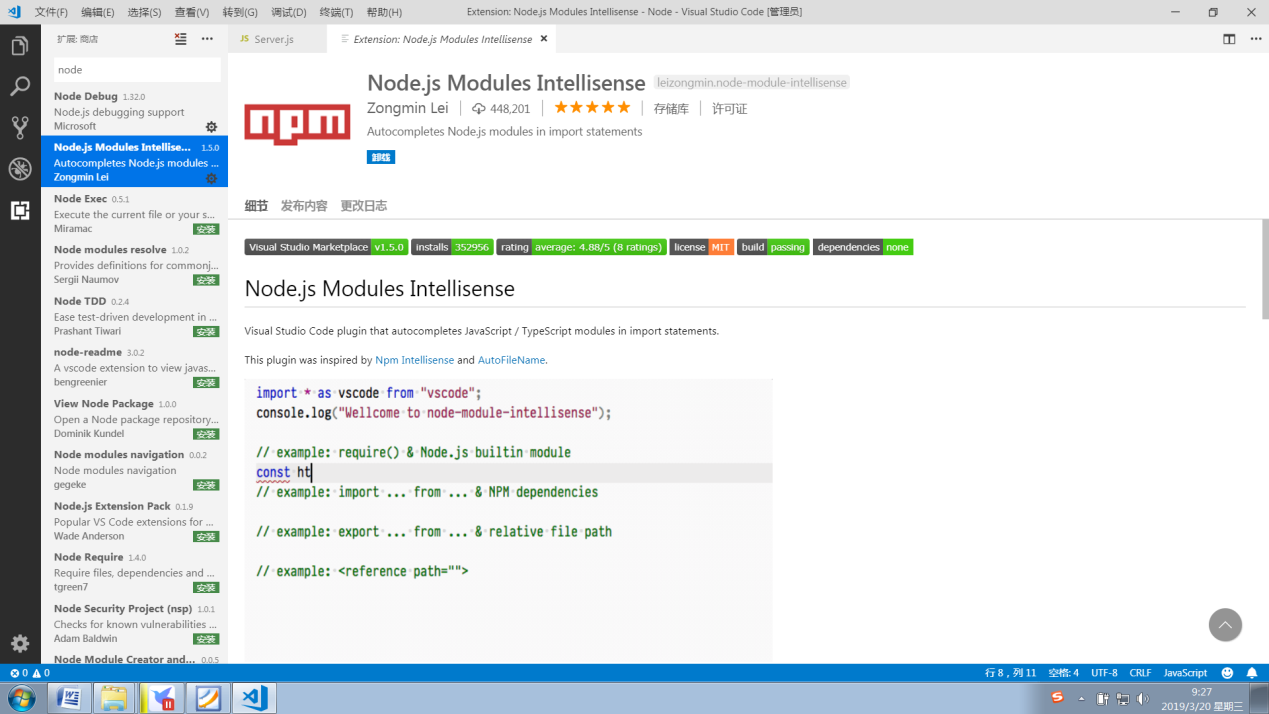
Node

1. 下载node 安装node环境
2. 使用vscode开发 简单配置开发环境

安装Nodejs开发插件



1. 开始使用Nodejs Http模块 url模块 和nodejs自启动工具supervisor

Nodejs创建第一个应用：

创建http服务器：

/\*\*

\* Created by Administrator on 2019/3/20 0020.

\*/

/\*

\* 1.引入http请求

\*

\* 2.创建http服务器

\* 参数1，request 获取url信息

参数2，response 浏览器返回响应信息

\* \*/

var http=require('http');

http.createServer(function (req,res){

// 设置http头部 状态 文件类型 字符集

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

//发送响应数据

res.write("我是node js");

//设置监听的端口号

//请求完成之后结束响应

res.end();

}).listen(8100);

console.log("http://192.168.0.253:8100");

url模块

/\*

url 模块

引入url 模块

var url=require('url');

方法：

url.parse() 解析url地址

\*/

var url=require('url');

var src='http://www.baidu.com?name=maodou&sex=0';

//解析地址

//参数1位解析的路径 参数2：可以将地址上的值解析出来

var path=url.parse(src,true);

console.log(path);

使用http模块+url模块解析路径上的值

var http=require("http");

var url=require("url");

http.createServer(function (req,res){

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

//解析路径输出路径上的值

var path=url.parse(req.url,true);

res.write(path.query.id);

res.end();

}).listen(8100);

1. 在nodejs中文件修改需要重启服务：这要开发比较慢，安装自启动工具supervisor

npm install –g supervisor

安装完成之后启动文件使用 supervisor \*\*\*\*.js

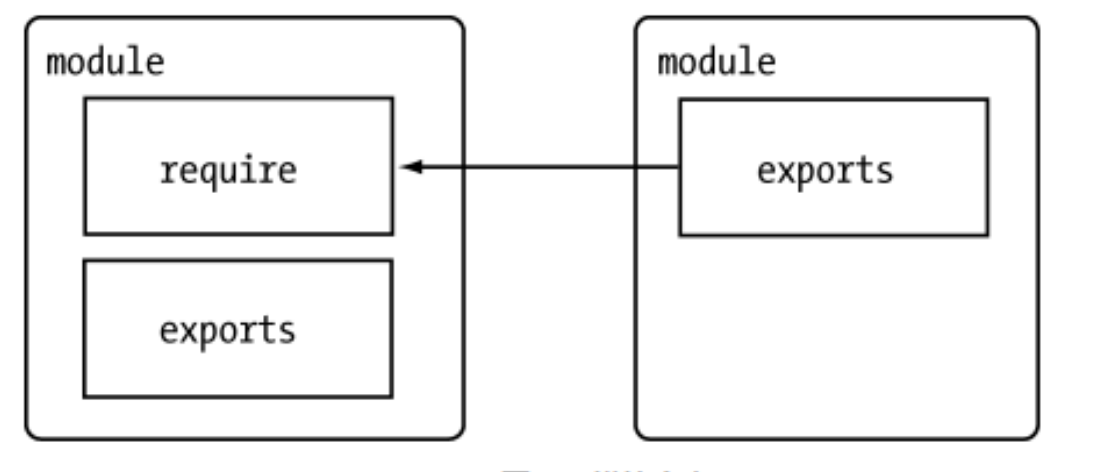
启动服务之后修改代码会自动启动服务

1. Commonjs 和Nodejs模块

Commonjs nodejs中的自定义模块

Commonjs规范的提出，弥补javascript没有标准的缺陷，提供一个类似后端语言的标准库，也就是说commonjs是模块化的标准，nodejs就是commonjs模块化的实现。在nodejs中除了http，url，fs等等都是nodejs的内置模块，可以直接使用，commonjs中自定义模块的实现：

在nodejs中将公共的功能抽离为单独的js文件作为模块，在外部是没有办法访问的（类似后端的私有的属性和方法）;要想使用模块，就必须在模块里面通过exports或者module.exports暴露属性或者方法。在需要的模块使用require引入模块。



Server.js

var http=require('http');

var ser=http.createServer(function (req,res){

// 设置http头部 状态 文件类型 字符集

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

//发送响应数据

res.write("我是node js");

//设置监听的端口号

//请求完成之后结束响应

res.end();

}).listen(8100);

console.log("http://192.168.0.253:8100");

//两种方式导出都可以

// exports.ser=ser;//导出模块

module.exports=ser;

common.js

/\*

在js文件里面引入Server.js模块文件

\*/

var server=require("./Server.js");//在当前的目录里面找文件 也可以省略.js后缀

todo.js

var stu={

name:"毛豆",

setwork:function (){

return "我在跑nodejs"

}

}

module.exports=stu;

common2.js

var s=require("./todo");

console.log(s.name);

console.log(s.setwork());

list.js

var stu={

name:"毛豆",

setwork:function (){

return "我在跑nodejs"

}

}

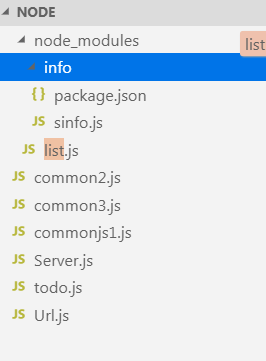
module.exports=stu;

common3.js

var s=require("./node\_Module/list");

console.log(s);

nodejs 可以自动找node\_modules文件下的文件 ：如果node\_modules文件下有文件夹 可在cd进入这个文件 使用命令npm init –yes 安装当前文件的package.json文件，直接请求require(“名字”);



6 ) 1 nodejs 第三方包的安装 和package.json 项目文件

创建 node项目之后先使用命令进入项目文件，安装项目的配置文件package.json文件。

使用命令 npm init /cnpm init 或者 npm init –yes/cnpm init –yes （这种表示严格模式命名）

{

"name": "nodedemo",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "",

"license": "ISC"

}

Nodejs的第三方包的使用： 第三方包使用建议官网：www.npmjs.com

比如第三方包： md5-node包

安装包：*npm install md5-node –save*

安装完之后项目里面生成一个node\_modules依赖文件

使用：

//使用第三方包

var md5=require('md5-node');

console.log(md5('123456')); //e10adc3949ba59abbe56e057f20f883e

1. nodejs 中的fs模块文件系统的使用

读取文件：

var fs=require("fs");

// Node.js 文件系统（fs 模块）模块中的方法均有异步和同步版本，例如读取文件内容的函数有异步的 fs.readFile() 和同步的 fs.readFileSync()。

// 异步的方法函数最后一个参数为回调函数，回调函数的第一个参数包含了错误信息(error)。

// 建议大家使用异步方法，比起同步，异步方法性能更高，速度更快，而且没有阻塞

//异步读取

fs.readFile("content.txt",function (err,data){

if(err)

{

console.log(err);

return false;

}

console.log(data.toString());

});

//同步读取

var data=fs.readFileSync("content.txt");

console.log(data.toString());

获取文件信息 判断是否是目录还是文件：

//读取文件信息

fs.stat("content.txt",function (err,stat){

if(err)

{

console.log(err);

return false;

}

console.log("是否为文件"+stat.isFile());

console.log("是否为目录"+stat.isDirectory());

