egg框架的总结

* Egg是一个通用，多线程的Java爬虫框架。
* Egg简单小巧，api非常简单，容易上手。
* Egg性能不错，并实现多种请求方式。
* 能够比较快的响应使用者的需求

egg框架继承于koa框架，egg框架提供了一个强大的插件机制，不需要引用大量的插件来实现功能。egg中使用的是严格模式，采用的是es6的class语法。

目录

**[1.首先初始化一个egg项目 3](#_Toc13560)**

[1.1启动项目： 3](#_Toc14060)

**[2.分析egg各目录下的文件的作用 4](#_Toc26496)**

[2.1如上，由框架约定的目录： 4](#_Toc12637)

[2.2由内置插件约定的目录： 5](#_Toc29865)

**[3.渲染模板 5](#_Toc27912)**

[3.1 先安装对应的插件 egg-view-nunjucks ： 5](#_Toc3667)

[3.2 开启插件： 5](#_Toc22443)

[3.3 新建一个index.nj文件 6](#_Toc20141)

[3.4 添加 Controller 和 Router 6](#_Toc8310)

[3.5在express框架中是这样子的： 7](#_Toc14049)

**[4.链接数据库 8](#_Toc32704)**

[4.1 安装插件：(这是egg框架中内置的,so下载时，就这样) 8](#_Toc12914)

[4.2开启插件 9](#_Toc7521)

[4.3然而express框架是这样的: 9](#_Toc4784)

**[5.创建集合 9](#_Toc4827)**

**[6.分析路由 11](#_Toc10950)**

[6.1 如果不想将路由分开写： 11](#_Toc12208)

[6.2分开写路由这样写： 11](#_Toc14629)

**[7.从数据库查找数据，再给客户端发送响应数据 12](#_Toc6558)**

**[8.登录时设置cookie以及session 14](#_Toc29019)**

**[9.退出时设置cookie以及清空session 15](#_Toc28573)**

**[10.异常处理 16](#_Toc20890)**

**[11.Service请求级别的对象 16](#_Toc3530)**

[11.1 步入接口： 17](#_Toc2423)

##### 1.首先初始化一个egg项目

$ npm i egg-init -g  
$ egg-init egg-example --type=simple  
$ cd egg-example  
$ npm i

1.1启动项目：

$ npm run dev

##### 2.分析egg各目录下的文件的作用

|  |
| --- |
| egg-project ├── package.json ├── app.js (可选) ├── agent.js (可选) ├── app | ├── router.js │ ├── controller │ | └── home.js │ ├── service (可选) │ | └── user.js │ ├── middleware (可选) │ | └── response\_time.js │ ├── schedule (可选) │ | └── my\_task.js │ ├── public (可选) │ | └── reset.css │ ├── view (可选) │ | └── home.tpl │ └── extend (可选) │ ├── helper.js (可选) │ ├── request.js (可选) │ ├── response.js (可选) │ ├── context.js (可选) │ ├── application.js (可选) │ └── agent.js (可选) ├── config | ├── plugin.js | ├── config.default.js │ ├── config.prod.js | ├── config.test.js (可选) | ├── config.local.js (可选) | └── config.unittest.js (可选) └── test  ├── middleware  | └── response\_time.test.js  └── controller  └── home.test.js |

2.1如上，由框架约定的目录：

app/router.js 用于配置 URL 路由规则，具体参见 [Router](https://eggjs.org/zh-cn/basics/router.html)。

app/controller/\*\* 用于解析用户的输入，处理后返回相应的结果，具体参见 [Controller](https://eggjs.org/zh-cn/basics/controller.html)。

app/service/\*\* 用于编写业务逻辑层，可选，建议使用，具体参见 [Service](https://eggjs.org/zh-cn/basics/service.html)。

app/middleware/\*\* 用于编写中间件，可选，具体参见 [Middleware](https://eggjs.org/zh-cn/basics/middleware.html)。

