

您查询的关键词是: **web socket** 以下是该网页在北京时间 2016年09月21日 11:25:19 的快照:

如果打开速度慢，可以尝试[快速版](#)；如果想更新或删除快照，可以[投诉快照](#)。

百度和网页 <http://www.zhihu.com/question/20215561/> 的作者无关,不对其内容负责。百度快照仅为网络故障时之索引,不代表被搜索网站的即时页面。

知乎

- 注册知乎
- 登录

提问

知乎搜索 搜索你感兴趣的内容...

搜索

- [首页](#)
- [话题](#)
- [发现](#)

消息

用户


赞同和感谢

[查看全部 »](#)

HTML5 HTTP WebSocket

2713 替同

反对

 [Ovear](#) 各种乱七八糟神马的。

2713 人赞同

额。。最高票答案没答到点子上，最后怎么跑到Nodejs上去了。。Websocket只是协议而已。。

我一个个来回答吧。

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

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？

(: 3] <) 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-----

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。

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）

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0AQ）

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：

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0AQ）

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哈
服务端：笑死我了哈哈哈哈哈

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，同时还不得不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~

[编辑于 2016-03-01](#) [151 条评论](#) [感谢](#) [分享](#) [收藏](#) · [没有帮助](#) · [举报](#) · [作者保留权利](#)

318 赞同 反对

 [董可人高频交易、量化交易、金融话题优秀回答者](#) 自选集《...

[318 人赞同](#)

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。

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](#))。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。

[发布于 2015-02-22](#) [26 条评论](#) [感谢](#) [分享](#) [收藏](#) • [没有帮助](#) • [举报](#) • [作者保留权利](#)

WebSocket 跟其他 API 比较不一样的是，它不仅仅依赖于浏览器支持，同时要求服务器和代理（假若需要经过代理的话）支持。WebSocket 本质上跟 HTTP 完全不一样，只不过为了兼容性，WebSocket 的握手是以 HTTP 的形式发起的，如果服务器或者代理不支持 WebSocket，它们会把这当做一个不认识的 HTTP 请求从而优雅地拒绝掉。

[websocket](#)的实现已经有朋友提到了基于HTTP来发起，我这里不讨论太多实现，而是主要回答一下题主补充的问题：“【】【】【】【】【】【】【】【】【】【补充】】】】】】】】】】：：：：：”
既然WebSocket和HTTP是两个协议 为什么要在HTML5才支持
如果说HTML5 出来以后可以用WebSocket了 就说明WebSocket是本来就有点东西只是HTML4不支持而已
http4时代 如何使用WebSocket呢？？ 谢谢”

而 websocket 的使用,更多的是“脚本”层面的事情,比如在javascript(不是非得javascript,但是javascript已经成为和浏览器交互的事实标准)中创建Websocket对象以便操作浏览器的websocket功能(这个功能便是浏览器根据HTML5草案实现出来的,参考草案的interface WebSocket部分定义),然后注册onopen/onmessage等回调接口,有数据或者状态变更之类的时机你就可以做相应的处理了。而这部分刚好在旧的标准没定义,浏览器也没有这样的功能,因此遵循HTML4编写的页面不会用到这些功能。

[发布于 2013-07-12](#) [添加评论](#) [感谢](#) [分享](#) [收藏](#) • [没有帮助](#) • [举报](#) • [作者保留权利](#)

“或者说WebSocket干脆就不是基于HTTP来执行的”，这句是对的，WebSocket 和 HTTP 都是基于 TCP 的，TCP是传输层协议， WebSocket 和 HTTP是应用层协议，就是因为HTTP不能满足特定的需求才被设计出来的，所以会支持你说的那些特性。

发布于 2012-05-05 添加评论 感谢 分享 收藏 · 没有帮助 · 举报 · 作者保留权利

12 赞同

反对

 知乎用户 <http://qixing318.com>[12 人赞同](#)

websocket约定了一个通信的规范，通过一个握手的机制，客户端和服务端之间能建立一个类似tcp的连接，从而方便它们之间的通信。在websocket出现之前，web交互一般是基于http协议的短连接或者长连接。websocket是一种全新的协议，不属于http无状态协议，协议名为“ws”，这意味着一个websocket连接地址会是这样的写法：ws://**。websocket协议本质上是一个基于tcp的协议。[分析HTML5中WebSocket的原理](#)

