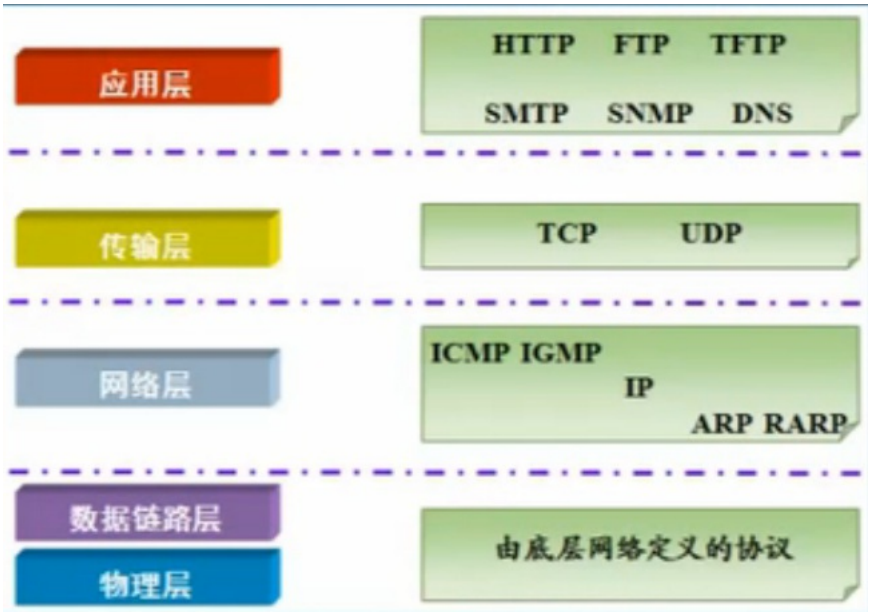
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=83 sentences=683, words=3227

1.本课学习目标

- 学习如何获取专业权威的一手知识
- 学习如何阅读RFC标准文档
- 学习扩展的巴科斯范式(ABNF)定义的通信协议语言
- 学习HTTP协议的实现和解析细节

1.1 TCP/IP参考模型

- TCP/IP协议被称为传输控制协议/互联网协议，又称网络通讯协议



2.GET

2.1 使用

2.1.1 http-server.js

```
const http = require('http');
const fs = require('fs');
const path = require('path');
const server = http.createServer(function(req,res){
  if(['/get.html'].includes(req.url)){
    res.writeHead(200,{'Context-type':"text-html"});
    res.end(fs.readFileSync(path.join(__dirname,'static',req.url.slice(1))));
  }else if(req.url === '/get'){
    res.writeHead(200,{'Context-type':"text-plain"});
    res.end('get');
  }
});
server.listen(8080);
```

2.1.2 static\get.html

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>gettitle>
head>
<body>
    <script>
        let xhr = new XMLHttpRequest();
        xhr.onreadystatechange = ()=>{
            console.log('onreadystatechange',xhr.readyState);
        }
        xhr.open("GET", "http://127.0.0.1:8080/get");
        xhr.responseType="text";
        xhr.setRequestHeader('name', 'zhufeng');
        xhr.setRequestHeader('age', '10');
        xhr.onload = () => {
            console.log('readyState',xhr.readyState);
            console.log('status',xhr.status);
            console.log('statusText',xhr.statusText);
            console.log('getAllResponseHeaders',xhr.getAllResponseHeaders());
            console.log('response',xhr.response);
        };
        xhr.send();
     script>
body>
html>
```

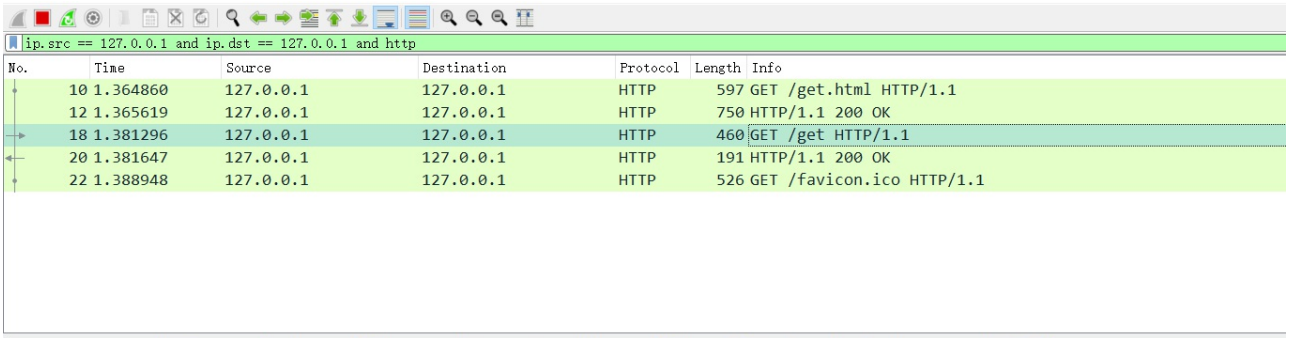
2.1.3 请求响应格式

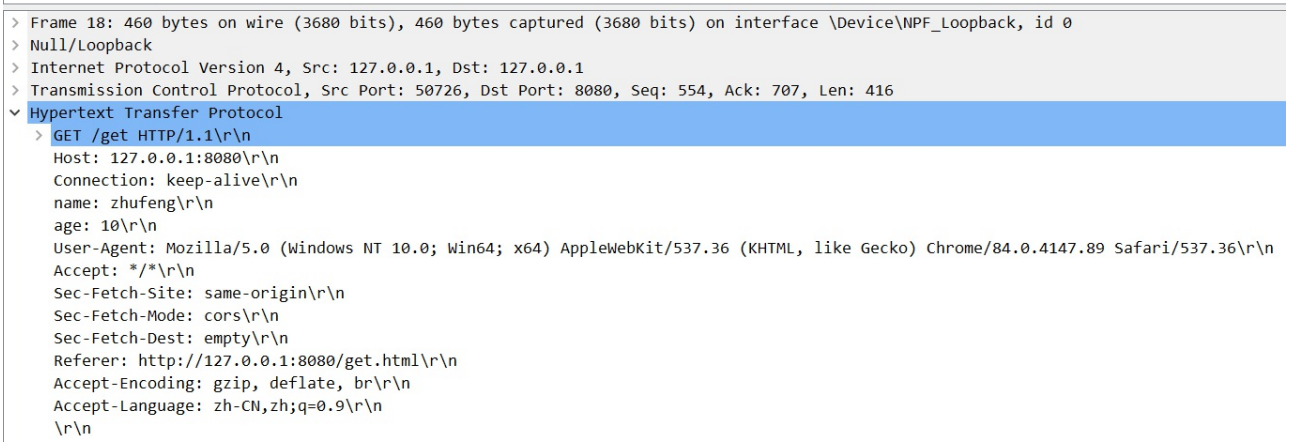
- 一个请求消息是从客户端到服务器端的,在消息首行里包含方法,资源指标符,协议版本

