egg-实战

> https://eggjs.org/zh-cn/

课程目标

- 基于插件的Swagger-doc接口定义
- 统一异常处理
- 基于扩展的helper响应统一处理
- Validate接口格式检查
- 三层结构
- jwt统一鉴权
- 文件上传

搭建egg平台

1. 创建项目

```
# 创建项目
npm i egg-init -g
egg-init egg-server --type=simple
cd egg-server
npm i

# 启动项目
npm run dev
open localhost:7001
```

2.添加swagger-doc

- 添加Controller方法

```
// app/controller/user.js
const Controller = require('egg').Controller
/**
 * @Controller 用户管理
 */
class UserController extends Controller {
  constructor(ctx) {
    super(ctx)
  }
```

```
  /**
   * @summary 创建用户
   * @description 创建用户，记录用户账户/密码/类型
   * @router post /api/user
   * @request body createUserRequest *body
   * @response 200 baseResponse 创建成功
   */
  async create() {
    const { ctx } = this
    ctx.body = 'user ctrl'
  }
}

module.exports = UserController
```

- contract

```
// app/contract/index.js
module.exports = {
  baseRequest:{
    id: { type: 'string', description: 'id 唯一键' ,required:true,example:'1'},

  },
  baseResponse: {
    code: { type: 'integer', required: true, example: 0 },
    data:{type: 'string',example: '请求成功' },
    errorMessage: { type: 'string', example: '请求成功' },
  },
};
```

```
// /app/contract/user.js
module.exports = {
    createUserRequest: {
        mobile: { type: 'string', required: true, description: '手机号'
,example: '18801731528', format: /^1[34578]\d{9}$/,},
        password: { type: 'string', required: true, description: '密码'
,example: '111111', },
        realName: { type: 'string', required: true, description: '姓名'
,example:'Tom'},
    },
}
```

- 添加SwaggerDoc功能

> Swagger 是一个规范和完整的框架，用于生成、描述、调用和可视化 RESTful 风格的 Web 服务。

```
npm install egg-swagger-doc-feat -s
```

```
// config/plugin
  swaggerdoc : {
    enable: true,
    package: 'egg-swagger-doc-feat',
  }
```

```
// config.default.js
  config.swaggerdoc = {
      dirScanner: './app/controller',
      apiInfo: {
        title: '开课吧接口',
        description: '开课吧接口 swagger-ui for egg',
        version: '1.0.0',
      },
      schemes: ['http', 'https'],
      consumes: ['application/json'],
      produces: ['application/json'],
      enableSecurity: false,
      // enableValidate: true,
      routerMap: true,
      enable: true,
    }
```

http://localhost:7001/swagger-ui.html

http://localhost:7001/swagger-doc

- 增加异常处理中间件
  - 异常统一处理
  - 开发环境返回详细异常信息
  - 生产环境不返回详细信息

```
// /middleware/error_handler.js
'use strict'
module.exports = (option, app) => {
  return async function (ctx, next) {
    try {
      await next()
    } catch (err) {
```

```
      // 所有的异常都在 app 上触发一个 error 事件，框架会记录一条错误日志
      app.emit('error', err, this)
      const status = err.status || 500
      // 生产环境时 500 错误的详细错误内容不返回给客户端，因为可能包含敏感信息
      const error = status === 500 && app.config.env === 'prod' ?
        'Internal Server Error' :
        err.message
      // 从 error 对象上读出各个属性，设置到响应中
      ctx.body = {
        code: status, // 服务端自身的处理逻辑错误(包含框架错误500 及 自定义业务逻辑错误533开始 ) 客户端请求参数导致的错误(4xx开始)，设置不同的状态码
        error: error
      }
      if (status === 422) {
        ctx.body.detail = err.errors
      }
      ctx.status = 200
    }
  }
}
```

```
// config.default.js
config.middleware = ['errorHandler']
```

通过注册onerror配置

```
config.onerror = {
    all(err, ctx) {
      // 所有的异常都在 app 上触发一个 error 事件，框架会记录一条错误日志
      ctx.app.emit('error', err, this)
      const status = err.status || 500
      // 生产环境时 500 错误的详细错误内容不返回给客户端，因为可能包含敏感信息
      const error = status === 500 && ctx.app.config.env === 'prod' ?
        'Internal Server Error' :
        err.message
      // 从 error 对象上读出各个属性，设置到响应中
      ctx.body = {
        code: status, // 服务端自身的处理逻辑错误(包含框架错误500 及 自定义业务逻辑错误533开始 ) 客户端请求参数导致的错误(4xx开始)，设置不同的状态码
        error: error
      }
      if (status === 422) {
        ctx.body.detail = err.errors
      }
      ctx.status = 200
    }
  }
```

