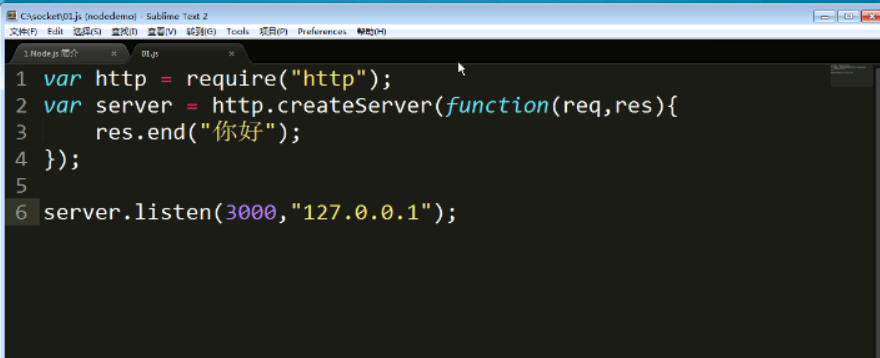
NODE从零开始

一 原生



1 http模块

Require引入模块,

http.createServer创建服务

Req,res

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

写内容 res.write()

结束res.send()

监听 .listen(3000);

2 url模块

解析url地址

引入模块

先拿到url通过http中req.url得到类似地址 /news?id=10&type=person

Url.parse(url,true)解析成对象

3 supervisor

热更新插件

npm -g install supervisor

启动文件 Supervisor \*\*.js

4 common.js (exports module.exports require)模块化标准

暴露模块的两种方式:

exports.somthing=str

需要挂在在一个属性上,而且暴露出去的是全部的,要访问str还需要先访问 somthing.

Var som = require(‘somthing.js’);

Som.somthing

Module.exports = somthing;

直接暴露somthing.直接访问即可

require( ‘./\*\*.js’)

Require(‘./\*\*’) 可省略.js

Require(‘\*\*’) 在node\_modules文件加下可省略路径

Node\_modules/nav/nav.js 引入此路径js

Require(‘nav’) 需要配置nav目录下的package.json的入口文件位nav.js

5 fs模块

引入模块

fs.stat 检测是文件还是目录

Fs.stat(‘’,(err,stats)=>{

Stats.isFile() 判断文件

Stats.isDirectory() 判断目录

})

fs.mkdir 创建目录

fs.writeFile 创建写入文件

Fs.appendFile 追加文件

Fs.readFile 读取wenjian

Fs.readdir 读取目录

Fs.rename 重命名(剪切)

Fs.rmdir 删除目录

Fs.unlink 删除文件

注意:fs操作都是异步,不能在for循环里用,因为异步不会被阻止,不会等待循环结束而是同时进行.需用自执行函数(function get(i){ get(i+1)})(0)

文件流方式读写数据(分步读写),数据量大时用文件流方式.防止卡死

fs.createReadStream(‘\*\*.txt’) 创建读取流

fs.createWriteStream(‘\*\*.txt’) 创建写入流

读取流用on接受 data || end ||error事件

写入用 write事件写入 用end标识结束 用on接受 finish || error事件

管道读写 读-->写

Readstream,pipe(writeStream)

6 events模块 事件驱动 广播和接受广播

引入events模块(默认模块)

Var EventEmitter = new events.EventEmitter(); 创建实例

EventEmitter.on(‘somthing’,(data)=>{}) 监听广播

EventEmitter.emit(‘somthing’,data) 发送广播

‘somthing’表示广播的名称,需要保持一致

在异步封装中,可在函数内发送广播,在函数外监听广播.当调用函数后,监听自动会 执行.