});

文件的写入 有覆盖 和追加

//异步写入文件 这种文件的写入会覆盖

fs.writeFile("content.txt","我在测试写入文件",function (err){

if(err)

{

console.log(err);

}

});

//追加文件的写入

var txt="追加文件的写入";

fs.appendFile("content.txt",txt,function (err){

if(err)

{

console.log(err);

}

});  
  
创建目录

//创建目录 如果目录已经存在不在创建 一般用于图片的上传与下载上

fs.mkdir("./upload",function (err){

if(err)

{

console.log(err);

return false;

}

console.log("创建upload目录成功！");

});

读取目录：

//读取目录

fs.readdir("upload",function (err,files){

if(err)

{

console.log(err);

return false;

}

//打印出当前目录里面的所有目录

console.log(files);

});

读取固定目录，判断里面的所有子目录是文件还是文件夹：

//读取目录

var filesinfo="upload";

fs.readdir(filesinfo,function (err,files){

if(err)

{

console.log(err);

return false;

}

//打印出当前目录里面的所有目录

for(var file in files)

{

//注意这里面的方法为异步执行

fs.stat(filesinfo+"/"+files[file],function (error,stat){

if(error)

{

console.log(error);

return false;

}

console.log(files[file],stat.isDirectory()?"是目录":"是文件");

})

}

});

1. nodejs中的stream流

Stream 是一个抽象接口，Node 中有很多对象实现了这个接口。例如，对http 服务器发起请求request 对象就是一个 Stream，还有stdout（标准输出）。

Node.js，Stream 有四种流类型：

**Readable** - 可读操作。

**Writable** - 可写操作。

**Duplex** - 可读可写操作.

**Transform** - 操作被写入数据，然后读出结果。

所有的 Stream 对象都是 EventEmitter 的实例。常用的事件有：

**data** - 当有数据可读时触发。

**end** - 没有更多的数据可读时触发。

**error** - 在接收和写入过程中发生错误时触发。

**finish** - 所有数据已被写入到底层系统时触发。

从流中读取数据

//从流中读取数据

var fs=require("fs");

//设置读取的数据

var data="";

// 创建流

var readStream=fs.createReadStream("content.txt");

//设置编码格式

readStream.setEncoding("UTF-8");

//处理流的事件

//当有数据可读时触发

readStream.on("data",function (res){

data+=res;

});

//没有数据可读时触发

readStream.on("end",function (){

console.log(data);

});

//在读取时发生错误时触发

readStream.on("error",function (err){

console.log(err);

});

console.log("读取完成");

写入流

//创建写入流

var text="我在给文件写字";

var writerStream=fs.createWriteStream("content.txt");

//设置写入的编码

//在这里可以不断的写入文件

writerStream.write(text,"UTF-8");

//写入完成

writerStream.end();

//监听事件

writerStream.on("finish",function (){

console.log("写入完成");

});

管道刘

//管道流 意思是读取流直接到写入流

// 创建流

var readStream1=fs.createReadStream("content.txt");

var writerStream1=fs.createWriteStream("data.txt");

readStream1.pipe(writerStream1);

1. 使用http模块 url模块 fs模块 path模块创建一个静态的web服务器

静态web服务器的目录结构：



server.js

//导入http模块

var http=require('http');

//导入fs模块

var fs=require("fs");

//导入path模块

var path=require("path");

var url=require("url");

var Exname=require("./Extname.js");

http.createServer(function (req,res){

//在这里可以对路径进行解析 获取pathname 这样可避免get传值之类的

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

//把无效的请求过滤

if(pathname!="/favicon.ico")

{

if(pathname=="/")

{

pathname="index.html";//默认

}

//根据目录找到对应的文件

fs.readFile("static/"+pathname,function (error,data){

if(error)

{

console.log("404");

//在页面404时可以请求404页面

return false;

}

//现在数据可以打印出去 但是文件的类型不一致 text/html text/css text/javascript 等等不一致

//根据后缀名 确认类型

var aname=path.extname(pathname);

var exname=Exname(aname);

res.writeHead(200,{"Content-Type":""+exname+";charset=UTF-8"});

res.write(data);

res.end();

});

}

}).listen(8100);

Extname.js

var mim=require("./Mime");

module.exports=function (exname){

return mim[exname];

// var type="";

// switch(exname)

// {

// case ".html": type="text/html";break;

// case ".css": type="text/css";break;

// case ".js": type="text/javascript";break;

// case ".json": type="text/json";break;

// default : type="text/html";break;

// }

// return type;

}

Mime.js

var fs=require("fs");

var data=null;

//读取文件

//异步获取数据暂时外边拿不到

// fs.readFile("mime.json",function (err,res){

// if(err)

// {

// console.log("404");

// return false;

// }

// data=JSON.parse(res.toString());

// });

data=JSON.parse(fs.readFileSync("mime.json").toString());

module.exports=data;

static 文件放入静态的资源

1. Nodejs 非阻塞的IO、异步以及事件驱动EventEmitter 处理异步

Nodejs单线程 非阻塞IO 事件驱动

Nodejs回调处理异步

Nodejs events模块处理异步

1. 回调函数处理异步 （非阻塞）

var fs=require("fs");

//阻塞请求数据 等待请求完成在执行后续代码

var data=fs.readFileSync("mime.json");

console.log(data.toString());

// 非阻塞式请求代码 nodejs中多数方法为异步请求方式

fs.readFile("mime.json",function (err,res){

if(err) throw err;

console.log(res.toString());

});

console.log(1);

//处理异步请求 外边可以获取到请求的数据的方式

//1.使用回调函数来获取数据

function getMime(callback){

fs.readFile("mime.json",function (err,res){

if(err) throw err;

callback(res.toString());

});

}

console.log(1);

//这样就可以在外部获取异步的数据 也可以暴露出去别的文件使用

module.exports=getMime(function (data){

console.log(data);

});

1. 使用nodejs内置模块事件的驱动 events 来进行数据的广播和监听

var fs=require("fs");

var event=require("events");

//实例化事件的对象

var EventEmitter=new event.EventEmitter();

//使用events模块 获取异步的数据

//在请求完数据之后进行广播

fs.readFile("mime.json",function (err,res){

if(err) throw err;

EventEmitter.emit("mime",res.toString());

});

//在外部进行监听广播

EventEmitter.on("mime",function (data){

console.log(data);

});

在不同的文件之间广播和监听使用都可以获取数据

1. Web服务器静态文件托管，GET POST路由和EJS模板引擎
2. 对web静态服务器的封装

//导入http模块

var http=require('http');

var router=require("./router.js");

http.createServer(function (req,res){

router(req,"static",res);

}).listen(8100);

Router

//导入fs模块

var fs=require("fs");

//导入path模块

var path=require("path");

var url=require("url");

var Exname=require("./Extname.js");

module.exports=(req,static,res)=>{

//在这里可以对路径进行解析 获取pathname 这样可避免get传值之类的

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

//把无效的请求过滤

if(pathname!="/favicon.ico")

{

if(pathname=="/")

{

pathname="index.html";//默认

}

//根据目录找到对应的文件

fs.readFile(static+"/"+pathname,function (error,data){

if(error)

{

console.log("404");

//在页面404时可以请求404页面

return false;

}

//现在数据可以打印出去 但是文件的类型不一致 text/html text/css text/javascript 等等不一致

//根据后缀名 确认类型

var aname=path.extname(pathname);

var exname=Exname(aname);

res.writeHead(200,{"Content-Type":""+exname+";charset=UTF-8"});

res.write(data);

res.end();

});

}

}

1. node 路由 简单路由配置

var http=require("http");

var url=require("url");

http.createServer(function (req,res){

var path=url.parse(req.url,true).pathname;

if(path=="/")

{

res.end("index.html");

}

else if(path=="/login")

{

res.end("login.html");

}

else{

res.end("regest.html");

}

}).listen(8100);

1. EJS模板引擎 （初步了解 在express框架讲）

EJS模板是后台模板，可以把数据库和文件读取的数据显示到html页面上，是一个第三方模块，需要安装 在项目的目录里面

npm install ejs –save/cnpm install ejs –save

官网：<https://ejs.bootcss.com/>

1. Nodejs 里面的get post 传值

Get 传值直接是在url的后边传值 （不安全） 获取get传值的方法是url模块解析路径获取传值；

POST获取值 是通过监听事件来获取数据：

var http=require("http");

var url=require("url");

//使用ejs模板

var ejs=require("ejs");

var fs= require("fs");

http.createServer(function (req,res){

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

var path=url.parse(req.url,true).pathname;

if(path=="/")

{

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

if(err) throw err;

res.end(data);

});

}

else if(path=="/login")

{

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

if(err) throw err;

res.end(data);

});

}

else if(path=="/dologin")

{

//在登录成功这里 获取传值 get可以直接解析路径获取值

//post 监听事件获取值

let info="";

req.on("data",(ck)=>{

info+=ck;

});

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

fs.appendFile("login.txt",info,(err)=>{

if(err) throw err;

console.log("写入成功！");

});

});

res.end("登录成功操作");

}

else{

res.end("regest.html");

}

}).listen(8100);

1. Nodejs 路由模块的封装 代码的优化 提取路由模块到外部进行优化代码

//使用ejs模板

var ejs=require("ejs");

var fs= require("fs");

let router={

'home':(req,res)=>{

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

if(err) throw err;

res.end(data);

});

},

"login":(req,res)=>{

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

if(err) throw err;

res.end(data);

});

},

"dologin":(req,res)=>{

//在登录成功这里 获取传值 get可以直接解析路径获取值

//post 监听事件获取值

let info="";

req.on("data",(ck)=>{

info+=ck;

});

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

fs.appendFile("login.txt",info,(err)=>{

if(err) throw err;

console.log("写入成功！");

});

});

res.end("登录成功操作");

}

}

module.exports=router;

1. Express方式封装路由模块

网址：<http://www.expressjs.com.cn/guide/routing.html>

var http=require("http");

var app=require("./Module/router");

//创建服务

//现在对于初始的http模块按照路由模块化进行封装

http.createServer(app).listen(8100);

