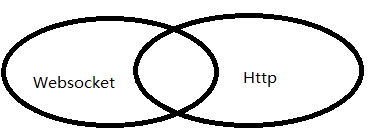
作者：Ovear  
链接：https://www.zhihu.com/question/20215561/answer/40316953  
来源：知乎  
著作权归作者所有，转载请联系作者获得授权。  
  
**一、WebSocket是HTML5出的东西（协议），也就是说HTTP协议没有变化，或者说没关系，但HTTP是不支持持久连接的（长连接，循环连接的不算）**  
首先HTTP有1.1和1.0之说，也就是所谓的keep-alive，把多个HTTP请求合并为一个，但是Websocket其实是一个新协议，跟HTTP协议基本没有关系，只是为了兼容现有浏览器的握手规范而已，也就是说它是HTTP协议上的一种补充可以通过这样一张图理解  
<img src="https://pic1.zhimg.com/6651f2f811ec133b0e6d7e6d0e194b4c\_b.jpg" data-rawwidth="374" data-rawheight="133" class="content\_image" width="374">有交集，但是并不是全部。有交集，但是并不是全部。  
另外Html5是指的一系列新的API，或者说新规范，新技术。Http协议本身只有1.0和1.1，而且跟Html本身没有直接关系。。  
通俗来说，你可以用HTTP**协议**传输非Html**数据**，就是这样=。=  
再简单来说，**层级不一样**。  
  
**二、Websocket是什么样的协议，具体有什么优点**  
首先，Websocket是一个**持久化**的协议，相对于HTTP这种**非持久**的协议来说。  
简单的举个例子吧，用目前应用比较广泛的PHP生命周期来解释。  
1) HTTP的生命周期通过Request来界定，也就是一个Request 一个Response，那么**在**HTTP1.0**中**，这次HTTP请求就结束了。  
在HTTP1.1中进行了改进，使得有一个keep-alive，也就是说，在一个HTTP连接中，可以发送多个Request，接收多个Response。  
但是请记住 Request = Response ， 在HTTP中永远是这样，也就是说一个request只能有一个response。而且这个response也是**被动**的，不能主动发起。  
  
**教练，你BB了这么多，跟Websocket有什么关系呢？**  
\_(:з」∠)\_好吧，我正准备说Websocket呢。。  
首先Websocket是基于HTTP协议的，或者说**借用**了HTTP的协议来完成一部分握手。  
在握手阶段是一样的  
-------以下涉及专业技术内容，不想看的可以跳过lol:，或者只看加黑内容--------  
首先我们来看个典型的Websocket握手（借用Wikipedia的。。）

GET /chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

Origin: http://example.com

熟悉HTTP的童鞋可能发现了，这段类似HTTP协议的握手请求中，多了几个东西。  
我会顺便讲解下作用。

Upgrade: websocket

Connection: Upgrade

这个就是Websocket的核心了，告诉Apache、Nginx等服务器：**注意啦，窝发起的是Websocket协议，快点帮我找到对应的助理处理~不是那个老土的HTTP。**

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

首先，Sec-WebSocket-Key 是一个Base64 encode的值，这个是浏览器随机生成的，告诉服务器：**泥煤，不要忽悠窝，我要验证尼是不是真的是Websocket助理。**  
然后，Sec\_WebSocket-Protocol 是一个用户定义的字符串，用来区分同URL下，不同的服务所需要的协议。简单理解：**今晚我要服务A，别搞错啦~**  
最后，Sec-WebSocket-Version 是告诉服务器所使用的Websocket Draft（协议版本），在最初的时候，Websocket协议还在 Draft 阶段，各种奇奇怪怪的协议都有，而且还有很多期奇奇怪怪不同的东西，什么Firefox和Chrome用的不是一个版本之类的，当初Websocket协议太多可是一个大难题。。不过现在还好，已经定下来啦~大家都使用的一个东西~ 脱水：**服务员，我要的是13岁的噢→\_→**  
  
