Websocket协议的简单介绍

Websocket是一种在单个TCP连接上进行全双工通讯的协议，双工（duplex）是指两台通讯设备之间，允许有双向的资料传输。全双工的是指，允许两台设备间同时进行双向资料传输。这是相对于半双工来说的，半双工不能同时进行双向传输，这期间的区别相当于手机和对讲机的区别，手机在讲话的同时也能听到对方说话，对讲机只能一个说完另一个才能说。

长话短说，在Websocket协议中，客户端和服务端只需要做一个握手的动作，就能形成一条通道，两者之间可以进行数据互相传送。

所以WebSocket协议分为两部分：

握手

数据传输

**握手**

客户端发送一个请求

GET / HTTP/1.1

Upgrade: websocket

Connection: Upgrade

Host: example.com

Origin: null

Sec-WebSocket-Key: sN9cRrP/n9NdMgdcy2VJFQ==

Sec-WebSocket-Version: 13

服务器回应

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: fFBooB7FAkLlXgRSz0BT3v4hq5s=

Sec-WebSocket-Origin: null

Sec-WebSocket-Location: ws://example.com/

握手时，客户端发送一个随机的Sec-WebSocket-Key，服务端根据这个key做一些处理，返回一个Sec-WebSocket-Accept的值给客户端，具体的原理在后面的文章中再具体说。

**数据传输**

这是Websocket的数据传输协议，聊天信息一般会按照这个协议的规则来传输，下图中的一整个东西称为一个数据帧，数据帧的成帧和解析是处理这个协议时最麻烦的一部分了。具体这个表怎么看可以参照

0 1 2 3

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

+-+-+-+-+-------+-+-------------+-------------------------------+

|F|R|R|R| opcode|M| Payload len | Extended payload length |

|I|S|S|S| (4) |A| (7) | (16/64) |

|N|V|V|V| |S| | (if payload len==126/127) |

| |1|2|3| |K| | |

+-+-+-+-+-------+-+-------------+ - - - - - - - - - - - - - - - +

| Extended payload length continued, if payload len == 127 |

+ - - - - - - - - - - - - - - - +-------------------------------+

| |Masking-key, if MASK set to 1 |

+-------------------------------+-------------------------------+

| Masking-key (continued) | Payload Data |

+-------------------------------- - - - - - - - - - - - - - - - +

: Payload Data continued ... :

+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +

| Payload Data continued ... |

+---------------------------------------------------------------+

具体每一bit的意思

FIN 1bit 表示信息的最后一帧

RSV 1-3 1bit each 以后备用的 默认都为 0

Opcode 4bit 帧类型，稍后细说

Mask 1bit 掩码，是否加密数据，默认必须置为1 （这里很蛋疼）

Payload 7bit 数据的长度

Masking-key 1 or 4 bit 掩码

Payload data (x + y) bytes 数据

Extension data x bytes 扩展数据

Application data y bytes 程序数据

协议的封装与传输

**1.握手协议的封装与传输**

**Handshake类是根据请求头**

GET / HTTP/1.1

Upgrade: websocket

Connection: Upgrade

Host: example.com

Origin: null

Sec-WebSocket-Key: sN9cRrP/n9NdMgdcy2VJFQ==

Sec-WebSocket-Version: 13

来封装的，由于这个请求头的字段顺序是随便的，我们可以用一个map来存储，发送消息时再写入Socket的输出流

下面是Handshakedata类，为了文章易读，简化了代码

public class Handshakedata

{

private byte[] content; //请求体，在握手协议中一般为空

private TreeMap<String, String> map; //用于存储请求头

}

初始化握手请求头，为了代码容易理解，稍微对Java-Websocket中的代码作了修改。

public Handshakedata postProcessHandshakeRequestAsClient(Handshakedata request)

{

request.put("Upgrade", "websocket");

request.put("Connection", "Upgrade");

request.put("Sec-WebSocket-Version", "8");

byte[] random = new byte[16];

this.reuseableRandom.nextBytes(random); //生成一个随机的Sec-WebSocket-Key

request.put("Sec-WebSocket-Key", Base64.encodeBytes(random));

return request;