app/public/\*\* 用于放置静态资源，可选，具体参见内置插件 [egg-static](https://github.com/eggjs/egg-static" \t "https://eggjs.org/zh-cn/basics/_blank)。

app/extend/\*\* 用于框架的扩展，可选，具体参见[框架扩展](https://eggjs.org/zh-cn/basics/extend.html)。

config/config.{env}.js 用于编写配置文件，具体参见[配置](https://eggjs.org/zh-cn/basics/config.html)。

config/plugin.js 用于配置需要加载的插件，具体参见[插件](https://eggjs.org/zh-cn/basics/plugin.html)。

test/\*\* 用于单元测试，具体参见[单元测试](https://eggjs.org/zh-cn/core/unittest.html)。

app.js 和 agent.js 用于自定义启动时的初始化工作，可选，具体参见[启动自定义](https://eggjs.org/zh-cn/basics/app-start.html)。关于agent.js的作用参见[Agent机制](https://eggjs.org/zh-cn/core/cluster-and-ipc.html" \l "agent-%E6%9C%BA%E5%88%B6)。

2.2由内置插件约定的目录：

app/public/\*\* 用于放置静态资源，可选，具体参见内置插件 [egg-static](https://github.com/eggjs/egg-static" \t "https://eggjs.org/zh-cn/basics/_blank)。

app/schedule/\*\* 用于定时任务，可选，具体参见[定时任务](https://eggjs.org/zh-cn/basics/schedule.html)。

若需自定义自己的目录规范，参见 [Loader API](https://eggjs.org/zh-cn/advanced/loader.html)

app/view/\*\* 用于放置模板文件，可选，由模板插件约定，具体参见[模板渲染](https://eggjs.org/zh-cn/core/view.html)。

app/model/\*\* 用于放置领域模型，可选，由领域类相关插件约定，如 [egg-sequelize](https://github.com/eggjs/egg-sequelize" \t "https://eggjs.org/zh-cn/basics/_blank)。

app：核心目录，controller文件夹(放置各接口文件的具体方法)、public静态资源文件夹(放置css、js、img、file等)，middle中间件文件夹(.js文件)，service数据处理组装文件夹、view层展现相关文件夹以及router.js路由文件都在此目录下，这里也是以后大家使用egg开发的主要场所。

config：核心目录，配置文件相关，其中config.default.js中存放的是和当前Node环境无关的配置(可以放置middleware、view、mongoose、security等设置)；config.[env].js文件则存放和Node执行环境相关的配置；plugin.js存放的则是各个插件的package名称和是否开启的配置。

logs：日志文件输出的目录。

index.js：项目的入口文件。(如果直接初始化一个egg框架是没有入口文件的)

##### 3.渲染模板

3.1 先安装对应的插件 [egg-view-nunjucks](https://github.com/eggjs/egg-view-nunjucks" \t "https://eggjs.org/zh-cn/intro/_blank) ：

$ npm i egg-view-nunjucks --save

3.2 开启插件：

// config/plugin.js  
exports.nunjucks = {  
 enable: true,  
 package: 'egg-view-nunjucks'  
};

// config/config.default.js

exports.keys = <此处改为你自己的 Cookie 安全字符串>;

// 添加 view 配置

exports.view = {  
 defaultViewEngine: 'nunjucks',  
 mapping: {  
 '.nj': 'nunjucks',  
 },  
};

3.3 新建一个index.nj文件

<!-- app/view/news/index.nj -->  
<html>  
 <head>  
 <title>Hacker News</title>  
 <link rel="stylesheet" href="/public/css/news.css" />  
 </head>  
 <body>

<h1>这是首页</h1>  
 </body>  
</html>

3.4 添加 Controller 和 Router

首先要知道：

第一：查看es6的语法中的Generator 函数以及async函数。然后如果有express框架的基础，你就要转换你的习惯了，先来了解以下的内容：

this.ctx: 当前请求的上下文 [Context](https://eggjs.org/zh-cn/basics/extend.html" \l "context) 对象的实例，通过它我们可以拿到框架封装好的处理当前请求的各种便捷属性和方法（相当于express框架的res,例如express框架渲染模板时res.render(‘/’,{title:’首页’});）。

this.app: 当前应用 [Application](https://eggjs.org/zh-cn/basics/extend.html" \l "application) 对象的实例，通过它我们可以拿到框架提供的全局对象和方法。

