link: null
title: 珠峰架构师成长计划
description: app/router.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=263 sentences=540, words=3214

1. egg.js

```
├── package.json
├── app.js (app.js 和 agent.js 用于自定义启动时的初始化工作)
├── agent.js (可选)
├── app
|   ├── router.js(用于配置 URL 路由规则)
|   ├── controller(用于解析用户的输入，处理后返回相应的结果)
|   |   └── home.js
|   ├── service (用于编写业务逻辑层，可选)
|   |   └── user.js
|   ├── middleware (用于编写中间件，可选)
|   |   └── response_time.js
|   ├── schedule (用于定时任务，可选)
|   |   └── my_task.js
|   ├── public (用于放置静态资源，可选)
|   |   └── reset.css
|   ├── extend (用于框架的扩展，可选)
|   |   └── application.js app 对象指的是 Koa 的全局应用对象，全局只有一个，在应用启动时被创建。
|   |       ├── context.js (Context 指的是 Koa 的请求上下文，这是 请求级别 的对象)
|   |       ├── request.js (Request 对象和 Koa 的 Request 对象相同，是 请求级别 的对象)
|   |       ├── response.js (Response 对象和 Koa 的 Response 对象相同，是 请求级别 的对象)
|   |       ├── helper.js (Helper 函数用来提供一些实用的 utility 函数)
|   ├── view (用于放置模板文件)
|   |   └── home.tpl
├── |── model (用于放置领域模型)
|   |   └── home.tpl
|   └── extend (用于框架的扩展)
|       ├── 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(用于编写配置文件)
|   ├── plugin.js(用于配置需要加载的插件)
|   ├── config.default.js
|   ├── config.prod.js
|   ├── config.test.js (可选)
|   ├── config.local.js (可选)
|   └── config.unittest.js (可选)
└── test(用于单元测试)
    ├── middleware
    |   └── response_time.test.js
    └── controller
        └── home.test.js
```

文件 app ctx service config logger helper Controller this.app this.ctx this.service this.config this.logger this.app.helper Service this.app this.ctx this.service this.config this.logger this.app.helper

> ctx.helper

2. 初始化项目

```
mkdir egg-news
cd egg-news
npm init -y
npm i egg --save
npm i egg-bin --save-dev
npm i mockjs express morgan egg-mock --save
```

3. 添加 npm scripts 到 package.json：

```
"scripts": {
    "dev": "egg-bin dev"
}
```

4. 跑通路由

```
├─app
|  |─router.js
|  ├─controller
|  |      news.js
├─config
|      config.default.js
|─package.json
```

app/router.js

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

app\controller\news.js

```
const { Controller } = require('egg');
class NewsController extends Controller {
    async index() {
        this.ctx.body = 'hello world';
    }
}
module.exports = NewsController;
```

config\config.default.js

```
exports.keys = 'zhufeng';
```

5. 静态文件中间件

- Egg 内置了 static 插件
- static 插件默认映射 /public/ -> *app/public/* 目录
- 把静态资源都放到 app/public 目录即可
- bootstrap (http://img.zhufengpeixun.cn/bootstrap.zip)

6. 使用模板引擎

```
├─app
│  ├─router.js
│  ├─controller
│  │      news.js
│  ├─public
│  │  ├─css
│  │  │      bootstrap.css
│  │  └─js
│  │          bootstrap.js
│  └─view
│          index.html
├─config
│  config.default.js
│  plugin.js
```

```
npm install egg-view-nunjucks --save
```

{ROOT}\config\plugin.js

```
exports.nunjucks = {
    enable: true,
    package: 'egg-view-nunjucks'
}
```

