

一、这是个什么样的项目？

二、项目中包含什么功能？

三、项目准备工作

四、项目开始

1、项目启动

2、项目依赖包

3、数据库连接

4、模型(model)创建

- 管理员模型

- 分类模型

- 文章模型

- 评论模型

- 回复模型

- 广告模型

5、全局的错误处理

错误处理中间件

6、Api接口编写

管理员接口

-管理员注册

-管理员登录

-管理员权限

分类接口

创建分类

获取所有分类

更新分类

删除分类

文章接口

创建文章

获取文章列表

更新文章

获取文章详情

删除文章

评论接口

添加评论

获取评论列表
获取评论详情
更新评论
删除评论
回复接口
创建回复
回复列表
回复详情
更新回复
删除回复
广告接口
创建广告
获取广告列表
获取广告详情
更新广告
删除广告

一、这是个什么样的项目？

基于Vue+Node.js koa2+mongoose实战开发的一套完整的博客项目网站，使用koa2二次开发一套适合多端的Restful API,同时配合完整的后台管理系统，且前端展示既有基于hbs服务器渲染，也有基于vue.js前后端分离的2套前端网站

全局安装脚手架

```
1 | npm install koa-generator -g
```

二、项目中包含什么功能？

2.1 Nodejs koa2 服务端Restful API

- 管理员与权限控制接口
- 文章管理接口
- 评论/回复功能接口
- 分类接口
- 广告接口

2.2 博客前端展示网站

- hbs服务器渲染
- vue.js前后端分离

2.3 后台管理系统

- 使用 vue.js + element-ui 搭建的后台管理系统

2.4 优势

- 使用精小而强大的 Node.js Koa2 框架做服务端 API 接口
- 前端既有服务端渲染，也有前后端分离，且做了大量的优化工作，前端展示网站打开快
- 性能优化方面的工作
 - 服务端使用了缓存机制，减少服务器的请求压力，如Redis 缓存，HTTP缓存
 - 使用了 CDN 加速，静态文件存储在七牛服务器上
 - ejs 服务端渲染尽量减少文件的引入，减少对 DOM的操作，且封装使用了图片懒加载，事件防抖和节流控制浏览器滚动监听事件。
 - Vue.js 项目使用了模块按需加载，使用浏览器缓存机制减少对服务器请求的压力

2.5 知识点

- 服务端：Node.js,koa2,mongodb,mongoose,redis
- 前端服务端渲染：hbs,jQuery,bootstrap
- 前后端分离：Vue.js,element-ui
- 后端管理系统:Vue.js,element-ui

- 性能优化

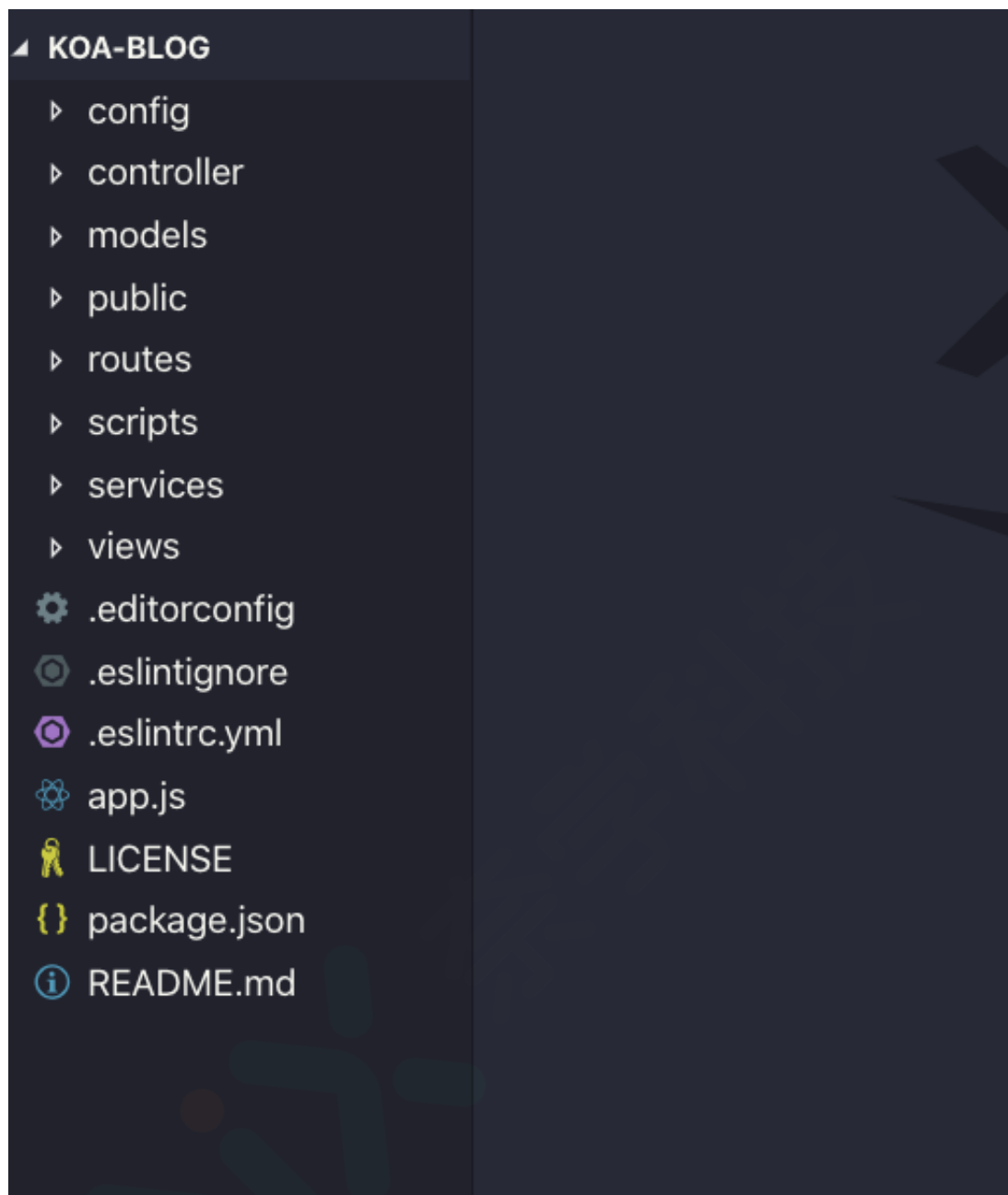
三、项目准备工作

- 安装mongodb
 - [window安装](#)
 - [mac安装](#)
- [安装Redis](#)
- 检查环境
 - 打开CMD或者终端。输入 `node -v` , 查看是否安装了node
- 电脑上安装[postman](#)测试接口工具

