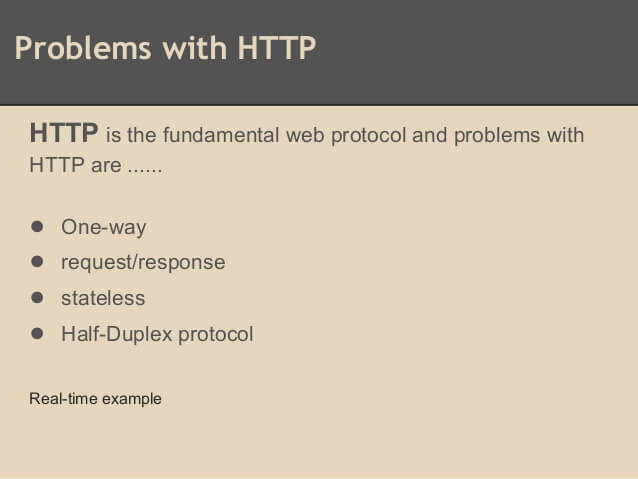
一、为什么需要 WebSocket？

初次接触 WebSocket 的人，都会问同样的问题：我们已经有了 HTTP 协议，为什么还需要另一个协议？它能带来什么好处？

答案很简单，因为 HTTP 协议有一个缺陷：通信只能由客户端发起。

举例来说，我们想了解今天的天气，只能是客户端向服务器发出请求，服务器返回查询结果。HTTP 协议做不到服务器主动向客户端推送信息。



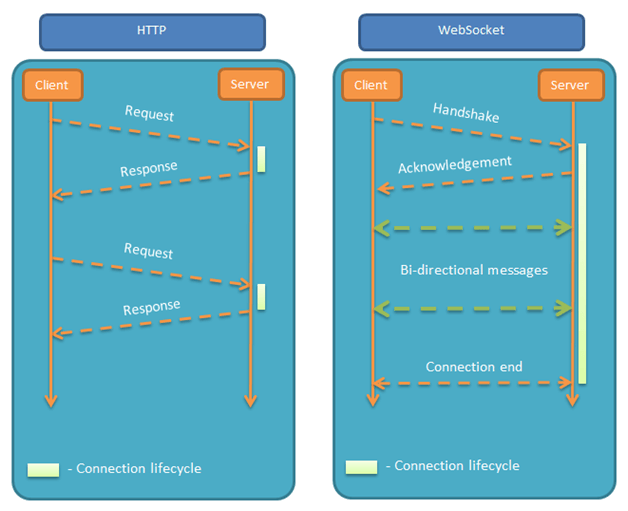
这种单向请求的特点，注定了如果服务器有连续的状态变化，客户端要获知就非常麻烦。我们只能使用["轮询"](https://www.pubnub.com/blog/2014-12-01-http-long-polling/" \t "_blank)：每隔一段时候，就发出一个询问，了解服务器有没有新的信息。最典型的场景就是聊天室。

轮询的效率低，非常浪费资源（因为必须不停连接，或者 HTTP 连接始终打开）。因此，工程师们一直在思考，有没有更好的方法。WebSocket 就是这样发明的。

## 二、简介

WebSocket 协议在2008年诞生，2011年成为国际标准。所有浏览器都已经支持了。

它的最大特点就是，服务器可以主动向客户端推送信息，客户端也可以主动向服务器发送信息，是真正的双向平等对话，属于[服务器推送技术](https://en.wikipedia.org/wiki/Push_technology)的一种。



其他特点包括：

（1）建立在 TCP 协议之上，服务器端的实现比较容易。

（2）与 HTTP 协议有着良好的兼容性。默认端口也是80和443，并且握手阶段采用 HTTP 协议，因此握手时不容易屏蔽，能通过各种 HTTP 代理服务器。

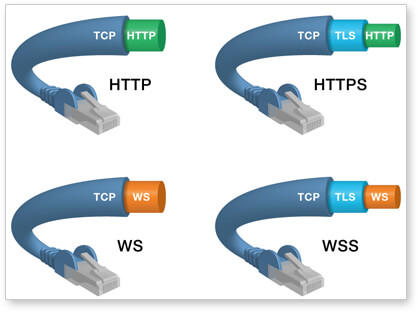
（3）数据格式比较轻量，性能开销小，通信高效。

（4）可以发送文本，也可以发送二进制数据。

（5）没有同源限制，客户端可以与任意服务器通信。

（6）协议标识符是ws（如果加密，则为wss），服务器网址就是 URL。

ws://example.com:80/some/path



## 三、客户端的简单示例

WebSocket 的用法相当简单。

下面是一个网页脚本的例子（点击[这里](http://jsbin.com/muqamiqimu/edit?js,console" \t "_blank)看运行结果），基本上一眼就能明白。

var ws = new WebSocket("[wss://echo.websocket.org](wss://echo.websocket.org/)");

ws.onopen = function(evt) {

console.log("Connection open ...");

ws.send("Hello WebSockets!");