{ROOT}\config\config.default.js

```
module.exports=app => {
    let config={};
    config.keys='zhufeng';
    config.view={
        defaultExtension: '.html',
        defaultViewEngine: 'nunjucks',
        mapping: {
            '.html':'nunjucks'
        }
    }
    return config;
}
```

app\view\index.html

```
    新闻列表

           {% for item in list%}

                       {{item.title}}

                        创建时间: {{item.createAt}}

            {% endfor %}
```

app\controller\news.js

```
const {Controller}=require('egg');
class NewsController extends Controller{
    async index() {
        const {ctx}=this;
        const list=[
            {
                id: '45154322_0',
                title: '世界首富早晚是这个人，坐拥7家独角兽公司，估值破数万！',
                url: 'http://tech.ifeng.com/a/20180904/45154322_0.shtml',
                image:'http://p0.ifengimg.com/pmop/2018/0905/CFFF918B94D561D2A61FB434ADA81589E8972025_size41_w640_h479.jpeg',
                createAt:new Date().toLocaleString()
            },
            {
                id: '16491630_0',
                title: '支付宝们来了！将来人民币会消失吗？',
                url: 'http://finance.ifeng.com/a/20180907/16491630_0.shtml',
                image:'http://p0.ifengimg.com/pmop/2018/0907/2AF684C2EC49B7E3C17FCB13D6DEEF08401D4567_size27_w530_h369.jpeg',
                createAt:new Date().toLocaleString()
            },
            {
                id: '2451982',
                title: '《福布斯》专访贝索斯：无业务边界的亚马逊 令对手生畏的CEO',
                url: 'https://www.jiemian.com/article/2451982.html',
                image:'https://img1.jiemian.com/101/original/20180907/153628523948814900_a580x330.jpg',
                createAt:new Date().toLocaleString()
            }
        ];
        await ctx.render('index',{list});
    }
}
module.exports=NewsController;
```

7. 读取远程接口服务

在实际应用中，Controller 一般不会自己产出数据，也不会包含复杂的逻辑，复杂的过程应抽象为业务逻辑层 Service。

config.default.js

```
config.news={
       pageSize:10,
    newsListUrl:'http://localhost:3000/news'
}
```

mock.js

```
let Mock = require('mockjs');
let express = require('express');
let logger = require('morgan');
let app = express();
app.use(logger('dev'));
app.get('/news', function (req, res) {
    let result = Mock.mock(
        {
            "data|10": [{
                "id": "@id",
                "title": "@csentence",
                "url": "@url",
                "image": "@image(600X500)",
                "createAt": "@datetime",
            }]
        });
    res.json(result);
});
app.get('/cache', function (req, res) {
    res.json({ title: '新闻标题' + Date.now() });
});
app.listen(3000, () => { console.log('mock server is running at port 3000') });
```

app/service/news.js

```
const {Service}=require('egg');
class NewsService extends Service {
    async list(pageNum,pageSize) {
        const {ctx}=this;
        const {newsListUrl}=this.config.news;
        const result=await ctx.curl(newsListUrl,{
            method: 'GET',
            data: {
                pageNum,pageSize
            },
            dataType:'json'
        });
        return result.data.data;
    }
}
module.exports=NewsService;
```

app/controller/news.js

```
const {Controller}=require('egg');
class NewsController extends Controller{
    async index() {
        const {ctx,service}=this;
        let {pageNum=1,pageSize=this.config.news.pageSize}=ctx.query;
        const list=await service.news.list(pageNum,pageSize);
        await ctx.render('index',{list});
    }
}
module.exports=NewsController;
```

8. 计划任务

我们还会有许多场景需要执行一些定时任务，例如：

- 定时上报应用状态。

- 定时从远程接口更新本地缓存。
- 定时进行文件切割、临时文件删除
- 所有的定时任务都统一存放在 app/schedule 目录下，每一个文件都是一个独立的定时任务，可以配置定时任务的属性和要执行的方法

app\schedule\update_cache.js

```
const { Subscription } = require('egg');
class UpdateCache extends Subscription {

    static get schedule() {
        return {
            interval: '1m',
            type: 'all',
        };
    }

    async subscribe() {
        console.log('subscribe');
        const res = await this.ctx.curl(this.config.cache.url, {
            dataType: 'json',
        });
        this.ctx.app.cache = res.data;
    }
}

module.exports = UpdateCache;
```

