# Nodejs简介

·Node.js 就是运行在服务端的 JavaScript

·Node.js 是一个基于Chrome JavaScript 运行时建立的一个平台

·Node.js是一个事件驱动I/O服务端JavaScript环境，基于Google的V8引擎，V8引擎执行Javascript的速度非常快，性能非常好

·Node.js最擅长高并发

# Node.js的安装

下载node.js

<https://nodejs.org/zh-cn/>

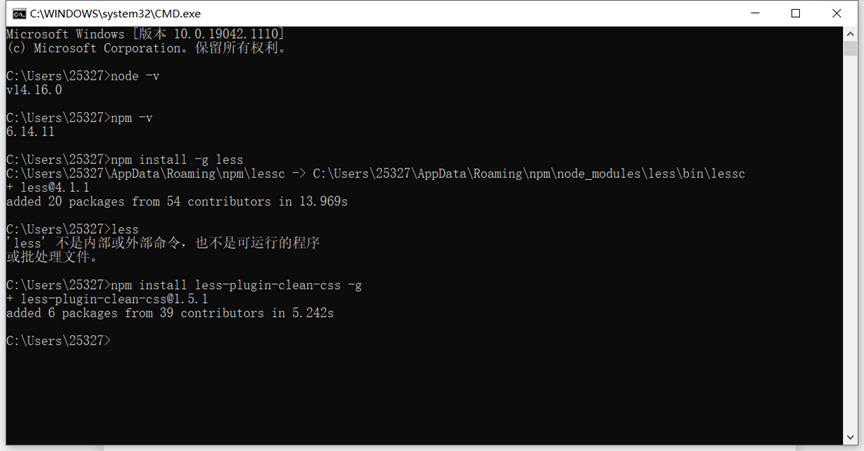
安装

<https://segmentfault.com/a/1190000023390756>

·查看是否安装成功

- node -v 查看 node 版本

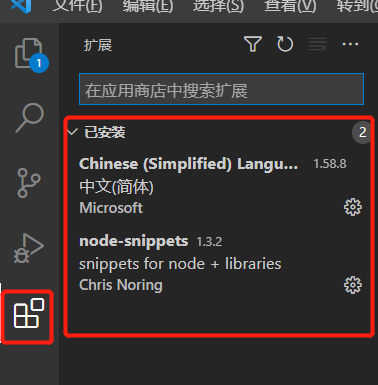
- npm -v 查看 npm 版本



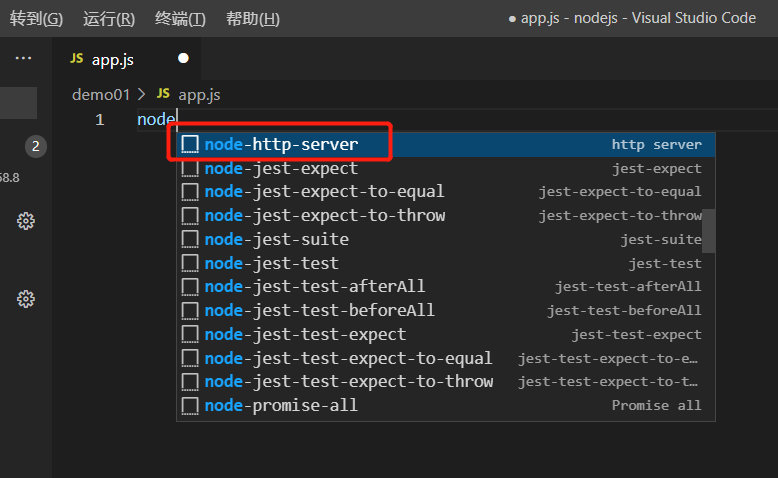
# vscode与使用

·https://code.visualstudio.com/Download

·安装这两个插件



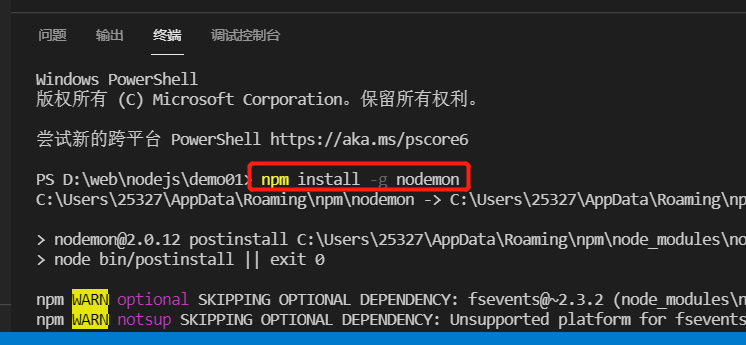
·快捷编辑



# 安装与使用nodemon

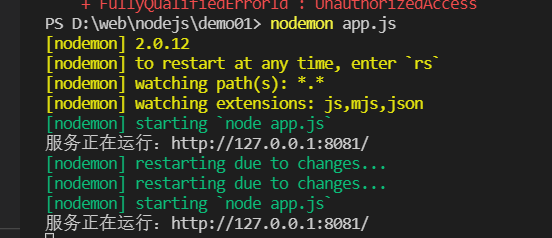
·node自启动工具

·npm install -g nodemon



·使用nodemon运行服务

- nodemon xxx.js

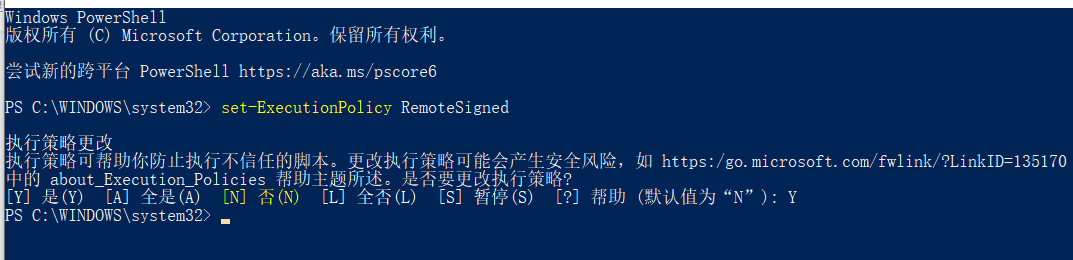


- 如果运行报错

-- win + X管理员打开powerShell

-- 输入set-ExecutionPolicy RemoteSigned

-- 输入Y



# 第一个node服务

## http模块

//引入http模块

const http = require('http');

//创建一个web服务

//request是客户端传递过来的url信息

//response给浏览器响应信息

http.createServer((require,response)=>{

    console.log(require.url);//获取url

    //设置响应头

    //第一个参数是状态码

    //类型是html

    //字符集是utf-8

    response.writeHead(200, {'Content-Type': 'text/html;charset="utf-8"'});

    // //给页面输出一句话，并结束响应头

    // response.end('hello');

    //向页面输出一句话

    response.write("<h2>你好，不会乱码</h2>");

    //结束响应

    response.end();

}).listen(8081);//8081是端口号

console.log('服务正在运行：http://127.0.0.1:8081/');

## url模块

//创建url模块

const url = require("url");

var api = "http://www.itying.com?name=zhangsan&age=20";

//解析url地址并打印,true是将query(url的参数)的值转换为对象

// console.log(url.parse(api,true));

//获取url的参数对象

var getValue = url.parse(api,true).query;

console.log(getValue);//{ name: 'zhangsan', age: '20' }

console.log(`姓名：${getValue.name},年龄：${getValue.age}`);//姓名：zhangsan,年龄：20

## 两者结合

//引入http模块

const http = require('http');

//引入url模块

const url = require("url");

//创建一个web服务

//request是客户端传递过来的url信息

//response给浏览器响应信息

http.createServer((require,response)=>{

    //url:http://127.0.0.1:8081?name=zhangsan&age=20

    // console.log(require.url);//获取客户端传过来的Url

    if(require.url!=="/favicon.ico"){//如果url不为/favicon.ico

        //获取url的参数对象

        var getValue = url.parse(require.url,true).query;

        console.log(`姓名：${getValue.name},年龄：${getValue.age}`);//姓名：zhangsan,年龄：20

    }

    //获取url传过来的name和age

    //设置响应头

    //第一个参数是状态码

    //类型是html

    //字符集是utf-8

    response.writeHead(200, {'Content-Type': 'text/html;charset="utf-8"'});

    // //给页面输出一句话，并结束响应头

    // response.end('hello');

    //向页面输出一句话

    response.write("<h2>你好，不会乱码</h2>");

    //结束响应

    response.end();

}).listen(8081);//8081是端口号

console.log('服务正在运行：http://127.0.0.1:8081/');

# 模块化

## CommonJs

·为了弥补当前 JavaScript 没有标准库的缺陷。提供一个类似 Python，Ruby 和 Java 语言的标准库

·CommonJS 就是模块化的标准，nodejs 就是 CommonJS（模块化）的实现

·把公共的功能抽离成为一个单独的 js 文件作为一个模块

·默认情况下面这个模块里面的方法或者属性，外面是没法访问的

·要让外部可以访问模块里面的方法或者属性，就必须在模块里面通过 exports 或者 module.exports 暴露属性或者方法

·在需要使用这些模块的文件中，通过 require 的方式引入这个模块。这个时候就可以使用模块里面暴露的属性和方法

## Nodejs 中的模块化

·核心模块

- Node 提供的模块

- 如：HTTP 模块 、URL 模块、Fs 模块

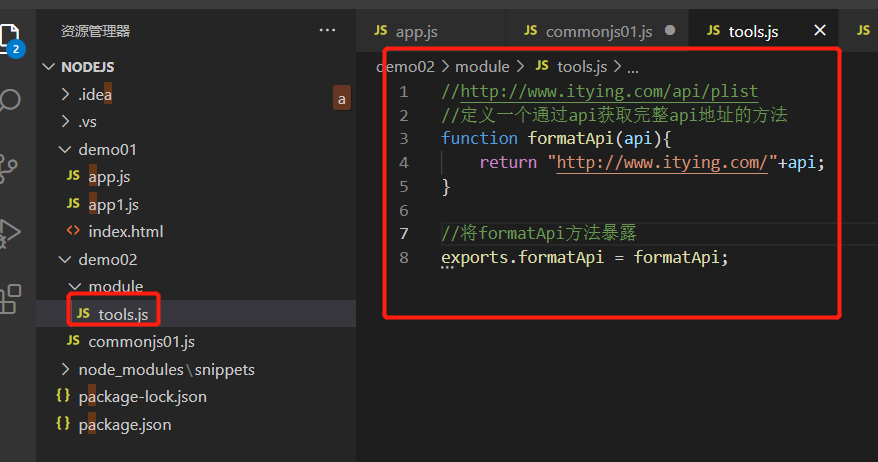
·文件模块

- 用户编写的模块

- 需要我们自己定义

### 通过expirts暴露模块

·定义一个模块



·引入模块

//引入http模块

const http = require('http');

//引入tools.js模块

 const tools = require("./module/tools.js");

 //查看tools模块暴露了哪些方法

//  console.log(tools);//{ formatApi: [Function: formatApi] }

//创建一个web服务

//request是客户端传递过来的url信息

//response给浏览器响应信息

http.createServer((require,response)=>{

    //设置响应头

    //第一个参数是状态码

    //类型是html

    //字符集是utf-8

    response.writeHead(200, {'Content-Type': 'text/html;charset="utf-8"'});

    // //给页面输出一句话，并结束响应头

    // response.end('hello');

    //向页面输出一句话

    response.write("<h2>你好，不会乱码</h2>");

    //调用tools模块的方法获取完整api

    var api = tools.formatApi("api/plist");

    response.write(api);//http://www.itying.com/api/plist

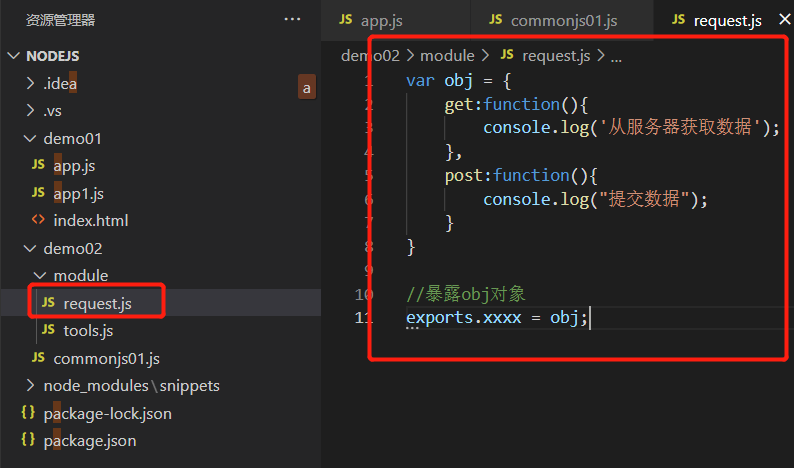
    //结束响应

    response.end();

}).listen(8081);//8081是端口号

console.log('服务正在运行：http://127.0.0.1:8081/');

·定义request.js模块



·引入request.js模块

//引入request.js模块

const request = require("./module/request");

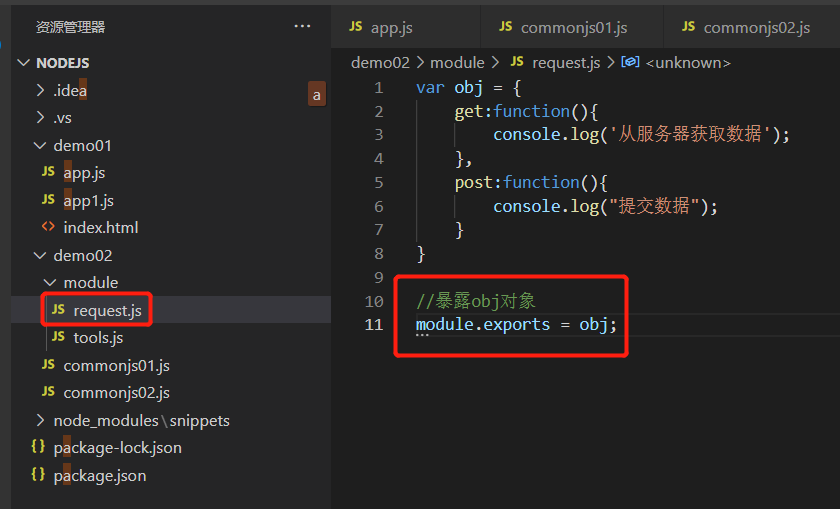
console.log(request);//{ xxxx: { get: [Function: get], post: [Function: post] } }

request.xxxx.get();//调用get方法

request.xxxx.post();//调用post方法

### 通过module.exports暴露模块

·定义一个模块



·引入模块

//引入request.js模块

const request = require("./module/request");

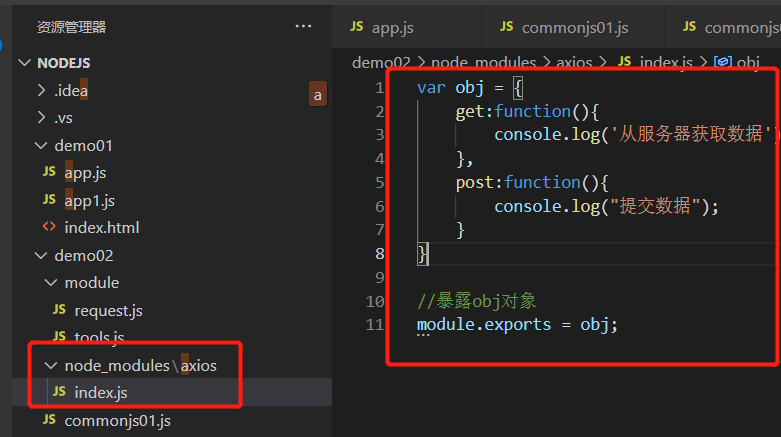
console.log(request);//{ get: [Function: get], post: [Function: post] }

request.get();//调用get方法

request.post();//调用post方法

### node\_modules引入方式

·在node\_modules文件夹中定义一个模块



·引入模块

//引入request.js模块

// const request = require("axios/index");

// console.log(request);//{ get: [Function: get], post: [Function: post] }

// request.get();//调用get方法

// request.post();//调用post方法

//也可以这样引入

//引入request.js模块

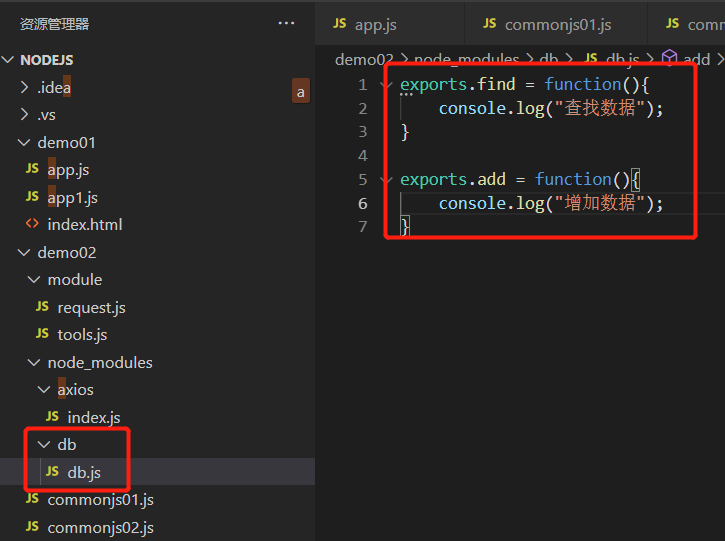
const request = require("axios");

console.log(request);//{ get: [Function: get], post: [Function: post] }

request.get();//调用get方法

request.post();//调用post方法

·在node\_modules文件夹下新建一个模块

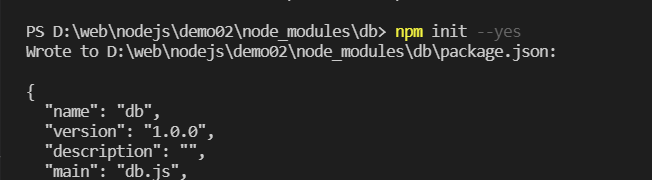


·尝试引入

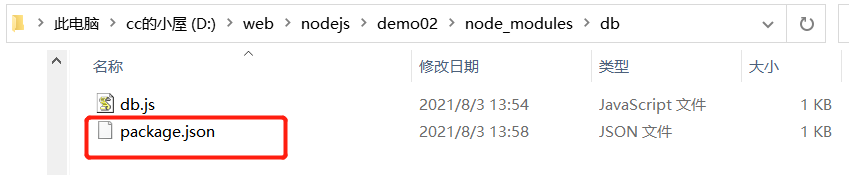
// const db = request("db");//第一次尝试引入报错，因为nodejs默认会找该文件夹下的index.js文件

·初始化设置

- npm init --yes



·会自动生成json配置文件



·再次尝试引入

// const db = request("db");//第一次尝试引入报错，因为nodejs默认会找该文件夹下的index.js文件

const db = require("db");//再次尝试引入不报错，因为生成了配置文件

console.log(db);//{ find: [Function (anonymous)], add: [Function (anonymous)] }

db.find();//查找数据

db.add();//增加数据

# 包与 NPM

## 包

·第三方模块

·也属于自定义模块

·块由包组成，可以通过包来对一组具有相互依赖关系的模块进行统一管理

·完全符合 CommonJs 规范的包目录一般包含如下这些文件

- package.json

-- 包描述文件

- bin

-- 用于存放可执行二进制文件的目录

- lib

-- 用于存放 JavaScript 代码的目录

- doc

-- 用于存放文档的目录。

·在 NodeJs 中通过 NPM 命令来下载第三方的模块（包）

## NPM

·npm 是世界上最大的开放源代码的生态系统

·npm 是随同 NodeJS 一起安装的包管理工具

·https://www.npmjs.com/

### NPM命令

·npm -v

- 查看 npm 版本

·npm init --yes

- 安装第三方包或引入模块前初始化

·npm install xxx

- 安装xxx模块

- 安装时建议在后边加上--save，可以在配置文件的dependencies中查看已安装模块，虽然不加也可以

·npm uninstall xxx

- 卸载xxx模块

·npm list

- 查看已安装的node包

·npm info xxx

- 查看xxx模块的版本

·npm install [jquery@1.8.0](mailto:jquery@1.8.0)

