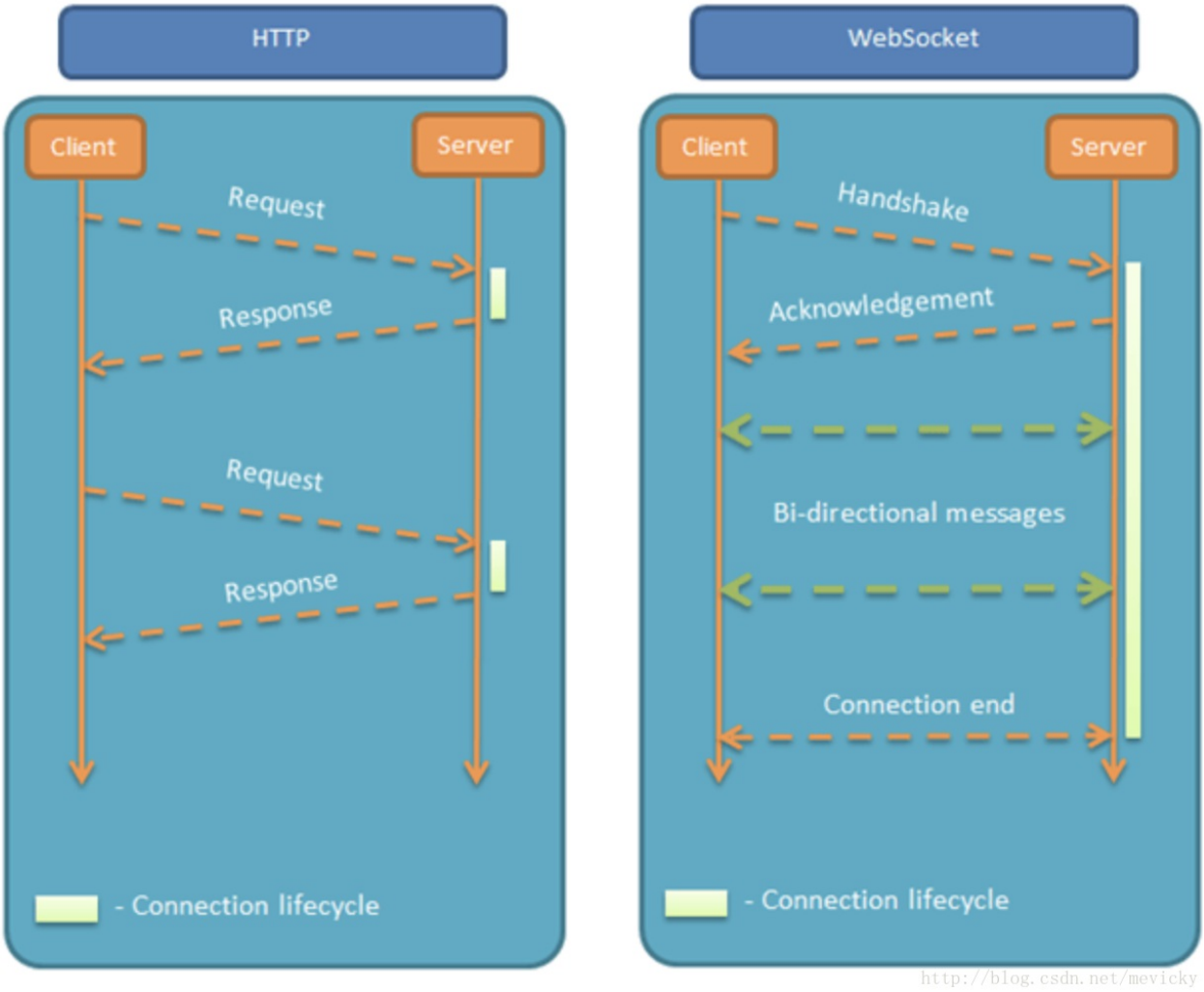
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=62 sentences=126, words=1163

