# Egg

Nodejs和koa整合的一个企业级的开发。品。

便于代码分层。

官网：<https://eggjs.org/zh-cn/intro/quickstart.html>

新建一个文件夹，然后在cd到文件夹里面，再

npm init egg --type=simple

npm i / npm install

npm run dev

Vscode 编写egg，安装egg插件，能更多的智能提示

# 路由配置

统一在router.js配置路由，在controller配置控制器。

Router.js 配置拦截/ 请求，并转发到home.js下的index方法里。

|  |
| --- |
| router.get('/', controller.home.index); |

Home.js

|  |
| --- |
| async index() {  const { ctx } = this;  ctx.body = 'hi, egg';  } |

# Post和get接收参数

在控制器，不需要整合模块

接收get请求的参数：this.ctx.query();

接收post请求的参数：this.ctx.request.body;

# 页面模板引擎

用ejs是因为可以使用后台数据对页面渲染。类似java的el和jstl表达式

1.添加依赖

npm i egg-view-ejs

2.在{app\_root}/config/plugin.js

exports.ejs = { enable: true, package: 'egg-view-ejs', };

1. 在{app\_root}/config/config.default.js

//配置 ejs 模板引擎

config.view = { mapping: { '.html': 'ejs', } };

1. 使用(它会在app/view 下找模板 ,所以直接写页面名就行)

Await This.ctx.render(“xx”,{});

# 图片、静态资源路径

在页面使用/public/xxx/xx.img /public/css/xx.css 路径去访问

# Controller和service

使用vscode ，安装egg插件。在建立好文件后，直接使用egg controller或者egg service 生成对应的代码。

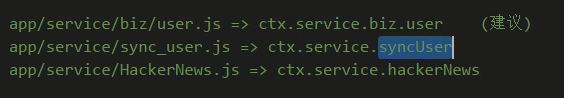
注意：controller和service都是创建在app文件夹下。

Router.js 指定controller处理

Controller调用service：await this.service.xx.xx //具体调用哪个js文件下的方法。

例子 await this.service.user.getUserInfo(); 调用service/user.js下的getUserInfo方法。

具体调用看图：（注意大小写）



定义的 Controller 类，会在每一个请求访问到 server 时实例化一个全新的对象，而项目中的 Controller 类继承于 egg.Controller，会有下面几个属性挂在 this 上。

This.app this.ctx this.service this.config this.logger

# Service

Service 文件必须放在 app/service 目录，可以支持多级目录，访问的时候可以通过目录名级联访问。

app/service/biz/user.js => ctx.service.biz.user

app/service/sync\_user.js => ctx.service.syncUser

app/service/HackerNews.js => ctx.service.hackerNews

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。

Service 需要通过 Class 的方式定义，父类必须是 egg.Service。

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。

# 获取接口的json数据

获取这个接口的数据：http://www.phonegap100.com/appapi.php?a=getPortalList&catid=20&page=1

Await ctx.this.curl(“”) 访问，返回的是一个buffer的数据，通过JSON.parse转换为json对象。



# Extend扩展

<https://eggjs.org/zh-cn/basics/extend.html>

# 将时间戳转为时间格式

使用silly-datetime 模块

Npm install silly-datetime

const dateFormat = require("silly-datetime");

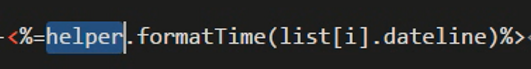
其他格式：<https://blog.csdn.net/wanghui374/article/details/96605134>

console.log(dateFormat.format(new Date(时间戳), 'YYYY-MM-DD HH:mm:ss'));

将时间戳转为日期类型，然后再转为指定的日期格式。

注意转为日期类型，需要的是时间戳是13位数字的。有些是10位数字的，需要\*1000

可以扩展helper，然后添加转换日期的方法，就可以在模板调用了。



# 配置中间件

1.中间件是一个放置在 app/middleware 目录下的单独文件，它需要 exports 一个普通的 function，接受两个参数：

options: 中间件的配置项，框架会将 app.config[${middlewareName}] 传递进来。

app: 当前应用 Application 的实例。

1.在app/middleware下面新建forbidip.js，编写中间件：

|  |
| --- |
| module.exports = (options, app) => {  return async function forbidipMiddleware(ctx, next) {  Await next();  console.log(options.forbidip); //配置时传入的参数  }  }; |

2.在config.default.js

|  |
| --- |
| //配置中间件  config.middleware = ['forbidip'];  //为中间件传参数  config.forbidip = {  ip: [  '127.0.0.1',  '192.168.0.1'  ],  }; |