- 指定版本安装

·可使用淘宝镜像安装

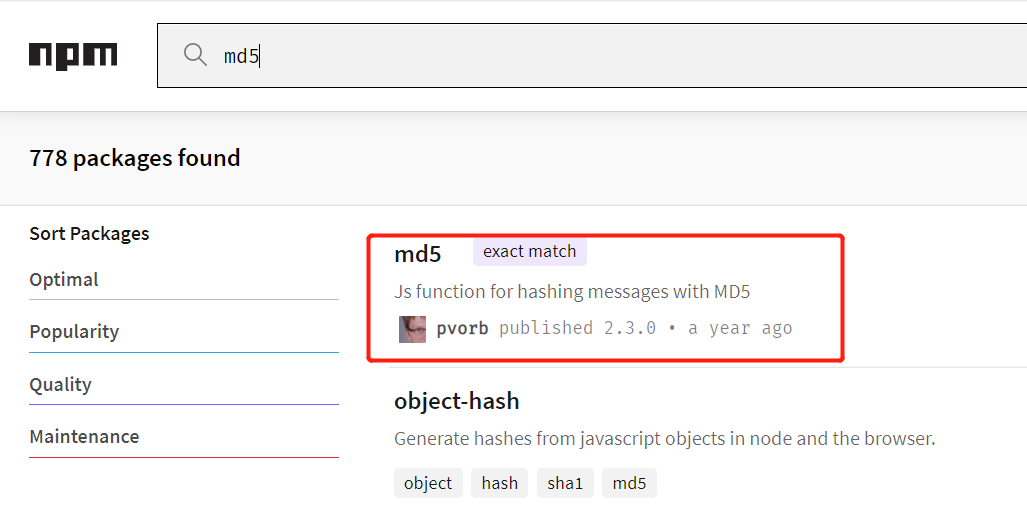
- npm --registry https://registry.npm.taobao.org install xxx

·npm i

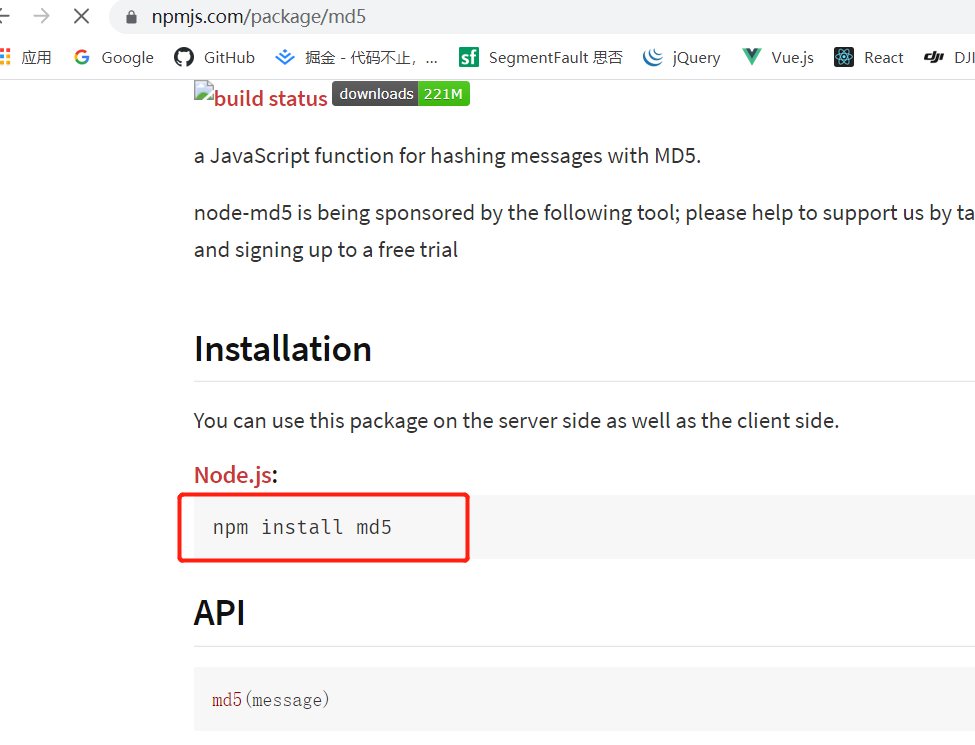
- 安装配置文件中所需要的所以模块

### 如果想要使用md5加密

·在官网中搜索md5



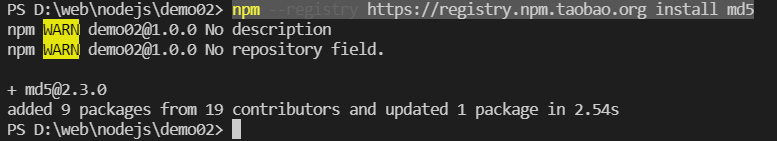
·使用命令安装



·初始化，并使用淘宝镜像安装

- npm init --yes

- npm --registry https://registry.npm.taobao.org install md5



·使用md5模块

//引入md5模块

const md5 = require("md5");

//使用md5对数据进行加密

let password = md5("123456qwe");

console.log(password);//e9f5c5240c0bb39488e6dbfbdb1517e0

### package.json

·nodejs的配置文件

·创建 package.json

- npm init –yes

·安装模块并把模块写入 package.json(依赖)

- npm install 模块 --save

- npm install 模块 --save-dev

- 最新版npm不写入--save和--save-dev也可以写入

- 使用cnpm安装时需要--save才能写入

·dependencies 与 devDependencies 之间的区别

- 使用 npm install node\_module –save 自动更新 dependencies 字段值

- 使用 npm install node\_module –save-dev 自动更新 devDependencies 字段值

- dependencie

-- 配置当前程序所依赖的其他包

- devDependencie

-- 配置当前程序所依赖的其他包，比如一些工具之类的配置在这里

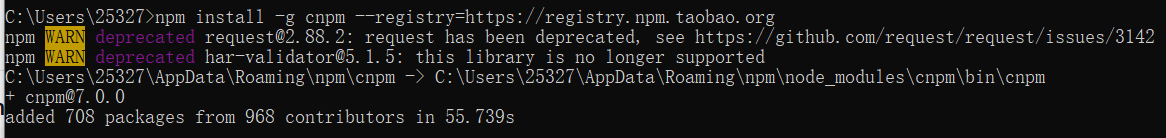
- ^表示第一位版本号不变，后面两位取最新的

- ~表示前两位不变，最后一个取最新

- \*表示全部取最新

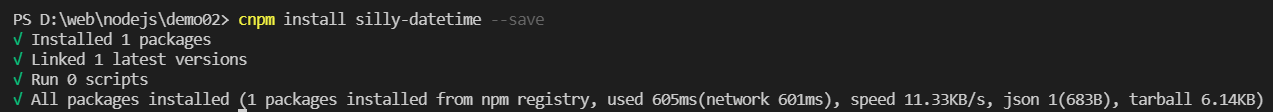
### 安装淘宝镜像

·npm install -g cnpm --registry=https://registry.npm.taobao.org



·使用

- cnpm install xxx --save



# fs模块

//引入fs模块

const fs=require('fs');

·fs.stat

- 检测是文件还是目录

// 1. fs.stat   检测是文件还是目录

    fs.stat('./html',(err,data)=>{

         if(err){

            console.log(err);

            return;

        }

        //isFile()判断是否是文件

        console.log(`是文件:${data.isFile()}`);//是文件:false

        //isDirectory()判断是否是目录

        console.log(`是目录:${data.isDirectory()}`);//是目录:true

    })

·fs.mkdir

- 创建目录

- 如果目录不存在则创建成功

- 如果目录已存在则抛出错误

//2、fs.mkdir  创建目录

    /\*

    path            将创建的目录路径

    mode            目录权限（读写权限），默认777

    callback        回调，传递异常参数err

    \*/

    fs.mkdir('./css',(err)=>{

        if(err){

            console.log(err);

            return;

        }

        console.log('创建成功');

    })

·fs.writeFile

- 创建并写入文件

- 如果文件已存在，则会替换文件

// 3. fs.writeFile  创建写入文件

    /\*

        filename (String)  文件名称

        data   (String|Buffer) 将要写入的内容，可以使字符串 或 buffer数据。

        options  (Object)    option数组对象，包含：

        ·encoding (string)    可选值，默认 ‘utf8′，当data使buffer时，该值应该为 ignored。

        · mode  (Number)    文件读写权限，默认值 438

        · flag  (String)    默认值 ‘w'

        callback {Function}   回调，传递一个异常参数err。

    \*/

    fs.writeFile('./html/index.html','你好nodejs',(err)=>{

        if(err){

            console.log(err);

            return;

        }

        console.log('创建写入文件成功');

    })

·fs.appendFile

- 给指定文件追加数据

- 如果文件不存在会创建并写入

// 4. fs.appendFile 给指定文件追加数据

fs.appendFile('./css/base.css','body{color:red}',(err)=>{

    if(err){

        console.log(err);

        return;

    }

    console.log('appendFile 成功');

})

·fs.readFile

- 读取文件

// 5.fs.readFile 读取文件

fs.readFile('./html/index.html',(err,data)=>{

    if(err){

        console.log(err);

        return;

    }

    console.log(data);

    console.log(data.toString());  //把Buffer 转化成string类型

});

·fs.readdir

- 读取目录

- 返回的data为数组

//6.fs.readdir读取目录   [ 'index.html', 'js', 'news.html' ]

fs.readdir('./html',(err,data)=>{

    if(err){

        console.log(err);

        return;

    }

    console.log(data);//[ 'index.css', 'index.html', 'js', 'news.html' ]

})

·fs.rename

- 重命名

// 7.fs.rename 重命名   功能:1、表示重命名 2、移动文件

//将./css/aaa.css重命名为./css/index.css

fs.rename('./css/aaa.css','./css/index.css',(err)=>{

    if(err){

        console.log(err);

        return;

    }

    console.log('重命名成功');

})

- 移动文件

//将./css/index.css移动到./html/index.css

fs.rename('./css/index.css','./html/index.css',(err)=>{

    if(err){

        console.log(err);

        return;

    }

    console.log('移动文件成功');

})

- 可以同时移动和重命名

·fs.rmdir

- 删除目录

- 只能删除空目录

//  8. fs.rmdir  删除目录

fs.rmdir('./aaaa',(err)=>{

    if(err){

        console.log(err);

        return;

    }

    console.log('删除目录成功');

})

·fs.unlink

- 删除文件

//9. fs.unlink 删除文件

fs.unlink('./aaaa/index.html',(err)=>{

    if(err){

        console.log(err);

        return;

    }

    console.log('删除文件成功');

})

·判断服务器上面有没有upload目录。如果没有创建这个目录，如果有的话不做操作

- /tmp/foo/bar/baz层级创建目录可使用mkdirp模块

-- 到官网搜索mkdirp

-- 安装并使用

// 判断服务器上面有没有upload目录。如果没有创建这个目录，如果有的话不做操作。   （图片上传）

//引入fs模块

const fs=require('fs');

//要检测的目录路径

var path='./upload';

//判断upload是否存在

fs.stat(path,(err,data)=>{

    if(err){//如果不存在

        //执行创建目录

        mkdir(path);

        return;

    }

    if(!data.isDirectory()){//如果存在，但不是目录

         //首先删除文件，再去执行创建目录

         fs.unlink(path,(err)=>{

            if(!err){

                //创建目录

                mkdir(path);

            }else{

                console.log('请检测传入的数据是否正确');

            }

        })

    }

})

//创建目录的方法

function mkdir(dir){

    fs.mkdir(dir,(err)=>{

        if(err){

            console.log(err);

            return;

        }

    });

}

·wwwroot文件夹下面有images css js 以及index.html , 找出 wwwroot目录下面的所有的目录，然后放在一个数组中

//练习：wwwroot文件夹下面有images css js 以及index.html , 找出 wwwroot目录下面的所有的目录，然后放在一个数组中

const fs=require('fs');

//错误的写法  注意:fs里面的方法是异步

/\*

    var path='./wwwroot';

    var dirArr=[];

    fs.readdir(path,(err,data)=>{

        if(err){

            console.log(err);

            return;

        }

        for(var i=0;i<data.length;i++){

            fs.stat(path+'/'+data[i],(error,stats)=>{

                if(stats.isDirectory()){

                    dirArr.push(data[i]);

                }

            })

        }

        console.log(dirArr);

    })

    console.log(dirArr);

\*/

//打印出 3个3

    /\*

        for(var i=0;i<3;i++){

            setTimeout(function(){

                console.log(i);

            },100)

        }

    \*/

//1、改造for循环  递归实现      2、nodejs里面的新特性  async await

var path='./wwwroot';

var dirArr=[];

fs.readdir(path,(err,data)=>{

    if(err){

        console.log(err);

        return;

    }

    (function getDir(i){

        if(i==data.length){ //执行完成

            console.log(dirArr);

            return;

        }

        fs.stat(path+'/'+data[i],(error,stats)=>{

            if(stats.isDirectory()){

                dirArr.push(data[i]);

            }

            getDir(i+1)

        })

    })(0)

})

# async和await处理异步操作

//练习：wwwroot文件夹下面有images css js 以及index.html , 找出 wwwroot目录下面的所有的目录，然后放在一个数组中

const fs = require('fs');

//1、定义一个isDir的方法判断一个资源到底是目录还是文件

async function isDir(path) {

    //返回一个promise方法

    return new Promise((resolve,reject) => {

        //检测是文件还是目录

        fs.stat(path, (error, stats) => {

            if (error) {

                console.log(error);

                reject(error)

                return;

            }

            //如果是目录

            if (stats.isDirectory()) {

                //修改promise的状态为成功

                resolve(true);

            } else {

                resolve(false);

            }

        })

    })

}

//2、获取wwwroot里面的所有资源  循环遍历

function main(){

    var path='./wwwroot'

    var dirArr=[];

    //读取目录

    fs.readdir(path,async (err,data)=>{  //注意

        if(err){

            console.log(err);

            return;

        }

        for(var i=0;i<data.length;i++){

            if(await isDir(path+'/'+data[i])){

                dirArr.push(data[i]);

            }

        }

        console.log(dirArr);

    })

}

main();

·fs.createReadStream

- 从文件流中读取数据

//引入fs模块

const fs=require('fs');

//创建读取流

var readStream=fs.createReadStream('./data/input.txt');

var count=0;//保存读取的次数

var str='';//保存数据的字符串

//监听读取状态，如果监听到data表示读取到了数据

readStream.on('data',(data)=>{//以流的方式读取是一点一点读取的

    str+=data;

    count++;//读取次数+1

})

//监听读取状态，如果监听到end表示读取完了

readStream.on('end',()=>{

    console.log(str);//输出读取到的数据

    console.log(count);//输出一共读取的次数

})

//监听读取状态，如果监听到error表示读取出现错误

readStream.on('error',(err)=>{

    console.log(err);

})

·fs.createWriteStream

- 写入文件

const fs=require('fs');//引入fs模块

var str = '';//要写入的数据

for(var i=0;i<500;i++){

    str+='我是从数据库获取的数据，我要保存起来'+i+"\r\n";

}

//创建写入流，传入要写入的文件

var writeStream=fs.createWriteStream('./data/output.txt');

//将数据写入到文件

writeStream.write(str);

//标记文件末尾

writeStream.end();

//监听写入状态，如果监听到finish表示写入完了

writeStream.on('finish',()=>{

    console.log('写入完成');

});

·管道流

- 管道提供了一个输出流到输入流的机制。通常我们用于从一个流中获取数据并将数据传递到另外一个流中

·使用管道流复制文件

//使用管道流复制图片

//引入fs模块

const fs=require('fs');

//创建读取流读取要复制的图片地址

var readStream=fs.createReadStream('./aaa.jpg');

//创建写入流写入要复制到的图片地址

var writeStream=fs.createWriteStream('./data/aaa.jpg');

//以流的方式把读取到的图片写入到另一个文件

readStream.pipe(writeStream);

//使用管道流复制文件

//创建读取流读取要复制的文件地址

var readStream=fs.createReadStream('./demo.zip');

//创建写入流写入要复制到的文件地址

var writeStream=fs.createWriteStream('./data/demo.zip');

//以流的方式把读取到的文件写入到另一个文件

readStream.pipe(writeStream);

# 创建静态 WEB 服务器

·问题

- html以外的url无法解析，如css和js

-- 通过path模块获取后缀名，并返回文件类型解决

- json的url后面带参数无法解析

-- 通过url模块解析url

- 图片和json等文件类型不正确

-- 以上问题均可引入mime.json解决

-- <https://blog.csdn.net/qq_33174548/article/details/104254575>

·可自动下载static目录下的zip文件

- <http://127.0.0.1:8081/xxx.zip>

·common.js

//获取解析文本类型的方法

//传入的参数是后缀名

// exports.getMime =function(extname){

//     switch(extname){

//         case ".html":

//             return "text/html";

//         case ".css":

//             return "text/css";

//         case ".js":

//             return "text/javascript";

//         default:

//             return "text/html";

//     }

// }

// const { rejects } = require("assert");

// //引入fs模块

// const fs = require("fs");

// exports.getFileMime =function(extname){

//    //读取mime.json文件，路径相对于引入模块的js文件路径

//     fs.readFile("./data/mime.json",(err,data)=>{

//         if(err){

//             console.log(err);

//             return;

//         }

//         //查看是否读取成功

//         // console.log(data.toString());

//         //将json数据转成js数据

//         let mime = JSON.parse(data.toString());

//         //获取对应后缀名的文件类型并返回

//         // console.log(mime[extname]);

//         // return mime[extname];//fs模块的方法是异步方法，不能之间返回数据，需要封装成promise对象

//     })

// }

//引入fs模块

// const fs = require("fs");

// const { resolve } = require("path");

// exports.getFileMime =function(extname){

//     //返回peomise对象

//    return new Promise((resolve,rejects)=>{

//         //读取mime.json文件的方法，路径相对于引入模块的js文件路径

//         fs.readFile("./data/mime.json",(err,data)=>{

//             if(err){

//                 rejects(err);

//             }

//             //查看是否读取成功

//             // console.log(data.toString());

//             //将json数据转成js数据

//             let mime = JSON.parse(data.toString());

//             //获取对应后缀名的文件类型并返回

//             resolve(mime[extname]);

//         })

//    })

// }

//引入fs模块

const fs = require("fs");

exports.getFileMime =function(extname){

    //readFileSync同步读取数据

    //解决异步调用读取mime.json文件的方法，路径相对于引入模块的js文件路径

    let data = fs.readFileSync("./data/mime.json");

    //将json数据转成js数据

    let mime = JSON.parse(data.toString());

    //获取对应后缀名的文件类型并返回

    return mime[extname];

}

·app.js

//引入http模块

const http = require('http');

//引入fs模块

const fs = require('fs');

//引入common.js模块

const common = require("./module/common.js");

//引入path模块

const path = require("path");

//引入url模块

const url = require("url");

//创建http服务

http.createServer(function (request, response) {

    //获取url地址，并解析url

    let pathname = url.parse(request.url).pathname;

    //如果pathname为/就让它变成/index.html

    pathname = pathname==="/"?"/index.html":pathname;

    //获取pathnmae的后缀名

    let extname = path.extname(pathname);

    if(pathname!=="/favicon.ico"){//如果url不是favicon.ico

        //通过fs模块读取文件

        fs.readFile("./static"+pathname,(err,data)=>{

            if(err){//如果是错误地址

                response.writeHead(404, {'Content-Type': 'text/html;charset="utf-8"'});