// http.createServer(function (req,res){

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

// //中间写服务

// res.end("服务启动");

// }).listen(8100);

var url=require("url");

var obj={};

//封装方法

var app=function (req,res){

var path=url.parse(req.url).pathname;

//执行 obj 中方法

if(path=="/favicon.ico")return;

console.log(obj);

if(obj[path])

{

obj[path](req,res);

}

else{

res.end("404");

}

}

//在app方法上挂在方法

app.get=function (string,callback){

obj[string]=callback;

}

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

res.end("login");

});

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

res.end("/login/info");

});

module.exports=app;

1. 封装类似express模块化路由 get和post 处理路由和数据

Router.js

var url=require("url");

//封装路由模块

module.exports=(()=>{

var obj=this;

this.\_get={};

this.\_post={};

var app=(req,res)=>{

res.send=(data)=>{

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

res.end(data);

}

var urlpath=url.parse(req.url,true);

var pathname=urlpath.pathname;

var method=req.method.toLowerCase();

if(obj['\_'+method][pathname])

{

if(method=="post")

{

var list="";

req.on("data",(ck)=>{

list+=ck;

});

req.on("end",(err,ck)=>{

if(err) throw err;

req.body=list;

obj['\_'+method][pathname](req,res);

})

}

else{

req.body=urlpath.query;

obj['\_'+method][pathname](req,res);

}

}

else{

res.end("404");

}

}

//封装get 和 post

app.get=(string,callback)=>{

obj.\_get[string]=callback;

}

app.post=(string,callback)=>{

obj.\_post[string]=callback;

}

return app;

})();

Server.js

var http=require("http");

var app=require("./module/router.js");

var ejs=require("ejs");

http.createServer(app).listen(8100);

app.get("/index",function (req,res){

ejs.renderFile("view/index.ejs",{msg:"首页"},(err,data)=>{

if(err) throw err;

res.send(data+JSON.stringify(req.body));

});

});

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

ejs.renderFile("view/login.ejs",{},(err,data)=>{

if(err) throw err;

res.send(data);

});

});

app.post("/dologin",function (req,res){

ejs.renderFile("view/dologin.ejs",{},(err,data)=>{

if(err) throw err;

res.send(data+JSON.stringify(req.body));

});

});

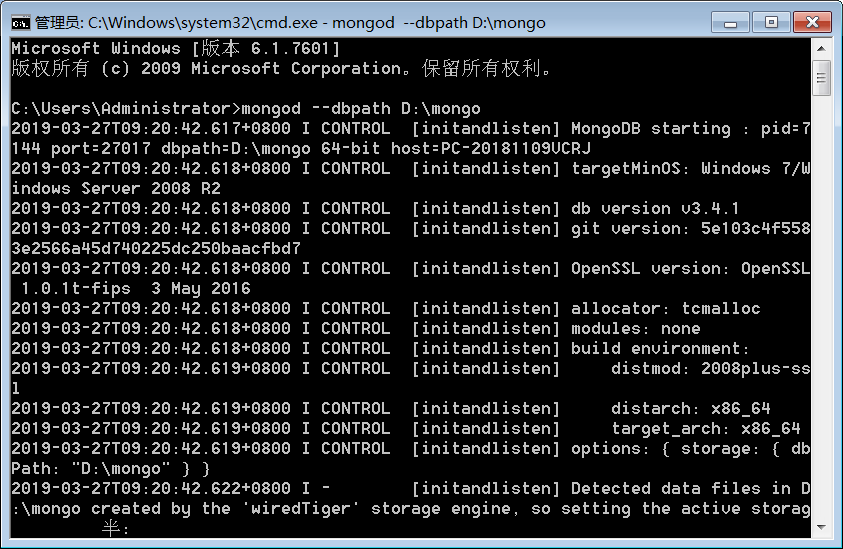
1. MongoDb数据库的介绍 安装使用

官网下载 安装 安装完成之后配置环境变量就可以使用

安装目录c盘-到bin文件 服务目录 在系统环境变量path中配置

在cmd输入mongo 只要不是内部命令就配置完成

启动mongo服务 mongod --dbpath 建立的文件路径



1. MongoDb 数据库增删改查

显示mongoDb里面的数据库 使用 show dbs

显示数据库当中的表（集合） show collections

删除数据库： db.dropDatabase()

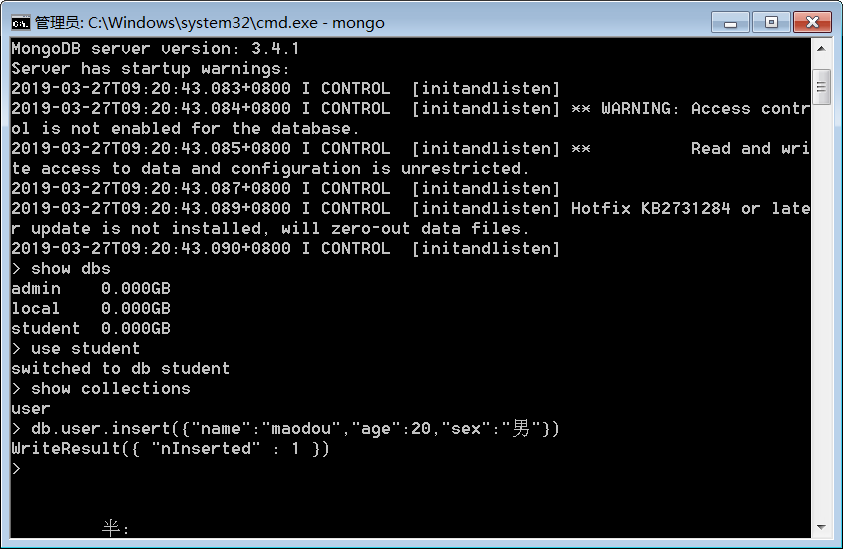
删除集合：db.user.drop()

创建数据库 （开始数据库没有，使用数据库 给库里面添加一条数据，库就生成了）

use student

给表中插入数据

db.user.insert({“name”:”maodou”}) 集合中的列可以添加的不一样



查找数据

db.user.find() 相当于select \* from user

查找去重复之后的数据（某列）

条件查询

db.user.find({‘age’:20}) 相当于select \* from user where age=20

查找大于

db.user.find({‘age’:{$gt:20}}) 大于 大于等于db.user.find({‘age’:{$gte:20}})

查找小于

db.user.find({‘age’:{$lt:20}}) 小于 小于等于db.user.find({‘age’:{$lte:20}})

查找范围数据

Db.user.fin({“age”:{$lte:20,$gte:30}}) 小于等于30大于等于20

模糊查询

Db.user.find({“name”:/go/}) 可用于搜索

查询指定列的数据

db.user.find({},{name:1,age:1}) 1相当true false除过查其他列

前面的{}写查找的条件

查找数据升序或者降序

Db.user.find().sort({“age”:1}) 1升序

Db.user.find().sort({“age”:-1}) -1降序

查询前5条数据

Db.user.find().limit(5)

查询10条以后的数据

Db.user.find().skip(10)

查询5-10条之间的数据

Db.user.find().limit(10).skip(5) 可用来做数据分页

查询某个结果的记录条数

Db.user.find().count()

修改数据

db.user.update({“name”:”maodou”,{$set:{“age”:”20”}}})

如果不写$set 会将整行数据替换

数据的删除

Db.user.remove({“name”:”maodou”}); 中间按照条件删除

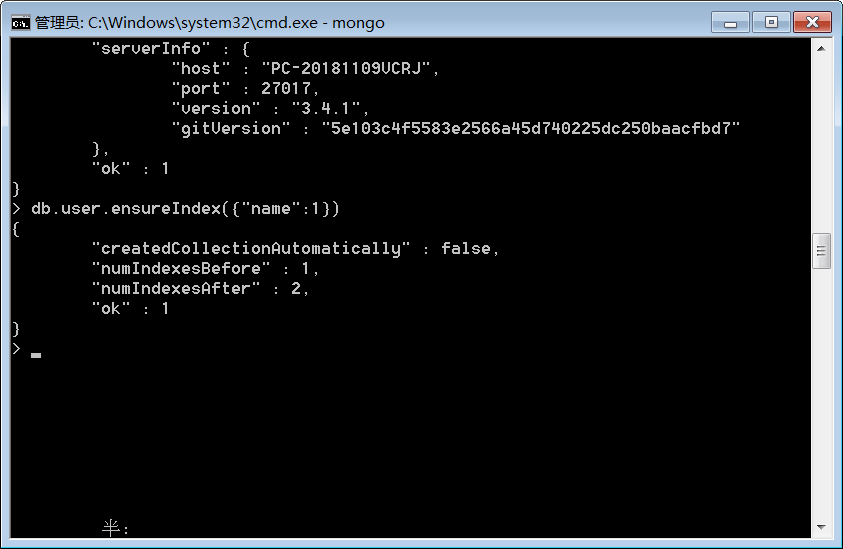
如果数据重复可以删除一条

Db.user.remove({“name”:”maodou”},{justOne:true}) 删除一条数据

1. MongoDb的索引和explain（主要检测sql的执行速度）的使用

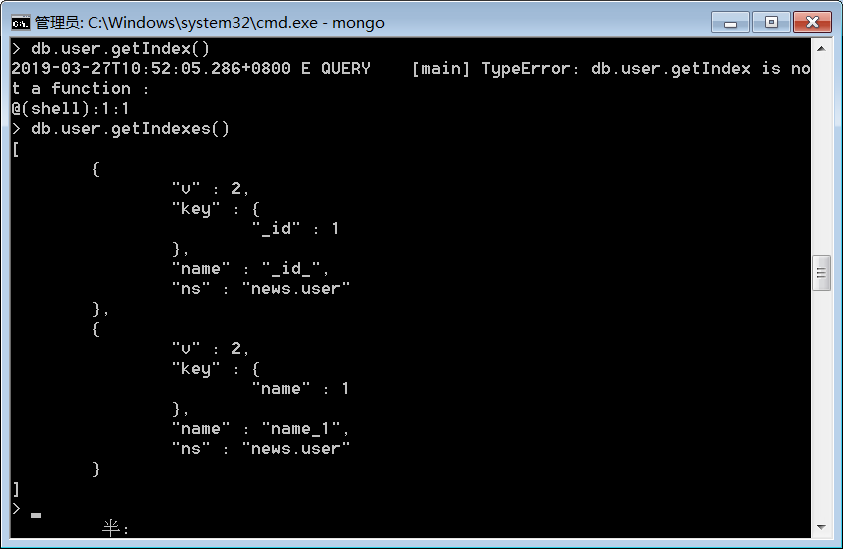
创建索引

Db.user.ensureIndex({“name”:1}) 1和-1为升序降序



获取索引

Db.user.getIndexes()