```
Request=Request-Line;
*((general-header|request-header|entity-header)CRLF)
CRLF
[message-body]
```

2.1.3.1 请求

```
GET /get HTTP/1.1
Host: 127.0.0.1:8080
Connection: keep-alive
name: zhufeng
age: 10
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.89 Safari/537.36
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
```

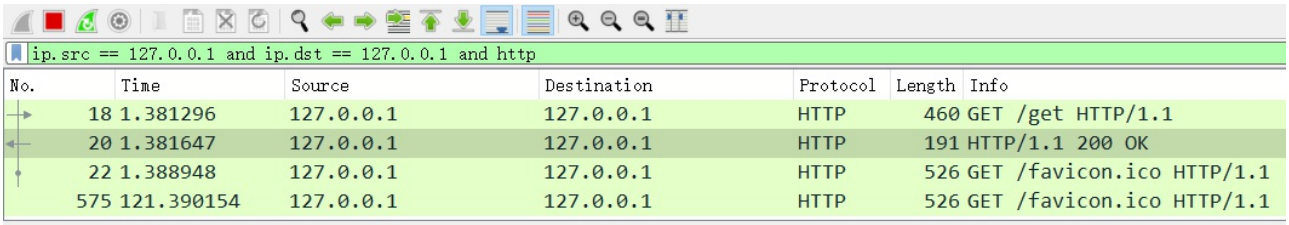


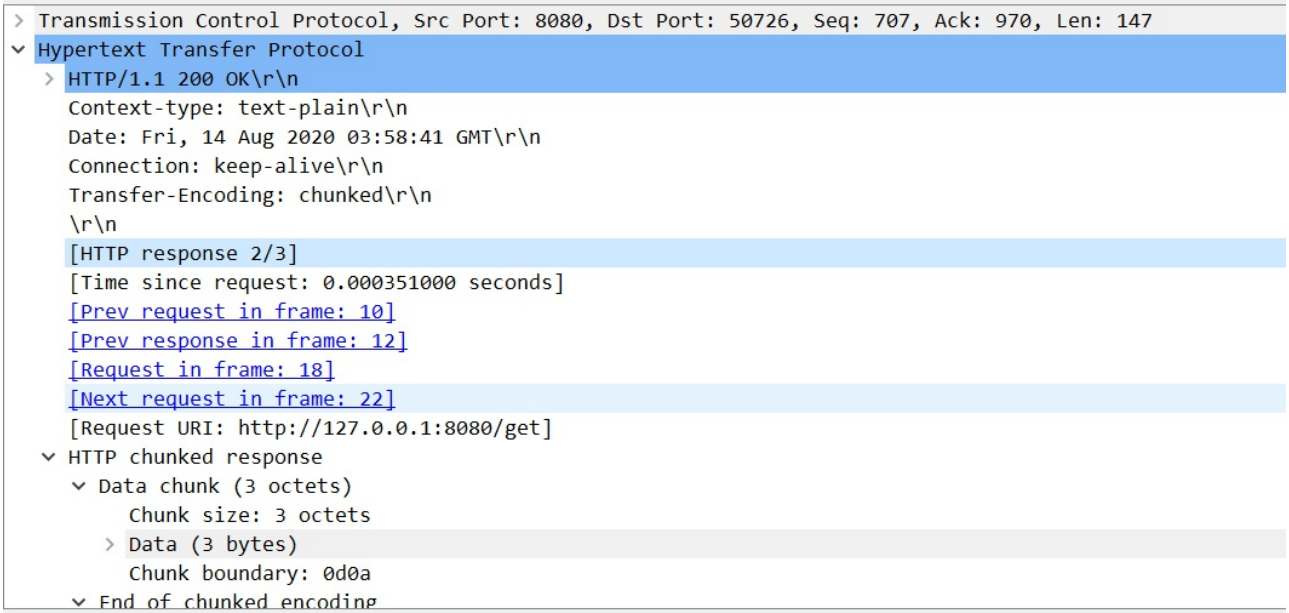


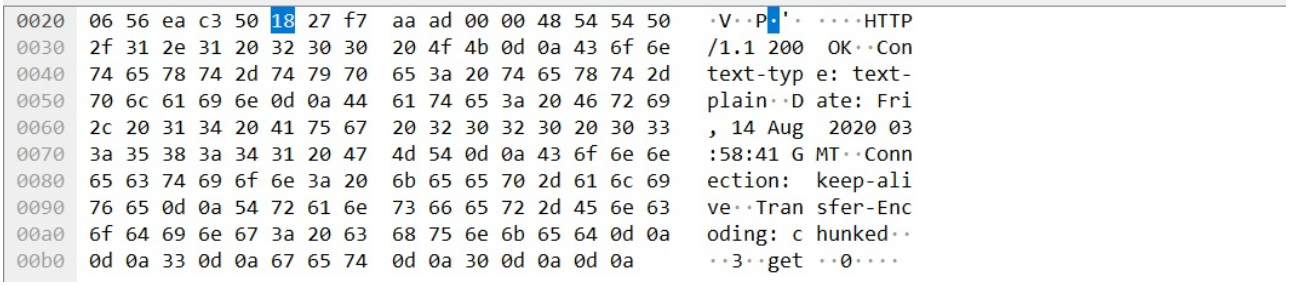
2.1.3.2 响应

