

基于Koa定制自己的企业级三层框架

基于Koa定制自己的企业级三层框架

回顾

课堂主题

课堂目标

知识点

Egg.js体验

实现分层架构

扩展点

总结

下节课预告

回顾

koa开发中常见任务

课堂主题

1. koa不足的地方
2. 封装路由
3. 实现控制器层
4. 实现服务层
5. 实现模型层
6. 加载中间件

课堂目标

1. 理解三层结构
2. 熟悉企业级web开发框架egg.js使用
3. 基于Koa定制自己的企业级三层结构框架

知识点

Egg.js体验

- 三层结构
 - 信息资源层 就是 action, 或者 servlet, 用来处理上下游数据结构。
 - 业务逻辑层一般应用中会有一层service抽象, 实现核心业务逻辑, 事务控制也在这一层实现。
 - 数据访问层也即dao层, 重点负责数据库访问, 完成持久化功能。
- 创建项目

```
// 创建项目
$ npm i egg-init -g
$ egg-init egg --type=simple
$ cd egg-example
$ npm i

// 启动项目
$ npm run dev
$ open localhost:7001
```

- 浏览项目结构:
 - Public
 - Router -> Controller -> Service -> Model
 - Schedule
- 创建一个路由, router.js

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

- 创建一个控制器, user.js

```
'use strict';

const Controller = require('egg').Controller;

class UserController extends Controller {
  async index() {
    this.ctx.body = [
      {name: 'tom'},
      {name: 'jerry'}
    ]
  }
}

module.exports = UserController;
```

约定优于配置（convention over configuration），也称作按约定编程，是一种软件设计范式，旨在减少软件开发人员需做决定的数量，获得简单的好处，而又不失灵活性。

- 创建一个服务，./app/service/user.js

```
'use strict';

const Service = require('egg').Service;

class UserService extends Service {
  async getAll() {
    return [
      {name: 'tom'},
      {name: 'jerry'}
    ]
  }
}

module.exports = UserService;
```

- 使用服务，./app/controller/user.js

```
async index() {
  const { ctx } = this;
  ctx.body = await ctx.service.user.getAll();
}
```

- 创建模型层：以mysql + sequelize为例演示数据持久化

- 安装：npm install --save egg-sequelize mysql2
- 在 config/plugin.js 中引入 egg-sequelize 插件

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

- 在 config/config.default.js 中编写 sequelize 配置

```
// const userConfig 中
sequelize: {
  dialect: "mysql",
  host: "127.0.0.1",
  port: 3306,
  username: "root",
  password: "example",
  database: "kaikeba"
}
```

- 编写User模型, ./app/model/user.js

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

  const User = app.model.define(
    "user",
    { name: STRING(30) },
    { timestamps: false }
  );

  // 数据库同步
  User.sync({force: true})

  return User;
};
```

- 服务中或者控制器中调用: ctx.model.User或app.model.User

```
class UserService extends Service {
  async getAll() {
    return await this.ctx.model.User.findAll()
  }
}

// 或者控制器
ctx.body = await this.ctx.model.User.findAll()
```

// 需要同步数据库

```
// 添加测试数据
const User = this.ctx.model.User
  await User.sync({ force: true })
  await User.create({
    name: "laowang"
  })
)
```

实现分层架构

- 目标是创建约定大于配置、开发效率高、可维护性强的项目架构
- 路由处理
 - 规范
 - 所有路由，都要放在routes文件夹中
 - 若导出路由对象，使用 动词+空格+路径 作为key，值是操作方法
 - 若导出函数，则函数返回第二条约定格式的对象
 - 路由定义：
 - 新建routes/index.js，默认Index.js没有前缀

```
module.exports = {
  'get /': async ctx=>{
    ctx.body = '首页'
  },
  'get /detail' : ctx=>{
    ctx.body = '详情页面'
  }
}
```

- 新建routes/user.js 路由前缀是/user