- 框架提供的定时任务默认支持两种类型，worker 和 all。worker 和 all 都支持上面的两种定时方式，只是当到执行时机时，会执行定时任务的 worker 不同：
    - worker 类型：每台机器上只有一个 worker 会执行这个定时任务，每次执行定时任务的 worker 的选择是随机的。
    - all 类型：每台机器上的每个 worker 都会执行这个定时任务。
- 执行日志会输出到 ${appInfo.root}/logs/{app_name}/egg-schedule.log

config\config.default.js

```
+config.cache = {
+   url: 'http://localhost:3000/cache',
+}
```

mock.js

```
app.get('/cache',function(req,res){
 res.json({title:'新闻列表'+Date.now()});
});
```

app\controller\news.js

```
const {Controller} = require('egg');
class NewsController extends Controller{
    async index(){
        const {ctx,service}=this;
        let {pageNum=1,pageSize=this.config.news.pageSize}=ctx.query;
        const list=await service.news.list(pageNum,pageSize);
+       await ctx.render('index',{list,title:this.app.cache?this.app.cache.title:'新闻列表'});
    }
}
module.exports = NewsController;
```

app\view\index.html

```
+   {{title}}

          {% for item in list%}

                      {{item.title}}

                       创建时间：{{item.createAt}}

           {% endfor %}
```

- 框架提供了统一的入口文件（app.js）进行启动过程自定义 app.js

```
module.exports = app => {
app.beforeStart(async () => {

  await app.runSchedule('update_cache');
});
};
```

9. MySQL

```
npm i --save egg-mysql
```

config/plugin.js

```
exports.mysql = {
  enable: true,
  package: 'egg-mysql',
};
```

```
CREATE TABLE `news`  (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL,
  `url` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL,
  `image` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NULL DEFAULT NULL,
  `createAt` datetime NULL DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 4 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_bin ROW_FORMAT = Compact;

INSERT INTO `news` VALUES (1, '世界首富早晚是这个人，坐拥7家独角兽公司，估值破数万！', 'http://tech.ifeng.com/a/20180904/45154322_0.shtml',
'http://p0.ifengimg.com/pmop/2018/0905/CFFF918B94D561D2A61FB434ADA81589E8972025_size41_w640_h479.jpeg', '2019-06-08 22:07:29');
INSERT INTO `news` VALUES (2, '支付宝们来了！将来人民币会消失吗？', 'http://finance.ifeng.com/a/20180907/16491630_0.shtml',
'http://p0.ifengimg.com/pmop/2018/0907/2AF684C2EC49B7E3C17FCB13D6DEEF08401D4567_size27_w530_h369.jpeg', '2019-06-08 22:08:24');
INSERT INTO `news` VALUES (3, '《福布斯》专访贝索斯：无业务边界的亚马逊 令对手生畏的CEO', 'https://www.jiemian.com/article/2451982.html',
'https://img1.jiemian.com/101/original/20180907/153628523948814900_a580x330.jpg', '2019-06-08 22:17:16');
```

config/config.${env}.js

```
config.mysql = {

  client: {

    host: 'localhost',

    port: '3306',

    user: 'root',

    password: 'root',

    database: 'cms'
  },

  app: true,

  agent: false,
};
```

app\service\news.js

```
const {Service}=require('egg');
class NewsService extends Service {
    async list(pageNum,pageSize) {
        const {ctx}=this;
        let result  = await this.app.mysql.query('select * from news');
        return result;
    }
}
module.exports=NewsService;
```