四、项目开始

1、项目启动

打开终端，输入 `koa2 --hbs koa-blog`



```
1 npm install 安装项目依赖
2 npm start 启动项目
3
4 访问
5 http://localhost:3000/
```

注意：启动项目时控制台会报错

Error: Engine not found for the ".hbs" file extension

修改app.js

```
1 app.use(bodyParser())
2   ...
3   map: { 'hbs': 'handlebars' },
4   ...
5   })
6   ...
```

再访问 `http://localhost:3000` , 成功显示

← → ↺ 🏠 ⓘ localhost:3000

Koa2

Welcome to Koa2

2、项目依赖包

```
1  "dependencies": {
2    "basic-auth": "^2.0.1",
3    "bcrypt": "^4.0.1",
4    "debug": "~2.6.8",
5    "hbs": "~4.0.1",
6    "jsonwebtoken": "^8.5.1",
7    "koa": "^2.3.0",
8    "koa-bodyparser": "^4.2.0",
9    "koa-bouncer": "^6.0.4",
10   "koa-convert": "^1.2.0",
11   "koa-json": "^2.0.2",
12   "koa-jwt": "^3.6.0",
13   "koa-logger": "^3.0.0",
14   "koa-onerror": "^3.1.0",
15   "koa-router": "^7.2.1",
16   "koa-static": "^3.0.0",
17   "koa-views": "^6.0.2",
18   "moment": "^2.24.0",
19   "mongoose": "^5.9.5",
20   "redis": "^3.0.2"
21 },
```

- `bcrypt` 对数据进行加盐
- `jsonwebtoken` JSON WEB令牌实现
- `koa-jwt` Koa JWT身份验证中间件

- `koa-bouncer` Koa路由的参数验证库
- `basic-auth` Nodejs基本身份验证解析器
- `mongoose` MongoDB DOM框架
- `redis` 高性能Redis客户端。
- `moment` 解析，验证，操作和显示日期时间库

```
1 yarn add bcrypt jsonwebtoken koa-jwt koa-bouncer  
  mongoose redis moment  
2 或者  
3 npm i bcrypt jsonwebtoken koa-jwt koa-bouncer  
  mongoose redis moment -S
```

下载 `bcrypt` 包时，请先全局下载 `node-gyp` 包


```
npm i node-gyp -g
```

下载此包过程有点缓慢，大家请耐心等待

[参考链接](#)

3、数据库连接

本项目中会使用mongodb作为主数据，redis数据库做缓存数据

：mongodb数据库和redis数据库服务端一定要启动

根目录下新建db文件夹


```
1 // db/index.js
2 const mongoose = require('mongoose');
3 const config = require('../config');
4
5 mongoose.connect(`mongodb://localhost/${config.
  db.dbName}`, { useNewUrlParser: true,
  useUnifiedTopology: true });
6 mongoose.set('useFindAndModify', false)
7 const db = mongoose.connection;
8 db.on('error', console.error.bind(console,
  'connection error:'));
9 db.once('open', async ()=>{
10   console.log('链接数据库成功');
11 })
12
```

相关配置文件放入到 `config/index.js`

```
1 module.exports = {  
2   port: 3000, //后台服务器端口号  
3   //mongodb数据库相关配置  
4   db:{  
5     port:27017,  
6     host:'localhost',  
7     dbName:'dbblog' //数据库名称  
8   }  
9 }  
10
```

通过mongoose链接数据

```
1 // app.js  
2 const db = require('./db');
```

控制台打印：链接数据库成功。表示数据库连接成功

4、模型(model)创建

根据在上面的介绍，我们本项目中的接口需要有如下：

- 管理员与权限控制接口
- 文章管理接口
- 评论/回复功能接口
- 分类接口
- 广告接口

根据接口，我们可以反映射模型

在 `models` 文件夹下,分别创建**管理员模型**、**分类模型**、**文章模型** 、**评论模型**、**回复模型**、以及**广告模型**

- 管理员模型

```
1 // models/AdminModel.js
2
3 // 对秘密加盐模块
4 const bcrypt = require('bcrypt');
5 // 配置加盐的位数
6 const SALT_WORK_FACTOR = 10 //配置加盐的位数
7 // 1.引入mongoose
8 const mongoose = require('mongoose')
9
10 // 2.字义Schema(描述文档结构)
11 const adminSchema = new mongoose.Schema({
12   nickname: { type: String, required: true },//角
    色名称
13   password: {
14     type: String,
15     required: true,
16     set:(val)=>{
17       // 加密
18       const salt =
19         bcrypt.genSaltSync(SALT_WORK_FACTOR);
20       // 生成加密密码
```

```

20     const psw = bcrypt.hashSync(val, salt);
21     return psw;
22 }
23 }, // 密码
24 })
25
26 // 3.定义Model(与几何对应,可以操作集合)
27 const AdminModel = mongoose.model('Admin',
    adminSchema);
28
29 // 4.向外暴露model
30 module.exports = AdminModel;

```

上面 `password` 字段中,调用了`set`方法,当我们写入数据时,`bcrypt`模块会将存入的密码进行哈希密码的加密

- 分类模型

```

1 // models/CategoryModel.js
2
3 // 1.引入mongoose
4 const mongoose = require('mongoose')
5
6 // 2.定义Schema(描述文档结构)
7 const categorySchema = new mongoose.Schema({

```

```
8   name: { type: String, require: true },//分类名称
9   keyword: { type: String, required: true }, //分类关键字
10 })
11
12 // 3.定义Model(与几何对应,可以操作集合)
13 const CategoryModel =
14   mongoose.model('Category', categorySchema);
15
16 // 4.向外暴露model
17 module.exports = CategoryModel;
```

