# Socket通信

## 简介websocket

websocket是HTML5出的东西（协议），是一种全双工通信机制，两端可以及时地互发事件，互发数据，相互通信，只需要浏览器和服务器建立一次连接，服务器就可以主动推送数据到浏览器实现实时数据更新。

### 背景

websocket  的实现需要后端搭建一个WebSocket服务器，但是如果想搭建一个WebSocket服务器就没有那么轻松了，因为WebSocket是一种新的通信协议，目前还是草案，没有成为标准，市场上也没有成熟的WebSocket服务器或者Library实现WebSocket协议，我们就必须自己动手写代码去解析和组装WebSocket的数据包。要这样完成一个WebSocket服务器，估计所有的人都想放弃，幸好的是市场上有几款比较好的开源库供我们使用，比如 PyWebSocket,WebSocket-Node, LibWebSockets等等，这些库文件已经实现了WebSocket数据包的封装和解析，我们可以调用这些接口，这在很大程度上减少了我们的工作量。

## Socket.io

Socket.io是一个实时应用提供跨平台实时通信的库。socket.io 旨在使实时应用在每个浏览器和移动设备上成为可能，模糊不同的传输机制之间的差异。

socket.io 的名字源于它使用了浏览器支持并采用的 HTML5 WebSocket 标准，因为并不是所有的浏览器都支持 WebSocket ，所以该库支持一系列降级功能：

* Websocket
* Adobe® Flash® Socket
* AJAX long polling
* AJAX multipart streaming
* Forever Iframe
* JSONP Polling

在大部分情境下，你都能通过这些功能选择与浏览器保持类似长连接的功能。

* 优点：跨平台、兼容性好、具有降级功能、所有传输机制接口对外统一、自带心跳。
* 缺点：要使用socket.io必须前后端都要用一套框架。
* 适用于：考虑更多兼容性，后端可以使用基于socket.io的框架的情景。（常见服务端实现框架有node.js,Netty-socket.io）

客户端示例代码：

<script src="/socket.io/socket.io.js"></script>

<script>

 var socket = io('http://localhost');

 socket.on('news', function (data) {

   console.log(data);

   socket.emit('my other event', { my: 'data' });

 });

</script>

服务器端示例代码：

var app = require('http').createServer(handler)

var io = require('socket.io')(app);

var fs = require('fs');

app.listen(80);

function handler (req, res) {

 fs.readFile(\_\_dirname + '/index.html',

 function (err, data) {

   if (err) {

     res.writeHead(500);

     return res.end('Error loading index.html');

   }

   res.writeHead(200);

   res.end(data);

 });

}

io.on('connection', function (socket) {

 socket.emit('news', { hello: 'world' });

 socket.on('my other event', function (data) {

   console.log(data);

 });

});

## 客户端核心API

Socket.io所有API详见这儿

### Io.connect建立一个连接

var socket = io.connect('ws://127.0.0.1:3000');

### socket.emit发送一个事件给服务器端（发送数据）

### socket.on 监听一个服务器端emit发送的事件（接收数据）

## （客户端和服务器端）默认事件：

### Connect

### Message

### Disconnect

## 服务器端核心API

### Io.on监听客户端的连接事件

io.on('connection', function (socket) {

}

### Socket.emit发送一个事件给客户端（推送数据）

### Socket.on监听一个客户端emit发送的事件（接收数据）

常用的Node实现有一下几种。

* µWebSockets
* Socket.IO
* WebSocket-Node
* websocketd

## socket中对websocket的使用

### 服务端

io.on(‘connection’,function(socket));//监听客户端连接,回调函数会传递本次连接的socket

　　io.sockets.emit(‘String’,data);//给所有客户端广播消息

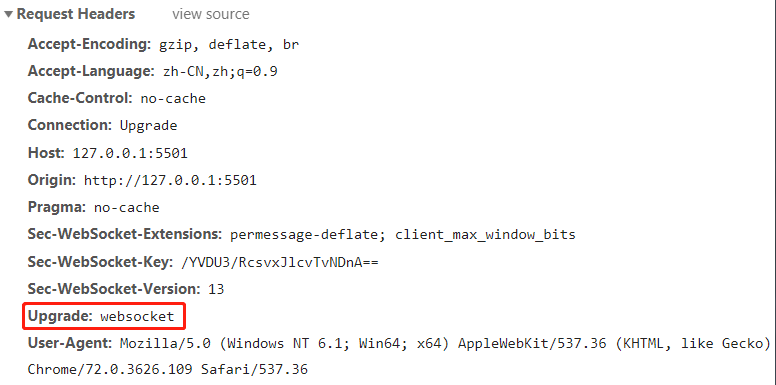
　　io.sockets.socket(socketid).emit(‘String’, data);//给指定的客户端发送消息

　　socket.on(‘String’,function(data));//监听客户端发送的信息

　　socket.emit(‘String’, data);//给该socket的客户端发送消息

## socket机制

* 1. WebSocket连接都始于一个HTTP请求
  2. WebSocket协议在第一次握手连接时，通过HTTP协议在传送WebSocket支持的版本号，协议的字版本号，原始地址，主机地址等等一些列字段给服务器端.



