NodeJS课程大纲

day01

一、什么是NodeJS?

> *1、Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行环境。*
>
> *2、Node.js 使用了一个事件驱动、非阻塞式 I/O 的模型，使其轻量又高效。*
>
> *3、Node.js 的包管理器 npm，成为世界上最大的开放源代码的生态系统。*
>
> *简单的说 Node.js 就是运行在服务端的 JavaScript。*
>
> *扩展；*
>
> *I ： input即输入端口*
>
> *o ： output即输出端口*
>
> *CPU与外部设备、存储器的链接和数据交换都需要通过接口设备来实现，前者称为I/O接口，后者被称为存储器接口*

二、什么是V8引擎？

> *每一个浏览器都有一个内核，内核中有引擎。引擎分为：渲染引擎(渲染DOM) 和 脚本引擎(运行脚本语言)*
>
> *脚本引擎中最流行的就是chrome中的V8引擎*

三、Node可以做什么？

> *1、node可以解析js代码(因为没有浏览器安全级别的限制)因此提供了许多系统级别的API*
>
> *2、node可以编写独立的服务端应用，无需借助任何web服务器，可以连接文件系统，以及操作数据库*
>
> *3、node一般在实际应用中用来做中间层服务器使用*

注意

注意：

在node中无法使用window对象下面的一些方法，因为node中没有DOM 和 BOM的概念，同时node中也有一些属性浏览器无法使用 例如：process global等对象

四、node的优点和适用的项目？

> *优点：*
>
> *高性能、速度快、效率高 适合做高并发的项目(I/O密集型的应用)*
>
> *缺点：*
>
> *不适合做大量的运算应用(CPU密集的应用)*

五、扩展版本号

> *6.11.4*
>
> *第一个是大版本号*
>
> *第二个是小版本号*
>
> *第三个是补丁版本号*
>
> *版本问题：LTS长期稳定版本 Current最新版本 偶数为稳定版本 基础为非稳定版本*

六、node交互模式

以前我们运行js必须基于浏览器这个环境，那么现在我们还可以在终端运行node的文件

> *1、建立一个hellow.js文件*
>
> *2、运行js文件*
>
> *node 文件名称*
>
> *3、进入交互模式*
>
> *node*
>
> *4、退出交互模式*
>
> *ctrl+d*

注意：

node的环境下 没有dom 和bom的概念

node里面的方法有的在浏览器中也不能运行 例如 process进程

process

> *process.env是一个对象，我们可以通过其属性名来获取具体的环境变量值*
>
> *设定一个环境变量，以达到简单区分不同机器，从而针对生产/开发环境运行不同的效果*
>
> *每个系统的环境变量几乎都不一样，我们可以利用环境变量中具体某个特定的值来区分不同的机器*
>
> *set:set命令作用主要是显示系统中已经存在的shell变量，以及设置shell变量的新变量值*
>
> *process.argv:获取命令行参数*
>
> *返回值是一个数组*
>
> *参数1：node的绝对路径*
>
> *参数2：文件的绝对路径*
>
> *参数3:…arg*
>
> *__filename:获取当前运行文件的目录，绝对路径*
>
> *__dirname：当前运行文件的绝对路径*

七、创建一个简单的服务器

> *1、require:引入相应模块*
>
> *2、createServer():创建服务器*
>
> *3、listen:绑定端口号 参数2个 第一个参数端口号 第二个参数地址*
>
> *4、request,response：接受和响应数据*

```
1.  //1、引入http模块
2.   var http = require("http");
3.
4.  //2、创建服务器
```

```
    http.createServer(function(request,response){
    //发送HTTP头部
    //状态值为200
    //内容类型：text/plain
    response.writeHead(200,{"Content-type":text/plain});
    //发送相应数据
    response.end("hellow")
    }).listen(8888);
    //终端打印信息
    console.log("Server running at http://127.0.0.1:8888/");
```

content-type:类型

> *1、text/plain:文本类型*
>
> *2、text/html:html文档*
>
> *3、image/* :图片类型*
>
> *4、 application/x-javascript ： javascript类型*
>
> *5、text/css ： css类型*
>
> *6、application/json; charset=utf-8*

八、request&&response常用的属性

> *req:包含请求的信息，例如请求头等*
>
> *req.url:请求的地址*
>
> *req.method:请求的方式*
>
> *req.header：请求头*
>
> *res:服务器的响应*
>
> *res.statusCode:设置状态码*
>
> *res.setHeader():设置响应头，其他Content-type：是设置浏览器相应数据的类型*
>
> 1. *res.writeHead(); statusCode与setHeader的综合写法*
>
> *res.write():写入响应数据,只能写入字符串*
>
> *res.end():结束响应，并返回数据*

九、node中模块的分类

> *1、node核心模块，require可以直接引入*
>
> *2、自定义模块：需要自己写的模块 用module.exports导出 require引入*
>
> *3、第三方模块：通过npm install 来安装 然后在require引入*

十、什么是模块化？

> *模块化是将一个功能拆分成若干个小功能的方法*
>
> *优点：*
>
> *代码复用、便于维护*
>
> *模块化划分的原则：*
>
> *1、功能复用次数较多*
>
> *2、功能逻辑较为独立*
>
> *扩展：*
>
> ***什么是高内聚、低耦合?***
>
> *高内聚：代表模块的独立性、独立性越强、内聚度越高*
>
> *耦合度：模块之间的关系，关系越紧密、耦合度越高*

现有的模块化规范：

AMD CMD COMMON ES6module

前端 前端 后端 前后端

require sea.js Node js引擎

异步 异步 同步 同步

十一、npm

1、什么是npm?

1、什么是npm:

> npm是node的包管理仓库
>
> npm是一个网站
>
> npm是一个命令

2、npm 常用命令

> 查看版本 npm -v
>
> 安装模块 npm install <module name>
>
> 全局安装 npm install <module name> -g
>
> 卸载模块 npm uninstall <module name>
>
> 更新模块 npm update <module name>
>
> 搜索模块 npm search <module name>
>
> 清除缓存 npm cache clear
>
> 查看包信息 npm info <包名>
>
> 查看包文档 npm docs <包名>
>
> 查看包版本信息 npm info <包名> versions
>
> 安装指定版本 npm install <包名>@版本号 -S
>
> 查看全局安装目录 npm root -g
>
> 修改存储目录 npm config set prefix “d\xxxx”

3、扩展nrm:

> 1、安装nrm: npm install nrm -g
>
> 2、查看可用的源：nrm ls
>
> 3、切换源：nrm use 源的名字
>
> 4、添加公司私有源 nrm add <源名称> http://xxxxx

十二、如何自己写的包上传到npm

> *1、注册一个npm账号 进行账号邮箱验证*
>
> *2、npm init name值一定要是全网唯一的*
>
> *3、当输入npm init时 会帮你在文件夹下建立一个package.json文件 当有这个文件的时候就代表你的目录已经是一个包了*
>
> *4、注册npm adduser 注意密码是不可见的*
>
> *5、登录 npm login*
>
> *6、npm publish 上传文件*
>
> *7、下载文件 npm install 文件名*

十三、yarn简介

1、npm的缺点:

> *1、包是同步下载的 速度非常慢*
>
> *2、在一个项目中下载完成后在另一个项目中还要继续下载*

2、yarn的优点:

> *1、包下载是异步的比npm快*
>
> *2、yarn有效的保证版本号不容易出错*
>
> *3、yarn本地包会有缓存，安装本地包会非常快*

3、yarn的常用命令:

> - *npm install yarn -g 安装yarn*
> - *npm install nrm -g 安装nrm*
> - *nrm ls 查看可用源*
> - *nrm use 源名 切换源*
> - *yarn init == npm init*
> - *yarn global add <name> == npm install -g <name>*

> - *yarn global bin == npm -g bin*
> - *yarn add 包名 == npm install 包名 —save*
> - *yarn add 包名 -dev == npm install 包名 —dev-save*
> - *yarn update 包名 == npm undate 包名 更新包*
> - *yarn remove 包名 == npm uninstall 包名 删除包*
> - *yarn bin 全局安装目录*
> - *yarn cache ls 查看缓存*
> - *yarn clear 清除缓存*
> - *yarn install 安装所有包*

十四、开发环境、测试环境、生产环境的

> *1、开发环境：项目尚且在编码阶段，我们的代码一般在开发环境中 不会在生产环境中，生产环境组成：操作系统 ，web服务器 ，语言环境。 php 。 数据库 。 等等*
>
> *2、测试环境：项目完成测试，修改bug阶段*
>
> *3、生产环境：项目数据前端后台已经跑通，部署在阿里云上之后，有客户使用，访问，就是网站正式运行了。*
>
> *三个环境也可以说是系统开发的三个阶段：开发->测试->上线*

save && —save-dev的区别

> *可分别将依赖（插件）记录到package.json中的dependencies和devDependencies下面。*
>
> *dependencies下记录的是项目在运行时必须依赖的插件，常见的例如react jquery等，即及时项目打包好了、上线了，这些也是需要用的，否则程序无法正常执行*
>
> *devDependencies下记录的是项目在开发过程中使用的插件，一旦项目打包发布、上线了之后，devDependencies中安装的东西就都没有用了*
>
> *如果模块是在开发环境中使用，那么我们安装依赖的时候需要—dev*

day02

一、url模块的基本使用

- *url.parse*
- *url.format*
- *url.resolve*

```
const url = require("url");

const path = "https://www.baidu.com/s?wd=%20node%20request%20post%E8%AF%B7%E6%B1%82&rsv_spt=1&rsv_iqid=0xb9778e3c000b4a9e&issp=1&f=8&rsv_bp=1&rsv_idx=2&ie=utf-8&rqlang=cn&tn=baiduhome_pg&rsv_enter=1&rsv_dl=tb&oq=request%2520post%25E8%25AF%25B7%25E6%25B1%2582&inputT=1575&rsv_t=3da2Tn%2F7uIuHWTsROFo6yvwMvultNv5wmOTlrQiVQ6KLFbNtnrLXkdd9HWuhNlyq7zNM&rsv_pq=feaadbf200091d61&rsv_sug3=35&rsv_sug2=0&rsv_sug4=2259";
const urlObject = url.parse(path,true);
//console.log(url.format(urlObject));

console.log(url.resolve("/a/b/c","d"));
console.log(url.resolve("/a/b/c","/d"));
```

二、queryString模块的基本使用

- *queryString.escape()*
- *queryString.parse()*
- *queryString.stringify()*
- *queryString.unescape()*

```
const qs = require("querystring");

var str = "name*alley#age*20";
var obj = qs.parse(str,"#","*");
//console.log(qs.stringify(obj,"@","!"))

var str = "name=alley=sex=19"
var key = qs.escape("name=alley=sex");
```

```
console.log(qs.unescape(key))
```

三、events模块的基本使用

> - *on:绑定事件*
> - *once:只执行一次事件*
> - *emit:执行匹配的事件*
> - *prependListener：将当前事件放到队列的最前面*
> - *removeALLListeners:移除所有事件*
> - *removeListener:移除指定事件监听*
> - *defaultMaxListeners：events默认的绑定事件只能绑定10个如果想绑定多个需要配置*

```
const EventEmitter = require("events");
class MyEventEmitter extends EventEmitter{};
const myEventEmitter = new MyEventEmitter();



/*
 on:事件订阅
 emit：事件触发
 removeListener：移除一个时间
 removeAllListeners：移除所有事件
 once:只绑定一次函数

*/


function fn1(val){
 console.log(111,val)
}

function fn2(val){
 console.log(222,val)
}

function fn3(val){
 console.log(333,val)
}

myEventEmitter.once("handle",fn1)
```

```
myEventEmitter.once("handle",fn1)
myEventEmitter.once("handle",fn2)
myEventEmitter.once("handle",fn3)

//myEventEmitter.removeListener("handle",fn1)
//myEventEmitter.removeAllListeners("handle")
myEventEmitter.emit("handle","alley")
myEventEmitter.emit("handle","alley")
```

四、File System模块基本使用

> - *得到文件与目录的信息：stat*
> - *创建一个目录：mkdir*
> - *创建文件并写入内容：writeFile,appendFile*
> - *读取文件的内容：readFile readFileSync*
> - *列出目录的东西：readdir*
> - *重命名目录与文件：rename*
> - *删除目录与文件：rmdir,unlink*

```
const fs = require("fs");
const path = require("path");
// fs.stat("./index.js",(err,data)=>{
// console.log(data.isDirectory());
// })

// var data = fs.statSync("./index.js");
// console.log(data.isFile())


// fs.mkdir(path.resolve(__dirname,"./demo"),(err)=>{});

// fs.writeFile(path.resolve(__dirname,"./demo/index.txt"),"abc",(err)=>{})

// fs.readFile(path.resolve(__dirname,"./demo/index.txt"),(err,data)=>{
// console.log(data+"");
// })


//fs.rename(path.resolve(__dirname,"./demo/index.txt"),path.resolve(__dirname,"./demo/list.txt"),(err)=>{})

```

```
22.
23. // fs.readdir(path.resolve(__dirname,"../../随堂案例"),(err,list)=>{
24. // console.log(list)
25. // })
26.
27.
28. //fs.unlink(path.resolve(__dirname,"./demo/list.txt"),()=>{})
29.
30.
31. // fs.rmdir(path.resolve(__dirname,"./demo"),(err)=>{})
32. /*
33.  stat:判断文件类型
34.  isFile：判断是否是一个文件
35.  isDirectory:判断是否是一个文件夹
36.
37.  mkdir：创建文件夹
38.
39.  writeFile(path,content,callback)：创建并写入
40.  path：路径
41.  content:内容
42.  callback：回调
43.
44.  readFile(path,callback(err,data)):读操作
45.
46.  rename:重命名文件或者文件夹
47.  oldPath
48.  newpath
49.
50.  readdir:列出文件夹中所有文件
51.
52.  unlink:删除文件
53.
54.  rmdir:删除文件夹
55. */
```

五、stream文件流

> - *fs.createReadStream():创建可读文件流 参数1：已有文件*
> - *fs.createWriteStream():创建可写文件流 参数1：文件名称*
> - *pipe:管道*

```
1.
2.  //链式使用pipe
3.  const fs = reqiure("fs");
4.  //压缩模块
5.  const zlib = require("zlib");
6.
7.  const read = fs.createReadStream("");
8.  const write = fs.createWriteStream("");
9.  //先进行压缩然后在给write文件 16进制
10. read.pipe(zlib.createGzip()).pipe(write);
```

六、path模块的基本使用

- *path.join*
- *path.resolve*

```
1.  const path = require("path");
2.
3.  /*
4.   join:路径拼接
5.   resolve:将参数转换为绝对路径
6.
7.  */
8.  // console.log(path.join("/a","b","c"));
9.  // console.log(path.join(__dirname,"./index.js"));
10.
11. // console.log(path.resolve(__dirname,"./index.js"));
```

七、process

- *process.env*
- *process.argv*

```
1.  console.log(process.env)
2.
3.  var arr = process.argv;
4.
5.  if(arr.indexOf("-v") !== -1){
6.   console.log("1.11.1");
7.  }else{
```

```
} else {
  console.log("help....")
 }
```

八、http模块

> - *http.createServer*
> - *http.get*
> - *http.post*
> - *request*
> - *cheerio小爬虫*

createServer

```
const http = require("http");

const server = http.createServer((req,res)=>{

 //console.log(req.url,req.method)
 console.log(req.headers)
 // res.statusCode = 404;
 // res.setHeader("content-type","text/plain;charset=utf8");

 res.writeHead(200,{"content-type":"text/plain;chartset=utf8"})
 res.write("我最近很好")
 res.write("不用你担心")
 res.write("再见")
 res.end("NodeJS")
})

server.listen(9000,()=>{
 console.log("server address:localhost:9000")
})
```

http.get

```
const http = require("http");

// const path = "http://www.mobiletrain.org/?pinzhuanbdtg=biaoti";(网页)
// http.get(path,(res)=>{
// var str = "";
// //因为数据接收是一块一块接收的 类似于流水一样
// res.on("data",(data)=>{
```

```
// str += data;
// })

// res.on("end",()=>{
// console.log(str);
// })
// })


var path = "http://m.maoyan.com/ajax/movieOnInfoList?token=";//接口

http.get(path,(res)=>{
 var str = "";

 res.on("data",(data)=>{
 str += data;
 })

 res.on("end",()=>{
 console.log(JSON.parse(str));
 })
})
```

request

```
const request = require("request");
const qs = require("querystring");

const options = {
 url: "https://app2.motie.com/category/detail?free=0&finish=0&group=1&sortId=&page=1&pageSize=10",
 method: "POST",
 headers: {
 "os": "wap"
 },
 body: qs.stringify({
 free: 0,
 finish: 0,
 group: 1,
 sortId: "",
 page: 1,
```

```
  pageSize: 10,
  })
}


request(options, (err, res, body) => {
 console.log(JSON.parse(body));
})


/*
  1、cnpm install request -S


  request(options,callback(err,res,body))

  err:错误
  res:整个响应体
  body：响应的数据


  options = {
  method:"",
  url:"",
  headers:{},
  body:{请求的参数}
  }
*/
```

小爬虫

```
const http = require("http");
//可以让我们以jQuery语法来进行dom查找 并不是对dom操作
const cheerio = require("cheerio");
const fs = require("fs");
const path = require("path");
const url = "http://www.mobiletrain.org/?pinzhuanbdtg=biaoti";

http.get(url,(res)=>{

 var str = "";

```

```
12.   res.on("data",(data)=>{
13.   str += data;
14.   })
15.
16.
17.   res.on("end",()=>{
18.   //将str封装到$符号中去
19.   var $ = cheerio.load(str);
20.   var banner_up_left = $(".banner_up_left>a");
21.   var arr = [];
22.   for(var i=0;i<banner_up_left.length;i++){
23.   var obj = {};
24.   obj.id = i;
25.   obj.text = banner_up_left.eq(i).find("span").text();
26.   arr.push(obj);
27.   }
28.
29.   fs.readFile(path.join(__dirname,"./data/index.json"),(err,data)=>{
30.   var dataList = JSON.parse(data+"")
31.
32.   if(!dataList.data){
33.   dataList.data = [];
34.   }
35.
36.
37.   dataList.data = arr;
38.
39.   fs.writeFile(path.join(__dirname,"./data/index.json"),JSON.stringify(dataList),(err)=>
{})
40.   })
41.   })
42.
43.  })
```