分类模型模块比较简单，只需要 `name` 和 `keyword` 字段即可

- 文章模型

```
1 //models/ArticleModel.js
2 // 1.引入mongoose
3 const mongoose = require('mongoose')
4
5 // 2.字义Schema(描述文档结构)
6 const ArticleSchema = new mongoose.Schema({
7   title: { type: String, require: true },//文章标题
```

```
8   author: { type: String, required: true }, //
作者
9   description: { type: String, required: true
}, // 文章简介
10  keyword: { type: String, required: true }, //
文章内容
11  content: { type: String, required: true }, //
文章关键字
12  cover: { type: String, required: true }, // 文
章封面
13  browse:{type:Number,default:0}, //文章浏览数
14
15  // 一个分类下有多篇文章文章 属于1对多关系
16  //分类表: 主表 文章表: 从表
17  //关联分类
18  category_id:{
19    type:mongoose.Schema.Types.ObjectId,
20    //ref属性表示引用 可以直接引用Category模型
21    ref: 'Category'
22  }
23  // 为表添加 创建的时间和更新的时间
24 }, { timestamps: { createdAt: 'created',
updatedAt: 'updated' }})
25
26 // 3.定义Model(与几何对应,可以操作集合)
```

```
27 const ArticleModel = mongoose.model('Article',
    ArticleSchema);
28
29 // 4.向外暴露model
30 module.exports = ArticleModel;
```

- 评论模型

```
1 //models/CommentModel.js
2 // 1.引入mongoose
3 const mongoose = require('mongoose')
4
5 // 2.定义Schema(描述文档结构)
6 const CommentSchema = new mongoose.Schema({
7   nickname: { type: String, require: true },//评论人昵称
8   content:{type:String,require:true}, //评论内容
9   target_id:{type:String,require:true},//评论目标人的id
10   target_type: { type: String, required: true }, // 评论类型,普通文章为:article,专栏文章为:column
11 })
12
13 // 3.定义Model(与几何对应,可以操作集合)
```

```
14 const CommentModel = mongoose.model('Comment',
    CommentSchema);
15
16 // 4.向外暴露model
17 module.exports = CommentModel;
```

- 回复模型

```
1 //models/ReplyModel.js
2 // 1.引入mongoose
3 const mongoose = require('mongoose')
4 const moment = require('moment')
5
6 // 2.定义Schema(描述文档结构)
7 const ReplySchema = new mongoose.Schema({
8   nickname: { type: String, require: true },//评
   论人昵称
9   content: { type: String, require: true }, //评
   论内容
10   createdAt: {
11     type: Date,
12     require: true,
13     default:Date.now,
14     // 处理时间
15     get(val){
16       return moment(val).format('YYYY-MM-DD')
```



```
17     }
18   },
19   comment_id:{
20     type: mongoose.Schema.Types.ObjectId,
21     // 引用
22     ref: 'Comment'
23   }
24 })
25 //只有将Schema设置如下配置, 才能调用get方法
26 ReplySchema.set('toJSON', { getters: true });
27 // 3.定义Model(与几何对应,可以操作集合)
28 const ReplyModel = mongoose.model('Reply',
  ReplySchema);
29
30 // 4.向外暴露model
31 module.exports = ReplyModel;
```

- 广告模型

```
1 //models/AdvertiseModel.js
2 const mongoose = require('mongoose');
3 const moment = require('moment')
4
5 const AdvertiseSchema = new mongoose.Schema({
6   title:{
7     type:String,
```

```
8     require:true, //广告标题
9 },
10 link:{
11     type:String,
12     require:true,
13 },
14 createAt: {
15     type: Date,
16     require: true,
17     default: Date.now,
18     // 处理时间
19     get(val) {
20         return moment(val).format('YYYY-MM-DD
21 HH:mm:ss')
22     },
23 })
24 AdvertiseSchema.set('toJSON', { getters: true
25 });
26 const AdvertiseModel =
27     mongoose.model('Advertise', AdvertiseSchema);
28 // 4.向外暴露model
29 module.exports = AdvertiseModel;
```

以上为相关模型模块的创建，当访问对应路由api接口的时候，模型才得以创建

5、全局的错误处理

新建 `core/http-exception.js`

```
1 class HttpException extends Error {
2   constructor(msg = '服务器异常', errorCode =
  10000, code = 400) {
3     super()
4     this.errorCode = errorCode
5     this.code = code
6     this.msg = msg
7   }
8 }
9 //参数错误
10 class ParameterException extends HttpException
  {
11   constructor(msg, errorCode) {
12     super()
13     this.code = 400
14     this.msg = msg || '参数错误'
15     this.errorCode = errorCode || 10000
16   }
17 }
18 //认证失败
19 class AuthFailed extends HttpException {
20   constructor(msg, errorCode) {
21     super()
```

```
22     this.code = 401
23     this.msg = msg || '授权失败'
24     this.errorCode = errorCode || 10004
25 }
26 }
27 //404
28 class NotFound extends HttpException {
29     constructor(msg, errorCode) {
30         super()
31         this.code = 404
32         this.msg = msg || '404找不到'
33         this.errorCode = errorCode || 10005
34     }
35 }
36 //禁止访问
37 class Forbidden extends HttpException {
38     constructor(msg, errorCode) {
39         super()
40         this.code = 403
41         this.msg = msg || '禁止访问'
42         this.errorCode = errorCode || 10006
43     }
44 }
45 //xxx已存在
46 class Existing extends HttpException {
47     constructor(msg, errorCode) {
```

```
48     super()
49     this.code = 412
50     this.msg = msg || '已存在'
51     this.errorCode = errorCode || 10006
52 }
53 }
54
55 module.exports = {
56   HttpException,
57   ParameterException,
58   AuthFailed,
59   NotFound,
60   Forbidden,
61   Existing
62 }
63
```

app.js 中使用

```
1 | const errors = require('./core/http-exception')
2 | global.errs = errors
```

这样在检查出对应的参数错误、404错误、权限错误、xxx已存在的错误可以这样使用

```
1 //例如
2 throw new global.errs.Existing('管理员已存在');
```