二 数据库 mongodb

1. 安装
2. 配置环境变量
3. 建一个数据库文件夹
4. mongod --dbpath ./ 开启数据库 ./新建数据库文件夹的路径
5. 另开cmd窗口 mongo 链接本地数据库
6. 命令

mongo 确认安装成功,查看版本,链接数据库

mongo 127.0.0.1:27017 链接远程数据库

链接成功后

Show dbs 查看数据库列表

use test 使用/创建数据库

1使用test数据库 先use test

show collections 查看所有表

使用某个表(用db+表明+操作命令)

查询部分

db.表名.insert({“name”:”小明”}) 插入数据

db.bm.find() 查询数据

db.bm.find({“age”:20})

db.bm.find({“age”:{$gt:20}}) 大于20的

db.bm.find({“age”:{$lt:20}}) 小于20的

gte 大于等于

lte 小于等于

db.bm.find({“name”:”zs”,”age”:20}) 多条件查询

db.bm.find(“age”:{$gte:20,$lte:30}) 大于20小于30的

模糊查询

db.bm.find({“title”:/文章/})

db.bm.find({“title”:/^文章/}) 以文章开头的

只查询某字段的数据

db.bm.find({“age”:{$gte:20,$lte:30},{“age”:1}}) 只返回 符合条件的age字段的数据,添加{“”:1}就可增加字段

升序降序 1升序 -1降序

db.bm.find().sort({“age”:1})

分页查询

db.bm.find().limit(2) 查前2条

db.bm.find.skip(2).limit(3) 跳过两条查3条

$or或者

db.bm.find({$or:[{“age”:20},{“age”:24}]}) 20或者24的 都查出来

查询第一条数据

db.bm.findOne()==db.bm.find().limit(1)

查询数据数量

db.bm.find().count() 所有数据条数

db.bm.find({“age”:50}).count() 50的条数

2创建数据库

创建

use sqlname

新建一个表,在表里插入数据才能创建成功

db.collectionname.insert({“”:””})

3修改/删除数据部分

db.bm.drop() 删除表

db.bm.update({“name”:”zs”},{$set:{“age”:35}}) 更改姓名位zs的年龄为35

db.bm.update({“name”:”zs”},{“age”:20}) 不写$set的话是zs这条数 据整条修改为{“age":20}

删除数据

db.bm.remove({“name”:”zs”}) 删除姓名的所有

db.bm.remove({“name”:”zs”},{justOne:true}) 删除一条

db.bm.remove({“name”:”zs”,”age”:20}) 删除zs且年龄20的

4 索引

db.bm.getIndexes() 查看索引

db.bm.dropIndex({“name”:1}) 删除name字段索引

db.bm.ensureIndex({“name”:1}) 给name字段设置索引

显示查询时间

db.bm.find().explain(“executionStats”)

复合索引(联合索引)

db.bm.ensureIndex({“name”:1,”age”:-1}) 正负1表示升序降序

唯一索引

db.bm.ensureIndex({“id”:1},{“unique”:true}) 保证插入的数据id不能重复

db.bm.ensureIndex({“id”:1,”age”:-1},{“unique”:true}}) 复合唯一索引

5 nodejs操作数据库

Mongodb文档

3.x http://mongodb.github.io/node-mongodb-native/3.0/quick-start/quick-start/

2.2

http://mongodb.github.io/node-mongodb-native/2.2/quick-start/quick-start/

1. 安装mongodb
   1. npm i [mongodb@3.x](mailto:mongodb@3.x) -D
2. 引入和链接(2.2和3.x区别在与db对象的获取)
   1. 都引入MongoClient
      1. Var MongoClient = require(‘mongodb’).MongoClient;
   2. 设置url数据库地址(有区别)
      1. 2.2中 url=’mongodb://127.0.0.1:27017/dbName’
      2. 3.x中 url=’mongodb://127.0.0.1:27017’

一个把地址精确到具体的数据库名(不是表名)

* 1. 获取db对象操作表的增删改查
     1. MongoClient.connect(url,(err,db)=>{

db.collection(‘collectionName’).insertOne({},callback)

})

这里db是直接获取到的,因为在url中直接到数据库名了

②MongoClient.connect(url,{useNewUrlParser:true},(err,client)=>{

const db = client.db(‘dbName’); //通过client和数据库名获 取db对象

剩下操作表基本一样

})

注上面标红的{useNewUrlParser:true}是链接本地mongodb地址解析 错误时加的,2.2中好像没这问题

关闭数据库链接:每次操作完都要关闭

db.close(); //在回调中写,确保数据操作完成再关闭

1. 增删改查方法
   1. insertOne() insertMany() 都有Many
   2. UpdateOne({},{$set:{}}) 先查再改
   3. deleteOne({})
   4. 查询