};

ws.onmessage = function(evt) {

console.log( "Received Message: " + evt.data);

ws.close();

};

ws.onclose = function(evt) {

console.log("Connection closed.");

};

## 四、客户端的 API

WebSocket 客户端的 API 如下。

### 4.1 WebSocket 构造函数

WebSocket 对象作为一个构造函数，用于新建 WebSocket 实例。

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

执行上面语句之后，客户端就会与服务器进行连接。

实例对象的所有属性和方法清单，参见[这里](https://developer.mozilla.org/en-US/docs/Web/API/WebSocket" \t "_blank)。

### 4.2 webSocket.readyState

readyState属性返回实例对象的当前状态，共有四种。

* CONNECTING：值为0，表示正在连接。
* OPEN：值为1，表示连接成功，可以通信了。
* CLOSING：值为2，表示连接正在关闭。
* CLOSED：值为3，表示连接已经关闭，或者打开连接失败。
* 下面是一个示例。
* switch (ws.readyState) {
* case WebSocket.CONNECTING:
* // do something
* break;
* case WebSocket.OPEN:
* // do something
* break;
* case WebSocket.CLOSING:
* // do something
* break;
* case WebSocket.CLOSED:
* // do something
* break;
* default:
* // this never happens
* break;
* }

### 4.3 webSocket.onopen

实例对象的onopen属性，用于指定连接成功后的回调函数。

ws.onopen = function () {

ws.send('Hello Server!');

}

如果要指定多个回调函数，可以使用addEventListener方法。

ws.addEventListener('open', function (event) {

ws.send('Hello Server!');

});

### 4.4 webSocket.onclose

实例对象的onclose属性，用于指定连接关闭后的回调函数。

ws.onclose = function(event) {

var code = event.code;

var reason = event.reason;

var wasClean = event.wasClean;

// handle close event

};

ws.addEventListener("close", function(event) {

var code = event.code;

var reason = event.reason;

var wasClean = event.wasClean;

// handle close event

});

### 4.5 webSocket.onmessage

实例对象的onmessage属性，用于指定收到服务器数据后的回调函数。

ws.onmessage = function(event) {

var data = event.data;

// 处理数据

};

ws.addEventListener("message", function(event) {

var data = event.data;

// 处理数据

});

注意，服务器数据可能是文本，也可能是二进制数据（blob对象或Arraybuffer对象）。

ws.onmessage = function(event){

if(typeof event.data === String) {

console.log("Received data string");

}

if(event.data instanceof ArrayBuffer){

var buffer = event.data;

console.log("Received arraybuffer");

}

}

除了动态判断收到的数据类型，也可以使用binaryType属性，显式指定收到的二进制数据类型。

// 收到的是 blob 数据

ws.binaryType = "blob";

ws.onmessage = function(e) {

console.log(e.data.size);

};

// 收到的是 ArrayBuffer 数据

ws.binaryType = "arraybuffer";

ws.onmessage = function(e) {

console.log(e.data.byteLength);

};

### 4.6 webSocket.send()

实例对象的send()方法用于向服务器发送数据。

发送文本的例子。

ws.send('your message');

发送 Blob 对象的例子。

var file = document

.querySelector('input[type="file"]')

.files[0];

ws.send(file);

发送 ArrayBuffer 对象的例子。

// Sending canvas ImageData as ArrayBuffer

var img = canvas\_context.getImageData(0, 0, 400, 320);

var binary = new Uint8Array(img.data.length);

for (var i = 0; i < img.data.length; i++) {

binary[i] = img.data[i];

}

ws.send(binary.buffer);

### 4.7 webSocket.bufferedAmount

实例对象的bufferedAmount属性，表示还有多少字节的二进制数据没有发送出去。它可以用来判断发送是否结束。

var data = new ArrayBuffer(10000000);

socket.send(data);

if (socket.bufferedAmount === 0) {

// 发送完毕

} else {

// 发送还没结束

}

### 4.8 webSocket.onerror

实例对象的onerror属性，用于指定报错时的回调函数。

socket.onerror = function(event) {

// handle error event

};

socket.addEventListener("error", function(event) {

// handle error event

});

五、服务端的实现

WebSocket 服务器的实现，可以查看维基百科的[列表](https://en.wikipedia.org/wiki/Comparison_of_WebSocket_implementations" \t "_blank)。