                //结束响应并输出文字

                response.end('404');

            }

            //如果正确地址

            //获取解析文本类型

            let mime =common.getFileMime(extname);

            //设置响应头

            response.writeHead(200, {'Content-Type': `${mime};charset="utf-8"`});

            //结束响应并输出文字

            response.end(data);

        });

    }

}).listen(8081);

console.log('Server running at http://127.0.0.1:8081/');

## nodejs封装静态web服务

·routes.js

//引入fs模块

const fs = require('fs');

//引入path模块

const path = require("path");

//引入url模块

const url = require("url");

//私有方法

let getFileMime =function(extname){

    //readFileSync同步读取数据

    //解决异步调用读取mime.json文件的方法，路径相对于引入模块的js文件路径

    let data = fs.readFileSync("./data/mime.json");

    //将json数据转成js数据

    let mime = JSON.parse(data.toString());

    //获取对应后缀名的文件类型并返回

    return mime[extname];

}

//封装静态web服务的方法

//staticPath为静态资源目录

exports.static =function(request,response,staticPath){

    //获取url地址，并解析url

    let pathname = url.parse(request.url).pathname;

    //如果pathname为/就让它变成/index.html

    pathname = pathname==="/"?"/index.html":pathname;

    //获取pathnmae的后缀名

    let extname = path.extname(pathname);

    if(pathname!=="/favicon.ico"){//如果url不是favicon.ico

        //通过fs模块读取文件

        fs.readFile(staticPath+pathname,(err,data)=>{

            if(err){//如果是错误地址

                let errhtml = fs.readFileSync("./static/404.html");

                response.writeHead(404, {'Content-Type': 'text/html;charset="utf-8"'});

                //结束响应并输出文字

                response.end(errhtml);

            }

            //如果正确地址

            //获取解析文本类型，getFileMime(extname)返回的是promise对象，需要await把异步改成同步

            let mime = getFileMime(extname);

            //设置响应头

            response.writeHead(200, {'Content-Type': `${mime};charset="utf-8"`});

            //结束响应并输出文字

            response.end(data);

        });

    }

}

·app.js

//引入http模块

const http = require('http');

//引入routes.js模块

const routes = require("./module/routes.js");

//创建http服务

http.createServer(function (request, response) {

    //创建静态web服务

    routes.static(request,response,"./static");

}).listen(8081);

console.log('Server running at http://127.0.0.1:8081/');

# 路由

·路由指的就是针对不同请求的 URL，处理不同的业务逻辑。

·routes.js

//引入fs模块

const fs = require('fs');

//引入path模块

const path = require("path");

//引入url模块

const url = require("url");

//私有方法

let getFileMime =function(extname){

    //readFileSync同步读取数据

    //解决异步调用读取mime.json文件的方法，路径相对于引入模块的js文件路径

    let data = fs.readFileSync("./data/mime.json");

    //将json数据转成js数据

    let mime = JSON.parse(data.toString());

    //获取对应后缀名的文件类型并返回

    return mime[extname];

}

//封装静态web服务的方法

//staticPath为静态资源目录

exports.static =function(request,response,staticPath){

    //获取url地址，并解析url

    let pathname = url.parse(request.url).pathname;

    //如果pathname为/就让它变成/index.html

    pathname = pathname==="/"?"/index.html":pathname;

    //获取pathnmae的后缀名

    let extname = path.extname(pathname);

    if(pathname!=="/favicon.ico"){//如果url不是favicon.ico

        try{

            //通过fs模块读取文件

            let data = fs.readFileSync(staticPath+pathname);

            //如果正确地址

             if(data){

                //获取解析文本类型，getFileMime(extname)返回的是promise对象，需要await把异步改成同步

                let mime = getFileMime(extname);

                //设置响应头

                response.writeHead(200, {'Content-Type': `${mime};charset="utf-8"`});

                //结束响应并输出文字

                response.end(data);

            }

        }catch(error){

        }

    }

}

·app.js

//引入http模块

const http = require('http');

//引入routes.js模块

const routes = require("./module/routes.js");

//引入url模块

const url = require("url");

//创建http服务

http.createServer(function (request, response) {

    //创建静态web服务

    routes.static(request,response,"./static");

    //获取url地址，并解析url

    let pathname = url.parse(request.url).pathname;

    if(pathname === "/login"){

        //设置响应头

        response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

        response.end("login");

    }else if(pathname === "/register"){

        //设置响应头

        response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

        response.end("register");

    }else if(pathname === "/admin"){

        //设置响应头

        response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

        response.end("处理后端业务逻辑");

    }else if(pathname === "/register"){

        //设置响应头

        response.writeHead(404, {'Content-Type': `text/html;charset="utf-8"`});

        response.end("页面不存在");

    }

}).listen(8081);

console.log('Server running at http://127.0.0.1:8081/');

# EJS模板引擎

·把数据库和文件读取的数据显示到 Html 页面上面

·安装

- npm install ejs –save

·login.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>这是一个登录页面</h1>

    <!--输出后端传递过来的数据-->

    <h2><%=msg%></h2>

    <ul>

        <!--将后端数据遍历到li中-->

        <%for(let i=0;i<list.length;i++){%>

            <li><%=list[i].title%></li>

        <%}%>

    </ul>

</body>

</html>

·app.js

//引入http模块

const http = require('http');

//引入routes.js模块

const routes = require("./module/routes.js");

//引入url模块

const url = require("url");

//引入ejs

const ejs = require("ejs");

//创建http服务

http.createServer(function (request, response) {

    //创建静态web服务

    routes.static(request,response,"./static");

    //获取url地址，并解析url

    let pathname = url.parse(request.url).pathname;

    if(pathname === "/login"){

        let msg = "这是数据库里面获取的数据";

        let list = [

            {title:"新闻1"},

            {title:"新闻2"},

            {title:"新闻3"},

            {title:"新闻4"}

        ]

        //使用ejs引擎渲染页面，将数据传递给前端页面

        //第二个参数是要传入的数据对象

        //第三个参数可以省略

        ejs.renderFile("./views/login.ejs",{

            msg:msg,

            list:list

        },(err,data)=>{

            //设置响应头

            response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

            //将数据返回给页面

            response.end(data);

        });

    }else if(pathname === "/register"){

        //设置响应头

        response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

        response.end("register");

    }else if(pathname === "/admin"){

        //设置响应头

        response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

        response.end("处理后端业务逻辑");

    }else if(pathname === "/register"){

        //设置响应头

        response.writeHead(404, {'Content-Type': `text/html;charset="utf-8"`});

        response.end("页面不存在");

    }

}).listen(8081);

console.log('Server running at http://127.0.0.1:8081/');

# Get和Post

·GET

- 从指定的资源请求数据。（一般用于获取数据）

·POST

- 向指定的资源提交要被处理的数据。（一般用于提交数据）

·form.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <form action="/doadmin" method="POST">

        用户名：<input type="text" name="username">

        <br>

        <br>

        密码：<input type="password" name="password">

        <br>

        <br>

        <input type="submit" value="提交">

    </form>

</body>

</html>

·app.js

//引入http模块

const http = require('http');

//引入routes.js模块

const routes = require("./module/routes.js");

//引入url模块

const url = require("url");

//引入ejs

const ejs = require("ejs");

//创建http服务

http.createServer(function (request, response) {

    //创建静态web服务

    routes.static(request,response,"./static");

    //获取url地址，并解析url

    let pathname = url.parse(request.url).pathname;

    //获取请求类型

    console.log(request.method);

    if(pathname === "/login"){

        let msg = "这是数据库里面获取的数据";

        let list = [

            {title:"新闻1"},

            {title:"新闻2"},

            {title:"新闻3"},

            {title:"新闻4"}

        ];

        //使用ejs引擎渲染页面，将数据传递给前端页面

        //第二个参数是要传入的数据对象

        //第三个参数可以省略

        ejs.renderFile("./views/login.ejs",{

            msg:msg,

            list:list

        },(err,data)=>{

            //设置响应头

            response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

            //将数据返回给页面

            response.end(data);

        });

    }else if(pathname === "/register"){

        //设置响应头

        response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

        response.end("register");

    }else if(pathname === "/admin"){

        //post提交数据

        ejs.renderFile("./views/form.ejs",{},(err,data)=>{

            response.end(data);

            //设置响应头

            response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

        });

    }else if(pathname === "/doadmin"){

        //获取post传值

        let postData = "";

        //有数据的时候

        request.on("data",(chunk)=>{

            postData+=chunk;

        })

        //数据传输完毕的时候

        request.on("end",()=>{

            console.log(postData);

            //设置响应头

            response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

            response.end(postData);

        })

    }else if(pathname === "/news"){

        //获取get传值

        var query = url.parse(request.url,true).query;

        console.log(query);

        //设置响应头

        response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

        response.end("get传值获取成功");

    }else{

        //设置响应头

        response.writeHead(404, {'Content-Type': `text/html;charset="utf-8"`});

        response.end("页面不存在");

    }

}).listen(8081);

console.log('Server running at http://127.0.0.1:8081/');

# 路由封装

·routes.js

//引入fs模块

const fs = require('fs');

//引入path模块

const path = require("path");

//引入url模块

const url = require("url");

//引入ejs

const ejs = require("ejs");

//私有方法

let getFileMime =function(extname){

    //readFileSync同步读取数据

    //解决异步调用读取mime.json文件的方法，路径相对于引入模块的js文件路径

    let data = fs.readFileSync("./data/mime.json");

    //将json数据转成js数据

    let mime = JSON.parse(data.toString());

    //获取对应后缀名的文件类型并返回

    return mime[extname];

}

//路由封装

let app = {

    static:(request,response,staticPath)=>{

        //获取url地址，并解析url

        let pathname = url.parse(request.url).pathname;

        //如果pathname为/就让它变成/index.html

        pathname = pathname==="/"?"/index.html":pathname;

        //获取pathnmae的后缀名

        let extname = path.extname(pathname);

        if(pathname!=="/favicon.ico"){//如果url不是favicon.ico

            try{

                //通过fs模块读取文件

                let data = fs.readFileSync("./"+staticPath+pathname);

                //如果正确地址

                 if(data){

                    //获取解析文本类型，getFileMime(extname)返回的是promise对象，需要await把异步改成同步

                    let mime = getFileMime(extname);

                    //设置响应头

                    response.writeHead(200, {'Content-Type': `${mime};charset="utf-8"`});

                    //结束响应并输出文字

                    response.end(data);

                }

            }catch(error){

            }

        }

    },

    login:(request,response)=>{

        let msg = "这是数据库里面获取的数据";

        let list = [

            {title:"新闻1"},

            {title:"新闻2"},

            {title:"新闻3"},

            {title:"新闻4"}

        ]

        //使用ejs引擎渲染页面，将数据传递给前端页面

        //第二个参数是要传入的数据对象

        //第三个参数可以省略

        ejs.renderFile("./views/login.ejs",{

            msg:msg,

            list:list

        },(err,data)=>{

            //设置响应头

            response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

            //将数据返回给页面

            response.end(data);

        });

    },

    register:(request,response)=>{

        //设置响应头

        response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

        response.end("register");

    },

    admin:(request,response)=>{

        //post提交数据

        ejs.renderFile("./views/form.ejs",{},(err,data)=>{

            response.end(data);

            //设置响应头

            response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

        });

    },

    doadmin:(request,response)=>{

        //获取post传值

        let postData = "";

        //有数据的时候

        request.on("data",(chunk)=>{

            postData+=chunk;

        })

        //数据传输完毕的时候

        request.on("end",()=>{

            console.log(postData);

            //设置响应头

            response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

            response.end(postData);

        })

    },

    news:(request,response)=>{

        //获取get传值

        var query = url.parse(request.url,true).query;

        console.log(query);

        //设置响应头

        response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

        response.end("get传值获取成功");

    },

    error:(request,response)=>{

        //设置响应头

        response.writeHead(200, {'Content-Type': `text/html;charset="utf-8"`});

        //找不到页面

        response.end("页面不存在");

    }

}

module.exports = app;

·app.js

//引入http模块

const http = require('http');

//引入routes.js模块

const routes = require("./module/routes.js");

//引入url模块

const url = require("url");

//引入ejs

const ejs = require("ejs");

//引入path模块

const path = require("path");

//创建http服务

http.createServer(function (request, response) {

    //创建静态web服务

    routes.static(request,response,"static");

    //获取url地址，并解析url

    let pathname = url.parse(request.url).pathname.replace("/","");

    let extname = path.extname(pathname);

    // console.log(extname);

    //动态切换路由

    if(!extname && pathname!==""){//如果没有后缀并且不为空

        try{

            //如果存在则加载路由

            routes[pathname](request,response);

        }catch(error){

            //如果不存在则加载404页面

            routes["error"](request,response);

        }

    }

}).listen(8081);

console.log('Server running at http://127.0.0.1:8081/');

# 封装类似express框架的路由

## 封装get方法

·route.js

const url= require("url");

let G={};

let app=function(req,res){

    // console.log('调用app方法')

    //http://127.0.0.1:3000/news

    //http://127.0.0.1:3000/login

    //http://127.0.0.1:3000/register

    //获取路径并解析

    let pathname=url.parse(req.url).pathname;

    if(G[pathname]){//如果存在

        G[pathname](req,res);  //执行方法

    }else{

        res.writeHead(404, { 'Content-Type': 'text/html;charset="utf-8"' });

        res.end('页面不存在');

    }

}

app.get=function(str,cb){

    //注册方法

    G[str]=cb;

}

module.exports=app;

·app.js

const http = require("http");

const app=require('./module/route')

//注册web服务

http.createServer(app).listen(8081);

//配置路由

app.get('/',function(req,res){

    res.writeHead(200, { 'Content-Type': 'text/html;charset="utf-8"' });

    res.end('首页');

})

//配置路由

app.get('/login',function(req,res){

    res.writeHead(200, { 'Content-Type': 'text/html;charset="utf-8"' });

    res.end('执行登录操作');

})

//配置路由

app.get('/news',function(req,res){

    res.writeHead(200, { 'Content-Type': 'text/html;charset="utf-8"' });

    res.end('新闻页面');

})

console.log("服务器运行中");

## 封装post方法

·route.js

const url = require("url");

//扩展send方法的函数

function changeRes(res){

    res.send=(data)=>{

        res.writeHead(200, { 'Content-Type': 'text/html;charset="utf-8"' });

        res.end(data);

    }

}

let server = () => {

    let G = {};

    //把get 和 post分开

    G.\_get={};

    G.\_post={};

    let app = function (req, res) {

        //扩展res的方法

        changeRes(res);

        let pathname = url.parse(req.url).pathname;

        //获取请求类型

        let method=req.method.toLowerCase();

        console.log(method);

        if (G['\_'+method][pathname]) {

            if(method=="get"){

                G['\_'+method][pathname](req, res);  //执行方法

            }else{

                //post  获取post的数据 把它绑定到req.body

                let postData = '';

                req.on('data',(chunk)=>{

                    postData+=chunk;

                })

                req.on('end',()=>{

                   req.body=postData;

                   G['\_'+method][pathname](req, res);  //执行方法

                })

            }

        } else {

            res.writeHead(404, { 'Content-Type': 'text/html;charset="utf-8"' });

            res.end('页面不存在');

        }

    }

    app.get = function (str, cb) {

        //注册方法

        G.\_get[str] = cb;

    }

    app.post = function (str, cb) {

        //注册方法

        G.\_post[str] = cb;

    }

    return app;

}

module.exports = server();

·app.js

const http = require("http");

const app=require('./module/route');

const ejs = require("ejs");

//注册web服务

http.createServer(app).listen(8081);

//配置路由

app.get('/',function(req,res){

    res.send("首页")

})

//配置路由

app.get('/login',function(req,res){

    // res.writeHead(200, { 'Content-Type': 'text/html;charset="utf-8"' });

    // res.end('执行登录操作');

    ejs.renderFile("./views/form.ejs",{},(err,data)=>{

        res.send(data)

    })

})

app.post('/doLogin',function(req,res){

    console.log(req.body);

    res.send(req.body)

})

## 封装静态web服务

·route.js

const fs = require('fs');

const path = require('path');

const url = require('url');

//扩展res

function changeRes(res) {

    res.send = (data) => {

        res.writeHead(200, { 'Content-Type': 'text/html;charset="utf-8"' });

        res.end(data);

    }

}

//根据后缀名获取文件类型

function getFileMime(extname) {

    var data = fs.readFileSync('./data/mime.json'); //同步方法

    let mimeObj = JSON.parse(data.toString());

    return mimeObj[extname];

}

//静态web服务的方法

function initStatic(req, res, staticPath) {

    //1、获取地址

    let pathname = url.parse(req.url).pathname;

    // pathname = pathname == '/' ? '/index.html' : pathname;

    let extname = path.extname(pathname);

    //2、通过fs模块读取文件

    if (extname) {  //如果有后缀名用静态web服务处理

        try {

            let data = fs.readFileSync('./' + staticPath + pathname);

            if (data) {

                let mime = getFileMime(extname);

                res.writeHead(200, { 'Content-Type': '' + mime + ';charset="utf-8"' });

                res.end(data);

            }

        } catch (error) {

            console.log(error);

        }

    }

}

let server = () => {

    let G = {

        \_get: {},

        \_post: {},

        staticPath: 'static' //默认静态web目录

    };

    let app = function (req, res) {

        //扩展res的方法

        changeRes(res);

        //配置静态web服务

        initStatic(req, res, G.staticPath);

        let pathname = url.parse(req.url).pathname;

        //获取请求类型

        let method = req.method.toLowerCase();

        // console.log(method);

        let extname = path.extname(pathname);

        if (!extname) {  //如果有后缀名用静态web处理

            if (G['\_' + method][pathname]) {

                if (method == "get") {

                    G['\_' + method][pathname](req, res);  //执行方法

                } else {

                    //post  获取post的数据 把它绑定到req.body

                    let postData = '';

                    req.on('data', (chunk) => {

                        postData += chunk;

                    })

                    req.on('end', () => {

                        req.body = postData;

                        G['\_' + method][pathname](req, res);  //执行方法

                    })

                }

            } else {

                res.writeHead(404, { 'Content-Type': 'text/html;charset="utf-8"' });

                res.end('页面不存在');

            }

        }

    }

    //get请求

    app.get = function (str, cb) {

        //注册方法

        G.\_get[str] = cb;

    }

    //post请求

    app.post = function (str, cb) {

        //注册方法

        G.\_post[str] = cb;

    }

    //配置静态web服务目录

    app.static = function (staticPath) {

        G.staticPath = staticPath;

    }

    return app;

}

module.exports = server();

·app.js

const http = require("http");

const app=require('./module/route');

const ejs = require("ejs");

//注册web服务

http.createServer(app).listen(8081);

app.static("static");    //修改默认静态web目录

//配置路由

app.get('/',function(req,res){

    res.send("首页")

})

//配置路由

app.get('/login',function(req,res){

    // res.writeHead(200, { 'Content-Type': 'text/html;charset="utf-8"' });

    // res.end('执行登录操作');

    ejs.renderFile("./views/form.ejs",{},(err,data)=>{

        res.send(data)

    })

})

app.post('/doLogin',function(req,res){

    // console.log(req.body);

    res.send(req.body)

})

# MongoDb数据库

·个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的 NoSql 数据库

·NoSQL

- 非关系型的数据库，以 key-value 形式存储

- 没有行、列的概念。用 JSON 来存储数据

- 集合就相当于“表”，文档就相当于“行”