Var list = [];

var res= db.collection(“collectionName”).Find() 先查到数据

res.each((error,doc)=>{ //mongo提供的方法循环拿出数据

If(error){return};

If(doc!=null){

list.push(doc); 将数据取出到list中

}else{ doc==null说明循环结束

return list; 返回拿到的数据

}

})

方法2:

Let res = db.collection(‘test’).find();

Res.toArray((err,docs)=>{

Clg(docs) //查询到的数据数组

})

三 Koa

1. Es6
   1. 字符串拼接

` \*\*\*\*\*\*\*${name}\*\*\*\* `

* 1. Promise

Let P = new Promise((resolve,reject)=>{

If(succes){

Resolve()

}else{

Reject()

}

})

P.then(res=>{}).catch(err=>{})

(3) Async await

Async 标记函数为异步,返回的是promise

可以用promise的方式.then()解决回调

Await 必须在async(或者返回值为promise对象)中使用

在最外层加async,逻辑在这一层完成,即可解决其他异步又可不必管最外 层async. Koa的方法都在async中完成的.

async function setName(){

return ‘谷守帅’;

}

async function getName(){ //此处相当于最外层async

Let name = await setName(); //此处为最后逻辑,都写在这里就可

Clg(name) //不考虑此异步再返回promins的问题

}

getName() //谷守帅

1. koa 路由router get/post请求

Koa-router 中间件

1 引用 const router = require(‘koa-router’)();

2 配置

router.get(‘/’,(ctx)=>{ //ctx对象包含req,res的上下文对象

ctx.body=”首页”

}).get(‘/news’,(ctx)=>{}) //链式写法

3 启动路由

app.use(router.routes()) //启动

app.use(router.allowedMethods()) //自动添加响应头

4 get传值取参数

ctx.query ctx.querystring ctx.url

ctx包含request中的所有信息,都可以直接访问,所以也可通过ctx.request获取相关参数

5 动态路由 /news/\*\*\*

.get(“/news/:nid/:page”,(ctx)=>{

let data = ctx.params; 动态路由参数

})

地址栏输入http://localhost:3000/news/a/3

data = {nid:a,page:3}

6 重定向

router.redirect('/o', '/'); //重定向到首页

router.all('/p',ctx=>{ //重定向,有待理解

ctx.redirect('/');

ctx.status = 301;

})

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

let username = ctx.cookies.get('username')||'';

if(!username){

ctx.redirect('/?code=0'); //cookie过期定向到首页,可以带些参数

return;

}

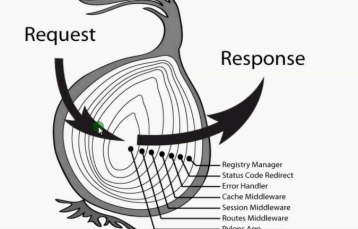
ctx.body = username;

})

3 中间价

匹配路由之前或之后所做的一切操作

不会受书写位置影响执行顺序.

洋葱顺序:中间件 | next | 自定义路由 | next | 中间件next后的逻辑(错误处理中间件)

1. 应用级中间件

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

Clg(匹配任何路由,在任何路由匹配到之前执行)

Await next() //不next表示匹配结束,将不在向下执行.

})

1. 路由级中间件

router.get(‘/’,async (ctx,next)=>{

await next()

})

1. 错误处理中间件

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

Clg(匹配所有路由)

Await Next();

If(ctx.status==404){

Ctx.body=’404’

}

})

1. 其他第三方中间件

ejs

--save koa-views

const Views = require('koa-views');

配置:

app.use(Views('views',{ 注意路径

map:{html:'ejs'} 在html文件里使用ejs语法

}))

注意:在渲染是要用await

await ctx.render('index',{title:'ejs'}) 参数:模板名称,数据

语法:

