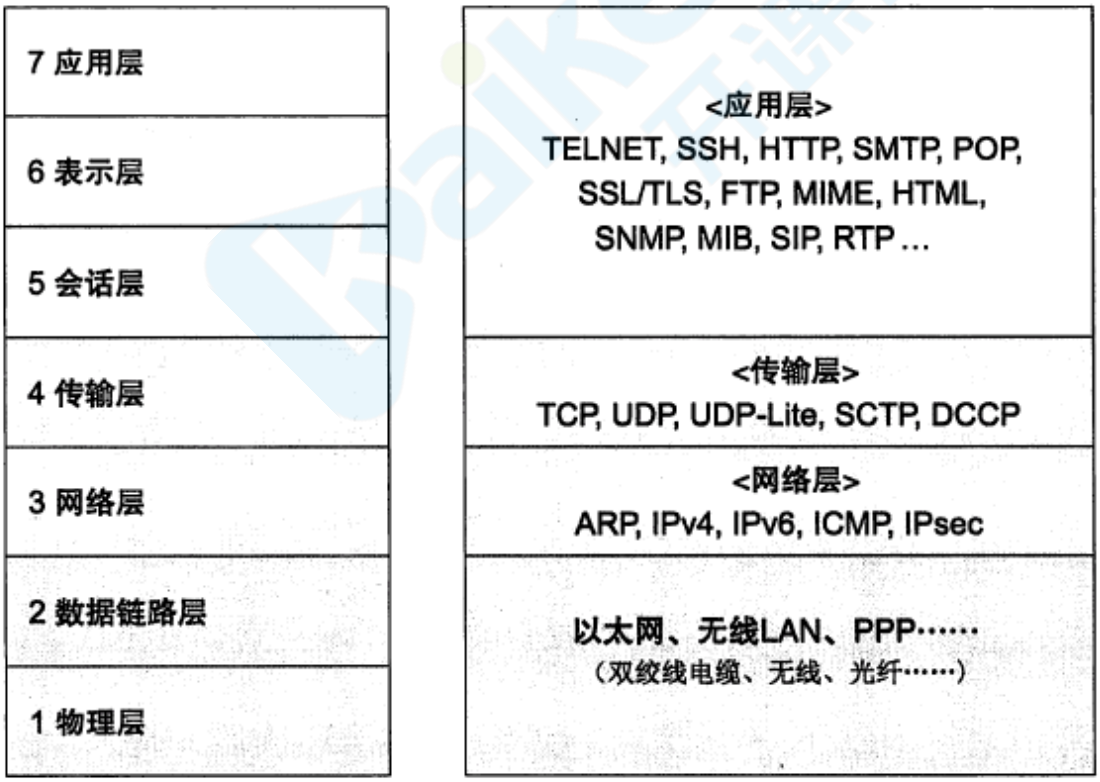
网络编程 http https http2 websocket

复习

- 看脑图

课堂目标

- 掌握HTTP协议
- 掌握跨域 CORS
- 掌握bodyparser原理
- 掌握上传原理
- 了解socketio实现实时聊天程序
- 爬虫



TCP协议 - 实现一个即时通讯IM

- Socket实现

原理：Net模块提供一个异步API能够创建基于流的TCP服务器，客户端与服务器建立连接后，服务器可以获得一个全双工Socket对象，服务器可以保存Socket对象列表，在接收某客户端消息时，推送给其他客户端。

```
// socket.js
const net = require('net')
const chatServer = net.createServer()
const clientList = []
chatServer.on('connection',client => {
    client.write('Hi!\n')
    clientList.push(client)
    client.on('data',data => {
        console.log('receive:',data.toString())
        clientList.forEach(v => {
            v.write(data)
        })
    })
})
chatServer.listen(9000)
```

通过Telnet连接服务器

```
telnet localhost 9000
```

HTTP协议

```
// 观察HTTP协议
curl -v http://www.baidu.com
```

- http协议详解
- 创建接口，api.js

```
// /http/api.js
const http = require("http");
const fs = require("fs");

http
  .createServer((req, res) => {
    const { method, url } = req;
    if (method == "GET" && url == "/") {
      fs.readFile("./index.html", (err, data) => {
        res.setHeader("Content-Type", "text/html");
```

```
        res.end(data);
      });
    } else if (method == "GET" && url == "/api/users") {
      res.setHeader("Content-Type", "application/json");
      res.end(JSON.stringify([{ name: "tom", age: 20 }]));
    }
  })
  .listen(4000, () => {
  console.log('api listen at ' + 4000)
});
```