1.websocket介绍

- WebSockets API (https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API)是HTML5开始提供的一种浏览器与服务器进行全双工通讯的网络技术
- 通俗地讲：在客户端和服务器保有一个持久的连接，两边可以在任意时间开始发送数据
- HTML5开始提供的一种浏览器与服务器进行全双工通讯的网络技术
- 属于应用层协议，它基于TCP传输协议，并复用HTTP的握手通道



2. websocket实战

2.1 server.js

server.js

```
const { Server } = require('ws');
const wss = new Server({ port: 8888 });
wss.on('connection', (socket) => {
    socket.on('message', (message) => {
        socket.send(message);
    });
});
```

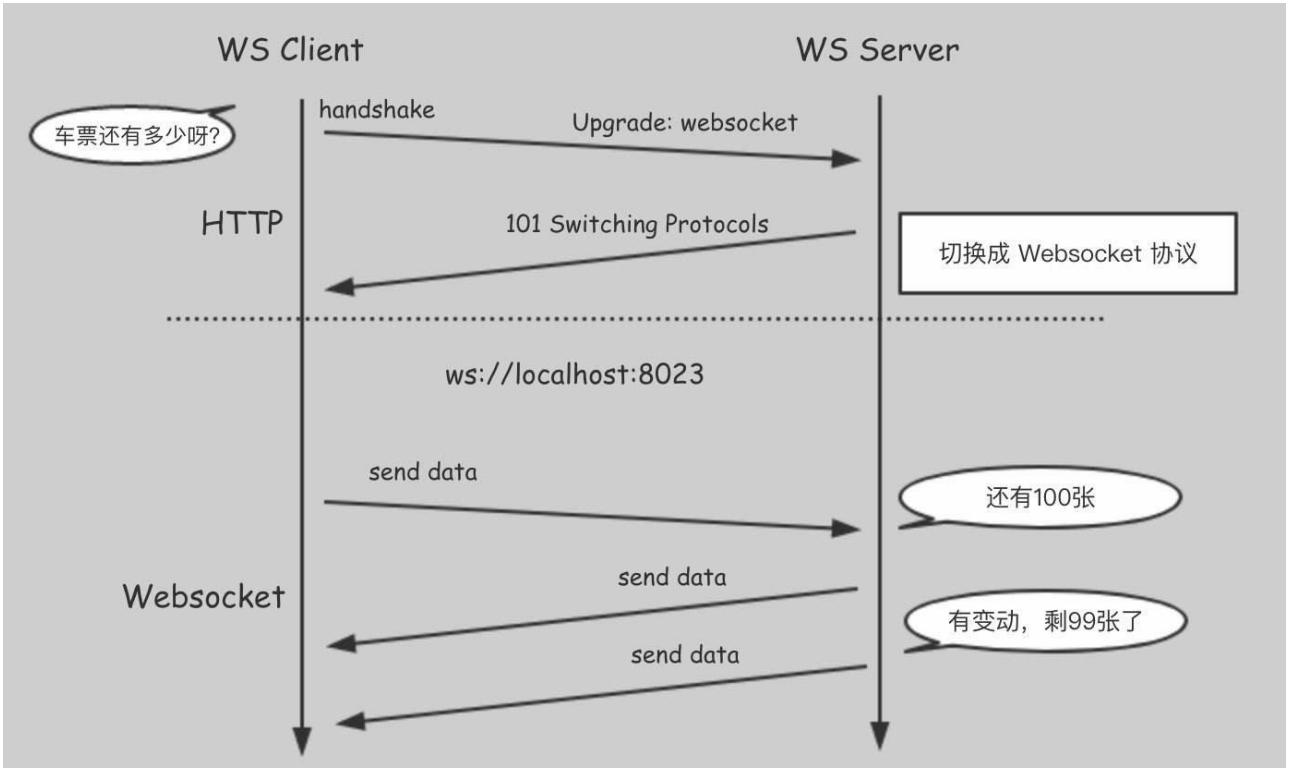
2.2 client.js

```
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>websockettitle>
head>
<body>
    <input type="text" id="text">
    <button onclick="send()">发送button>
    <script type="text/javascript">
        let text = document.getElementById('text');
        var ws = new WebSocket("ws://localhost:8888");
        ws.onopen = function () {
            ws.send("server");
        };
        ws.onmessage = function (event) {
            console.log('onmessage', event.data);
        };
        function send() {
            ws.send(text.value);
            text.value = '';
        }
    script>
body>
html>
```