10. Sequelize

- 在一些较为复杂的应用中，我们可能会需要一个 ORM 框架来帮助我们管理数据层的代码。而在 Node.js 社区中，sequelize 是一个广泛使用的 ORM 框架，它支持 MySQL、PostgreSQL、SQLite 和 MSSQL 等多个数据源
- egg-sequelize插件会辅助我们将定义好的Model对象加载到 app和 ctx上

```
$ npm install --save egg-sequelize mysql2
```

```
exports.sequelize = {
  enable: true,
  package: 'egg-sequelize',
};
```

config/config.default.js

```
config.sequelize = {
    dialect: 'mysql',
    host: 'localhost',
    port: 3306,
    username: "root",
    password: "root",
    database: 'cms-development'
};
```

config/config.test.js

```
module.exports=app => {
    let config={};
    config.sequelize = {
        dialect: 'mysql',
        host: 'localhost',
        port: 3306,
        username: "root",
        password: "root",
        database: 'cms-test',
    };
    return config;
}
```

- sequelize 提供了 sequelize-cli 工具来实现 Migrations
- sequelize-cli用于支持数据迁移和项目引导。通过迁移，可以将现有数据库迁移到另一个状态，反之亦然
- 这些迁移文件会被保存在迁移文件中，迁移文件描述了怎样到达新状态以及如何恢复更改以返回到迁移前的旧状态

```
npm install --save sequelize sequelize-cli
```

- 我们希望将所有数据库 Migrations 相关的内容都放在 database 目录下，所以我们在项目根目录下新建一个 .sequelizerc 配置文件
    - config 包含配置文件，它告诉 CLI 如何连接数据库
    - migrations-path 包含所有迁移文件
    - seeders-path 包含所有种子文件,seeders 来在初始化数据表中中初始化一些基础数据
    - models-path 包含您的项目的所有模型

```
const path = require('path');
module.exports = {
  config: path.join(__dirname, 'database/config.json'),
  'migrations-path': path.join(__dirname, 'database/migrations'),
  'seeders-path': path.join(__dirname, 'database/seeders'),
  'models-path': path.join(__dirname, 'app/model'),
};
```

- 在项目的演进过程中，每一个迭代都有可能对数据库数据结构做变更，怎样跟踪每一个迭代的数据变更，并在不同的环境（开发、测试、CI）和迭代切换中，快速变更数据结构呢？这时候我们就需要 Migrations 来帮我们管理数据结构的变更了
- 初始化 Migrations 配置文件和目录
- 执行完后会生成 database/config.json 文件和 database/migrations

```
npx sequelize init:config
npx sequelize init:migrations
```

- key是环境变量 NODE_ENV的值，默认就是 development
- set NODE_ENV=test

config.json

```
{
  "development": {
    "username": "root",
    "password": "root",
    "database": "cms-development",
    "host": "127.0.0.1",
    "dialect": "mysql",
    "operatorsAliases": false
  },
  "test": {
    "username": "root",
    "password": "root",
    "database": "cms-test",
    "host": "127.0.0.1",
    "dialect": "mysql",
    "operatorsAliases": false
  }
}
```

- 编写项目的第一个 Migration 文件来创建我们的一个 users 表

```
npx sequelize migration:generate --name=init-users
```

> 执行完后会在 database/migrations 目录下生成一个 migration 文件(${timestamp}-init-users.js)

database\migrations\20190608143311-init-users.js

```
module.exports = {

  up: async (queryInterface, Sequelize) => {
    const { INTEGER, DATE, STRING } = Sequelize;
    await queryInterface.createTable('users', {
      id: { type: INTEGER, primaryKey: true, autoIncrement: true },
      name: STRING(30),
      age: INTEGER,
      created_at: DATE,
      updated_at: DATE,
    });
  },

  down: async queryInterface => {
    await queryInterface.dropTable('users');
  },
};
```

```
# 升级数据库
npx sequelize db:migrate
# 如果有问题需要回滚，可以通过 `db:migrate:undo` 回退一个变更
# npx sequelize db:migrate:undo
# 可以通过 `db:migrate:undo:all` 回退到初始状态
# npx sequelize db:migrate:undo:all
```

```
sequelize seed:create --name init-users
npx sequelize db:seed:all
npx sequelize db:seed:all --env development
```