Js逻辑放在<% %>中

渲染结果放<%= %>中

渲染html标签<%- %>

1. For循环

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

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

<%}%>

1. 引入其他模板

<% include ./header.html%>

1. If else

<%if(num>10){%>

<h3><%=num%>大于10</h3>

<%}else{%>

<h3><%=num%>小于等于10</h3>

<%}%>

1. 所有页面直接用的公共数据

// 添加公共数据供所有页面调用

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

let name = '谷守帅';

ctx.state.name = name;

await next();

})

Html中调用渲染没区别<%=name%>

koa-bodyparser post获取数据(只能表单)

bodyparser = require('koa-bodyparser');

app.use(bodyparser());

router.post('/user',async (ctx,next)=>{

let data = ctx.request.body; //获取post数据并格式化

console.log(data);

ctx.body = data;

await next();

})

Koa-multer 上传文件和表单提交

(其他上传文件中间件[koa-body](https://cnpmjs.org/package/koa-body" \t "https://blog.csdn.net/simple__dream/article/details/_blank) [busboy](https://www.npmjs.com/package/busboy" \t "https://blog.csdn.net/simple__dream/article/details/_blank)  koa-better-body)

<https://github.com/expressjs/multer/blob/master/doc/README-zh-cn.md> 此文档针对express,可参考,大同小异

安装 npm i --save koa-multer

multer = require('koa-multer'),

配置

//配置

var storage = multer.diskStorage({

//文件保存路径

destination: (ctx, file, cb)=> {

cb(null, \_\_dirname+'/public')

},

//修改文件名称

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

Console.log(file) //包含一些文件基本信息,但不包括文件大小,所以不能在此控制文件大小

let fileFormat = (file.originalname).split(".");

let extname = fileFormat[fileFormat.length - 1]; //后缀

cb(null, Date.now() + "." + extname); //重命名

}

})

var fileFilter = (ctx,file,cb)=>{

console.log(file);

// 过滤只上传的后缀为txt的文件

if (file.originalname.split('.').splice(-1) == 'txt') {

cb(null, true); //true表示接收文件

} else {

cb(null, false); //false表示不接收文件

}

}

var limits = { //限制文件大小

fileSize:1024

}

//加载配置

var upload = multer({

storage: storage,

fileFilter:fileFilter,

// limits: limits //设置大小限制后,超过限制时会直接报错,还不知道怎么解决

});

//路由

router.post('/up', upload.single('file'), async (ctx, next) => {

console.log(ctx.req.body); //此为multer提供的 可获取文本表单提交的内容

ctx.body = {

filename: ctx.req.file //返回文件详细信息包括文件名,文件大小,不过到此步时文件已经上传成功,目前不知道怎么在上传成功之前做一些操作(在配置步骤可以,但操作空间有限,比如重命名的信息结合表单提交的文本信息)

}

})

Html部分需要设置提交方式

<form id="f1" action="/up" method="POST" enctype="multipart/form-data">

<input type="file" name="file">

<input type="text" name="test" value="post请求">

<input type="text" name="test2" value="post请求22">

</form>

Koa-static 静态资源中间件

static = require('koa-static')

app.use(static(\_\_dirname+'/public'))

app.use(static(\_\_dirname+'/static'))

<link rel="stylesheet" href="/css/index.css"> //public下的css目录下的index.css

<link rel="stylesheet" href="/static.css"> //static文件夹下的static.css

可以配置多个静态服务

Art-template 模板引擎

<http://aui.github.io/art-template/zh-cn/docs/syntax.html>

<http://aui.github.io/art-template/zh-cn/koa/index.html>

安装 npm install --save art-template  
 npm install --save koa-art-template

render = require('koa-art-template'),

render(app,{

root:path.join(\_\_dirname,'views'),

extname:'.html',

debug: process.env.NODE\_ENV !== 'production'

})

await ctx.render('art',{data:data})

语法有两种:1 和ejs一样

2 标准语法{{}}

{{value}}  
 {{data.key}}  
 {{@html}} 渲染标签  
 {{a ? b : c}}  
 {{a || b}}  
 {{a + b}}

<ul>

{{each data.list}} 循环可以用each {{$index}}{{$value}}直接拿索引和值

<li>{{$value}}</li> 也可以用for(var i....)