3. websocket连接

- WebSocket复用了HTTP的握手通道
- 具体指的是,客户端通过HTTP请求与WebSocket服务端协商升级协议
- 协议升级完成后,后续的数据交换则遵照WebSocket的协议



▼ General

Request URL: ws://localhost:8888/

Request Method: GET

Status Code: 101 Switching Protocols

▼ Response Headers view source

Connection: Upgrade

Sec-WebSocket-Accept: nFC7Rafae2kuoydqvia7BYAzdOY=

Upgrade: websocket

▼ Request Headers view source

Accept-Encoding: gzip, deflate, br

Accept-Language: zh-CN,zh;q=0.9

Cache-Control: no-cache

Connection: Upgrade

Host: localhost:8888

Origin: http://127.0.0.1:8080

Pragma: no-cache

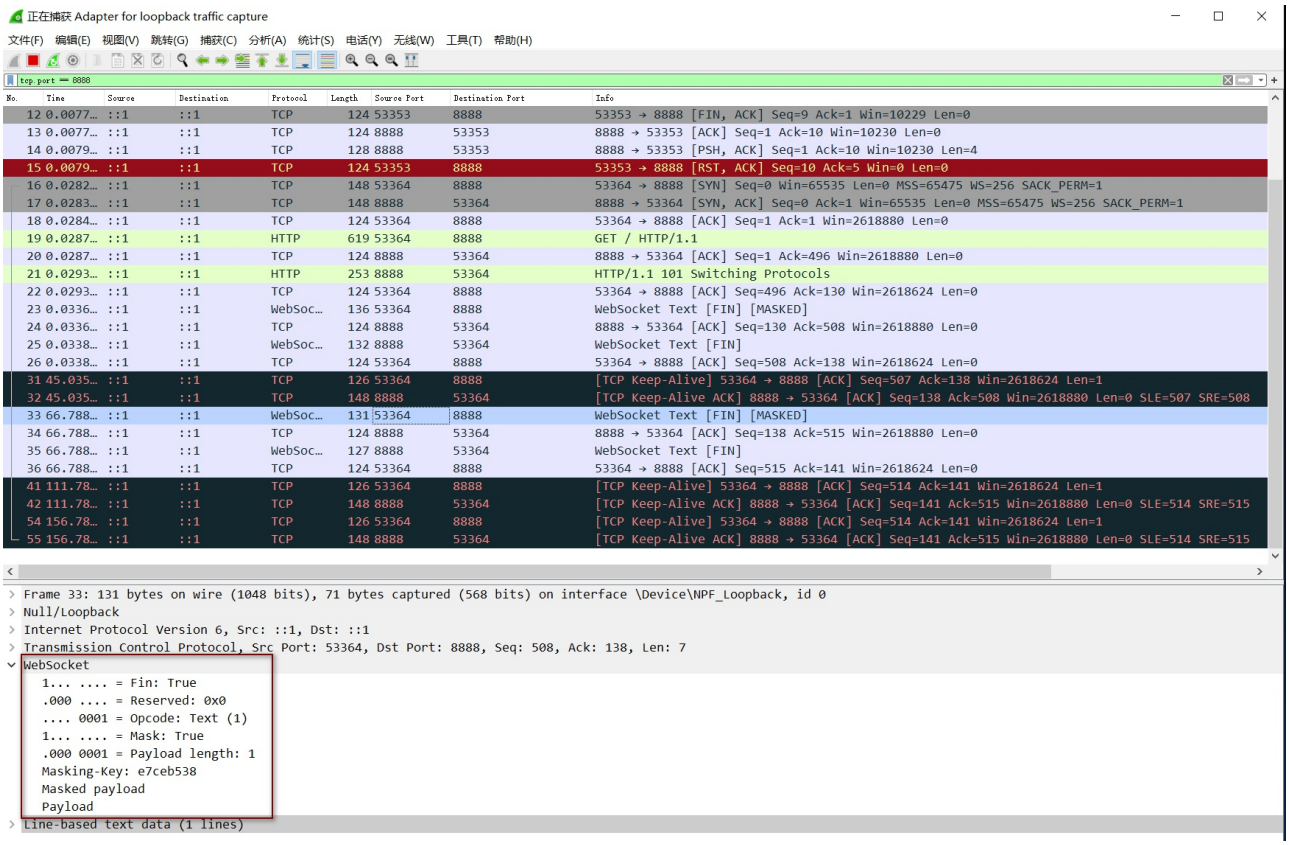
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits

Sec-WebSocket-Key: PTJljznYuIh0LHj5uWUegg==

Sec-WebSocket-Version: 13

Upgrade: websocket

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36



3.1 客户端：申请协议升级

- 首先客户端发起协议升级请求
- 请求采用的是标准的HTTP报文格式，且只支持GET方法

```
GET ws:
Host: localhost:8888
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: IHfMdf8a0aQXbwQO1pkGdA==
```

字段 含义 Connection: Upgrade 表示要升级协议 Upgrade: websocket 表示要升级到websocket协议 Sec-WebSocket-Version: 13 表示websocket的版本 Sec-WebSocket-Key 与后面服务端响应首部的Sec-WebSocket-Accept是配套的，提供基本的防护，比如恶意的连接，或者无意义的连接

3.2 服务端：响应协议升级

- 服务端返回内容如下
    - 状态代码101表示协议切换
- 到此完成协议升级，后续的数据交互都按照新的协议来

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: aWAY+V/uyz5ILZEoWuWdxjnlb7E=
```

字段 含义 Connection: Upgrade 升级协议 Upgrade: websocket 升级到websocket协议 Sec-WebSocket-Accept Accept字符串

3.3 Sec-WebSocket-Accept的计算

- Sec-WebSocket-Accept根据客户端请求首部的Sec-WebSocket-Key计算出来
- 计算公式为：
    - 将Sec-WebSocket-Key跟258EAFA5-E914-47DA-95CA-C5AB0DC85B11拼接
    - 通过SHA1计算出摘要，并转成base64字符串