```

module.exports = {
  "get /": async ctx => {
    ctx.body = "用户首页";
  },
  "get /info": ctx => {
    ctx.body = "用户详情页面";
  }
};

```

- 路由加载器，新建kkb-loader.js

```

const fs = require("fs");
const path = require("path");
const Router = require("koa-router");

// 读取指定目录下文件
function load(dir, cb) {
  // 获取绝对路径
  const url = path.resolve(__dirname, dir);
  // 读取路径下的文件
  const files = fs.readdirSync(url);
  // 遍历路由文件，将路由配置解析到路由器中
  files.forEach(filename => {
    // 去掉后缀名
    filename = filename.replace(".js", "");
    // 导入文件
    const file = require(url + "/" + filename);
    // 处理逻辑
    cb(filename, file);
  });
}

function initRouter() {
  const router = new Router();
  load("routes", (filename, routes) => {
    // 若是index无前缀，别的文件前缀就是文件名
    const prefix = filename === "index" ? "" : `/${filename}`;

    // 遍历路由并添加到路由器
    Object.keys(routes).forEach(key => {
      const [method, path] = key.split(" ");
      console.log(
        `正在映射地址: ${method.toLocaleUpperCase()} ${prefix}${path}`
      );
      // 执行router.method(path, handler)注册路由
      router[method](prefix + path, routes[key]);
    });
  });
}

```

```

    });
  });
  return router;
}

module.exports = { initRouter };

```

- 测试，引入kkb-loader.js

```

// index.js
const app = new (require('koa'))()
const {initRouter} = require('./kkb-loader')
app.use(initRouter().routes())
app.listen(3000)

```

- 封装，创建kkb.js

```

// kkb.js
const koa = require("koa");
const {initRouter} = require("./kkb-loader");

class kkb {
  constructor(conf) {
    this.$app = new koa(conf);
    this.$router = initRouter();
    this.$app.use(this.$router.routes());
  }

  start(port) {
    this.$app.listen(port, () => {
      console.log("服务器启动成功，端口" + port);
    });
  }
}

module.exports = kkb;

```

- 修改app.js

```

const kkb = require("./kkb");
const app = new kkb();
app.start(3000);

```

- 控制器：抽取route中业务逻辑至controller

- 约定：controller文件夹下面存放业务逻辑代码，框架自动加载并集中暴露
- 新建controller/home.js

```
module.exports = {  
  index: async ctx => {  
    ctx.body = "首页";  
  },  
  detail: ctx => {  
    ctx.body = "详情页面";  
  }  
}
```

- 修改路由声明，routes/index.js

```
// 需要传递kkb实例并访问其$ctrl中暴露的控制器  
module.exports = app => ({  
  "get /": app.$ctrl.home.index,  
  "get /detail": app.$ctrl.home.detail  
});
```

- 加载控制器，更新kkb-loader.js

```
function initController() {  
  const controllers = {};  
  // 读取控制器目录  
  load("controller", (filename, controller) => {  
    // 添加路由  
    controllers[filename] = controller;  
  });  
  
  return controllers;  
}  
  
module.exports = { initController };
```

- 初始化控制器，kkb.js


```
const {initController} = require("../kkb-loader");

class kkb {
  constructor(conf) {
    //...
    this.$ctrl = initController(); // 先初始化控制器，路由对它有依赖
    this.$router = initRouter(this); // 将kkb实例传进去
    //...
  }
}
```

- 修改路由初始化逻辑，能够处理函数形式的声明, kkb-loader.js

```
function initRouter(app) { // 添加一个参数
  load("routes", (filename, routes) => {
    // ...