{{/each}}

</ul>

{{if data.num>20}} 条件不用写{}但得有闭合的{{/if}}

<h4>{{data.num}}大于20</h4>

{{else}}

<h4>{{data.num}}小于等于20</h4>

{{/if}}

Cookie

ctx.cookies.set('username','gss',{ //设置cookie

maxAge:1\*1000, //有效时长

path:’/news’ //规定可以访问到此cookie的页面

httpOnly:true //表示此cookie只能服务端访问,客户端js不能访问

})

let username = ctx.cookies.get('username')||''; //获取cookie

Cookie不能设置中文,需要通过Buffer来编码解码实现

new Buffer(‘谷守帅’).toString('base64') 6LC35a6I5biF base64编码

new Buffer(‘6LC35a6I5biF’, 'base64').toString() 谷守帅 转为中文

Session

以cookie为基础,设置session形成key和value

Key发送到客户端,value储存到服务器

客户端通过key访问服务器,服务器拿取储存的值

Koa-session

/\* koa-session 配置 基本固定配置\*/

app.keys = ['some secret hurr'];

const CONFIG = {

key: 'koa:sess', //默认

maxAge: 5000, //过期时间

autoCommit: true,

overwrite: true, //允许重写,true/false都可重写(设置无效)

httpOnly: true, //只能服务端访问

signed: true, //默认 签名

rolling: false, //每次访问都更新过期时间

renew: true, //快过期时更新过期时间 和rolling效果一样

};

app.use(session(CONFIG, app)); //配置中间件

ctx.session.views = 10 //设置session

let n = ctx.session.views; //获取session

怎么设置多个不同配置的session???

**4. mongodb封装**

**1 创建module文件夹,用于存放封装的模块**

**2 创建config.js配置文件,**

**配置const config = {dbUrl:’mongodb://127.0.0.1:27017’**

**dbName:’itying’}**

**暴露 module.exports = config;**

**创建db.js 封装数据库操作**

**引入模块**

const MongoClient = require('mongodb').MongoClient,

ObjectID = require('mongodb').ObjectID,

//用于解析\_id的值 5c08f2e98712dd4730ad469c => ObjectId("5c08f2e98712dd4730ad469c")

\_CONFIG = require('./config');

**创建Db类**

**思路:**

1. **创建Db类**
2. **添加链接,查找,删除等方法**
3. **这些方法都是需要暴露出去的,通过Db类实例直接访问的**
4. **所以通过promise暴露出db对象或查找,删除等结果**
5. **路由中在async 下通过await即可直接获取结果**
6. **封装中第一个问题:每个操作数据库的方法都要先链接数据库获取db对象才能操作,就出现了多次链接数据库的问题,**
   1. **解决:构造函数中定义db属性,第一次链接数据库时将db对象赋值,通过判断,再需要链接时,如果有db对象则不链接,直接返回构造函数中存好的db对象**
7. **实例化一次,调用不同数据库操作方法需要多次链接数据库的问题解决了,但如果实例化多次,就会导致每次实例化都产生新的实例,也就会重置构造函数中的db为空,所以每次实例又会重新链接数据库;**
   1. **解决: 单例模式 返回同一个实例.通过定义类的静态方法和静态属性,此静态属性即存放类的实例.通过此静态方法判断静态属性是否存在.不存在:赋值实例,返回,存在:直接返回.保证了每次返回的都是最初的唯一的实例.**

class Db {

static getDbInstrance(){ //静态实例方法(单例模式关键)

if(!Db.instrance){

Db.instrance=new Db();

}

return Db.instrance;

}

constructor(){

//构造函数,内部只执行一次,因为静态方法中返回的是一个实例(单例:只实例化一次)

this.db='';

//构造函数的属性接受db对象,避免每次重新链接数据库才能获取db对象

this.connect(); //初始化时先链接数据库,防止第一次链接慢

}

connect(){

//链接数据库,用promise暴露db对象,判断构造函数db属性,避免每次重新链接

return new Promise((res,rej)=>{

console.time('b');

if(!this.db){

console.time('w');

MongoClient.connect(\_CONFIG.dbUrl,(err,client)=>{

if(err) {rej(err);return};

this.db = client.db(\_CONFIG.dbName);

console.timeEnd('w');

res(this.db);