·特点

- 高 性能、易部署、易使用，存储数据非常方便

- 支持的查询语言非常强大

- 几乎可以实现类似关系数据库单表查询的绝大部分功能

## MongoDb 安装

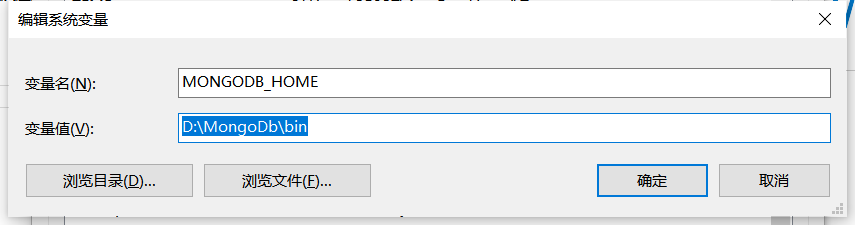
·官网下载

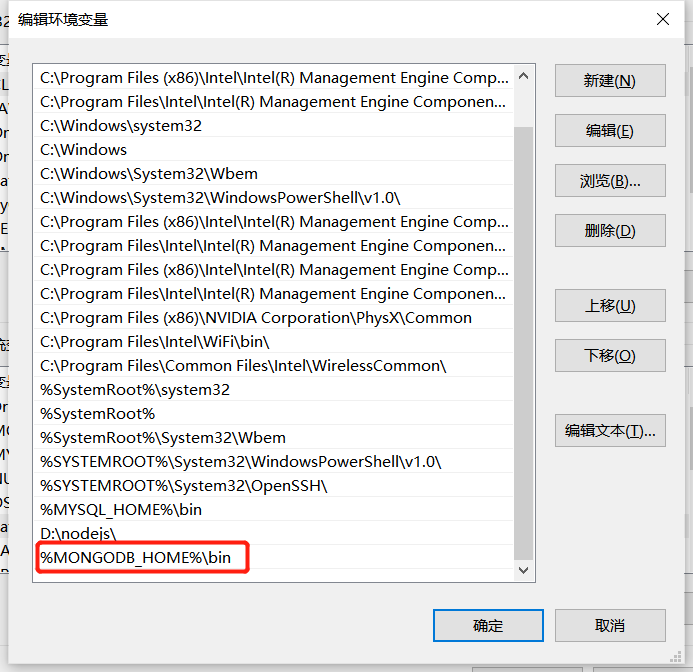
- <https://www.mongodb.com/try/download/community>

·安装教程

- <https://www.runoob.com/mongodb/mongodb-window-install.html>

·配置环境变量





## 使用 MongoDb

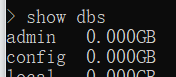
·mongo

- 连接数据库



·show dbs

- 查看数据库列表



·cls

- 清屏

·use 数据库名

- 使用数据库

- 创建数据库

-- 必须插入一个数据才能创建成功



·db.集合名.insert({“数据名”:”数据值”});

- 向表中插入（增加）数据

- 给当前数据库的user集合中插入数据



·show collections

- 显示当前数据库的数据集合列表



·db.集合名.find()

- 查看集合中的所有数据



·db.集合名.drop()

- 删除集合



·db.dropDatabase()

- 删除当前所在的数据库



### 增加数据

·db.集合名.insert({“数据名”:”数据值”})

- 向表中插入（增加）数据

- 给当前数据库的user集合中插入数据



·使用循环语句插入多条数据

for(var i=0;i<100;i++){

db.集合名.insert({“数据名”:”数据值”+i})

}

### 查找数据

·以user集合为例

·db.user.find()

- 查看集合中的所有数据



·db.user.find().count()

- 统计数据

·db.user.find(条件)

- 查找指定数据

- db.user.find({"age": 22})

-- 查询 age = 22 的数据

- db.user.find({age: {$gt: 22}})

-- 查询 age > 22 的数据

- db.user.find({age: {$lt: 22}})

-- 查询 age < 22 的数据

- db.user.find({age: {$gte: 25}})

-- 查询 age >= 25 的记录

- db.user.find({age: {$lte: 25}})

-- 查询 age <= 25 的记录

- db.user.find({age: {$gte: 23, $lte: 26}})

-- 查询 age >= 23 并且 age <= 26

-- 注意书写格式

/条件/正则查找

- db.user.find({name: /mongo/})

-- 查询 name 中包含 mongo 的数据

-- 模糊查询用于搜索

- db.user.find({name: /^mongo/})

-- 查询 name 中以 mongo 开头的数据

- db.user.find({}, {name: 1, age: 1})

-- 查询指定列 name、age 数据

-- 只显示name和age字段

- db.user.find({age: {$gt: 25}}, {name: 1, age: 1})

-- 查询指定列 name、age 数据, age > 25

- db.user.find().sort({age: 1})

-- 按照年龄排序

-- 1 升序 -1 降序

- db.user.find({name: 'zhangsan', age: 22})

-- 查询 name = zhangsan, age = 22 的数据

- db.user.find().limit(5)

-- 查询第5条之前的数据

-- 包含第5条

- db.user.find().skip(10)

-- 查询第10条以后的数据

-- 不包含第10条

- db.user.find().limit(10).skip(5)

-- 查询在（5,10]之间的数据

-- 可用于分页

- db.user.find({$or: [{age: 22}, {age: 25}]})

-- 查询年龄为22或25的数据

- db.user.findOne()

-- 查询第一条数据

- db.user.find({age: {$gte: 25}}).count()

-- 查询某个结果集的记录条数

-- 如果要返回限制之后的记录数量，要使用 count(true)或者 count(非 0) db.users.find().skip(10).limit(5).count(true)

### 修改数据

·以student集合为例

·db.student.update(查找，修改)

- 修改数据

- 为数据新增字段

- 默认只修改查找到的第一条数据

- db.student.update({"name":"小明"},{$set:{"age":16}})

-- 查找name为小明的数据，把age修改为 16

- db.student.update({"score.shuxue":70},{$set:{"age":33}})

-- 查找数学成绩是 70，把年龄更改为 33 岁

- db.student.update({"sex":"男"},{$set:{"age":33}},{multi: true})

- 修改多条数据

- 增加{multi: true}

- db.student.update({"name":"小明"},{"name":"大明","age":16})

- 完整替换

### 删除数据

·以user集合为例

·一定要写删除条件，不然会删除所有数据

·默认删除查到到的所有数据

·db.user.remove( { "name": "张三" } )

- 删除name为张三的数据

·db.user.remove({“age”:{$gt:30}})

- 删除年龄大于30的所有数据

·db.user.remove({“age”:{$gt:30}},{justOne:true}})

- 只删除查找到的第一条数据

- 添加{justOne:true}

## 索引

### 索引基础

·索引是对数据库表中一列或多列的值进行排序的一种结构

·可以让我们查询数据库变得更快

·1是升序，-1是降序

·默认给id设置索引

·db.user.find().explain( "executionStats" )

- 查询具体的时间

·db.user.ensureIndex({"username":1})

- 给username设置索引

·db.user.getIndexes()

- 获取user集合的索引

·db.user.dropIndex({"username":1})

- 给username删除索引

·db.user.ensureIndex({"username":1, "age":-1})

- 创建复合索引

- 基于 username 和 age 的查询将会用到该索引，或者是基于 username 的查询也会用到该索引，但是只是基于 age 的查询将不会用到该复合索引

- 在查询条件中包含复合索引中的前 N 个索引列，才会应用该索引

·db.user.ensureIndex({"username":1},{"name":"userindex"})

- 设置索引并指定索引名

### 唯一索引

·设置唯一索引之后该字段的值不能重复

·如果插入的数据中不包含 userid 键，那么该文档中该键的值为 null

·db.user.ensureIndex({"userid":1},{"unique":true})

- 为userid字段创建唯一索引

## 账户权限配置

·创建超级管理用户

- use admin



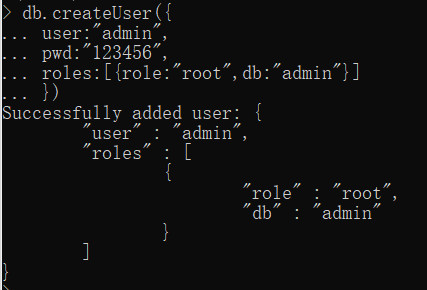
- db.createUser({

user:'admin',

pwd:'123456',

roles:[{role:'root',db:'admin'}]

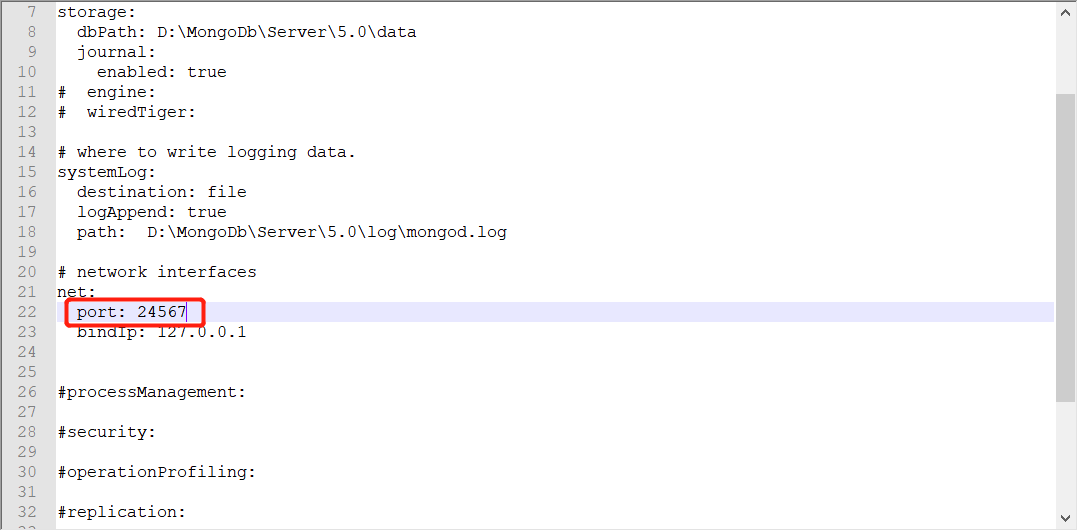
})



·修改 Mongodb 数据库配置文件

- MongoDB\Server\5.0\bin\mongod.cfg

- 修改端口号



·设置security

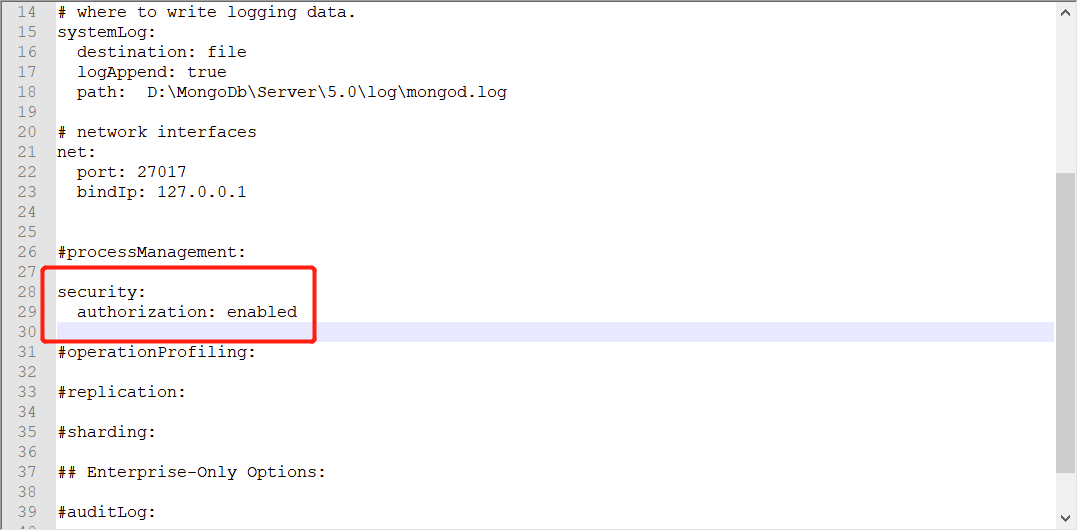
- security:

authorization: enabled

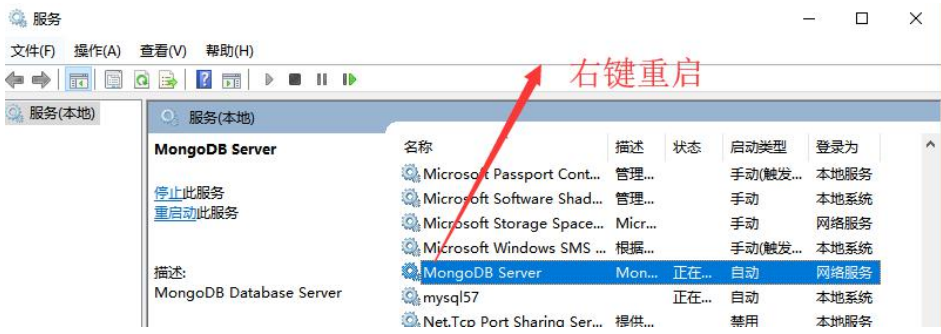
- 不能有多余的空格

- authorization前面空2格

- enabled前面空1格

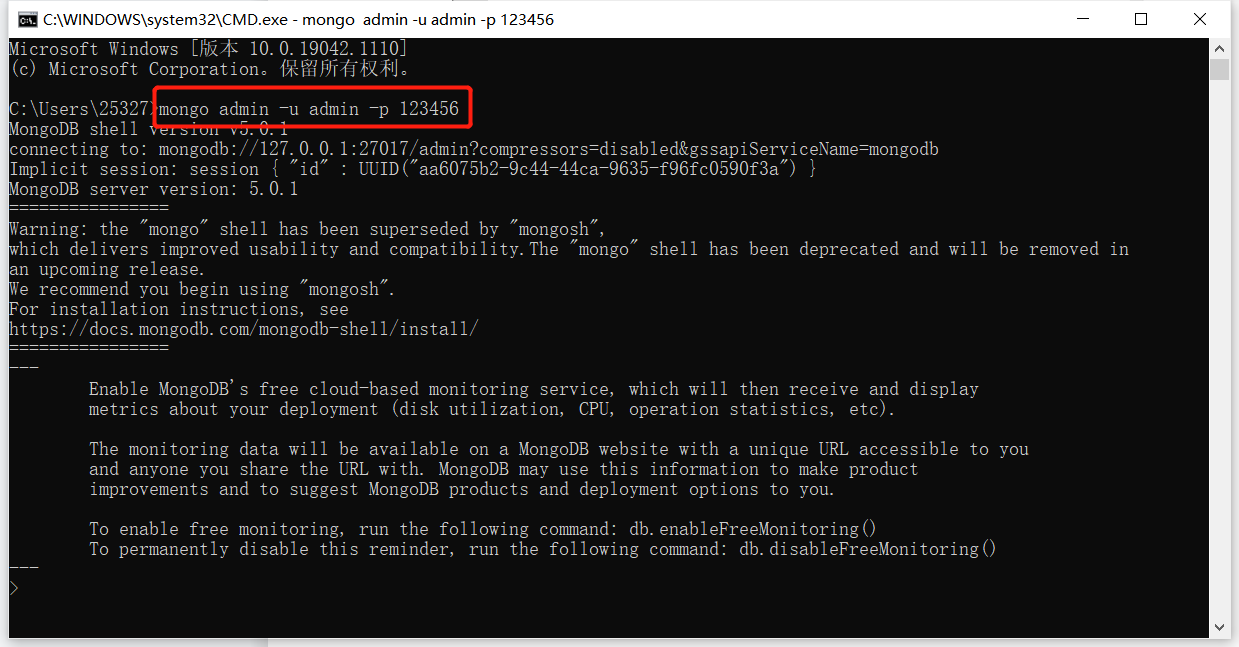


·第三步重启 mongodb 服务



·连接数据库

- 打开cmd，输入：mongo admin -u 用户名 -p 密码



·给user数据库创建一个用户 只能访问user不能访问其他数据库

use user

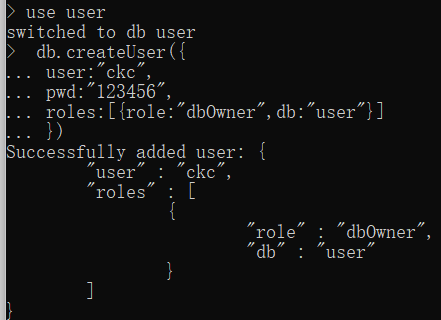
db.createUser({

user: "用户名",

pwd: "密码",

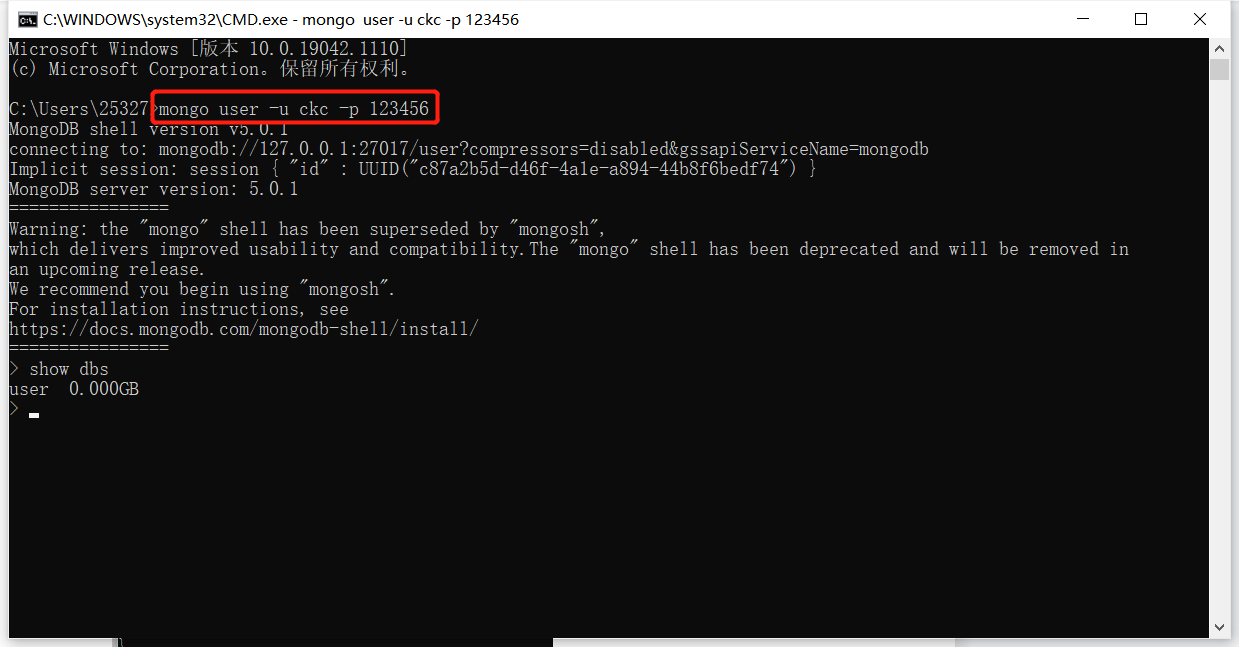
roles: [ { role: "角色", db: "数据库名" } ]

})



·连接数据库

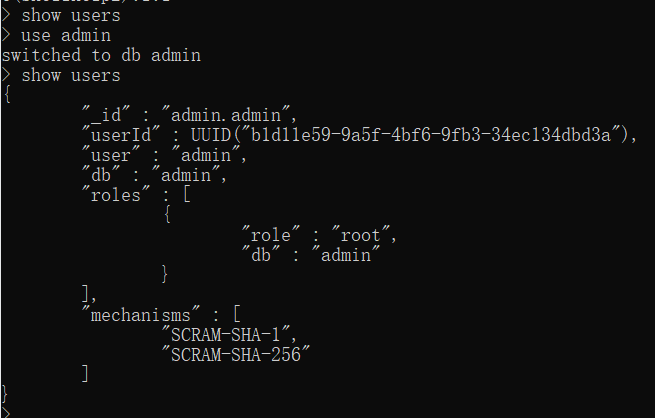
- mongo 数据库名 -u 用户名 -p 密码



## Mongodb 账户权限配置中常用的命令

·show users

- 查看数据库下的当前用户



·db.dropUser("用户名")