然后就再结合这个，你就会了。

|  |
| --- |
| *// app/controller/home.js* **const** Controller = require('egg').Controller;  **class** **NewsController** **extends** **Controller** {  **async** index() {  **await** **this**.ctx.render('news/index.nj', {title:’首页’});  } }  module.exports = NewsController;  *// app/router.js* module.exports = app => {  **const** { router, controller } = app;  router.get('/news', controller.home.index); }; |

这里的contorller.home.index 相当于 //app/controller/home.index

启动浏览器，访问 http://localhost:7001/news 即可看到渲染后的页面。

3.5在express框架中是这样子的：

可以直接用express生成器快速的生成一个express项目:

$ npm install express-generator -g

$ express -h

$ express express-example

$ cd express-example

$ cnpm install

$ node start

/app.js

app.set(‘view’,path.join(\_\_dirname,’view’));

app.set(‘view engine’,’ejs’);

/views/index.ejs

<html>  
 <head>  
 <title><%= title %></title>  
 <link rel="stylesheet" href="/public/css/news.css" />  
 </head>  
 <body>

<h1>这是首页</h1>  
 </body>  
</html>

/routes/user.js

exports.index= function index(req, res,next) {

res.render('/',{

title:'首页'

})

/routes.js

var express = require('express');

var router = express.Router();

const user = require(‘./toutes/user’);//你要导入这个js

router.get('/',user.index);

module.exports = router;

/app.js (增加的，其他的在这里就不展示了)

const routes = require(‘./routes’);

app.use(‘/’,routes);

##### 4.链接数据库

4.1 安装插件：(这是egg框架中内置的,so下载时，就这样)

$ npm i egg-mongoose --save

4.2开启插件

//config/plugin.js

exports.mongoose = {

enable: true,

package: 'egg-mongoose',

};

// config/config.default.js （放在这个js里的，你可以在其他地方调用哦，是这样的app.config.\*\*,例如你在router.js中，console.log(app.config.mongoose.url);）

exports.mongoose = {

url: 'mongodb://127.0.0.1/example',

options: {}

};

此时链接数据库成功。

4.3然而express框架是这样的:

//app/model/db.js

const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/example'');

var db = mongoose.connection;

db.on('error',function () {

console.log('数据库链接失败了');

})

db.once('open',function () {

console.log('数据库链接成功');

})

module.exports = db;

它虽然不分开写，可是到创建集合时，你会发现，每个集合里都要引入它，很麻烦的，不过我已经习惯，用egg反而不习惯。

##### 5.创建集合

在app文件夹下新建model文件夹,再创建需要的文件(例如：//app/model/user.js)，你不需要导入mongoose模块，你也不需要导入连接数据库的js。

module.exports = app => {

const mongoose = app.mongoose;

const UserSchema = new mongoose.Schema({

name: {

type: String

},

age: {

type: Number

},

phoneNumber: {

type:Number

}

});

return mongoose.model('User', UserSchema);

}

在express框架中是这样的：

const mongoose = require('mongoose');

const db = require('./db');//你必须引入连接数据库的db.js

const UserSchema = mongoose.Schema({

name: {

type: String

},

age: {

type: Number

},

phoneNumber: {

type:Number

}

})

UserSchema.statics = {//这里你可以写一些，查找的方法，供其他路由查找集合里的数据使用。

findUserById:(id,callback)=>{

User.findById({'\_id':id}).then((user)=>{

callback(null, user);

}).catch((err) => {

callback(err);

})

},

}

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

module.exports = User;

##### 6.分析路由

6.1 如果不想将路由分开写：

//app/router.js

model.exports = app => {

app.get(‘/index/user/user.html’,’index.user.renderPage’);

app.get(‘/index/user/userInfor’,’index.user.getUserInfor);