当出现任何的错误时，我们需要错误处理中间件来处理

错误处理中间件

新建 `middlewares/exception.js`

```
1 const { HttpException } = require('../core/http-exception');
2 const bouncer = require('koa-bouncer');
3 const catchError = async (ctx, next) => {
4   try {
5     await next();
6   } catch (error) {
7     //判断校验类型错误
8     if (error instanceof
bouncer.ValidationError) {
9       ctx.body = {
10         name: error.name,
11         message: error.message,
12         request: `${ctx.method}
${ctx.path}`
13       };
14       return;
```

```
15     }
16     //401权限错误处理
17     if (error.status === 401) {
18         ctx.status = 401;
19         ctx.body = {
20             error_code:error.status,
21             msg: error.originalError ?
error.originalError.message : error.message,
22             request: `${ctx.method}
${ctx.path}`
23         };
24         return ;
25     }
26     // 判断当前错误是否为Http请求错误
27     const isHttpException = error
instanceof HttpException;
28     if (isHttpException){
29         // 设置状态码
30         ctx.status = error.code;
31         // 设置错误响应数据
32         ctx.body = {
33             msg:error.msg,
34             error_code:error.errorCode,
35             request: `${ctx.method}
${ctx.path}`
36         }
```

```

37         }else{
38             // 未知错误
39             ctx.response.status = 500
40             ctx.body = {
41                 msg: "未知错误! ",
42                 error_code: 9999,
43                 request: `${ctx.method}
                        ${ctx.path}`
44             }
45         }
46     }
47 }
48 module.exports = catchError;

```

app.js使用该错误处理中间件

```

1  const catchError =
    require('./middlewares/exception')
2  app.use(catchError);

```

6、Api接口编写

所有的接口前缀

```

1  http://localhost:3000/api/v1

```


管理员接口

-管理员注册

routes文件夹下新建admin.js

```
1 module.exports = (router) => {  
2   //路由前缀  
3   router.prefix('/api/v1/admin')  
4 }
```

app.js 注册路由

```
1 const admin = require('./routes/admin')  
2 admin(router)
```

在 route.js/admin.js 管理员路由接口进行管理

```
1 const AdminController =  
  require('../controller/admin')  
2 module.exports = (router) => {  
3   router.prefix('/api/v1/admin')  
4   // 注册管理员  
5   router.post('/register',  
    AdminController.createAdmin)  
6  
7 }
```

```
1  const AdminModel =  
    require('../models/AdminModel');  
2  const { registerValidator } =  
    require('../validators/admin')  
3  const { Resolve } = require('../lib/helper');  
4  const res = new Resolve();  
5  class AdminController{  
6      //注册管理员  
7      static async createAdmin(ctx,next){  
8          // 注册参数校验  
9          registerValidator(ctx);  
10         const { nickname, password2 } =  
            ctx.request.body;  
11         const adminUser = await  
            AdminModel.findOne({ nickname:  
            ctx.vals.nickname })  
12         if (adminUser) {  
13             throw new global.errs.Existing('管理员已存  
            在');  
14         }  
15         const admin = await AdminModel.create({  
16             nickname: nickname,  
17             password: password2,  
18         })
```

```

19      // 返回结果
20      ctx.response.status = 200;
21      ctx.body = res.json(admin);
22    }
23  }
24  module.exports = AdminController;

```

由注册管理员的逻辑看，我们从前台传来的 `nickname`、`password1`、`password2` 字段需要在后端进行校验。

`koa-bouncer` 参数校验

下载 `koa-bouncer` 插件, [文档](#)

```

1 | npm i koa-bouncer -S

```

`app.js`中注册中间件

```

1 | app.use(bouncer.middleware());

```

这样我们就可以通过 `koa-bouncer` 提供的方法对参数进行校验

新建 `validators/admin.js`

```

1 | function registerValidator(ctx) {
2 |   ctx.validateBody('nickname')
3 |     .required('用户名是必须的') //只是要求有uname
   |  字段

```

```

4      .isString() //确保输入的字段是字符串或者可以转
      换成字符串
5      .trim()
6      .isLength(3, 16, '用户名长度必须是3~16位')
7      ctx.validateBody('password1')
8      .required('密码是必填项')
9      .isLength(6, 16, '密码必须是6~16位字符')
10     .match(/^(?![0-9]+$)(?![a-zA-Z]+$)[0-9A-Za-
      z]/, '密码长度必须在6~22位之间, 包含字符、数字和 _
      ')
11     ctx.validateBody('password2')
12     .required('确认密码是必填项')
13     .eq(ctx.vals.password1, '两次密码不一致')
14 }
15
16
17 module.exports = {
18   registerValidator,
19 }

```

封装返回结果帮助模块

lib/helper.js

```

1 class Resolve {
2   json(data, msg = 'success', errorCode = 0,
      code = 200) {

```

```
3     return {
4         code,
5         msg,
6         errorCode,
7         data
8     }
9 }
10 success(msg = 'success', errorCode = 0, code
    = 200) {
11     return {
12         msg,
13         errorCode,
14         code
15     }
16 }
17 }
18 module.exports = {
19     Resolve
20 }
```

测试接口

打开 `postman`

The screenshot shows a REST client on the left and a MongoDB interface on the right. In the REST client, a POST request to `http://localhost:3000/api/v1/admin/register` is shown with a body containing `nickname: 小马哥`, `password1: 123456`, and `password2: 123456`. The response is a JSON object with `code: 200`, `msg: "success"`, and `data` containing the user's details. In the MongoDB interface, the `admins` collection is shown with a single document containing the same user details.