九、路由

> - *什么是路由？*
> - *路由的作用*
> - *前端路由与后端路由*
> - *路由核心属性*

- *supervisor*

```
const http = require("http");
const path = require("path");
const fs = require("fs");
const url = require("url");
const server = http.createServer((req, res) => {
 const { pathname, query } = url.parse(req.url, true);

 if (pathname == "/") {

 fs.readFile(path.join(__dirname, "./public/index.html"), (err, data) => {
 res.writeHead(200, { "contentType": "text/html;charset=utf8" });
 res.end(data);
 })

 } else if (pathname == "/order") {
 fs.readFile(path.join(__dirname, "./public/html/order.html"), (err, data) => {
 res.writeHead(200, { "contentType": "text/html;charset=utf8" });
 res.end(data);
 })
 } else if (pathname == "/list") {
 fs.readFile(path.join(__dirname, "./public/html/list.html"), (err, data) => {
 res.writeHead(200, { "contentType": "text/html;charset=utf8" });
 res.end(data);
 })
 } else if (pathname == "/css/index.css") {
 fs.readFile(path.join(__dirname, "./public/css/index.css"), (err, data) => {
 res.writeHead(200, { "content-type": "text/css;charset=utf8" });
 res.end(data);
 })
 } else if (pathname == "/js/index.js") {
 fs.readFile(path.join(__dirname, "./public/js/index.js"), (err, data) => {
 res.writeHead(200, { "content-type": "application/x-javascript;charset=utf8" });
 res.end(data);
 })
 } else if (/\/img\/(.*)\.(jpg|png|gif)/.test(pathname)) {
 fs.readFile(path.join(__dirname, "./public/img/" + RegExp.$1 + "." + RegExp.$2), (err,
data) => {
 res.writeHead(200, { "content-type": "image/" + RegExp.$2 });
 res.end(data);
```

```
39.   })
40.   } else if (pathname == "/users/register") {
41.   let { username, password } = query;
42.   fs.readFile(path.join(__dirname, "./public/data/user.json"), (err, data) => {
43.   let userData = JSON.parse(data.toString());
44.   var bStop = true;
45.   for (var i = 0; i < userData.data.length; i++) {
46.   if (userData.data[i].username == username) {
47.   bStop = false;
48.   break;
49.   }
50.   }
51.
52.   if (bStop) {
53.   userData.data.push({ username, password });
54.
55.   fs.writeFile(path.join(__dirname, "./public/data/user.json"), JSON.stringify(userData),
(err) => {
56.   if (!err) {
57.   res.writeHead(200, { "content-Type": "application/json;charset=utf-8" })
58.   res.end(JSON.stringify({
59.   code: 200,
60.   errMsg: "",
61.   data: {
62.   code: 1,
63.   info: "注册成功"
64.   }
65.   }))
66.   }
67.
68.   })
69.
70.   } else {
71.   res.writeHead(200, { "content-Type": "application/json;charset=utf-8" })
72.   res.end(JSON.stringify({
73.   code: 200,
74.   errMsg: "",
75.   data: {
76.   code: 0,
77.   info: "用户名重复"
78.   }
```

```
  }))
  }


  })

  }
})

server.listen(9000, () => {
 console.log("server address:127.0.0.1:9000")
})
```

十、封装路由级静态资源处理

```

const url = require("url");
const fs = require("fs");
const path = require("path");
const qs = require("querystring")
//收集事件
const routerMap = {
 get:{},
 post:{}
}

const router = function(req,res){
 //给res添加一个json方法
 res.json = function(obj){
 res.end(JSON.stringify(obj))
 }

 //处理静态资源
 handleStatic(req,res);


 //获取用户请求的方式
 var method = req.method.toLowerCase();
 //获取用户请求的地址
 var {pathname,query} = url.parse(req.url,true);
```

```
26.
27.   if(method === "get"){
28.   if(routerMap.get[pathname]){
29.   //将query的值赋值给req.query
30.   req.query = query;
31.   routerMap.get[pathname](req,res)
32.   }else{
33.   res.end("404")
34.   }
35.
36.
37.   }else if(method === "post"){
38.   if(routerMap.post[pathname]){
39.   var str = "";
40.
41.   //获取post传递的参数
42.   req.on("data",(data)=>{
43.   str += data;
44.   })
45.
46.   req.on("end",()=>{
47.   req.body = qs.parse(str);
48.   routerMap.post[pathname](req,res)
49.   })
50.
51.   }
52.   }
53. }
54.
55. //注册事件
56. router.get = function(path,callback){
57.   routerMap.get[path] = callback;
58. }
59. //注册事件
60. router.post = function(path,callback){
61.   routerMap.post[path] = callback;
62. }
63.
64.
65.
66. //处理所有的静态资源访问
```

```
function handleStatic(req,res){
 var {pathname} = url.parse(req.url,true);
 //获取文件的后缀
 var ext = pathname.substring(pathname.lastIndexOf("."));

 if(pathname ==="/"){
 res.writeHead(200,{"content-type":"text/html;charset=utf8"});
 res.end(getFile(path.join(__dirname,"../public/index.html")))
 }else if(ext === ".css"){
 res.writeHead(200,{"content-type":"text/css;charset=utf8"});
 res.end(getFile(path.join(__dirname,"../public",pathname)));
 }else if(ext === ".js"){
 res.writeHead(200,{"content-type":"application/x-javascript;charset=utf8"});
 res.end(getFile(path.join(__dirname,"../public",pathname)));

 }else if(/.*\.(jpg|png|gif)/.test(ext)){
 res.writeHead(200,{"content-type":`image/${RegExp.$1};charset=utf8`});
 res.end(getFile(path.join(__dirname,"../public",pathname)));
 }else if(ext === ".html"){
 res.writeHead(200,{"content-type":"text/html;charset=utf8"});
 res.end(getFile(path.join(__dirname,"../public",pathname)));
 }
}


function getFile(filePath){
 return fs.readFileSync(filePath);
}

module.exports = router;
```

十一、mongoodb级mongoose的基本使用

1、mongodb安装

> - *下载地址:https://www.mongodb.com/download-center/community*
> - *下一步傻瓜式安装*
> - *配置环境变量*
> - *启动mongod —dbpath c:\data\db 开启数据库服务器（开启）*
> - *终端末尾显示27017代表成功*

2、关系型数据库与非关系数据库区别

> *优点：*
> *1）成本：nosql数据库简单易部署，基本都是开源软件，不需要像使用oracle那样花费大量成本购买使用，相比关系型数据库价格便宜。*
> *2）查询速度：nosql数据库将数据存储于缓存之中，关系型数据库将数据存储在硬盘中，自然查询速度远不及nosql数据库。*
> *3）存储数据的格式：nosql的存储格式是key,value形式、文档形式、图片形式等等，所以可以存储基础类型以及对象或者是集合等各种格式，而数据库则只支持基础类型。*
> *4）扩展性：关系型数据库有类似join这样的多表查询机制的限制导致扩展很艰难。*
> *缺点：*
> *1）维护的工具和资料有限，因为nosql是属于新的技术，不能和关系型数据库10几年的技术同日而语。*
> *2）不提供对sql的支持，如果不支持sql这样的工业标准，将产生一定用户的学习和使用成本。*
> *3）不提供关系型数据库对事物的处理。*

3、mongodb常用命令

> *mongod —dbpath c:\data\db 开启数据库服务器（开启）*
>
> *27017——》mongodb的端口号 http:80 || 8080 https:443*
>
> *mongo 进入数据库服务器*
>
> *show dbs 查看数据库服务器中有多少数据库*
>
> *use 数据库名称 创建/切换数据库 如果数据库中没有表的情况下 数据库并不会真正的创建*
>
> *db || db.getName() 查看当前使用的数据库*
>
> *db.dropDatabase() 删除数据库*
>
> *表的操作*
>
> *db.createCollection(“表名称”) 创建表*
>
> *db.getCollectionNames() 获取数据库中所有的表*
>
> *db.getCollection(‘表名称’) 使用某一张表*
>
> *db.表名.drop() 删除某一张表*
>
> *增*
>
> *db.表名.save({字段})*

删

db.表名.remove({}) 删除所有

db.表名.remove({字段}) 删除指定字段

改

db.表名.update({字段},{$set:{字段}})

db.表名.update({字段},{$inc:{字段(数字)}})

查

db.表名.find() 查找所有数据

db.表名.find({age:19}) 查找指定字段

db.表名.find({age:{$gt:22}}) 大于22的数据

db.表名.find({age:{$lt:22}}) 小于22的数据

db.表名.find({age:{$gte:22}}) 大于等于22的数据

db.表名.find({age:{$lte:22}) 小于等于22的数据

db.表名.find({age:{$gte:30,$lte:22})大于等于30小于等于22的数据

db.表名.find({name:/alley/}) 模糊查询 找到数据中有alley的数据

db.表名.find({age:/^alley/}) 查找首字符是alley

db.表名.find({age:/alley$/}) 查找尾字符是alley

db.表名.find({},{name:1,_id:0}) 只显示name这个字段

db.表名.find().sort({age:1}) 正序

db.表名.find().sort({age-1}) 降序

db.表名.find().limit(n) 显示多少条数据

db.表名.find().skip(n) 跳过多少条数据

db.表名.find().skip(n).limit(m) 跳过多少条 显示多少条

> *db.表名.findOne({}) 查找一条数据*
>
> *db.表名.find().count() 查询表中有多少条数据*

day03

一、NodeJS如何链接mongodb

> *NodeJS中链接Mongodb*
>
> *1、安装*
>
> *cnpm install mongodb -S*
>
> *2、创建链接的地址*
>
> *3、链接*

```
const mongoClient = require("mongodb").MongoClient;

//2、创建链接的方式
const url = "mongodb://127.0.0.1:27017";

//3、创建链接的数据库
const db_name = "gp17";

//4、链接
mongoClient.connect(url,(err,client)=>{
 if(err){
 console.log("链接失败")
 }else{
 console.log("链接成功")
 //5、链接数据库以及表
 var user = client.db(db_name).collection("user");
 }
})
```

二、mongoose的基本使用

1、mongoose与mongodb的区别

> *参考上面的*

2、NodeJS链接mongoose

```
const mongoose = require('mongoose');
//是一个构造函数 用来限制表的字段类型
const Schema = mongoose.Schema;
//链接的地址
const url = "mongodb://127.0.0.1:27017/gp17";

//链接
mongoose.connect(url,(err)=>{
 if(err){
 console.log("链接失败");
 }else{
 console.log("链接成功")
 }
})

//链接表/创建表
var StudentsSchema = new Schema({
 sname:String,
 sage:Number
})
//返回值是一个构造函数
const Students = mongoose.model("student",StudentsSchema);//如果表的名称是复数则 数据库中的表名称不会变 如果不是复数则数据库中标的名称会自动加s

// mongoose.model("students",{
// sname:String,
// sage:Number
// })
```

三、用所封装好的路由+nodeJS+Mongoose+MVC实现登录注册

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
6.   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7.   <title>Document</title>
8.   <link rel="stylesheet" href="./css/index.css">
9.  </head>
10.  <body>
11.   <h1>index页面</h1>
12.   <div>
13.   <img src="./img/u=2071817741,4218539796&fm=58&s=EE7A2FC0DE0120D6CF6D1509010070D2.jpg"
alt="">
14.   <img src="./img/u=2708490274,319030395&fm=58&s=9E3269850EF36DB70811A15B03008060.jpg"
alt="">
15.   <img src="./img/u=3473750589,1700978550&fm=58&s=4E946D84D2417CFE5C3FB5D90300D0B9.jpg"
alt="">
16.   <img src="./img/u=3508817781,1654561115&fm=58&s=6D83639716736DB3087070600300E070.jpg"
alt="">
17.   </div>
18.
19.   <form id="user">
20.   用户名: <input type="text" id="username">
21.   密码: <input type="text" id="password">
22.   <input type="submit">
23.   </form>
24.  </body>
25.  </html>
26.  <script src="./js/index.js"></script>
27.  <script src="https://cdn.bootcss.com/jquery/1.11.3/jquery.js"></script>
28.
29.  <script>
30.   $("#user").on("submit",function (e) {
31.   e.preventDefault()
32.   var username = $("#username").val();
33.   var password = $("#password").val();
34.
35.
36.   $.ajax({
37.   url:"/user/register",
38.   method:"post",
39.   data:{
40.   username,
41.   password
42.   }
```

```
  success:function(data){
  console.log(data);
  }
  })
  })

</script>
const http = require("http");
const router = require("./router")
const server = http.createServer(router)

router.post("/user/register",(req,res)=>{
 console.log(req.body);

 res.json({
 code:200,
 errMsg:''
 })
})


server.listen(9000)

const url = require("url");
const fs = require("fs");
const path = require("path");
const qs = require("querystring")
//收集事件
const routerMap = {
 get:{},
 post:{}
}

const router = function(req,res){
 //给res添加一个json方法
 res.json = function(obj){
 res.end(JSON.stringify(obj))
 }

 //处理静态资源
```

```
    handleStatic(req,res);


    //获取用户请求的方式
    var method = req.method.toLowerCase();
    //获取用户请求的地址
    var {pathname,query} = url.parse(req.url,true);

    if(method === "get"){
    if(routerMap.get[pathname]){
    //将query的值赋值给req.query
    req.query = query;
    routerMap.get[pathname](req,res)
    }else{
    res.end("404")
    }


    }else if(method === "post"){
    if(routerMap.post[pathname]){
    var str = "";

    //获取post传递的参数
    req.on("data",(data)=>{
    str += data;
    })

    req.on("end",()=>{
    req.body = qs.parse(str);
    routerMap.post[pathname](req,res)
    })

    }
    }
}

//注册事件
router.get = function(path,callback){
    routerMap.get[path] = callback;
}
//注册事件
```

```
router.post = function(path,callback){
 routerMap.post[path] = callback;
}


//处理所有的静态资源访问
function handleStatic(req,res){
 var {pathname} = url.parse(req.url,true);
 //获取文件的后缀
 var ext = pathname.substring(pathname.lastIndexOf("."));

 if(pathname ==="/"){
 res.writeHead(200,{"content-type":"text/html;charset=utf8"});
 res.end(getFile(path.join(__dirname,"../public/index.html")))
 }else if(ext === ".css"){
 res.writeHead(200,{"content-type":"text/css;charset=utf8"});
 res.end(getFile(path.join(__dirname,"../public",pathname)));
 }else if(ext === ".js"){
 res.writeHead(200,{"content-type":"application/x-javascript;charset=utf8"});
 res.end(getFile(path.join(__dirname,"../public",pathname)));

 }else if(/.*\.(jpg|png|gif)/.test(ext)){
 res.writeHead(200,{"content-type":`image/${RegExp.$1};charset=utf8`});
 res.end(getFile(path.join(__dirname,"../public",pathname)));
 }else if(ext === ".html"){
 res.writeHead(200,{"content-type":"text/html;charset=utf8"});
 res.end(getFile(path.join(__dirname,"../public",pathname)));
 }
}


function getFile(filePath){
 return fs.readFileSync(filePath);
}

module.exports = router;
```

四、express的基本入门

1、什么是express？

> *Express 是一个保持最小规模的灵活的 Node.js Web 应用程序开发框架，为 Web 和移动应用程序提供一组强大的功能*

2、什么是中间件

> *请求和回复之间的一个应用*

3、常见的中间件有哪些？

> *1、应用层中间件(http请求之类的应用)*

```
const express = require("express");
const app = express();

//应用层的中间件
app.use((req,res,next)=>{
console.log(123);
next()
})

app.use((req,res,next)=>{
console.log(456)
next()
})

app.use((req,res,next)=>{
console.log(789);
next();
})

app.use("/home",(req,res,next)=>{
res.end("home");
})

app.get("/list",(req,res,next)=>{
res.end("list")
})

app.post("/list",(req,res,next)=>{
res.end("list")
})
```

```
31.
32.
33.  app.listen(9000,()=>{
34.  console.log("server address:127.0.0.1:9000")
35.  })
36.
37.  /*
38.  express中的中间件会分为哪几类?
39.  1、应用层中间件(http请求之类的应用)
40.  app.use app.get app.post
41.
42.
43.  2、内置中间件
44.  3、错误处理中间件
45.  4、路由中间件
46.  5、第三方中间件
47.  6、自定义中间件
48.
49.
50.  如何使用中间件?
51.  app.use(中间件) 使用中间件
52.
53.  app.use中的参数
54.  path:选填
55.  callback || router
56.
57.  如果是callback的请求下 这个回调中会有3个参数
58.  req:请求
59.  res:响应
60.  next:执行下一个中间件
61.
62.
63.  只要http这个服务跑一次那么久会将app.use中的所有函数统一执行一遍
64.  */
```

2、内置中间件

```
1.  const express = require("express");
2.  const app = express();
3.  const path = require("path");
4.  //内置中间件
5.  app.use(express.static(path.join(__dirname,"./public")))
```

```
6.
7.
8.  app.listen(9000,()=>{
9.  console.log("server address:127.0.0.1:9000")
10. })
```

3、错误处理中间件

```
1.  const express = require("express");
2.  const app = express();
3.  const path = require("path");
4.  //内置中间件
5.  app.use(express.static(path.join(__dirname,"./public")))
6.
7.
8.
9.  //错误中间件处理一般写在程序的最后面
10. app.use((req,res,next)=>{
11. res.status(404).send("没有这个页面")
12. })
13.
14. app.listen(9000,()=>{
15. console.log("server address:127.0.0.1:9000")
16. })
```

4、路由中间件

```
1.  const express = require("express");
2.  const app = express();
3.  const path = require("path");
4.  const indexRouter = require("./router/index");
5.
6.  //内置中间件
7.  app.use(express.static(path.join(__dirname,"./public")))
8.
9.  //路由级别的中间件
10. app.use("/user",indexRouter)
11.
12. app.listen(9000,()=>{
13. console.log("server address:127.0.0.1:9000")
14. })
```

5、第三方中间件

例如 cheerio jsonwebToken等

> 例如 cheerio jsonwebtoken等

6、自定义中间件

> 自己根据需求封装的中间件

day04

一、手动封装body-parser中间件 cookie-parser中间件

1、封装axios请求方式

body-parser