- 请求接口

```
// index.html
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
    (async () => {
        const res = await axios.get("/api/users")
        console.log('data',res.data)
        document.writeln(`Response : ${JSON.stringify(res.data)}`)
    })()
</script>
```

- 埋点更容易

```
const img = new Image()
img.src='/api/users?abc=123'
```

- 跨域：浏览器同源策略引起的接口调用问题

```
// proxy.js
const express = require('express')
const app = express()
app.use(express.static(__dirname + '/'))
module.exports = app
```

```
// index.js
const api = require('./api')
const proxy = require('./proxy')
```

```
// 或者通过baseURL方式
axios.defaults.baseURL = 'http://localhost:4000'
```

- 浏览器抛出跨域错误

```
Access to XMLHttpRequest at 'http://localhos (index):1
t:3000/users' from origin 'http://localhost:3001' has
been blocked by CORS policy: No 'Access-Control-Allow-
Origin' header is present on the requested resource.
▶Uncaught (in promise) Error: Network      spread.js:25
Error
    at e.exports (spread.js:25)
    at XMLHttpRequest.l.onerror (spread.js:25)
```

- 常用解决方案：

1. JSONP(JSON with Padding)，前端+后端方案，绕过跨域

> 前端构造script标签请求指定URL（由script标签发出的GET请求不受同源策略限制），服务器返回一个函数执行语句，该函数名称通常由查询参callback的值决定，函数的参数为服务器返回的json数据。该函数在前端执行后即可获取数据。

2. 代理服务器

> 请求同源服务器，通过该服务器转发请求至目标服务器，得到结果再转发给前端。
>
> 前端开发中测试服务器的代理功能就是采用的该解决方案，但是最终发布上线时如果web应用和接口服务器不在一起仍会跨域。

3. CORS(Cross Origin Resource Share) - 跨域资源共享，后端方案，解决跨域

> 预检请求
>
> https://www.jianshu.com/p/b55086cbd9af

原理：cors是w3c规范，真正意义上解决跨域问题。它需要服务器对请求进行检查并对响应头做相应处理，从而允许跨域请求。

具体实现：

- 响应简单请求: 动词为get/post/head，没有自定义请求头，Content-Type是application/x-www-form-urlencoded，multipart/form-data或text/plain之一，通过添加以下响应头解决：

```
res.setHeader('Access-Control-Allow-Origin', 'http://localhost:3000')
```

> 该案例中可以通过添加自定义的x-token请求头使请求变为preflight请求

```
// index.html
axios.defaults.baseURL = 'http://localhost:3000';
axios.get("/users", {headers:{'X-Token':'jilei'}})
```

- 响应preflight请求，需要响应浏览器发出的options请求（预检请求），并根据情况设置响应头：

```
        else if (method == "OPTIONS" && url == "/api/users") {
          res.writeHead(200, {
            "Access-Control-Allow-Origin": "http://localhost:3000",
            "Access-Control-Allow-Headers": "X-Token,Content-Type",
            "Access-Control-Allow-Methods": "PUT"
          });
          res.end();
    }
```

则服务器需要允许x-token，若请求为post，还传递了参数：

```
// index.html
const ret = await axios.post("/api/users", {foo:'bar'}, {headers:{'X-
Token':'jilei'}})
// http-server.js
 else if ((method == "GET" || method == "POST") && url == "/users") {}
```

> 则服务器还需要允许content-type请求头

- 如果要携带cookie信息，则请求变为credential请求：

```
// index.js
// 预检options中和/users接口中均需添加
res.setHeader('Access-Control-Allow-Credentials', 'true');
// 设置cookie
res.setHeader('Set-Cookie', 'cookie1=va222;')

// index.html
// 观察cookie存在
console.log('cookie',req.headers.cookie)
// ajax服务
axios.defaults.withCredentials = true
```

Proxy代理模式

```
var express = require('express');
const proxy = require('http-proxy-middleware')

const app = express()
app.use(express.static(__dirname + '/'))
app.use('/api', proxy({ target: 'http://localhost:4000', changeOrigin: false
}));
module.exports = app
```

对比一下nginx 与webpack devserver

```
// vue.config.js
module.exports = {
  devServer: {
    disableHostCheck: true,
    compress: true,
    port: 5000,
    proxy: {
      '/api/': {
        target: 'http://localhost:4000',
        changeOrigin: true,
      },
    },
  },
}
```

nginx

```
server {
    listen       80;
    # server_name  www.josephxia.com;
    location / {
        root   /var/www/html;
        index  index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    location /api {
            proxy_pass  http://127.0.0.1:3000;
            proxy_redirect     off;
            proxy_set_header   Host             $host;
            proxy_set_header   X-Real-IP        $remote_addr;
            proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
    }
}
```