```
Response=Status-Line;
*((general-header|response-header|entity-header)CRLF)
CRLF
[message-body]
```

```
HTTP/1.1 200 OK
Context-type: text-plain
Date: Fri, 14 Aug 2020 03:58:41 GMT
Connection: keep-alive
Transfer-Encoding: chunked

3
get
0
```







2.2 实现

2.2.1 tcp-get-client.js

- readyState (https://developer.mozilla.org/zh-CN/docs/Web/API/XMLHttpRequest/readyState)

tcp-get-client.js

```
let net = require('net');
const ReadyState = {
    UNSENT:0,
    OPENED:1,
    HEADERS_RECEIVED:2,
    LOADING:3,
    DONE:4
}
class XMLHttpRequest {
    constructor(){
        this.readyState = ReadyState.UNSENT;
        this.headers = {};
    }
    open(method, url) {
        this.method = method||'GET';
        this.url = url;
        let {hostname,port,path} = require('url').parse(url);
        this.hostname = hostname;
        this.port = port;
        this.path = path;
        this.headers.Host=`${hostname}:${port}`;
        const socket = this.socket =  net.createConnection({port: this.port,hostname:this.hostname},()=>{
            socket.on('data', (data) => {
                data = data.toString();
                let [response,bodyRows] = data.split('\r\n\r\n');
                let [statusLine,...headerRows] = response.split('\r\n');
                let [,status,statusText] = statusLine.split(' ');
                this.status = status;
                this.statusText = statusText;
                this.responseHeaders = headerRows.reduce((memo,row)=>{
                    let [key,value] = row.split(': ');
                    memo[key]= value;
                    return memo;
                },{});
                this.readyState = ReadyState.HEADERS_RECEIVED;
                xhr.onreadystatechange&&xhr.onreadystatechange();
                this.readyState = ReadyState.LOADING;
                xhr.onreadystatechange&&xhr.onreadystatechange();
                let [,body,] = bodyRows.split('\r\n');
                this.response = this.responseText = body;
                this.readyState = ReadyState.DONE;
                xhr.onreadystatechange&&xhr.onreadystatechange();
                this.onload&&this.onload();
            });
            socket.on('error', (err) => {
                this.onerror&&this.onerror(err);
            });
         });
         this.readyState = ReadyState.OPENED;
         xhr.onreadystatechange&&xhr.onreadystatechange();
    }
    getAllResponseHeaders(){
        let allResponseHeaders='';
        for(let key in this.responseHeaders){
            allResponseHeaders+=`${key}: ${this.responseHeaders[key]}\r\n`;
        }
        return allResponseHeaders;
    }
    setRequestHeader(header,value){
        this.headers[header]= value;
    }
    send() {
        let rows = [];
        rows.push(`${this.method} ${this.path} HTTP/1.1`);
        rows.push(...Object.keys(this.headers).map(key=>`${key}: ${this.headers[key]}`));
        this.socket.write(rows.join('\r\n')+'\r\n\r\n');
    }
}

let xhr = new XMLHttpRequest();
xhr.onreadystatechange = ()=>{
    console.log('onreadystatechange',xhr.readyState);
}
xhr.open("GET", "http://127.0.0.1:8080/get");
xhr.responseType="text";
xhr.setRequestHeader('name', 'zhufeng');
xhr.setRequestHeader('age', '10');
xhr.onload = () => {
    console.log('readyState',xhr.readyState);
    console.log('status',xhr.status);
    console.log('statusText',xhr.statusText);
    console.log('getAllResponseHeaders',xhr.getAllResponseHeaders());
    console.log('response',xhr.response);
};
xhr.send();
```

2.2.2 tcp-get-sever

tcp-get-sever.js

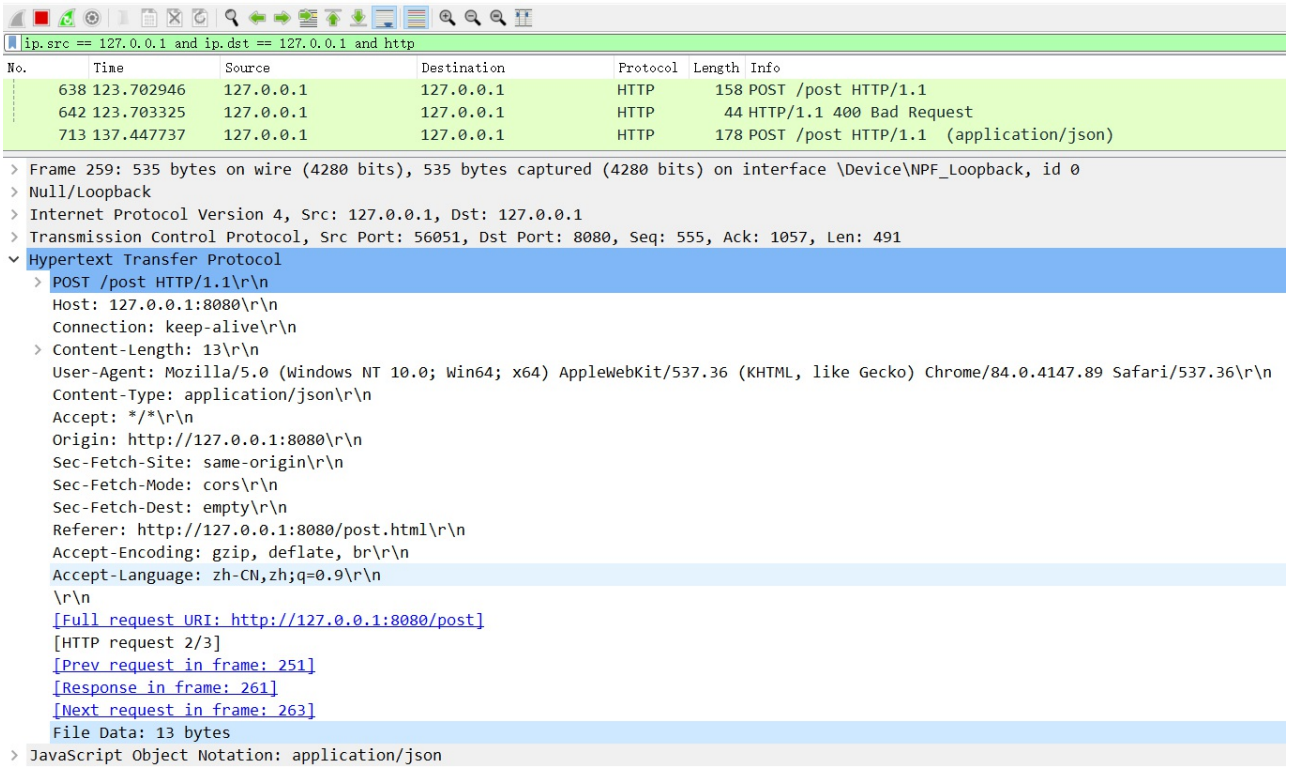
```
const net = require('net');
const server = net.createServer((socket) => {
  socket.on('data',(data)=>{
    let request = data.toString();
    let [requestLine,...headerRows] = request.split('\r\n');
    let [method,path] = requestLine.split(' ');
    let headers = headerRows.slice(0,-2).reduce((memo,row)=>{
      let [key,value] = row.split(': ');
      memo[key] = value;
      return memo;
    },{});
    console.log('method',method);
    console.log('path',path);
    console.log('headers',headers);