    // 判断路由类型，若为函数需传递app进去
    routes = typeof routes == "function" ? routes(app) : routes;

    // ...
  });
}
```

- 服务：抽离通用逻辑至service文件夹，利于复用
 - 新建service/user.js

```
const delay = (data, tick) => new Promise(resolve => {
  setTimeout(() => {
    resolve(data)
  }, tick)
})

// 可复用的服务 一个同步，一个异步
module.exports = {
  getName() {
    return delay('jerry', 1000)
  },
  getAge() {
    return 20
  }
};
```

- 加载service

```
//kkb-loader.js
function initService() {
  const services = {};
  // 读取控制器目录
  load("service", (filename, service) => {
    // 添加路由
    services[filename] = service;
  });

  return services;
}
module.exports = { initService };

// kkb.js
this.$service = initService();
```

- 挂载和使用service

```
// kkb-loader.js
function initRouter(app) {
  // ...
  // router[method](prefix + path, routes[key])
  router[method](prefix + path, async ctx => { // 传入ctx
    app.ctx = ctx; // 挂载至app
    await routes[key](app); // 路由处理器现在接收到的是app
  });
  //...
}
```

- 更新路由

```
// routes/user.js
module.exports = {
  "get /": async (app) => {
    const name = await app.$service.user.getName();
    app.ctx.body = "用户:" + name;
  },
  "get /info": app => {
    app.ctx.body = "用户年龄: " + app.$service.user.getAge();
  }
};

// routes/index.js
module.exports = app => ({
  "get /": app.$ctrl.home.index,
```

```

    "get /detail": app.$ctrl.home.detail
  });

  // controller/home.js
  module.exports = (app) => ({
    index: async ctx => {
      // ctx.body = 'Ctrl Index'
      console.log('index ctrl')
      const name = await app.$service.user.getName()
      app.ctx.body = 'ctrl user' + name

    },
    detail: async ctx => {
      ctx.body = 'Ctrl Detail'
    }
  })

  // controller科里化
  // initController
  controllers[filename] = controller(app)
  this.$ctrl = initController(this)

```

- 数据库集成

- 集成sequelize: `npm install sequelize mysql2 --save`
- 约定:
 - config/config.js中存放项目配置项
 - key表示对应配置目标
 - model中存放数据模型
- 配置sequelize连接配置项, index.js

```

// config/index.js
module.exports = {
  db:{
    dialect:'mysql',
    host:'localhost',
    database:'kaikeba',
    username:'root',
    password:'example'
  }
}

```

- 新增loadConfig, kkb-loader.js

```

const Sequelize = require("sequelize");
function loadConfig(app) {
  load("config", (filename, conf) => {
    if (conf.db) {
      app.$db = new Sequelize(conf.db);
    }
  });
}
module.exports = { loadConfig };

// kkb.js
//先加载配置项
loadConfig(this);

```

- 新建数据库模型, model/user.js

```

const { STRING } = require("sequelize");
module.exports = {
  schema: {
    name: STRING(30)
  },
  options: {
    timestamps: false
  }
};

```

- loadModel和loadConfig初始化, kkb-loader.js

```

function loadConfig(app) {
  load("config", (filename, conf) => {
    if (conf.db) {
      app.$db = new Sequelize(conf.db);

      // 加载模型
      app.$model = {};
      load("model", (filename, { schema, options }) => {
        app.$model[filename] = app.$db.define(filename, schema,
options);
      });
      app.$db.sync();
    }
  });
}

```

- 在controller中使用\$db

```

module.exports = {
  // index: async ctx => {
  //   ctx.body = '首页'
  // },
  index: async app => { // app已传递
    app.ctx.body = await app.$model.user.findAll()
  },
  detail: app => {
    app.ctx.body = '详细页面'
  }
}

```

- 在service中使用\$db

```

// 修改service结构, service/user.js
module.exports = app => ({
  getName() {
    // return delay('jerry',1000)
    return app.$model.user.findAll() // 添加
  },
  getAge(){
    return 20
  }
})

// 修改kkb-loader.js
function initService(app) { // 增加参数
  const services = {}
  load('service', (filename, service) => {
    services[filename] = service(app) // 服务变参数
  })
  console.log('service', services)
  return services
}

// 修改kkb.js
this.$service = initService(this)

```

- 中间件
 - 规定koa中间件放入middleware文件夹
 - 编写一个请求记录中间件, ./middleware/logger.js