```
1.
2.  const qs = require("querystring");
3.  var bodyparse = (function bodyparse() {
4.
5.    function common(type) {
6.    return (req, res, next) => {
7.    let contentType = req.headers["content-type"];
8.
9.
10.   if(contentType == "application/json" || contentType == "application/x-www-form-
    urlencoded"){
11.   let str = "";
12.
13.   req.on("data",(data)=>{
14.   str+=data;
15.   })
16.
17.   req.on("end",()=>{
18.
19.   if(contentType == "application/json"){
20.   req.body = JSON.parse(str);
21.   next();
22.   }else if( contentType == "application/x-www-form-urlencoded"){
23.   req.body = qs.parse(str);
24.   next();
25.   }
```

```
  next();
  })

  }else{
  next();
  }


  }
  }

  //解析appliaction/json
  function json() {
  let type = "json"
  return common(type)
  }

  //解析appliaction/x-www-form-urlencoded
  function urlencoded() {
  let type = "urlencoded";
  return common(type)
  }

  return {
  json,
  urlencoded
  }
 })()

 module.exports = bodyparse;
```

cookie-parser

```
 const cookieparse = function(){
```

```
  return (req,res,next)=>{
  let cookies = req.headers.cookie;

  let obj = cookies.split("; ").reduce((prev,curr)=>{
  var key = curr.split("=")[0];
  var val = curr.split("=")[1];
  prev[key] = val;
  return prev;
  },{})

  req.headers.cookie = obj;

  next();

  }
 }

module.exports = cookieparse;
```

二、express原理解析

1、app.use的作用以及实现

```
const http = require("http")
const url = require("url");
const path = require("path");
const fs = require("fs");
const express = () => {

  //将所有注册的事件添加到routesMap中去
  var routesMap = {
  get: {

  },
  post: {

  }
  }

  //app这个函数中做的执行
```

```
var app = function (req, res) {
//获取用户请求的路径
var pathname = url.parse(req.url).pathname;
//获取用户请求的方式
var method = req.method.toLowerCase();

if (pathname !== "/favicon.ico") {
//next函数用来执行下一个中间件
var next = function () {
var handle;
//执行是没有路径的函数
while (handle = routesMap.all.shift()) {
handle(req, res, next)
}
}

next();

//执行app.use带路径的函数
for (var key in routesMap) {
//判断路径是否与用户请求的路径相同
if (key === pathname) {
routesMap[key](req, res, next);
break;
}
}

//判断用户是否是get请求
if (method == "get") {
//如果是get请求则找到get请求相对应得路径 执行对应的函数
for (var key in routesMap.get) {
if (key == pathname) {
routesMap.get[key](req, res, next);
break;
}
}


} else if (method == "post") {

}
```

```
}

}

}


//注册事件
app.use = function () {
//判断use中的第一个参数是路径还是一个函数
if (typeof arguments[0] === "string") {
routesMap[arguments[0]] = arguments[1]
} else if (typeof arguments[0] === "function") {
//建议all这个key值写成Symbol
if (!routesMap.all) {
routesMap.all = [];
}
routesMap.all.push(arguments[0]);
}
}
//注册事件
app.get = function (path, callback) {
routesMap.get[path] = callback;
}


//创建服务
app.listen = function (port, callback) {
http.createServer(app).listen(port, callback)
}
return app;
}

//静态资源访问
express.static = function (staticPath) {

function getFile(filePath) {
return fs.readFileSync(filePath);
}

```

```
99.
100.  return (req, res) => {
101.  var { pathname } = url.parse(req.url, true);
102.  //获取文件的后缀
103.  var ext = pathname.substring(pathname.lastIndexOf("."));
104.
105.  if (pathname === "/") {
106.  res.writeHead(200, { "content-type": "text/html;charset=utf8" });
107.  res.end(getFile(path.join(staticPath, "index.html")))
108.  } else if (ext === ".css") {
109.  res.writeHead(200, { "content-type": "text/css;charset=utf8" });
110.  res.end(getFile(path.join(staticPath, pathname)));
111.  } else if (ext === ".js") {
112.  res.writeHead(200, { "content-type": "application/x-javascript;charset=utf8" });
113.  res.end(getFile(path.join(staticPath, pathname)));
114.
115.  } else if (/.*\.(jpg|png|gif)/.test(ext)) {
116.  res.writeHead(200, { "content-type": `image/${RegExp.$1};charset=utf8` });
117.  res.end(getFile(path.join(staticPath, pathname)));
118.  } else if (ext === ".html") {
119.  res.writeHead(200, { "content-type": "text/html;charset=utf8" });
120.  res.end(getFile(path.join(staticPath, pathname)));
121.  }
122.  }
123.  }
124.
125.
126.  module.exports = express;
```

三、express-generator生成器以及代码解析

1、app.js解析

2、www文件解析

3、ejs的基本使用

> *用<% … %> 开始符和结束符之间写js代码*
>
> *用<%= … %>输出变量数据 变量若包含 ‘<’ ‘>’ ‘&’等字符 会被转义*
>
> *用<%- … %>输出变量（html的解析前面需要加-） 不转义*

> *用<%- include(‘user/show’) %>引入其他模板 包含 ./user/show.ejs*
>
> *用<%# some comments %>来注释，不执行不输出*
>
> *<%% 转义为 ‘<%’<% … -%> 删除新的空白行模式?*
>
> *<% … %> 删除空白符模式开始符和结束符内部不可以嵌套*

四、前端渲染和后端渲染的区别

> *后端渲染HTML的情况下，浏览器会直接接收到经过服务器计算之后的呈现给用户的最终的HTML字符串，这里的计算就是服务器经过解析存放在服务器端的模板文件来完成的，在这种情况下，浏览器只进行了HTML的解析，以及通过操作系统提供的操纵显示器显示内容的系统调用在显示器上把HTML所代表的图像显示给用户。*
> *前端渲染就是指浏览器会从后端得到一些信息，这些信息或许是适用于题主所说的angularjs的模板文件，亦或是JSON等各种数据交换格式所包装的数据，甚至是直接的合法的HTML字符串。这些形式都不重要，重要的是，将这些信息组织排列形成最终可读的HTML字符串是由浏览器来完成的，在形成了HTML字符串之后，再进行显示*

五、用所学的express生成器+ejs做一个简单的列表页

六 、socket的基本使用

1、什么是socket?

> *网络上两个程序通过一个双向的通信连接实现数据交换，这个连接的一端称为socket*

2、http请求与socket的区别

> *1、在以前我们实现数据交换已经有了HTTP协议,为什么还要学习socket？*
> *回顾：当输出www.baidu.com的时候浏览器执行了那些操作？*
>
> *2、http通信的特点： 1、连接属于非持久性连接:TCP的三次握手*
> *2、客户端只能访问服务端，服务端无法访问客户端，属于单项通信*
>
> *TCP三次握手： TCP三次握手过程中不传递数据,只为同步连接双方的序列号和确认号传递数据，在握手后服务端和客户端才开始传输数据，在理想状态下，TCP连接一旦建立，在通信的双方中任何一方主动断开连接之前TCP连接会一直保持下去。*
>
> *socket通信特点：1、持久性连接*
> *2、双向通信，客户端能访问服务端，服务端也能访问客户端*

> 2、双向通信，客户端能访问服务端，服务端也能访问客户端
>
> socket是对TCP/IP协议的封装，socket只是一个接口而不是一个协议，通过Socket我们才能使用TCP/IP/UDP协议

3、通过node内置模块net实现简单版聊天

> 1、socket连接需要由2个节点：（1）clientSocket （2）serverSocket
>
> 2、文本流：readline

```
const net = require("net");

const server = net.createServer();

const clients = [];
//当有人连接服务端的时候回触发当前函数
server.on("connection",(client)=>{
 //给每一个用户添加一个唯一的标识
 client.id = clients.length;
 clients.push(client);

 //将客户端的消息转发给所有的用户
 client.on("data",(data)=>{
 clients.forEach(item=>{
 //判断用户是否存在
 if(item){
 item.write("服务端:"+data);
 }
 })
 })

 //判断用户是否退出群聊
 client.on("close",()=>{
 clients[client.id] = null;
 })

})

server.listen(9000);
```

```
const net = require("net");
```

```
const client = new net.Socket();
//创建I/o模型 创建出入和输出流
const readline = require("readline");

//创建
const rl = readline.Interface({
 input: process.stdin,
 output: process.stdout
})

//客户端连接服务端
client.connect(9000, () => {
 client.on("data", (data) => {
 console.log(data.toString());
 })
})

//获取终端的内容将内容转发到服务端
rl.on("line", (val) => {
 client.write(val);
})
```

七、webSocket的基本使用

```
const WebSocket = require('ws');
//创建服务
const server = new WebSocket.Server({ port: 9000 });

//监控用户连接
server.on('connection', (client,req)=>{
 var ip = req.connection.remoteAddress;


 //监听客户端传递到服务端的消息
 client.on('message', (data)=>{
 //将消息转发给所有的用户
 server.clients.forEach((item)=>{
 //判断用户是否是连接的状态
 if (item.readyState === WebSocket.OPEN) {
 item.send(ip+":"+data);
```

```
  }
  });
  });
});
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <input type="text" id="txt">
  <button id="btn">点击</button>
  <ul id="list"></ul>
</body>
</html>
<script>
  var client = new WebSocket("ws://10.60.15.150:9000");
  var txt = document.getElementById("txt");
  var list = document.getElementById("list");
  var btn = document.getElementById("btn");


  //接受服务端消息 onmessage
  client.onmessage = function(e){
  var li = document.createElement("li");
  li.innerText = e.data;
  list.appendChild(li);
  li.scrollIntoView();
  }

  //向服务端发送消息 send
  btn.onclick = function(){
  var val = txt.value;
  client.send(val);
  txt.value = "";
  }

```

```
</script>
```

八、socket.io的基本使用

```
var express = require('express')
var app = express();
//文件在服务器运行
var http = require('http')
var server = http.createServer(app);
var path = require("path")
//因为我们要做持久性通信 因此不能够用http连接 http连接与socket.io进相关联
var io = require("socket.io")(server);
app.use(express.static(path.join(__dirname,"./public")))


//当用户连接的时候触发的函数
io.on('connection', (socket)=>{
 console.log('a user connected');

 //接受客户端消息 将消息转发给所有用户
 socket.on("sendMessage",(mes)=>{
 io.emit("message",mes)
 })

 //当用户断开连接的时候
 socket.on('disconnect', function(){
 console.log('user disconnected');
 });
});
server.listen(3000, function(){
 console.log('listening on *:3000');
});
```

九、利用websocket实现一个聊天室

> *1、多人聊天*
>
> *2、图片发送*
>
> *3、表情发送*

```
var express = require('express')
var app = express();
//文件在服务器运行
var http = require('http')
var server = http.createServer(app);
var path = require("path")
//因为我们要做持久性通信 因此不能够用http连接 http连接与socket.io进相关联
var io = require("socket.io")(server);
app.use(express.static(path.join(__dirname,"./public")))

var users = [];

//当用户连接的时候触发的函数
io.on('connection', (socket)=>{
 //监听用户连接
 socket.on("system",(username,type)=>{
 if(users.includes(username)){
 socket.emit("login",0)
 }else{
 socket.userIndex = users.length;
 socket.username = username;
 users.push(username);
 io.emit("login",1,username)
 }
 })

 socket.emit("userList",users);

 socket.on("ClientsendMessage",(usename,message)=>{
 socket.broadcast.emit("serverSendMessage",usename,message)
 })

 socket.on("sendImg",(username,base64Img)=>{
 socket.broadcast.emit("serverSendImg",usename,base64Img)
 })

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

```
<!doctype html>
<html>

<head>
 <title>Socket.IO chat</title>
 <link rel="stylesheet" href="./css/iconfont/iconfont.css">
 <style>
 * {
 margin: 0;
 padding: 0;
 box-sizing: border-box;
 }
 ul,li{
 list-style: none;
 }
 html,
 body {
 font: 13px Helvetica, Arial;
 height: 100%;
 }

 form {
 background: #000;
 padding: 3px;
 position: fixed;
 bottom: 0;
 width: 100%;
 }

 form input {
 border: 0;
 padding: 10px;
 width: 90%;
 margin-right: .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;
}

#action {
width: 100%;
height: 30px;
display: flex;
align-items: center;
}

#action input[type='color'] {
width: 40px;
height: 30px;
}

#upload,
#fontColor {
width: 40px;
height: 30px;
position: relative;
}

#action input[type='file'],
#fontColor input[type="color"] {
width: 40px;
height: 30px;
position: absolute;
left: 0%;
```

```
    top: 0;
    opacity: 0;
    z-index: 5;
    }

    #action i,
    #fontColor i {
    width: 100%;
    height: 100%;
    position: absolute;
    left: 0;
    top: 0;
    color: #fff;
    font-size: 20px;
    }

    #mask {
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, .3);
    position: absolute;
    z-index: 10;
    }

    #content{
    width: 100%;
    height: 100%;
    display: flex;
    justify-content:space-between;
    }

    #content ul:nth-child(2){
    width:200px;
    height: 100%;
    border-left:1px solid #ccc;
    }
    #userList {
    overflow: scroll;
    }
    #userList li{
```

```
    line-height: 30px;
    border-bottom: 1px solid #bbb;
    width:100%;
    }
    .userDate{
    color: green;
    line-height: 20px;
    font-size:18px;
    }
    .userInfo{
    color: #000;
    line-height: 20px;
    font-size:14px;
    }
    #messages>div{
    min-height: 60px;
    }
    #system{
    color: #c33;
    font-size:18px;
    }
    </style>
</head>

<body>
    <div id="mask"></div>
    <div id="content">
    <ul id="messages"></ul>
    <ul id="userList">
    <li>用户列表</li>
    </ul>
    </div>
    <form id="form">
    <div id="action">
    <div id="fontColor">
    <input type="color">
    <i class="iconfont">&#xec85;</i>
    </div>
    <div id="upload">
    <input type="file" id="file">
    <i class="iconfont">&#xe674;</i>
```

```
  </div>
  <ul></ul>
  </div>
  <input id="m" autocomplete="off" />
  <input type="submit" value="提交">
  </form>
</body>

</html>
<script src="/socket.io/socket.io.js"></script>
<script src="https://code.jquery.com/jquery-1.11.1.js"></script>
<script src="./js/index.js"></script>
```

index.js

```
class SocketAlley {
 constructor() {
 this.mask = $("#mask");
 this.userListEl = $("#userList");
 this.messages = $("#messages");
 this.socket = io();
 this.init();

 }
 init() {
 if (!sessionStorage.getItem("status")) {
 this.username = window.prompt("请输入您的姓名");
 if (this.username) {
 sessionStorage.setItem("status", this.username);
 //当用户连接进来的时候通知服务器
 this.socket.emit("system", this.username, "login")
 //检测是否连接成功
 this.socket.on("login", (data, username) => {
 if (data == 1) {
 alert("连接成功");
 this.mask.hide();

 //全局通知
 var div = $("<div class='system'></div>");
 var str = username + "进入聊天室"
 div.text(str);
 this.messages.append(div);
```

```
28.  } else {
29.  alert("用户名重复请求重新编写");
30.  }
31.  })
32.  }
33.  } else {
34.  this.mask.hide();
35.  this.username = sessionStorage.getItem("status");
36.
37.  this.socket.on("login", (data, username) => {
38.  //全局通知
39.  var div = $("<div class='system'></div>");
40.  var str = username + "进入聊天室"
41.  div.text(str);
42.  this.messages.append(div);
43.
44.  })
45.  }
46.
47.  this.userList();
48.  this.serverMessage();
49.  this.userSendInfo();
50.  this.sendImg();
51.  }
52.  userList() {
53.  this.socket.on("userList", this.handleUserListSucc.bind(this))
54.  }
55.  handleUserListSucc(data) {
56.  data.forEach(this.handleUserListEach.bind(this))
57.  }
58.  handleUserListEach(item) {
59.  var li = $("<li></li>");
60.  li.text(item);
61.  this.userListEl.append(li);
62.  }
63.  userSendInfo() {
64.  $("#form").on("submit", this.handleUserSendInfo.bind(this))
65.  }
66.  handleUserSendInfo(e) {
67.  e.preventDefault();
68.  var val = $("#m").val();
```

```
69.
70.
71.   this.infoStyle(this.username, val);
72.   //向服务端发送消息
73.   this.socket.emit("ClientsendMessage", this.username, val);
74.   }
75.   serverMessage() {
76.   this.socket.on("serverSendMessage", (username, message) => {
77.   this.infoStyle(username, message);
78.   })
79.
80.
81.   this.socket.on("serverSendImg", (username, message) => {
82.   this.infoImg(username, message);
83.   })
84.   }
85.   infoImg(username, message) {
86.   var parentDiv = $("<div></div>");
87.   var childDiv = $("<div class='userDate'></div>");
88.   var contentDiv = $("<div class='userInfo'></div>");
89.   var d = new Date();
90.   if (/(\d{2}:\d{2}:\d{2})/.test(d)) {
91.
92.   childDiv.text(username + RegExp.$1);
93.   var img = $("<img/>");
94.   img.attr("src",message);
95.   contentDiv.append(img);
96.
97.   parentDiv.append(childDiv);
98.   parentDiv.append(contentDiv)
99.   this.messages.append(parentDiv);
100.  parentDiv[0].scrollIntoView();
101.  }
102.  }
103.  infoStyle(username, message) {
104.
105.  var parentDiv = $("<div></div>");
106.  var childDiv = $("<div class='userDate'></div>");
107.  var contentDiv = $("<div class='userInfo'></div>");
108.  var d = new Date();
109.  if (/(\d{2}:\d{2}:\d{2})/.test(d)) {
```

```
110.
111.   childDiv.text(username + RegExp.$1);
112.   contentDiv.text(message);
113.
114.   parentDiv.append(childDiv);
115.   parentDiv.append(contentDiv)
116.   this.messages.append(parentDiv);
117.   parentDiv[0].scrollIntoView();
118.   }
119.   }
120.   sendImg() {
121.   $("#file").on("change", this.sendImgCb.bind(this))
122.   }
123.   sendImgCb() {
124.   var _this = this;
125.   //只能从原生JS中拿到file对象
126.   var file = $("#file")[0].files[0];
127.   //将file对象转换dataurl(base64的文件形式)
128.   var fileReader = new FileReader()
129.
130.   fileReader.onload = function (e) {
131.   _this.socket.emit("sendImg", _this.username, e.target.result);
132.   _this.infoImg(_this.username, e.target.result)
133.   }
134.
135.   //将file转换成dataUrl的形式
136.   fileReader.readAsDataURL(file);
137.   }
138. }
139.
140.
141. new SocketAlley();
```