database\seeders\20190803152323-init-users.js

```
module.exports = {
    up: (queryInterface, Sequelize) => {
        return queryInterface.bulkInsert('users', [{
            name: 'zhufeng',
            age:1,
            created_at: new Date(),
            updated_at: new Date()
        },{
            name: 'jiagou',
            age:2,
            created_at: new Date(),
            updated_at: new Date()
        }],{});
    },

    down: (queryInterface, Sequelize) => {
        return queryInterface.bulkDelete('users', null, {});
    }
};
```

app\model\user.js

```
module.exports = app => {
  const { STRING, INTEGER, DATE } = app.Sequelize;

  const User = app.model.define('User', {
    id: { type: INTEGER, primaryKey: true, autoIncrement: true },
    name: STRING(30),
    age: INTEGER,
    created_at: DATE,
    updated_at: DATE,
  });

  return User;
};
```

> 这个 Model 就可以在 Controller 和 Service 中通过 app.model.User 或者 ctx.model.User 访问到了

app\router.js

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

app\controller\users.js

```
const { Controller } = require('egg');
class UserController extends Controller {
    async index() {
       const {ctx,service}=this;
       ctx.body = await ctx.model.User.findAll();
    }
}
module.exports = UserController;
```

```
npx sequelize db:migrate --env test
```

- 通过 factory-girl 模块来快速创建测试数据

```
npm install --save-dev factory-girl
```

test/factories.js

```
const { factory } = require('factory-girl');

module.exports = app => {

  app.factory = factory;

  factory.define('user', app.model.User, {
    name: factory.sequence('User.name', n => `name_${n}`),
    age: 18,
  });
};
```

test/.setup.js

```
const { app } = require('egg-mock/bootstrap');
const factories = require('./factories');

before(() => factories(app));
afterEach(async () => {
  await Promise.all([
    app.model.User.destroy({ truncate: true, force: true }),
  ]);
});
```

test/app/controller/users.test.js

```

const { assert, app } = require('egg-mock/bootstrap');

describe('test/app/controller/users.test.js', () => {
  describe('GET /users', () => {
    it('should work', async () => {

      await app.factory.createMany('user', 3);
      const res = await app.httpRequest().get('/users');
      assert(res.status === 200);
      assert(res.body.length === 3);
      assert(res.body[0].name);
      assert(res.body[0].age);
    });
  });
});
```

```
"scripts": {
    "dev": "egg-bin dev",
+    "test": "egg-bin test"
}
```

```
npm run test
```

11. 国际化（I18n）

- 为了方便开发多语言应用，框架内置了国际化（I18n）支持，由 egg-i18n 插件提供。

// config/config.default.js

```
exports.i18n = {
  defaultLocale: 'zh-CN',
};
```

多种语言的配置是独立的，统一存放在 config/locale/*.js 下。

- config/locale/

- en-US.js
- zh-CN.js

config/locale/en-US.js

```
module.exports = {
    Email: 'Email',
    'Welcome back, %s!': 'welcome back, %s!',
    'Hello {0}! My name is {1}.': '你好 {0}! 我的名字叫 {1}。',
};
```

config/locale/zh-CN.js

```
module.exports = {
    Email: '邮箱',
    'Welcome back, %s!': '欢迎回来, %s!',
    'Hello {0}! My name is {1}.': '你好 {0}! 我的名字叫 {1}.',
};
```

- 我们可以使用 __ (Alias: gettext) 函数获取 locale 文件夹下面的多语言文本。

```
ctx.__('Email')
```

app\router.js

```
router.get('/hello', controller.news.hello);
```

app\controller\news.js

```
const { Controller } = require('egg');
class NewsController extends Controller {
      async index() {
        const { ctx, service } = this;
        let { pageNum = 1, pageSize = this.config.news.pageSize } = ctx.query;
        const list = await service.news.list(pageNum, pageSize);
+       await ctx.render('index', { list, name: 'zhufeng', names: ['zhufeng', 'jiagou'] });
    }
    async hello(){
        const {ctx}=this;
+        let email = ctx.__('Email');
+        let welcome = ctx.__('Welcome back, %s!', 'zhufeng');
+        let Hello = ctx.__('Hello {0}! My name is {1}.', ['zhufeng','jiagou']);
+        ctx.body = email+welcome+Hello;
    }
}
module.exports = NewsController;
```

app\view\index.html

```
{{__('Email')}}
{{__('Welcome back, %s!', name)}}
{{__('Hello {0}! My name is {1}.', names)}}
```