请求方式: POST 接口地址: http://localhost:3000/api/v1/admin/register

Key: nickname, password1, password2 Value: 小马哥, 123456, 123456

返回结果: { "code": 200, "msg": "success", "errorCode": 0, "data": { "_id": "5ea67d69899d9666f7a32bb8", "nickname": "小马哥", "password": "\$2b\$10\$952EC1FHdBPhtAm/ZIb.ORjnm80DpSoAlIwmYnhRX...", "__v": 0 } } }

Admin模型

测试 **管理员存在** 出现的错误情况,再次发送请求,错误返回情况如下,这样我们在之前编写的错误处理中间件就成功了。

```
1 {
2   "msg": "管理员已存在",
3   "error_code": 10006,
4   "request": "POST /api/v1/admin/register"
5 }
```

-管理员登录

router/admin.js

```
1 // 管理员登录
2 router.post('/login', AdminController.login)
```

controller/admin.js

```
1  const LoginManager =  
  require('../services/login');  
2  class AdminController{  
3      static async login(ctx,next){  
4          adminValidator(ctx);  
5          const { nickname,password } =  
ctx.request.body;  
6          const userInfo = await  
LoginManager.adminLogin({nickname,password});  
7          ctx.status = 200;  
8          ctx.body = {  
9              code:200,  
10             msg: '登录成功',  
11             data:userInfo  
12         }  
13     }  
14 }
```

登录参数校验

validators/admin.js

```
1 function adminValidator(ctx) {
2   ctx.validateBody('password')
3     .required('密码是必须的')
4     .trim()
5     .isLength(6, 16, '密码至少6个字符，最多16个字符')
6     .match(/^(?![0-9]+$)(?![a-zA-Z]+$)[0-9A-Za-z]/, '密码长度必须在6~22位之间，包含字符、数字和 _')
7 }
8
9 module.exports = {
10   registerValidator,
11   adminValidator
12 }
```

管理员登录业务层

在 `services/login.js` 中操作管理员登录的业务

```
1 const AdminModel =
   require('../models/AdminModel')
2 const bcrypt = require('bcrypt')
3 const { generateToken } =
   require('../core/util')
4 class LoginManager {
```



```
5 // 管理员登录
6 static async adminLogin(params) {
7     const { nickname, password } = params;
8
9     // 验证账号和密码是否正确
10    const admin = await AdminModel.findOne({
11        nickname }).lean();
12    if (!admin) {
13        throw new global.errs.AuthFailed('账号不存在或者密码不正确');
14    }
15    // 验证密码是否正确
16    const correct =
17    bcrypt.compareSync(password, admin.password);
18    if (!correct) {
19        throw new global.errs.AuthFailed('账号不存在或者密码不正确');
20    }
21    // 生成token
22    return {
23        nickname: admin.nickname,
24        token: generateToken(admin._id)
25    }
26 }
27 module.exports = LoginManager
```

在 `core/util.js` 声明颁布token模块

```
1  const jwt = require('jsonwebtoken');
2  //颁布令牌
3  const generateToken = function (_id) {
4      const secretKey =
5      global.config.security.secretKey
6      const expiresIn =
7      global.config.security.expiresIn
8      const token = jwt.sign(
9          {
10             data: _id, //由于签名不是加密,令牌不要存放敏感数据
11             exp: expiresIn//过期时间一分钟
12         }, secretKey
13     )
14     return token
15 }
16 module.exports = {
17     generateToken
18 }
```

`global.config.security.secretKey`的使用, 提前要在`app.js`中对此声明

`app.js`

```
const config = require('./config');
```

```
global.config = config
```

测试接口

The screenshot shows the Postman interface with a POST request to `http://localhost:3000/api/v1/admin/login`. The request body is `x-www-form-urlencoded` with the following data:

Key	Value	Description
nickname	小马哥	
password	mjj123456	

The response is a JSON object:

```
{  "code": 200,  "msg": "登录成功",  "data": {    "nickname": "小马哥",    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjoiaWVhbnR5cCI6IkpXVCJ9.eyJkYXRhIjoiaWVhbnR5cCI6IkpXVCJ9"  }}
```

The status is 200 OK and the time taken is 174 ms.

-管理员权限

```
routes/admin.js
```

```
1 const AdminController =  
  require('../controller/admin')  
2 const jwtAuth = require('koa-jwt');  
3 module.exports = (router) => {  
4   const secretKey =  
    global.config.security.secretKey  
5   // 获取用户信息  
6   router.get('/auth', jwtAuth({ secret:  
    secretKey }), AdminController.getAuth);  
7 }
```

controller/admin.js

```
1 static async getAuth(ctx,next){  
2   console.log(ctx.state);  
3   // 可以通过ctx.state命名空间获取用户数据  
4   const _id = ctx.state.user.data;  
5   // 查询用户信息  
6   let userInfo = await  
    AdminModel.findById({_id});  
7   if(!userInfo){  
8     throw new global.errs.AuthFailed('账号不存在或者密码不正确')  
9   }  
10   ctx.status = 200;  
11   ctx.body = res.json({
```

```
12     _id: userInfo._id,  
13     nickname: userInfo.nickname,  
14   });  
15 }
```

测试接口

权限 Examples (0)

GET http://localhost:3000/api/v1/admin/auth Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

Key	Value	Description
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...	
New key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK Time: 117 ms

必须在请求头中设置该字段

Pretty Raw Preview JSON Save Response

```
1 {  
2   "code": 200,  
3   "msg": "success",  
4   "errorCode": 0,  
5   "data": {  
6     "_id": "5ea6997e800be16f8640a8a1",  
7     "nickname": "小马哥"  
8   }  
9 }
```

分类接口

创建分类

新建routes/category.js

```
1  const CategoryController =  
    require('../controller/category')  
2  const jwtAuth = require('koa-jwt');  
3  module.exports = (router) => {  
4      router.prefix('/api/v1')  
5      const secretKey =  
        global.config.security.secretKey  
6      // 创建分类  
7  
      router.post('/category', jwtAuth({secret: secretKey}), CategoryController.createCategory);  
8  }
```

app.js

```
1  //注册路由  
2  const category = require('../routes/category');  
3  category(router)
```

controller/category.js

```
1  const CategoryModel =  
    require('../models/CategoryModel')
```

```
2  const { categoryValidator } =  
    require('../validators/category');  
3  class CategoryController{  
4      // 创建分类  
5      static async createCategory(ctx,next){  
6          // 校验参数  
7          categoryValidator(ctx);  
8          const {name,keyword} = ctx.request.body;  
9          const hasCategory = await  
    CategoryModel.findOne({name});  
10         if (hasCategory){  
11             throw new global.errs.Existing('分类已存  
    在');  
12         }  
13         const category = await  
    CategoryModel.create({name,keyword});  
14         ctx.status = 200;  
15         ctx.body = res.json(category);  
16     }  
17 }  
18 module.exports = CategoryController;
```