删除索引

Db.user.dropIndex({“name”:1})

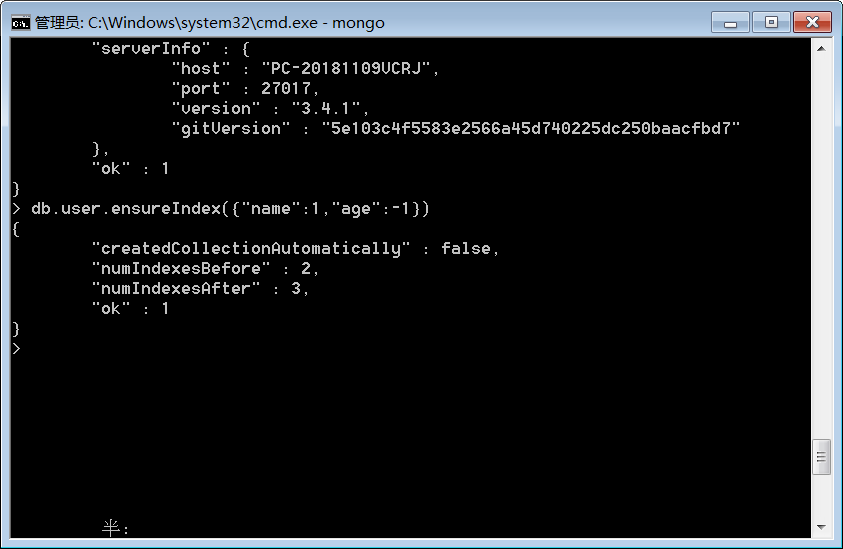
创建一个1000万条数据的数据库 查找数据 检查执行速度

\*\*\*\*.explain(“executionStats”)

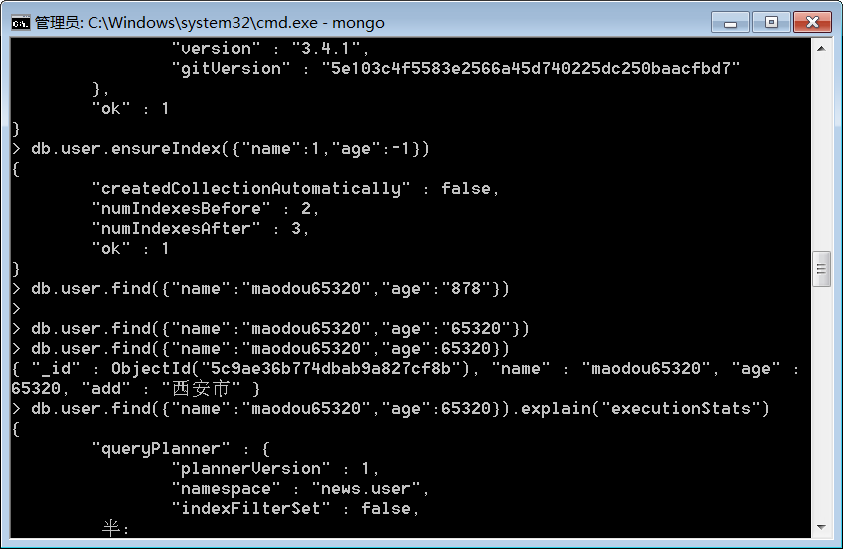


给查找的列添加索引 大幅度减少查找的速度

多列查询创建复合索引



多列查找速度更快



设置唯一索引 unique 存储的值不能重复

Db.user.ensureIndex({“id”:1},{“unique”:true})

如果插入重复的值mongodb会报错

也可以添加复合唯一索引

Db.user.ensureIndex({“id”:1,”name”:1},{“unique”:true})

如果在为已有数据的文档创建索引，可以执行下面的命令，使mongodb在后台创建索引，这样就不会阻塞其他操作

Db.user.ensureIndex({“id”:1},{“background”:true})

17) nodejs 操作mongodb数据库的方法

1） 安装mongodb

cnpm install mongodb –save

2 ) 引入mongodb下面的MongoClient

var MongoClient = require('mongodb').MongoClient;

代码 ---链接数据库创建表

// 1.创建mongoClient 对象

var MongoClient=require("mongodb").MongoClient;

//2.定义数据库的链接地址和数据库名称

var url="mongodb://localhost:27017";

var dbname="news";

MongoClient.connect(url,function (err,client){

if(err) throw err;

console.log("数据库创建完成");

var dbase=client.db(dbname);//获取db对象

//创建表

dbase.createCollection("sorce",function (err,res){

if(err) throw err;

console.log("创建集合成功！");

client.close();//关闭

});

});

------表中增加单条数据

// 1.创建mongoClient 对象

var MongoClient=require("mongodb").MongoClient;

//2.定义数据库的链接地址和数据库名称

var url="mongodb://localhost:27017";

var dbname="news";

MongoClient.connect(url,function (err,client){

if(err) throw err;

console.log("数据库创建完成");

var dbase=client.db(dbname);//获取db对象

//创建表

// dbase.createCollection("sorce",function (err,res){

// if(err) throw err;

// console.log("创建集合成功！");

// client.close();//关闭

// });

//集合中添加数据

var data={name:"毛豆",math:120};

dbase.collection("sorce").insertOne(data,function (err,res){

if(err) throw err;

console.log("添加数据成功！");

client.close();

});

});

------表中增加多条数据

// 1.创建mongoClient 对象

var MongoClient=require("mongodb").MongoClient;

//2.定义数据库的链接地址和数据库名称

var url="mongodb://localhost:27017";

var dbname="news";

MongoClient.connect(url,function (err,client){

if(err) throw err;

console.log("数据库创建完成");

var dbase=client.db(dbname);//获取db对象

//表中增加多条数据

var data=[

{name:"毛豆",math:120},

{name:"毛豆",math:120},

{name:"毛豆",math:120}

];

dbase.collection("sorce").insertMany(data),function (err,res){

if(err)throw err;

console.log("增加成功！");

client.close();

}

});

1. 使用路由模块和 nodej服务器进行mongodb数据库的增删改查

var http=require("http");

var app=require("./module/express-router");

//连接数据库

var MongoClient=require("mongodb").MongoClient;

//mongodb的服务器地址

var url="mongodb://127.0.0.1:27017/news"

http.createServer(app).listen(8100,"127.0.0.1");

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

res.send("login");

});

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

res.send("user");

});

//创建集合

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

MongoClient.connect(url,{ useNewUrlParser: true },(err,result)=>{

if(err) throw err;

var resbase=result.db("news");

resbase.createCollection("news",(error,resdata)=>{

if(error) throw error;

res.send("创建成功！");

result.close();

});

});

});

//添加数据

app.get("/add",function (req,res){

MongoClient.connect(url,(err,result)=>{

if(err) throw err;

var resbase=result.db("news");

//给表news里面添加数据

resbase.collection("news").insertOne({

title:"nodejs",

age:20

},(error,data)=>{

if(error) throw error;

res.send("增加数据成功");

result.close();

});

});

});

app.get("/usermore",function (req,res){

MongoClient.connect(url,(err,result)=>{

if(err) throw err;

var resbase=result.db("news");

//给表news里面添加多条数据数据

var more=[

{

title:"nodejs",

age:20

},

{

title:"Vue",

age:20

}

]

resbase.collection("news").insertMany(more,(error,data)=>{

if(error) throw error;

res.send("增加数据成功");

result.close();

});

});

});

app.get("/delete",function (req,res){

MongoClient.connect(url,(err,result)=>{

if(err) throw err;

var resbase=result.db("news");

//删除表里一条数据

var base={age:20};

resbase.collection("news").deleteOne(base,(error,data)=>{

if(error) throw error;

res.send("删除数据成功");

result.close();

});

});

});

app.get("/deletemore",function (req,res){

MongoClient.connect(url,(err,result)=>{

if(err) throw err;

var resbase=result.db("news");

//删除表里多条数据

var base={age:20};

resbase.collection("news").deleteMany(base,(error,data)=>{

if(error) throw error;

res.send("删除多条数据成功");

result.close();

});

});

});

//排序

app.get("/paixu",function (req,res){

MongoClient.connect(url,{ useNewUrlParser: true },(err,result)=>{

if(err) throw err;

var resbase=result.db("news");

//根据表里面的数据排序 1 升序 -1 降序

var base={age:1};

resbase.collection("news").find({}).sort(base).toArray((error,data)=>{

if(error) throw error;

res.send(data.toString());

result.close();

});

});

});

// 查询分页

app.get("/page",function (req,res){

MongoClient.connect(url,{ useNewUrlParser: true },(err,result)=>{

if(err) throw err;

var resbase=result.db("news");

//数据分页

resbase.collection("news").find({}).limit(5).toArray((error,data)=>{

if(error) throw error;

res.send(data.toString());

result.close();

});

});

});

// skip(): 跳过前面两条数据，读取两条数据

//可以制作数据分页

app.get("/pageskip",function (req,res){

MongoClient.connect(url,{ useNewUrlParser: true },(err,result)=>{

if(err) throw err;

var resbase=result.db("news");

//数据分页

resbase.collection("news").find({}).skip(5).limit(5).toArray((error,data)=>{

if(error) throw error;

res.send(data.toString());

result.close();

});

});

});