}

//app/controller/index/user.js

module.exports = app => {

return class extends app.controller{

async renderPage(){

await this.ctx.render(‘../view/user.html’,{title:’首页’});

}

async getUserInfor(ctx) {

try{

let res = await ctx.model.User.find();

ctx.body = {code:200,msg:’success’,data:res};

}catch(error){

console.log(error);

Throw new Error(error);

}

}

}

}

6.2分开写路由这样写：

就是在app中在新建一个router文件夹放置分发的路由文件

//app/router.js

moule.exports = app => {

require(‘./router/index/user’)(app);//引入分开的路由

};

//app/router/index/user.js

modules.exports = app => {

const {router,controller} = app;

router.get(‘/user/index/user.html’,contorller.index.user.renderPage);//渲染页面

router.get(‘/index/user/getUserInfor’,controller.index.user.getUserInfor);

}

##### 7.从数据库查找数据，再给客户端发送响应数据

不需要给函数添加参数（req,res,next），直接使用this.ctx代替req、res。而且也不用导入user.js，以及查找集合里的数据也可以一句搞定，通过ctx.set(key,value)方法可以设置响应头。

// app/controller/home.js

const Controller = require('egg').Controller;  
  
class NewsController extends Controller {  
 async index() {  
 await this.ctx.render('news/index.nj', {title:’首页’});  
 }

async user(){

try {

let query = this.ctx.request.body;//post请求的请求体中的内容（相当于express中的req.body）

let name = query.params.name;

let pwd = query.params.pwd;

let user= await this.ctx.model.User.findOne({name:name});

if(!user){

console.log('不存在');

let people = {

name:name,

pwd:pwd

}

let newUser = await this.ctx.model.User.create(people);

let error;

let message = '';

let success = {

error:0,

message:'注册成功',

}

this.ctx.body = { code: 200,msg: 'success',data:user};

}else{

// 请去登陆

console.log('存在');

let successed = {

error:1,

message:'已注册，请登陆'

}

this.ctx.body = { code: 200,msg: 'successed',data:user};

}

} catch (error) {

throw error;

}

};  
  
module.exports = NewsController;

// app/router.js

module.exports = app => {  
 const { router, controller } = app;  
 router.get('/news', controller.home.index);

Router.post(‘/user’,controller.home.user);  
};

在express框架里是这样的：

/routes/user.js

const User = require('../model/user');

exports.user= (req, res, next) => {

uname = req.body.uname;

password = req.body.password;

User.findOne({ uname: uname }).exec().then((user) => {

if (user) {

return res.json({ error: 1, message: '该用户已注册,请登录....' });

}

let user= new User(req.body);

user.save().then((result)=>{

res.end( '保存成功.' );

}).catch((err)=>{

res.end(err);

})

res.json({ error: 0, message: 'success' });

}).catch((err) => {

res.json({ error: 1, message: '注册失败' });

})

};

##### 8.登录时设置cookie以及session

 ctx.cookies.get(key, options);//获取cookie中的某个key值对应的value

this.ctx.cookies.set(key,value,options);//设置cookie,添加字段到cookie中。

this.ctx.session;//获取session

this.ctx.session.userId = ‘234567’;//给session添加字段

async loginS(){

try{

let getUserInfo = this.ctx.request.body.params;

let nameL = getUserInfo.name;

let pwdL = getUserInfo.pwd;

let error;

let message = '';

let fail;

let userL = await this.ctx.model.User.findOne({name:nameL});

if(userL ){

// 验证用户

console.log('有该用户');

if(userL .pwd != pwdL){

let success = {

error:1,

message:'密码错误'

}

this.ctx.body = success;

}else{

let successed = {

error:0,

message:'登陆成功'

}

//设置cookie和session

let clientId = `${nameL}##`;

if(!this.ctx.cookies.get('loginUser')){

this.ctx.cookies.set('loginUser',clientId,{

maxAge: new Date(2100, 1, 1, 0, 0, 0)