校验参数模块

新建 `validators/category.js`

```
1 function categoryValidator(ctx) {  
2     ctx.validateBody('name')  
3     .required('分类 title名字不能为空')  
4     .isString()  
5     .trim()  
6     ctx.validateBody('keyword')  
7     .required('分类关键字 keyword不能为空')  
8     .isString()  
9     .trim()  
10 }  
11 module.exports = {  
12     categoryValidator  
13 }
```

测试接口

添加分类

POST http://localhost:3000/api/v1/category Params Send

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

	Key	Value	Description
<input checked="" type="checkbox"/>	name	Vue	
<input checked="" type="checkbox"/>	keyword	vue,vue-router	
	New key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview JSON

```
1 {
2   "code": 200,
3   "msg": "success",
4   "errorCode": 0,
5   "data": {
6     "_id": "5ea793a3c85e3d79157e9ca5",
7     "name": "Vue",
8     "keyword": "vue,vue-router",
9     "__v": 0
10  }
11 }
```

获取所有分类

routes/category.js

```
1 // 获取所有分类
2 router.get('/category',
  CategoryController.getCategoryList);
```

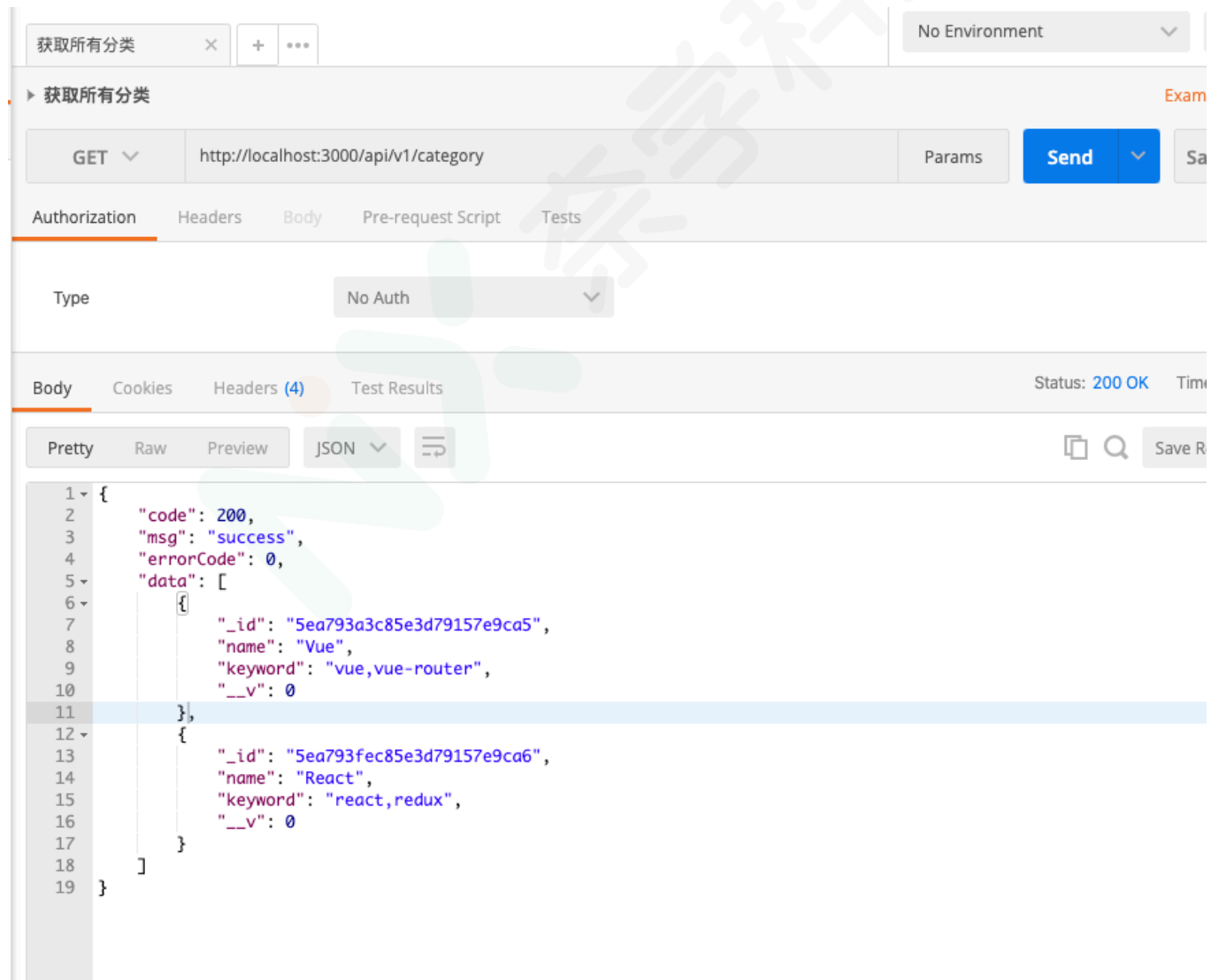
controller/category.js

```

1 // 获取所有分类
2 static async getCategoryList(ctx, next) {
3
4     const categoryList = await
5     CategoryModel.find();
6     ctx.status = 200;
7     ctx.body = res.json(categoryList);
8 }

```

测试接口



The screenshot shows a REST client interface with the following details:

- Request:** GET `http://localhost:3000/api/v1/category`
- Response Status:** 200 OK
- Response Body (JSON):**

```

{
  "code": 200,
  "msg": "success",
  "errorCode": 0,
  "data": [
    {
      "_id": "5ea793a3c85e3d79157e9ca5",
      "name": "Vue",
      "keyword": "vue,vue-router",
      "__v": 0
    },
    {
      "_id": "5ea793fec85e3d79157e9ca6",
      "name": "React",
      "keyword": "react,redux",
      "__v": 0
    }
  ]
}

```

更新分类

routes/category.js

```
1 // 更新分类
2 router.put('/category/:_id', jwtAuth({ secret:
  secretKey })),
  CategoryController.updateCategory);
```

controller/category.js

```
1 // 更新分类
2 static async updateCategory(ctx, next) {
3   const _id = ctx.params._id;
4   const { name, keyword } = ctx.request.body;
5   // category 为通过_id获取的已有分类数据
6   const category = await
  CategoryModel.findByIdAndUpdate({ _id }, {
    name, keyword });
7   if(!category){
8     throw new global.errs.NotFound('没有找到
  相关分类')
9   }
10   ctx.body = res.success('更新分类成功');
11 }
```