    let rows = [];
    rows.push(`HTTP/1.1 200 OK`);
    rows.push(`Context-type: text-plain`);
    rows.push(`Date: ${new Date().toGMTString()}`);
    rows.push(`Connection: keep-alive`);
    rows.push(`Transfer-Encoding: chunked`);
    let responseBody = 'get';
    rows.push(`\r\n${Buffer.byteLength(responseBody).toString(16)}\r\n${responseBody}\r\n0`);
    let response = rows.join('\r\n');
    socket.end(response);
  });
})
server.on('error', (err) => {
  console.error(err);
});

server.listen(8080,() => {
  console.log('服务器已经启动', server.address());
});
```

5.POST方法

5.1 实战

5.1.1 请求和响应

5.1.1.1 请求



ip.src == 127.0.0.1 and ip.dst == 127.0.0.1 and http

No.	Time	Source	Destination	Protocol	Length	Info
638	123.702946	127.0.0.1	127.0.0.1	HTTP	158	POST /post HTTP/1.1
642	123.703325	127.0.0.1	127.0.0.1	HTTP	44	HTTP/1.1 400 Bad Request
713	137.447737	127.0.0.1	127.0.0.1	HTTP	178	POST /post HTTP/1.1 (application/json)

```
> Frame 259: 535 bytes on wire (4280 bits), 535 bytes captured (4280 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 56051, Dst Port: 8080, Seq: 555, Ack: 1057, Len: 491
v Hypertext Transfer Protocol
  > POST /post HTTP/1.1\r\n
    Host: 127.0.0.1:8080\r\n
    Connection: keep-alive\r\n
  > Content-Length: 13\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.89 Safari/537.36\r\n
    Content-Type: application/json\r\n
    Accept: */*\r\n
    Origin: http://127.0.0.1:8080\r\n
    Sec-Fetch-Site: same-origin\r\n
    Sec-Fetch-Mode: cors\r\n
    Sec-Fetch-Dest: empty\r\n
    Referer: http://127.0.0.1:8080/post.html\r\n
    Accept-Encoding: gzip, deflate, br\r\n
    Accept-Language: zh-CN,zh;q=0.9\r\n
    \r\n
    [Full request URI: http://127.0.0.1:8080/post]
    [HTTP request 2/3]
    [Prev request in frame: 251]
    [Response in frame: 261]
    [Next request in frame: 263]
    File Data: 13 bytes
> JavaScript Object Notation: application/json
```

```
0120  6e 0d 0a 41 63 63 65 70  74 3a 20 2a 2f 2a 0d 0a   n··Accep t: */*··
0130  4f 72 69 67 69 6e 3a 20  68 74 74 70 3a 2f 2f 31   Origin:  http://1
0140  32 37 2e 30 2e 30 2e 31  3a 38 30 38 30 0d 0a 53   27.0.0.1 :8080··S
0150  65 63 2d 46 65 74 63 68  2d 53 69 74 65 3a 20 73   ec-Fetch -Site: s
0160  61 6d 65 2d 6f 72 69 67  69 6e 0d 0a 53 65 63 2d   ame-orig in··Sec-
0170  46 65 74 63 68 2d 4d 6f  64 65 3a 20 63 6f 72 73   Fetch-Mo de: cors
0180  0d 0a 53 65 63 2d 46 65  74 63 68 2d 44 65 73 74   ··Sec-Fe tch-Dest
0190  3a 20 65 6d 70 74 79 0d  0a 52 65 66 65 72 65 72   : empty· ·Referer
01a0  3a 20 68 74 74 70 3a 2f  2f 31 32 37 2e 30 2e 30   : http:/ /127.0.0
01b0  2e 31 3a 38 30 38 30 2f  70 6f 73 74 2e 68 74 6d   .1:8080/ post.htm
01c0  6c 0d 0a 41 63 63 65 70  74 2d 45 6e 63 6f 64 69   l··Accep t-Encodi
01d0  6e 67 3a 20 67 7a 69 70  2c 20 64 65 66 6c 61 74   ng: gzip , deflat
01e0  65 2c 20 62 72 0d 0a 41  63 63 65 70 74 2d 4c 61   e, br··A ccept-La
01f0  6e 67 75 61 67 65 3a 20  7a 68 2d 43 4e 2c 7a 68   nguage:  zh-CN,zh
0200  3b 71 3d 30 2e 39 0d 0a  0d 0a 7b 22 6e 61 6d 65   ;q=0.9·· ··{"name
0210  22 3a 22 7a 66 22 7d                               ":"zf"}
```

5.1.1.2 响应 #