}

生成数据帧，由于是通过Socket传输消息，最终传输的内容要写入到Socket的OutputStream中，需要一个把握手消息转换成bytebuffer的方法，再通过这个bytebuffer写入流中

public ByteBuffer createHandshake(Handshakedata handshakedata) {

StringBuilder bui = new StringBuilder(100);

bui.append("GET ");

bui.append(handshakedata.getResourceDescriptor());

bui.append(" HTTP/1.1");

bui.append("\r\n");

Iterator it = handshakedata.iterateHttpFields();

while (it.hasNext()) {

String fieldname = (String)it.next();

String fieldvalue = handshakedata.getFieldValue(fieldname);

bui.append(fieldname);

bui.append(": ");

bui.append(fieldvalue);

bui.append("\r\n");

}

bui.append("\r\n");

byte[] httpheader = Charsetfunctions.asciiBytes(bui.toString());

byte[] content = withcontent ? handshakedata.getContent() : null;

ByteBuffer bytebuffer = ByteBuffer.allocate((content == null ? 0 : content.length) + httpheader.length);

bytebuffer.put(httpheader);

bytebuffer.flip();

return bytebuffer;

}

最后写入Socket的流中

ByteBuffer buffer = (ByteBuffer)WebSocketClient.this.engine.outQueue.take(); //从消息队列中取出刚才转换好的bytebuffer

WebSocketClient.this.ostream.write(buffer.array(), 0, buffer.limit()); //this.ostream = this.socket.getOutputStream() Socket的输出流

WebSocketClient.this.ostream.flush(); //刷新，发送消息

以上是客户端发送握手协议的过程。

客户端接收服务端回应

服务端接收到客户端的握手请求后，需要返回响应

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: fFBooB7FAkLlXgRSz0BT3v4hq5s=

Sec-WebSocket-Origin: null

Sec-WebSocket-Location: ws://example.com/

收到这一段响应后，**客户端需要比对Sec-WebSocket-Accept值，这个值表示服务器同意握手建立连接，是客户端传输过来的Sec-WebSocket-Key跟“258EAFA5-E914-47DA-95CA-C5AB0DC85B11”拼接后，用SHA-1加密，并进行BASE-64编码得来的。**

客户端收到Sec-WebSocket-Accept后，将本地的Sec-WebSocket-Key进行同样的编码，然后比对。

public Draft.HandshakeState acceptHandshakeAsClient(ClientHandshake request, ServerHandshake response)

throws InvalidHandshakeException

{

if ((!request.hasFieldValue("Sec-WebSocket-Key")) || (!response.hasFieldValue("Sec-WebSocket-Accept"))) {

return Draft.HandshakeState.NOT\_MATCHED;

}

//Sec-WebSocket-Key和Sec-WebSocket-Accept进行比对

String seckey\_answere = response.getFieldValue("Sec-WebSocket-Accept");

String seckey\_challenge = request.getFieldValue("Sec-WebSocket-Key");

seckey\_challenge = generateFinalKey(seckey\_challenge);

if (seckey\_challenge.equals(seckey\_answere))

return Draft.HandshakeState.MATCHED;

return Draft.HandshakeState.NOT\_MATCHED;

}

//产生Sec-WebSocket-Accept的方法

private String generateFinalKey(String in) {

String seckey = in.trim();

String acc = seckey + "258EAFA5-E914-47DA-95CA-C5AB0DC85B11";

MessageDigest sh1;

try {

sh1 = MessageDigest.getInstance("SHA1");

} catch (NoSuchAlgorithmException e) {

throw new RuntimeException(e);

}

return Base64.encodeBytes(sh1.digest(acc.getBytes()));