//删除集合

app.get("/dropnews",function (req,res){

MongoClient.connect(url,{ useNewUrlParser: true },(err,result)=>{

if(err) throw err;

var resbase=result.db("news");

//数据分页

resbase.collection("news").drop((error,data)=>{

if(error) throw error;

res.send("删除完成");

result.close();

});

});

});

//修改数据

app.get("/update",function (req,res){

MongoClient.connect(url,{ useNewUrlParser: true },(err,result)=>{

if(err) throw err;

var resbase=result.db("news");

//数据分页

var info={age:20};

var up={$set:{name:"recat"}};

resbase.collection("news").updateOne(info,up,(error,data)=>{

if(error) throw error;

res.send("修改完成");

result.close();

});

});

});

//修改多条

app.get("/updatemore",function (req,res){

MongoClient.connect(url,{ useNewUrlParser: true },(err,result)=>{

if(err) throw err;

var resbase=result.db("news");

//数据分页

var info={age:20};

var up={$set:{name:"recat"}};

resbase.collection("news").updateMany(info,up,(error,data)=>{

if(error) throw error;

res.send("修改多条完成");

result.close();

});

});

});

1. 操作mongodb 数据库模块的封装

//操作数据库

//链接数据库

var MongoClient=require("mongodb").MongoClient;

var setting=require("./settings");

var connect=(callback)=>{

MongoClient.connect(setting.url,{ useNewUrlParser: true },(error,client)=>{

if(error){

console.log(error);

return;

}

callback(error,client);

});

}

exports.insertOne=(tablename,json,callback)=>{

connect((error,client)=>{

//进行增加一条数据

var dbbase=client.db(setting.name);

dbbase.collection(tablename).insertOne(json,(err,res)=>{

if(err)

{

console.log(err);

return;

}

callback(err,res);

client.close();

});

});

}

exports.insertMany=(tablename,json,callback)=>{

connect((error,client)=>{

//进行增加多条数据

var dbbase=client.db(setting.name);

dbbase.collection(tablename).insertMany(json,(err,res)=>{

if(err)

{

console.log(err);

return;

}

callback(err,res);

client.close();

});

});

};

exports.findMore=(tablename,json,callback)=>{

connect((error,client)=>{

//数据查询 支持条件查询

var dbbase=client.db(setting.name);

dbbase.collection(tablename).find(json).toArray((err,res)=>{

if(err)

{

console.log(err);

return;

}

callback(err,res);

client.close();

});

});

}

exports.updateOne=(tablename,json,update,callback)=>{

connect((error,client)=>{

//修改单条数据

var dbbase=client.db(setting.name);

dbbase.collection(tablename).updateOne(json,update,(err,res)=>{

if(err)

{

console.log(err);

return;

}

callback(err,res);

client.close();

});

});

};

exports.updateMany=(tablename,json,update,callback)=>{

connect((error,client)=>{

//修改多条数据

var dbbase=client.db(setting.name);

dbbase.collection(tablename).updateMany(json,update,(err,res)=>{

if(err)

{

console.log(err);

return;

}

callback(err,res);

client.close();

});

});

};

exports.deleteone=(tablename,json,callback)=>{

connect((error,client)=>{

//删除一条数据

var dbbase=client.db(setting.name);

dbbase.collection(tablename).deleteone(json,(err,res)=>{

if(err)

{

console.log(err);

return;

}

callback(err,res);

client.close();

});

});

};

exports.deletemore=(tablename,json,callback)=>{

connect((error,client)=>{

//删除多条数据

var dbbase=client.db(setting.name);

dbbase.collection(tablename).deleteMany(json,(err,res)=>{

if(err)

{

console.log(err);

return;

}

callback(err,res);

client.close();

});

});

};

1. 使用mongodb模块和http 模块和模拟的express路由 结合ejs模板 获取数据动态渲染数据

var http=require("http");

var mongo=require("./module/mongodb");

var ejs=require("ejs");

http.createServer((req,respone)=>{

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

mongo.findMore("user",{

},(err,res)=>{

if(err)

{

respone.end("增加失败");

return;

}

ejs.renderFile("./html/index.ejs",{item:res},(err,str)=>{

if(err){

console.log("ejs渲染失败");

return;

}

respone.end(str);

});

});

}).listen(8100);

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

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

<title>Document</title>

</head>

<body>

<p>111</p>

<ul>

<% for(var i in item){%>

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

<%}%>

</ul>

</body>

</html>

1. express框架介绍 安装 路由 动态路由传值 get传值

express安装

创建pagejson 文件 cnpm init –yes

安装express cnpm install express –save

<http://www.expressjs.com.cn/> express 中文网

安装完成之后直接使用：

//安装espress框架之后

var express=require("express");

var app=new express();

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

res.send("首页");

});

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

res.send("news");

});

app.listen(8100,"127.0.0.1")

在express中路由的配置，get路由

Express中的路由动态传参和动态路由

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

//动态获取get路由传值

var id=req.params.id;

res.send("news"+id);

});

Express中的get传值 类似于[www.baidu.com?id=1&name=2](http://www.baidu.com?id=1&name=2)

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

//路由的get传值在req的query属性上

// console.log(req.query);

var id=req.query.id;

res.send("路由的get传值---"+id);

});

21) express ejs 使用 静态文件的托管

在express框架中使用ejs模板绑定数据

var express=require("express");

var app=new express();

//在express框架中使用ejs模板

//设置views 文件 渲染ejs

//固定写法

app.set("view engine","ejs");

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

//res.send("首页");

//给ejs模板直接渲染 不许要写路径 ejs模板必须放在views文件下

var item=[1,2,3];

res.render("index",{

data:item

});

});

app.listen(8100,"127.0.0.1");

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

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

<title>Document</title>

</head>

<body>

<p>首页</p>

<ul>

<% for(var a in data){%>

<li><%=data[a]%></li>

<%}%>

</ul>

</body>

</html>

//views 文件为默认的模板的位置

//可以修改位置

app.set("views",\_\_dirname+"/statics");

ejs模板的使用：

ejs文件中引入模板页面：

<%-include public/header.ejs%>

ejs中数据绑定

<%=data[a]%>

ejs中绑定html数据

<%-data%>

ejs中数据循环绑定

<ul>

<% for(var a in data){%>

<li><%=data[a]%></li>

<%}%>

</ul>

ejs模板中判断语句

<% if(true){%>

<div>true</div>

<%}else{%>

<div>false</div>

<%}%>

使用express.static托管静态文件也成静态路由

在项目目录建立public文件

//设置静态路由 也是静态资源管理

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

//配置虚拟的静态web服务

//路径/static/image/1.jpg 会去public文件下找

app.use("/static",express.static("public"));

1. express中的中间件

var express=require("express");

var app=new express();

//body-parser第三方中间件的使用

//1.安装body-parser cnpm install body-parser --save

//2.引入

var bodyParser = require('body-parser');

//3.复制官网的代码 www.npmjs.com

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

app.use(bodyParser.json());

//express中使用ejs模板

//设置ejs

app.set("view engine","ejs");

//express中的中间件

//应用级中间件

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

console.log(new Date());

//在中间中 匹配路由之前的中间件执行相关代码 可以做类似权限的判断之类的

//next() 继续向下匹配

next();

});

//express中的路由

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

// res.send("首页");

res.render("index",(err,str)=>{

if(err) throw err;

res.end(str);

});

});

//路由中间件 和应用级中间件类似

//可以挂载函数和对象

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

console.log("路由中间件");

next();

});

//相同

//挂载函数

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

console.log("路由中间件");

next();

});

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

res.send("news");

});

//第三方中间件 例如 前台post传值后台接受

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

res.render("login");

});

//使用第三方中间件 body-parser 获取post传值

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

res.send(JSON.stringify(req.body));

});

//错误中间件

//在匹配路由时，没有可匹配的路由 报出404

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

res.status(404).render("404");

})

app.listen(8100,"127.0.0.1");

login.ejs

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

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

<title>Document</title>

</head>

<body>

<form action="./userlogin" method="POST">

<ul>

<li>账户：<input type="text" name="id"></li>

<li>密码：<input type="password" name="pass"></li>

<li><input type="submit" value="登录"></li>

</ul>

</form>

</body>

</html>

1. express 中间件cookie的使用

cookie 的特点：

cookie保存在浏览器本地，正常设置cookie是不加密的，用户可以删除或者禁用cookie，cookie可以被篡改，可用于攻击，且存储量比较小，可能会被stroage取代

var express=require("express");

var app=new express();

//1.安装cookie cnpm install cookie-parser --save

//2.引入cookie

var cookie=require("cookie-parser");

//3.设置中间件

app.use(cookie());

//4.存储cookie

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

res.cookie("user","maodou",{maxAge:60000});

res.send("设置cookie成功");

});

//5.读取cookie

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

console.log(req.cookies.user);

res.send("获取cookie");

});

app.listen(8100);

cookie 第三个参数domain 设置域名访问cookie 一级域名二级域名

//在设置cookie时，可以设置域名的访问 以及域名例如 baidu.com 二级域名 news.baidu.com maodou.baidu.com 都可以访问这个cookie

//C:\Windows\System32\drivers\etc 修改host文件

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

res.cookie("user","maodou",{maxAge:60000,domain:".maodou.com"});

res.send("设置cookie成功");

});

cookie 第三个参数path 设置cookie在哪个域名下有效

//设置cookie的第三个参数 path属性 规定当前设置的cookie在哪个路径下有效

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

res.cookie("user","maodou",{maxAge:60000,path:"/getcookie"});

res.send("设置cookie成功");

});

//设置cookie的第三个参数 httpOnly 为true 时 只能在后端可以设置 防止js等脚本设置或者攻击

Cookie加密 signed：true 给cookie签名

//添加随机数字或者字符串 根据自身的算法进行加密  
app.use(cookie("12345"));

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

//如果给cookie设置签名 篡改之后的值会被重置为原始值

res.cookie("user","maodou",{maxAge:60000,signed:true});

res.send("设置cookie成功");

});

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

console.log(req.signedCookies);

res.send("获取cookie");

});