然后服务器会返回下列东西，表示已经接受到请求， 成功建立Websocket啦！

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=

Sec-WebSocket-Protocol: chat

这里开始就是HTTP最后负责的区域了，告诉客户，我已经成功切换协议啦~

Upgrade: websocket

Connection: Upgrade

依然是固定的，告诉客户端即将升级的是Websocket协议，而不是mozillasocket，lurnarsocket或者shitsocket。  
然后，Sec-WebSocket-Accept 这个则是经过服务器确认，并且加密过后的 Sec-WebSocket-Key。服务器：**好啦好啦，知道啦，给你看我的ID CARD来证明行了吧。。**  
后面的，Sec-WebSocket-Protocol 则是表示最终使用的协议。  
  
至此，HTTP已经完成它所有工作了，接下来就是完全按照Websocket协议进行了。  
具体的协议就不在这阐述了。  
------------------技术解析部分完毕------------------  
  
<img src="https://pic2.zhimg.com/afe119b52e096016139edabc2dfa9661\_b.jpg" data-rawwidth="161" data-rawheight="187" class="content\_image" width="161">你TMD又BBB了这么久，那到底Websocket有什么鬼用，http long poll，或者ajax轮询不都可以实现实时信息传递么。你TMD又BBB了这么久，那到底Websocket有什么鬼用，http long poll，或者ajax轮询不都可以实现实时信息传递么。  
<img src="https://pic1.zhimg.com/20110e661edb1e93755a99c1d826e264\_b.jpg" data-rawwidth="176" data-rawheight="193" class="content\_image" width="176">  
  
好好好，年轻人，那我们来讲一讲Websocket有什么用。  
来给你吃点胡（苏）萝（丹）卜（红）  
<img src="https://pic4.zhimg.com/31ddf0cfbeecef21568d85ca60b5f1ff\_b.jpg" data-rawwidth="53" data-rawheight="65" class="content\_image" width="53">https://pic4.zhimg.com/31ddf0cfbeecef21568d85ca60b5f1ff_b.jpg**三、Websocket的作用**  
在讲Websocket之前，我就顺带着讲下 long poll 和 ajax轮询 的原理。  
首先是 ajax轮询 ，ajax轮询 的原理非常简单，让浏览器隔个几秒就发送一次请求，询问服务器是否有新信息。  
场景再现：  
客户端：啦啦啦，有没有新信息(Request)  
服务端：没有（Response）  
客户端：啦啦啦，有没有新信息(Request)  
服务端：没有。。（Response）  
客户端：啦啦啦，有没有新信息(Request)  
服务端：你好烦啊，没有啊。。（Response）  
客户端：啦啦啦，有没有新消息（Request）  
服务端：好啦好啦，有啦给你。（Response）  
客户端：啦啦啦，有没有新消息（Request）  
服务端：。。。。。没。。。。没。。。没有（Response） ---- loop  
  
long poll   
long poll 其实原理跟 ajax轮询 差不多，都是采用轮询的方式，不过采取的是阻塞模型（一直打电话，没收到就不挂电话），也就是说，客户端发起连接后，如果没消息，就一直不返回Response给客户端。直到有消息才返回，返回完之后，客户端再次建立连接，周而复始。  
场景再现  
客户端：啦啦啦，有没有新信息，没有的话就等有了才返回给我吧（Request）  
服务端：额。。 等待到有消息的时候。。来 给你（Response）  
客户端：啦啦啦，有没有新信息，没有的话就等有了才返回给我吧（Request） -loop  
  
从上面可以看出其实这两种方式，都是在不断地建立HTTP连接，然后等待服务端处理，可以体现HTTP协议的另外一个特点，**被动性**。  
何为被动性呢，其实就是，服务端不能主动联系客户端，只能有客户端发起。  
简单地说就是，服务器是一个很懒的冰箱（这是个梗）（不会、不能主动发起连接），但是上司有命令，如果有客户来，不管多么累都要好好接待。  
  