常用的 Node 实现有以下三种。

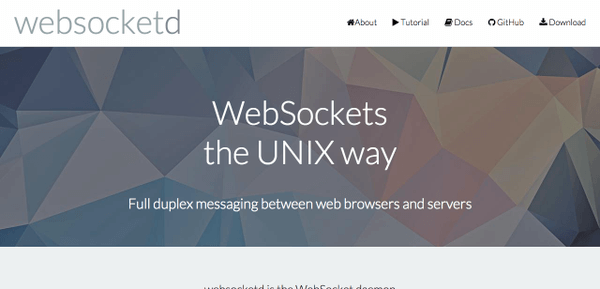
* [µWebSockets](https://github.com/uWebSockets/uWebSockets)
* [Socket.IO](http://socket.io/)
* [WebSocket-Node](https://github.com/theturtle32/WebSocket-Node)

具体的用法请查看它们的文档，这里不详细介绍了。

六、WebSocketd

下面，我要推荐一款非常特别的 WebSocket 服务器：[Websocketd](http://websocketd.com/" \t "_blank)。

它的最大特点，就是后台脚本不限语言，标准输入（stdin）就是 WebSocket 的输入，标准输出（stdout）就是 WebSocket 的输出。



举例来说，下面是一个 Bash 脚本counter.sh。

#!/bin/bash

echo 1

sleep 1

echo 2

sleep 1

echo 3

命令行下运行这个脚本，会输出1、2、3，每个值之间间隔1秒。

$ bash ./counter.sh

1

2

3

现在，启动websocketd，指定这个脚本作为服务。

$ websocketd --port=8080 bash ./counter.sh

上面的命令会启动一个 WebSocket 服务器，端口是8080。每当客户端连接这个服务器，就会执行counter.sh脚本，并将它的输出推送给客户端。

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

ws.onmessage = function(event) {

console.log(event.data);

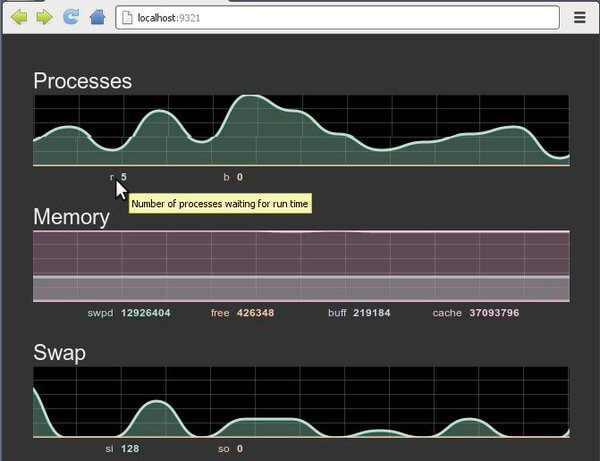
};

上面是客户端的 JavaScript 代码，运行之后会在控制台依次输出1、2、3。

有了它，就可以很方便地将命令行的输出，发给浏览器。

$ websocketd --port=8080 ls

上面的命令会执行ls命令，从而将当前目录的内容，发给浏览器。使用这种方式实时监控服务器，简直是轻而易举（[代码](https://github.com/joewalnes/web-vmstats" \t "_blank)）。



更多的用法可以参考[官方示例](https://github.com/joewalnes/websocketd/tree/master/examples/bash)。

* Bash 脚本[读取客户端输入](https://github.com/joewalnes/websocketd/blob/master/examples/bash/greeter.sh" \t "_blank)的例子
* 五行代码实现一个最简单的[聊天服务器](https://github.com/joewalnes/websocketd/blob/master/examples/bash/chat.sh" \t "_blank)



websocketd 的实质，就是命令行的 WebSocket 代理。只要命令行可以执行的程序，都可以通过它与浏览器进行 WebSocket 通信。下面是一个 Node 实现的回声服务[greeter.js](https://github.com/joewalnes/websocketd/blob/master/examples/nodejs/greeter.js" \t "_blank)。

process.stdin.setEncoding('utf8');

process.stdin.on('readable', function() {

var chunk = process.stdin.read();

if (chunk !== null) {

process.stdout.write('data: ' + chunk);

}

});

启动这个脚本的命令如下。

$ websocketd --port=8080 node ./greeter.js

官方仓库还有其他[各种语言](https://github.com/joewalnes/websocketd/tree/master/examples)的例子。

七、参考链接

* [How to Use WebSockets](http://cjihrig.com/blog/how-to-use-websockets/)
* [WebSockets - Send & Receive Messages](https://www.tutorialspoint.com/websockets/websockets_send_receive_messages.htm)
* [Introducing WebSockets: Bringing Sockets to the Web](https://www.html5rocks.com/en/tutorials/websockets/basics/)

（完）