}

**首先Websocket是基于TCP协议的，只是借用了HTTP的协议来完成一部分握手。**

**在握手阶段是一样的**

首先我们来看个典型的Websocket握手（借用Wikipedia的。。）

GET /chat HTTP/1.1

Host: server.example.com

**Upgrade: websocket**

**Connection: Upgrade**

**Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==**

**Sec-WebSocket-Protocol: chat, superchat**

**Sec-WebSocket-Version: 13**

Origin: http://example.com

熟悉HTTP的童鞋可能发现了，这段类似HTTP协议的握手请求中，多了几个东西。  
我会顺便讲解下作用。

Upgrade: websocket

Connection: Upgrade

这个就是Websocket的核心了，告诉Apache、Nginx等服务器：**注意啦，窝发起的是Websocket协议，快点帮我找到对应的助理处理~不是那个老土的HTTP。**

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

首先，Sec-WebSocket-Key 是一个Base64 encode的值，这个是浏览器随机生成的，告诉服务器：**泥煤，不要忽悠窝，我要验证尼是不是真的是Websocket助理。**  
然后，Sec\_WebSocket-Protocol 是一个用户定义的字符串，用来区分同URL下，不同的服务所需要的协议。简单理解：**今晚我要服务A，别搞错啦~**  
最后，Sec-WebSocket-Version 是告诉服务器所使用的Websocket Draft（协议版本），在最初的时候，Websocket协议还在 Draft 阶段，各种奇奇怪怪的协议都有，而且还有很多期奇奇怪怪不同的东西，什么Firefox和Chrome用的不是一个版本之类的，当初Websocket协议太多可是一个大难题。。不过现在还好，已经定下来啦~大家都使用的一个东西~ 脱水：**服务员，我要的是13岁的噢→\_→**  
  
然后服务器会返回下列东西，表示已经接受到请求， 成功建立Websocket啦！

HTTP/1.1 101 Switching Protocols

**Upgrade: websocket**

**Connection: Upgrade**

**Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=**

**Sec-WebSocket-Protocol: chat**

这里开始就是HTTP最后负责的区域了，告诉客户，我已经成功切换协议啦~

Upgrade: websocket

Connection: Upgrade

依然是固定的，告诉客户端即将升级的是Websocket协议，而不是mozillasocket，lurnarsocket或者shitsocket。  
然后，Sec-WebSocket-Accept 这个则是经过服务器确认，并且加密过后的 Sec-WebSocket-Key。服务器：**好啦好啦，知道啦，给你看我的ID CARD来证明行了吧。。**  
后面的，**Sec-WebSocket-Protocol 则是表示最终使用的协议。**  
  
至此，HTTP已经完成它所有工作了，接下来就是完全按照Websocket协议进行了。  
具体的协议就不在这阐述了。

**WebSocket 是一种协议，基于 TCP 协议；HTTP 也是一种协议，基于 TCP 协议。  
连接要保持还是关闭是由你服务器应用来控制的。**  
**WebSocket 协议和 HTTP 协议是两种不同的东西，它们扯上关系是只是因为：  
客户端开始建立 WebSocket 连接时要发送一个 header 标记了 Upgrade 的 HTTP 请求，表示请求协议升级。**所以服务器端做出响应的简便方法是，直接在现有的 HTTP 服务器软件和现有的端口上实现 WebSocket 协议，重用现有代码（比如解析和认证这个 HTTP 请求。如果在 TCP 协议上实现，这两个功能就要重新实现），**然后再回一个状态码为 101 的 HTTP 响应完成握手，再往后发送数据时就没 HTTP 的事了。**

**websocket约定了一个通信的规范，通过一个握手的机制，客户端（浏览器）和服务器（web server）之间能建立一个类似tcp的连接，从而方便c－s之间的通信。**

**在websocket出现之前，web交互一般是基于http协议的短连接或者长连接。**

[博客首页](http://www.qixing318.com/blog/) /[HTML5](http://www.qixing318.com/blog/category/HTML5" \o "HTML5)/[WebSocket](http://www.qixing318.com/blog/category/WebSocket" \o "WebSocket)

[顶](http://www.qixing318.com/article/643129914.html" \o "顶一下)