说完这个，我们再来说一说上面的缺陷（原谅我废话这么多吧OAQ）  
从上面很容易看出来，不管怎么样，上面这两种都是非常消耗资源的。  
ajax轮询 需要服务器有很快的处理速度和资源。（速度）  
long poll 需要有很高的并发，也就是说同时接待客户的能力。（场地大小）  
所以ajax轮询 和long poll 都有可能发生这种情况。  
  
**客户端：啦啦啦啦，有新信息么？**  
**服务端：月线正忙，请稍后再试（503 Server Unavailable）**  
**客户端：。。。。好吧，啦啦啦，有新信息么？**  
**服务端：月线正忙，请稍后再试（503 Server Unavailable）**  
**客户端：**<img src="https://pic1.zhimg.com/7c0cf075c7ee4cc6cf52f4572a4c1c10\_b.jpg" data-rawwidth="143" data-rawheight="50" class="content\_image" width="143">https://pic1.zhimg.com/7c0cf075c7ee4cc6cf52f4572a4c1c10_b.jpg  
**然后服务端在一旁忙的要死：冰箱，我要更多的冰箱！更多。。更多。。（我错了。。这又是梗。。）**  
  
--------------------------  
**言归正传，我们来说Websocket吧**  
通过上面这个例子，我们可以看出，这两种方式都不是最好的方式，需要很多资源。  
一种需要更快的速度，一种需要更多的'电话'。这两种都会导致'电话'的需求越来越高。  
哦对了，忘记说了HTTP还是一个无状态协议。（感谢评论区的各位指出OAQ）  
通俗的说就是，服务器因为每天要接待太多客户了，是个**健忘鬼**，你一挂电话，他就把你的东西全忘光了，把你的东西全丢掉了。你第二次还得再告诉服务器一遍。  
  
所以在这种情况下出现了，Websocket出现了。  
他解决了HTTP的这几个难题。  
首先，**被动性**，当服务器完成协议升级后（HTTP->Websocket），服务端就可以主动推送信息给客户端啦。  
所以上面的情景可以做如下修改。  
客户端：啦啦啦，我要建立Websocket协议，需要的服务：chat，Websocket协议版本：17（HTTP Request）  
服务端：ok，确认，已升级为Websocket协议（HTTP Protocols Switched）  
客户端：麻烦你有信息的时候推送给我噢。。  
服务端：ok，有的时候会告诉你的。  
服务端：balabalabalabala  
服务端：balabalabalabala  
服务端：哈哈哈哈哈啊哈哈哈哈  
服务端：笑死我了哈哈哈哈哈哈哈  
  
就变成了这样，只需要经过**一次HTTP请求**，就可以做到源源不断的信息传送了。（在程序设计中，这种设计叫做回调，即：你有信息了再来通知我，而不是我傻乎乎的每次跑来问你）  
这样的协议解决了上面同步有延迟，而且还非常消耗资源的这种情况。  
那么为什么他会解决服务器上消耗资源的问题呢？  
其实我们所用的程序是要经过两层代理的，即**HTTP协议在Nginx等服务器的解析下**，然后再传送给相应的**Handler（PHP等）**来处理。  
简单地说，我们有一个非常快速的接**线员（Nginx）**，他负责把问题转交给相应的**客服（Handler）**。  
本身**接线员基本上速度是足够的**，但是每次都卡在**客服（Handler）**了，老有**客服**处理速度太慢。，导致客服不够。  
Websocket就解决了这样一个难题，建立后，可以直接跟接线员建立持**久连接**，有信息的时候客服想办法通知接线员，然后**接线员**在统一转交给客户。  
这样就可以解决客服处理速度过慢的问题了。  
  