});

}

if(!this.ctx.session.user){

this.ctx.session.user= `管理员的名字为${nameL}`

}

console.log(`${this.ctx.session.user} ,

登陆时间:

${new Date().getFullYear()}年

${new Date().getMonth()}月

${new Date().getDate()}日

${new Date().getHours()}点

${new Date().getMinutes()}`);

this.ctx.body = successed;

}

}else{

// // 提醒用户你未注册，请去登陆

console.log('没有');

let without = {

fail:1,

message:'未注册'

}

this.ctx.body = without;

}

} catch (error) {

console.log(error);

throw new Error(error);

}

}

##### 9.退出时设置cookie以及清空session

this.ctx.cookies.set(key,'');//删除cookie里的某个key值

this.ctx.session = null;//清空session

async logout(){

this.ctx.cookies.set('loginUser','');

if(this.ctx.session){

console.log(`${this.ctx.session} ,

退出时间:

${new Date().getFullYear()}年

${new Date().getMonth()}月

${new Date().getDate()}日

${new Date().getHours()}点

${new Date().getMinutes()}`);

this.ctx.session = null;

}

if(this.ctx.session == null){

console.log('退出成功');

}

this.ctx.body = '退出成功';

};

##### 10.异常处理

因为框架支持的异步编程模型，错误完全可以用 try catch 来捕获。在编写应用代码时，所有地方都可以直接用 try catch 来捕获异常。

按照正常代码写法，所有的异常都可以用这个方式进行捕获并处理，但是一定要注意一些特殊的写法可能带来的问题。打一个不太正式的比方，我们的代码全部都在一个异步调用链上，所有的异步操作都通过 await 串接起来了，但是只要有一个地方跳出了异步调用链，异常就捕获不到了。

框架提供了 ctx.runInBackground(scope) 辅助方法，通过它又包装了一个异步链，所有在这个 scope 里面的错误都会统一捕获。

async editSend(){

try {

let param = this.ctx.request.body.params;

let id = param.id;

let user = await this.ctx.model.User.findOne({\_id:id});

let name = param.name;

let age = param.age;

let phoneNumber = param.phoneNumber;

let userN = {

name:name,

age:age,

phoneNumber:phoneNumber

}

This.ctx.runInBackground(async () => {

// 这里面的异常都会统统被 Backgroud 捕获掉，并打印错误日志

await this.ctx.model.User.update(user,userN);

});

this.ctx.body = '编辑成功';

} catch (error) {

console.log(error);

throw new Error(error);

}

};

##### 11.Service请求级别的对象

你可以将一些数据库查找方法，写在这里，也可以将一些复杂的方法写在这里，供其他router调用。

注意：

1. 一个service文件只能包含一个类，这个类需要通过module.exports的方式返回。
2. Service需要通过calss的方式定义，父类必须是egg.Service.
3. service不是单例，是请求级别的对象，框架在每次请求中首次访问，ctx.service.xx时延迟实例化，所以service中可以通过this.ctx获取到当前请求的上下文。

11.1 步入接口：

1.首先安装插件:

cnpm i ali-oss --save

2.将oss的账号密码，bucket等信息写在config.default.js中。

//config.default.js

config.oss = {

accessKeyId: process.env.OSS\_ID || 'LTAI3cuPYRRhFYUk',

accessSecret: process.env.OSS\_Secret || 'jeFtIsxBjqX3ZL1Ojy2RikRk97zVQc',

endpoint: process.env.OSS\_endpoint || 'oss-cn-hangzhou.aliyuncs.com ',

bucket: process.env.OSS\_BUCKET || 'yuanliting'

};