```
    715 137.452195    127.0.0.1           127.0.0.1           HTTP      201 HTTP/1.1 200 OK

> Frame 715: 201 bytes on wire (1608 bits), 201 bytes captured (1608 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 8080, Dst Port: 56102, Seq: 1, Ack: 135, Len: 157
v Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    Context-type: text-plain\r\n
    Date: Fri, 14 Aug 2020 07:36:45 GMT\r\n
    Connection: keep-alive\r\n
    Transfer-Encoding: chunked\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.004458000 seconds]
    [Request in frame: 713]
    [Request URI: http://127.0.0.1:8080/post]
  > HTTP chunked response
    File Data: 13 bytes
  v Data (13 bytes)
      Data: 7b226e616d65223a227a66227d
      [Length: 13]

0000  02 00 00 00 45 00 00 c5  e9 5a 40 00 40 06 00 00   ····E··· ·Z@·@···
0010  7f 00 00 01 7f 00 00 01  1f 90 db 26 b7 b1 79 fc   ········ ···&··y·
0020  9e 12 41 2a 50 18 27 f9  70 5c 00 00 48 54 54 50   ··A*P·'· p\··HTTP
0030  2f 31 2e 31 20 32 30 30  20 4f 4b 0d 0a 43 6f 6e   /1.1 200  OK··Con
0040  74 65 78 74 2d 74 79 70  65 3a 20 74 65 78 74 2d   text-typ e: text-
0050  70 6c 61 69 6e 0d 0a 44  61 74 65 3a 20 46 72 69   plain··D ate: Fri
0060  2c 20 31 34 20 41 75 67  20 32 30 32 30 20 30 37   , 14 Aug  2020 07
0070  3a 33 36 3a 34 35 20 47  4d 54 0d 0a 43 6f 6e 6e   :36:45 G MT··Conn
0080  65 63 74 69 6f 6e 3a 20  6b 65 65 70 2d 61 6c 69   ection:  keep-ali
0090  76 65 0d 0a 54 72 61 6e  73 66 65 72 2d 45 6e 63   ve··Tran sfer-Enc
00a0  6f 64 69 6e 67 3a 20 63  68 75 6e 6b 65 64 0d 0a   oding: c hunked··
00b0  0d 0a 64 0d 0a 7b 22 6e  61 6d 65 22 3a 22 7a 66   ··d··{"n ame":"zf
00c0  22 7d 0d 0a 30 0d 0a 0d  0a                       "}··0··· ·
```

5.1.2 http-server.js

http-server.js

```
const http = require('http');
const fs = require('fs');
const path = require('path');
const server = http.createServer(function(req,res){
+  if(['/get.html','/post.html'].includes(req.url)){
    res.writeHead(200,{'Context-type':"text-html"});
    res.end(fs.readFileSync(path.join(__dirname,'static',req.url.slice(1))));
  }else if(req.url
    res.writeHead(200,{'Context-type':"text-plain"});
    res.end('get');
+  }else if(req.url === '/post'){
+    let buffers = [];
+    req.on('data',(data)=>{
+      buffers.push(data);
+    });
+    req.on('end',()=>{
+      console.log('method',req.method);
+      console.log('url',req.url);
+      console.log('headers',req.headers);
+      let body = Buffer.concat(buffers);
+      console.log('body',body.toString());
+      res.statusCode = 200;
+      res.setHeader('Context-type',"text-plain");
+      res.write(body);
+      res.end();
+    });
  }
});
server.listen(8080);
```

5.1.3 static\post.html

static\post.html

```
   get

       let xhr = new XMLHttpRequest();
       xhr.onreadystatechange = ()=>{
           console.log('onreadystatechange',xhr.readyState);
       }
<span class="hljs-addition">+       xhr.open("POST", "http://127.0.0.1:8080/post");</span>
       xhr.responseType="text";
<span class="hljs-addition">+       xhr.setRequestHeader('Content-Type','application/json');</span>
       xhr.onload = () => {
           console.log('readyState',xhr.readyState);
           console.log('status',xhr.status);
           console.log('statusText',xhr.statusText);
           console.log('getAllResponseHeaders',xhr.getAllResponseHeaders());
           console.log('response',xhr.response);
       };
<span class="hljs-addition">+       xhr.send(JSON.stringify({name:'zhufeng',age:10}));</span>
```

5.2 实现

5.2.1 tcp-post-client.js

tcp-post-client.js

