[nodejs实现Websocket的数据接收发送](https://www.cnblogs.com/axes/p/4514199.html)

　　在去年的时候，写过一篇关于websocket的博文：<http://www.cnblogs.com/axes/p/3586132.html> ，里面主要是借助了nodejs-websocket这个插件，后来还用了socket.io做了些demo，但是，这些都是借助于别人封装好的插件做出来的，websocket到底是怎么实现的呢自己之前真没怎么去想过，最近在看朴灵大神的《深入浅出nodejs》时候，看到websocket那一章，看了一下websocket的数据帧的定义，就琢磨着自己用nodejs来实现一下。

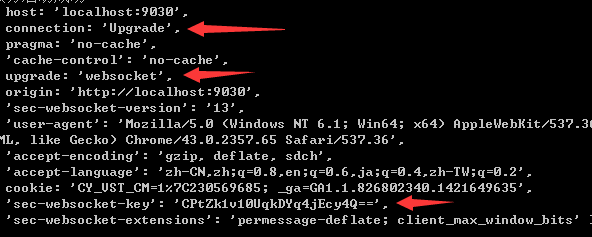
　　客户端的代码就不说了，websocket的API还是很简单的，就通过onmessage、onopen、onclose，以及send方法就可以实现了。

　　主要说服务端的代码：

　　首先是协议的升级，这个比较简单，就简述一下：

　　当在客户端执行new Websocket("ws://XXX.com/")的时候，客户端就会发起请求报文进行握手申请，报文中有个很重要的key就是Sec-WebSocket-Key，服务端获取到key，然后将这个key与字符串258EAFA5-E914-47DA-95CA-C5AB0DC85B11相连，对新的字符串通过sha1安全散列算法计算出结果后，再进行base64编码，并且将结果放在请求头的"Sec-WebSocket-Accept"中写出即可完成握手。然后即可进行数据传输

　　客户端请求头截图：



　　而服务端的响应则请看代码：

[复制代码](javascript:void(0);)

server.on('upgrade', function (req, socket, upgradeHead) {

var key = req.headers['sec-websocket-key'];

key = crypto.createHash("sha1").update(key + "258EAFA5-E914-47DA-95CA-C5AB0DC85B11").digest("base64");

var headers = [

'HTTP/1.1 101 Switching Protocols',

'Upgrade: websocket',

'Connection: Upgrade',

'Sec-WebSocket-Accept: ' + key

];

socket.setNoDelay(true);

socket.write(headers.join("\r\n") + "\r\n\r\n", 'ascii');

var ws = new WebSocket(socket);

webSocketCollector.push(ws);

callback(ws);

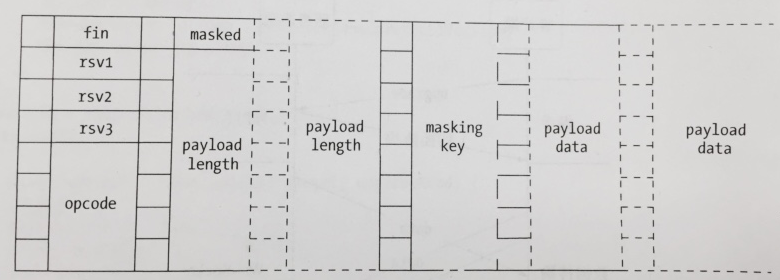
});

[复制代码](javascript:void(0);)

　　upgrade事件其实是http这个模块的封装，再往底层就是net模块的实现，其实都差不多，如果直接用net模块来实现的话，就是监听net.createServer返回的server对象的data事件，接收到的第一份数据就是客户端发来的升级请求报文。

　　上面那段代码就完成了websocket的握手，然后就可以开始数据传输了。

　　看数据传输之前，先看看websocket数据帧的定义（因为觉得深入浅出nodejs里的帧定义图最容易理解，所以就贴这张了）：



　　上面的图中，每一列就是一个字节，一个字节总共是8位，每一位就是一个二进制数，不同位的值会对应不同的意义。

**fin：**指示这个是消息的最后片段。第一个片段可能也是最后的片段。如果为1即为最后片段。

**rsv1、rsv2、rsv3：** 各占一个位，用于扩展协商，基本上不怎么需要理，一般都是0

**opcode：**占四个位，可以表示0~15的十进制，0表示为附加数据帧，1表示为文本数据帧，2表示二进制数据帧，8表示发送一个连接关闭的数据帧，9表示ping，10表示pong，ping和pong都是用于心跳检测，当一端发送ping时，另一端必须响应pong表示自己仍处于响应状态。