```
const crypto = require('crypto');
const CODE = '258EAFA5-E914-47DA-95CA-C5AB0DC85B11';
function toAcceptKey(wsKey) {
    return crypto.createHash('sha1').update(wsKey + CODE).digest('base64');;
}
const webSocketKey = 'IHfMdf8a0aQXbwQO1pkGdA==';
console.log(toAcceptKey(webSocketKey));
```

4. 数据帧格式

- WebSocket客户端、服务端通信的最小单位是帧 (https://tools.ietf.org/html/rfc6455#section-5.2)，由1个或多个帧组成一条完整的消息（message）
- 发送端 将消息切割成多个帧，并发送给服务端
- 接收端 接收消息帧，并将关联的帧重新组装成完整的消息

4.1 bit和byte

- 比特就是bit 二进制数系统中,每个0或1就是一个位(bit),位是数据存储的最小单位
- 其中8个bit就称为一个字节(Byte)

4.2 位运算符

4.2.1 按位与(&)

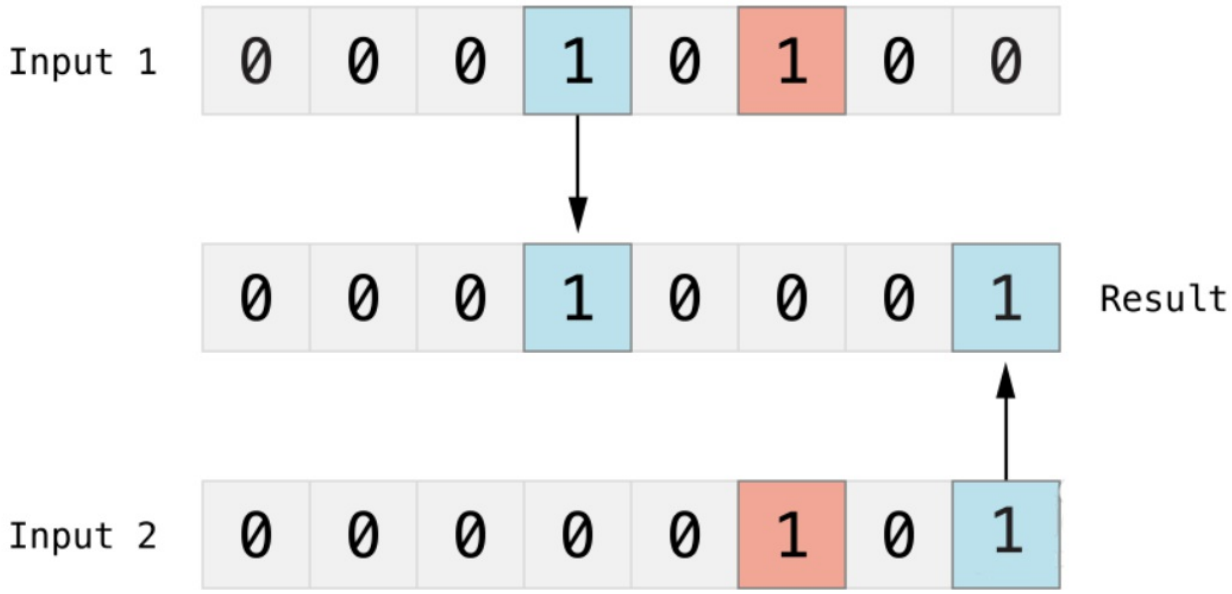
- 两个输入数的同一位都为1才为1

4.2.2 按位或(|)

- 两个输入数的同一位只要有一个为1就是1

4.2.3 按位异或(^)

- 两个输入数的同一位不同就是1,如果相同就设为0



4.3 数据帧格式

- 单位是比特 比如FIN、RSV1各占据1比特,opcode占据4比特