```
let net = require('net');
const ReadyState = {
    UNSENT:0,//（代理被创建，但尚未调用 open() 方法。
    OPENED:1,//open() 方法已经被调用
    HEADERS_RECEIVED:2,//send() 方法已经被调用，并且头部和状态已经可获得。
    LOADING:3,//（交互）正在解析响应内容
    DONE:4 //（完成）响应内容解析完成，可以在客户端调用了
}
class XMLHttpRequest {
    constructor(){
        this.readyState = ReadyState.UNSENT;
        this.headers = {};
    }
    open(method, url) {
        this.method = method||'GET';
        this.url = url;
        let {hostname,port,path} = require('url').parse(url);
        this.hostname = hostname;
        this.port = port;
        this.path = path;
        this.headers.Host=`${hostname}:${port}`;
+       this.headers.Connection=`keep-alive`;
        const socket = this.socket =  net.createConnection({port: this.port,hostname:this.hostname},()=>{
            socket.on('data', (data) => {
                data = data.toString();
                console.log(data);
                let [response,bodyRows] = data.split('\r\n\r\n');
                let [statusLine,...headerRows] = response.split('\r\n');
                let [,status,statusText] = statusLine.split(' ');
                this.status = status;
                this.statusText = statusText;
                this.responseHeaders = headerRows.reduce((memo,row)=>{
                    let [key,value] = row.split(': ');
                    memo[key]= value;
                    return memo;
                },{});
                this.readyState = ReadyState.HEADERS_RECEIVED;
                xhr.onreadystatechange&&xhr.onreadystatechange();
                this.readyState = ReadyState.LOADING;
                xhr.onreadystatechange&&xhr.onreadystatechange();
                let [,body,] = bodyRows.split('\r\n');
                this.response = this.responseText = body;
                this.readyState = ReadyState.DONE;
                xhr.onreadystatechange&&xhr.onreadystatechange();
                this.onload&&this.onload();
            });
            socket.on('error', (err) => {
                this.onerror&&this.onerror(err);
            });
         });
         this.readyState = ReadyState.OPENED;
         xhr.onreadystatechange&&xhr.onreadystatechange();
    }
    getAllResponseHeaders(){
        let allResponseHeaders='';
        for(let key in this.responseHeaders){
            allResponseHeaders+=`${key}: ${this.responseHeaders[key]}\r\n`;
        }
        return allResponseHeaders;
    }
    setRequestHeader(header,value){
        this.headers[header]= value;
    }
    send(body) {
        let rows = [];
        rows.push(`${this.method} ${this.path} HTTP/1.1`);
+       this.headers["Content-Length"]=Buffer.byteLength(body);
        rows.push(...Object.keys(this.headers).map(key=>`${key}: ${this.headers[key]}`));
+       let request = rows.join('\r\n')+'\r\n\r\n'+body;
        console.log(request);
        this.socket.write(request);
    }
}

let xhr = new XMLHttpRequest();
xhr.onreadystatechange = ()=>{
    console.log('onreadystatechange',xhr.readyState);
}
xhr.open("POST", "http://127.0.0.1:8080/post");
xhr.responseType="text";
xhr.setRequestHeader('Content-Type','application/json');
xhr.onload = () => {
    console.log('readyState',xhr.readyState);
    console.log('status',xhr.status);
    console.log('statusText',xhr.statusText);
    console.log('getAllResponseHeaders',xhr.getAllResponseHeaders());
    console.log('response',xhr.response);
};
xhr.send(`{"name":"zf"}`);
```

5.2.2 tcp-post-server.js

```
const net = require('net');
const Parer = require('./Parser');
const server = net.createServer((socket) => {
  socket.on('data',(data)=>{
+   let parser = new Parer();
+   let {method,url,headers,body} = parser.parse(data);
+   console.log('method',method);
+   console.log('url',url);
+   console.log('headers',headers);
+   console.log('body',body);
    let rows = [];
    rows.push(`HTTP/1.1 200 OK`);
    rows.push(`Context-type: text-plain`);
    rows.push(`Date: ${new Date().toGMTString()}`);
    rows.push(`Connection: keep-alive`);
    rows.push(`Transfer-Encoding: chunked`);
    rows.push(`\r\n${Buffer.byteLength(body).toString(16)}\r\n${body}\r\n0`);
    let response = rows.join('\r\n');
    socket.end(response);
  });
})
server.on('error', (err) => {
  console.error(err);
});

server.listen(8080,() => {
  console.log('服务器已经启动', server.address());
});
```

5.2.3 Parser.js

```
let LF = 10,
    CR = 13,
    SPACE = 32,
    COLON = 58;
let PARSER_UNINITIALIZED=0,
    START=1,
    REQUEST_LINE=2,
    HEADER_FIELD_START=3,
    HEADER_FIELD=4,
    HEADER_VALUE_START=5,
    HEADER_VALUE=6,
    READING_BODY=7;
class Parser {
    constructor(){
        this.state = PARSER_UNINITIALIZED;
    }
    parse(buffer) {
        let self =this,
            requestLine='',
            headers = {},
            body='',
            i=0,
            char,
            state = START,
            headerField='',
            headerValue='';
            console.log(buffer.toString());
        for (i = 0; i < buffer.length; i++) {
            char = buffer[i];
            switch (state) {
                case START:
                    state = REQUEST_LINE;
                    self['requestLineMark']=i;
                case REQUEST_LINE:
                    if (char == CR) {
                        requestLine=buffer.toString('utf8', self['requestLineMark'], i);
                        break;
                    }else if(char == LF){
                        state = HEADER_FIELD_START;
                    }
                    break;
                case HEADER_FIELD_START:
                    if(char === CR){
                        state = READING_BODY;
                        self['bodyMark'] = i+2;
                        break;
                    }else{
                        state = HEADER_FIELD;
                        self['headerFieldMark'] = i;
                    }
                case HEADER_FIELD:
                   if (char == COLON) {
                    headerField=buffer.toString('utf8', self['headerFieldMark'], i);
                    state = HEADER_VALUE_START;
                   }
                   break;
                case HEADER_VALUE_START:
                    if (char == SPACE) {
                        break;
                    }
                    self['headerValueMark'] = i;
                    state = HEADER_VALUE;
                case HEADER_VALUE:
                    if (char === CR) {
                        headerValue=buffer.toString('utf8', self['headerValueMark'], i);
                        headers[headerField] = headerValue;
                        headerField = '';
                        headerValue = '';
                    }else if(char === LF){
                        state = HEADER_FIELD_START;
                    }
                    break;
                default:
                    break;
            }
        }
        let [method,url] =requestLine.split(' ');
        body=buffer.toString('utf8', self['bodyMark'], i);
        return {method,url,headers,body};
    }
}
module.exports = Parser;
```

6.文件上传

6.1 实战

6.1.1 请求和响应

6.1.1.1 请求

```
http
```

No.	Time	Source	Destination	Protocol	Length	Info
5615	1232.823955	::1	::1	HTTP	354	POST /upload HTTP/1.1 (text/plain)