测试接口

更新分类

No Environment

更新分类

更新分类的ID

PUT http://localhost:3000/api/v1/category/5ea793fec85e3d79157e9ca6 Params Send

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

Key	Value	Description
<input checked="" type="checkbox"/> name	Redux	
<input checked="" type="checkbox"/> keyword	react,redux	
New key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview JSON

```
1 {
2   "msg": "更新分类成功",
3   "errorCode": 0,
4   "code": 200
5 }
```

删除分类

routes/category.js

```
1 //删除分类
2 router.delete('/category/:_id', jwtAuth({ secret:
  secretKey })),
  CategoryController.deleteCategory);
```

controller/category.js

```
1 // 删除分类
2 static async deleteCategory(ctx,next){
3     const _id = ctx.params._id;
4     const category = await
CategoryModel.findByIdAndDelete({_id});
5     if (!category) {
6         throw new global.errs.NotFound('没有找到相
关分类')
7     }
8     ctx.body = res.success('删除分类成功')
9 }
```

⚠ 删除分类之前，需要对id参数进行校验

测试接口

删除分类

+

...

No Environment

删除分类

DELETE

http://localhost:3000/api/v1/category/5ea793fec85e3d79157e9ca6

Params

Send

Authorization

Headers

Body

Pre-request Script

Tests

Type

No Auth

Body

Cookies

Headers (4)

Test Results

Status: 200 OK

Pretty

Raw

Preview

JSON

↔

📄 🔍

1

{

2

"msg": "删除分类成功",

3

"errorCode": 0,

4

"code": 200

5

}

文章接口

创建文章

新建 routes/article.js

```
1 | const ArticleController =  
  | require('../controller/article')  
2 | const jwtAuth = require('koa-jwt');  
  
3 | module.exports = (router) => {  
4 |   // 创建文章  
5 |   router.post('/article', jwtAuth({ secret:  
  | global.config.security.secretKey })),  
  | ArticleController.createArticle);  
6 | }  
7
```

app.js

```
1 | //注册路由  
2 | const article = require('../routes/article');  
3 | article(router)
```

新建 controller/article.js

```
1 | const ArticleModel =  
  | require('../models/ArticleModel');  
2 | const { articleValidator } =  
  | require('../validators/article')  
3 | const { Resolve } = require('../lib/helper');  
4 | const res = new Resolve();
```

```

5 class ArticleController {
6   // 创建文章
7   static async createArticle(ctx, next) {
8     // 验证参数
9     articleValidator(ctx);
10    const { title } = ctx.request.body;
11    const hasArticle = await
ArticleModel.findOne({ title })
12    if (hasArticle) {
13      throw new global.errs.Existing('文章已存
在')
14    }
15    await
ArticleModel.create(ctx.request.body);
16    ctx.body = res.success('创建文章成功');
17  }
18 }
19 module.exports = ArticleController;

```

校验参数

新建 `validators/article.js`

```

1 function articleValidator(ctx) {
2   ctx.validateBody('title')
3     .required('文章标题 title不能为空')
4     .isString()

```



```
5     .trim()
6 ctx.validateBody('author')
7     .required('文章作者 author 不能为空')
8     .isString()
9     .trim()
10 ctx.validateBody('cover')
11     .required('文章封面 cover 不能为空')
12     .isString()
13     .trim()
14 ctx.validateBody('keyword')
15     .required('文章关键字 keyword不能为空')
16     .isString()
17     .trim()
18 ctx.validateBody('description')
19     .required('文章简介 description 不能为空')
20     .isString()
21     .trim()
22 ctx.validateBody('content')
23     .required('文章内容 content 不能为空')
24     .isString()
25     .trim()
26 ctx.validateBody('category_id')
27     .required('文章分类 category_id 不能为空')
28     .isString()
29     .trim()
30
```

```
31 | }
32 module.exports = {
33   articleValidator
34 }
```

获取文章列表

routes/article.js

```
1 // 获取所有文章列表
2 router.get('/article/',
  ArticleController.getArticleList);
```

controller/article.js

```
1 // 获取文章列表
2 static async getArticleList(ctx, next) {
3   // 获取分类id 当前分页的页数, 每页显示的内容数
  关键字
4   let { category_id = null, pageIndex = 1,
  pageSize = 4, keyword } = ctx.query;
5   // 是否存在分类id
6   let filter = {};
7   if (category_id) {
8     filter.category_id = category_id;
9   }
10  // 转整型
```

```
11     pageIndex = parseInt(pageIndex);
12     pageSize = parseInt(pageSize);
13     // 文章
14     // const reg = new RegExp(keyword, 'i');//不
区分大小写
15     // 获取文章总数量
16     const total = await
ArticleModel.find().count()
17     // 获取分页之后的文章列表
18     const articleList = await ArticleModel
19     .find()
20     .where(filter)
21     .skip((pageIndex - 1) * pageSize)
22     .limit(pageSize)
23     .or([
24         // 模糊查询  query.or([{条件1},{条件2}])
25         {
26             keyword: {
27                 $regex: new RegExp(keyword,
'i')
28             }
29         }
30     ])
31     .sort({ _id: -1 }) //倒序
32     .populate('category_id') //连表查询
33     const data = {
```

```
34         articleList,  
35         currentPage:pageIndex,  
36         total,  
37         pageSize  
38     }  
39     ctx.body = res.json(data);  
40 }
```

测试接口

更新文章

routes/article.js

```
1 // 更新文章  
2 router.put('/article/:_id',jwtAuth({ secret:  
    global.config.security.secretKey })),  
    ArticleController.updateArticle);
```

controller/article.js

```
1 // 更新文章
2 static async updateArticle(ctx,next){
3     const _id = ctx.params._id;
4     let article = await
ArticleModel.findByIdAndUpdate({_id},ctx.request
.body)
5     if(!article){
6         throw new global.errs.NotFound('没有找到相
关文章');
7     }
8     ctx.body = res.success('更新文章成功');
9 }
```