发布于 2013-12-25 [添加评论](#) [感谢](#) [分享](#) [收藏](#) · [没有帮助](#) · [举报](#) · [作者保留权利](#)

11 赞同

反对

 [panszAndroid 开发、Linux话题优秀回答者](#) 我说的大多是一…[11 人赞同](#)

是服务器实现的。

客户端通过html5与服务端交互。http是不持续连接的，而websocket是。必须服务端支持websocket协议，才有效。对于不支持websocket服务的服务器，你客户端怎么写代码都没用。

web服务器必须支持websocket协议。我想这可以简单的回答楼主的问题。

编辑于 2015-02-21 [3 条评论](#) [感谢](#) [分享](#) [收藏](#) · [没有帮助](#) · [举报](#) · [作者保留权利](#)

48 赞同

反对

 [长天之云前端开发话题优秀回答者](#) head, body { content }[48 人赞同](#)

WebSocket 是一种协议，基于 TCP 协议；HTTP 也是一种协议，基于 TCP 协议。连接要保持还是关闭是由你服务器应用来控制的。

WebSocket 协议和 HTTP 协议是两种不同的东西，它们扯上关系是只是因为：

客户端开始建立 WebSocket 连接时要发送一个 header 标记了 Upgrade 的 HTTP 请求，表示请求协议升级。

所以服务器端做出响应的简便方法是，直接在现有的 HTTP 服务器软件和现有的端口上实现 WebSocket 协议，重用现有代码（比如解析和认证这个 HTTP 请求。如果在 TCP 协议上实现，这两个功能就要重新实现），然后再回一个状态码为 101 的 HTTP 响应完成握手，再往后发送数据时就没 HTTP 的事了。

=====

看了补充，问题已经越来越奇怪了，变成了历史问题。这种概念上的问题多看点书很容易解决。

互联网一共才十几年历史，而 WebSocket 从提议变成推荐标准就需要几年，因为中间要经过大量的安全验证和实验。

Web 1 的时代人们访问 Web 页面是即停即走。Web 2 之后单个页面停留时间越来越长，页面功能越来越丰富——这时有 RIA 的概念，改变了客户端的编程模型——更甚至许多实时应用根本不用离开页面，比如聊天、游戏应用。

客户端浏览器决定了客户端编程语言的能拥有的功能，以前如何做那些交互性很高的应用呢？

一些技术有 XHR, iframe, 实时性要求非常高的就只能用第三方插件，比如 Flash 或 Silverlight。

但 XHR 和 iframe 存在一些根本避免不了问题：1) 每次交互就需要两个 HTTP 请求 2) 即使单个 HTTP 请求也要传送很多字节 (header 笨重) 3) 客户端不知道消息何时能够到达，只能轮询。服务器肯定会表示压力很大！

插件则需要额外安装，还有安全性问题和移动设备根本不能被支持的问题。

有了需要之后才有了解决方案——WebSocket 就是这种灵丹妙药，看看主要特性：实时交互、服务器能够主动推送内容、只需要建立一次连接、快速（延迟小，每条消息可以小到两个字节）、开发者友好（接口简单，并是熟悉的事件模型）等等。

所以，HTML 4 选择权很小，是否要支持 WebSocket 依据需求和环境而定；而选择 HTML 5 的话，有了 socket 通信和图形编程的能力，能够开发出什么精彩应用只取决于你的想象力。

PS：如果要找现成方案，这个 Wiki 上有很多：<https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills>

编辑于 2012-05-06 [3 条评论](#) [感谢](#) [分享](#) [收藏](#) · [没有帮助](#) · [举报](#) · [作者保留权利](#)

239 赞同

反对

 [潘良虎](#) JiaThis联合创始人

[239 人赞同](#)

Web领域的实时推送技术，也被称作Realtime技术。这种技术要达到的目的是让用户不需要刷新浏览器就可以获得实时更新。它有着广泛的应用场景，比如在线聊天室、在线客服系统、评论系统、WebIM等。

WebSocket简介

谈到Web实时推送，就不得不说WebSocket。在WebSocket出现之前，很多网站为了实现实时推送技术，通常采用的方案是轮询(Polling)和Comet技术，Comet又可细分为两种实现方式，一种是长轮询机制，一种称为流技术，这两种方式实际上是对轮询技术的改进，这些方案带来很明显的缺点，需要由浏览器对服务器发出HTTP request，大量消耗服务器带宽和资源。面对这种状况，HTML5定义了WebSocket协议，能更好的节省服务器资源和带宽并实现真正意义上的实时推送。