```
> POST /upload HTTP/1.1\r\n
  Host: localhost:8080\r\n
  Connection: keep-alive\r\n
> Content-Length: 290\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.89 Safari/537.36\r\n
  Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryMa4dRBGIvIjdDpws\r\n
  Accept: */*\r\n
  Origin: http://localhost:8080\r\n
  Sec-Fetch-Site: same-origin\r\n
  Sec-Fetch-Mode: cors\r\n
  Sec-Fetch-Dest: empty\r\n
  Referer: http://localhost:8080/upload.html?\r\n
  Accept-Encoding: gzip, deflate, br\r\n
  Accept-Language: zh-CN,zh;q=0.9\r\n
  \r\n
  [Full request URI: http://localhost:8080/upload]
  [HTTP request 1/1]
  [Response in frame: 5783]
  File Data: 290 bytes
v MIME Multipart Media Encapsulation, Type: multipart/form-data, Boundary: "----WebKitFormBoundaryMa4dRBGIvIjdDpws"
  [Type: multipart/form-data]
  First boundary: ------WebKitFormBoundaryMa4dRBGIvIjdDpws\r\n
v Encapsulated multipart part:
    Content-Disposition: form-data; name="username"\r\n\r\n
  > Data (7 bytes)
  Boundary: \r\n------WebKitFormBoundaryMa4dRBGIvIjdDpws\r\n
v Encapsulated multipart part:  (text/plain)
    Content-Disposition: form-data; name="avatar"; filename="file.txt"\r\n
    Content-Type: text/plain\r\n\r\n
  > Line-based text data: text/plain (1 lines)
  Last boundary: \r\n------WebKitFormBoundaryMa4dRBGIvIjdDpws--\r\n
```

```
0270  22 0d 0a 0d 0a 7a 68 75  66 65 6e 67 0d 0a 2d 2d   "····zhu feng··--
0280  2d 2d 2d 2d 57 65 62 4b  69 74 46 6f 72 6d 42 6f   ----WebK itFormBo
```

6.1.1.2 响应

181	11.315331	::1	::1	HTTP	210	HTTP/1.1 200 OK

```
> Frame 181: 210 bytes on wire (1680 bits), 210 bytes captured (1680 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 8080, Dst Port: 51990, Seq: 1, Ack: 824, Len: 146
v Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    Context-type: text-plain\r\n
    Date: Sat, 15 Aug 2020 01:28:44 GMT\r\n
    Connection: keep-alive\r\n
    Transfer-Encoding: chunked\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 8.242493000 seconds]
    [Request in frame: 21]
    [Request URI: http://localhost:8080/upload]
  v HTTP chunked response
    v Data chunk (2 octets)
        Chunk size: 2 octets
      > Data (2 bytes)
        Chunk boundary: 0d0a
    v End of chunked encoding
        Chunk size: 0 octets
      \r\n
    File Data: 2 bytes
  v Data (2 bytes)
      Data: 6f6b
      [Length: 2]
```

```
0000  6f 6b                                             ok
```

6.1.2 http-server.js

```
const http = require('http');
const fs = require('fs');
const path = require('path');
const formidable = require('formidable');
const url = require('url');
const server = http.createServer(function(req,res){
  const {pathname} = url.parse(req.url);
  if(['/get.html','/post.html','/upload.html'].includes(pathname)){
    res.writeHead(200,{'Context-type':"text-html"});
    res.end(fs.readFileSync(path.join(__dirname,'static',pathname.slice(1))));
  }else if(pathname
    res.writeHead(200,{'Context-type':"text-plain"});
    res.end('get');
  }else if(pathname
    let buffers = [];
    req.on('data',(data)=>{
      buffers.push(data);
    });
    req.on('end',()=>{
      console.log('method',req.method);
      console.log('url',req.url);
      console.log('headers',req.headers);
      let body = Buffer.concat(buffers);
      console.log('body',body.toString());
      res.statusCode = 200;
      res.setHeader('Context-type',"text-plain");
      res.write(body);
      res.end();
    });
  }else if(req.url
+    const form = formidable();
+    form.parse(req, (err, fields, files) => {
+      console.log('fields',fields);
+      console.log('files',files);
+      let avatar = files.avatar;
+      let filePath = path.join(__dirname,'static',avatar.name);
+      fs.writeFileSync(filePath,fs.readFileSync(avatar.path));
+      res.statusCode = 200;
+      res.setHeader('Context-type',"text-plain");
+      res.write(JSON.stringify({...fields,avatar:filePath}));
+      res.end();
+    });
  }else{
    res.statusCode = 404;
    res.end();
  }
});
server.listen(8080);
```

6.1.3 static\upload.html

static\upload.html

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>gettitle>
head>
<body>
    <form onsubmit="upload(event)">
        <input type="text" id="username"/>
        <input type="file" id="file"/>
        <input type="submit"/>
    form>
    <script>
       function upload(event){
        event.preventDefault();
        let username= document.getElementById('username').value;
        let file= document.getElementById('file').files[0];
        let xhr = new XMLHttpRequest();
        xhr.open("POST", "http://localhost:8080/upload");
        var formData=new FormData();
        formData.append("username",username);
        formData.append("file",file);
        xhr.responseType="text";
        xhr.onload = () => {
            console.log(xhr.response);
        };
        xhr.send(formData);
       }
     script>
body>
html>
```

