[介绍 2](#_Toc374804365)

[特点 2](#_Toc374804366)

[连接 2](#_Toc374804367)

[连接握手规范 2](#_Toc374804368)

[通信技术比较 4](#_Toc374804369)

[Ajax 4](#_Toc374804370)

[轮询 4](#_Toc374804371)

[长轮询 4](#_Toc374804372)

[流（Comet服务器推送） 5](#_Toc374804373)

[WebSocket 5](#_Toc374804374)

[支持情况 6](#_Toc374804375)

[客户端支持情况 6](#_Toc374804376)

[服务器端支持情况 6](#_Toc374804377)

[后备方案 7](#_Toc374804378)

[基本操作 7](#_Toc374804379)

[局限性 9](#_Toc374804380)

[参考资料 9](#_Toc374804381)

# 介绍

# 特点

* 全双工传输
* 实时通信
* 一次连接，任意通信
* 浏览器原生支持，无须安装插件
* 基于TCP协议
* 支持跨域
* 兼容二进制数据

# 连接

WebSocket 协议本质上是一个基于 TCP 的协议。

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，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，客户端和服务器端的 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。

## 连接握手规范

当打开了到服务器的连接后，客户端必须向服务器发送一次opening握手--它由HTTP Upgrade请求以及一系列必须和可选的头部数据组成。

* 请求必须是Get方法，且HTTP协议版本至少为1.1。
* 必须有Host,Upgrade(值必须包含websocket关键字),Connection(值必须包含"Upgrade")，Sec-WebSocket-Key 请求头。
* 如果请求者是浏览器，必须包含 Origin头。
* 必须包含 Sec-WebSocket-Version头，且其值为13。
* 可以包含 Sec-WebSocket-Protocol以及Sec-WebSocket-Extensions头
* 请求可以包含任何其他的头部域，譬如cookies或认证相关的头如 Authorization。

而后客户端就在等待服务器端的响应并进行验证：

* 确定状态码是不是101，如果不是则遵循HTTP的响应处理过程。否则往下走
* 确保响应头中必须包含Upgrade域，其中包含"websocket"值，否则连接失败；
* 确保响应头包含Connect域，且有值"Upgrade"，否则连接失败；
* 确保响应头包含Sec-WebSocket-Accept域，且其值应该等于Base64(SHA-1( |Sec-WebSocket- Key|)+"
* 258EAFA5-
* E914-47DA-95CA-C5AB0DC85B11")；否则连接失败；
* 如果响应头中包含了 Sec-WebSocket-Extensions域，而请求头并未包含，那么客户端必须中断连接；
* 如果响应头中包含Sec-WebSocket-Protocol域，但在请求头中并没有包含此子协议值，客户端必须中断连接.

如果以上条件俱都满足，那么这个连接就建立了起来，并且处于open状态，可以进行后续的数据发送/接收等操作。

典型的请求头结构如下：

**Http header代码  [收藏代码](javascript:void())**

1. GET /socket.io/1/websocket/1166156958668058955 HTTP/1.1
2. Host: localhost:8077
3. User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:12.0) Gecko/20100101 Firefox/12.0
4. Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8
5. Accept-Language: en-us,en;q=0.5
6. Accept-Encoding: gzip, deflate
7. Connection: keep-alive, Upgrade
8. Sec-WebSocket-Version: 13
9. Origin: http://localhost:8077
10. Sec-WebSocket-Key: xQelQMxXk7J8q1whTHKPJA==
11. Pragma: no-cache
12. Cache-Control: no-cache
13. Upgrade: websocket

 响应头结构如下：

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: V6OpbVjbdllb6RNrRdFmG+pIzXk=

# 通信技术比较

## Ajax

从客户端发出Http请求，服务器端发送Http请求回应相应请求的内容。这种模式通常用于异步加载内容。

轮询（或长轮询）和服务器推送技术都是基于Ajax技术的。

## 轮询

这是最早的一种实现实时 Web 应用的方案。客户端以一定的时间间隔向服务端发出请求，以频繁请求的方式来保持客户端和服务器端的同步。这种同步方案的最大问题是，当客户端以固定频率向服务器发起请求的时候，服务器端的数据可能并没有更新，这样会带来很多无谓的网络传输，所以这是一种非常低效的实时方案。

## 长轮询

长轮询是对定时轮询的改进和提高，目地是为了降低无效的网络传输。当服务器端没有数据更新的时候，连接会保持一段时间周期直到数据或状态改变或者时间过期，通过这种机制来减少无效的客户端和服务器间的交互。当然，如果服务端的数据变更非常频繁的话，这种机制和定时轮询比较起来没有本质上的性能的提高。

## 流（Comet服务器推送）

流技术方案通常就是在客户端的页面使用一个隐藏的窗口向服务端发出一个长连接的请求。服务器端接到这个请求后作出回应并不断更新连接状态以保证客户端和服务器端的连接不过期。通过这种机制可以将服务器端的信息源源不断地推向客户端。

