Websocke

# 简介

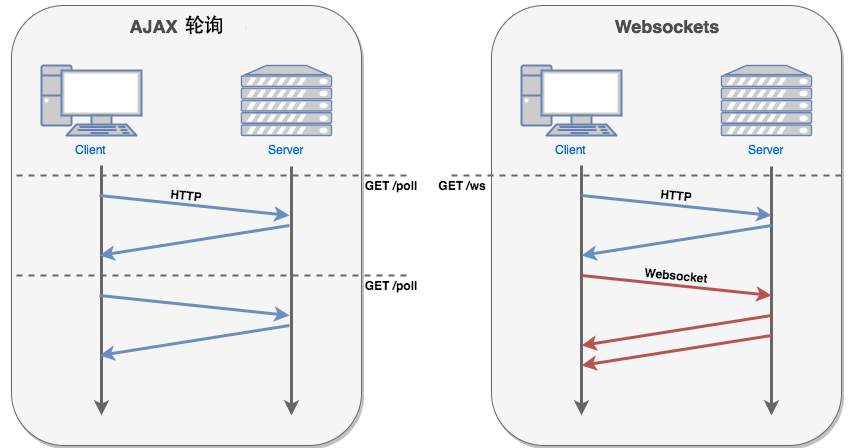
（<https://www.runoob.com/html/html5-websocket.html>）

WebSocket 是 HTML5 开始提供的一种在单个 TCP 连接上进行全双工通讯的协议。

WebSocket 使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在 WebSocket API 中，浏览器和服务器只需要完成一次握手，两者之间就直接可以创建持久性的连接，并进行双向数据传输。

现在，很多网站为了实现推送技术，所用的技术都是 Ajax 轮询。轮询是在特定的的时间间隔（如每1秒），由浏览器对服务器发出HTTP请求，然后由服务器返回最新的数据给客户端的浏览器。这种传统的模式带来很明显的缺点，即浏览器需要不断的向服务器发出请求，然而HTTP请求可能包含较长的头部，其中真正有效的数据可能只是很小的一部分，显然这样会浪费很多的带宽等资源。

HTML5 定义的 WebSocket 协议，能更好的节省服务器资源和带宽，并且能够更实时地进行通讯。



# WebSocket协议

WebSocket并不是全新的协议，而是利用了HTTP协议来建立连接。我们来看看WebSocket连接是如何创建的。

首先，WebSocket连接必须由浏览器发起，因为请求协议是一个标准的HTTP请求，格式如下：

GET ws:*//localhost:3000/ws/chat HTTP/1.1*

Host: localhost

Upgrade: websocket

Connection: Upgrade

Origin: http:*//localhost:3000*

Sec-WebSocket-Key: client-random-string

Sec-WebSocket-Version: 13

该请求和普通的HTTP请求有几点不同：

1. GET请求的地址不是类似/path/，而是以ws://开头的地址；
2. 请求头Upgrade: websocket和Connection: Upgrade表示这个连接将要被转换为WebSocket连接；
3. Sec-WebSocket-Key是用于标识这个连接，并非用于加密数据；
4. Sec-WebSocket-Version指定了WebSocket的协议版本。

随后，服务器如果接受该请求，就会返回如下响应：

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: server-random-string

该响应代码101表示本次连接的HTTP协议即将被更改，更改后的协议就是Upgrade: websocket指定的WebSocket协议。

版本号和子协议规定了双方能理解的数据格式，以及是否支持压缩等等。如果仅使用WebSocket的API，就不需要关心这些。

现在，一个WebSocket连接就建立成功，浏览器和服务器就可以随时主动发送消息给对方。消息有两种，一种是文本，一种是二进制数据。通常，我们可以发送JSON格式的文本，这样，在浏览器处理起来就十分容易。

为什么WebSocket连接可以实现全双工通信而HTTP连接不行呢？实际上HTTP协议是建立在TCP协议之上的，TCP协议本身就实现了全双工通信，但是HTTP协议的请求－应答机制限制了全双工通信。WebSocket连接建立以后，其实只是简单规定了一下：接下来，咱们通信就不使用HTTP协议了，直接互相发数据吧。

安全的WebSocket连接机制和HTTPS类似。首先，浏览器用wss://xxx创建WebSocket连接时，会先通过HTTPS创建安全的连接，然后，该HTTPS连接升级为WebSocket连接，底层通信走的仍然是安全的SSL/TLS协议

# 浏览器

很显然，要支持WebSocket通信，浏览器得支持这个协议，这样才能发出ws://xxx的请求。目前，支持WebSocket的主流浏览器如下：

* Chrome
* Firefox
* IE >= 10
* Sarafi >= 6
* Android >= 4.4
* iOS >= 8

# 服务器

由于WebSocket是一个协议，服务器具体怎么实现，取决于所用编程语言和框架本身。Node.js本身支持的协议包括TCP协议和HTTP协议，要支持WebSocket协议，需要对Node.js提供的HTTPServer做额外的开发。已经有若干基于Node.js的稳定可靠的WebSocket实现，我们直接用npm安装使用即可。

Npm I ws

# 前端websocket的实现

var socket=new WebSocket(url) //创建websocket对象

socket.onopen=function(){ //打开连接，创建与服务的通信

}

socket.onmessage=function(data){ //服务返回的数据

}

socket.onerror=function(){ //通信发生错误

}

socket.onclose=function(){ //连接关闭时触发

}

socket.send() //主动给服务器发送数据

socket.close() //关闭连接

# 后端websocket服务提供

1. 安装cnpm install ws
2. 创建服务

var WebSocketServer = require('ws').Server;//引入websocket

//开启服务9088端口

var wss = new WebSocketServer({

port: 9088

});

//连接通信调用

wss.on('connection', function connection(ws, req) {

//前端请求send调用事件

ws.on('message', function incoming(message) {

//message 前端请求参数

wss.clients.forEach((client) => {

client.send(message)

})

});

})