12. 扩展工具方法

- 框架提供了一种快速扩展的方式，只需在 app/extend目录下提供扩展脚本即可
- Helper 函数用来提供一些实用的 utility 函数。
- 访问方式 通过 ctx.helper 访问到 helper 对象

app\extend\helper.js

```
const moment=require('moment');
moment.locale('zh-cn');
exports.fromNow=dateTime => moment(new Date(dateTime)).fromNow();
```

app\controller\news.js

```
class NewsController extends Controller {
    async index() {
        const { ctx, service } = this;
        let { pageNum = 1, pageSize = this.config.news.pageSize } = ctx.query;
        const list = await service.news.list(pageNum, pageSize);
+       list.forEach(item => {
+            item.createAt = ctx.helper.fromNow(item.createAt);
+        });
        await ctx.render('index', { list, name: 'zhufeng', names: ['zhufeng', 'jiagou'] });
    }
}
```

或者 app\view\index.html

```
+    创建时间: {{helper.fromNow(item.createAt)}}
```

13. 中间件

app/middleware/robot.js

```
module.exports=(options,app) => {
    return async function(ctx,next) {
        const source=ctx.get('user-agent')||'';
        const matched=options.ua.some(ua => ua.test(source));
        if (matched) {
            ctx.status=403;
            ctx.body='你没有访问权限';
        } else {
            await next();
        }
    }
}
```

config.default.js

```
    config.middleware=[
        'robot'
    ]
    config.robot={
        ua: [
            /Chrome/
        ]
    }
```

```
curl -v  --user-agent 'Chrome'  http:
curl -v  --user-agent 'Chrome'  http:
```

14.运行环境

框架有两种方式指定运行环境：

- 通过 config/env 文件指定，该文件的内容就是运行环境，如 prod。
- 通过 EGG_SERVER_ENV 环境变量指定。
- 框架提供了变量 app.config.env 来表示应用当前的运行环境。
- 支持按环境变量加载不同的配置文件，如 config.local.js， config.prod.js 等等

EGG_SERVER_ENV 说明 local 本地开发环境 prod 生产环境

```
npm install  cross-env --save-dev
```

```
"scripts": {
    "dev": "cross-env EGG_SERVER_ENV=local  egg-bin dev",
    "debug": "egg-bin debug"
}
```

15. 单元测试

- 代码质量持续有保障
- 重构正确性保障
- 增强自信心
- 自动化运行
- 我们约定 test 目录为存放所有测试脚本的目录，测试所使用到的 fixtures 和相关辅助脚本都应该放在此目录下。
- 测试脚本文件统一按 ${filename}.test.js 命名，必须以 .test.js 作为文件后缀。 一个应用的测试目录示例：

```
test
├── controller
│   └── news.test.js
└── service
  └── news.test.js
```

统一使用 egg-bin 来运行测试脚本， 自动将内置的 Mocha、co-mocha、power-assert，nyc 等模块组合引入到测试脚本中， 让我们聚焦精力在编写测试代码上，而不是纠结选择那些测试周边工具和模块。

```
  "scripts": {
    "test": "egg-bin test",
    "cov": "egg-bin cov"
  }
```

- 如果要完整手写一个 app 创建和启动代码，还是需要写一段初始化脚本的， 并且还需要在测试跑完之后做一些清理工作，如删除临时文件，销毁 app。
- 常常还有模拟各种网络异常，服务访问异常等特殊情况。
- egg.js单独为框架抽取了一个测试 mock 辅助模块： egg-mock， 有了它我们就可以非常快速地编写一个 app 的单元测试，并且还能快速创建一个 ctx 来测试它的属性、方法和 Service 等。

```
npm i egg-mock -D
```