这种机制在用户体验上有一点问题，需要针对不同的浏览器设计不同的方案来改进用户体验；同时这种机制在并发比较大的情况下，对服务器端的资源是一个极大的考验；另外，这种模式不能满足“服务器能主动发出信息，无须客户端发出请求”的应用场合，如即时通信系统、股票报价系统等。

## WebSocket

HTML5 WebSocket 设计出来的目的就是要取代轮询和 Comet 技术，使客户端浏览器具备像 C/S 架构下桌面系统的实时通讯能力。 浏览器通过 JavaScript 向服务器发出建立 WebSocket 连接的请求，连接建立以后，客户端和服务器端就可以通过 TCP 连接直接交换数据。因为 WebSocket 连接本质上就是一个 TCP 连接，所以在数据传输的稳定性和数据传输量的大小方面，和轮询以及 Comet 技术比较，具有很大的性能优势。

在 WebSocket API，浏览器和服务器只需要要做一个握手的动作，然后，浏览器和服务器之间就形成了一条快速通道。两者之间就直接可以数据互相传送。在此WebSocket 协议中，为我们实现即时服务带来了两大好处：

1. Header

互相沟通的Header是很小的-大概只有 2 Bytes

2. Server Push

服务器可以主动传送数据给客户端

Websocket.org 网站对传统的轮询方式和 WebSocket 调用方式作了一个详细的测试和比较，将一个简单的 Web 应用分别用轮询方式和 WebSocket 方式来实现，在这里引用一下他们的测试结果图：

图 1. 轮询和 WebSocket 实现方式的网络负载对比图



通过这张图可以清楚的看出，在流量和负载增大的情况下，WebSocket 方案相比传统的 Ajax 轮询方案有极大的性能优势。这也是为什么我们认为 WebSocket 是未来实时 Web 应用的首选方案的原因。

# 支持情况

## 客户端支持情况

因为是在浏览器中执行，因此浏览器需要支持Http和WebSocket协议：

Firefox 4、Chrome 4、Opera 10.70以及Safari 5，IE10

## 服务器端支持情况

支持Http和WebSocket协议的服务器：

* Java阵营：[Tomcat 7](http://tomcat.apache.org/tomcat-7.0-doc/web-socket-howto.html#Application_development)和[Netty](https://netty.io/)
* .Net阵营：[System.Net.WebSockets](http://msdn.microsoft.com/en-us/library/hh159285%28v=vs.110%29)
* Nodejs阵营：[Socket.io](http://socket.io/)

## 后备方案

对于不支持WebSocket的浏览器（如IE9及以下），可以使用以下技术：

* Flash技术

Flash可以提供一个简单的替换。 使用Flash最明显的缺点是并非所有客户端都安装了Flash，而且某些客户端，如iPhone/iPad，不支持Flash。

* AJAX Long-Polling技术

用AJAX的long-polling来模拟WebSocket在业界已经有一段时间了。它是一个可行的技术，但它不能优化发送的信息。也就是说，它是一个解决方案，但不是最佳的技术方案。

* 使用Socket.IO库
  + 它可以在所有浏览器和移动设备中创建实时的应用程序。通过检测功能来判断是否建立WebSocket连接，或者是AJAX long-polling连接，或Flash等。支持WebSocket, Flash Socket, AJAX long-polling, AJAX multipart streaming, Forever IFrame, JSONP polling。
  + 它的API与WebSocket的API很相似。
  + Socket.IO提供NodeJs的API，可以与NodeJs配合使用。

# 基本操作

客户端

if('WebSocket' in window){

// 创建websocket实例

var socket = new WebSocket('ws://localhost:8080');

//打开

socket.onopen = function(event) {

// 发送

socket.send('I am the client and I\'m listening!');

// 监听

socket.onmessage = function(event) {

console.log('Client received a message',event);

};

// 关闭监听

socket.onclose = function(event) {

console.log('Client notified socket has closed',event);

};

// 关闭

//socket.close()

};

}else{

alert('本浏览器不支持WebSocket哦~');  
}

服务器端：

详见Socket.IO的基本操作。

# 局限性

* WebSocket 规范目前还处于草案阶段，也就是它的规范和 API 还是有变动的可能
* IE9及以下不支持

# 参考资料

[WebSocket API](http://msdn.microsoft.com/zh-cn/library/ie/hh772770.aspx)

[The WebSocket Protocol](http://tools.ietf.org/html/rfc6455#page-10)

[WebSockets浅探及实践 - 1](http://ustczxf.iteye.com/blog/1544895)

[使用 HTML5 WebSocket 构建实时 Web 应用](http://www.ibm.com/developerworks/cn/web/1112_huangxa_websocket/)