使用cookie 实现 类似 你喜欢什么东西？ 或者你浏览过什么？项目案例

var express=require("express");

var app=new express();

var cookie=require("cookie-parser");

app.use(cookie());

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

//读取cookie

var city=req.cookies.city;

console.log(city);

res.send("我浏览过--"+city.toString());

});

app.get("/lvyou",function (req,res){

var citys=req.cookies.city;

var city=req.query.city;

if(citys){

citys.push(city);

}

else{

citys=[];

citys.push(city);

}

res.cookie("city",citys,{maxAge:1000\*60\*60});

res.send("我浏览-"+city);

});

app.listen(8100);

1. Session 的使用 express-session

var express=require("express");

var app=new express();

//可以看官网 www.npmjs.com 搜索

//1.安装 session cnpm install express-session --save

//2.引入session

var session=require("express-session");

//3.设置中间件 可以先拿官方的 进行配置

app.use(session({

secret: 'keyboard cat',

resave: false,

saveUninitialized: true,

cookie: { secure: false }

}))

//4.使用session

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

var user=req.session.user;

res.send("首页--"+(user?user+" 欢迎":"未登录"));

});

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

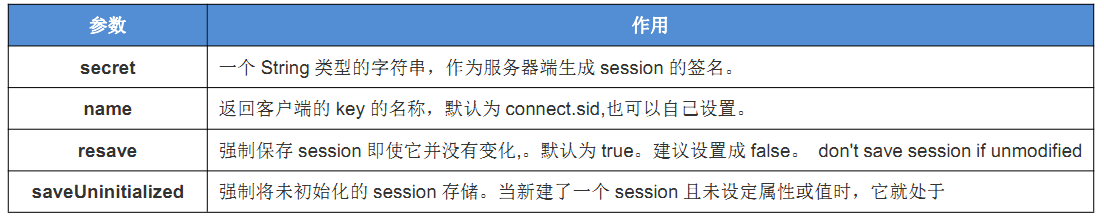
//设置session

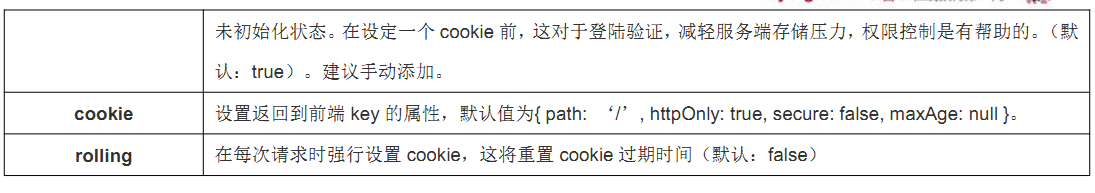
req.session.user="张三";

res.send("登录成功");

});

app.listen(8100);





Session 中的常用方法：

清除session

var express=require("express");

var app=new express();

//可以看官网 www.npmjs.com 搜索

//1.安装 session cnpm install express-session --save

//2.引入session

var session=require("express-session");

//3.设置中间件 可以先拿官方的 进行配置

app.use(session({

secret: 'keyboard cat',

resave: false,

saveUninitialized: true,

cookie: { secure: false }

}))

//4.使用session

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

var user=req.session.user;

res.send("首页--"+(user?user+" 欢迎":"未登录"));

});

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

//设置session

req.session.user="张三";

res.send("登录成功");

});

app.get("/loginout",function (req,res){

//可以修改session的有效时间

//req.session.cookie.maxAge=0;

//或者直接销毁session

req.session.destory(function (err){

console.log(err);

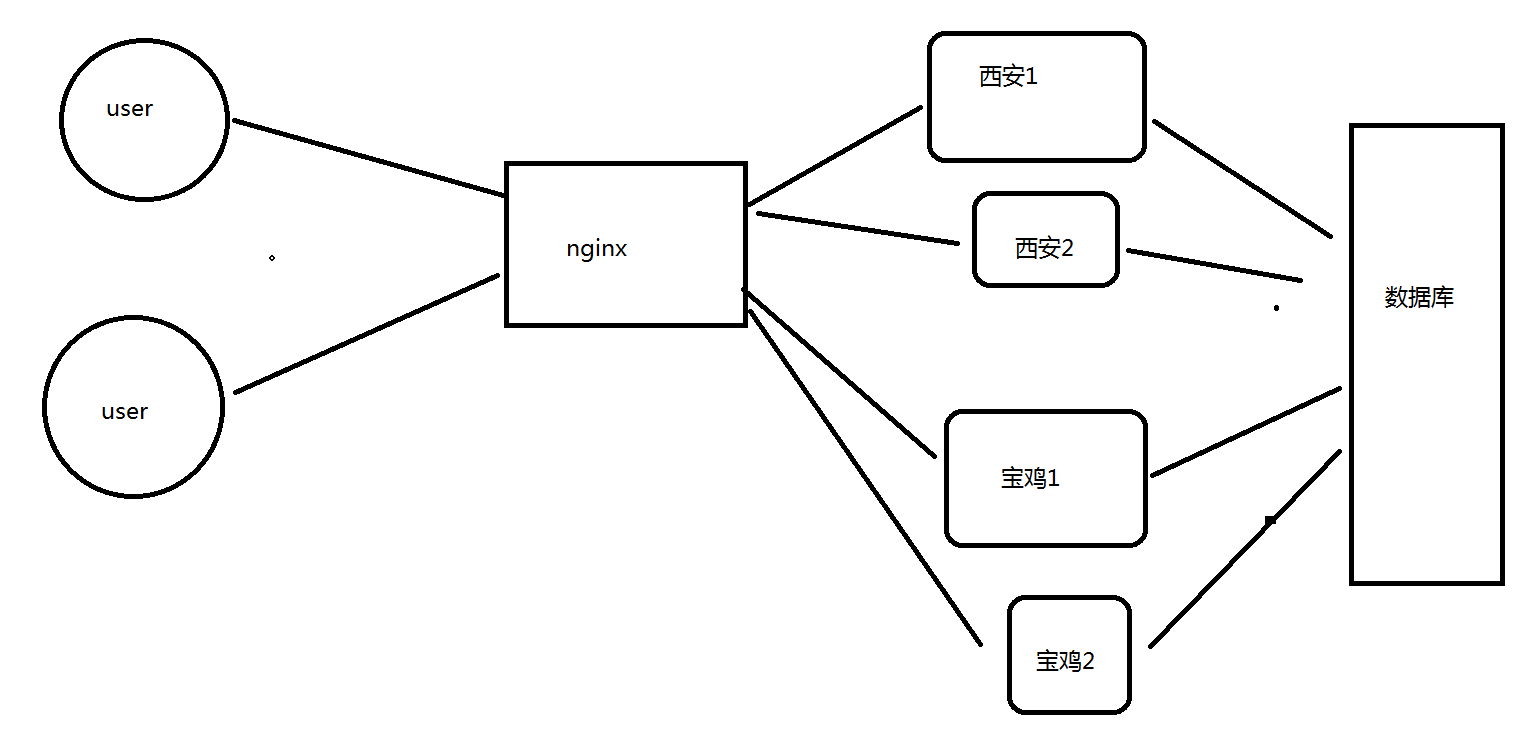
});

res.send("退出成功！");

});

app.listen(8100);

负载均衡session 把session保存到数据库



var express=require("express");

var app=new express();

var session=require("express-session");

//1.安装connect-mongo 模块 链接mongo模块 cnpm install connect-mongo --save

//2.引入

var mongostore=require("connect-mongo")(session);

//3.配置中间件

app.use(session({

secret: 'keyboard cat',

resave: false,

saveUninitialized: true,

rolling:true,

cookie: { maxAge:1000\*30\*60},

store:new mongostore({

url:"mongodb://127.0.0.1/student",//数据库的地址

touchAfter:24\*3600//24小时之内不变 除非这个值被修改

})

}))

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

var user=req.session.user;

res.send("首页--"+(user?user+" 欢迎":"未登录"));

});

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

req.session.user="张三";

res.send("登录成功");

});

app.get("/loginout",function (req,res){

req.session.destroy(function (err){

console.log(err);

});

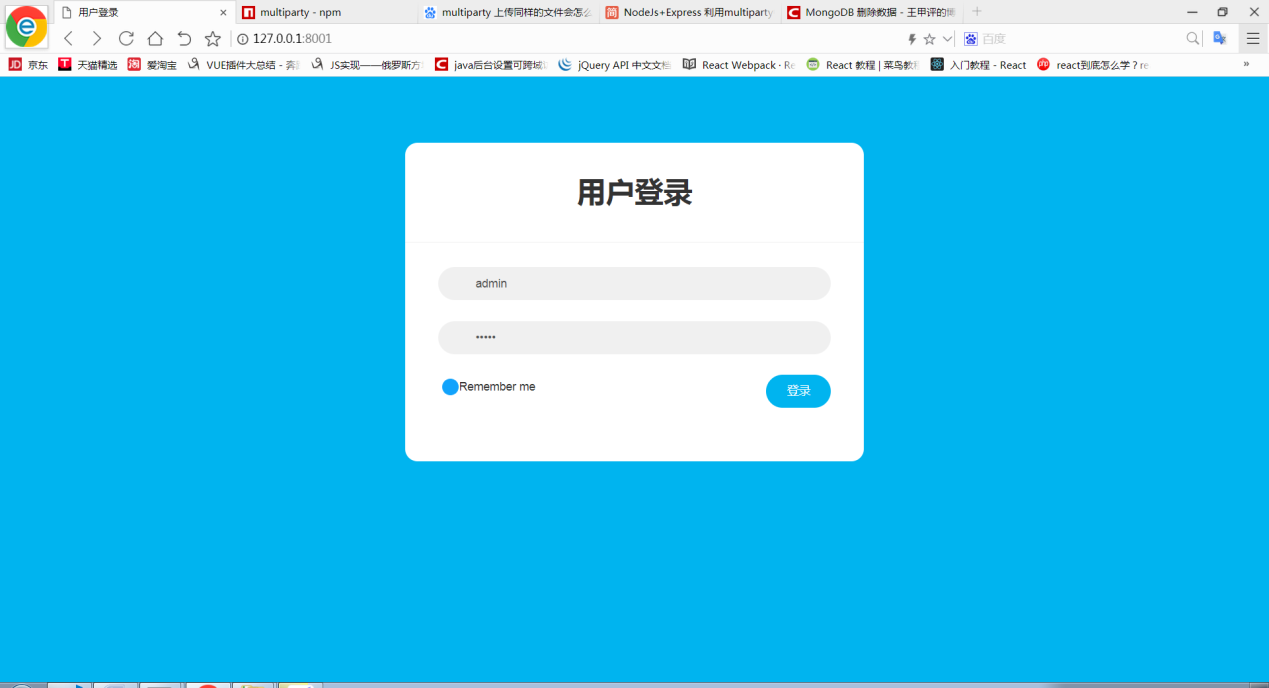
res.send("退出成功！");