十、利用socket实现一个简单版的多人点餐功能

```
1. const express = require("express");
2. const app = express();
3. const http = require("http");
4. const server = http.createServer(app);
5. const io = require("socket.io")(server);
```

```
const path = require("path");
app.use(express.static(path.join(__dirname, "./public")))



io.on("connection", (socket) => {
 socket.on('add', (data) => {
 socket.broadcast.emit("serverAdd",data);
 })

 socket.on('reducer', (data) => {
 socket.broadcast.emit("serverReducer",data);
 })
})



server.listen(9000)
```

```
<!DOCTYPE html>
<html lang="en">

<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <meta http-equiv="X-UA-Compatible" content="ie=edge">
 <title>Document</title>
 <link rel="stylesheet" href="./css/reset.css">
 <style>
 #goodsList {
 width: 100%;
 height: 100%;
 padding: .1rem;
 }

 .goodsItem {
 display: flex;
 width: 100%;
 padding: .2rem;
 }

```

```
.goodsItem>div:nth-child(1) {
margin-right: .2rem;
}

.goodsItem img {
width: 2rem;
height: 2.5rem;
}

.goodsDes {
flex: 1;
display: flex;
justify-content: space-between;
flex-direction: column;
}

.goodsDes>div:nth-child(3) {
display: flex;

}

.goodsDes>div:nth-child(3) .reducer {
width: 30px;
height: 30px;
text-align: center;
line-height: 30px;
background: #ccc;
color: #fff;
font-size: 16px;
}

.goodsDes>div:nth-child(3) .add {
width: 30px;
height: 30px;
text-align: center;
line-height: 30px;
background: #ccc;
color: #fff;
font-size: 16px;
}
```

```
.goodsDes>div:nth-child(3) input {
width: 80px;
height: 30px;
border: 0;

}
</style>
</head>

<body>
<!--
1、需求评审

2、(需求肯定出来了)前后端开会->定义接口(后端为主 前端为辅) 你写代码的速度 * 需求的难易程度 * 1.5 = 模块开发的时间 cnpm install json-server -g json-server中的增删改查 查：GET 增：POST 删：delete
改：patch -->
<div id="goodsList">

</div>
</body>

</html>
<script src="/socket.io/socket.io.js"></script>
<script src="https://cdn.bootcss.com/jquery/1.11.3/jquery.min.js"></script>
<script>
class Goods {
constructor() {
this.socket = io();
}
init() {
this.getGoods();
this.watchServer();

}
getGoods() {
$.ajax({
type: "get",
url: "http://localhost:3000/data",
success: this.handleGetGoodsSucc.bind(this)
```

```
102.  })
103.  }
104.  handleGetGoodsSucc(data) {
105.  var str = "";
106.  for (var i = 0; i < data.length; i++) {
107.  str += `<div class="goodsItem" data-id="${data[i].id}">
108.  <div>
109.  <img src="${data[i].goodsPic}" >
110.  </div>
111.  <div class="goodsDes">
112.  <div>${data[i].goodsName}</div>
113.  <div>单价:<span data-price="${data[i].price}">${data[i].price}</span>$</div>
114.  <div>
115.  <div class="reducer">-</div>
116.  <input type="text" value="${data[i].num}">
117.  <div class="add">+</div>
118.  </div>
119.  </div>
120.  </div>
121.  `
122.  }
123.
124.
125.  $("#goodsList").html(str);
126.  this.add();
127.  this.reducer();
128.  }
129.
130.
131.  add() {
132.  $(".add").each(this.handleAddEach.bind(this))
133.  }
134.  handleAddEach(index) {
135.  $(".add").eq(index).on("click", this.handleAddCb.bind(this, index))
136.  }
137.  handleAddCb(index) {
138.  var n = $(".add").eq(index).prev().attr("value");
139.  var id = $(".add").eq(index).parent().parent().parent().attr("data-id");
140.  n++;
141.  $(".add").eq(index).prev().attr("value", n);
142.
```

```
143.   var price =
$(".add").eq(index).parent().parent().find("div").eq(1).find("span").attr("data-price");
144.   $(".add").eq(index).parent().parent().find("div").eq(1).find("span").text(price * n)
145.
146.
147.
148.   //socket部分
149.   this.socket.emit("add", { id: id, num: n });
150.
151.
152.
153.   }
154.
155.   reducer() {
156.   $(".reducer").each(this.handleReducerEach.bind(this))
157.   }
158.   handleReducerEach(index) {
159.   $(".reducer").eq(index).on("click", this.handleReducerCb.bind(this, index))
160.   }
161.   handleReducerCb(index) {
162.   var n = $(".reducer").eq(index).next().attr("value");
163.   var id = $(".reducer").eq(index).parent().parent().parent().attr("data-id");
164.   if (n == 1) {
165.   n = 1;
166.   } else {
167.   --n;
168.   this.socket.emit("reducer", { id: id, num: n });
169.   }
170.
171.
172.
173.   $(".reducer").eq(index).next().attr("value", n);
174.   var price =
$(".reducer").eq(index).parent().parent().find("div").eq(1).find("span").attr("data-
price");
175.   $(".reducer").eq(index).parent().parent().find("div").eq(1).find("span").text(price *
n)
176.
177.   }
178.   watchServer() {
179.   this.socket.on("serverAdd", this.handleServerAddSucc.bind(this));
```

```
180.  this.socket.on("serverReducer", this.handleServerReducerSucc.bind(this))
181.  }
182.  handleServerAddSucc(data) {
183.
184.  $(".add").each(this.handleAddEachServer.bind(this, data))
185.  }
186.  handleAddEachServer(data, index) {
187.  var id = $(".add").eq(index).parent().parent().parent().attr("data-id");
188.  if (id == data.id) {
189.  var val = $(".add").eq(index).prev().val();
190.  $(".add").eq(index).prev().val(Number(data.num));
191.  }
192.  }
193.  handleServerReducerSucc(data) {
194.  $(".reducer").each(this.handleReducerEachServer.bind(this, data))
195.  }
196.
197.  handleReducerEachServer(data, index) {
198.  var id = $(".reducer").eq(index).parent().parent().parent().attr("data-id");
199.  if (id == data.id) {
200.  var val = $(".reducer").eq(index).next().val();
201.  $(".reducer").eq(index).next().val(Number(data.num));
202.  }
203.
204.  }
205.
206.  }
207.
208.  new Goods().init();
209.
210.
211.  </script>
```

day05、eventLoop详解

一、前端EventLoop

1、什么是eventLoop?

> *同步任务和异步任务在js中是如何执行的呢？js的代码运行会形成一个主线程和一个任务队列。主线程会从上到下一步步执行我们的js代码，形成一个执行栈。同步任务就会被放到这个执行栈中依次执行。而异步任务被放入到任务队列中执行，执行完就会在任务队列中打一个标记，形成一个对应的事件。当执行栈中的任务全部运行完毕，js会去提取并执行任务队列中的事件。这个过程是循环进行的，这就是我们今天想要了解的event loop*

2、为什么js是单线程

> *想要了解event loop我们就要从js的工作原理说起。首先，大家都知道js是单线程的。所谓单线程就是进程中只有一个线程在运行。那么，js为什么是单线程而不是做成多线程的呢？个人理解，js是用来实现浏览器与用户之间的交互的。如果同时要处理用户点击，用户输入，用户关闭等操作，浏览器无法知道这个时间我到底应该做什么。所以js是从上至下按顺序运行下去的*

3、什么是宏任务&&什么是微任务

> *宏任务： 需要多次事件循环才能执行完，事件队列中的每一个事件都是一个宏任务。浏览器为了能够使得js内部宏任务与DOM任务有序的执行，会在一个宏任务执行结束后，在下一个宏执行开始前，对页面进行重新渲染 （task->渲染->task->…）鼠标点击会触发一个事件回调，需要执行一个宏任务，然后解析HTML*
>
> *微任务： 微任务是一次性执行完的。微任务通常来说是需要在当前task执行结束后立即执行的任务，例如对一些动作做出反馈或者异步执行任务又不需要分配一个新的task，这样便可以提高一些性能*

4、案例

```
console.log("script start");

setTimeout(function(){
 console.log("setTimeout");
},0)

newPromise(resolve=>{
 console.log("promise start");
 resolve();
}).then(function(){
 console.log("promise1");
}).then(()=>{
 console.log("promise2");
})

console.log("script end");
console.log(1);
```

```
console.log(1);

setTimeout(()=>{
 console.log(2);
})

newPromise((resolve)=>{
 console.log(4)
 resolve()
}).then(()=>{
setTimeout(()=>{
 console.log(5);
 })
}).then(()=>{
 console.log(6)
})

console.log(7)
setTimeout(() => {
 console.log(5)
• new Promise(resolve => {
 console.log(6)
• setTimeout(() => {
 console.log(7)
 })
 resolve()
 }).then(() => {
 console.log(8)
 })
}, 500)
new Promise(resolve => {
 console.log(9)
 resolve()
}).then(() => {
 console.log(10)
• setTimeout(() => {
 console.log(11)
 }, 0)
})
•console.log(12)
```

二、后端EventLoop

1、NodeJS中的宏任务分类

> *Timers 类型的宏任务队列*
>
> - *setTimeout()*
> - *setInterval*
>
> *Check 类型的宏任务队列*
>
> - *setImmediate()*
>
> *Close callback 类型的宏任务队列*
>
> - *socket.on('close', () => {})*
>
> *Poll 类型的宏任务队列*
>
> - *除了上面几种的其他所有回调*

2、nodeJs 里面的微任务队列

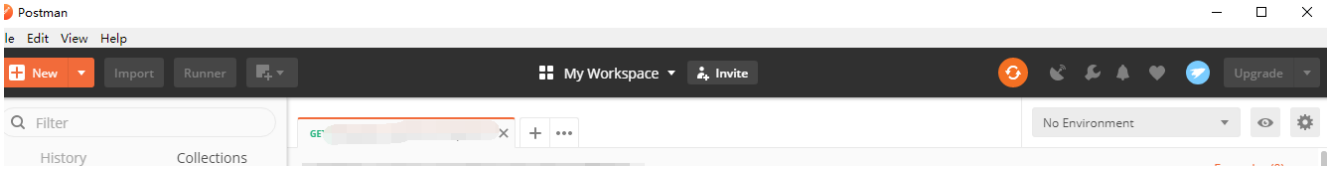
> *process.nextTick()* `Promise.then() process.nextTick()` *的优先级高于所有的微任务,每一次清空微任务列表的时候，都是先执行 `process.nextTick()*

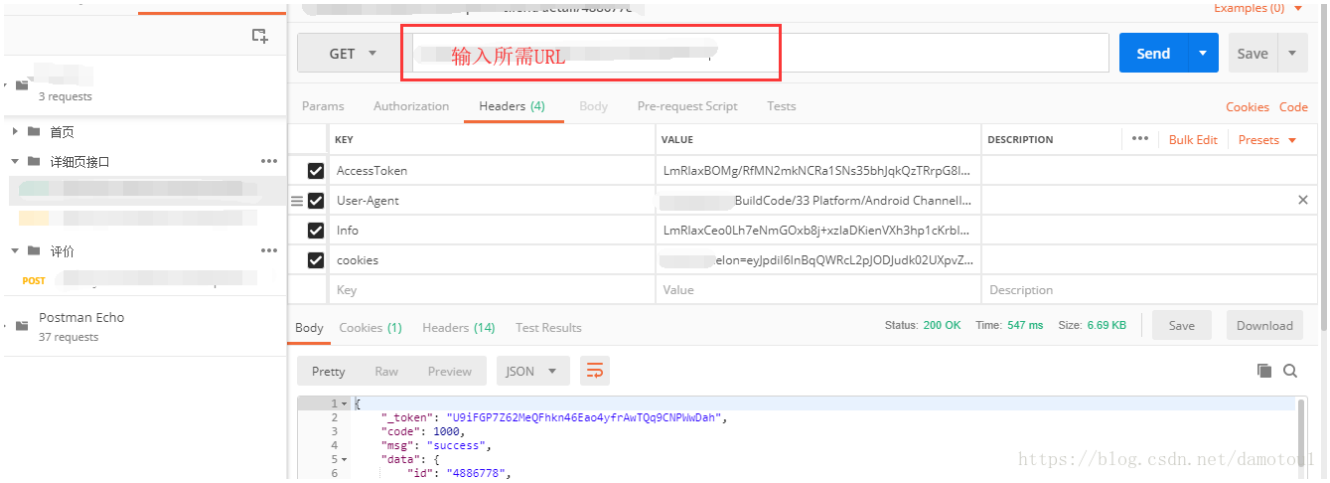
3、setTimeout && setImmediate执行顺序

> *Node 并不能保证 timers 在预设时间到了就会立即执行，因为 Node 对 timers 的过期检查不一定靠谱，它会受机器上其它运行程序影响，或者那个时间点主线程不空闲*
>
> *虽然 setTimeout 延时为 0，但是一般情况 Node 把 0 会设置为 1ms，所以，当 Node 准备 event loop 的时间大于 1ms 时，进入 timers 阶段时，setTimeout 已经到期，则会先执行 setTimeout；反之，若进入 timers 阶段用时小于 1ms，setTimeout 尚未到期，则会错过 timers 阶段，先进入 check 阶段，而先执行 setImmediate*

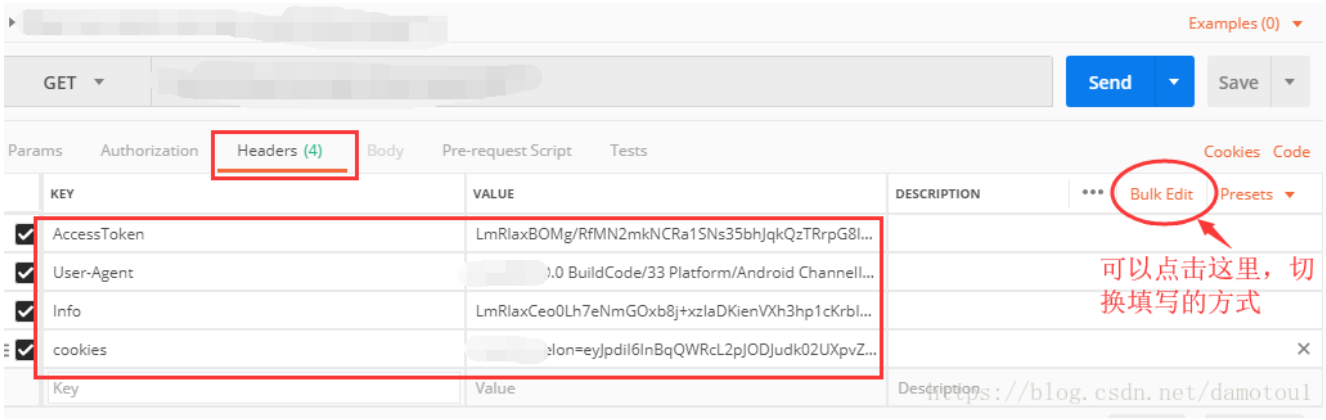
三、postman的基本使用

1、打开postman之后，首先输入URL，方法选择GET

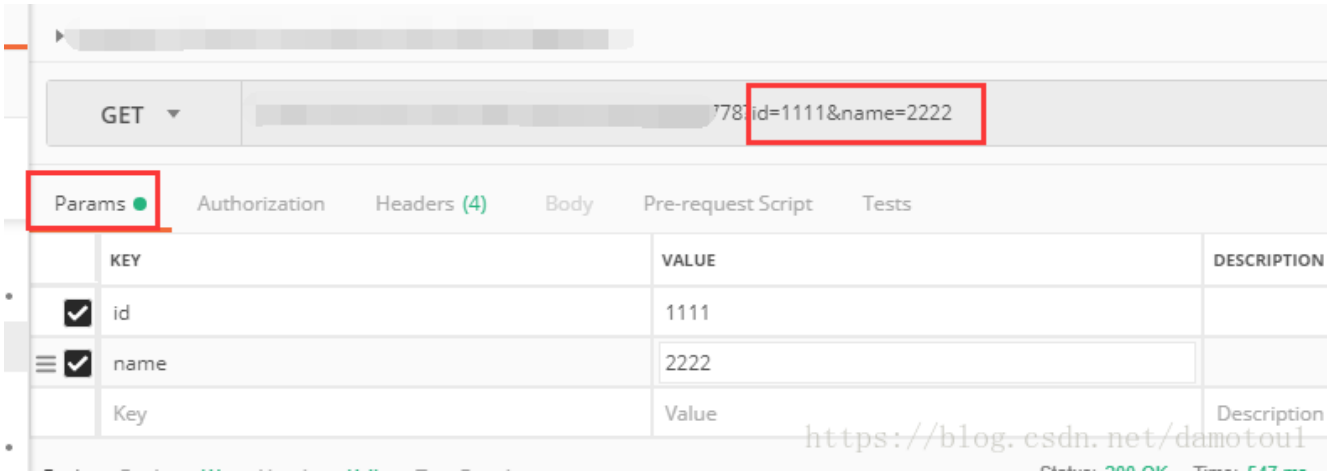




2、然后我们在Headers里面输入我们所需要的内容，如果需要Cookie也填写在这里面即可（不需要可以不填），填写的时候可以单个填写，也可以点击【Bulk Edit】进行填写方式切换，切换后可以一次填写所有内容