# 表单提交问题（安全问题）

Egg在提交表单时，会进行验证，防止csrf攻击。否则，egg会拒绝访问。

所以：

1.打开表单页面是，需要返回一个csrf值。

2.请求时，表单或者url上必须带着csrf。

3.后台获取数据时，直接使用ctx.request.body。

在后台，通过ctx.csrf 每次获取都是不一样的值。

所以每次渲染页面时，需要传入参数

|  |
| --- |
| Let csrf=ctx.csrf;  Await ctx.render(“index”,{csrf}); |

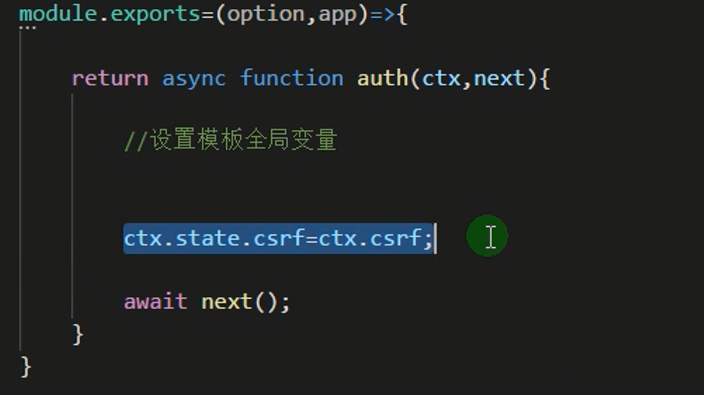
然后页面需要将这个csrf拼接到url上，或者是一个hidden的text中。：

|  |
| --- |
| <form action="/news/doAdd?\_csrf=<%=csrf%> " method="POST">  用户名: <input type="text" name="username" /> <br><br>  密 码: <input type="text" name="password" type="password" />  <button type="submit">提交</button>  </form> |
| <form action="/news/doAdd> " method="POST">  <input type="hidden" name="\_csrf" value="<%=csrf%>">  用户名: <input type="text" name="username" /> <br><br>  密 码: <input type="text" name="password" type="password" />  <button type="submit">提交</button>  </form> |

这样，egg就不会拒绝请求了。

进阶：

每次渲染有form表单的页面，都需要传入csrf值。这时候，可以配置一个中间件，让他每次都生成一个全局变量，就不用在渲染时传入了。而是能够直接用。



具体配置看中间件。

## 疑惑

内部是如何实现的，是通过生成csrf后，访问带上csrf，是有判断前后是否一致吗。还是说只判断是否存在这个csrf。存在就通过

真的能防止csrf吗，通过访问页面获取到csrf的值，然后拼接到自己页面的form表单中，实验结果呢。

使用全局变量，会出现线程安全吗，就是两个线程，线程1访问后csrf是1，然后form表单传到后台。这时候线程2访问，将csrf改为2。这时候线程1的csrf值和后台的全局变量的值不一致。？？？

# Cookie

Ctx.cookies.set(“name”,”value”,options)

Ctx.cookies.get(“name”);

清除：ctx.cookies.set('name',null);

建议写法：

ctx.cookies.set(key, value, {

maxAge:24 \* 3600 \* 1000,

httpOnly: true, // by default it's true

encrypt: true, // cookies are encrypted during network transmission

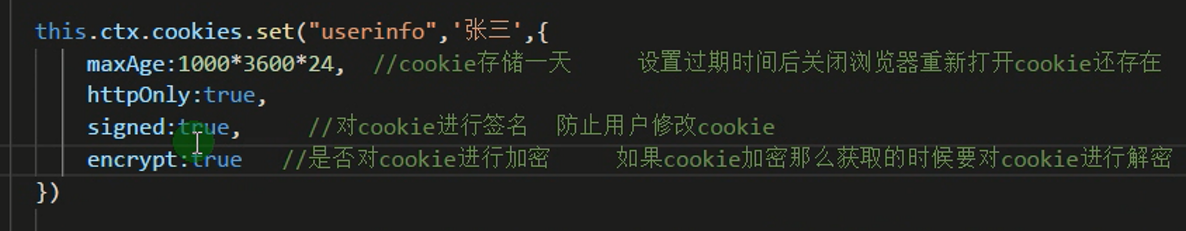
});

ctx.cookies.get('frontend-cookie', {

encrypt: true

});

Set的时候设置encrypt为true，即对cookie进行加密，那么中文也会被编码，获取的时候get，同时设置参数，那么就会进行解密。这样就可以使得cookie存储中文。



# Session使用

ctx.session.userinfo={ name:'张三', age:'20' }

var userinfo=this.ctx.session.userinfo

session的默认值：

exports.session = {

key: 'EGG\_SESS',

maxAge: 24 \* 3600 \* 1000, // 1 day

httpOnly: true,

encrypt: true

};

修改默认值（在config.default.js）：

|  |
| --- |
| config.session={  key:'SESSION\_ID',  maxAge:864000, //这个到底是保存在服务端的session的过期，还是保存在浏览器cookie的sessionid的过期时间  renew: true //延长会话有效期  } |

# 文件上传

1.先配置

|  |
| --- |
| // config.defult.js  config.multipart = {  mode: 'file'  }; |

2.使用

|  |
| --- |
| // 获取上传参数  const params = this.ctx.request.body;  // 获取上传文件  const files = this.ctx.request.files; |

# Mysql

[https://eggjs.org/zh-cn/tutorials/mysql.html#egg-mysql](https://eggjs.org/zh-cn/tutorials/mysql.html" \l "egg-mysql)