- helper方法实现统一响应格式

  Helper 函数用来提供一些实用的 utility 函数。

  它的作用在于我们可以将一些常用的动作抽离在 helper.js 里面成为一个独立的函数，这样可以用 JavaScript 来写复杂的逻辑，避免逻辑分散各处。另外还有一个好处是 Helper 这样一个简单的函数，可以让我们更容易编写测试用例。

  框架内置了一些常用的 Helper 函数。我们也可以编写自定义的 Helper 函数。

```
// controller/user.js
const res = {abc:123}

// 设置响应内容和响应状态码
ctx.helper.success({ctx, res})
```

```
// extend/helper.js
const moment = require('moment')

// 格式化时间
exports.formatTime = time => moment(time).format('YYYY-MM-DD HH:mm:ss')

// 处理成功响应
exports.success = ({ ctx, res = null, msg = '请求成功' })=> {
  ctx.body = {
    code: 0,
    data: res,
    msg
  }
  ctx.status = 200
}
```

- Validate检查

```
npm i egg-validate -s
```

```
// config/plugin.js
validate: {
    enable: true,
    package: 'egg-validate',
  },
```

```
async create() {
    const { ctx, service } = this
    // 校验参数
    ctx.validate(ctx.rule.createUserRequest)
}
```

- 添加Model层

```
npm install egg-mongoose -s
```

```
// plugin.js

mongoose : {
  enable: true,
  package: 'egg-mongoose',
},
```

```
// config.default.js
config.mongoose = {
    url: 'mongodb://127.0.0.1:27017/egg_x',
    options: {
      // useMongoClient: true,
      autoReconnect: true,
      reconnectTries: Number.MAX_VALUE,
      bufferMaxEntries: 0,
    },
  }
```

```
// model/user.js
module.exports = app => {
  const mongoose = app.mongoose
  const UserSchema = new mongoose.Schema({
    mobile: { type: String, unique: true, required: true },
```

```
    password: { type: String, required: true },
    realName: { type: String, required: true },
    avatar: { type: String, default:
'https://1.gravatar.com/avatar/a3e54af3cb6e157e496ae430aed4f4a3?s=96&d=mm'},
    extra: { type: mongoose.Schema.Types.Mixed },
    createdAt: { type: Date, default: Date.now }
  })
  return mongoose.model('User', UserSchema)
}
```

添加Service层

```
npm install egg-bcrypt -s
```

```
bcrypt : {
    enable: true,
    package: 'egg-bcrypt'
  }
```

```
// service/user.js
const Service = require('egg').Service

class UserService extends Service {

  /**
   * 创建用户
   * @param {*} payload
   */
  async create(payload) {
    const { ctx } = this
    payload.password = await this.ctx.genHash(payload.password)
    return ctx.model.User.create(payload)
  }
}


module.exports = UserService
```

- Controller调用

```
/**
   * @summary 创建用户
   * @description 创建用户，记录用户账户/密码/类型
   * @router post /api/user
   * @request body createUserRequest *body
   * @response 200 baseResponse 创建成功
   */
  async create() {
    const { ctx, service } = this
    // 校验参数
    ctx.validate(ctx.rule.createUserRequest)
    // 组装参数
  const payload = ctx.request.body || {}
    // 调用 Service 进行业务处理
    const res = await service.user.create(payload)
    // 设置响应内容和响应状态码
    ctx.helper.success({ctx, res})
  }
```

// 完整内容 粘贴模板

- 补足servcie层
- 补足Controller层

通过生命周期初始化数据

> https://eggjs.org/en/basics/app-start.html#mobileAside

```
// /app.js
/**
 *  全局定义
 * @param app
 */

class AppBootHook {
    constructor(app) {
        this.app = app;
        app.root_path = __dirname;
    }

    configWillLoad() {
        // Ready to call configDidLoad,
        // Config, plugin files are referred,
        // this is the last chance to modify the config.
    }

    configDidLoad() {
```

```
        // Config, plugin files have been loaded.
    }

    async didLoad() {
        // All files have loaded, start plugin here.
    }

    async willReady() {
        // All plugins have started, can do some thing before app ready
    }

    async didReady() {
        // Worker is ready, can do some things
        // don't need to block the app boot.
        console.log('========Init Data=========')
        const ctx = await this.app.createAnonymousContext();
        await ctx.model.User.remove();

        await ctx.service.user.create({
            mobile: '13611388415',
            password: '111111',
            realName: '老夏',
        })
    }

    async serverDidReady() {

    }

    async beforeClose() {
        // Do some thing before app close.
    }
}

module.exports = AppBootHook;
```

用户鉴权模块

注册jwt模块

```
npm i egg-jwt -s
```

```
// plugin.js
jwt: {
  enable: true,
  package: 'egg-jwt',
}
```

```
 // config.default.js
  config.jwt = {
    secret: 'Great4-M',
    enable: true, // default is false
    match: /^\/api/, // optional
  }
```