12

# **分析HTML5中WebSocket的原理**

2013-12-25  分类： [HTML5](http://www.qixing318.com/blog/category/HTML5" \o "HTML5) 、[WebSocket](http://www.qixing318.com/blog/category/WebSocket" \o "WebSocket) [0条评论](http://www.qixing318.com/article/643129914.html" \l "article-comments" \o ") 浏览11581次     分享到 [分享到豆瓣](http://www.qixing318.com/article/643129914.html" \l "douban" \o "分享到豆瓣)[分享到新浪微博](http://www.qixing318.com/article/643129914.html" \l "sina" \o "分享到新浪微博)[分享到QQ好友](http://www.qixing318.com/article/643129914.html" \l "tencent" \o "分享到QQ好友)[分享到腾讯微博](http://www.qixing318.com/article/643129914.html" \l "qq" \o "分享到腾讯微博)[分享到人人网](http://www.qixing318.com/article/643129914.html" \l "renren" \o "分享到人人网)[分享到搜狐微博](http://www.qixing318.com/article/643129914.html" \l "sohu" \o "分享到搜狐微博)

目录结构

# **一、什么是Websocket**

**websocket是html5提出的一个协议规范**，参考[rfc6455](http://tools.ietf.org/html/rfc6455" \t "http://www.qixing318.com/article/_blank). 不过目前还都是在草案，没有成为标准，毕竟html5还在路上。

websocket约定了一个通信的规范，通过一个握手的机制，客户端（浏览器）和服务器（web server）之间能建立一个类似tcp的连接，从而方便c－s之间的通信。

在websocket出现之前，web交互一般是基于http协议的短连接或者长连接。

## 短连接的过程大概有下面几个步骤：

（1）建立tcp连接；

（2）浏览器发出http请求；

（3）web server应答；

（4）断开tcp连接；

优点是简洁明了，缺点也很明显，每一次的交互中，建立和断开tcp连接带来了比较大的开销，而且http协议头比较长，也会带来带宽的浪费。

通过设置http头中的keep－alive域可以实现http长连接，避免了建立和断开连接的开销，但是http协议头的问题仍然无法解决。

除此之外，web server主动向浏览器推送数据的处理会比较麻烦，要么是浏览器发起轮询（毫无疑问，这是一个低效的方式），或者利用comet技术，比较复杂，而且这已经不是在html层面上解决问题了。

而web socket的出现，解决了上面描述的问题。

# **二、Websocket握手协议**

websocket是一种全新的协议，不属于http无状态协议，协议名为"ws"，这意味着一个websocket连接地址会是这样的写法：ws://\*\*。

websocket协议本质上是一个基于tcp的协议。建立连接需要握手，客户端（浏览器）首先向服务器（web server）发起一条特殊的http请求，web server解析后生成应答到浏览器，这样子一个websocket连接就建立了，直到某一方关闭连接。

由于是草案的原因，前前后后就出现了多个版本的握手协议，分情况说明一下：

## （1） 基于flash的握手协议

使用场景是IE的多数版本，因为IE的多数版本不都不支持WebSocket协议，以及FF、CHROME等浏览器的低版本，还没有原生的支持WebSocket。此处，server唯一要做的，就是准备一个WebSocket-Location域给client，没有加密，可靠性很差。

### 客户端请求：

GET /ls HTTP/1.1

Upgrade: WebSocket

Connection: Upgrade

Host: www.qixing318.com

Origin: http://www.qixing318.com

### 服务器返回：

HTTP/1.1 101 Web Socket Protocol Handshake

Upgrade: WebSocket

Connection: Upgrade

WebSocket-Origin: http://www.qixing318.com

WebSocket-Location: ws://www.qixing318.com/ls

## （2）基于md5加密方式的握手协议

### 客户端请求：

GET /demo HTTP/1.1

Host: example.com

Connection: Upgrade

Sec-WebSocket-Key2: \*\*

Upgrade: WebSocket

Sec-WebSocket-Key1: \*\*

Origin: http://www.qixing318.com

[8-byte security key]

服务端返回：

HTTP/1.1 101 WebSocket Protocol Handshake

Upgrade: WebSocket

Connection: Upgrade

WebSocket-Origin: http://www.qixing318.com

WebSocket-Location: ws://example.com/demo

[16-byte hash response]