- 删除用户



·db.updateUser( "用户名",{pwd:"密码"})

- 修改用户密码

·db.auth("admin","password")

- 密码认证



## Mongodb 数据库角色

·数据库用户角色：read、readWrite;

·数据库管理角色：dbAdmin、dbOwner、userAdmin；

·集群管理角色：clusterAdmin、clusterManager、clusterMonitor、hostManager；

·备份恢复角色：backup、restore

·所有数据库角色：readAnyDatabase、readWriteAnyDatabase、userAdminAnyDatabase、 dbAdminAnyDatabase

·超级用户角色：root

·nodejs连接数据库的时候需要配置账户密码

- const url = 'mongodb://admin:123456@localhost:27017/';

## 关系数据库表与表的 3 种关系

·一对一的关系

- 例如：一个人对应一个唯一的身份证号，即为一对一的关系

·一对多关系

- 例如：一个班级对应多名学生，一个学生只能属于一个班级，即为一对多关系

·多对多关系

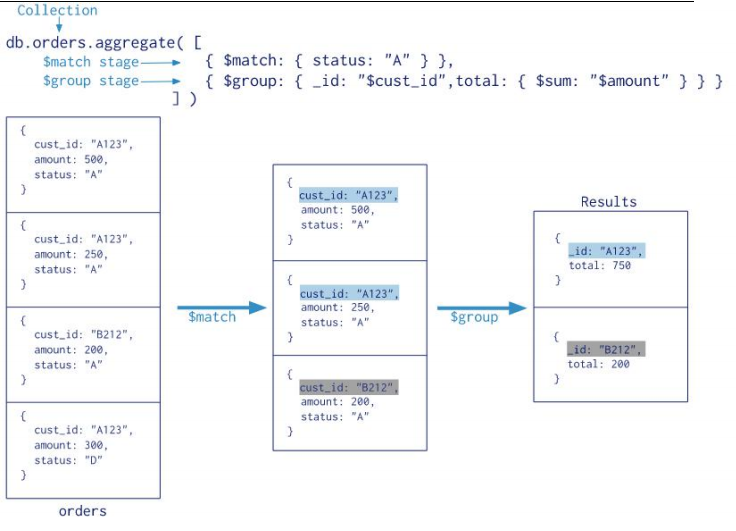
- 例如：一个学生可以选多门课程，而同一门课程可以被多个学生选修，彼此的对应关系 即是多对多关系

## MongoDB 聚合管道

·使用聚合管道可以对集合中的文档进行变换和组合

·实际项目

- 表关联查询、数据的统计



### 数据准备

·订单主表

·订单号、订单id、交易号、总价格、总数量

db.order.insert({"order\_id":"1","uid":10,"trade\_no":"111","all\_price":100,"all\_num":2})

db.order.insert({"order\_id":"2","uid":7,"trade\_no":"222","all\_price":90,"all\_num":2})

db.order.insert({"order\_id":"3","uid":9,"trade\_no":"333","all\_price":20,"all\_num":6})

·订单从属表

·订单id、商品名、价格、数量

db.order\_item.insert({"order\_id":"1","title":"商品鼠标 1","price":50,num:1})

db.order\_item.insert({"order\_id":"1","title":"商品键盘 2","price":50,num:1})

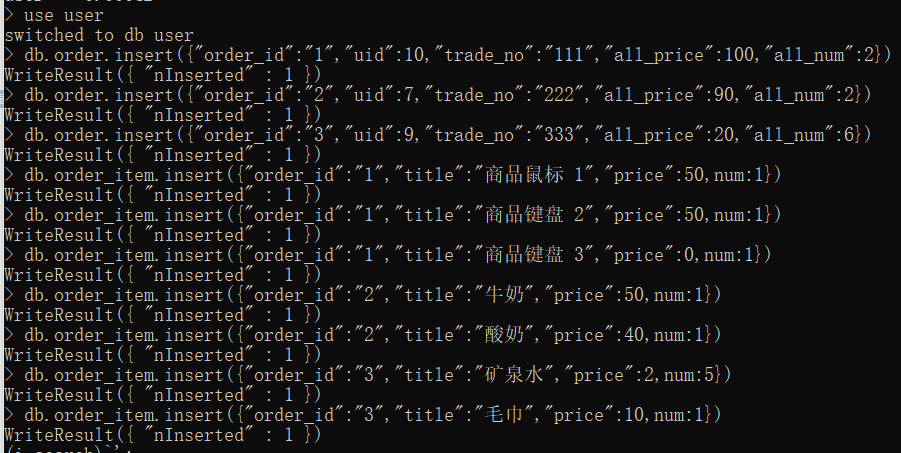
db.order\_item.insert({"order\_id":"1","title":"商品键盘 3","price":0,num:1})

db.order\_item.insert({"order\_id":"2","title":"牛奶","price":50,num:1})

db.order\_item.insert({"order\_id":"2","title":"酸奶","price":40,num:1})

db.order\_item.insert({"order\_id":"3","title":"矿泉水","price":2,num:5})

db.order\_item.insert({"order\_id":"3","title":"毛巾","price":10,num:1})



### 管道操作符

·$project

- 增加、删除、重命名字段

- 要求查找 order 只显示文档中 trade\_no 和 all\_price 字段

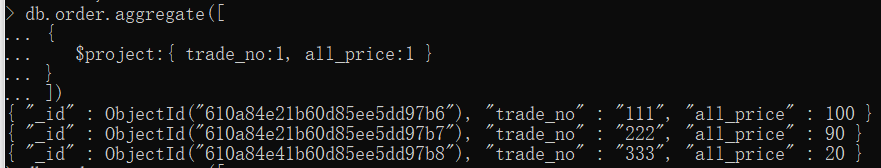
db.order.aggregate([

{

$project:{ trade\_no:1, all\_price:1 }

}

])



·$match

- 条件匹配。只满足条件的文档才能进入下一阶段

- 要求查找 order 只显示文档中 trade\_no 和 all\_price 字段，并且all\_price大于或等于90的字段

db.order.aggregate([

{

$project:{trade\_no:1, all\_price:1 }

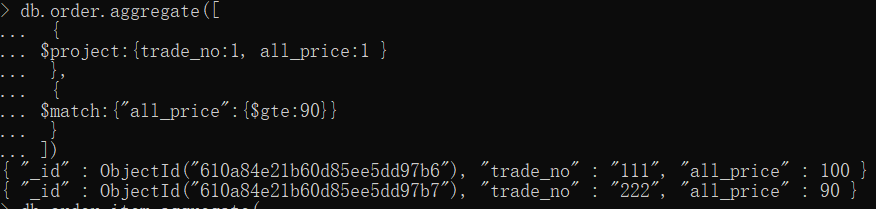
},

{

$match:{"all\_price":{$gte:90}}

}

])



·$limit

- 限制结果的数量

- 只显示查询结果的前2条数据

db.order.aggregate([

{

$project:{ trade\_no:1, all\_price:1

}

},

{

$sort:{"all\_price":-1}

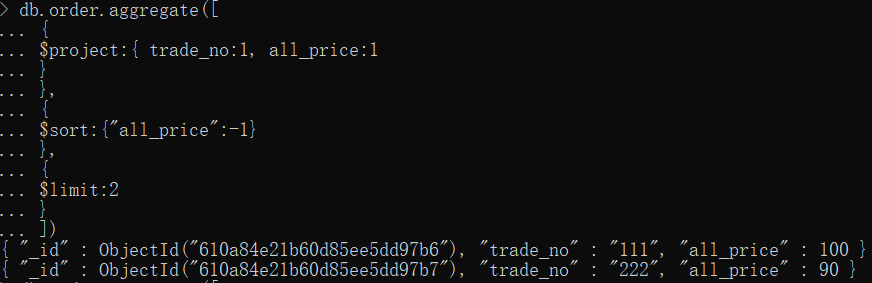
},

{

$limit:2

}

])



·$skip

- 跳过文档的数量

- 跳过查询结果的前2条数据

db.order.aggregate([

{

$project:{ trade\_no:1, all\_price:1

}

},

{

$sort:{"all\_price":-1}

},{

$skip:2

}

])

·$sort

- 条件排序

- 查询结果以all\_price降序排列

db.order.aggregate([

{

$project:{ trade\_no:1, all\_price:1 }

},

{

$match:{"all\_price":{$gte:90}}

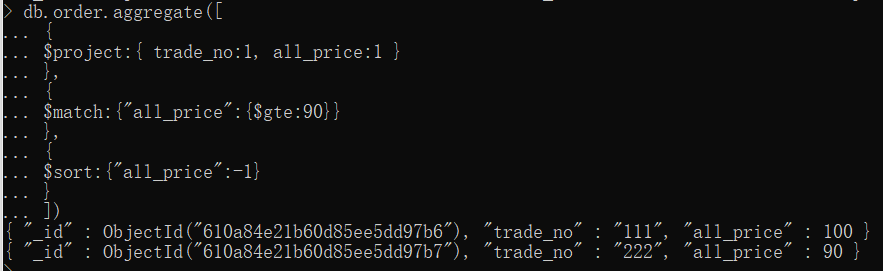
},

{

$sort:{"all\_price":-1}

}

])



·$group

- 条件组合结果、统计

- 统计每个order\_id的订单数量，按照order\_id分组

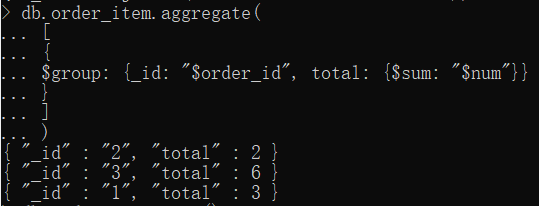
db.order\_item.aggregate([

{

$group: {\_id: "$order\_id", total: {$sum: "$num"}}

}

])



·$lookup

- 引入其它集合的数据 （表关联查询）

- 关联order和order\_item表，order的关联字段为order\_id，order\_item的关联字段为order\_id，把查询到的数据放到items字段下

db.order.aggregate([

{

$lookup:

{

from: "order\_item",

localField: "order\_id",

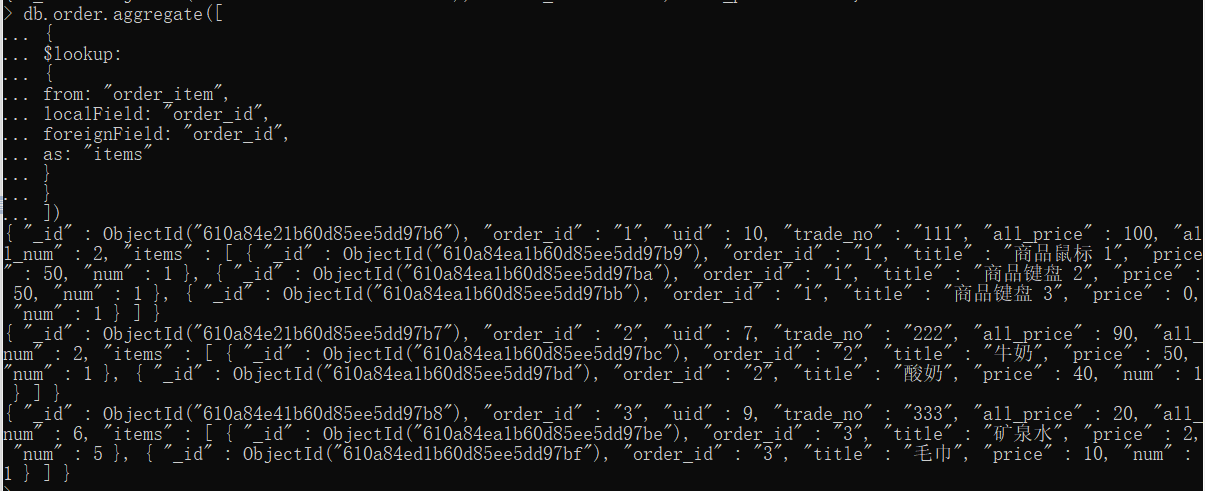
foreignField: "order\_id",

as: "items"

}

}

])



- 使用json格式化工具查看



- 关联order和order\_item表，order的关联字段为order\_id，order\_item的关联字段为order\_id，把查询到的数据放到items字段下，并筛选出all\_price大于等于90的数据

db.order.aggregate([

{

$lookup:{

from: "order\_item",

localField: "order\_id",

foreignField: "order\_id",

as: "items"

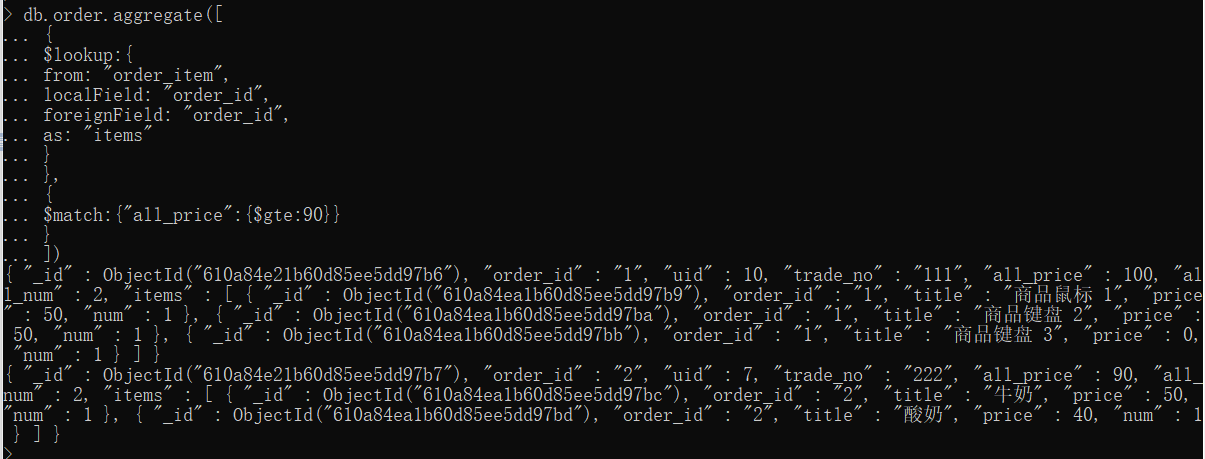
}

},

{

$match:{"all\_price":{$gte:90}}

}

])



### 管道表达式

·管道操作符作为“键”,所对应的“值”叫做管道表达式

- 例如{$match:{status:"A"}}，$match 称为管道操作符，而 status:"A"称为管道表达式， 是管道操作符的操作数(Operand)

·$addToSet

- 将文档指定字段的值去重

·$max

- 文档指定字段的最大值

·$min

- 文档指定字段的最小值

·$sum

- 文档指定字段求和

·$avg

- 文档指定字段求平均

·$gt

- 大于给定值

·$lt

- 小于给定值

·$eq

- 等于给定值

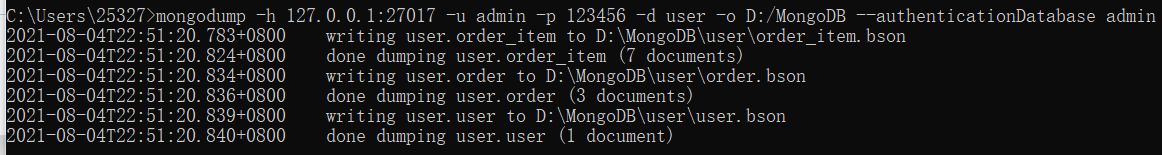
## 数据库备份

·mongodump -h 主机名 -d 数据库名 -o 备份的路径

·mongodump -h 主机名:端口号 -u admin -p 超级管理员密码 -d 数据库名 -o 备份的路径 --authenticationDatabase admin

- 将本机的user数据库备份到D:\MongoDb目录下

-- mongodump -h 127.0.0.1:27017 -u admin -p 123456 -d user -o D:/MongoDB --authenticationDatabase admin

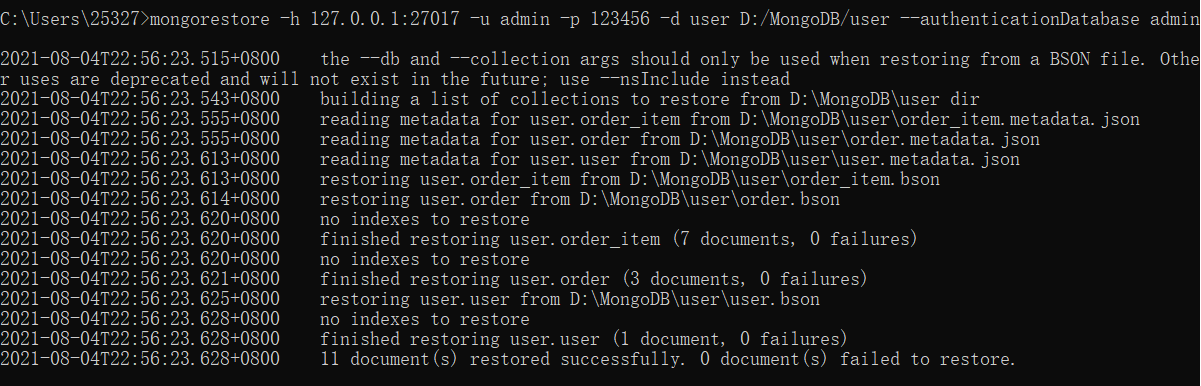


## 数据库恢复

·mongorestore -h 主机名:端口号 -u admin -p 超级管理员密码 -d 数据库名 要还原的数据库路径 --authenticationDatabase admin

- 将user数据库还原到本机

-- mongorestore -h 127.0.0.1:27017 -u admin -p 123456 -d user D:/MongoDB/user --authenticationDatabase admin



# nodejs中操作mongodb数据库

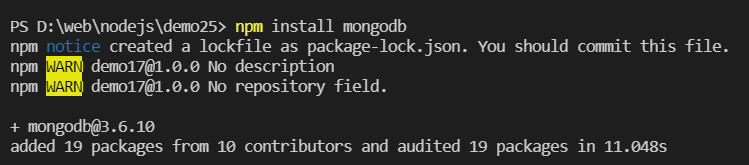
·Nodejs 操作 mongodb 数据库官方文档

·生成配置文件

- npm init --yes

·安装mongodb模块

- npm install mongodb --save



## 连接方式1

// 1 cnpm install mongodb --save

//2、用es6的方式引入mongodb

const { MongoClient } = require('mongodb');

//3、定义数据库连接的地址

// const url = 'mongodb://127.0.0.1:27017';//无密码的连接方法

const url = 'mongodb://admin:123456@localhost:27017/';//有密码的连接方法

//4、定义要操作的数据库

const dbName = 'user';

//5、实例化MongoClient 传入数据库连接地址 配置{ useUnifiedTopology: true }后没有提示，不加也可

const client = new MongoClient(url, { useUnifiedTopology: true });

//6、连接数据库 操作数据

client.connect((err) => {

    if (err) {//数据库连接失败

        console.log(err);

        return;

    }

    console.log("数据库连接成功");

    //切换到要操作的数据库

    let db = client.db(dbName);

    //操作数据库完毕以后一定要 关闭数据库连接

    // client.close();

    //1、查找数据，并将查找到的数据转换成数组

    // db.collection("user").find({"age":13}).toArray((err,data)=>{

    //    if(err){

    //         console.log(err);

    //         return;

    //     }

    //     console.log(data);

    //     //操作数据库完毕以后一定要 关闭数据库连接

    //     client.close();

    // })

    //2、增加数据

    // db.collection("user").insertOne({"username":"nodejs操作mongodb1","age":11},(err,result)=>{

    //     if(err){ //增加失败

    //         console.log(err);

    //         return;

    //     }

    //     console.log("增加成功");

    //     //查看成功后的数据\_id值

    //     console.log(result.insertedId);

    //       //操作数据库完毕以后一定要 关闭数据库连接

    //      client.close();

    // })

    //3、修改数据

    // db.collection("user").updateOne({ "name": "张三0" }, { $set: { "age": 30 } }, (err, result) => {