})

}else{

console.time('w2');

console.timeEnd('w2');

res(this.db);

}

console.timeEnd('b');

})

}

find(collectionName,json) {

/\*

封装查找方法,接受表名和查找条件两个参数

\*返回包含查询结果的promise 可以用 await直接接收

\*/

return new Promise((res,rej)=>{

this.connect().then((db) => {

console.time('f');

let data = db.collection(collectionName).find(json);

data.toArray((err, docs) => {

if(err){rej(err);return;}

console.timeEnd('f');

res(docs);

})

}).catch(err => {

console.log(err);

})

})

}

insert(collectionName,json){

return new Promise((res,rej)=>{

this.connect().then(db=>{

db.collection(collectionName).insertOne(json,(err,data)=>{

if(err) rej(err);

res(data.result.ok);

});

})

})

}

update(collectionName,json1,json2) {

return new Promise((res,rej)=>{

this.connect().then(db=>{

db.collection(collectionName).updateOne(json1,{$set:json2},(err,data)=>{

if(err) rej(err);

console.log(data.result.ok);

res(data.result.ok);

})

})

})

}

remove(collectionName, json) {

return new Promise((res,rej)=>{

this.connect().then(db=>{

db.collection(collectionName).removeOne(json,(err,data)=>{

if(err) rej(err);

console.log(data.result.ok);

res(data.result.ok);

})

})

})

}

getObjectId(id){

/\*\*

\* 通过默认生成的\_id查询时,必须格式统一才能查找到

\* 即 "\_id": ObjectId("5c08f2ce8712dd4730ad469b")

\* 通常获取的\_id 是 5c08f2ce8712dd4730ad469b

\* 所以通过此函数能还愿正规格式,查询到结果

\* 例: {"\_id":Db.getObjectId(id)}

\*/

return new ObjectID(id);

}

}

module.exports=Db.getDbInstrance(); //暴露出的是一个实例,这个实例即单例

**注意:再模板中渲染\_id时{{@$value.\_id}} @符是art-template中渲染html结构的方式,可以去掉多余的引号**

**5 路由模块化和视图模块化**

**将路由模块化 一般分为 admin后台 web前台 api接口**

**创建routers文件夹**

**创建admin.js web.js api.js**

**在每个文件中引入router,并配置路由,然后暴露路由启动router.routes()**

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

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

ctx.body='admin首页';

})

module.exports = router.routes();

在app.js中引入配置路由模块

const Koa = require('koa'),

router = require('koa-router')(),

admin = require('./routers/admin'); //引入路由模块

api = require('./routers/api');

const app = new Koa();

router.use('/admin',admin); //配置路由模块

router.use('/api',api);

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

ctx.body = '首页'

})

app.use(router.routes()); //启动路由

app.use(router.allowedMethods());

app.listen(3000);

访问

<http://localhost:3000/admin> //admin首页

每个大模块内有份很多小模块比如 admin后台模块又包括:user news ...

按照上边的方法,再建user文件夹,再见user相关的路由并配置,暴露

视图模块化即在views中分文件夹,分文件即可.在app.js中配置好render后能在其他子路由中直接用,不必每个都配置.(自己封装的db对象需要每个引入)

WebSocket(插座)和socket.io库

Socket.io库:https://socket.io

1. 安装:

Npm i socket.io

1. 引入,初始化

首先需要创建server服务,是http模块提供的,不是koa中的

Var server = require(‘http’).createServer(app.callback()) 用app.callback()关联

Var io = require(‘socket.io’)(server)

1. 监听链接

//链接转成tcp协议

io.on("connection",function(socket){

console.log("1个客户端连接了");

//接收广播,’formdata’保持前后端一直

Socket.on(‘formdata’,(data)=>{

//广播,将接收的data数据又广播给链接的用户

Io.emit(‘back’,data)

Socket.emit(‘back’,data) //socket和io都可以广播

})

})

1. 注意,此时开启服务要用server.listen(3000)
2. 前端页面部分,启动socket后会产生一个js,路由地址是http://localhost:3000/socket.io/socket.io.js
3. 引入socket.io.js

<script type="text/javascript" src="/socket.io/socket.io.js"></script>

1. 初始化

var socket = io(); //初始化socke对象

socket.emit('formdata','广播的数据'); //广播

socket.on('back',function(data) { //接收

//data :广播的数据

})

Windows上设置mongodb服务器自动启动

需要将启动mongodb添加到”服务”中去.

