包括用户名在内的个人信息并且提交至服务器；接着，服务器在向客户端回传相应的超文本的同时也会发回这些个人信息，当然这些信息并不是存放在 HTTP 响应体（Response Body）中的，而是存放于 HTTP 响应头（Response Header）；当客户端浏览器接收到来自服务器的响应之后，浏览器会将这些信息存放在一个统一的位置，对于 Windows 操作系统而言，我们可以从：[系统盘]:\Documents and Settings\[用户名]\Cookies 目录中找到存储的 Cookie；自此，客户端再向服务器发送请求的时候，都会把相应的 Cookie 再次发回至服务器。而这次，Cookie 信息则存放在 HTTP 请求头（Request Header）了。

有了 Cookie 这样的技术实现，服务器在接收到来自客户端浏览器的请求之后，就能够通过分析存放于请求头的 Cookie 得到客户端特有的信息，从而动态生成与该客户端相对应的内容。通常，我们

可以从很多网站的登录界面中看到“请记住我”这样的选项，如果你勾选了它之后再登录，那么在下一次访问该网站的时候就不需要进行重复而繁琐的登录动作了，而这个功能就是通过 Cookie 实现的。

与 Cookie 相对的一个解决方案是 Session，它是通过服务器来保持状态的。由于 Session 这个词汇包含的语义很多，因此需要在这里明确一下 Session 的含义。首先，我们通常都会把 Session 翻译成会话，因此我们可以把客户端浏览器与服务器之间一系列交互的动作称为一个 Session。从这个语义出发，我们会提到 Session 持续的时间，会提到在 Session 过程中进行了什么操作等等；其次，Session 指的是服务器端为客户端所开辟的存储空间，在其中保存的信息就是用于保持状态。从这个语义出发，我们则会提到往 Session 中存放什么内容，如何根据键值从 Session 中获取匹配的内容等。

要使用 Session，第一步当然是创建 Session 了。那么 Session 在何时创建呢？当然还是在服务器端程序运行的过程中创建的，不同语言实现的应用程序有不同创建 Session 的方法，而在 Java 中是通过调用 HttpServletRequest 的 getSession 方法(使用 true 作为参数)创建的。在创建了 Session 的同时，服务器会为该 Session 生成唯一的 Session id，而这个 Session id 在随后的请求中会被用来重新获得已经创建的 Session；在 Session 被创建之后，就可以调用 Session 相关的方法往 Session 中增加内容了，而这些内容只会保存在服务器中，发到客户端的只有 Session id；当客户端再次发送请求的时候，会将这个 Session id 带上，服务器接受到请求之后就会依据 Session id 找到相应的 Session，从而再次使用之。正式这样一个过程，用户的状态也就得以保持了。有关 Session 的内容还比较多，在以后的 Post 中，我还将继续讲述。

综上所述，HTTP 本身是一个无状态的连接协议，为了支持客户端与服务器之间的交互，我们就需要通过不同的技术为交互存储状态，而这些不同的技术就是 Cookie 和 Session 了

文章出处：https://www.cnblogs.com/open-source-java/p/10443918.html