在测试运行之前，我们首先要创建应用的一个 app 实例， 通过它来访问需要被测试的 Controller、Middleware、Service 等应用层代码。

```

const { app, mock, assert } = require('egg-mock/bootstrap');

describe('test/controller/news.test.js', () => {

});
```

test/order.test.js

```
describe('egg test', () => {
  before(() => console.log('order 1'));
  before(() => console.log('order 2'));
  after(() => console.log('order 6'));
  beforeEach(() => console.log('order 3'));
  afterEach(() => console.log('order 5'));
  it('should worker', () => console.log('order 4'));
});
```

test/controller/news.test.js

```
const { app, mock, assert } = require('egg-mock/bootstrap');
describe('test/controller/news.test.js', () => {
  it('should get a ctx', () => {
    const ctx=app.mockContext({
          session: {
            user:{name:'zhufeng'}
        }
    });
    assert(ctx.method === 'GET');
    assert(ctx.url==='/');
    assert(ctx.session.user.name == 'zhufeng');
  });
});
```

test/controller/news.test.js

- egg-bin 支持测试异步调用，它支持多种写法

test/controller/user.test.js

app/router.js

```
router.get('/add',controller.user.add);
router.post('/doAdd',controller.user.doAdd);
```

app/controller/user.js

```
const {Controller}=require('egg');
let users=[];
class UserController extends Controller{
    async index() {
        let {ctx}=this;
        await ctx.render('user/list',{users});
    }
    async add() {
        let {ctx}=this;
        await ctx.render('user/add',{});
    }
    async doAdd() {
        let {ctx}=this;
        let user=ctx.request.body;
        user.id=users.length>0?users[users.length-1].id+1:1;
        users.push(user);
        ctx.body = user;
    }
}
module.exports=UserController;
```

test/controller/user.test.js

```
const { app, mock, assert } = require('egg-mock/bootstrap');
it('test post',async () => {
        let user={username: 'zhufeng'};
        app.mockCsrf();
        let response=await app.httpRequest().post('/doAdd').send(user).expect(200);
        assert(response.body.id == 1);
});
```

- Service 相对于 Controller 来说，测试起来会更加简单
- 我们只需要先创建一个 ctx，然后通过 `ctx.service.${serviceName}` 拿到 Service 实例， 然后调用 Service 方法即可。

test/service/user.test.js

```
const { app, mock, assert } = require('egg-mock/bootstrap');
const {app,assert}=require('egg-mock/bootstrap');
describe('test/service/news.test.js',() => {
    it('newsService',async () => {
        let ctx=app.mockContext();
        let result=await ctx.service.news.list(1,5);
        assert(result.length == 3);
    });
});
```

应用可以对 Application、Request、Response、Context 和 Helper 进行扩展。 我们可以对扩展的方法或者属性针对性的编写单元测试。

egg-mock 创建 app 的时候，已经将 Application 的扩展自动加载到 app 实例了， 直接使用这个 app 实例访问扩展的属性和方法即可进行测试。

app/extend/application.js

```
let cacheData={};
exports.cache={
    get(key) {
        return cacheData[key];
    },
    set(key,val) {
        cacheData[key]=val;
    }
}
```

test/app/extend/cache.test.js

```
const { app, mock, assert } = require('egg-mock/bootstrap');
describe('test/app/extend/cache.test.js', () => {
    it('cache',async () => {
        app.cache.set('name','zhufeng');
        assert(app.cache.get('name') == 'zhufeng');
  });
});
```

- Context 测试只比 Application 多了一个 `app.mockContext()` 步骤来模拟创建一个 Context 对象。

app\extend\context.js

```
exports.language=function () {
    return this.get('accept-language');
}
```

test/app/extend/context.test.js

```
const { app, mock, assert } = require('egg-mock/bootstrap');
describe('test/app/extend/context.test.js',() => {
    let language="zh-cn";
    it('test language',async () => {
        const ctx=app.mockContext({headers: {'Accept-Language':language}});

        assert(ctx.language() == language);
  });
});
```