    //     if (err) { //修改失败

    //         console.log(err);

    //         return;

    //     }

    //     console.log("修改成功");

    //     //修改成功后返回的结果

    //     console.log(result);

    //     //操作数据库完毕以后一定要 关闭数据库连接

    //     client.close();

    // })

    //4、删除一条数据

    // db.collection("user").deleteOne({ "username": "张三" }, (err)=>{

    //     if (err) {

    //         console.log(err);

    //         return;

    //     }

    //     console.log("删除一条数据成功");

    //     client.close();

    // })

     //5、删除多条数据

    db.collection("user").deleteMany({ "age": 5 }, (err)=>{

        if (err) {

            console.log(err);

            return;

        }

        console.log("删除多条数据成功");

        client.close();

    })

})

## 连接方式2

// 1 cnpm install mongodb --save

//2、引入mongodb

const { MongoClient } = require('mongodb');

//3、定义数据库连接的地址

const url = 'mongodb://admin:123456@localhost:27017/';

//4、定义要操作的数据库

const dbName = 'user';

//5、连接数据库 查找

// MongoClient.connect(url, { useUnifiedTopology: true },(err,client)=>{

//     if (err) {

//         console.log(err);

//         return;

//     }

//     console.log("数据库连接成功");

//     let db = client.db(dbName);

//   //1、查找数据

//     db.collection("user").find({"age":11}).toArray((err,data)=>{

//        if(err){

//             console.log(err);

//             return;

//         }

//         console.log(data);

//         //操作数据库完毕以后一定要 关闭数据库连接

//         client.close();

//     })

// })

//6、连接数据库 增加

MongoClient.connect(url, { useUnifiedTopology: true },(err,client)=>{

    if (err) {

        console.log(err);

        return;

    }

    console.log("数据库连接成功");

    let db = client.db(dbName);

    //   增加数据

    db.collection("user").insertOne({"name":"李四","age":10},(err,result)=>{

        if(err){ //增加失败

            console.log(err);

            return;

        }

        console.log("增加成功");

        console.log(result);

          //操作数据库完毕以后一定要 关闭数据库连接

         client.close();

    })

})

## 前后端交互

·route.js

const fs = require('fs');

const path = require('path');

const url = require('url');

//扩展res

function changeRes(res){

    res.send=(data)=>{

        res.writeHead(200, { 'Content-Type': 'text/html;charset="utf-8"' });

        res.end(data);

    }

}

//根据后缀名获取文件类型

function getFileMime(extname) {

    var data = fs.readFileSync('./data/mime.json'); //同步方法读取文件

    let mimeObj = JSON.parse(data.toString());//获取文件数据并解析

    return mimeObj[extname];//将文件类型返回

}

//静态web服务的方法

function initStatic(req, res, staticPath) {

    //1、获取地址

    let pathname = url.parse(req.url).pathname;

    pathname = pathname == '/' ? '/index.html' : pathname;

    //获取后缀

    let extname = path.extname(pathname);

    //2、通过fs模块读取文件

    try {

        let data = fs.readFileSync('./' + staticPath + pathname);//读取静态web文件

        if (data) {

            let mime = getFileMime(extname);//获取文件类型

            //设置文件头

            res.writeHead(200, { 'Content-Type': '' + mime + ';charset="utf-8"' });

            res.end(data);

        }

    } catch (error) {

    }

}

let server = () => {

    let G = {

        \_get:{},

        \_post:{},

        staticPath:'static' //默认静态web目录

    };

    let app = function (req, res) {

        //扩展res的方法

        changeRes(res);

        //配置静态web服务

        initStatic(req, res,G.staticPath);

        //解析url并返回路径名

        let pathname = url.parse(req.url).pathname;

        //获取请求类型

        let method=req.method.toLowerCase();

        // console.log(method);

        //如果路径存在

        if (G['\_'+method][pathname]) {

            // console.log("成功"+pathname);

            //如果是get请求

            if(method=="get"){

                G['\_'+method][pathname](req, res);  //执行get请求的方法

            }else{

                //post  获取post的数据 把它绑定到req.body

                let postData = '';

                req.on('data',(chunk)=>{

                    postData+=chunk;

                })

                req.on('end',()=>{

                    //设置请求体

                   req.body=postData;

                   G['\_'+method][pathname](req, res);  //执行post方法

                })

            }

        } else {//如果页面不存在

            //获取pathname的后缀

            let extname = path.extname(pathname);

            //判断是不是其他文件类型

            if(extname!==""){//如果后缀不为空

            }else{

                console.log("失败"+pathname);

                res.writeHead(404, { 'Content-Type': 'text/html;charset="utf-8"' });

                res.end('页面不存在');

            }

        }

    }

    //get请求

    app.get = function (str, cb) {

        //注册方法

        G.\_get[str] = cb;

    }

    //post请求

    app.post = function (str, cb) {

        //注册方法

        G.\_post[str] = cb;

    }

    //配置静态web服务目录

    app.static=function(staticPath){

        G.staticPath=staticPath;

    }

    return app;

}

//暴露

module.exports = server();

·index.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h2>获取用户列表数据</h2>

    <!--将后端传过来的数据渲染到ul表-->

    <ul>

        <%for(var i=0;i<list.length;i++){%>

         <li><%=list[i].name%>---<%=list[i].age%></li>

        <%}%>

    </ul>

</body>

</html>

·register.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <link rel="stylesheet" href="./css/style.css" />

</head>

<body>

    <h2>注册页面</h2>

    <form action="/doRegister" method="post">

        用户名：<input type="text" name="name" />

        <br>

        <br>

         年　龄： <input type="text" name="age" />

        <br>

        <br>

        <!--提交注册表单-->

        <input type="submit" value="提交">

    </form>

</body>

</html>

·app.js

const http = require("http");

const app = require('./module/route');

const ejs = require("ejs");

//引入querystring模块

const querystring = require('querystring');

const { MongoClient } = require('mongodb');

//数据库连接的地址

const url = 'mongodb://admin:123456@localhost:27017/';

//要操作的数据库

const dbName = 'user';

// const client = new MongoClient(url,{ useUnifiedTopology: true });

//注册web服务

http.createServer(app).listen(8081);

// app.static("public");    //修改默认静态web目录

//配置路由

//首页

app.get('/', function (req, res) {

    //连接数据库

    MongoClient.connect(url,{ useUnifiedTopology: true }, (err, client) => {

        if (err) {

            console.log(err);

            return;

        }

        //切换到daName的数据库

        let db = client.db(dbName);

        //查询数据

        db.collection("user").find({}).toArray((err, result) => {

            if (err) {

                console.log(err);

                return;

            }

            //关闭数据库连接

            client.close();

            //渲染首页，并将获取到的数据传给首页

            ejs.renderFile("./views/index.ejs", {

                list: result

            }, (err, data) => {

                res.send(data);

            })

        })

    })

})

//注册

app.get('/register', function (req, res) {

    //渲染注册页面

    ejs.renderFile("./views/register.ejs",{},(err,data)=>{

        res.send(data);

    })

})

//注册处理

app.post('/doRegister', function (req, res) {

    console.log("注册处理");

    // name=zhangsan&age=13

    // {

    //     "name":"zhangsan",

    //     "age":13

    // }

    //解析请求体，将url参数转换成js对象

    let body=querystring.parse(req.body);

    //连接数据库

    MongoClient.connect(url,{ useUnifiedTopology: true },(err,client)=>{

        if(err){

            console.log(err);

            return;

        }

        //切换到要操作的数据库

        let db=client.db(dbName);

        //增加数据

        db.collection("user").insertOne(body,(err,result)=>{

            if(err){

                console.log(err);

                return;

            }

            console.log("增加数据成功");

            res.send("增加数据成功");

        })

    })

})

console.log("服务器已启动");

# Express框架

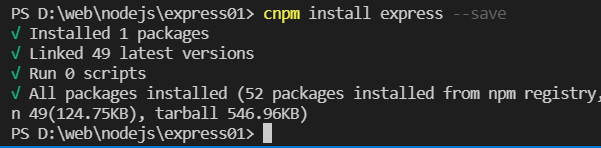
·安装rester插件

- https://www.extfans.com/search/extensions/rester/

·安装express模块

- npm init --yes

- npm install express --save



## Express 框架中的路由

//引入express

const express = require('express')

const app=express()

app.get("/",(req,res)=>{

    res.send("你好 express")

})

app.get("/article",(req,res)=>{

    res.send("新闻页面")

})

app.get("/login",(req,res)=>{

    res.send("登录")

})

app.get("/register",(req,res)=>{  //get:显示数据

    res.send("注册页面")

})

app.post("/doLogin",(req,res)=>{   //post:增加数据

    console.log("执行登录")

    res.send("执行登录")

})

app.put("/editUser",(req,res)=>{  //put：主要用于修改数据

    console.log("修改用户")

    res.send("修改用户")

})

app.delete("/deleteUser",(req,res)=>{  //delete：主要用于删除数据

    console.log("执行删除")

    res.send("执行删除")

})

//路由里面配置多级目录  http://localhost:3000/admin/user/edit

app.get("/admin/user/add",(req,res)=>{

    res.send("admin user add")

})

app.get("/admin/user/edit",(req,res)=>{

    res.send("admin user  edit")

})

//动态路由  配置路由的时候也要注意顺序

app.get("/article/add",(req,res)=>{

    res.send("article add")

})

app.get("/article/:id",(req,res)=>{

    var id=req.params["id"]    //获取动态路由的id值

    res.send("动态路由"+id)

})

//get 传值  http://localhost:3000/product?id=123&cid=123

app.get("/product",(req,res)=>{

    let query = req.query   //获取get传值

    console.log(query)

    res.send("product-"+query.id)

})

//设置端口号

app.listen(8081);

## Express 框架中 ejs 的安装使用

·安装ejs和express模块

### 基本使用

·footer.ejs

<footer>

    <h1>公共的底部</h1>

</footer>

·index.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h2>我是一个ejs模板引擎</h2>

    <p><%=title%></p>

    <!--引入文件并解析html标签-->

    <%- include('footer.ejs') %>

</body>

</html>

·news.ejs

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h2>绑定数据</h2>

    <p><%=userinfo.username%>---<%=userinfo.age%></p>

    <p><%=article%></p>

    <p><%-article%></p>

    <h2>条件判断</h2>

    <!--条件判断-->

    <%if(flag==true){%>

        <strong>flag=true</strong>

    <%}%>

    <%if(score>=60){%>

        <p>及格</p>

    <%}else{%>

       不及格

    <%}%>

    <h2>循环遍历</h2>

    <ul>

        <%for(let i=0;i<list.length;i++){%>

            <li><%=list[i]%></li>

        <%}%>

    </ul>

    <br>

    <ul>

        <%for(let i=0;i<newsList.length;i++){%>

            <li><%=newsList[i].title%></li>

        <%}%>

    </ul>

    <%- include('footer.ejs') %>

</body>

</html>

·app.js

/\*

1、安装  cnpm install ejs --save

2、app.set("view engine","ejs")

3、使用  (默认加载模板引擎的文件夹是views)

    res.render("index",{

    })

\*/

//引入express模块,express已经内置了ejs模板引擎，不需要引入，但是需要安装ejs

const express = require("express");

//实例化express()

const app = express()

app.get("/",(req,res)=>{

    let title = "你好ejs";

    //渲染模板引擎，默认加载模板引擎的文件夹是views

    //将title传到前端页面

    res.render("index",{

        title:title

    })

})

app.get("/news",(req,res)=>{

    let userinfo={

        username:"张三",

        age:20

    }

    let article="<h3>我是一个h3</h3>"

    let list=["1111","22222","3333333"]

    let newsList=[

        {

            title:"新闻1111",

        },

        {

            title:"新闻122222",

        },

        {

            title:"新闻33331",

        },

        {

            title:"新闻44444",

        }

    ]

    res.render("news",{

        userinfo:userinfo,

        article:article,

        flag:true,

        score:60,

        list:list,

        newsList:newsList

    })

})

//监听端口  端口号建议写成3000以上

app.listen(8081);

### 配置

·指定模板位置 ，默认模板位置在 views

- app.set('views', \_\_dirname + '/views');

·app.js

/\*

1.在 app.js 的头上定义 ejs:,代码如下:

var ejs = require('ejs');

2.注册 html 模板引擎代码如下：

app.engine('html',ejs.\_\_express);

3.将模板引擎换成 html 代码如下:

app.set('view engine', 'html');

4.修改模板文件的后缀为.html。

\*/

//引入express

const express = require("express");

//引入ejs

const ejs = require("ejs");

//实例化express()

const app = express()

//配置模板引擎的后缀名为html

app.engine("html",ejs.\_\_express)

//将后缀名改为html

app.set("view engine","html")

//配置静态web目录

app.use(express.static("static"))

app.get("/",(req,res)=>{

    let title = "你好ejs";

    res.render("index",{

        title:title

    })

})

app.get("/news",(req,res)=>{

    let userinfo={

        username:"张三",

        age:20

    }

    let article="<h3>我是一个h3</h3>"

    let list=["1111","22222","3333333"]

    let newsList=[

        {

            title:"新闻1111",

        },

        {

            title:"新闻122222",

        },

        {

            title:"新闻33331",

        },

        {

            title:"新闻44444",

        }

    ]

    res.render("news",{

        userinfo:userinfo,

        article:article,

        flag:true,

        score:60,

        list:list,

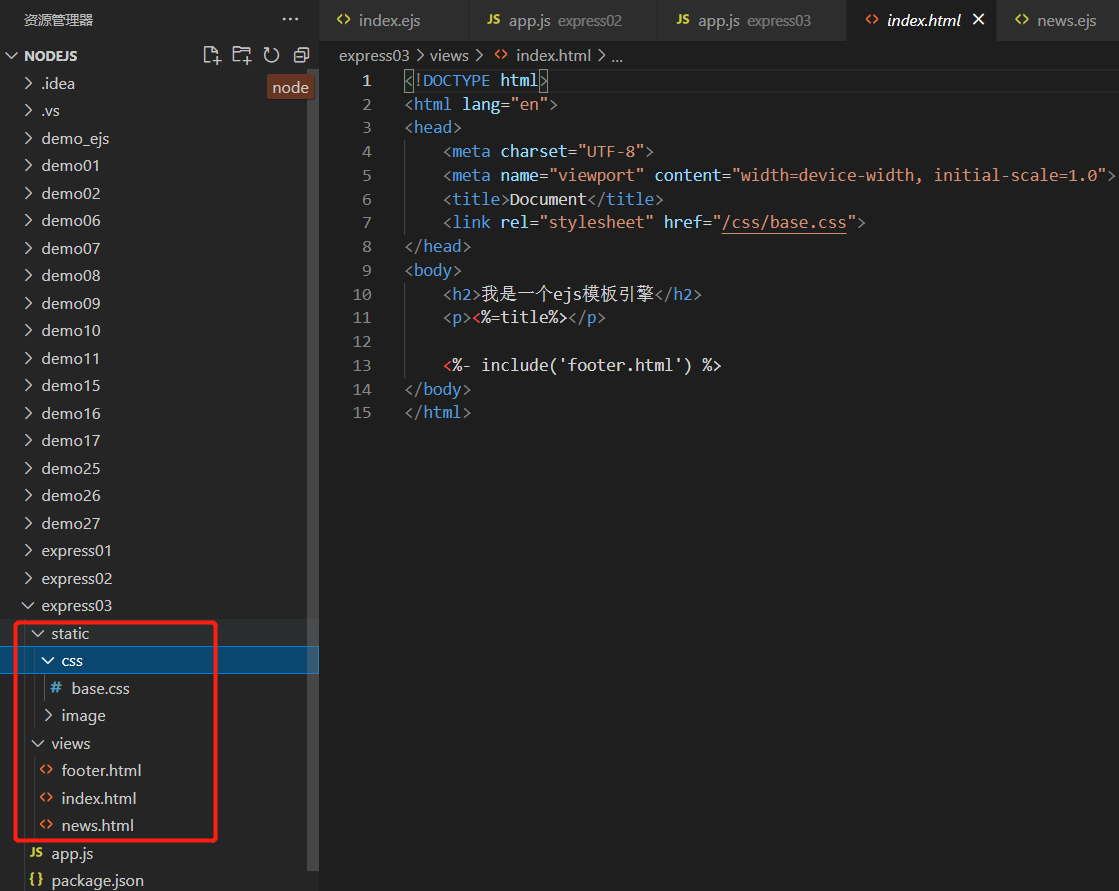
        newsList:newsList

    })

})

//监听端口  端口号建议写成3000以上

app.listen(8081)



## Express 中间件

·中间件就是匹配路由之前或者匹配路由完成做的一系列的操作

·中间件中如果想往下 匹配的话，那么需要写 next()

·中间件的功能

- 执行任何代码

- 修改请求和响应对象

- 终结请求-响应循环

- 调用堆栈中的下一个中间件

·Express 应用可使用如下几种中间件

- 应用级中间件

- 路由级中间件

- 错误处理中间件

- 内置中间件

- 第三方中间件

/\*\*

     中间件：就是匹配路由之前和匹配路由之后做的一系列的操作

     中间件可以用来做权限判断：没有登录 跳转到登录页面，登录以后就显示登录以后的页面

     Express 应用可使用如下几种中间件：应用级中间件、路由级中间件(用的比较少)、错误处理中间件、内置中间件、第三方中间件

 \*/

const express = require("express");

const app = express()

//4、内置中间件 配置静态web目录

app.use(express.static("static"))

//1、应用级中间件 (用于权限判断)

//当访问其他路径结束时，会自动调用此中间件方法

app.use((req,res,next)=>{

    console.log(new Date())

    // res.send("aaa");

    //如果不调用next页面会卡死

    next()

})

app.get("/",(req,res)=>{

    res.send("首页")

})

app.get("/login",(req,res)=>{

    res.send("执行登录")

})

//2、路由级中间件(用的比较少)

app.get("/news/add",(req,res,next)=>{

    // res.send("执行增加新闻")

    console.log("执行增加新闻")

    //向下匹配符号条件的路由

    next()

})

app.get("/news/:id",(req,res)=>{

    res.send("新闻动态路由")

})

//3、错误处理中间件  /css/base.css

app.use((req,res,next)=>{

    //如果状态码为404时

   res.status(404).send("404")

})

app.listen(8081)

## 使用第三方中间件body-Parser

·安装body-Parser

- npm install body-parser --save

·login.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <link rel="stylesheet" href="/css/base.css">

</head>

<body>

  <h2>post提交数据</h2>

  <form action="/doLogin" method="post">

    用户名: <input type="text" name="username" /> <br> <br>

    密码: <input type="password" name="password" /><br> <br>

    <input type="submit" value="提交">

</form>

</body>

</html>

·app.js

/\*

获取post传过来的数据

1、cnpm install body-parser --save

2、var bodyParser = require('body-parser')

3、配置中间件

    app.use(bodyParser.urlencoded({ extended: false }))

    app.use(bodyParser.json())

4、接收post数据

    req.body

\*/

const express = require("express");

//引入body-parser中间件模块

const bodyParser = require('body-parser')

const ejs = require("ejs");

const app = express()

//配置模板引擎

app.engine("html",ejs.\_\_express)

//修改ejs模板后缀名

app.set("view engine","html")

//配置静态web目录

app.use(express.static("static"))

//配置第三方bodyParser中间件

app.use(bodyParser.urlencoded({ extended: false }))//接收from表单传递过来的数据

app.use(bodyParser.json())//接收json数据

app.get("/",(req,res)=>{

    res.send("首页")

})

app.get("/login",(req,res)=>{

    //渲染login页面

   // req.query 获取get传值

   res.render("login",{})

})

//处理登录逻辑

app.post("/doLogin",(req,res)=>{

   // req.body 获取post传值，必须先配置bodyParser中间件

   var body = req.body;

   console.log(body)

   res.send("执行提交"+body.username)

})