其中 Sec-WebSocket-Key1，Sec-WebSocket-Key2 和 [8-byte security key] 这几个头信息是web server用来生成应答信息的来源，依据 draft-hixie-thewebsocketprotocol-76 草案的定义。

web server基于以下的算法来产生正确的应答信息：

1）逐个字符读取 Sec-WebSocket-Key1 头信息中的值，将数值型字符连接到一起放到一个临时字符串里，同时统计所有空格的数量；

2）将在第（1）步里生成的数字字符串转换成一个整型数字，然后除以第（1）步里统计出来的空格数量，将得到的浮点数转换成整数型；

3）将第（2）步里生成的整型值转换为符合网络传输的网络字节数组；

4）对 Sec-WebSocket-Key2 头信息同样进行第（1）到第（3）步的操作，得到另外一个网络字节数组；

5）将 [8-byte security key] 和在第（3）、（4）步里生成的网络字节数组合并成一个16字节的数组；

6）对第（5）步生成的字节数组使用MD5算法生成一个哈希值，这个哈希值就作为安全密钥返回给客户端，以表明服务器端获取了客户端的请求，同意创建websocket连接

## （3） 基于sha加密方式的握手协议

**也是目前见的最多的一种方式**，这里的版本号目前是需要13以上的版本。

### 客户端请求：

GET /ls HTTP/1.1

Upgrade: websocket

Connection: Upgrade

Host: www.qixing318.com

Sec-WebSocket-Origin: http://www.qixing318.com

Sec-WebSocket-Key: 2SCVXUeP9cTjV+0mWB8J6A==

Sec-WebSocket-Version: 13

服务器返回：

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: mLDKNeBNWz6T9SxU+o0Fy/HgeSw=

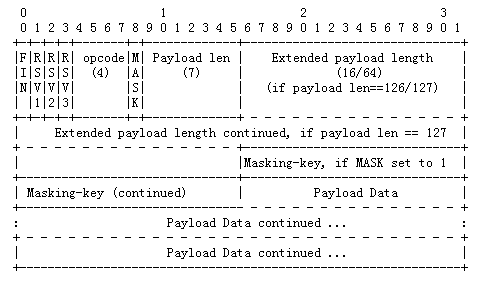
这个的原理，就是把客户端上报的key拼上一段GUID “258EAFA5-E914-47DA-95CA-C5AB0DC85B11″，拿这个字符串做SHA-1 hash计算，然后再把得到的结果通过base64加密，最后在返回给客户端。

# **三、Websocket数据帧**

**WebScoket协议中，数据以帧序列的形式传输**，具体的协议标准可以参考rfc6455

（1）**客户端向服务器传输的数据帧必须进行掩码处理：服务器若接收到未经过掩码处理的数据帧，则必须主动关闭连接。**

（2）**服务器向客户端传输的数据帧一定不能进行掩码处理。客户端若接收到经过掩码处理的数据帧，则必须主动关闭连接。**针对上情况，发现错误的一方可向对方发送close帧（状态码是1002，表示协议错误），以关闭连接。



### FIN：1位

表示这是消息的最后一帧（结束帧），一个消息由一个或多个数据帧构成。若消息由一帧构成，起始帧即结束帧。

RSV1，RSV2，RSV3：各1位

MUST be 0 unless an extension is negotiated that defines meanings for non-zero values.

If a nonzero value is received and none of the negotiated extensions defines the meaning of such a nonzero value, the receiving endpoint MUST \_Fail the WebSocket Connection\_

大致意思是如果未定义扩展，各位是0；如果定义了扩展，即为非0值。如果接收的帧此处非0，扩展中却没有该值的定义，那么关闭连接。

### OPCODE：4位

解释PayloadData，如果接收到未知的opcode，接收端必须关闭连接。

0×0表示附加数据帧

0×1表示文本数据帧

0×2表示二进制数据帧

0×3-7暂时无定义，为以后的非控制帧保留

0×8表示连接关闭

0×9表示ping

0xA表示pong

0xB-F暂时无定义，为以后的控制帧保留

### MASK：1位

用于标识PayloadData是否经过掩码处理。如果是1，Masking-key域的数据即是掩码密钥，用于解码PayloadData。客户端发出的数据帧需要进行掩码处理，所以此位是1。