WebSocket协议本质上是一个基于TCP的协议，它由通信协议和编程API组成，WebSocket能够在浏览器和服务端之间建立双向连接，以基于事件的方式，赋予浏览器实时通信能力。既然是双向通信，就意味着服务器端和客户端可以同时发送并响应请求，而不再像HTTP的请求和响应。

为了建立一个WebSocket连接，客户端浏览器首先要向服务器发起一个HTTP请求，这个请求和通常的HTTP请求不同，包含了一些附加头信息，其中附加头信息”Upgrade: WebSocket”表明这是一个申请协议升级的HTTP请求，服务器端解析这些附加的头信息然后产生应答信息返回给客户端，客户端和服务端端的WebSocket连接就建立起来了，双方就可以通过这个连接通道自由的传递信息，并且这个连接会持续存在直到客户端或者服务器端的某一方主动的关闭连接。

一个典型WebSocket客户端请求头：

前面讲到WebSocket是HTML5中新增的一种通信协议，这意味着一部分老版本浏览器（主要是IE10以下版本）并不具备这个功能，[通过百度统计的公开数据显示](#)，IE8目前仍以33%的市场份额占据榜首，好在chrome浏览器市场份额逐年上升，现在以超过26%的市场份额位居第二，同时微软前不久宣布停止对IE6的技术支持并提示用户更新到新版本浏览器，这个曾经让无数前端工程师为之头疼的浏览器有望退出历史舞台，再加上几乎所有的智能手机浏览器都支持HTML5，所以使得WebSocket的实战意义大增，但是无论如何，我们实际的项目中，仍然要考虑低版本浏览器的兼容方案：在支持WebSocket的浏览器中采用新技术，而在不支持WebSocket的浏览器里启用Comet来接收发送消息。

WebSocket实战

本文将以多人在线聊天应用作为实例场景，我们先来确定这个聊天应用的基本需求。

需求分析

- 1、兼容不支持WebSocket的低版本浏览器。
- 2、允许客户端有相同的用户名。
- 3、进入聊天室后可以看到当前在线的用户和在线人数。
- 4、用户上线或退出，所有在线的客户端应该实时更新。
- 5、用户发送消息，所有客户端实时收取。

在实际的开发过程中，为了使用WebSocket接口构建Web应用，我们首先需要构建一个实现了 WebSocket规范的服务端，服务端的实现不受平台和开发语言的限制，只需要遵从WebSocket规范即可，目前已经出现了一些比较成熟的WebSocket服务端实现，比如本文使用的Node.js+<http://Socket.IO>。为什么选用这个方案呢？先来简单介绍下他们两。

Node.js

Node.js采用C++语言编写而成，它不是Javascript应用，而是一个Javascript的运行环境，据Node.js创始人Ryan Dahl回忆，他最初希望采用Ruby来写Node.js，但是后来发现Ruby虚拟机的性能不能满足他的要求，后来他尝试采用V8引擎，所以选择了C++语言。

Node.js支持的系统包括*nux、Windows，这意味着程序员可以编写系统级或者服务器端的Javascript代码，交给Node.js来解释执行。Node.js的Web开发框架Express，可以帮助程序员快速建立web站点，从2009年诞生至今，Node.js的成长的的速度有目共睹，其发展前景获得了技术社区的充分肯定。

<http://Socket.IO>

<http://Socket.IO>是一个开源的WebSocket库，它通过Node.js实现WebSocket服务端，同时也提供客户端JS库。<http://Socket.IO>支持以事件为基础的实时双向通讯，它可以工作在任何平台、浏览器或移动设备。

<http://Socket.IO>支持4种协议：WebSocket、htmlfile、xhr-polling、jsonp-polling，它会自动根据浏览器选择适合的通讯方式，从而让开发者可以聚焦到功能的实现而不是平台的兼容性，同时<http://Socket.IO>具有不错的稳定性和性能。

编码实现

本文一开始的的插图就是效果演示图：可以[点击这里](#)查看在线演示，整个开发过程非常简单，下面简单记录了开发步骤：

安装Node.js

根据自己的操作系统，去[Node.js官网](#)下载安装即可。如果成功安装。在命令行输入node -v和npm -v应该能看到相应的版本号。

搭建WebSocket服务端

这个环节我们尽可能的考虑真实生产环境，把WebSocket后端服务搭建成一个线上可以用域名访问的服务，如果你是在本地开发环境，可以换成本地ip地址，或者使用一个虚拟域名指向本地ip。

