

WebSocket 教程

作者： 阮一峰

日期： 2017年5月15日



本站由 珠峰培训（专业前端培训）独家赞助

[WebSocket](#) 是一种网络通信协议，很多高级功能都需要它。

本文介绍 WebSocket 协议的使用方法。



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。

Problems with HTTP

HTTP is the fundamental web protocol and problems with HTTP are

- One-way
- request/response
- stateless
- Half-Duplex protocol

Real-time example

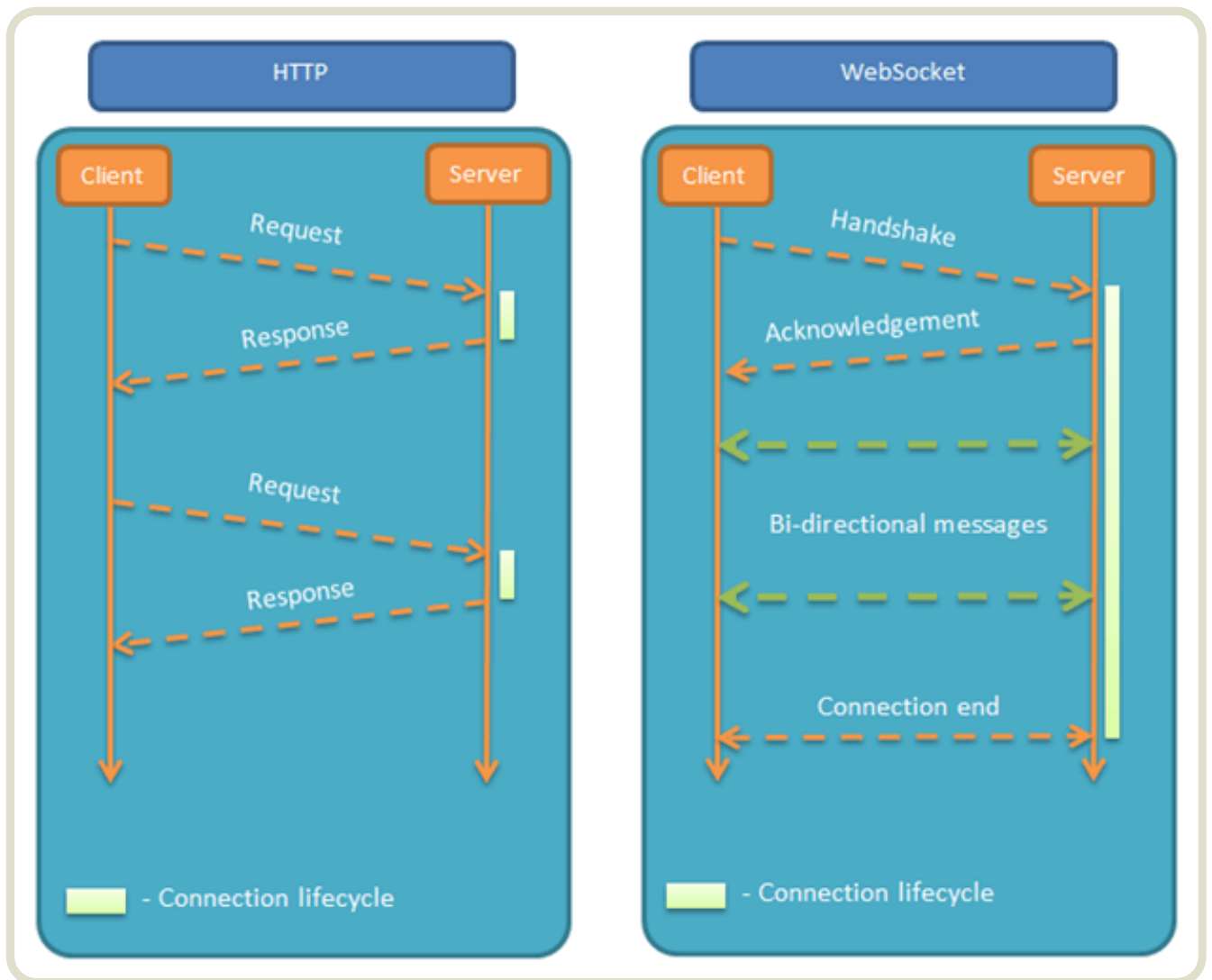
这种单向请求的特点，注定了如果服务器有连续的状态变化，客户端要获知就非常麻烦。我们只能使用"[轮询](#)"：每隔一段时候，就发出一个询问，了解服务器有没有新的信息。最典型的场景就是聊天室。