//监听端口  端口号建议写成3000以上

app.listen(8081)

## Express Cookie 的基本使用

·cookie 是存储于访问者的计算机中的变量。可以让我们用同一个浏览器访问同一个域 名的时候共享数据

·当访问一个页面的时候，服务器在下行 HTTP 报文中， 命令浏览器存储一个字符串; 浏览器再访问同一个域的时候，将把这个字符串携带到上行 HTTP 请求中。第一次访问一个服务器，不可能携带 cookie。 必须是服务器得到这次请求， 在下行响应报头中，携带 cookie 信息，此后每一次浏览器往这个服务器发出的请求，都会 携带这个 cookie

·单个 cookie 保存的数据大小不能超过 4K，很多浏览器都限制一个站点最多保存 20 个 cookie

·Cookie 特点

- cookie 保存在浏览器本地

- 正常设置的 cookie 是不加密的，用户可以自由看到

- 用户可以删除 cookie，或者禁用它

- cookie 可以被篡改

- cookie 可以用于攻击

- cookie 存储量很小。未来实际上要被 localStorage 替代，但是后者 IE9 兼容

·安装cookie-parser模块

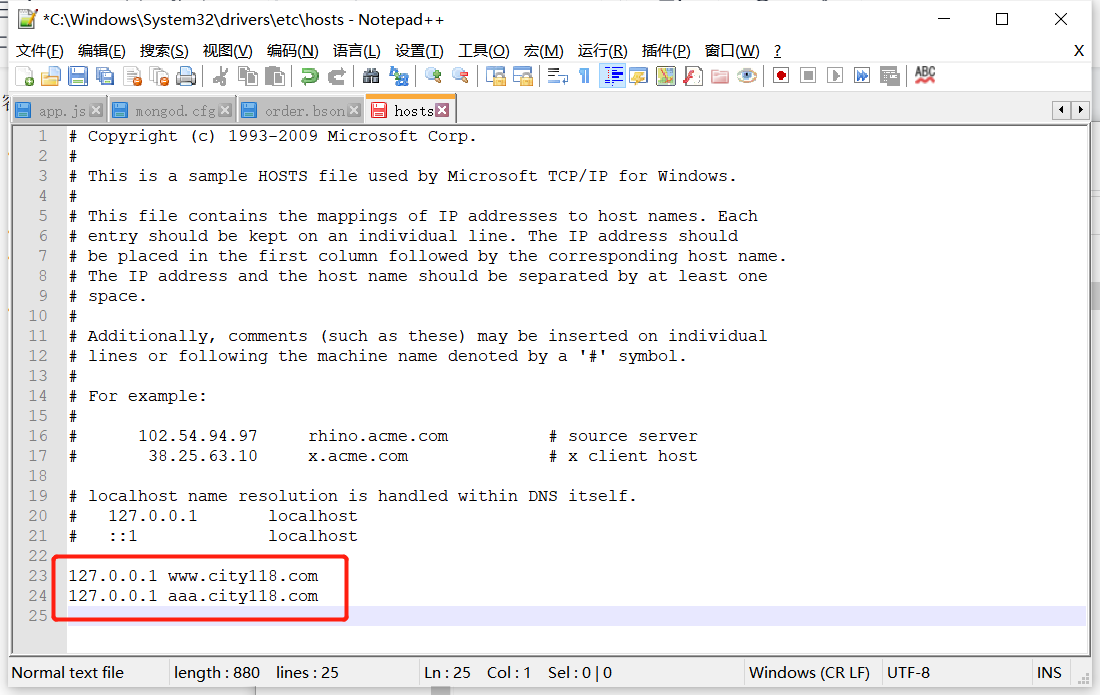
- npm install cookie-parser --save

·http默认端口是80

·https默认端口是443

·windows域名解析

- 打开C:\Windows\System32\drivers\etc\hosts文件



const express = require('express')

const app=express()

//引入cookie-parser

const cookieParser = require('cookie-parser')

//配置cookieParser中间件，传入加密参数

app.use(cookieParser("city118"))

app.get("/",(req,res)=>{

    //设置cookie，第一个参数是key，第二个参数是value，maxAge是过期时间

    //如果cookie没有过期的话，关闭浏览器后重新打开,cookie不会销毁

    // res.cookie("username","zhangsan",{maxAge:1000\*60\*60})

    //path是设置获取cookie的指定路由

    // res.cookie("username","zhangsan",{maxAge:1000\*60\*60,path:"/article"})

    //domain设置多个域名共享cookie  www.city118.com   bbb.city118.com

    // res.cookie("username","zhangsan1111",{maxAge:1000\*60\*60,domain:".city118.com"})

    //signed设置加密

    res.cookie("username","张三",{maxAge:1000\*60\*60,signed:true})

    res.send("你好 express aaa")

})

app.get("/article",(req,res)=>{

    //获取之前设置的cookie

    let username=req.cookies.username;

    console.log(username)

    res.send("新闻页面--"+username)

})

app.get("/user",(req,res)=>{

     //获取cookie

     let username=req.cookies.username;

     console.log(username)

    res.send("用户--"+username)

})

app.get("/product",(req,res)=>{

    //获取加密的cookie

    let username=req.signedCookies.username

    console.log(username)

   res.send("product--"+username)

})

app.listen(80)

## Express Session 的基本使用

·session 是另一种记录客户状态的机制，不同的是 Cookie 保存在客户端浏览器中，而 session 保存在服务器上

·安装express-session

- express-session

const express = require('express')

const session = require('express-session')

const app=express()

//配置session的中间件

app.use(session({

    secret: 'this is session', //服务器端生成 session 的签名

    name:"itying", //修改session对应cookie的名称

    resave: false, //强制保存 session 即使它并没有变化

    saveUninitialized: true, //强制将未初始化的 session 存储

    cookie: {

        maxAge:1000\*60\*30,//设置过期时间

        secure: false  // true 表示只有https协议才能访问cookie

    },

    rolling:true  //在每次请求时强行设置 cookie，这将重置 cookie 过期时间（默认：false）

}))

app.get("/",(req,res)=>{

    //获取seesion

    if(req.session.username || req.session.age){//查看session里是否有该用户

        res.send(req.session.username+"--"+req.session.age+"-已登录")

    }else{

        res.send("没有登录")

    }

})

app.get("/login",(req,res)=>{

    //设置seesion

    req.session.username="张三"

    req.session.age=20

    res.send("执行登录")

})

app.get("/loginOut",(req,res)=>{

    //1、设置session的过期时间为0  (它会把所有的session都销毁)

    // req.session.cookie.maxAge=0

    //2、销毁指定session

    req.session.username=""

    //3、销毁session  destroy

    req.session.destroy()

    res.send("退出登录")

})

app.listen(8081)

### 负载均衡配置 Session

·把 Session 保存到数据库里面

·安装connect-mongo模块

- npm install connect-mongo --save

const express = require('express')

const session = require('express-session')

const MongoStore = require('connect-mongo')(session);

const app=express()

//配置session的中间件

app.use(session({

    secret: 'this is session', //服务器端生成 session 的签名

    name:"itying", //修改session对应cookie的名称

    resave: false, //强制保存 session 即使它并没有变化

    saveUninitialized: true, //强制将未初始化的 session 存储

    cookie: {

        maxAge:1000\*60\*30,

        secure: false  // true 表示只有https协议才能访问cookie

    },

    rolling:true,  //在每次请求时强行设置 cookie，这将重置 cookie 过期时间（默认：false）

    store: new MongoStore({

        //如果数据库设置了管理员，应该用该管理员的身份验证登录

        url: 'mongodb://ckc:123456@localhost:27017/user',

        touchAfter: 24 \* 3600 // 不管发出了多少请求 在24小时内只更新一次session， 除非你改变了这个session

    })

}))

app.get("/",(req,res)=>{

    //获取seesion

    if(req.session.username || req.session.age){

        res.send(req.session.username+"--"+req.session.age+"-已登录")

    }else{

        res.send("没有登录")

    }

})

app.get("/login",(req,res)=>{

    //设置seesion

    req.session.username="张三"

    req.session.age=20

    res.send("执行登录")

})

app.get("/loginOut",(req,res)=>{

    //1、设置session的过期时间为0  (它会把所有的session都销毁)

    // req.session.cookie.maxAge=0

    //2、销毁指定session

    // req.session.username=""

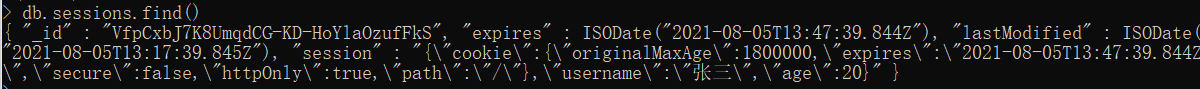
    //3、销毁session  destroy

    req.session.destroy()

    res.send("退出登录")

})

app.listen(8081)



## Express 路由模块化

### 路由模块化使用

·login.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

    <link rel="stylesheet" href="/css/base.css">

</head>

<body>

  <h2>post提交数据</h2>

  <form action="/login/doLogin" method="post">

    用户名: <input type="text" name="username" /> <br> <br>

    密码: <input type="password" name="password" /><br> <br>

    <input type="submit" value="提交">

</form>

</body>

</html>

·login.js

const express = require("express");

//实例化Router()

var router = express.Router()

router.get("/",(req,res)=>{

    // req.query 获取get传值

    res.render("login",{})

 })

router.post("/doLogin",(req,res)=>{

    // req.body 获取post传值

    var body = req.body;

    console.log(body)

    res.send("执行提交"+body.username)

 })

//暴露

 module.exports = router

·app.js

const express = require("express");

const bodyParser = require('body-parser')

const ejs = require("ejs");

const app = express()

//引入外部模块

const login = require("./routes/login")

//配置模板引擎

app.engine("html",ejs.\_\_express)

//将ejs后缀名改为html

app.set("view engine","html")

//配置静态web目录

app.use(express.static("static"))

//配置第三方中间件

app.use(bodyParser.urlencoded({ extended: false }))

app.use(bodyParser.json())

//挂载login模块

app.use("/login",login)

app.get("/",(req,res)=>{

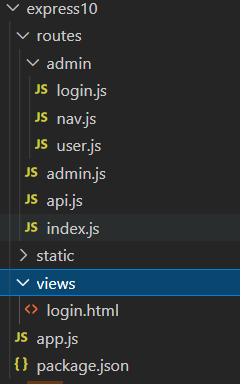
    res.send("首页")

})

//监听端口  端口号建议写成3000以上

app.listen(8081)

### 企业级路由模块化搭建



·admin.js

const express = require("express");

var router = express.Router()

//引入模块

const user = require("./admin/user")

const login = require("./admin/login")

const nav = require("./admin/nav")

router.get("/",(req,res)=>{

   res.send("后台管理中心")

})

//挂载路由

router.use("/user",user)

router.use("/login",login)

router.use("/nav",nav)

module.exports = router

- login.js

const express = require("express");

var router = express.Router()

router.get("/", (req, res) => {

    res.send("用户登录页面")

})

router.get("/doLogin", (req, res) => {

    res.send("执行登录")

})

module.exports = router

- nav.js

const express = require("express");

var router = express.Router()

router.get("/", (req, res) => {

    res.send("导航列表")

})

router.get("/add", (req, res) => {

    res.send("增加导航")

})

router.get("/edit", (req, res) => {

    res.send("修改导航")

})

router.post("/doAdd", (req, res) => {

    res.send("执行增加")

})

router.post("/doEdit", (req, res) => {

    res.send("执行修改")

})

module.exports = router

- user.js

const express = require("express");

var router = express.Router()

router.get("/", (req, res) => {

    res.send("用户列表")

})

router.get("/add", (req, res) => {

    res.send("增加用户")

})

router.get("/edit", (req, res) => {

    res.send("修改用户")

})

router.post("/doAdd", (req, res) => {

    res.send("执行增加")

})

router.post("/doEdit", (req, res) => {

    res.send("执行修改")

})

module.exports = router

·api.js

const express = require("express");

var router = express.Router()

router.get("/",(req,res)=>{

   res.send("api接口")

 })

 module.exports = router

·index.js

const express = require("express");

var router = express.Router()

router.get("/",(req,res)=>{

   res.send("首页")

})

 module.exports = router

·app.js

const express = require("express");

const bodyParser = require('body-parser')

const ejs = require("ejs");

//引入外部模块

const admin = require("./routes/admin")

const index = require("./routes/index")

const api = require("./routes/api")

const app = express()

//配置模板引擎

app.engine("html",ejs.\_\_express)

app.set("view engine","html")

//配置静态web目录

app.use(express.static("static"))

//配置第三方中间件

app.use(bodyParser.urlencoded({ extended: false }))

app.use(bodyParser.json())

//配置外部路由模块

app.use("/admin",admin)

app.use("/api",api)

app.use("/",index)

//监听端口  端口号建议写成3000以上

app.listen(8081)

## Express 应用生成器

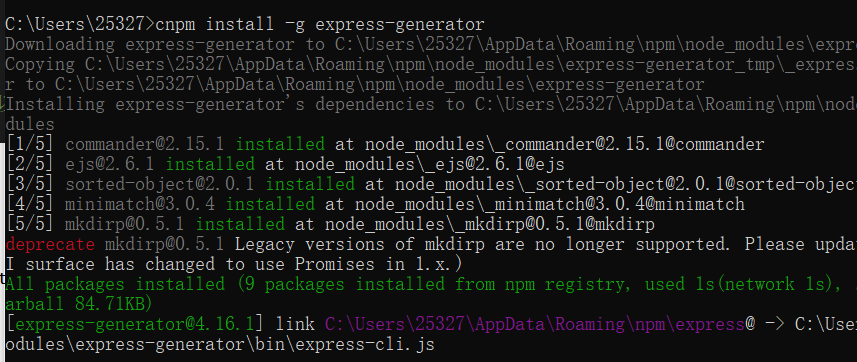
·通过应用生成器工具 express-generator 可以快速创建一个应用的骨架

·安装express-generator

- npx express-generator

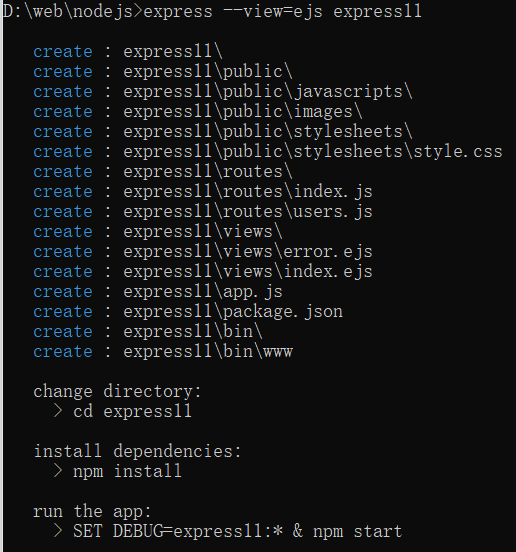
- 或者

- cnpm install -g express-generator



·使用

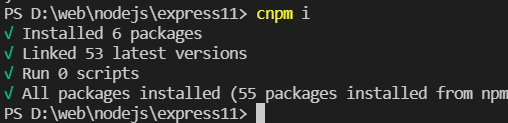
- express --view=ejs express09





·运行项目

- cnpm i



- node bin/www



## Express 结合 multer 上传文件

### Multer 模块介绍

·Multer 是一个 node.js 中间件，用于处理 multipart/form-data 类型的表单数据，它主要用于上传文件

·Multer 不会处理任何非 multipart/form-data 类型的表单数据

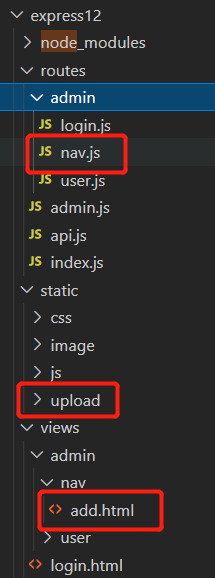
### Express 上传文件模块 multer 的使用

·使用之前写好的模块化

·安装 multer

- npm install multer --save

·本节要修改的文件



·add.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>增加导航</title>

</head>

<body>

    <!--enctype执行上传的图片-->

    <form action="/admin/nav/doAdd" method="post" enctype="multipart/form-data">

        标题：<input type="text" name="title" id="title" />

        <br>

        <br>

        <!--设置name属性-->

        图片：<input type="file" name="pic" id="pic" />

        <br>

        <br>

        描述：<textarea name="desc" id="desc" cols="30" rows="10"></textarea>

        <br>

        <br>

        <input type="submit" value="提交">

    </form>

</body>

</html>

·nav.js

const express = require("express");

//引入multer模块

const multer = require('multer');

const path = require('path')

var storage = multer.diskStorage({

    //配置上传的目录

    destination: function (req, file, cb) {

        cb(null, 'static/upload') //相对路径 上传之前目录必须存在

        // cb(null, 'D:/file') //绝对路径 上传之前目录必须存在

    },

    //修改上传后的文件名

    filename: function (req, file, cb) {

        //1、获取后缀名

        let extname= path.extname(file.originalname);

        //2、根据时间戳生成文件名

        cb(null, Date.now()+extname)

    }

})

var upload = multer({ storage: storage })

var router = express.Router()

router.get("/", (req, res) => {

    res.send("导航列表")

})

router.get("/add", (req, res) => {

    res.render("admin/nav/add")

})

router.get("/edit", (req, res) => {

    res.send("修改导航")

})

//upload.single("pic")接收上传的文件pic为前端页面的name属性

router.post("/doAdd", upload.single("pic"), (req, res) => {

    //获取表单传过来的数据

    res.send({

        body: req.body,

        //接收文件

        file: req.file

    });

})

router.post("/doEdit", (req, res) => {

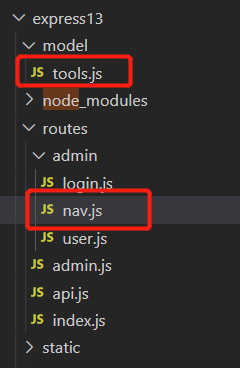
    res.send("执行修改")

})

module.exports = router

### 模块化封装文件上传

·要修改的文件



·tools.js

const multer = require('multer');

const path = require('path');

let tools={

    //上传文件的方法

    multer(){

        var storage = multer.diskStorage({

            //配置上传的目录

            destination: function (req, file, cb) {

                cb(null, 'static/upload') //上传之前目录必须存在

            },

            //修改上传后的文件名

            filename: function (req, file, cb) {

                //1、获取后缀名

                let extname= path.extname(file.originalname);

                //2、根据时间戳生成文件名

                cb(null, Date.now()+extname)

            }

        })

        var upload = multer({ storage: storage })

        return upload;

    },

    md5(){

    }

}

module.exports=tools

·nav.js

const express = require("express");

//引入工具模块

const tools = require("../../model/tools");

var router = express.Router();

router.get("/", (req, res) => {

    res.send("导航列表")

})

router.get("/add", (req, res) => {

    res.render("admin/nav/add")

})

router.get("/edit", (req, res) => {

    res.send("修改导航")

})

//调用工具模块的上传文件方法

router.post("/doAdd",tools.multer().single("pic"), (req, res) => {

    //获取表单传过来的数据

    res.send({

        body: req.body,

        file: req.file

    });

})

router.post("/doEdit", (req, res) => {

    res.send("执行修改")

})

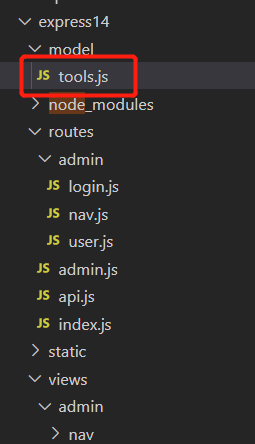
module.exports = router

### Express 按照日期生成上传文件目录

·安装silly-datetime模块

- npm install silly-datetime --save

·要修改的文件



const multer = require('multer');

const path = require('path');