### Payload length：7位，7+16位，7+64位, PayloadData的长度（以字节为单位）。

(1) 如果其值在0-125，则是payload的真实长度。

(2) 如果值是126，则后面2个字节形成的16位无符号整型数的值是payload的真实长度。注意，网络字节序，需要转换。

(3) 如果值是127，则后面8个字节形成的64位无符号整型数的值是payload的真实长度。注意，网络字节序，需要转换。

长度表示遵循一个原则，用最少的字节表示长度（我理解是尽量减少不必要的传输）。举例说，payload真实长度是124，在0-125之间，必须用前7位表示；不允许长度1是126或127，然后长度2是124，这样违反原则。

### 分片：

这种情况就比较复杂，具体可以参考rfc，一般在日常中用到的应该会比较少。

# **四、客户端API**

websocket的客户端api描述可以参考这里，一个简单的代码参考：

<!DOCTYPE HTML><html><head><script type="text/javascript">function WebSocketTest() {

if ("WebSocket" in window) {

alert("WebSocket is supported by your Browser!");

// Let us open a web socket

var ws = new WebSocket("ws://localhost:9998/echo");

ws.onopen = function() {

// Web Socket is connected, send data using send()

ws.send("Message to send");

alert("Message is sent...");

};

ws.onmessage = function(evt) {

var received\_msg = evt.data;

alert("Message is received...");

};

ws.onclose = function() {

// websocket is closed.

alert("Connection is closed...");

};

} else {

// The browser doesn't support WebSocket

alert("WebSocket NOT supported by your Browser!");

}

}</script></head><body><div id="sse">

<a href="javascript:WebSocketTest()">Run WebSocket</a></div></body></html>

五、服务器处理

目前开源的websocket server的代码还是不少的。

自己也整理过一份c的，支持上面三种握手情况，并嵌在reactor框架下使用，在chrome下测试过。

[https://github.com/gaccob/gbase/blob/master/net/wsconn.h](https://github.com/gaccob/gbase/blob/master/net/wsconn.h" \t "http://www.qixing318.com/article/_blank)

[https://github.com/gaccob/gbase/blob/master/net/wsconn.c](https://github.com/gaccob/gbase/blob/master/net/wsconn.c" \t "http://www.qixing318.com/article/_blank)

六、目前主流的浏览器支持情况

|  |  |
| --- | --- |
| Chrome | Supported in version 4+ |
| Firefox | Supported in version 4+ |
| Internet Explorer | Supported in version 10+ |
| Opera | Supported in version 10+ |
| Safari | Supported in version 5+ |

1. 参考文章

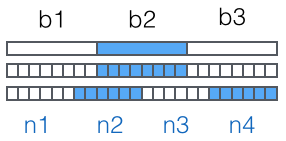
Base64是一种用64个字符来表示任意二进制数据的方法。

用记事本打开exe、jpg、pdf这些文件时，我们都会看到一大堆乱码，因为二进制文件包含很多无法显示和打印的字符，所以，如果要让记事本这样的文本处理软件能处理二进制数据，就需要一个二进制到字符串的转换方法。Base64是一种最常见的二进制编码方法。

Base64的原理很简单，首先，准备一个包含64个字符的数组：

['A', 'B', 'C', ... 'a', 'b', 'c', ... '0', '1', ... '+', '/']

然后，对二进制数据进行处理，每3个字节一组，一共是3x8=24bit，划为4组，每组正好6个bit：



这样我们得到4个数字作为索引，然后查表，获得相应的4个字符，就是编码后的字符串。

所以，Base64编码会把3字节的二进制数据编码为4字节的文本数据，长度增加33%，好处是编码后的文本数据可以在邮件正文、网页等直接显示。

如果要编码的二进制数据不是3的倍数，最后会剩下1个或2个字节怎么办？Base64用\x00字节在末尾补足后，再在编码的末尾加上1个或2个=号，表示补了多少字节，解码的时候，会自动去掉