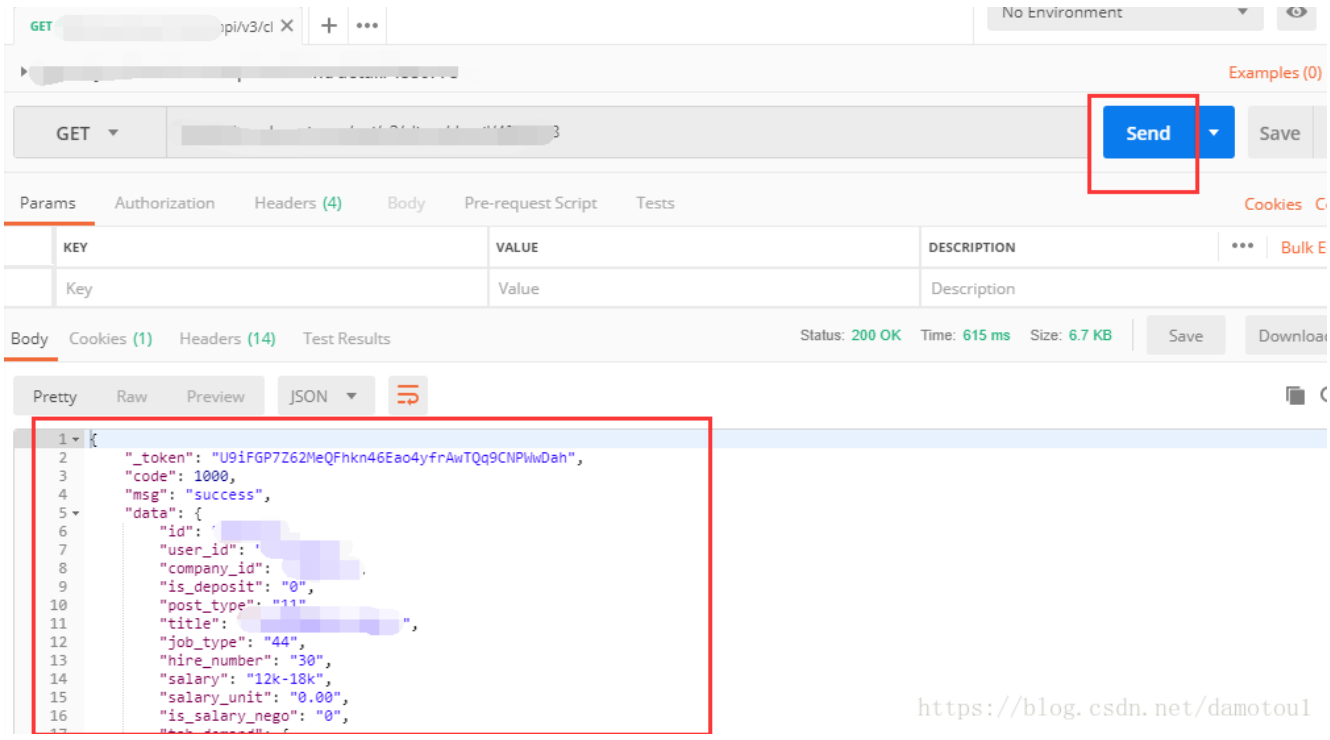


3、填写了Headers之后，如果这个get请求里面有传参，我们接下来可以填写参数，如果不需要也可以不填。填了参数之后，内容就会拼接在我们的url里面



4、最后，我们点击send，就可以看到返回值了

4、最后，我们点击send，就可以看到返回值了



post同理

四、项目的基本搭建

一、express生成器搭建项目结构

1、安装 express-generator

```
npm install express-generator -g
```

2、通过express -e生成项目结构

```
```

3、安装依赖

```
1. $ cd myapp
2. $ npm install
```

4、启动项目

```
npm start
```

二、MVC架构思想

> *MVC即Model-View-Controller（模型-视图-控制器）是一种软件设计模式，最早出现在Smalltalk语言中，后被Sun公司推荐为Java EE平台的设计模式。*
>
> *MVC把应用程序分成了上面3个核心模块，这3个模块又可被称为业务层-视图层-控制层。顾名思义，它们三者在应用程序中的主要作用如下：*
>
> ***业务层**：负责实现应用程序的业务逻辑，封装有各种对数据的处理方法。它不关心它会如何被视图层显示或被控制器调用，它只接受数据并处理，然后返回一个结果。*
>
> ***视图层**：负责应用程序对用户的显示，它从用户那里获取输入数据并通过控制层传给业务层处理，然后再通过控制层获取业务层返回的结果并显示给用户。*
>
> ***控制层**：负责控制应用程序的流程，它接收从视图层传过来的数据，然后选择业务层中的某个业务来处理，接收业务层返回的结果并选择视图层中的某个视图来显示结果。*
>
> *可以用下图来表示MVC模式中三者之间的关系：*

三、图片上传

1. 安装multer模块

```
npm install multer
```

1. 引用模块
   它是依赖于express的一个模块

```
//引用express并配置
var express = require("express");
var app = express();
app.listen(3000);
var multer = require('multer');
```

```
/*var upload = multer({
  //如果用这种方法上传，要手动添加文明名后缀
  //如果用下面配置的代码，则可以省略这一句
  dest: 'uploads/'
})*/
```

1. 配置

   设置保存文件的地方，并根据上传的文件名对应文件添加后缀

   可以通过 `filename` 属性定制文件保存的格式

属性值	用途
`destination`	设置资源的保存路径。注意，如果没有这个配置项，默认会保存在 `/tmp/uploads` 下。此外，路径需要自己创建
`filename`	设置资源保存在本地的文件名

```
var storage = multer.diskStorage({
  //设置上传后文件路径，uploads文件夹会自动创建。
  destination: function(req, file, cb) {
  cb(null, './uploads')
  },
  //给上传文件重命名，获取添加后缀名
  filename: function(req, file, cb) {
  var fileFormat = (file.originalname).split(".");
  //给图片加上时间戳格式防止重名名
  //比如把 abc.jpg图片切割为数组[abc,jpg],然后用数组长度-1来获取后缀名
  cb(null, file.fieldname + '-' + Date.now() + "." + fileFormat[fileFormat.length - 1]);
  }
});
var upload = multer({
  storage: storage
});
```

1. 接受文件

   `upload.single('xxx')` ，xxx与表单中的name属性的值对应

   这里虽然用到post请求，但实际上不需要 `bodyParser` 模块处理

```
app.post('/upload-single', upload.single('logo'), function(req, res, next) {
  console.log(req.file)
  console.log('文件类型：%s', req.file.mimetype);
  console.log('原始文件名：%s', req.file.originalname);
  console.log((req.file.originalname).split("."))
```

```
6.  console.log('文件大小：%s', req.file.size);
7.  console.log('文件保存路径：%s', req.file.path);
8.  res.send({
9.  ret_code: '0'
10.  });
11.  });
```

1. 多图上传

多图上传只要更改一下地方，前端往file输入框加多一个 `multiple="multiple"` 属性值，此时就可以在选图的时候多选了，当然也可以并列多个file输入框(不推荐多个上传图片输入框)，这样体验会不好

```
1.  <input type="file" name="logo" multiple="multiple" />
```

后端也需要相应的改变

```
1.  app.post('/upload-single', upload.single('logo'), function(req, res, next) {
2.  //upload.single('logo')变为upload.array('logo', 2)，数字代表可以接受多少张图片
3.  app.post('/upload-single', upload.array('logo', 2), function(req, res, next) {
```

如果不想有图片数量上传限制，我们可以用 `upload.any()` 方法

```
1.  app.post('/upload-single', upload.any(), function(req, res, next) {
2.  res.append("Access-Control-Allow-Origin","*");
3.  res.send({
4.  wscats_code: '0'
5.  });
6.  });
```

1. 前端部分

- formData表单提交

```
1.  <form action="http://localhost:3000/upload-single" method="post"
    enctype="multipart/form-data">
2.  <h2>单图上传</h2>
3.  <input type="file" name="logo">
4.  <input type="submit" value="提交">
5.  </form>
```

- formData表单+ajax提交

```
1.  <form id="uploadForm">
2.  <p>指定文件名： <input type="text" name="filename" value="" /></p>
3.  <p>上传文件： <input type="file" name="logo" /></ p>
4.  <input type="button" value="上传" onclick="doUpload()" />
5.  </form>
```

`FormData` 对象，是可以使用一系列的键值对来模拟一个完整的表单，然后使用 `XMLHttpRequest` 发送这个"表单"

达这个`表单`

注意点

- processData设置为false。因为data值是FormData对象，不需要对数据做处理。
- `<form>`标签添加`enctype="multipart/form-data"`属性。
- cache设置为false，上传文件不需要缓存。
- contentType设置为false。因为是由`<form>`表单构造的FormData对象，且已经声明了属性`enctype="multipart/form-data"`，所以这里设置为false

上传后，服务器端代码需要使用从查询参数名为logo获取文件输入流对象，因为`<input>`中声明的是`name="logo"`

```
function doUpload() {
 $.ajax({
 url: 'http://localhost:3000/upload-single',
 type: 'POST',
 cache: false, //不必须
 data: new FormData($('#uploadForm')[0]),
 processData: false,//必须
 contentType: false,//必须
 success: function(data) {
 console.log(data)
 }
 })
 }
```

四、编写接口

> *api接口*
>
> - *RestfulApi 规范*
> - *接口文档的生成(apidoc)*
> - *接口请求方式区别*
>
> *跨域解决*
>
> - *cors*
> - *jsonp*
> - *proxy*

五、JWT

JWT

- 用户登录 服务器端产生一个token（加密字符串）发送给前端
- 前端将token 进行保存
- 前端发起数据请求的时候携带token
- 服务端 验证token 是否合法 如果合法继续操作 不合法终止操作
- token 的使用场景 无状态请求 保持用户的登录状态 第三方登录（token+auth2.0）

非对称加密 通过私钥产生token 通过公钥解密token

```
// 1.产生公钥和私钥
// 产生私钥 openssl genrsa -out ./private_key.pem 1024 1024 代表私钥长度
// 产生公钥 openssl rsa -in ./private_key.pem -pubout -out ./public_key.pem

 let private_key=fs.readFileSync(path.join(__dirname,'./private_key.pem'))
 let public_key=fs.readFileSync(path.join(__dirname,'./public_key.pem'))
 var token = jwt.sign(palyload, private_key,{ algorithm: 'RS256'});
 console.log(token)
 let token='eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IueUqOaIt2lkIiwiaWFOIjoxNTUxMTUyNzk1fQ.TI_xDBvObHGAH7EV4OWWpQemm5nx077Gdjq-pzDxONWN5YFd4OS7XcLmgoDdYscLM7vMOPOc7z1183JUixqk7IBjBCU-tMNo_G5_-LGkQjV3vDYq_3TkXT142lgmFA-EBey7W6W1PgPfYlowyHAyp-O7hXaMRevgVkXm2lPEFXo'

 var decoded = jwt.verify(token, public_key);
const jwt=require('jsonwebtoken')
const scrict='sdjfksdjflajflasjflasjflksf'

function creatToken(palyload){
 // 产生token
 palyload.ctime=Date.now()
 return jwt.sign(palyload,scrict)
}
function checkToken(token){
 return new Promise((resovle,reject)=>{
 jwt.verify(token,scrict,(err,data)=>{
 if(err){ reject('token 验证失败')}
 resovle(data)
 })
 })

```

```
}
module.exports={
  creatToken,checkToken
}
```

Cookie+Session

```
const cookieParse=require('cookie-parser')
const session = require('express-session')

app.use(session({
  secret: 'hubwizApp', //为了安全性的考虑设置secret属性
  cookie: {maxAge: 60 * 1000 * 60 * 24 }, //设置过期时间
  resave: true, // 即使 session 没有被修改，也保存 session 值，默认为 true
  saveUninitialized: false, //无论有没有session cookie，每次请求都设置个session cookie ，默认给个标示为 connect.sid
}));
```

登录成功

```
req.session.sign = true;
req.session.name = us;
```

需要验证的接口判断是否存在

注销session

```
app.get('/out', function(req, res){
  req.session.destroy();
  res.redirect('/');
})
```

六、登录验证码

```
const svgCaptcha = require('svg-captcha');

生成验证码 返回图片格式
async generateVerifCode() {
  const codeConfig = {
  size: 4, // 验证码长度
  ignoreChars: '0o01ilI', // 验证码字符中排除 0o01ilI
  noise: 2, // 干扰线条的数量
  width: 160,
```

```
  height: 50,
  fontSize: 50,
  color: true, // 验证码的字符是否有颜色，默认没有，如果设定了背景，则默认有
  background: '#eee',
  };
  const captcha = svgCaptcha.create(codeConfig);
  this.ctx.session.verifCode = captcha.text.toLowerCase(); // 存session用于验证接口获取文字码
  this.ctx.body = captcha.data;
  }
```

day06

项目要求

> *1、登录注册*
>
> *2、权限验证*
>
> *3、数据增删改查*
>
> *4、分页、模糊查询*
>
> *5、mongoose表关联*
>
> *6、canvas做数据统计图*
>
> *7、项目上线*

app. js

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
var session = require("cookie-session")
var fs = require("fs");
var url = require("url");

var indexRouter = require('./routes/index');
```

```
var usersRouter = require('./routes/users');
var uploadRouter = require("./routes/upload")
var booksRouter = require("./routes/books");
var articleRouter = require("./routes/article")
var commentRouter = require("./routes/comment")
var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');


app.use(session({
 name: 'gp17',
 secret:"BK-GP-17",
 maxAge: 24 * 60 * 60 * 1000 //最大的存储时间
}))

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use("/upload",uploadRouter)
app.use("/books",booksRouter);
app.use("/article",articleRouter);

app.use("/comment",commentRouter)
// catch 404 and forward to error handler
app.use(function(req, res, next) {
 next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
 // set locals, only providing error in development
 res.locals.message = err.message;
 res.locals.error = req.app.get('env') === 'development' ? err : {};
```

```
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

router文件夹

```
//article.js
const express = require("express");
const router = express.Router();
const articleController = require("../controller/article");
//添加章节
router.post("/addArticle",articleController.addArticle)

//获取章节
router.get("/articleList",articleController.articleList)

//章节内容
router.get("/details",articleController.details)
module.exports = router;

//books.js
const express = require("express");
const router = express.Router();
const booksController = require("../controller/books");
const userAuth = require("../utils/sessionAuth")
//添加书籍
router.post("/add",userAuth.sessionAuth,booksController.add);

//书籍列表
router.get("/list",userAuth.sessionAuth,booksController.list);

//书籍删除
router.get("/delete",userAuth.sessionAuth,booksController.deleteCb);

//数据更新
router.post("/update",userAuth.sessionAuth,booksController.update);

//书籍搜索
```

```
//书籍搜索
router.get("/search",userAuth.sessionAuth,booksController.search);


//所有书籍
router.get("/booksAll",userAuth.sessionAuth,booksController.booksAll);
module.exports = router;

//comment.js
const express = require("express");
const router = express.Router();
const commentController = require("../controller/comment")


router.post("/add",commentController.add)


module.exports = router;

//upload.js
const express = require("express");
const router = express.Router();
const uploadController = require("../controller/upload")

//上传图片
router.post("/image", uploadController.upload)


module.exports = router;


//user.js
var express = require('express');
var router = express.Router();
var userController = require("../controller/user");

//验证码
router.get('/captch', userController.captch);

//注册
router.post("/register",userController.register)
```

```
73.
74.  //登录
75.  router.post("/login",userController.login)
76.
77.  //用户列表
78.  router.get("/userList",userController.userList);
79.
80.  //关闭用户登录状态
81.  router.post("/toggleStatus",userController.toggleStatus);
82.
83.  //搜索用户
84.  router.get("/searchList",userController.searchList);
85.
86.  //搜索用户状态
87.  router.get("/userstatus",userController.userstatusCb)
88.
89.  //所有用户
90.  router.get("/search",userController.search)
91.
92.  //修改用户信息
93.  router.post("/updateInfo",userController.updateInfo);
94.
95.
96.  //退出登录
97.  router.get("/logout",userController.logout)
98.  module.exports = router;
```

model

```
1.  //article.js
2.  const mongoose = require("../utils/database");
3.  const Schema = mongoose.Schema;
4.
5.  //定义字段
6.
7.  const articleSchema = new Schema({
8.   title:String,
9.   content:String,
10.  booksInfo:{
11.  type:Schema.Types.ObjectId,
12.  ref:"books"
13.  }
```

```
})


const Article = mongoose.model("articles",articleSchema)

//存

const articleSave = (articleInfo)=>{
 var article = new Article(articleInfo);
 return article.save()
}

//查

const articleList = (articleInfo)=>{
 return Article.find({booksInfo:articleInfo.booksInfo}).populate("booksInfo")
}

//详情
const articleDetails = (id)=>{
 return Article.findOne({_id:id},{title:1,content:1,_id:0})
}

module.exports = {
 articleSave,
 articleList,
 articleDetails
}

//books.js
const mongoose = require("../utils/database");

//定义表 和表的字段
const Schema = mongoose.Schema;
const booksSchema = new Schema({
 booksName: String,
 booksAuth: String,
 booksIntroduction: String,
```

```
  booksType: [String],
  booksStatus: String,
  booksUrl: String,

 })

 const Books = mongoose.model("books", booksSchema);


 //添加书籍
 const booksSave = (booksInfo) => {
  var books = new Books(booksInfo);
  return books.save();
 }

 //分页

 const booksList = (booksInfo) => {
  return Books.find().skip((booksInfo.page - 1) *
booksInfo.pageSize).limit(booksInfo.pageSize)
 }

 //书籍的总数量
 const booksListCount = () => {
  return Books.find().count()
 }


 //删除
 const booksDelete = (_id) => {
  return Books.remove({ _id })
 }


 //更新数据
 const booksUpdate = (_id, booksInfo) => {
  return Books.update({ _id }, { $set: booksInfo });
 }