通过 ctx.request 来访问 Request 扩展的属性和方法，直接即可进行测试。 app\extend\request.js

```
module.exports={
    get isChrome() {
        const userAgent=this.get('User-Agent').toLowerCase();
        return userAgent.includes('chrome');
    }
}
```

test\extend\request.test.js

```
const { app, mock, assert } = require('egg-mock/bootstrap');
describe('test/app/extend/request.test.js',() => {
    it('cache',async () => {
        const ctx=app.mockContext({
            headers: {
                'User-Agent':'I love Chrome'
            }
        });
        assert(ctx.request.isChrome);
  });
});
```

Response 测试与 Request 完全一致。 通过 ctx.response 来访问 Response 扩展的属性和方法，直接即可进行测试。 app\extend\response.js

```
module.exports = {
  get isSuccess() {
    return this.status === 200;
  },
};
```

test\extend\response.test.js

```
describe('isSuccess()', () => {
  it('should true', () => {
    const ctx = app.mockContext();
    ctx.status = 200;
    assert(ctx.response.isSuccess === true);
  });

  it('should false', () => {
    const ctx = app.mockContext();
    ctx.status = 404;
    assert(ctx.response.isSuccess === false);
  });
});
```

- Helper 测试方式与 Service 类似，也是通过 ctx 来访问到 Helper，然后调用 Helper 方法测试

app\extend\helper.js

```
module.exports = {
  money(val) {
    const lang = this.ctx.get('accept-language');
    if (lang.includes('zh-cn')) {
      return `￥ ${val}`;
    }
    return `$ ${val}`;
  },
};
```

test\extend\helper.test.js

```
describe('money()', () => {
  it('should RMB', () => {
    const ctx = app.mockContext({

      headers: {
        'Accept-Language': 'zh-cn',
      },
    });
    assert(ctx.helper.money(100) === '￥ 100');
  });

  it('should US Dolar', () => {
    const ctx = app.mockContext();
    assert(ctx.helper.money(100) === '$ 100');
  });
});
```

- 我们可以通过 app.runSchedule(schedulePath) 来运行一个定时任务
- app.runSchedule 接受一个定时任务文件路径 （app/schedule 目录下的相对路径或者完整的绝对路径），执行对应的定时任务，返回一个 Promise

update_cache.test.js

```
const mock = require('egg-mock');
const assert = require('assert');

it('should schedule work fine', async () => {
  const app = mock.app();
  await app.ready();
  await app.runSchedule('update_cache');
  assert(app.cache);
});
```

16. 布署

- 框架内置了 egg-cluster 来启动 Master 进程，Master 有足够的稳定性，不再需要使用 pm2 等进程守护模块。
- 框架也提供了 egg-scripts 来支持线上环境的运行和停止

```
$ npm i egg-scripts --save
```

添加 npm scripts 到 package.json：

```
{
  "scripts": {
    "start": "egg-scripts start --daemon",
    "stop": "egg-scripts stop"
  }
}
```

17.使用 VSCode 进行调试

- 使用-egg-bin-调试 (https://eggjs.org/zh-cn/core/development.html#%E4%BD%BF%E7%94%A8-egg-bin-%E8%B0%83%E8%AF%95)
- 方式一：开启 VSCode 配置 Debug: Toggle Auto Attach，然后在 Terminal 执行 npm run debug 即可。

- 方式二：配置 VSCode 的 .vscode/launch.json，然后 F5 一键启动即可。（注意，需要关闭方式一中的配置）

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Launch Egg",
      "type": "node",
      "request": "launch",
      "cwd": "${workspaceRoot}",
      "runtimeExecutable": "npm",
      "windows": { "runtimeExecutable": "npm.cmd" },
      "runtimeArgs": [ "run", "debug" ],
      "console": "integratedTerminal",
      "protocol": "auto",
      "restart": true,
      "port": 9229,
      "autoAttachChildProcesses": true
    }
  ]
}
```

18.参考