//引入日期格式化模块

const sd = require('silly-datetime');

//引入新建文件夹模块

const mkdirp = require('mkdirp')

let tools={

    multer(){

        var storage = multer.diskStorage({

            //配置上传的目录

            //定义async

            destination: async (req, file, cb)=>{

                //1、获取当前日期并格式化成年月日 2020\_07\_03

                let day=sd.format(new Date(), 'YYYY\_MM\_DD');

                // 拼接路径，static/upload/20200703

                let dir=path.join("static/upload",day)

                //2、按照日期生成图片存储目录  mkdirp是一个异步方法，所以要加await等待目录创建完毕

                //如果目录已经存在则不会再次创建

                await mkdirp(dir)

                cb(null, dir) //上传之前目录必须存在

            },

            //修改上传后的文件名

            filename: (req, file, cb)=> {

                //1、获取后缀名

                let extname= path.extname(file.originalname);

                //2、根据时间戳生成文件名

                cb(null, Date.now()+extname)

            }

        })

        var upload = multer({ storage: storage })

        return upload;

    },

    md5(){

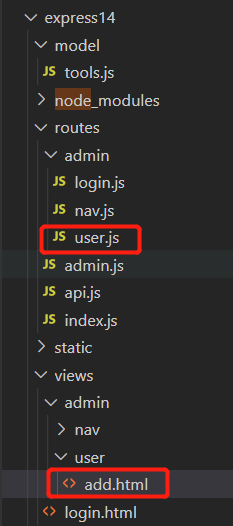
    }

}

module.exports=tools

### 多文件上传

·要修改的文件



·add.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>增加导航</title>

</head>

<body>

    <!--enctype执行上传的图片-->

    <form action="/admin/user/doAdd" method="post" enctype="multipart/form-data">

        标题：<input type="text" name="title" id="title" />

        <br>

        <br>

        <!--设置name属性-->

        图片1：<input type="file" name="pic1" id="pic1" />

        <br>

        <br>

        <!--设置name属性-->

        图片2：<input type="file" name="pic2" id="pic2" />

        <br>

        <br>

        描述：<textarea name="desc" id="desc" cols="30" rows="10"></textarea>

        <br>

        <br>

        <input type="submit" value="提交">

    </form>

</body>

</html>

·user.js

const express = require("express");

const tools = require("../../model/tools");

var router = express.Router()

router.get("/", (req, res) => {

    res.send("用户列表")

})

router.get("/add", (req, res) => {

    res.render("admin/user/add")

})

//一次接收多个文件

let cpUpload = tools.multer().fields([{ name: 'pic1', maxCount: 1 }, { name: 'pic2', maxCount: 1 }])

router.post("/doAdd",cpUpload, (req, res) => {

    res.send({

        body:req.body,

        //这里是req.files

        files:req.files

    })

})

module.exports = router

# mongoose操作数据库

## mongoose 介绍

·Mongoose 是在 node.js 异步环境下对 mongodb 进行便捷操作的对象模型工具

·Mongoose 是 NodeJS 的驱动，不能作为其他语言的驱动

·Mongoose 的特点

- 通过关系型数据库的思想来设计非关系型数据库

- 基于 mongodb 驱动，简化操作

## 基本使用

·安装mongoose模块

- npm i mongoose --save

//1.引入mongoose

const mongoose = require('mongoose');

//2、建立连接

mongoose.connect('mongodb://ckc:123456@localhost:27017/user');

//3、操作users表（集合）   定义一个Schema   Schema里面的对象和数据库表里面的字段需要一一对应

var UserSchema=mongoose.Schema({

    name:String,

    age:Number

})

//4、定义数据库模型  操作数据库

// model里面的第一个参数 要注意：1首字母大写  2、要和数据库表（集合 ）名称对应  这个模型会和模型名称相同的复数的数据库表建立连接

// var User=mongoose.model('User',UserSchema);    // 默认会操作 users表（集合）

// var User=mongoose.model('User',UserSchema);

var User=mongoose.model('User',UserSchema,'user');  //如果有第三个参数，默认会操作第三个参数配置的表  user表（集合）

//5、查询users表的数据

    // User.find({},function(err,doc){

    //     if(err){

    //         console.log(err);

    //         return;

    //     }

    //     console.log(doc);

    // })

//6、增加数据

// 6.1实例化 Model     通过实例化User Molde 创建增加的数据

    //要增加的数据

    var u=new User({

        name:'李四',

        age:20

    });

    //6.2 实例.save()

    u.save(function(err){

        if(err){

            console.log(err);

            return;

        }

        console.log('成功')

    });  //执行增加操作

//1.引入mongoose

const mongoose = require('mongoose');

//2、建立连接

mongoose.connect('mongodb://ckc:123456@localhost:27017/user');

//3、定义一个Schema

var NewsSchema=mongoose.Schema({

    title:"string",//数据类型可用字符串小写

    author:String,

    pic:String,

    content:String,

    status:Number

})

//4、定义操作数据库的Model

var News=mongoose.model('News',NewsSchema,'news');

//5、增加数据

//通过实例化 Model 创建增加的数据

    // var news=new News({

    //     title:"我是一个新闻",

    //     author:'张三',

    //     content:'我是新闻的内容',

    //     status:1

    // });

    // news.save(function(err){

    //     if(err){

    //         return console.log(err);

    //     }

    //      console.log('成功')

    // });

//6、修改数据

// News.updateOne(

//     {"\_id":"610cbcdc6182c36218302bdf"},//查找的数据

//     {"title":"我是一个新闻2222"},//修改的数据

//     function(err,doc){

//         if(err){

//                 return console.log(err);

//             }

//             console.log(doc)

//     })

//删除数据

// News.deleteOne(

//     {"\_id":"610cbcdc6182c36218302bdf"},

//      (err,result)=>{

//             if(err){

//                 return console.log(err);

//             }

//             console.log(result)

// })

## 默认参数

//mongoose 默认参数、mongoose 模块化 、mongoose性能疑问

// mongoose 默认参数：增加数据的时候，如果不传入数据会使用默认配置的数据

var mongoose=require('mongoose');

//useNewUrlParser这个属性会在url里识别验证用户所需的db,未升级前是不需要指定的,升级到一定要指定。

mongoose.connect('mongodb://ckc:123456@localhost:27017/user',{ useNewUrlParser: true },function(err){

        if(err){

            console.log(err);

            return;

        }

        console.log('数据库连接成功')

});

// 定义数据表（集合的）映射  注意：字段名称必须和数据库保持一致

var UserSchema=mongoose.Schema({

    name:String,

    age:Number,

    status:{

        type:Number,

        default:1   //指定默认数据

    }

})

//定义model操作数据库

var UserModel=mongoose.model("User",UserSchema,'users');

//数据的查找

// UserModel.find({},function(err,doc){

//     if(err){

//         console.log(err);

//         return;

//     }

//     console.log(doc);

// })

//数据的增加

var user=new UserModel({

    name:'张三88888888888888',

    age:40,

    status:1,

    //sex不会保存到数据库

    sex:"男"

})

user.save(function(err){

    if(err){

        console.log(err);

        return;

    }

    console.log('增加数据成功')

});

var user=new UserModel({

    name:'张三6666666666666666',

    age:30

    //status默认为1

})

user.save(function(err){

    if(err){

        console.log(err);

        return;

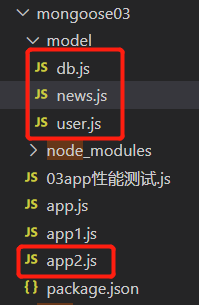
    }

    console.log('增加数据成功')

});

## mongoose 模块化

·要修改的文件



·db.js

//连接数据库

var mongoose=require('mongoose');

//useNewUrlParser这个属性会在url里识别验证用户所需的db,未升级前是不需要指定的,升级到一定要指定。

mongoose.connect('mongodb://ckc:123456@localhost:27017/user',{ useNewUrlParser: true },function(err){

        if(err){

            console.log(err);

            return;

        }

        console.log('数据库连接成功')

});

module.exports=mongoose;

·user.js

//user表操作

//引入连接数据库模块

var mongoose=require('./db.js');

//定义schema

var UserSchema=mongoose.Schema({

    name:String,

    age:Number

})

//切换到user表并暴露

module.exports=mongoose.model('User',UserSchema,'user');

·news.js

//新闻表操作

var mongoose=require('./db.js');

var NewsSchema=mongoose.Schema({

    title:"string",

    author:String,

    pic:String,

    content:String,

    status:{

        type:Number,

        default:1

    }

})

//切换到news表并暴露

module.exports=mongoose.model('News',NewsSchema,'news');

·app2.js

var UserModel=require('./model/user.js');

var NewsModel=require('./model/news.js');

var user=new UserModel({

    name:"李四666",

    age:40

})

//增加user表的数据

user.save(function(err){

    if(err){

        console.log(err);

        return;

    }

    //获取user表的数据

    UserModel.find({},function(err,docs){

        if(err){

            console.log(err);

            return;

        }

        console.log(docs);

    })

})

// NewsModel.find({},function(err,docs){

//     if(err){

//         console.log(err);

//         return;

//     }

//     console.log(docs);

// })

·只有第一次操作才会连接数据库，其余操作不会再次连接数据库，提高了性能

## mongoose 预定义模式修饰符

·对我们增加的数据进行格式化

·lowercase

- 全部字母转小写

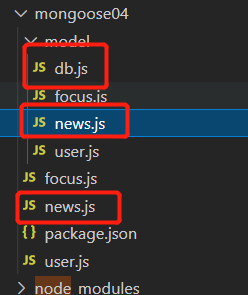
·uppercase

- 全部字母转大写

·trim

- 去除两边空格

·要修改的文件



·db.js

//连接数据库

var mongoose=require('mongoose');

//useNewUrlParser这个属性会在url里识别验证用户所需的db,未升级前是不需要指定的,升级到一定要指定。

mongoose.connect('mongodb://ckc:123456@localhost:27017/user',{ useNewUrlParser: true },function(err){

        if(err){

            console.log(err);

            return;

        }

        console.log('数据库连接成功')

});

module.exports=mongoose;

·news.js

var mongoose=require('./db.js');

var NewsSchema=mongoose.Schema({

    title:{

        type:String,

        trim:true    //定义 mongoose模式修饰符 去掉两边空格

    },

    author:String,

    pic:String,

    content:String,

    status:{

        type:Number,

        default:1 //设置默认值

    }

})

module.exports=mongoose.model('News',NewsSchema,'news');

·news.js

var NewsModel=require('./model/news.js');

var news =new NewsModel({

    title:"    我是一个国际新闻3333333   ",

    author:'张三333',

    pic:'http://www.xxx.com/x.png'

})

news.content='content'

news.save(function(err){

    if(err){

        console.log(err);

        return;

    }

    NewsModel.find({},function(err,docs){

        if(err){

            console.log(err);

            return;

        }

        console.log(docs);

    })

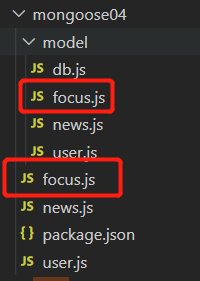
})

Getters 与 Setters 自定义修饰符

·通过 set（建议使用） 修饰符在增加数据的时候对数据进行格式化

·通过 get（不建议使用）在实例获取数据的时候对数据进行格式化

Setters  
·要操作的文件



var mongoose=require('./db.js');

var FocusSchema=mongoose.Schema({

    title:{

        type:String,

        trim:true    //定义 mongoose模式修饰符 去掉空格

    },

    pic:String,

    redirect:{

        type:String,

        set(parmas){   //增加数据的时候对redirect字段进行处理

            // parmas可以获取redirect的值 、    返回的数据就是redirect在数据库实际保存的值

            /\*

             www.baidu.com              http://www.baidu.com

             http://www.baidu.com       http://www.baidu.com

            \*/

            if(!parmas){//如果数据为空，返回空串

                return ''

            }else{

               if(parmas.indexOf('http://')!=0 && parmas.indexOf('https://')!=0){//如果没有http://或https://

                    return 'http://'+parmas;

               }

                return parmas

            }

        }

    },

    status:{

        type:Number,

        default:1

    }

})

module.exports=mongoose.model('Focus',FocusSchema,'focus');

var FocusModel=require('./model/focus.js');

var focus =new FocusModel({

    title:"    我是一个国际新闻666666   ",

    pic:'http://www.xxx.com/x.png',

    redirect:'www.bbbbb.com'

})

focus.save(function(err){

    if(err){

        console.log(err);

        return;

    }

    FocusModel.find({},function(err,docs){

        if(err){

            console.log(err);

            return;

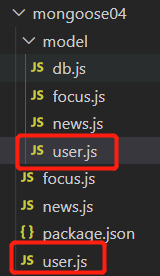
        }

        console.log(docs);

    })

});

Getters  
·要操作的文件



var mongoose=require('./db.js');

var UserSchema=mongoose.Schema({

    name:{

        type:String,

        get(params){   //不建议使用

            return "a001"+params;//返回"a001"+params，没有多大意义

        }

    },

    age:Number,

    status:{

        type:Number,

        default:1

    }

})

module.exports=mongoose.model('User',UserSchema,'user');

var UserModel=require('./model/user.js');

var user=new UserModel({

    name:'张三',

    age:21

})

console.log(user.name,user.age);//a001张三 21

//保存数据 实际上是 张三 21

user.save(function(err){

    if(err){

        console.log(err);

        return;

    }

   console.log('成功')

})

Mongoose 索引

var mongoose=require('./db.js');

var UserSchema=mongoose.Schema({

    name:{

        type:String,

        unique:true //设置唯一索引

    },

    sn:{

        type:String,

        index:true //设置普通索引

    },

    age:Number,

    status:{

        type:Number,

        default:1

    }

})

Mongoose 内置 CURD

·https://mongoosejs.com/docs/queries.html

·Model.deleteMany()

- 从集合中删除所有匹配的文档

- 无论single选项如何，会删除所有匹配的文档

await Character.deleteMany({ name: /Stark/, age: { $gte: 18 } });

·Model.deleteOne() 

- 从集合中删除第一个匹配的文档

- 无论single选项如何，最多删除一个文档

await Character.deleteOne({ name: '张三' });

·Model.find() 

- 在发送命令之前，Mongoose 会强制转换以匹配模型的模式

// 查找所有文档

await MyModel.find({});

// 查找所有名为john并且大于18岁的数据

await MyModel.find({ name: 'john', age: { $gte: 18 } }).exec();

// 执行查询并将结果传递给回调

MyModel.find({ name: 'john', age: { $gte: 18 }}, function (err, docs) {});

// 执行，名字中有john，只查看“name”和“friends”字段

await MyModel.find({ name: /john/i }, 'name friends').exec();

// 查看10条以后的数据

await MyModel.find({ name: /john/i }, null, { skip: 10 }).exec();

·Model.findById() 

- 通过 \_id 字段查找单个文档

·Model.findByIdAndDelete() 

·Model.findByIdAndRemove() 

·Model.findByIdAndUpdate() 

·Model.findOne() 

·Model.findOneAndDelete() 

·Model.findOneAndRemove() 

·Model.findOneAndUpdate() 

·Model.replaceOne() 

·Model.updateMany() 

·Model.updateOne()

## 扩展 Mongoose CURD 方法

var mongoose=require('./db.js');

var UserSchema=mongoose.Schema({

    name:{

        type:String,

        unique:true //设置唯一索引

    },

    sn:{

        type:String,

        index:true //设置普通索引

    },

    age:Number,

    status:{

        type:Number,

        default:1

    }

})

//静态方法

UserSchema.statics.findBySn=function(sn,cb){

    //通过 find方法获取 sn的数据    this 关键字获取当前的model

    this.find({"sn":sn},function(err,docs){

        cb(err,docs)

    })

}

// 实例方法   (基本不用)

UserSchema.methods.print=function(){

    console.log('我是一个实例方法')

    console.log(this.name)

}

module.exports=mongoose.model('User',UserSchema,'user');

var UserModel=require('./model/user.js');

// var user = new UserModel({

//     name: '赵六',

//     sn:'123456781',

//     age: 29

// });

// user.save();

// UserModel.findBySn('123456781',function(){})

//使用静态方法

UserModel.findBySn('123456782',function(err,docs){

    if(err){

        console.log(err);

        return;

    }

    console.log(docs)

})

var user = new UserModel({

    name: '赵六',

    sn:'123456781',

    age: 29

});

// user.save();

//使用实例方法

user.print();   //赵六

## 数据校验

·required

- 表示这个数据必须传入

·max

- 用于 Number 类型数据，最大值

·min

- 用于 Number 类型数据，最小值

·enum

- 枚举类型，要求数据必须满足枚举值

- 只能用在string类型

- enum: ['0', '1', '2']

·match

- 增加的数据必须符合

- 只能用在string类型

- match（正则）的规则

·maxlength

- 最大长度

·minlength

- 最小长度

var mongoose=require('./db.js');

//mongoose数据校验:用户通过mongoose给mongodb数据库增加数据的时候，对数据的合法性进行的验证

//mongoose里面定义Schema:字段类型，修饰符、默认参数 、数据校验都是为了数据库数据的一致性

//Schema，为数据库对象的集合,每个schema会映射到mongodb中的一个collection,定义Schema可以理解为表结构的定义

var UserSchema=mongoose.Schema({

    name:{

        type:String,//指定类型

        trim:true,   //修饰符

        required:true //name必须传入

    },

    sn:{

        type:String,

        index:true,  //索引.

        set(val){  //自定义修饰符

            return val;

        },

        // maxlength:20,//设置最大长度

        // minlength:10//设置最小长度

        // match:/^sn(.\*)/ ,//必须符合此正则表达式 只能用于string类型

        validate: function(sn) {//自定义验证器

            return sn.length >= 10;//长度必须大于等于10

        }

    },

    age:{

        type:Number,

        min:0,    //age的最小值，用在number类型上面

        max:150 //age的最大值

    },

    status:{

        type:String,

        default:'success', //默认值

        enum:['success','error']   //表示status的值必须为success或error 注意：枚举只能用在string类型

    }

})

module.exports=mongoose.model('User',UserSchema,'user');

## MongoDB 聚合管道

·使用聚合管道可以对集合中的文档进行变换和组合

·实际项目

- 表关联查询、数据的统计

/\*查询order\_item，找出商品名称是酸奶的商品，酸奶这个商品对应的订单的订单号以及订单的总价格\*/

var OrderItemModel=require('./model/order\_item.js');

var OrderModel=require('./model/order.js');

var mongoose=require('mongoose');

//第一种实现方式

    // //通过\_id找到酸奶

    // OrderItemModel.find({"\_id":"5b743da92c327f8d1b360546"},function(err,docs){

    //     // console.log(docs);

    //     //先解析成json数据，在解析成ｊｓ对象

    //     var order\_item=JSON.parse(JSON.stringify(docs));

    //     //获取查询到的order\_id字段值

    //     var order\_id=order\_item[0].order\_id;

    //     //通过order\_id关联两表

    //     OrderModel.find({"order\_id":order\_id},function(err,order){

    //         //    console.log(order);

    //         //将获取到的结果复制给主表的order\_info字段

    //         order\_item[0].order\_info=order[0];

    //         console.log(order\_item)

    //     })

    // })

//第二种方式

OrderItemModel.aggregate([

    {

      $lookup:

        {

          from: "order",//关联的表

          localField: "order\_id",//主表的关联字段

          foreignField: "order\_id",//附表的关联字段

          as: "order\_info"//将查询到的结果返回给order\_info字段

        }

   },

  　{

    //mongoose中获取ObjectId           mongoose.Types.ObjectId

    $match:{\_id: mongoose.Types.ObjectId('5b743da92c327f8d1b360546')}//查询到酸奶的数据

  　}

],function(err,docs){

        if(err){

            console.log(err)

            return;

        }

        //查看查询到的数据

        console.log(JSON.stringify(docs))

})