先进入到你的工作目录，比如 /workspace/wwwroot/plhwin/<http://realtime.plhwin.com>，新建一个名

为package.json的文件，内容如下：

接下来使用npm命令安装express和<http://socket.io>

安装成功后，应该可以看到工作目录下生成了一个名为node_modules的文件夹，里面分别是express和<http://socket.io>，接下来可以开始编写服务端的代码了，新建一个文件：index.js

命令行运行node index.js，如果一切顺利，你应该会看到返回的listening on *:3000字样，这说明服务已经成功搭建了。此时浏览器中打开 <http://localhost:3000> 应该可以看到正常的欢迎页面。

如果你想要让服务运行在线上服务器，并且可以通过域名访问的话，可以使用Nginx做代理，在nginx.conf中添加如下配置，然后将域名（比如：<http://realtime.plhwin.com>）解析到服务器IP即可。

完成以上步骤，<http://realtime.plhwin.com:3000>的后端服务就正常搭建了。

服务端代码实现

前面讲到的index.js运行在服务端，之前的代码只是一个简单的WebServer欢迎内容，让我们把WebSocket服务端完整的实现代码加入进去，整个服务端就可以处理客户端的请求了。完整的index.js代码如下：

客户端代码实现

进入客户端工作目录/workspace/wwwroot/plhwin/<http://demo.plhwin.com/chat>，新建一个index.html：

上面的html内容本身没有什么好说的，我们主要看看里面的4个文件请求：

- 1、<http://realtime.plhwin.com:3000/socket.io/socket.io.js>
- 2、style.css
- 3、json3.min.js
- 4、client.js

第1个JS是<http://Socket.IO>提供的客户端JS文件，在前面安装服务端的步骤中，当npm安装完<http://socket.io>并搭建起WebServer后，这个JS文件就可以正常访问了。

第2个style.css文件没什么好说的，就是样式文件而已。

第3个JS只在IE8以下版本的IE浏览器中加载，目的是让这些低版本的IE浏览器也能处理json，这是一个开源的JS，详见：[JSON 3](#)

第4个client.js是完整的客户端的业务逻辑实现代码，它的内容如下：

至此所有的编码开发工作全部完成了，在浏览器中打开 <http://demo.plhwin.com/chat/> 就可以看到效果了。

上面所有的客户端和服务端的代码可以从Github上获得，地址：<https://github.com/plhwin/node-is-socketio-chat>

下载本地后有两个文件夹 `client` 和 `server`，`client`文件夹是客户端源码，可以放在Nginx/Apache的WebServer中，也可以放在Node.js的WebServer中。后面的`server`文件夹里的代码是websocket服务端代码，放在Node.js环境中，使用npm安装完 `express` 和 <http://socket.io> 后，`node index.js` 启动后端服务就可以了。

留给我们的思考

1、假设是一个在线客服系统，里面有许多公司使用你的服务，每个公司自己的用户可以通过一个专属URL地址进入该公司的聊天室，聊天是一对一的，每个公司可以新建多个客服人员，每个客服人员可以同时和客户端的多个用户聊天。

2、又假设是一个在线WebIM系统，实现类似微信，qq的功能，客户端可以看到好友在线状态，在线列表，添加好友，删除好友，新建群组等，消息的发送除了支持基本的文字外，还能支持表情、图片和文件。

有兴趣的同学可以继续深入研究。

上面是我前段时间写的一篇与WebSocket这个主题相关的文档，就直接贴过来了，原文请见：[使用Node.js+Socket.IO搭建WebSocket实时应用](#)

加入知乎

与世界分享你的知识、经验和见解

验证码

验证码

注册

已有帐号？[登录](#)

下载知乎 App

关注问题

3568 人关注该问题

换一换

相关问题

- [如何理解 TCP/IP, SPDY, WebSocket 三者之间的关系？](#) 3 个回答
- [如何理解 Web 语义化？](#) 18 个回答
- [目前的 HTML5 开发跟一年前比进展如何，国内国外有没有区别？](#) 16 个回答
- [苹果官网是怎么做到完美保证多平台浏览体验的？](#) 3 个回答
- [现在的页游，一般用到什么样的技术，请系统的说明一下？](#) 6 个回答
- [刘看山](#)
- [移动应用](#)
- [加入知乎](#)
- [知乎协议](#)
- [联系我们](#)