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。

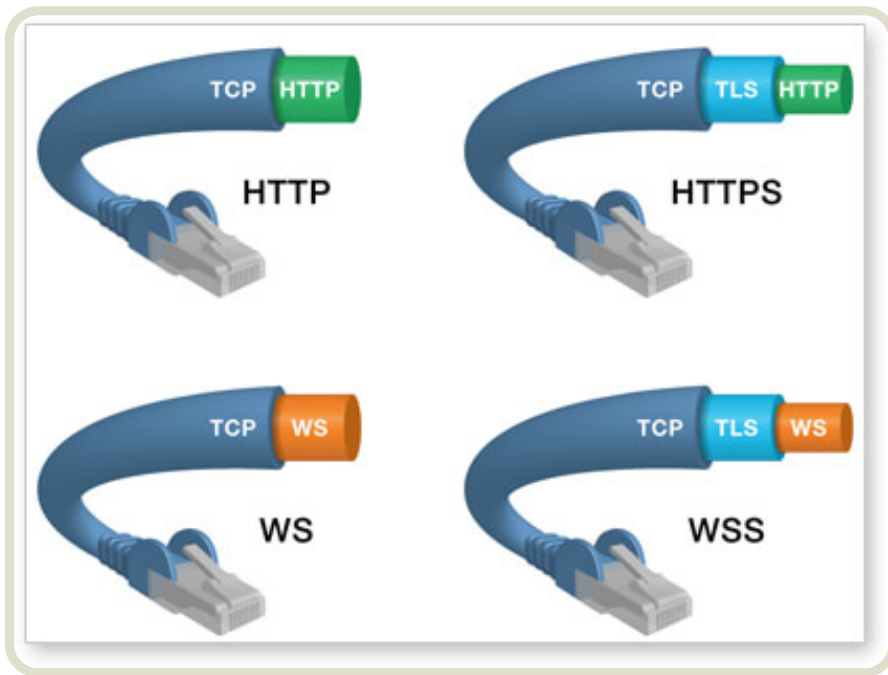
它的最大特点就是，服务器可以主动向客户端推送信息，客户端也可以主动向服务器发送信息，是真正的双向平等对话，属于[服务器推送技术](#)的一种。



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](#)看运行结果），基本上一眼就能明白。

```
var ws = new WebSocket("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](#)。

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](#)。

常用的 Node 实现有以下三种。

- [uWebSockets](#)
- [Socket.IO](#)
- [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](#)。

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](#)）。



更多的用法可以参考[官方示例](#)。

- Bash 脚本[读取客户端输入](#)的例子
- 五行代码实现一个最简单的[聊天服务器](#)

Please enter your name:

[Mon May 15 08:13:51 CST 2017] Welcome to the chat 老张!

[Mon May 15 08:13:52 CST 2017] 老张> 早上好

[2017年 05月 15日 星期一 08:14:05 CST] 老李 joined the chat

[2017年 05月 15日 星期一 08:14:11 CST] 老李> 天气不错啊

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`。

```
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](#)的例子。

七、参考链接

- [How to Use WebSockets](#)
- [WebSockets - Send & Receive Messages](#)
- [Introducing WebSockets: Bringing Sockets to the Web](#)

(完)

文档信息

- 版权声明：自由转载-非商用-非衍生-保持署名（创意共享3.0许可证）
- 发表日期：2017年5月15日
- 更多内容：档案 » 开发者手册
- 文集：《前方的路》，《未来世界的幸存者》
- 社交媒体： twitter,  weibo



京程一灯
www.yidengxuetang.com

育精英前端 冲月薪3万

阿里巴巴 Alibaba.com Tencent 腾讯 Bai 百度



QCon [北京站·2018]
全球软件开发大会

跻身顶级程序员所需要的全局视野

100+海内外实践先驱 20+领域的关键落地经验

北京·国际会议中心
会议：04月20-22日 / 培训：04月23-24日

立即了解

相关文章

- **2018.03.05:** [HTTP/2 服务器推送 \(Server Push\) 教程](#)

HTTP/2 协议的主要目的是提高网页性能。

■ **2018.02.27:** [Nginx 容器教程](#)

春节前，我看到 Nginx 加入了 HTTP/2 的 server push 功能，就很想试一下。

■ **2018.02.13:** [Docker 微服务教程](#)

Docker 是一个容器工具，提供虚拟环境。很多人认为，它改变了我们对软件的认识。

■ **2018.02.09:** [Docker 入门教程](#)

2013年发布至今，Docker 一直广受瞩目，被认为可能会改变软件行业。

广告（购买广告位）



CODING
CLOUD DEVELOPMENT

体验极速
代码托管服务

—
3月31日前，点击注册并激活
立赠30天付费会员



优达学城
UDACITY

× 微信 腾讯微信官方合作课程

微信小程序开发

4周入门实战

免费试听 >

2018 © 我的邮件 | 微博 | 推特 | GitHub