```

```
95.  const booksSearch = (booksInfo) => {
96.
97.
98.   if (booksInfo.searchName&&booksInfo.searchType&&booksInfo.searchStatus) {
99.   return Books.find({booksName:new
RegExp(booksInfo.searchName),booksType:booksInfo.searchType,booksStatus:booksInfo.searchS
tatus})
100.  } else if (booksInfo.searchName && booksInfo.searchType) {
101.  return Books.find({ booksName: new RegExp(booksInfo.searchName), booksType:
booksInfo.searchType })
102.  } else if (booksInfo.searchName && booksInfo.searchStatus) {
103.  return Books.find({ booksName: new RegExp(booksInfo.searchName), booksStatus:
booksInfo.searchStatus })
104.  } else if (booksInfo.searchName) {
105.  return Books.find({ booksName: new RegExp(booksInfo.searchName)
}).skip((booksInfo.page-1)*booksInfo.pageSize).limit(booksInfo.pageSize);
106.  } else {
107.  return
Books.find().skip((booksInfo.page-1)*booksInfo.pageSize).limit(booksInfo.pageSize);
108.  }
109. }
110.
111.
112. //所有书籍
113.
114. const booksAllList = ()=>{
115.  return Books.find()
116. }
117.
118. module.exports = {
119.  booksSave,
120.  booksList,
121.  booksListCount,
122.  booksDelete,
123.  booksUpdate,
124.  booksSearch,
125.  booksAllList
126. }
127.
128. //comment.js
129. const mongoose = require("../utils/database");
```

```
const Schema = mongoose.Schema;

//定义字段

const commentSchema = new Schema({
 userId: Schema.Types.ObjectId,
 Nickname: String,
 userPic: String,
 content: String,
 date: Number,
 booksInfo: {
 type: Schema.Types.ObjectId,
 ref: "books"
 }
})


const Comment = mongoose.model("comments", commentSchema)




const commentSave = (commentInfo) => {
 var comment = new Comment({
 userId: commentInfo.userId,
 Nickname: commentInfo.Nickname,
 userPic: commentInfo.userPic,
 content: commentInfo.content,
 date: commentInfo.date,
 booksInfo: commentInfo.booksInfo
 })

 return comment.save();
}


module.exports = {
 commentSave
}
```

```
}

//user.js
const mongoose = require("../utils/database");

//定义表 和表的字段
const Schema = mongoose.Schema;
const userSchema = new Schema({
  username:String,
  password:String,
  registerDate:{ type: Number, default: Date.now },
  userPic:String,
  Nickname:String,
  userStatus:Boolean,
  userLevel:Number
})

const Users = mongoose.model("user",userSchema);


//查
const userFind = function(userInfo){
  return Users.findOne(userInfo)
}

//存

const userSave = function(userInfo){
  var user = new Users(userInfo);

  return user.save();
}


//list查
const userFindList = function(userInfo){
  return Users.find({},
{username:1,registerDate:1,userPic:1,Nickname:1,userStatus:1}).skip((userInfo.page-1)*use
rInfo.pageSize).limit(userInfo.pageSize)
}
```

```
209.
210. //查多少用户
211. const userListCount = (userInfo)=>{
212.  return Users.find(userInfo).count();
213. }
214.
215.
216. //修改用户登录状态
217.
218. const userStatusUpdate = (id)=>{
219.  return Users.update({_id:id},{$set:{userStatus:false}})
220. }
221.
222. //查用户名称
223. const userSearchnickname = (userInfo)=>{
224.  return Users.find({Nickname:new RegExp(userInfo.nickname)});
225. }
226.
227. //用户状态查询
228. const userSearchStatus = (userInfo)=>{
229.
230.  return Users.find({userStatus:userInfo.userStatus});
231. }
232.
233.
234. const userSearch = (userInfo)=>{
235.  return Users.find({Nickname:new
    RegExp(userInfo.nickname),userStatus:userInfo.userStatus})
236. }
237.
238.
239. //修改用户信息
240.
241. const userUpdate = (id,userInfo)=>{
242.  return Users.update({_id:id},{$set:userInfo})
243. }
244.
245. module.exports = {
246.  userFind,
247.  userSave,
248.  userFindList,
```

```
  userListCount,
  userStatusUpdate,
  userSearchnickname,
  userSearchStatus,
  userSearch,
  userUpdate
 }
```

controller文件

```
//article.js
const articleModel = require("../model/article");

const addArticle = async (req,res)=>{
 console.log(123)
 let {title,content,booksInfo} = req.body;
 let data = await articleModel.articleSave({title,content,booksInfo});
 res.json({
 code:200,
 errMsg:"",
 data:{
 info:"添加成功",
 code:1
 }
 })

}


const articleList = async (req,res)=>{
 let {booksInfo} = req.query;
 let data = await articleModel.articleList({booksInfo});

 res.json({
 code:200,
 errMsg:"",
 data:{
 info:"获取成功",
 data
 }
 })
```

```
33.
34.  }
35.
36.
37.
38.  const details = async (req,res)=>{
39.   let {booksInfo} = req.query;
40.
41.   let data = await articleModel.articleDetails(booksInfo);
42.   if(data){
43.   res.json({
44.   code:200,
45.   errMsg:"",
46.   data:{
47.   info:'获取成功',
48.   data,
49.   code:1
50.   }
51.   })
52.   }else{
53.   res.json({
54.   code:200,
55.   errMsg:"",
56.   data:{
57.   info:'获取失败',
58.   code:2
59.   }
60.   })
61.   }
62.
63.  }
64.
65.  module.exports = {
66.   addArticle,
67.   articleList,
68.   details
69.  }
70.
71.
72.  //books.js
73.  const booksModel = require("../model/books");
```

```
const add = async (req,res,next)=>{

  var {booksName,booksAuth,booksIntroduction,booksType,booksStatus,booksUrl} = req.body;

  if(!booksUrl){
  booksUrl = "http://10.60.15.150:3000/img/default.jpg"
  }
  var arr = booksType.split(",");
  let data = await booksModel.booksSave({booksName,booksAuth,booksIntroduction,booksType:arr,booksStatus,booksUrl:booksUrl})
  res.json({
  code:200,
  errMsg:"",
  data:{
  info:"添加成功",
  code:1
  }
  })

}


const list = async (req,res)=>{
  let {pageSize,page} = req.query;

  let data = await booksModel.booksList({pageSize:Number(pageSize),page:Number(page)});
  let count = await booksModel.booksListCount();

  res.json({
  code:200,
  errMsg:"",
  data:{
  info:"获取成功",
  data,
  count
  }
  })


}
```

```
}


const deleteCb = async (req,res)=>{
  var {_id} = req.query;

  var data = await booksModel.booksDelete(_id);
  if(data.ok == 1){
  res.json({
  code:200,
  errMsg:"",
  data:{
  info:"删除成功",
  code:1
  }
  })
  }else{
  res.json({
  code:200,
  errMsg:"",
  data:{
  info:"删除失败",
  code:2
  }
  })
  }


}


const update = async (req,res)=>{
  //const {booksName,booksAuth,booksIntroduction,booksType,booksStatus,booksUrl,_id} = req.body;
  const {_id,...rest} = req.body;
  const {booksType,...rests} = rest;

  var arr = booksType.split(",");
```

```
  let data = await booksModel.booksUpdate(_id,{booksType:arr,...rests});

  if(data.ok == 1){
  res.json({
  code:200,
  errMsg:"",
  data:{
  info:"修改成功",
  code:1
  }
  })
  }else{
  res.json({
  code:200,
  errMsg:"",
  data:{
  info:"服务器错误",
  code:2
  }
  })
  }

}


const search = async (req,res)=>{
  var {searchName,searchType,searchStatus,pageSize,page} = req.query;
  let data = await booksModel.booksSearch({searchName,searchType,searchStatus,pageSize:Number(pageSize),page:Number(page)});
  let count = await booksModel.booksListCount();
  res.json({
  code:200,
  errMsg:"",
  data:{
  data,
  code:1,
  count
  }
  })
```

```
191.  }
192.
193.  const booksAll = async (req,res)=>{
194.   let data = await booksModel.booksAllList();
195.   res.json({
196.   code:200,
197.   errMsg:"",
198.   data:{
199.   info:"获取成功",
200.   data
201.   }
202.   })
203.  }
204.
205.  module.exports = {
206.   add,
207.   list,
208.   deleteCb,
209.   update,
210.   search,
211.   booksAll
212.  }
213.
214.  //comment.js
215.  const commentModel = require("../model/comment");
216.
217.
218.  const add = async (req,res)=>{
219.   var {userId,Nickname,userPic,booksId,content} = req.body;
220.   var date = new Date().getTime();
221.
222.   let data = await
commentModel.commentSave({userId,Nickname,userPic,booksId,content,date});
223.
224.   res.json({
225.   code:200,
226.   errMsg:"",
227.   data:{
228.   info:"发布成功",
229.   code:200
230.   }
```

```
  })


}


module.exports = {
 add
}

//upload.js
const cpUpload = require("../utils/upload");

const upload = (req, res, next) => {
 cpUpload(req, res, (err) => {
 if (err) {
 res.json({
 code: 200,
 errMsg: "",
 data: {
 info: "服务器错误",
 code: 0,
 err: err
 }
 })
 } else {
 var urlpath = "http://10.60.15.150:3000/img/" + req.files.booksUrl[0].filename;
 res.json({
 code: 200,
 errMsg: "",
 data: {
 url: urlpath,
 code: 1
 }
 })
 }


 })
}

```

```
272.
273. module.exports = {
274.  upload
275. }
276.
277. //user.js
278. //引入验证码依赖包
279. var svgCaptcha = require('svg-captcha');
280. var userModel = require("../model/user")
281. var crypto = require('crypto');
282.
283.
284.
285.
286.
287.
288. var store = {}
289. const captch = (req, res) => {
290.  const captcha = svgCaptcha.create({
291.  size: 4, // 验证码长度
292.  ignoreChars: '0o1i', // 验证码字符中排除 0o1i
293.  noise: 3, // 干扰线条的数量
294.  color: true, // 验证码的字符是否有颜色，默认没有，如果设定了背景，则默认有
295.  background: '#cc9966', // 验证码图片背景颜色
296.  })
297.
298.  //在服务端保存生成的验证码
299.  store.captch = captcha.text.toLowerCase();
300.
301.  //captcha 是一个对象 {data:svg地址,text:验证码};
302.  res.send(captcha)
303. }
304.
305.
306. const register = async (req, res, next) => {
307.  var { username, password, captch } = req.body;
308.
309.  if ( store.captch === captch.toLowerCase()) {
310.  let data = await userModel.userFind({ username })
311.  if (data) {
312.  res.json({
```

```
312.    res.json({
313.    code: 200,
314.    errMsg: "",
315.    data: {
316.    info: "用户名重复",
317.    code: 2
318.    }
319.    })
320.    } else {
321.    //创建加密方式
322.    var hash = crypto.createHash("sha256");
323.    //需要加密的数据
324.    hash.update(password)
325.
326.
327.    var obj = {};
328.    obj.username = username;
329.    //获取加密的数据
330.    obj.password = hash.digest('hex');
331.    obj.registerDate = new Date().getTime();
332.    obj.userPic = "http://10.60.15.150:3000/img/timg.jpg";
333.    obj.Nickname = Math.random().toString(16).substring(2, 8);
334.    obj.userStatus = true;
335.    obj.userLevel = 0;
336.
337.
338.    let userSaveData = await userModel.userSave(obj);
339.
340.    res.json({
341.    code: 200,
342.    errMsg: "",
343.    data: {
344.    info: "注册成功",
345.    code: 1
346.    }
347.    })
348.
349.    }
350.
351.
352.    } else {
```

```
  res.json({
  code: 200,
  errmsg: "",
  data: {
  info: "验证码错误",
  code: -1
  }
  })
  }


}


const login = async (req, res) => {
  var { username, password, captch } = req.body;

  if ( store.captch === captch.toLowerCase()) {

  let data = await userModel.userFind({ username });
  if (data.userStatus) {
  //创建加密方式
  var hash = crypto.createHash("sha256");
  //需要加密的数据
  hash.update(password)

  if (data.password === hash.digest('hex')) {

  req.session["userId"] = username;


  res.json({
  code: 200,
  errmsg: "",
  data: {
  info: "登录成功",
  _id: data._id,
  userPic: data.userPic,
  Nickname: data.Nickname,
```

```
394.   code: 1
395.   }
396.   })
397.
398.
399.
400.   } else {
401.   res.json({
402.   code: 200,
403.   errmsg: "",
404.   data: {
405.   info: "密码错误",
406.   code: 3
407.   }
408.   })
409.   }
410.
411.
412.   } else {
413.   res.json({
414.   code: 200,
415.   errmsg: "",
416.   data: {
417.   info: "您有不良行为，请联系管理",
418.   code: 2
419.   }
420.   })
421.   }
422.
423.   } else {
424.   res.json({
425.   code: 200,
426.   errmsg: "",
427.   data: {
428.   info: "验证码错误",
429.   code: -1
430.   }
431.   })
432.   }
433. }
434.
```

```
435.
436.
437.  const userList = async (req, res) => {
438.   let { pageSize, page } = req.query;
439.   let data = await userModel.userFindList({ pageSize: Number(pageSize), page:
      Number(page) });
440.   let count = await userModel.userListCount();
441.
442.   res.json({
443.   code: 200,
444.   errMsg: "",
445.   data: {
446.   code: 1,
447.   data,
448.   count
449.   }
450.   })
451.
452.  }
453.
454.
455.
456.  const toggleStatus = async (req, res) => {
457.   let { userId, actionId } = req.body;
458.   let data = await userModel.userFind({ _id: actionId });
459.   console.log(data);
460.   if (data.userLevel == 1001) {
461.   let userData = await userModel.userStatusUpdate(userId);
462.   res.json({
463.   code: 200,
464.   errMsg: "",
465.   data: {
466.   info: "OK",
467.   code: 1
468.   }
469.   })
470.
471.
472.   } else {
473.   res.json({
474.   code: 200
```

```
    code: 200,
    errMsg: "",
    data: {
    info: "您没有权限访问该功能，请联系相关人员授权",
    code: 2
    }
    })
    }
}


const searchList = async (req,res)=>{
    var {nickname} = req.query;
    let data = await userModel.userSearchnickname({nickname});
    res.json({
    code:200,
    errmsg:"",
    data:{
    info:"查询成功",
    code:1,
    data
    }
    })

}


const userstatusCb = async (req,res)=>{
    var {userStatus} = req.query;
    var bStop = true;


    if(userStatus==2){

    bStop = false;
    }

    let data = await userModel.userSearchStatus({userStatus:bStop});
    res.json({
    code:200,
    errmsg:"",
```

```
  data:{
  info:"查询成功",
  code:1,
  data
  }
  })


 }


const search = async (req,res)=>{
  var {nickname,userStatus} = req.query;
  if(nickname && userStatus){
  var bStop = true;
  if(userStatus==2){
  bStop = false;
  }

  let data = await userModel.userSearch({nickname,userStatus:bStop});

  res.json({
  code:200,
  errmsg:"",
  data:{
  info:"查询成功",
  code:1,
  data
  }
  })
  }


 }


const updateInfo = async (req,res)=>{
  var {Nickname,userPic,id} = req.body;

  let data = await userModel.userUpdate(id,{Nickname,userPic});

```

```
  if(data.ok == 1){
  res.json({
  code:200,
  errMsg:"",
  data:{
  info:"修改成功",
  code:1,
  Nickname,
  userPic,
  id
  }
  })
  }
 }

 const logout = (req,res)=>{
  delete req.session["userId"];
  res.json({
  code:200,
  errMsg:"",
  data:{
  info:"退出登录",
  code:1
  }
  })
 }

 module.exports = {
  captch,
  register,
  login,
  userList,
  toggleStatus,
  searchList,
  userstatusCb,
  search,
  updateInfo,
  logout
 }
```

utils文件夹

```
//database.js
const mongoose = require("mongoose");
const url = "mongodb://127.0.0.1:27017/gp17";

mongoose.connect(url,(err)=>{
 if(err){
 console.log("连接失败")
 }else{
 console.log("连接成功")
 }
})


module.exports = mongoose;

//session_auth.js
const sessionAuth = (req,res,next)=>{
 if(req.session.userId){
 next();
 }else{
 res.json({
 code:200,
 errMsg:"",
 data:{
 info:"非法登录",
 code:2
 }
 })
 }
}


module.exports = {
 sessionAuth
}

//upload.js
//1、引入
const multer = require("multer");

//2、设置文件的存储位置以及文件的名称
```

```
var storage = multer.diskStorage({
 //将文件存储到指定的位置
 destination: function (req, file, cb) {
 cb(null, './public/img')
 },
 //更改上传后文件的名称
 filename: function (req, file, cb) {
 cb(null, Date.now() + '-' + file.originalname)

 }
})

//3、应用上面的配置项
var upload = multer({ storage: storage })

//4、设置当前字段最多能上传多少个文件
var cpUpload = upload.fields([{ name: 'booksUrl', maxCount: 1 }])


module.exports = cpUpload;
```