6.1.4 tcp-upload-client.js

tcp-upload-client.js

```
let net = require('net');
let fs = require('fs');
let path = require('path');
const ReadyState = {
    UNSENT:0,//（代理被创建，但尚未调用 open() 方法。
    OPENED:1,//open() 方法已经被调用
    HEADERS_RECEIVED:2,//send() 方法已经被调用，并且头部和状态已经可获得。
    LOADING:3,//（交互）正在解析响应内容
    DONE:4 //（完成）响应内容解析完成，可以在客户端调用了
}
class XMLHttpRequest {
    constructor(){
        this.readyState = ReadyState.UNSENT;
        this.headers = {};
```

```
    }
    open(method, url) {
        this.method = method||'GET';
        this.url = url;
        let {hostname,port,path} = require('url').parse(url);
        this.hostname = hostname;
        this.port = port;
        this.path = path;
        this.headers.Host=`${hostname}:${port}`;
        this.headers.Connection=`keep-alive`;
        const socket = this.socket =  net.createConnection({port: this.port,hostname:this.hostname},()=>{
            socket.on('data', (data) => {
                data = data.toString();
                console.log(data);
                let [response,bodyRows] = data.split('\r\n\r\n');
                let [statusLine,...headerRows] = response.split('\r\n');
                let [,status,statusText] = statusLine.split(' ');
                this.status = status;
                this.statusText = statusText;
                this.responseHeaders = headerRows.reduce((memo,row)=>{
                    let [key,value] = row.split(': ');
                    memo[key]= value;
                    return memo;
                },{});
                this.readyState = ReadyState.HEADERS_RECEIVED;
                xhr.onreadystatechange&&xhr.onreadystatechange();
                this.readyState = ReadyState.LOADING;
                xhr.onreadystatechange&&xhr.onreadystatechange();
                let [,body,] = bodyRows.split('\r\n');
                this.response = this.responseText = body;
                this.readyState = ReadyState.DONE;
                xhr.onreadystatechange&&xhr.onreadystatechange();
                this.onload&&this.onload();
            });
            socket.on('error', (err) => {
                this.onerror&&this.onerror(err);
            });
        });
        this.readyState = ReadyState.OPENED;
        xhr.onreadystatechange&&xhr.onreadystatechange();
    }
    getAllResponseHeaders(){
        let allResponseHeaders='';
        for(let key in this.responseHeaders){
            allResponseHeaders+=`${key}: ${this.responseHeaders[key]}\r\n`;
        }
        return allResponseHeaders;
    }
    setRequestHeader(header,value){
        this.headers[header]= value;
    }
+    send(formData) {
+        let rows = [];
+        let boundary = '----WebKitFormBoundaryF5odcsAPqFAB2mkm';
+        this.headers['Content-Type']= `multipart/form-data; boundary=${boundary}`;
+        let parts = [];
+        for(let key in formData){
+            let value = formData[key];
+            if(typeof value === 'string'){
+                let rows = [];
+                rows.push(`Content-Disposition: form-data; name="${key}"\r\n`);
+                rows.push(value);
+                parts.push(rows.join('\r\n'));
+            }else{
+                let rows = [];
+                rows.push(`Content-Disposition: form-data; name="${value.name}"; filename="${value.filename}"`);
+                rows.push(`Content-Type: ${value.contentType}\r\n`);
+                rows.push(value.content);
+                parts.push(rows.join('\r\n'));
+            }
+        }
+        let body = parts.join('\r\n'+'--'+boundary+'\r\n');
+        body = '--'+boundary+'\r\n'+body+'\r\n--'+boundary+'--';
+        this.headers["Content-Length"]=Buffer.byteLength(body);
+        rows.push(`${this.method} ${this.path} HTTP/1.1`);
+        rows.push(...Object.keys(this.headers).map(key=>`${key}: ${this.headers[key]}`));
+        let request = rows.join('\r\n');
+        request += '\r\n\r\n';
+        let buffers = [Buffer.from(request)];
+        buffers.push(Buffer.from(body));
+        this.socket.write(Buffer.concat(buffers));
    }
}
+class FormData{
+    append(key,value){
+        this[key]=value;
+    }
+}
let xhr = new XMLHttpRequest();
xhr.onreadystatechange = ()=>{
    console.log('onreadystatechange',xhr.readyState);
}
+xhr.open("POST", "http://127.0.0.1:8080/upload");
xhr.responseType="text";
+let formData=new FormData();
+formData.append("username",'zhufeng');
+let file = fs.readFileSync(path.join(__dirname,'file.txt'));
+formData.append("avatar",{name:'file',filename:'file.txt',contentType:'text/plain',content:file});
xhr.onload = () => {
    console.log('readyState',xhr.readyState);
    console.log('status',xhr.status);
    console.log('statusText',xhr.statusText);
    console.log('getAllResponseHeaders',xhr.getAllResponseHeaders());
    console.log('response',xhr.response);
```

```
};
+xhr.send(formData);
```

6.1.5 tcp-upload-server.js

tcp-upload-server.js

```
const net = require('net');
const path = require('path');
const fs = require('fs');
const Parer = require('./Parser');
const server = net.createServer((socket) => {
  socket.on('data',(data)=>{
    let parser = new Parer();
    let {method,url,headers,body} = parser.parse(data);
    console.log('method',method);
    console.log('url',url);
    console.log('headers',headers);
    console.log('body',body);
+    let [,boundary] = headers['Content-Type'].match(/boundary=([^;]+)/i);
+    let parts = body.split('--'+boundary).slice(1,-1);
+    parts= parts.map(item=>item.slice(2,-2));
+    let fields = {};
+    let files = {};
+    for(let i=0;i
+      let part = parts[i];
+      let rows = part.split('\r\n');
+      if(rows.length==3){
+        let [key,,value] = rows;
+        let [,name] = key.toString().match(/name="([^"]+?)"/);
+        fields[name]= value.toString();
+      }else if(rows.length==4){
+        let [key,type,,value] = rows;
+        let [,name,filename] = key.toString().match(/name="([^"]+?)"; filename="([^"]+?)"/);
+        let filePath = path.join(__dirname,'static',filename);
+        fs.writeFileSync(filePath,value);
+        files[name]={name,filename,path:filePath};
+      }
+    }
+    console.log(fields);
+    console.log(files);
    let rows = [];
    rows.push(`HTTP/1.1 200 OK`);
    rows.push(`Context-type: text-plain`);
    rows.push(`Date: ${new Date().toGMTString()}`);
    rows.push(`Connection: keep-alive`);
    rows.push(`Transfer-Encoding: chunked`);
+    let responseBody = JSON.stringify({...fields,...files});
+    rows.push(`\r\n${Buffer.byteLength(responseBody).toString(16)}\r\n${responseBody}\r\n0`);
    let response = rows.join('\r\n');
    socket.end(response);
  });
})
server.on('error', (err) => {
  console.error(err);
});

server.listen(8080,() => {
  console.log('服务器已经启动', server.address());
});
```