**masked：**占一个位，表示是否进行掩码处理，客户端发送给服务端时为1，服务端发送给客户端时为0

**payload length：**占7位，或者7+16位、或者7+64位。如果第二个字节的后面七个位的十进制值小于或等于125，则直接用这七个位表示数据长度；如果该值为126，说明 125<数据长度<65535（16个位能描述的最大值，也就是16个1的时候），就用第三个字节及第四个字节即16个位来表示；如果该值为127，则说明数据长度已经大于65535，16个位也已经不足以描述数据长度了，就用第三到第十个字节这八个字节来描述数据长度。

**masking key：**当masked为1的时候才存在，用于对我们需要的数据进行解密。

**payload data：**我们需要的数据，如果masked为1，该数据会被加密，要通过masking key进行异或运算解密才能获取到真实数据。

　　上面的帧定义中，fin位是初学的时候最容易搞混的，fin位看似简单代表分片的开始和结束，但是有一点要注意的就是，什么情况下才会分片？我刚开始理解的以为是，当发送的数据比较大的时候会进行分片，但是实际操作发现并不是，当我在浏览器上发送上万字节数据的时候，服务端收到的并不是分片数据，而是分块数据。

　　什么是分块数据？其实就是把一段符合websocket数据规范的数据分成多块传输，从而服务端收到的第一块数据里fin位是1，但是附带的数据长度却比palyload length要少很多，在接下来又收到多块数据，后面收到的数据是没有上面那些控制头的，而是纯数据。总的来说整个数据可以看成是一整块，然后分成多块后发给服务端。其实这个很容易理解，就是socket的分包发送而已。

　　那什么时候会分片？据我了解，只有当传输的数据长度不确定的时候，才会进行分片，比如一边读某个数据一边发送给服务端。如果发送数据长度是确定的，就算数据量很大也只是会进行分块而不会分片。

　　引用开涛博客里对分片的解释：

　　分片的主要目的是允许当消息开始但不必缓冲该消息时发送一个未知大小的消息。如果消息不能被分片，那么端点将不得不缓冲整个消息以便在首字节发生之前统计出它的长度。对于分片，服务器或中间件可以选择一个合适大小的缓冲，当缓冲满时，写一个片段到网络。

　　帧定义解释完了，就可以根据数据来进行解析了，当有data过来的时候，先获取需要的数据信息，下面这段代码将获取到数据在data里的位置，以及数据长度，masking key以及opcode：

[复制代码](javascript:void(0);)

WebSocket.prototype.handleDataStat = function (data) {

if (!this.stat) {

var dataIndex = 2; //数据索引，因为第一个字节和第二个字节肯定不为数据，所以初始值为2

var secondByte = data[1]; //代表masked位和可能是payloadLength位的第二个字节

var hasMask = secondByte >= 128; //如果大于或等于128，说明masked位为1

secondByte -= hasMask ? 128 : 0; //如果有掩码，需要将掩码那一位去掉

var dataLength, maskedData;

//如果为126，则后面16位长的数据为数据长度，如果为127，则后面64位长的数据为数据长度

if (secondByte == 126) {

dataIndex += 2;

dataLength = data.readUInt16BE(2);

} else if (secondByte == 127) {

dataIndex += 8;

dataLength = data.readUInt32BE(2) + data.readUInt32BE(6);

} else {

dataLength = secondByte;

}

//如果有掩码，则获取32位的二进制masking key，同时更新index

if (hasMask) {

maskedData = data.slice(dataIndex, dataIndex + 4);

dataIndex += 4;

}

//数据量最大为10kb

if (dataLength > 10240) {

this.send("Warning : data limit 10kb");

} else {

//计算到此处时，dataIndex为数据位的起始位置，dataLength为数据长度，maskedData为二进制的解密数据

this.stat = {

index: dataIndex,

totalLength: dataLength,

length: dataLength,

maskedData: maskedData,

opcode: parseInt(data[0].toString(16).split("")[1] , 16) //获取第一个字节的opcode位

};

}

} else {

this.stat.index = 0;

}

};

[复制代码](javascript:void(0);)

　　代码中均有注释，理解起来应该不难，直接看下一步，获取到数据信息后，就要对数据进行实际解析了：

　　经过上面handleDataStat方法的处理，stat中已经有了data的相关数据，先判断opcode，如果为9说明是客户端发起的ping心跳检测，直接返回pong响应，如果为10则为服务端发起的心跳检测。如果有masking key，则遍历数据段，对每个字节都与masking key的字节进行异或运算（网上看到一个说法很形象：就是轮流发生X关系），^符号就是进行异或运算啦。如果没有masking key则直接通过slice方法把数据截取下来。

　　获取到数据后，放进datas里保存，因为有可能数据被分块了，所以再将stat里的长度减去当前数据长度，只有当stat里的长度为0的时候，说明当前帧为最后一帧，然后通过Buffer.concat将所有数据合并，此时再判断一下opcode，如果opcode为8，则说明客户端发起了一个关闭请求，而我们获取到的数据则是关闭原因。如果不为8，则这数据就是我们需要的数据。然后再将stat重置为null，datas数组置空即可。至此，我们的数据解析就完成了。