同时，在传统的方式上，要不断的建立，关闭HTTP协议，由于HTTP是非状态性的，每次都要**重新传输identity info（鉴别信息）**，来告诉服务端你是谁。  
虽然接线员很快速，但是每次都要听这么一堆，效率也会有所下降的，同时还得不断把这些信息转交给客服，不但浪费客服的**处理时间**，而且还会在网路传输中消耗**过多的流量/时间。**  
但是Websocket只需要**一次HTTP握手，所以说整个通讯过程是建立在一次连接/状态中**，也就避免了HTTP的非状态性，服务端会一直知道你的信息，直到你关闭请求，这样就解决了接线员要反复解析HTTP协议，还要查看identity info的信息。  
同时由**客户主动询问**，转换为**服务器（推送）有信息的时候就发送（当然客户端还是等主动发送信息过来的。。）**，没有信息的时候就交给接线员（Nginx），不需要占用本身速度就慢的**客服（Handler）**了  
--------------------  
至于怎么在不支持Websocket的客户端上使用Websocket。。答案是：**不能**  
但是可以通过上面说的 long poll 和 ajax 轮询来 **模拟出类似的效果**  
-----  
\_(:з」∠)\_两天写了两篇科普类文章。。好累OAQ，求赞。。  
对啦，如果有错误，欢迎大家在底下留言指出噢~

作者：董可人  
链接：https://www.zhihu.com/question/20215561/answer/40250050  
来源：知乎  
著作权归作者所有，转载请联系作者获得授权。

你可以把 WebSocket 看成是 HTTP 协议为了支持长连接所打的一个大补丁，它和 HTTP 有一些共性，是为了解决 HTTP 本身无法解决的某些问题而做出的一个改良设计。在以前 HTTP 协议中所谓的 keep-alive connection 是指在一次 TCP 连接中完成多个 HTTP 请求，但是对每个请求仍然要单独发 header；所谓的 polling 是指从客户端（一般就是浏览器）不断主动的向服务器发 HTTP 请求查询是否有新数据。这两种模式有一个共同的缺点，就是除了真正的数据部分外，服务器和客户端还要大量交换 HTTP header，信息交换效率很低。它们建立的“长连接”都是伪.长连接，只不过好处是不需要对现有的 HTTP server 和浏览器架构做修改就能实现。  
  
WebSocket 解决的第一个问题是，通过第一个 HTTP request 建立了 TCP 连接之后，之后的交换数据都不需要再发 HTTP request了，使得这个长连接变成了一个真.长连接。但是不需要发送 HTTP header就能交换数据显然和原有的 HTTP 协议是有区别的，所以它需要对服务器和客户端都进行升级才能实现。在此基础上 WebSocket 还是一个双通道的连接，在同一个 TCP 连接上既可以发也可以收信息。此外还有 multiplexing 功能，几个不同的 URI 可以复用同一个 WebSocket 连接。这些都是原来的 HTTP 不能做到的。  
  
另外说一点技术细节，因为看到有人提问 WebSocket 可能进入某种半死不活的状态。这实际上也是原有网络世界的一些缺陷性设计。上面所说的 WebSocket 真.长连接虽然解决了服务器和客户端两边的问题，但坑爹的是网络应用除了服务器和客户端之外，另一个巨大的存在是中间的网络链路。一个 HTTP/WebSocket 连接往往要经过无数的路由，防火墙。你以为你的数据是在一个“连接”中发送的，实际上它要跨越千山万水，经过无数次转发，过滤，才能最终抵达终点。在这过程中，中间节点的处理方法很可能会让你意想不到。  
  
比如说，这些坑爹的中间节点可能会认为一份连接在一段时间内没有数据发送就等于失效，它们会自作主张的切断这些连接。在这种情况下，不论服务器还是客户端都不会收到任何提示，它们只会一厢情愿的以为彼此间的红线还在，徒劳地一边又一边地发送抵达不了彼岸的信息。而计算机网络协议栈的实现中又会有一层套一层的缓存，除非填满这些缓存，你的程序根本不会发现任何错误。这样，本来一个美好的 WebSocket 长连接，就可能在毫不知情的情况下进入了半死不活状态。  
  
而解决方案，WebSocket 的设计者们也早已想过。就是让服务器和客户端能够发送 Ping/Pong Frame（[RFC 6455 - The WebSocket Protocol](//link.zhihu.com/?target=https%3A//tools.ietf.org/html/rfc6455%23section-5.5.2)）。这种 Frame 是一种特殊的数据包，它只包含一些元数据而不需要真正的 Data Payload，可以在不影响 Application 的情况下维持住中间网络的连接状态。