```
 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 ...                |
+---------------------------------------------------------------+
```

字段 含义 FIN 1个比特 如果是1,表示这是消息（message）的最后一个分片（fragment）,如果是0,表示不是是消息(message)的最后一个分片(fragment) RSV1, RSV2, RSV3 各占1个比特。一般情况下全为0 Opcode 4个比特,操作代码 Mask 1个比特。表示是否要对数据载荷进行掩码操作,从客户端向服务端发送数据时，需要对数据进行掩码操作；从服务端向客户端发送数据时，不需要对数据进行掩码操作,如果Mask是1，那么在Masking-key中会定义一个掩码键(masking key),并用这个掩码键来对数据载荷进行反掩码。所有客户端发送到服务端的数据帧，Mask都是1 Payload length 数据载荷的长度 Masking-key 0或4字节(32位) 所有从客户端传送到服务端的数据帧,数据载荷都进行了掩码操作,Mask为1,且携带了4字节的Masking-key。如果Mask为0,则没有Masking-key。载荷数据的长度，不包括mask key的长度 Payload data 载荷数据

Opcode

字段 含义 %x0 表示一个延续帧。当Opcode为0时，表示本次数据传输采用了数据分片，当前收到的数据帧为其中一个数据分片 %x1 表示这是一个文本帧 %x2 表示这是一个二进制帧 %x3-7 保留的操作代码 %x8 表示连接断开 %x9 表示这是一个ping操作 %xA 表示这是一个pong操作 %xB-F 保留的操作代码

4.4 Payload length

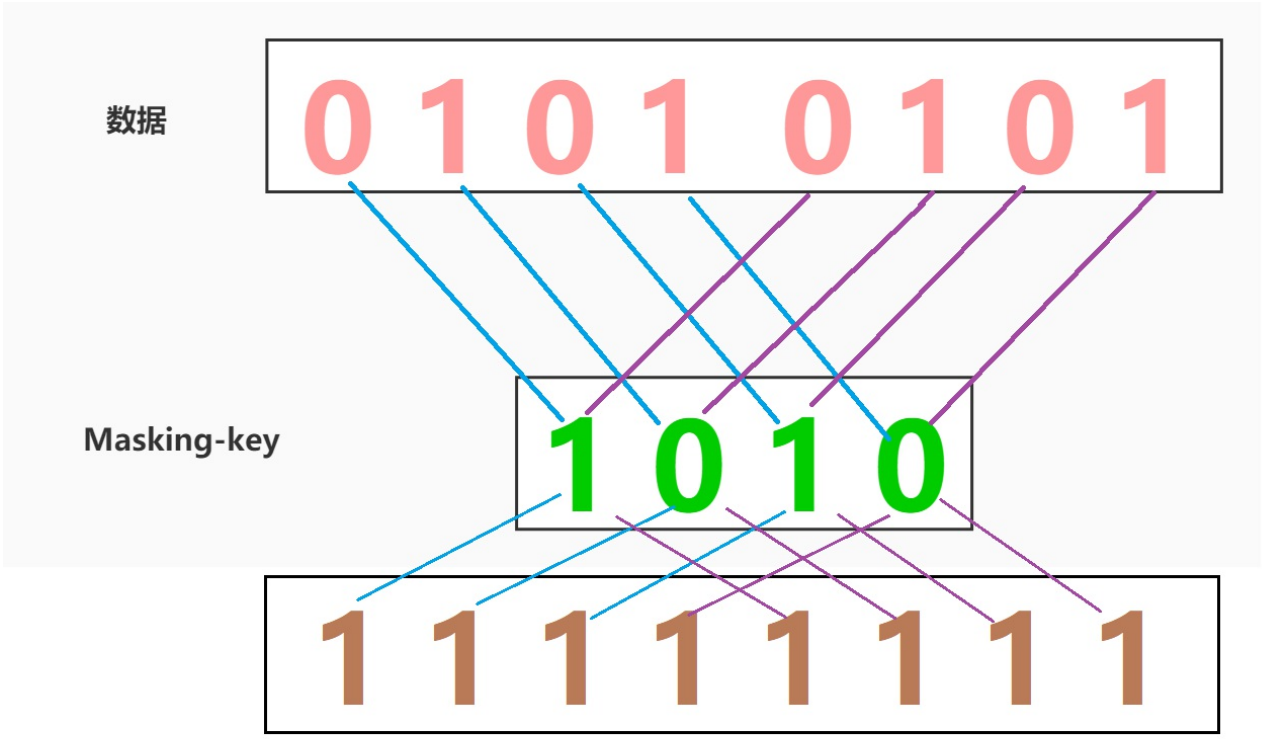
- Payload length：数据载荷的长度，单位是字节。为7位，或7+16位，或7+64位
    - Payload length=x为0~125：数据的长度为x字节。
    - Payload length=x为126：后续2个字节代表一个16位的无符号整数，该无符号整数的值为数据的长度
    - Payload length=x为127：后续8个字节代表一个64位的无符号整数（最高位为0），该无符号整数的值为数据的长度。
    - 如果payload length占用了多个字节的话，payload length的二进制表达采用网络序（big endian，重要的位在前)
- readBigUInt64BE 用指定的字节序[readBigInt64BE() 读取为大端序,readBigInt64LE() 读取为小端序]从 buf 中指定的 offset 读取一个有符号的 64 位整数值
- Big-endian(大端序) 高位字节在前
- Little-endian(小端序) 低位字节在前

```
let buffer = Buffer.from([0b00000001, 0b00000000]);
console.log(Math.pow(2, 8));
console.log(buffer.readUInt16BE(0));
console.log(buffer.readUInt16LE(0));
```