Bodyparser

- application/x-www-form-urlencoded

```
<form action="/api/save" method="post">
  <input type="text" name="abc" value="123">
  <input type="submit" value="save">
</form>
```

```
// api.js
else if (method === "POST" && url === "/api/save") {
  let reqData = [];
  let size = 0;
  req.on('data', data => {
    console.log('>>>req on', data);
    reqData.push(data);
    size += data.length;
  });
  req.on('end', function () {
    console.log('end')
    const data = Buffer.concat(reqData, size);
    console.log('data:', size, data.toString())
    res.end(`formdata:${data.toString()}`)
  });
}
```

- application/json

```
await axios.post("/api/save", {
            a: 1,
            b: 2
        })
```

```
// 模拟application/x-www-form-urlencoded
await axios.post("/api/save", 'a=1&b=3', {
            headers: {
                'Content-Type': 'application/x-www-form-urlencoded',
            },
        })
```

上传文件

```
// Stream pipe
request.pipe(fis)
response.end()
```

```
// Buffer connect
request.on('data',data => {
  chunk.push(data)
  size += data.length
  console.log('data:',data ,size)
})
request.on('end',() => {
  console.log('end...')
  const buffer = Buffer.concat(chunk,size)
  size = 0
  fs.writeFileSync(outputFile,buffer)
  response.end()
})
```

```
// 流事件写入
request.on('data', data => {
  console.log('data:',data)
  fis.write(data)
})
request.on('end', () => {
  fis.end()
  response.end()
})
```

实战一个爬虫

原理：服务端模拟客户端发送请求到目标服务器获取页面内容并解析，获取其中关注部分的数据。

```
// spider.js
const originRequest = require("request");
const cheerio = require("cheerio");
const iconv = require("iconv-lite");

function request(url, callback) {
  const options = {
    url: url,
    encoding: null
```

```
  };
  originRequest(url, options,callback);
}

for (let i = 100553; i < 100563; i++) {
  const url = `https://www.dy2018.com/i/${i}.html`;
  request(url, function(err, res, body) {
    const html = iconv.decode(body, "gb2312");
    const $ = cheerio.load(html);
    console.log($(".title_all h1").text());
  });
}
```

实现一个即时通讯IM

- Socket实现

原理：Net模块提供一个异步API能够创建基于流的TCP服务器，客户端与服务器建立连接后，服务器可以获得一个全双工Socket对象，服务器可以保存Socket对象列表，在接收某客户端消息时，推送给其他客户端。

```
// socket.js
const net = require('net')
const chatServer = net.createServer()
const clientList = []
chatServer.on('connection',client => {
    client.write('Hi!\n')
    clientList.push(client)
    client.on('data',data => {
        console.log('receive:',data.toString())
        clientList.forEach(v => {
            v.write(data)
        })
    })
})
chatServer.listen(9000)
```

通过Telnet连接服务器

```
telnet localhost 9000
```

- Http实现

原理：客户端通过ajax方式发送数据给http服务器，服务器缓存消息，其他客户端通过轮询方式查询最新数据并更新列表。

```
<html>
```

```
<head>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
</head>

<body>
    <div id="app">
        <input v-model="message">
        <button v-on:click="send">发送</button>
        <button v-on:click="clear">清空</button>
        <div v-for="item in list">{{item}}</div>
    </div>

    <script>
        const host = 'http://localhost:3000'
        var app = new Vue({
            el: '#app',
            data: {
                list: [],
                message: 'Hello Vue!'
            },
            methods: {
                send: async function () {
                    let res = await axios.post(host + '/send', {
                        message: this.message
                    })
                    this.list = res.data
                },
                clear: async function () {
                    let res = await axios.post(host + '/clear')
                    this.list = res.data
                }
            },
            mounted: function () {
                setInterval(async () => {
                    const res = await axios.get(host + '/list')
                    this.list = res.data
                }, 1000);
            }
        });
    </script>
</body>
</html>
```

```
const express = require('express')
const app = express()
const bodyParser = require('body-parser');
const path = require('path')
```

```
app.use(bodyParser.json());

const list = ['ccc', 'ddd']

app.get('/', (req,res)=>{
    res.sendFile(path.resolve('./index.html'))
})

app.get('/list', (req, res) => {
    res.end(JSON.stringify(list))
})

app.post('/send', (req, res) => {
    list.push(req.body.message)
    res.end(JSON.stringify(list))
})

app.post('/clear', (req, res) => {
    list.length = 0
    res.end(JSON.stringify(list))
})

app.listen(3000);
```