前端JS文件

```
class Details{
 constructor(){
 this.container = $("#details");
 this.init()
 }
 init(){
 var id = this.getBooksId();
 this.getBooksDetails(id);
 }
 getBooksId(){
 var path = window.location.href;
 return path.split("?")[1].split("=")[1]
 }
 getBooksDetails(id){
 $.ajax({
 type:"get",
 url:"/article/details",
 data:{
 booksInfo:id
```

```
20.   },
21.   success:this.handleGetBooksDetailsSucc.bind(this)
22.   })
23.   }
24.   handleGetBooksDetailsSucc(data){
25.
26.   if(data.data.code == 1){
27.   var h1 = $("<h1></h1>");
28.   var div = $("<div class='content'></div>");
29.   h1.text(data.data.data.title);
30.   div.html(data.data.data.content);
31.   this.container.append(h1);
32.   this.container.append(div);
33.   }else{
34.   alert(data.data.info);
35.   window.location.href="http://10.60.15.150:3000/html/list.html";
36.   }
37.   }
38.
39.  }
40.
41.  new Details();
42.
43.
44.  class AddBooks {
45.   constructor() {
46.   this.container = $(".list-content");
47.   this.booksInfo = {};
48.   this.booksTypeFlag = false;
49.   }
50.   init() {
51.   this.createContent();
52.   }
53.   createContent() {
54.   this.container.html(AddBooks.template);
55.   this.upload();
56.   this.add();
57.   this.booksTypesChange();
58.   }
59.   upload() {
60.   $("#booksImage").on("change", this.handleUploadCb.bind(this))
```

```
61.   }
62.   handleUploadCb() {
63.   var file = $("#booksImage")[0].files[0];
64.
65.   // 1、模拟form表单上传文件 new FileReader 文件转换base64 dataUrl(小)
66.   var formData = new FormData();
67.   //2、将需要上传的对象通过append添加到formData中去 第一个参数是key值 第二个参数是value
      值
68.   formData.append("booksUrl", file)
69.
70.
71.   $.ajax({
72.   type: "post",
73.   url: "/upload/image",
74.   //3、取消JQ中ajax的默认配置项
75.   cache: false,
76.   processData: false,
77.   contentType: false,
78.   data: formData,
79.   success: this.handleUploadSucc.bind(this)
80.   })
81.   }
82.   handleUploadSucc(data) {
83.   if (data.data.code == 1) {
84.   var img = $("<img/>");
85.   img.attr("src", data.data.url);
86.   img.css({
87.   width: 85,
88.   height: 120
89.   })
90.   $(".upload>div").html(img);
91.
92.   this.booksInfo.booksUrl = data.data.url;
93.   }
94.   }
95.   add() {
96.   $("#addBooksForm").on("submit", this.handleAddBooksCb.bind(this))
97.   }
98.   handleAddBooksCb(e) {
99.   e.preventDefault();
100.  if (this.booksTypeFlag) {
```

```
100.  if (this.booksTypeFlag) {
101.  this.booksInfo.booksName = $("#booksName").val();
102.  this.booksInfo.booksAuth = $("#booksAuth").val();
103.  this.booksInfo.booksIntroduction = $("#booksIntroduction").val();
104.  this.booksInfo.booksType = $("#booksType").val()
105.  this.booksInfo.booksStatus = $("#booksStatus").val();
106.
107.  $.ajax({
108.  type: "post",
109.  url: "/books/add",
110.  data: this.booksInfo,
111.  success: this.handleAddBooksSucc.bind(this)
112.  })
113.  }else{
114.  alert("书籍类型格式有误")
115.  }
116.
117.
118.  }
119.  handleAddBooksSucc(data) {
120.  if (data.data.code == 1) {
121.  alert("添加成功");
122.  new SliderBar().handleTabBarCb(2);
123.  }
124.  }
125.  booksTypesChange() {
126.  $("#booksType").on("change", this.handleBooksTypesChange.bind(this))
127.  }
128.  handleBooksTypesChange(e) {
129.  var val = e.target.value;
130.  console.log(val);
131.  if (/^([\u4e00-\u9fa5]+,?){1,4}$/.test(val)) {
132.  this.booksTypeFlag = true;
133.  }
134.  }
135. }
136.
137. AddBooks.template = `
138.  <div class="booksForm">
139.  <form id="addBooksForm">
140.  <div class="form-group">
```

```
  <label for="booksName">书籍名称</label>
  <input type="text" class="form-control" id="booksName" placeholder="请输入书籍名称">
  </div>
  <div class="form-group">
  <label for="booksAuth">书籍作者</label>
  <input type="text" class="form-control" id="booksAuth" placeholder="请输入书籍作者">
  </div>
  <div class="form-group">
  <label for="booksIntroduction">书籍简介</label>
  <textarea class="form-control" rows="3" placeholder="请输入书籍简介"
id="booksIntroduction"></textarea>
  </div>
  <div class="form-group">
  <label for="booksType">书籍类型</label>
  <input type="text" class="form-control" id="booksType" placeholder="请选择书籍类型">
  </div>
  <div class="form-group">
  <label for="booksStatus">书籍状态</label>
  <select class="form-control" id="booksStatus">
  <option>已完结</option>
  <option>连载中</option>
  </select>
  </div>
  <div class="form-group">
  <label for="booksImage">书籍封面</label>
  <div class="upload">
  <div>
  <span>+</span>
  </div>
  <input type="file" id="booksImage">
  </div>
  </div>
  <button type="submit" class="btn btn-primary">添加书籍</button>
  </form>
  </div>
 `


class ArticleAction {
  constructor() {
  this.container = $(".list-content")
```

```
}
init() {
this.createContent();
}
createContent() {
this.container.html(ArticleAction.template);
this.getBooksNameSelect();
this.createwangEditor();
this.publicArticle();

}
getBooksNameSelect() {
$.ajax({
type: "get",
url: "/books/booksAll",
success: this.handleGetBooksNameSelect.bind(this)
})
}
handleGetBooksNameSelect(data) {
var dataList = data.data.data;
var str = "";
for (var i = 0; i < dataList.length; i++) {
str += `
<option value="${dataList[i]._id}">${dataList[i].booksName}</option>
`
}
$(".booksList").html(str);
this.getBooksArticeList();
this.seletedBooks();
}
createwangEditor(){
this.Editor = new wangEditor("#content");
this.Editor.create();
}
publicArticle(){
$("#publicArticle").on("submit",this.handlePublicArticleCb.bind(this))
}
handlePublicArticleCb(e){
e.preventDefault();
var title = $("#articleTitle").val();
var content = this.Editor.txt.html();
```

```
var booksInfo = $(".booksList").val();

$.ajax({
type:"post",
url:"/article/addArticle",
data:{
title,
content,
booksInfo

},
success:this.handlePublicSucc.bind(this)
})
}
handlePublicSucc(data){
if(data.data.code == 1){
alert("添加成功");
this.getBooksArticeList();
}
$('#addArticle').modal('hide')
}
getBooksArticeList(){
var booksInfo = $(".booksList").val();

$.ajax({
type:'get',
url:"/article/articleList",
data:{
booksInfo
},
success:this.handleGetBooksArticleListSucc.bind(this)
})
}
handleGetBooksArticleListSucc(data){
var dataList = data.data.data;

var str = "";

for(var i=0;i<dataList.length;i++){
str += `<div data-id=${dataList[i]._id}>${dataList[i].title}</div>
```

```
    `
    }

    $(".articleList").html(str);
    this.articleDes();
    }
    seletedBooks(){
    $(".booksList").on("change",this.handleSelectedCb.bind(this))
    }
    handleSelectedCb(){
    this.getBooksArticeList();
    }
    articleDes(){
    $(".articleList>div").each(this.handleArticleListDesEach.bind(this))
    }
    handleArticleListDesEach(index){

$(".articleList>div").eq(index).on("click",this.handleArticleListDesCb.bind(this,index))
    }
    handleArticleListDesCb(index){
    var booksInfo = $(".articleList>div").eq(index).attr("data-id");
    window.location.href = "http://10.60.15.150:3000/html/details.html?id="+booksInfo
    }

  }

  ArticleAction.template = `
    <div class="ArticleAction">
    <div class="form-inline">
    <select class="form-control booksList">

    </select>
    <button class="btn btn-primary" data-toggle="modal" data-target="#addArticle">添加章节</button>
    </div>
    <div class="articleList">
    <div>第1章 被杀的光杆一个的皇帝</div>
    <div>第1章 被杀的光杆一个的皇帝</div>
    <div>第1章 被杀的光杆一个的皇帝</div>
    <div>第1章 被杀的光杆一个的皇帝</div>
```

```
  <div>第1章 被杀的光杆一个的皇帝</div>
  <div>第1章 被杀的光杆一个的皇帝</div>
  <div>第1章 被杀的光杆一个的皇帝</div>
  <div>第1章 被杀的光杆一个的皇帝</div>
  </div>
  <div class="modal fade" id="addArticle" tabindex="-1" role="dialog" aria-labelledby="myModalLabel">
  <div class="modal-dialog" role="document">
  <div class="modal-content">
  <div class="modal-header">
  <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-hidden="true">&times;</span></button>
  <h4 class="modal-title" id="myModalLabel">添加章节</h4>
  </div>
  <div class="modal-body">
  <form id="publicArticle">
  <div class="form-group">
  <label for="articleTitle">章节标题</label>
  <input type="text" class="form-control" id="articleTitle" placeholder="请输入章节标题">
  </div>

  <div class="form-group">
  <label for="articleTitle">章节内容</label>
  <div id="content"></div>
  </div>
  <button type="submit" class="btn btn-primary">发布</button>
  </form>
  </div>
  </div>
  </div>
  </div>
  </div>
  `

 class BooksAction {
  constructor() {
  this.container = $(".list-content");
  this.booksListData = {
  pageSize: 5,
  page: 1
```

```
339.    }
340.    this.flag = true;
341.    this.activeBooksInfo = {};
342.    }
343.    init() {
344.    this.createContent();
345.    }
346.    createContent() {
347.    this.container.html(BooksAction.template);
348.    this.getBooksList(this.booksListData);
349.    this.upload();
350.    this.booksSearch();
351.    }
352.    getBooksList({ pageSize, page }) {
353.    // limit 10 page 1
354.    this.flag = true;
355.    $.ajax({
356.    type: "get",
357.    url: "/books/list",
358.    data: {
359.    pageSize,
360.    page
361.    },
362.    success: this.handleGetBooksListSucc.bind(this)
363.    })
364.    }
365.    handleGetBooksListSucc(data) {
366.    if (data.data.data.length > 0) {
367.    var str = "";
368.    var type = "";
369.    this.dataList = data.data.data;
370.
371.
372.    for (var i = 0; i < this.dataList.length; i++) {
373.    for (var j = 0; j < this.dataList[i].booksType.length; j++) {
374.    type += `
375.    <span>${this.dataList[i].booksType[j]}</span>
376.    `
377.    }
378.
379.
```

```
  str += `
  <tr>
  <td>${ this.dataList[i].booksName}</td>
  <td>${ this.dataList[i].booksAuth}</td>
  <td><img src="${ this.dataList[i].booksUrl}"></td>
  <td>
  ${type}
  </td>
  <td>
  ${ this.dataList[i].booksStatus}
  </td>
  <td data-id="${ this.dataList[i]._id}">
  <button type="button" class="btn btn-link btn-actions" data-toggle="modal" data-target="#booksModal">操作</button>
  <button type="button" class="btn btn-link btn-delete">删除</button>
  </td>
  </tr>`

  var type = "";

  }
  $(".booksActions tbody").html(str);

  //删除操作
  this.booksdelete();
  //修改操作
  this.booksUpdate();

  if (this.flag) {
  this.createPageList(data.data.count)
  }

  this.flag = false;

  }
  }
  createPageList(count) {
  var _that = this;
  layui.use('laypage', function () {
  var laypage = layui.laypage;
  laypage.render({
```

```
elem: 'pageList',
count,
limit: 5,
groups: 5,
jump: _that.handleCreatePageListCb.bind(_that)
});
});
}
handleCreatePageListCb(obj, first) {
if (!first) {
this.getBooksList({ pageSize: obj.limit, page: obj.curr })
}
}
booksdelete() {
$(".btn-delete").each(this.handleBtnDeleteEach.bind(this))
}
handleBtnDeleteEach(index) {
$(".btn-delete").eq(index).on("click", this.handleBtnDeleteCb.bind(this, index))
}
handleBtnDeleteCb(index) {
this.id = $(".btn-delete").eq(index).parent().attr("data-id");
this.booksName = $(".btn-delete").eq(index).parent().parent().children().eq(0).text();


layui.use("layer", this.handleLayUICb.bind(this))

}
handleLayUICb() {
var layer = layui.layer;
layer.open({
type: 0,
title: "删除书籍",
content: `您确定要删除《${this.booksName}》书籍吗？`, //这里content是一个普通的String
btn: ['确认', '取消'],
yes: this.handleLayOk.bind(this),
btn2: function (index, layero) { },
cancel: function () { }
});
}
handleLayOk(index, layero) {
```

```
460.    $.ajax({
461.    type: "get",
462.    url: "/books/delete",
463.    data: {
464.    _id: this.id
465.    },
466.    success: this.handlebooksDeleteSucc.bind(this, index)
467.    })
468.    }
469.    handlebooksDeleteSucc(index, data) {
470.    if (data.data.code == 1) {
471.    this.getBooksList(this.booksListData)
472.    layer.close(index);
473.    } else {
474.    alert(data.data.info)
475.    }
476.
477.    }
478.    //修改操作
479.    booksUpdate() {
480.    $(".btn-actions").each(this.handleBtnActionsEach.bind(this))
481.    }
482.    handleBtnActionsEach(index) {
483.    $(".btn-actions").eq(index).on("click", this.handleBtnActionsCb.bind(this, index))
484.    }
485.    handleBtnActionsCb(index) {
486.    var id = $(".btn-actions").eq(index).parent().attr("data-id");
487.    //拿到当前行的数据
488.    for (var i = 0; i < this.dataList.length; i++) {
489.    if (id == this.dataList[i]._id) {
490.    this.activeBooksInfo = this.dataList[i];
491.    break;
492.    }
493.    }
494.
495.
496.    this.updateBooksName = $("#updatebooksName");
497.    this.updatebooksAuth = $("#updatebooksAuth")
498.    this.updatebooksIntroduction = $("#updatebooksIntroduction")
499.    this.updatebooksType = $("#updatebooksType")
500.    this.updatebooksStatus = $("#updatebooksStatus")
```

```
501.  this.updateBooksForm = $("#updateBooksForm");
502.
503.  this.updateBooksName.val(this.activeBooksInfo.booksName);
504.  this.updatebooksAuth.val(this.activeBooksInfo.booksAuth);
505.  this.updatebooksIntroduction.val(this.activeBooksInfo.booksIntroduction);
506.  this.updatebooksType.val(this.activeBooksInfo.booksType.join());
507.  this.updatebooksStatus.val(this.activeBooksInfo.booksStatus);
508.
509.  this.modifyBooks();
510.  }
511.  modifyBooks() {
512.  this.updateBooksForm.on("submit", this.handleModifyBooksCb.bind(this))
513.  }
514.  handleModifyBooksCb(e) {
515.
516.  e.preventDefault();
517.  this.activeBooksInfo.booksName = this.updateBooksName.val();
518.  this.activeBooksInfo.booksAuth = this.updatebooksAuth.val();
519.  this.activeBooksInfo.booksIntroduction = this.updatebooksIntroduction.val();
520.  this.activeBooksInfo.booksType = this.updatebooksType.val();
521.  this.activeBooksInfo.booksStatus = this.updatebooksStatus.val();
522.
523.
524.  $.ajax({
525.  type: "post",
526.  url: "/books/update",
527.  data: this.activeBooksInfo,
528.  success: this.handleMofidySucc.bind(this)
529.  })
530.  }
531.  handleMofidySucc(data) {
532.  if (data.data.code == 1) {
533.  alert(data.data.info);
534.  $('#booksModal').modal('hide')
535.  this.getBooksList(this.booksListData);
536.  } else {
537.  alert(data.data.info)
538.  }
539.  }
540.  //上传图片
541.  upload() {
```

```
542.  $("#updatebooksImage").on("change", this.handleUploadCb.bind(this))
543.  }
544.  handleUploadCb() {
545.  var file = $("#updatebooksImage")[0].files[0];
546.
547.  // 1、模拟form表单上传文件 new FileReader 文件转换base64 dataUrl(小)
548.  var formData = new FormData();
549.  //2、将需要上传的对象通过append添加到formData中去 第一个参数是key值 第二个参数是value
值
550.  formData.append("booksUrl", file)
551.
552.
553.  $.ajax({
554.  type: "post",
555.  url: "/upload/image",
556.  //3、取消JQ中ajax的默认配置项
557.  cache: false,
558.  processData: false,
559.  contentType: false,
560.  data: formData,
561.  success: this.handleUploadSucc.bind(this)
562.  })
563.  }
564.  handleUploadSucc(data) {
565.  if (data.data.code == 1) {
566.  var img = $("<img/>");
567.  img.attr("src", data.data.url);
568.  img.css({
569.  width: 85,
570.  height: 120
571.  })
572.  $(".upload>div").html(img);
573.
574.  this.activeBooksInfo.booksUrl = data.data.url;
575.  console.log(this.activeBooksInfo)
576.  }
577.  }
578.  booksSearch(){
579.  $(".form-search").on("submit",this.handleBooksSearchCb.bind(this))
580.  }
581.  handleBooksSearchCb(e){
```

```
582.   e.preventDefault();
583.
584.   var searchName = $("#booksName_search").val();
585.   var searchType = $("#booksType_search").val();
586.   var searchStatus = $("#booksStatus_search").val();
587.   $.ajax({
588.   type:"get",
589.   url:"/books/search",
590.   data:{
591.   searchName,
592.   searchType,
593.   searchStatus,
594.   pageSize:5,
595.   page:1
596.   },
597.   success:this.handleBooksSearchSucc.bind(this)
598.   })
599.
600.   }
601.   handleBooksSearchSucc(data){
602.   this.flag = true;
603.   this.handleGetBooksListSucc(data)
604.   }
605.  }
606.
607.  BooksAction.template = `
608.   <div class="booksActions">
609.   <form class="form-search form-inline">
610.   <div class="form-group">
611.   <label for="booksName_search">书籍名称</label>
612.   <input type="text" class="form-control" id="booksName_search" placeholder="请输入要搜索
      的书籍名称">
613.   </div>
614.   <div class="form-group">
615.   <label for="booksStatus_search">书籍状态</label>
616.   <select class="form-control" id="booksStatus_search">
617.   <option value="">全部</option>
618.   <option value="连载中">连载中</option>
619.   <option value="已完结">已完结</option>
620.   </select>
621.   </div>
```

```
</div>
<div class="form-group">
<label for="booksType_search">书籍类型</label>
<select class="form-control" id="booksType_search">
<option value="">全部</option>
<option value="玄幻">玄幻</option>
<option value="修仙">修仙</option>
<option value="爱情">爱情</option>
<option value="动作">动作</option>
</select>
</div>
<button type="submit" class="btn btn-primary">搜索</button>
</form>
<table class="table table-striped">
<thead>
<tr>
<th>书籍名称</th>
<th>作者</th>
<th>书籍封面</th>
<th>书籍类型</th>
<th>书籍状态</th>
<th>操作</th>
</tr>
</thead>
<tbody>
<tr>
<td>完美世界</td>
<td>辰东</td>
<td><img style="width:50px;height:70px;" src="../../../img/1578736782091-u=3508817781,1654561115&fm=58&s=6D836397167360B3087070600300E070.jpg"/></td>
<td>
<span>武侠</span>
<span>爱情</span>
<span>修仙</span>
</td>
<td>
连载中
</td>
<td>