```
function getLength(buffer) {
    const byte = buffer.readUInt8(1);
    let length = parseInt(byte.toString(2).substring(1), 2);
    if (length === 126) {
        length = buffer.readUInt16BE(2);
    } else if (length === 127) {
        length = buffer.readBigUInt64BE(2);
    }
    return length;
}
console.log(126..toString(2));
console.log(127..toString(2));
console.log(getLength(Buffer.from([0b10000001, 0b10000001])));
console.log(getLength(Buffer.from([0b10000001, 0b11111110, 0b00000000, 0b00000001])));
console.log(getLength(Buffer.from([0b10000001, 0b11111111, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000000, 0b00000001])));
```

4.5 掩码算法

- 掩码键（Masking-key）是由客户端挑选出来的 32bit的随机数,掩码操作不会影响数据载荷的长度
- 掩码和反掩码操作都采用如下算法
- 对索引 i模以4得到结果并对原来的索引进行异或操作




```
function unmask(buffer, mask) {
    const length = buffer.length;
    for (let i = 0; i < length; i++) {
        buffer[i] ^= mask[i % 4];
    }
}

let mask = Buffer.from([1, 0, 1, 0]);
let buffer = Buffer.from([0, 1, 0, 1, 0, 1, 0, 1]);
unmask(buffer, mask);
console.log(buffer);
```

5. 实现**websocket**服务器 **#**

```
const net = require('net');
const { EventEmitter } = require('events');
const crypto = require('crypto');
const CODE = '258EAFA5-E914-47DA-95CA-C5AB0DC85B11';
const OP_CODES = {
    TEXT: 1,
    BINARY: 2
};
class Server extends EventEmitter {
    constructor(options) {
        super(options);
        this.options = options;
        this.server = net.createServer(this.listener);
        this.server.listen(options.port);
    }
    listener = (socket) => {
        socket.setKeepAlive(true);
        socket.send = function (payload) {
            let _opcode;
            if (Buffer.isBuffer(payload)) {
                _opcode = OP_CODES.BINARY;
            } else {
                _opcode = OP_CODES.TEXT;
                payload = Buffer.from(payload);
            }
            let length = payload.length;
            let buffer = Buffer.alloc(2 + length);
            buffer[0] = 0b10000000 | _opcode;
            buffer[1] = length;
            payload.copy(buffer, 2);
            socket.write(buffer);
        }
        socket.on('data', (chunk) => {
            if (chunk.toString().match(/Upgrade: websocket/)) {
                this.upgrade(socket, chunk.toString());
            } else {
                this.onmessage(socket, chunk);
            }
        });
        this.emit('connection', socket);
    }
    onmessage = (socket, chunk) => {
        let FIN = (chunk[0] & 0b10000000) === 0b10000000;
        let opcode = chunk[0] & 0b00001111;
        let masked = (chunk[1] & 0b10000000) === 0b10000000;
        let payloadLength = chunk[1] & 0b01111111;
        let payload;
        if (masked) {
            let masteringKey = chunk.slice(2, 6);
            payload = chunk.slice(6);
            unmask(payload, masteringKey);
        }
        if (FIN) {
            switch (opcode) {
                case OP_CODES.TEXT:
                    socket.emit('message', payload.toString());
                    break;
                case OP_CODES.BINARY:
                    socket.emit('message', payload);
                    break;
                default:
                    break;
            }
        }
    }
    upgrade = (socket, chunk) => {
        let rows = chunk.split('\r\n');
        let headers = toHeaders(rows.slice(1, -2));
        let wsKey = headers['Sec-WebSocket-Key'];
        let acceptKey = toAcceptKey(wsKey);
        let response = [
            'HTTP/1.1 101 Switching Protocols',
            'Upgrade: websocket',
            `Sec-WebSocket-Accept: ${acceptKey}`,
            'Connection: Upgrade',
            '\r\n'
        ].join('\r\n');
        socket.write(response);
    }
}
function toAcceptKey(wsKey) {
    return crypto.createHash('sha1').update(wsKey + CODE).digest('base64');;
}
function toHeaders(rows) {
    const headers = {};
    rows.forEach(row => {
        let [key, value] = row.split(': ');
        headers[key] = value;
    });
    return headers;
}
function unmask(buffer, mask) {
    const length = buffer.length;
    for (let i = 0; i < length; i++) {
        buffer[i] ^= mask[i & 3];
    }
}

exports.Server = Server;
```