- Socket.IO实现
    - 安装：`npm install --save socket.io`
    - 两部分：nodejs模块，客户端js

```
// 服务端：chat-socketio.js
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);

app.get('/', function(req, res){
  res.sendFile(__dirname + '/index.html');
});

io.on('connection', function(socket){
  console.log('a user connected');

  //响应某用户发送消息
  socket.on('chat message', function(msg){
      console.log('chat message:' + msg);

    // 广播给所有人
    io.emit('chat message', msg);
    // 广播给除了发送者外所有人
```

```
    // socket.broadcast.emit('chat message', msg)
  });

  socket.on('disconnect', function(){
    console.log('user disconnected');
  });
});

http.listen(3000, function(){
  console.log('listening on *:3000');
});
```

```
// 客户端：index.html
<!DOCTYPE html>
<html>
  <head>
    <title>Socket.IO chat</title>
    <style>
      * {
        margin: 0;
        padding: 0;
        box-sizing: border-box;
      }
      body {
        font: 13px Helvetica, Arial;
      }
      form {
        background: #000;
        padding: 3px;
        position: fixed;
        bottom: 0;
        width: 100%;
      }
      form input {
        border: 0;
        padding: 10px;
        width: 90%;
        margin-right: 0.5%;
      }
      form button {
        width: 9%;
        background: rgb(130, 224, 255);
        border: none;
        padding: 10px;
      }
      #messages {
        list-style-type: none;
        margin: 0;
        padding: 0;
```

```
      }
      #messages li {
        padding: 5px 10px;
      }
      #messages li:nth-child(odd) {
        background: #eee;
      }
    </style>
  </head>
  <body>
    <ul id="messages"></ul>
    <form action="">
      <input id="m" autocomplete="off" /><button>Send</button>
    </form>

    <script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js">
</script>
    <script src="http://libs.baidu.com/jquery/2.1.1/jquery.min.js"></script>
    <script>
      $(function() {
        var socket = io();
        $("form").submit(function(e) {
          e.preventDefault(); // 避免表单提交行为
          socket.emit("chat message", $("#m").val());
          $("#m").val("");
          return false;
        });

        socket.on("chat message", function(msg) {
          $("#messages").append($("<li>").text(msg));
        });
      });
    </script>
  </body>
</html>
```

> Socket.IO库特点：
>
> - 源于HTML5标准
> - 支持优雅降级
>   - WebSocket
>   - WebSocket over FLash
>   - XHR Polling
>   - XHR Multipart Streaming
>   - Forever Iframe
>   - JSONP Polling

Https（安全课再讲）

- 创建证书

```
# 创建私钥
openssl genrsa -out privatekey.pem 1024
# 创建证书签名请求
openssl req -new -key privatekey.pem -out certrequest.csr
# 获取证书，线上证书需要经过证书授证中心签名的文件；下面只创建一个学习使用证书
openssl x509 -req -in certrequest.csr -signkey privatekey.pem -out
certificate.pem
# 创建pfx文件
openssl pkcs12 -export -in certificate.pem -inkey privatekey.pem -out
certificate.pfx
```

Http2（优化的时候讲）

- 多路复用 - 雪碧图、多域名CDN、接口合并
  - 官方演示 - https://http2.akamai.com/demo
  - 多路复用允许同时通过单一的 HTTP/2 连接发起多重的请求-响应消息；而HTTP/1.1协议中，浏览器客户端在同一时间，针对同一域名下的请求有一定数量限制。超过限制数目的请求会被阻塞**
- 首部压缩
  - http/1.x 的 header 由于 cookie 和 user agent很容易膨胀，而且每次都要重复发送。http/2 使用 encoder 来减少需要传输的 header 大小，通讯双方各自 cache一份 header fields 表，既避免了重复 header 的传输，又减小了需要传输的大小。高效的压缩算法可以很大的压缩 header，减少发送包的数量从而降低延迟
- 服务端推送
  - 在 HTTP/2 中，服务器可以对客户端的一个请求发送多个响应。举个例子，如果一个请求请求的是index.html，服务器很可能会同时响应index.html、logo.jpg 以及 css 和 js 文件，因为它知道客户端会用到这些东西。这相当于在一个 HTML 文档内集合了所有的资源