<button type="button" class="btn btn-link ">操作</button>
<button type="button" class="btn btn-link ">删除</button>
```

```
  </td>
  </tr>
  </tbody>
  </table>
  <div id="pageList"></div>
  </div>
  `
class Comment{
  constructor(){
  this.container = $(".list-content")
  }
  init(){
  this.createContent();
  }
  createContent(){
  this.container.html(Comment.template)
  this.getBooksNameSelect();
  this.createwangEditor();
  this.public();

  }
  getBooksNameSelect() {
  $.ajax({
  type: "get",
  url: "/books/booksAll",
  success: this.handleGetBooksNameSelect.bind(this)
  })
  }
  handleGetBooksNameSelect(data) {
  var dataList = data.data.data;
  var str = "";
  for (var i = 0; i < dataList.length; i++) {
  str += `
  <option value="${dataList[i]._id}">${dataList[i].booksName}</option>
  `
  }
  $(".booksList").html(str);

  }
  createwangEditor(){
  this.Editor = new wangEditor("#content");
```

```
  this.Editor.create();
  }
  public(){
  $(".public").on("click",this.handlePublicCb.bind(this))
  }
  handlePublicCb(){
  var content = this.Editor.txt.text();
  var booksId = $(".booksList").val();
  var userInfo = JSON.parse(window.sessionStorage.getItem("userinfo"));

  $.ajax({
  type:"post",
  url:"/comment/add",
  data:{
  userId:userInfo._id,
  Nickname:userInfo.Nickname,
  userPic:userInfo.userPic,
  booksId,
  content
  },
  success:this.handlePublicSucc.bind(this)
  })
  }
  handlePublicSucc(data){
  console.log(data);
  }

 }
 Comment.template =`
 <div class="ArticleAction">
 <div class="form-inline">
  <select class="form-control booksList">

  </select>
  <button class="btn btn-primary" data-toggle="modal" data-target="#addArticle">写评论
</button>
 </div>
  <div class="articleList">

  </div>
  <div class="modal fade" id="addArticle" tabindex="-1" role="dialog"
```

```
aria-labelledby="myModalLabel">
  <div class="modal-dialog" role="document">
  <div class="modal-content">
  <div class="modal-header">
  <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span
aria-hidden="true">&times;</span></button>
  <h4 class="modal-title" id="myModalLabel">写评论</h4>
  </div>
  <div class="modal-body">

  <div class="form-group">
  <label for="articleTitle">写评论</label>
  <div id="content"></div>
  </div>
  <button type="button" class="btn btn-primary public">发布</button>

  </div>
  </div>
  </div>
  </div>
</div>
`


class Home{
  constructor(){
  this.container = $(".list-content")
  }
  init(){
  this.createContent();
  }
  createContent(){
  this.container.html(Home.template)
  }
}
Home.template = `
  <div>home</div>
`

class Mine {
  constructor() {
```

```
781.  this.container = $(".list-content");
782.  }
783.  init() {
784.  this.createContent();
785.  }
786.  createContent() {
787.  this.container.html(Mine.template);
788.  if(window.sessionStorage.getItem("userinfo")){
789.  this.userInfo = JSON.parse(window.sessionStorage.getItem("userinfo"));
790.  $(".userInfo_update>img").attr("src",this.userInfo.userPic)
791.  $("#username").val(this.userInfo.Nickname)
792.  }
793.
794.  this.updatUserPic();
795.  this.update();
796.  }
797.  //修改头像
798.  updatUserPic(){
799.  $("#userPic").on("change",this.handleUpdateUserPic.bind(this))
800.  }
801.  handleUpdateUserPic(){
802.  var file = $("#userPic")[0].files[0];
803.
804.  var formData = new FormData();
805.  formData.append("booksUrl",file);
806.
807.  $.ajax({
808.  type:"post",
809.  url:"/upload/image",
810.  data:formData,
811.  cache:false,
812.  processData:false,
813.  contentType:false,
814.  success:this.handleUpdatePicSucc.bind(this)
815.  })
816.  }
817.  handleUpdatePicSucc(data){
818.  if(data.data.code == 1){
819.  $(".userInfo_update>img").attr("src",data.data.url);
820.  }
```

```
821.    }
822.    update(){
823.    $("#userInfo").on("submit",this.handleUpdateCb.bind(this))
824.    }
825.    handleUpdateCb(e){
826.    e.preventDefault();
827.    $.ajax({
828.    type:"post",
829.    url:"/users/updateInfo",
830.    data:{
831.    Nickname: $("#username").val(),
832.    userPic:$(".userInfo_update>img").attr("src"),
833.    id:this.userInfo._id
834.    },
835.    success:this.handleUpdateInfoSucc.bind(this)
836.    })
837.    }
838.    handleUpdateInfoSucc(data){
839.    if(data.data.code == 1){
840.    var obj = {};
841.    obj.Nickname = data.data.Nickname;
842.    obj.userPic = data.data.userPic;
843.    obj._id = data.data.id;
844.
845.    window.sessionStorage.setItem("userinfo",JSON.stringify(obj))
846.    }
847.    }
848.
849.  }
850.
851.  Mine.template = `
852.    <div class="mine">
853.    <div class="content">
854.    <form id="userInfo">
855.    <div class="form-group">
856.    <label for="username">用户昵称</label>
857.    <input type="text" class="form-control" id="username" placeholder="请输入新的昵称">
858.    </div>
859.    <div class="form-group">
860.    <label for="exampleInputPassword1">修改头像</label>
861.    <div class="userInfo_update">
```

```
<img src=""/>
<input type="file" class="form-control" id="userPic" >
</div>
</div>
<button type="submit" class="btn btn-primary update">修改</button>
</form>
</div>
</div>
`
class UsersAction {
constructor() {
this.container = $(".list-content");
this.userListData = {
pageSize: 5,
page: 1
}
}
init() {
this.createContent();
}
createContent() {
this.container.html(UsersAction.template);
this.getUserList();
this.userSearch();
this.userStatusSearch();
this.userSearchBtn();
}
getUserList() {
$.ajax({
type: "get",
url: "/users/userList",
data: this.userListData,
success: this.handleGetUserListSucc.bind(this)
})
}
handleGetUserListSucc(data) {

var dataList = data.data.data;
var str = "";
for (var i = 0; i < dataList.length; i++) {
str += `
```

```
<tr>
<td>${dataList[i].username}</td>
<td>${dataList[i].Nickname}</td>
<td><img src="${dataList[i].userPic}"/></td>
<td>${ moment(dataList[i].registerDate).format("YYYY-MM-DD hh:mm:ss")}</td>
<td>${dataList[i].userStatus?'开启':'关闭'}</td>
<td data-id="${dataList[i]._id}">
<button type="button" class="btn btn-link btn-off">关闭</button>
<button type="button" class="btn btn-link btn-on">开启</button>
</td>
</tr>
`
}

$(".usersAction tbody").html(str);
this.userOff();
}
userOff(){
$(".btn-off").each(this.handleUserOffEach.bind(this))
}
handleUserOffEach(index){

$(".btn-off").eq(index).on("click",this.handleBtnOffCb.bind(this,index))
}
handleBtnOffCb(index){

var id = $(".btn-off").eq(index).parent().attr("data-id");
var actionId = JSON.parse(window.sessionStorage.getItem("userinfo"))._id
$.ajax({
type:"post",
url:"/users/toggleStatus",
data:{
userId:id,
actionId
},
success:this.handleBtnOffSucc.bind(this)
})
}
handleBtnOffSucc(data){
if(data.data.code == 1){
this.getUserList()
```

```
944.  }else{
945.  window.location.href="http://10.60.15.150:3000/error.html"
946.  }
947.  }
948.  userSearch(){
949.  $("#user_search").on("keydown",this.handleFormDown.bind(this))
950.
951.  }
952.  handleFormDown(e){
953.  if(e.keyCode == 13){
954.
955.  var val = $("#user_search").val();
956.  $.ajax({
957.  type:"get",
958.  url:"/users/searchList",
959.  data:{
960.  nickname:val
961.  },
962.  success:this.handleUserSearchSucc.bind(this)
963.  })
964.
965.  }
966.  }
967.  handleUserSearchSucc(data){
968.  this.handleGetUserListSucc(data);
969.  }
970.  userStatusSearch(){
971.  $("#users_Status").on("change",this.handleUserStatusChange.bind(this))
972.  }
973.  handleUserStatusChange(){
974.  console.log(123)
975.  $.ajax({
976.  type:"get",
977.  url:"/users/userstatus",
978.  data:{
979.  userStatus: $("#users_Status").val()
980.  },
981.  success:this.handleUserStatusSucc.bind(this)
982.  })
983.  }
984.  handleUserStatusSucc(data){
```

```
984.   handleUserStatusSucc(data){
985.   this.handleGetUserListSucc(data);
986.   }
987.   userSearchBtn(){
988.   $("#user_search-btn").on("click",this.handleSearchBtn.bind(this))
989.   }
990.   handleSearchBtn(){
991.   var nickname = $("#user_search").val();
992.   var userStatus = $("#users_Status").val();
993.
994.   $.ajax({
995.   type:"get",
996.   url:"/users/search",
997.   data:{
998.   nickname,
999.   userStatus
1000.  },
1001.  success:this.handleBtnSucc.bind(this)
1002.  })
1003.  }
1004.  handleBtnSucc(data){
1005.  this.handleGetUserListSucc(data);
1006.  }
1007. }
1008. UsersAction.template = `
1009.  <div class="usersAction">
1010.  <div class="form-search form-inline">
1011.  <div class="form-group">
1012.  <label for="user_search">用户昵称</label>
1013.  <input type="text" class="form-control" id="user_search" placeholder="请输入要搜索的用
     户昵称">
1014.  </div>
1015.  <div class="form-group">
1016.  <label for="users_Status">用户状态</label>
1017.  <select class="form-control" id="users_Status">
1018.  <option value="">全部</option>
1019.  <option value="1">开启</option>
1020.  <option value="2">关闭</option>
1021.  </select>
1022.  </div>
1023.  <button type="button" class="btn btn-primary" id="user_search-btn">搜索</button>
```

```
</div>
<table class="table table-striped">
<thead>
<tr>
<th>用户账号</th>
<th>用户昵称</th>
<th>用户头像</th>
<th>注册时间</th>
<th>登录状态</th>
<th>操作</th>
</tr>
</thead>
<tbody>
<tr>
<td>alley90088</td>
<td>辰东</td>
<td><img src="../../../img/1578736782091-u=3508817781,1654561115&fm=58&s=6D83639716736DB3087070600300E070.jpg"/></td>
<td>2019:11:11</td>
<td>开启</td>
<td>
<button type="button" class="btn btn-link btn-off">关闭</button>
<button type="button" class="btn btn-link btn-on">开启</button>
</td>
</tr>
</tbody>
</table>
<div id="pageList"></div>
</div>
`


class SliderBar {
constructor() {
this.TabBar = $(".list-sliderBar>ul>li");
}
init() {
this.TabBarToggle()
this.handleTabBarCb(6);
this.setUsername();
```

```
1064.   this.logout();
1065.   }
1066.   TabBarToggle() {
1067.   this.TabBar.each(this.handleTabBarEach.bind(this))
1068.   }
1069.   handleTabBarEach(index) {
1070.   this.TabBar.eq(index).on("click", this.handleTabBarCb.bind(this, index))
1071.   }
1072.   handleTabBarCb(index) {
1073.   this.TabBar.eq(index).addClass("active").siblings().removeClass("active");
1074.
1075.   switch (index) {
1076.   case 0:
1077.   new Home().init();
1078.   break;
1079.   case 1:
1080.   new AddBooks().init();
1081.   break;
1082.   case 2:
1083.   new BooksAction().init();
1084.   break;
1085.   case 3:
1086.   new UsersAction().init();
1087.   break;
1088.   case 4:
1089.   new ArticleAction().init();
1090.   break;
1091.   case 5:
1092.   new Mine().init();
1093.   case 6:
1094.   new Comment().init();
1095.   break;
1096.
1097.   }
1098.   }
1099.   setUsername(){
1100.   let userInfo = JSON.parse(window.sessionStorage.getItem("userinfo"))
1101.   $(".nav-username>span").eq(0).text(userInfo.Nickname)
1102.   }
1103.   logout(){
1104.   $("#logout").on("click",this.handleLogout.bind(this))
```

```
1105.   }
1106.   handleLogout(){
1107.   $.ajax({
1108.   type:"get",
1109.   url:"/users/logout",
1110.   success:this.handleLogoutSucc.bind(this)
1111.   })
1112.   }
1113.   handleLogoutSucc(data){
1114.   if(data.data.code == 1){
1115.   window.sessionStorage.removeItem("userinfo");
1116.   window.location.href="http://10.60.15.150:3000"
1117.   }
1118.   }
1119. }
1120.
1121. new SliderBar().init();
1122.
1123.
1124.
1125. class Login {
1126.   constructor(container) {
1127.   this.container = container;
1128.   }
1129.   init() {
1130.   this.createContent();
1131.   }
1132.   createContent() {
1133.   this.container.html(Login.template);
1134.   this.toggleContent();
1135.   this.createCaptch();
1136.   this.userLogin();
1137.   }
1138.   toggleContent() {
1139.   $("#user_login .text-success").on("click", this.handleToggleContentCb.bind(this))
1140.   }
1141.   handleToggleContentCb() {
1142.   new Page().createContent(false)
1143.   }
1144.   createCaptch() {
1145.   $.ajax({
```

```
$.ajax({
type: "get",
url: "/users/captch",
success: this.handleCreateCaptchSucc.bind(this)
})
}
handleCreateCaptchSucc(data) {
$(".captch>svg").remove();
$(".captch").append(data.data);
this.randomCaptch();
}
randomCaptch() {
$(".captch>svg").on("click", this.handlerandomCaptchCb.bind(this))
}
handlerandomCaptchCb() {
this.createCaptch();
}
userLogin(){
$("#user_login").on("submit",this.handleUserLoginCb.bind(this))
}
handleUserLoginCb(e){
e.preventDefault();
var username = $("#user_login_username").val();
var password = $("#user_login_password").val();
var captch = $("#user_login_captch").val();

$.ajax({
type:"post",
url:"/users/login",
data:{
username,
password,
captch
},
success:this.handleLoginSucc.bind(this)
})

}
handleLoginSucc(data){
if(data.data.code === 1){
var obj = {};
```

```
1186.   obj.userPic = data.data.userPic;
1187.   obj._id = data.data._id;
1188.   obj.Nickname = data.data.Nickname;
1189.   window.sessionStorage.setItem("userinfo",JSON.stringify(obj));
1190.   window.location.href="http://10.60.15.150:3000/html/list.html";
1191.
1192.   }else{
1193.   alert(data.data.info);
1194.   window.location.reload();
1195.   }
1196.   }
1197.  }
1198.
1199.  Login.template = `
1200.  <div id="container">
1201.   <div class="logo">
1202.   <img src="https://cas.1000phone.net/cas/images/login/logo.png">
1203.   </div>
1204.   <form id="user_login">
1205.   <div class="form-group">
1206.   <label for="user_login_username">用户名</label>
1207.   <input type="text" class="form-control" id="user_login_username" placeholder="请输入用
      户名">
1208.   </div>
1209.   <div class="form-group">
1210.   <label for="user_login_password">密码</label>
1211.   <input type="password" class="form-control" id="user_login_password"
      placeholder="Password">
1212.   </div>
1213.   <div class="form-group">
1214.   <label for="user_login_password">验证码</label>
1215.   <div class="captch">
1216.   <input type="text" class="form-control" id="user_login_captch" placeholder="验证码">
1217.   </div>
1218.   </div>
1219.   <p class="text-success">没有账号?立即注册</p>
1220.   <button type="submit" class="btn btn-primary user_btn">登录</button>
1221.   </form>
1222.   </div>
1223.  `
1224.
```

```
1225.
1226. class Page{
1227.  constructor(){
1228.  this.user = $("#user");
1229.  }
1230.  init(){
1231.  this.createContent(true);
1232.  }
1233.  createContent(flag){
1234.  if(flag){
1235.  this.login = new Login(this.user).init();
1236.
1237.  }else{
1238.  this.register = new Register(this.user).init();
1239.  }
1240.  }
1241. }
1242.
1243. new Page().init();
1244.
1245.
1246. class Register {
1247.  constructor(container) {
1248.  this.container = container;
1249.  }
1250.  init() {
1251.  this.createContent();
1252.  }
1253.  createContent() {
1254.  this.container.html(Register.template);
1255.  this.toggleContent();
1256.  this.createCaptch();
1257.  this.userRegister();
1258.  }
1259.  toggleContent(){
1260.  $("#user_register .text-success").on("click",this.handleToggleContentCb.bind(this))
1261.  }
1262.  handleToggleContentCb(){
1263.  new Page().createContent(true);
1264.  }
1265.  createCaptch(){
```

```
$.ajax({
type:"get",
url:"/users/captch",
success:this.handleCreateCaptchSucc.bind(this)
})
}
handleCreateCaptchSucc(data){
$(".captch>svg").remove();
$(".captch").append(data.data);
this.randomCaptch();
}
randomCaptch(){
$(".captch>svg").on("click",this.handlerandomCaptchCb.bind(this))
}
handlerandomCaptchCb(){
this.createCaptch();
}
userRegister(){
$("#user_register").on("submit",this.handleUserRegisterCb.bind(this))
}
handleUserRegisterCb(e){
e.preventDefault();
var username = $("#user_register_username").val();
var password = $("#user_register_password").val();
var captch = $("#user_register_captch").val();

$.ajax({
type:"post",
url:"/users/register",
data:{
username,
password,
captch
},
success:this.handleRgisterSucc.bind(this)
})
}
handleRgisterSucc(data){
if(data.data.code == 1){
alert("注册成功");
new Page().createContent(false);
```

```
  }else{
  alert(data.data.info);
  window.location.reload();
  }
  }
 }

 Register.template = `
 <div id="container">
  <div class="logo">
  <img src="https://cas.1000phone.net/cas/images/login/logo.png">
  </div>
  <form id="user_register">
  <div class="form-group">
  <label for="user_register_username">用户名</label>
  <input type="text" class="form-control" id="user_register_username" placeholder="请输入用户名">
  </div>
  <div class="form-group">
  <label for="user_register_password">密码</label>
  <input type="password" class="form-control" id="user_register_password" placeholder="Password">
  </div>
  <div class="form-group">
  <label for="user_register_password">验证码</label>
  <div class="captch">
  <input type="text" class="form-control" id="user_register_captch" placeholder="验证码">
  </div>
  </div>
  <p class="text-success">已有账号,立即登录</p>
  <button type="submit" class="btn btn-primary user_btn">注册</button>
  </form>
  </div>
  `
```

扩展内网穿透

natApp详解

> *1、https://natapp.cn［官网］*
>
> *2、首先在本站注册账号 点击注册*
>
> *3、登录后,购买隧道,免费/付费均可*

4、根据需要选择隧道协议,这里以web演示,购买隧道

5、本机建立web服务,如 nginx/apache/iis 等,默认80端口

6、在 natapp.cn 根据您的本机下载对应的客户端,比如我的本机是win10,64位,则下载Windows 64位的客户端

7、下载之后,解压至任意目录,得到natapp.exe

8、**取得authtoken** 在网站后台,我的隧道处,可以看到刚才购买的隧道

9、**运行natapp**

详情请参考:https://natapp.cn/article/natapp_newbie