```

module.exports = async (ctx, next) => {
  console.log(ctx.method + " " + ctx.path);
  const start = new Date();
  await next();
  const duration = new Date() - start;
  console.log(
    ctx.method + " " + ctx.path + " " + ctx.status + " " + duration +
    "ms"
  );
};

```

- 配置中间件, ./config/config.js

```

module.exports = {
  db:{...},
  middleware: ['logger'] // 以数组形式, 保证执行顺序
}

```

- 加载中间件, kkb-loader.js

```

function loadConfig(app) {
  load("config", (filename, conf) => {
    // 如果有middleware选项, 则按其规定循序应用中间件
    if (conf.middleware) {
      conf.middleware.forEach(mid => {
        const midPath = path.resolve(__dirname, "middleware", mid);
        app.$app.use(require(midPath));
      });
    }
  });
}

```

- 调用, kkb.js

```

class kkb {
  constructor(conf) {
    this.$app = new koa(conf);
    //先加载配置项
    loadConfig(this);
    //...
  }
}

```

- 定时任务

- 使用Node-schedule来管理定时任务

```
npm install node-schedule --save
```

- 约定：schedule目录，存放定时任务，使用crontab格式来启动定时

```
//log.js
module.exports = {
  interval: '* / 3 * * * * *',
  handler(){
    console.log('定时任务 嘿嘿 三秒执行一次'+ new Date())
  }
}

// user.js
module.exports = {
  interval: '30 * * * * *',
  handler(){
    console.log('定时任务 嘿嘿 每分钟第30秒执行一次'+ new Date())
  }
}
```

定时格式是符合linux的crontab

*	*	*	*	*	*
T	T	T	T	T	T
					└ day of week (0 - 7) (0 or 7 is Sun)
				└	month (1 - 12)
			└	day of month (1 - 31)	
		└	hour (0 - 23)		
	└	minute (0 - 59)			
└	second (0 - 59, optional)				

6个占位符从左到右分别代表：秒、分、时、日、月、周几，"表示通配符，匹配任意，当秒是"时，表示任意秒数都触发，其它类推

```
每分钟的30秒触发： '30 * * * * *'
每小时的1分30秒触发： '30 1 * * * *'
每天的凌晨1点1分30秒触发： '30 1 1 * * *'
每月的1日1点1分30秒触发： '30 1 1 1 * *'
2020年的1月1日1点1分30秒触发： '30 1 1 1 2020 *'
每周1的1点1分30秒触发： '30 1 1 * * 1'
每三秒： '* / 3 * * * * *'
```

- 新增loadSchedule函数，kkb-loader.js

```
const schedule = require("node-schedule");
function initSchedule() {
```

开课吧web全栈架构师

```
// 读取控制器目录
load("schedule", (filename, scheduleConfig) => {
  schedule.scheduleJob(scheduleConfig.interval,
    scheduleConfig.handler);
});
}
module.exports = { initRouter, initController,
  initService,initSchedule };

// kkb.js
const {initSchedule} = require("../kkb-loader");
class kkb {
  constructor(conf) {
    initSchedule();
  }
}
```

```
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:19:57 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:00 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:03 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:06 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:09 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:12 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:15 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:18 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:21 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:24 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:27 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:30 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 每分钟第30秒执行一次Fri Mar 15 2019 16:20:30 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:33 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:36 GMT+0800 (GMT+08:00)
定时任务 嘿嘿 三秒执行一次Fri Mar 15 2019 16:20:39 GMT+0800 (GMT+08:00)
```

通过约定文件夹的形式，开始MVC开发之旅，这个框架学习了eggjs的核心架构思想，到现在你已经构建了自己的MVC框架了，刷个火箭吧！

扩展点

1. 扩展

1. 安全
2. 日志
3. 线上部署
4. 单元测试等等

总结

1. 知识点
2. 学习方法
3. 收获

下节课预告

1. Koa实现原理
2. 常见Koa中间件实现

作业：自动加载模块 作业截图