- Service层

```
// service/actionToken.js
'use strict'

const Service = require('egg').Service

class ActionTokenService extends Service {
  async apply(_id) {
    const {ctx} = this
    return ctx.app.jwt.sign({
      data: {
        _id: _id
      },
      exp: Math.floor(Date.now() / 1000) + (60 * 60 * 24 * 7)
    }, ctx.app.config.jwt.secret)
  }
}

module.exports = ActionTokenService
```

```
// service/userAccess.js
'use strict'
const Service = require('egg').Service
class UserAccessService extends Service {

  async login(payload) {
    const { ctx, service } = this
    const user = await service.user.findByMobile(payload.mobile)
    if(!user){
```

```
      ctx.throw(404, 'user not found')
    }
    let verifyPsw = await ctx.compare(payload.password, user.password)
    if(!verifyPsw) {
      ctx.throw(404, 'user password is error')
    }
    // 生成Token令牌
    return { token: await service.actionToken.apply(user._id) }
  }

  async logout() {
  }

  async current() {
    const { ctx, service } = this
    // ctx.state.user 可以提取到JWT编码的data
    const _id = ctx.state.user.data._id
    const user = await service.user.find(_id)
    if (!user) {
      ctx.throw(404, 'user is not found')
    }
    user.password = 'How old are you?'
    return user
  }
}

module.exports = UserAccessService
```

Contract层

```
// app/contract/userAccess.js
module.exports = {
  loginRequest: {
    mobile: { type: 'string', required: true, description: '手机号', example:
'18801731528', format: /^1[34578]\d{9}$/, },
    password: { type: 'string', required: true, description: '密码', example:
'111111', },
  },
}
```

Controller层

```
// controller/userAccess.js
'use strict'
const Controller = require('egg').Controller
/**
 * @Controller 用户鉴权
```

```
 */
class UserAccessController extends Controller {

  constructor(ctx) {
    super(ctx)
  }

  /**
   * @summary 用户登入
   * @description 用户登入
   * @router post /auth/jwt/login
   * @request body loginRequest *body
   * @response 200 baseResponse 创建成功
   */
  async login() {
    const { ctx, service } = this
    // 校验参数
    ctx.validate(ctx.rule.loginRequest);
    // 组装参数
    const payload = ctx.request.body || {}
    // 调用 Service 进行业务处理
    const res = await service.userAccess.login(payload)
    // 设置响应内容和响应状态码
    ctx.helper.success({ ctx, res })
  }

  /**
   * @summary 用户登出
   * @description 用户登出
   * @router post /auth/jwt/logout
   * @request body loginRequest *body
   * @response 200 baseResponse 创建成功
   */
  async logout() {
    const { ctx, service } = this
    // 调用 Service 进行业务处理
    await service.userAccess.logout()
    // 设置响应内容和响应状态码
    ctx.helper.success({ ctx })
  }
}

module.exports = UserAccessController
```

// 粘贴测试页面

文件上传

```
npm i await-stream-ready stream-wormhole image-downloader -s
```

- controller

```
// app/controller/upload.js
const fs = require('fs')
const path = require('path')
const Controller = require('egg').Controller
const awaitWriteStream = require('await-stream-ready').write
const sendToWormhole = require('stream-wormhole')

/**
 * @Controller 上传
 */
class UploadController extends Controller {
    constructor(ctx) {
        super(ctx)
    }

    // 上传单个文件
    /**
     * @summary 上传单个文件
     * @description 上传单个文件
     * @router post /api/upload/single
     */
    async create() {
        const { ctx } = this
        // 要通过 ctx.getFileStream 便捷的获取到用户上传的文件，需要满足两个条件：
        // 只支持上传一个文件。
        // 上传文件必须在所有其他的 fields 后面，否则在拿到文件流时可能还获取不到
fields。
        const stream = await ctx.getFileStream()
        // 所有表单字段都能通过 `stream.fields` 获取到
        const filename = path.basename(stream.filename) // 文件名称
        const extname = path.extname(stream.filename).toLowerCase() // 文件扩展
名称
        const uuid = (Math.random() * 999999).toFixed()

        // 组装参数 stream
        const target = path.join(this.config.baseDir, 'app/public/uploads',
`${uuid}${extname}`)
        const writeStream = fs.createWriteStream(target)
        // 文件处理，上传到云存储等等
        try {
            await awaitWriteStream(stream.pipe(writeStream))
        } catch (err) {
            // 必须将上传的文件流消费掉，要不然浏览器响应会卡死
            await sendToWormhole(stream)
```

```
            throw err
        }
        // 调用 Service 进行业务处理
        // 设置响应内容和响应状态码
        ctx.helper.success({ ctx })
    }
}


module.exports = UploadController
```