解释：

accessKeyId:自己或公司的oss账号

accessSecret:oss密码

endpoint:看下面的图

Bucket:这个很重要，这是你创建bucket时，设置的bucket名字,看下面的图片



3. 链接oss，将文件上传到oss的方法以及获取oss上的文件的方法，都写在service文件夹中的oss.js中，供之后对oss操作时使用。

//app/service/oss.js 后面report2018.js调用oss.js中的方法

const OSS = require('ali-oss').Wrapper;

module.exports = app => {

let clientBucket = app.config.oss.bucket;//app上的方法，无需引入

let client = new OSS({

endpoint: app.config.oss.endpoint,

accessKeyId: app.config.oss.accessKeyId,

accessKeySecret: app.config.oss.accessSecret,

bucket: clientBucket

});

return class extends app.Service {

setConfig(bucket, endpoint, accessKeyId, accessSecret){

if(bucket == clientBucket) return;

clientBucket = bucket;

client = new OSS({

endpoint: endpoint || app.config.oss.endpoint,

accessKeyId: accessKeyId || app.config.oss.accessKeyId,

accessKeySecret: accessSecret || app.config.oss.accessSecret,

bucket: bucket

});

}

//设置文件，提交到oss上

\*setContent(fileName, content){

try {

yield client.put(fileName, new Buffer(content));

} catch(error) {

}

}

//获取oss上的文件

\*getContent(fileName){

try {

let result = yield client.get(fileName);

return Buffer.from(result.content).toString();

} catch(error) {

return false;

}

}

//可能时获取多个oss文件

\*getObjects(path){

try {

let files = [];

let result = {

isTruncated: true

};

while(result.isTruncated) {

result = yield client.list({

prefix: path,

marker: result.nextMarker

});

result.objects.map((file) => {

files.push(file);

})

}

return files

} catch(error) {

return false;

}

}

}

}

4. 写操作oss的方法，也就是去调用oss的方法：

//app/service/report2018.js

module.exports = app => {

return class extends app.Service {

//将文件存储到OSS上

\*build(sampleCode) {

const oss = this.service.oss;

oss.setConfig(app.config.oss.bucket);

let reporting = {};

reporting.name = 'yuan',

reporting.age = 23;

yield oss.setContent('report/' + sampleCode + '-' + reporting.name

+ '.json', JSON.stringify(reporting));//oss.setContent(文件路径，文件 内容)

return reporting;

}

//获取OSS上的文件

\*getReportContent(sampleCode) {

const oss = this.service.oss;

oss.setConfig(app.config.oss.bucket);

let reportingContent = yield oss.getContent('report/' + sampleCode

+ '.json');

let reportingN = JSON.parse(reportingContent);

console.log(123456,reportingN);

return reportingN;

}

}

}

其中report是，oss的一个bucket下的report文件夹

5.铺垫这么多，终于要调用前面的service文件夹下的各个文件中的方法了。

//app/controller/report2018.js

module.exports = app => {

return class extends app.Controller {

\*index( ctx ) {

try {

const reportService = ctx.service.report2018;

const sampleCode = ctx.request.body.params;

const reporting = yield reportService.build(sampleCode);

ctx.body = {code:200, msg:'success'};

} catch ( error ) {

console.log( error );

throw new Error( error );

}

}

\*getReport( ctx ) {

const reportService = ctx.service.report2018;

const sampleCode = ctx.request.body.params;

const reporting = yield reportService.getReportContent(sampleCode);

console.log( '获取的数据', reporting );

ctx.body = {code:200, msg:'success'};

}

}

};

6.就是把路由写上去了，这样你的接口才起到作用，此处省略。（寒风呼呼的吹，哈哈哈）

7.就是你页面发请求，以及请求的路由信息，还是省略。(over)

##### 不同服务器的跨域问题

1. egg后台需要的
   1. 安装插件

$ npm i egg-cors --save

* 1. 开启插件

//app/config/config.default.js

exports.cors **=** {

  enable**:** true,

  package**:** 'egg-cors',

};

* 1. 只允许本地跨域

//app/config/config.local.js

//本地跨域

exports.cors = {

origin: '\*',

allowMethods: 'GET,HEAD,PUT,POST,DELETE,PATCH'

};

这样就没后端的事了，接下来就是前端的请求了

1. vue前端发送跨域请求