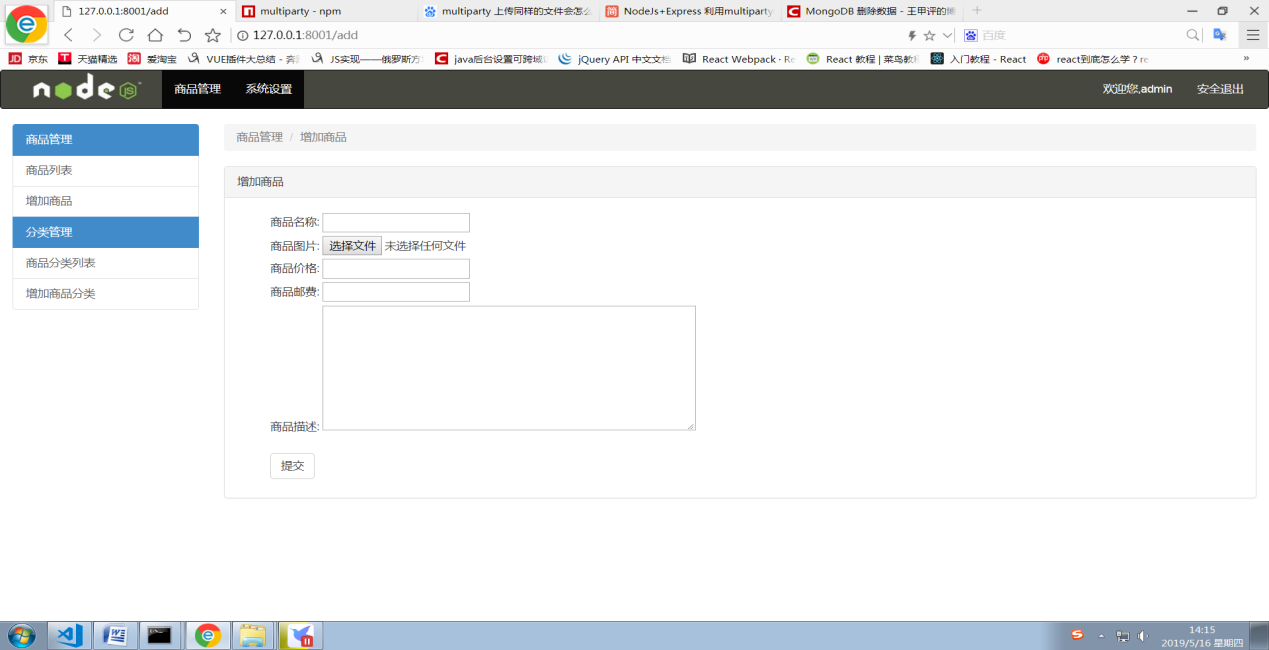
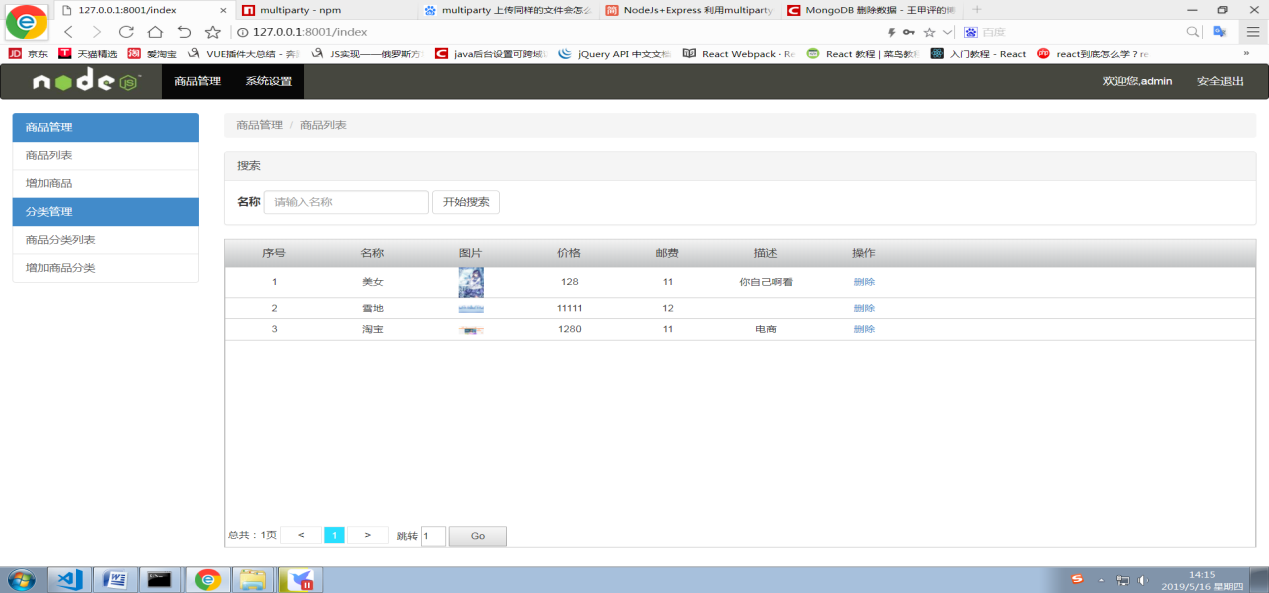
});

app.listen(8100);

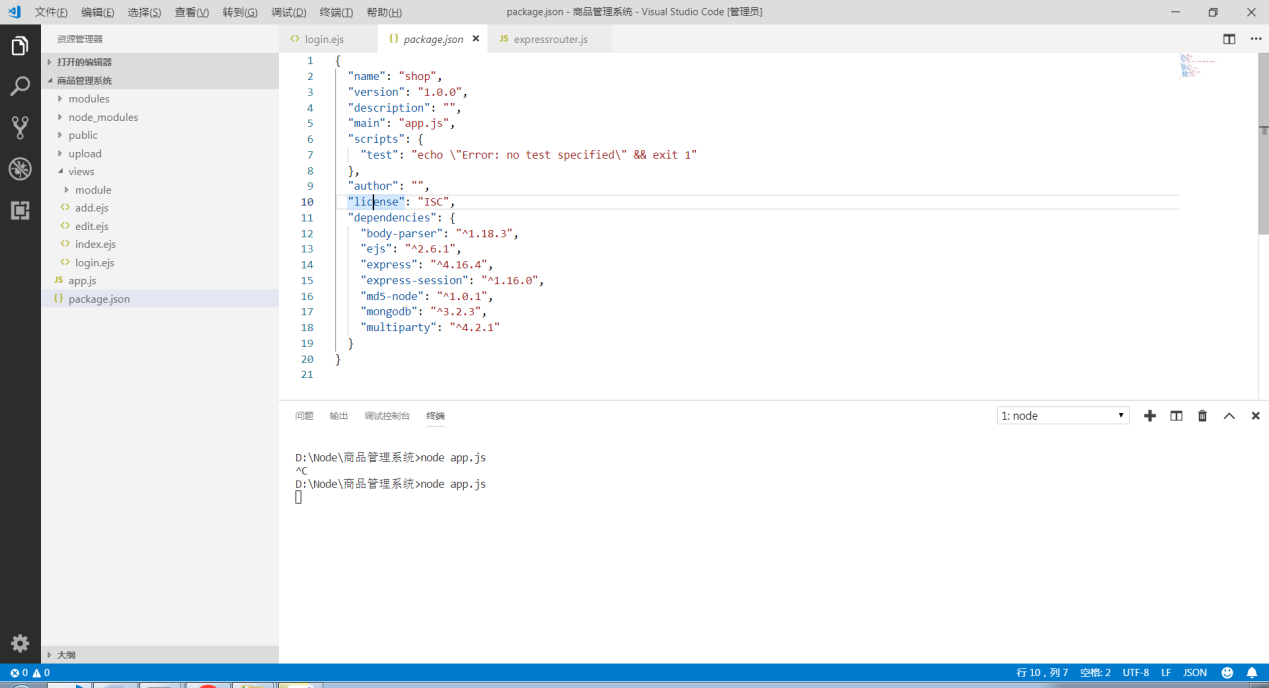
cookie 和session 区别？？？

1. Express 商品管理系统环境搭建（项目）





项目文件和使用的package.json



25 ) koa框架和环境搭建

Node.js是异步的，官方API支持的都是callback形式的异步编程模式，会带来许多的问题，比如callback嵌套问题，异步函数中可能调用同步函数callback返回，带来的数据不一致性，为了解决这些问题koa出现，

Koa是基于Node.js平台的一个web开发框架，它的开发思路和express差不多，最大的特点是可以避免异步嵌套。

26）Koa2.x框架的安装和使用

1.安装Node 8.x以上版本

2.安装koa-------安装koa框架和之前安装其他的模块一样

npm install koa –save / cnpm install koa –save

简单使用

//引入koa

var koa=require("koa");

//实例化koa对象

var app=new koa();

//创建中间件

//async 异步 方法

app.use(async function (ctx){

ctx.body="你好";

});

//监听端口

app.listen(8001);

27）koa中使用es6语法：async await promise的使用（重点）

Promise对象：

Promise 是异步编程的一种解决方案，比传统的解决方案——回调函数和事件——更合理和更强大。它由社区最早提出和实现，ES6 将其写进了语言标准，统一了用法，原生提供了Promise对象。