注意：

Upgrade首部，用来把当前的HTTP请求升级到WebSocket协议，这是HTTP协议本身的内容，是为了扩展支持其他的通讯协议。 如果服务器支持新的协议，则必须返回101.

一个WebSocket连接是在客户端与服务器之间HTTP协议的初始握手阶段将其升级到Web Socket协议来建立的，其底层仍是TCP/IP连接

#### Websocket连接与HTTP请求的异同

相同点：

（1）都是建立在TCP之上，通过TCP协议来传输数据。  
（2）都是可靠性传输协议。  
（3）都是应用层协议。

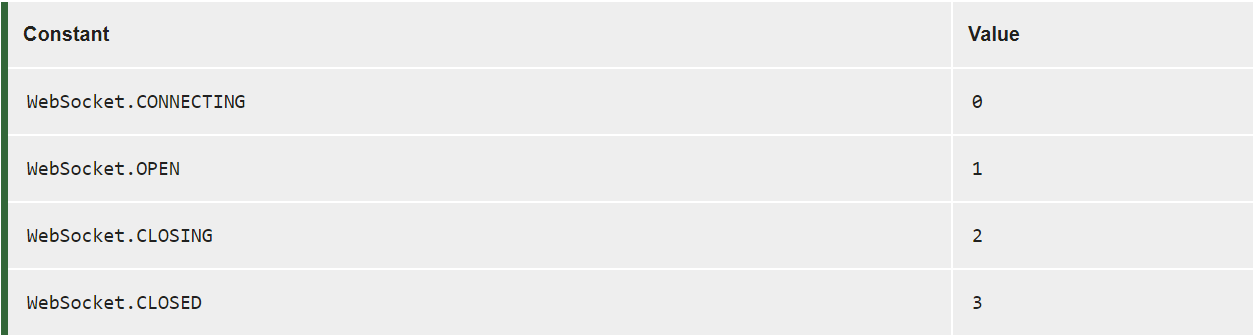
不同点：

（1）WebSocket支持持久连接，HTTP不支持持久连接。

（2）WebSocket是双向通信协议，HTTP是单向协议，只能由客户端发起，做不到服务器主动向客户端推送信息。

## Websocket使用

### 常量



### 属性

[**WebSocket.binaryType**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/binaryType)

使用二进制的数据类型连接。

[**WebSocket.bufferedAmount**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/bufferedAmount)**只读**

未发送至服务器的字节数。

[**WebSocket.extensions**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/extensions)**只读**

服务器选择的扩展。

[**WebSocket.onclose**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/onclose)

用于指定连接关闭后的回调函数。

[**WebSocket.onerror**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/onerror)

用于指定连接失败后的回调函数。

[**WebSocket.onmessage**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/onmessage)

用于指定当从服务器接受到信息时的回调函数

[**WebSocket.onopen**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/onopen)

用于指定连接成功后的回调函数。

[**WebSocket.protocol**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/protocol)**只读**

服务器选择的下属协议。

[**WebSocket.readyState**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/readyState)**只读**

当前的链接状态。

[**WebSocket.url**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/url)**只读**

WebSocket 的绝对路径。

### 方法

[**WebSocket.close([code[, reason]])**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/close)

关闭当前链接。

[**WebSocket.send(data)**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/send)

对要传输的数据进行排队。

### 事件

使用 addEventListener() 或将一个事件监听器赋值给本接口的 oneventname 属性，来监听下面的事件。

**close**

当一个 WebSocket 连接被关闭时触发。  
也可以通过 [onclose](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/onclose" \o "onclose) 属性来设置。

[**error**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/error_event)

当一个 WebSocket 连接因错误而关闭时触发，例如无法发送数据时。  
也可以通过 [onerror](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/onerror" \o "onerror) 属性来设置.

[**message**](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/message_event)

当通过 WebSocket 收到数据时触发。  
也可以通过 [onmessage](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/onmessage" \o "onmessage) 属性来设置。

**open**

当一个 WebSocket 连接成功时触发。  
也可以通过 [onopen](https://developer.mozilla.org/zh-CN/docs/Web/API/WebSocket/onopen" \o "onopen) 属性来设置。

#### 示例