[复制代码](javascript:void(0);)

WebSocket.prototype.dataHandle = function (data) {

this.handleDataStat(data);

var stat;

if (!(stat = this.stat)) return;

//如果opcode为9，则发送pong响应，如果opcode为10则置pingtimes为0

if (stat.opcode === 9 || stat.opcode === 10) {

(stat.opcode === 9) ? (this.sendPong()) : (this.pingTimes = 0);

this.reset();

return;

}

var result;

if (stat.maskedData) {

result = new Buffer(data.length-stat.index);

for (var i = stat.index, j = 0; i < data.length; i++, j++) {

//对每个字节进行异或运算，masked是4个字节，所以%4，借此循环

result[j] = data[i] ^ stat.maskedData[j % 4];

}

} else {

result = data.slice(stat.index, data.length);

}

this.datas.push(result);

stat.length -= (data.length - stat.index);

//当长度为0，说明当前帧为最后帧

if (stat.length == 0) {

var buf = Buffer.concat(this.datas, stat.totalLength);

if (stat.opcode == 8) {

this.close(buf.toString());

} else {

this.emit("message", buf.toString());

}

this.reset();

}

};

[复制代码](javascript:void(0);)

　　完成了客户端发来的数据解析，还需要一个服务端发数据至客户端的方法，也就是按照上面所说的帧定义来组装数据并且发送出去。下面的代码中基本上每一行都有注释，应该还是比较容易理解的。

[复制代码](javascript:void(0);)

//数据发送

WebSocket.prototype.send = function (message) {

if(this.state !== "OPEN") return;

message = String(message);

var length = Buffer.byteLength(message);

// 数据的起始位置，如果数据长度16位也无法描述，则用64位，即8字节，如果16位能描述则用2字节，否则用第二个字节描述

var index = 2 + (length > 65535 ? 8 : (length > 125 ? 2 : 0));

// 定义buffer，长度为描述字节长度 + message长度

var buffer = new Buffer(index + length);

// 第一个字节，fin位为1，opcode为1

buffer[0] = 129;

// 因为是由服务端发至客户端，所以无需masked掩码

if (length > 65535) {

buffer[1] = 127;

// 长度超过65535的则由8个字节表示，因为4个字节能表达的长度为4294967295，已经完全够用，因此直接将前面4个字节置0

buffer.writeUInt32BE(0, 2);

buffer.writeUInt32BE(length, 6);

} else if (length > 125) {

buffer[1] = 126;

// 长度超过125的话就由2个字节表示

buffer.writeUInt16BE(length, 2);

} else {

buffer[1] = length;

}

// 写入正文

buffer.write(message, index);

this.socket.write(buffer);

};

[复制代码](javascript:void(0);)

　　除此之外还要实现一个功能，就是心跳检测：防止服务端长时间不与客户端交互而导致客户端关闭连接，所以每隔十秒都会发送一次ping进行心跳检测

[复制代码](javascript:void(0);)

//每隔10秒进行一次心跳检测，若连续发出三次心跳却没收到响应则关闭socket

WebSocket.prototype.checkHeartBeat = function () {

var that = this;

setTimeout(function () {

if (that.state !== "OPEN") return;

if (that.pingTimes >= 3) {

that.close("time out");

return;

}

//记录心跳次数

that.pingTimes++;

that.sendPing();

that.checkHeartBeat();

}, 10000);

};

WebSocket.prototype.sendPing = function () {

this.socket.write(new Buffer(['0x89', '0x0']))

};

WebSocket.prototype.sendPong = function () {

this.socket.write(new Buffer(['0x8A', '0x0']))

};

[复制代码](javascript:void(0);)

　　最后，在主函数里直接调用，一收到消息就广播。。。

[复制代码](javascript:void(0);)

var server = http.createServer(function(req , res){

router.route(req , res);

}).listen(9030);

websocket.update(server , function(ws){

ws.on('close' , function(reason){

console.log("socket closed:"+reason);

});

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

websocket.brocast(data);

});

});

[复制代码](javascript:void(0);)

　　至此，整个websocket的实现就完成了，此demo只是大概实现了一下websocket而已，在安全之类方面肯定还是有很多问题，若是真正生产环境中还是用socket.io这类成熟的插件比较好。不过这还是很值得一学的。

　　附上该demo的github地址：<https://github.com/whxaxes/node-test/tree/master/server/websocket>

　　里面的socket.js是该文中引用的代码完整版。socket\_2.js则是后面我在公司内部进行了一次分享时写的优化版。

　　如果觉得demo能帮到你，就给个star或者fork呗