所谓Promise，简单说就是一个容器，里面保存着某个未来才会结束的事件（通常是一个异步操作）的结果。从语法上说，Promise 是一个对象，从它可以获取异步操作的消息。Promise 提供统一的 API，各种异步操作都可以用同样的方法进行处理

// promise 异步的使用

//resolve 成功 reject 失败

var promise=new Promise(function (resolve,reject){

var name="张三";

if(1==1)

{

resolve(name);

}else

{

reject(name);

}

});

promise.then(function (data){

console.log(data);

});

Async函数和await

ES2017 标准引入了 async 函数，使得异步操作变得更加方便。

//async 异步函数 和 promise 联合使用 加载异步

function get(){

return new Promise(function (resolve){

setTimeout(function (){

var name="张三";

resolve(name);

},2000);

});

}

async function getData(){

var a=await get();

console.log(a);

}

getData();

或者：async返回时一个promise对象 写成：

function get(){

return new Promise(function (resolve){

setTimeout(function (){

var name="张三";

resolve(name);

},2000);

});

}

async function getData(){

var a=await get();

return a;

}

getData().then(function (data){

console.log(data);

});

async function getStockPriceByName(name) {

const symbol = await getStockSymbol(name);

const stockPrice = await getStockPrice(symbol);

return stockPrice;

}

getStockPriceByName('goog').then(function (result) {

console.log(result);

});

可以查看es6官方文档

28） koa中的路由，get传值，动态路由传值

1.koa路由

路由（routing）是一个url和一个特定的http方法（get,post）等组成，涉及应用如何响应客户端对某个网站节点的访问

通俗：就是路由时根据不同的url地址，加载不同的页面实现不同的功能。

Koa中的路由和express中的路由不同，express中直接引入express模块直接配置路由，在koa中得安装路由模块—koa-router路由模块来实现。

cnpm install koa-router --save /npm install koa-router –save

// 1.安装koa 路由

var koa=require("koa");

//2.引入路由

var router=require("koa-router")();

//或者 上下两个一样

// var Router=require("koa-router");

// var router=new Router();

var app=new koa();

//配置路由

router.get("/",function (ctx,next){

ctx.body="你好";

});

//启动路由

app.use(router.routes());

//根据官方设置 //根据最后的状态设置响应头

app.use(router.allowedMethods());

app.listen(8001);

29）学习koa之前es6语法准备工作 promise async await学习

// promise 异步的使用

//resolve 成功 reject 失败

// var promise=new Promise(function (resolve,reject){

// var name="张三";

// if(1==1)

// {

// resolve(name);

// }else

// {

// reject(name);

// }

// });

// promise.then(function (data){

// console.log(data);

// });

//async 异步函数 和 promise 联合使用 加载异步

// function get(){

// return new Promise(function (resolve){

// setTimeout(function (){

// var name="张三";

// resolve(name);

// },2000);

// });

// }

// async function getData(){

// var a=await get();

// console.log(a);

// }

// getData();

function get(){

return new Promise(function (resolve){

setTimeout(function (){

var name="张三";

resolve(name);

},2000);

});

}

async function getData(){

var a=await get();

return a;

}

getData().then(function (data){

console.log(data);

});

// async function getStockPriceByName(name) {

// const symbol = await getStockSymbol(name);

// const stockPrice = await getStockPrice(symbol);

// return stockPrice;

// }

// getStockPriceByName('goog').then(function (result) {

// console.log(result);

// });

30）koa路由传值 get传值 动态路由

Koa路由的配置 <https://www.npmjs.com/package/koa-router>

// 1.安装koa 路由

var koa=require("koa");

//2.引入路由

var router=require("koa-router")();

//或者 上下两个一样

// var Router=require("koa-router");

// var router=new Router();

var app=new koa();

//配置路由

router.get("/",function (ctx,next){

ctx.body="你好";

});

//启动路由

app.use(router.routes());

//根据官方设置 //根据最后的状态设置响应头

app.use(router.allowedMethods());

app.listen(8001);

\*\*\*\*先来看koa里面的有几种get获取值的方式：

在 koa2 中 GET 传值通过 request 接收，但是接收的方法有两种：query 和 querystring。

query：返回的是格式化好的参数对象。

querystring：返回的是请求字符串

//http://localhost:8001/news?aid=123&name=zhangsan

router.get("/news",async (ctx)=>{

//ctx 参数结合了req res

ctx.body="新闻";

/\*

\* 在 koa2 中 GET 传值通过 request 接收，但是接收的方法有两种：query 和 querystring。

\* query：返回的是格式化好的参数对象。

\* querystring：返回的是请求字符串

\*/

console.log(ctx.query);//{ aid: '123', name: 'zhangsan' } 推荐使用

console.log(ctx.querystring);//aid=123&name=zhangsan

console.log(ctx.request.query);//{ aid: '123', name: 'zhangsan' }

console.log(ctx.request.querystring);//aid=123&name=zhangsan

});

Koa里面的动态路由传值：

//koa动态路由传值 获取值

//动态传一个值

//http://localhost:8001/page/11

router.get("/page/:id",async (ctx)=>{

ctx.body="page";

console.log(ctx.params);//{ id: '11' }

});

//动态传多个值

//http://localhost:8001/pagelist/1/2

router.get("/pagelist/:id/:aid",async (ctx)=>{

ctx.body="page";

console.log(ctx.params);//{ id: '1', aid: '2' }

});

31）koa中的中间件

通俗的讲：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，我们就可以

把它叫做中间件。

在express中间件 （Middleware）是一个函数， 它可以访问请求对象 （request object (req)） ,

响应对象（response object (res)）, 和 web 应用中处理请求-响应循环流程中的中间件，一

般被命名为 next 的变量。在 Koa 中中间件和 express 有点类似。

中间件的功能包括：

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。

如果我的 get、post 回调函数中，没有 next 参数，那么就匹配上第一个路由，就不会往下匹

配了。如果想往下匹配的话，那么需要写 next()

1. 应用级中间件

// 1.安装koa 路由

var koa=require("koa");

//2.引入路由

var router=require("koa-router")();

//或者 上下两个一样

// var Router=require("koa-router");

// var router=new Router();

var app=new koa();

//配置中间件 在执行所有路由之前执行的操作 任意代码

// app.use(async (ctx,next)=>{

// ctx.body="中间件";

// //不写next() 执行中间件 不会向下执行

// });

app.use(async (ctx,next)=>{

console.log(new Date());

await next();

});

//配置路由

router.get("/",async function (ctx,next){

ctx.body="首页";

});

router.get("/news",async (ctx)=>{

ctx.body="新闻";

});

//启动路由

app.use(router.routes());

//根据官方设置 //根据最后的状态设置响应头

app.use(router.allowedMethods());

app.listen(8001);

2．路由级中间件

//比如在匹配新闻页面之前 做相关的操作

//写一个路由中间件

router.get("/news",async (ctx,next)=>{

console.log("这是新闻页面的中间件");

await next();

});

router.get("/news",async (ctx)=>{

ctx.body="新闻";

});

3.错误处理中间件

//koa中的错误处理中间件

//在koa框架中中间件执行和前后顺序无关

app.use(async (ctx,next)=>{

await next();

if(ctx.status==404)

{

console.log(404);

ctx.body="这个页面是404";

}

else{

console.log("正常");

}

});

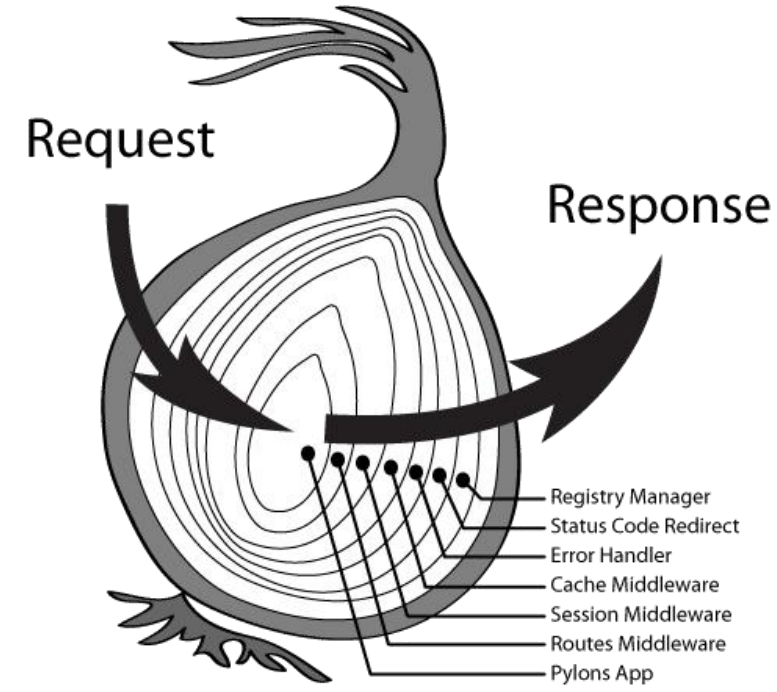
router.get("/news",async (ctx)=>{

ctx.body="新闻";

});

Koa中中间件的执行顺序

中间件洋葱图



Node 链接mysql 使用mysql

/\*\*

\* node 链接mysql 配置

\* 1.安装mysql

\* 2.mysql 链接配置 port 默认3306

\* 3.链接mysql

\*/

var mysqlSettings={

host:"localhost",

user:"root",

password:"12345678",

database:"student",

port:"3308"

}

var mysql=require('mysql');

var connection=mysql.createConnection({

host:mysqlSettings.host,

user:mysqlSettings.user,

password:mysqlSettings.password,

database:mysqlSettings.database,

port:mysqlSettings.port

});

//链接

connection.connect();

/\*\*

\*下面执行sql语句

\* ? 和值类似键值 相当于占位符

\*/

var sql="select \* from user where id=?";

connection.query(sql,2,function(err,res){

if(err) throw err;

console.log(res);

});

//关闭数据库链接

connection.end();