1.管理员身份运行”命令提示符”

2.在其中输入mongod --bind\_ip 0.0.0.0 --logpath "C:\Program Files\MongoDB\Server\3.4\logs\mongodb.log" --logappend --dbpath "D:\MongDB-server" --port 27017 --serviceName "MongoDB" --serviceDisplayName "MongoDB" --install

注释:ip:0.0.0.0表示任何ip都能访问

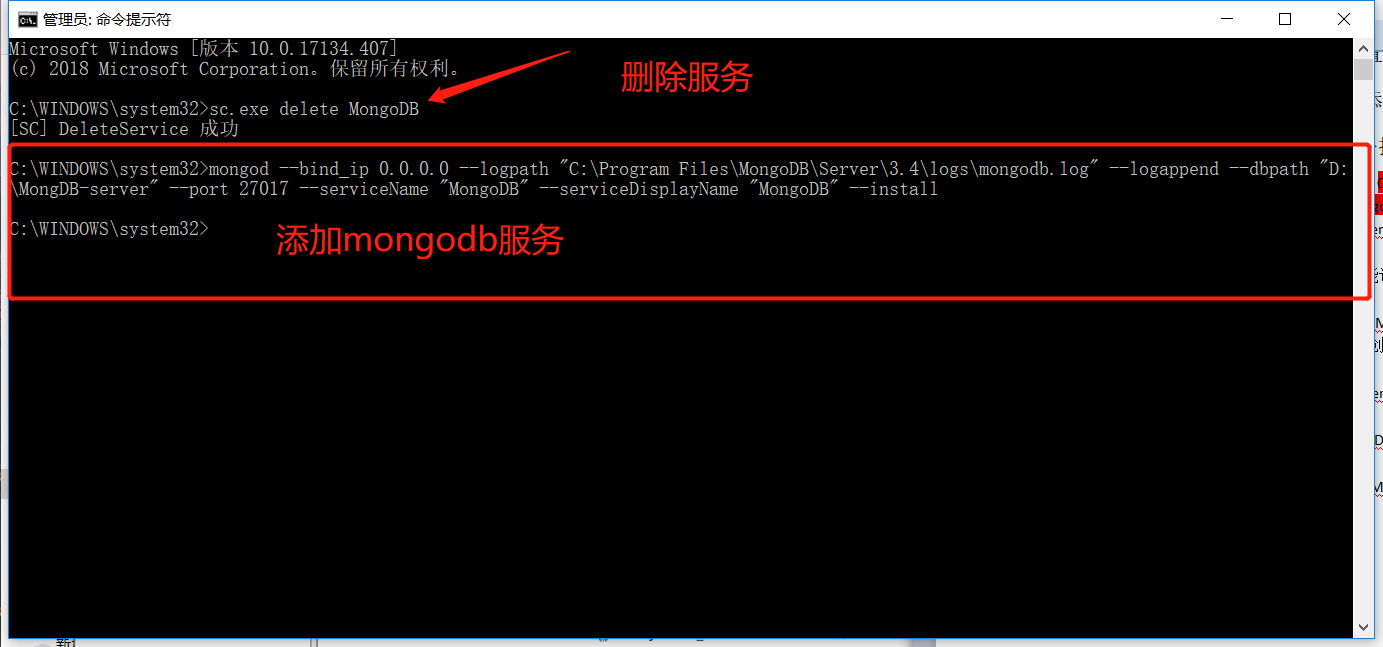
软件安装位置:C:\Program Files\MongoDB\Server\3.4\logs\mongodb.log

其中logs\mongodb.log 是手动创建的文件夹和文件,用来打印日志

开启服务位置:D:\MongDB-server

服务名称: serviceName "MongoDB"

显示名称:serviceDisplayName "MongoDB"



1. 运行后,在计算机管理--服务中找到MongoDB右键启动即可 