测试接口

获取文章详情

routes/article.js

```
1 // 获取文章详情
2 router.get('/article/:_id',
ArticleController.getArticleDetail);
```

controller/article.js

```
1 // 获取文章详情
```

```
2 static async getArticleDetail(ctx,next){
3     const _id = ctx.params._id;
4     console.log(_id);
5
6     // 获取文章详情表
7     const articleDetail = await
ArticleModel.findById({_id}).populate('category
_id');
8     // 更新文章浏览数
9     await ArticleModel.findByIdAndUpdate({ _id
}, { browse: ++articleDetail.browse })
10     if (!articleDetail){
11         throw new global.errs.NotFound('没有找到
相关文章');
12     }
13     /*
14     todo:
15     获取该文章下的评论列表
16     const commentList = [];
17     */
18     const data = {
19         articleDetail,
20         commentList
21     }
22     //
23     ctx.body = res.json(data);
```

注意：文章详情下有相关的评论数据，未来做这件事情

测试接口

删除文章

routes/article.js

```
1 // 删除文章
2 router.delete('/article/:_id', jwtAuth({ secret:
  global.config.security.secretKey
}), ArticleController.deleteArticle);
```

controller/article.js

```
1 // 删除文章
2 static async deleteArticle(ctx,next){
3     const _id = ctx.params._id;
4     let article = await
ArticleModel.findByIdAndDelete({_id})
5     if(!article){
6         throw new global.errs.NotFound('没有找到相
关文章')
7     }
8     ctx.status = 200;
9     ctx.body = res.success('删除文章成功');
10 }
```

测试接口

评论接口

添加评论

新建 routes/comment.js


```
1 | const CommentController =  
  | require('../controller/comment')  
2 | module.exports = (router) => {  
3 |   router.post('/comment',  
    | CommentController.createComment)  
  
4 | }
```

app.js

```
1 | const comment = require('../routes/comment.js')  
2 | comment(router);
```

controller/comment.js

```
1 | const CommentModel =  
  | require('../models/CommentModel')  
2 | const { commentValidator } =  
  | require('../validators/comment');  
3 | const { Resolve } = require('../lib/helper')  
4 | const res = new Resolve();  
5 | class CommentController {  
6 |   static async createComment(ctx, next){  
7 |     // 校验参数  
8 |     commentValidator(ctx);  
9 |     await CommentModel.create(ctx.request.body)
```

```
10     ctx.body = res.success('创建评论成功');
11 }
12
13 }
14 module.exports = CommentController;
```

校验参数

validators/comment.js

```
1 function commentValidator(ctx) {
2   ctx.validateBody('nickname')
3     .required('评论人名字 nickname不能为空')
4     .isString()
5     .trim()
6   ctx.validateBody('content')
7     .required('评论内容 content不能为空')
8     .isString()
9     .trim()
10  ctx.validateBody('target_id')
11    .required('目标 target_id不能为空')
12    .isString()
13    .trim()
14  ctx.validateBody('target_id')
15    .required('目标 target_id不能为空')
16    .isString()
17    .trim()
}
```

```
18 }
19 module.exports = {
20   commentValidator
21 }
```

测试接口

创建评论

POST http://localhost:3000/api/v1/comment Params Send

Authorization Headers (1) Body Pre-request Script Tests

☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary

	Key	Value	Description
<input checked="" type="checkbox"/>	nickname	小黄	
<input checked="" type="checkbox"/>	content	讲的真好	
<input checked="" type="checkbox"/>	target_id	5ea7d9e29117178057ca6905	
<input checked="" type="checkbox"/>	target_type	article	
	New key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview JSON

```
1 {
2   "msg": "创建评论成功",
3   "errorCode": 0,
4   "code": 200
5 }
```

获取评论列表

routes/comment.js

```
1 //获取评论列表接口

2 router.get('/comment',
  CommentController.getCommentList)
```

controller/comment.js

```
1 // 获取评论列表
2 static async getCommentList(ctx, next) {
3   const { pageIndex = 1, pageSize = 4 } =
  ctx.query;
4   // 获取总数
5   const total = await
  CommentModel.find().count();
6   const commentList = await
  CommentModel.find().skip((parseInt(pageIndex -
  1) * pageSize)).sort([[ '_id',
  -1]]).limit(parseInt(pageSize)).lean();
7   ctx.status = 200;
8   const data = {
9     commentList,
10    currentPage: parseInt(pageIndex),
11    total,
12    pageSize: parseInt(pageSize)
13  }
```

```
14     ctx.body = res.json(data);
15 }
```

测试接口

获取评论列表

GET Params

Authorization Headers Body Pre-request Script Tests

Type

Body Cookies Headers (4) Test Results Status: 200 OK Time: 4

Pretty Raw Preview JSON

```
1 {
2   "code": 200,
3   "msg": "success",
4   "errorCode": 0,
5   "data": {
6     "commentList": [
7       {
8         "_id": "5ea8012b061833013bfeed0",
9         "nickname": "小黄",
10        "content": "讲的真好",
11        "target_id": "5ea7d9e29117178057ca6905",
12        "target_type": "article",
13        "__v": 0
14      }
15    ],
16    "currentPage": 1,
17    "total": 1,
18    "pageSize": 4
19  }
20 }
```

获取评论详情

routes/comment.js

```
1 | router.get('/:id', CommentController.get
  | CommentDetailById)
```

controller/comment.js

```
1 static async getCommentDetailById(ctx,next){
2     const _id = ctx.params._id;
3     const commentDetail = await
CommentModel.findById({_id}).lean();
4     if(!commentDetail){
5         throw new global.errs.NotFound('没有找到
相关评论信息')
6     }
7     // todo: 获取该评论的回复列表
8     const replyList = [];
9     const data = {
10         commentDetail,
11         replyList
12     }
13     ctx.status = 200;
14     ctx.body = res.json(data);
15 }
```

测试接口

获取评论详情

GET http://localhost:3000/api/v1/comment/5ea8012b061833013bfeed0 Params Send

Authorization Headers Body Pre-request Script Tests

Type No Auth

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview JSON

```
1 {
2   "code": 200,
3   "msg": "success",
4   "errorCode": 0,
5   "data": {
6     "commentDetail": {
7       "_id": "5ea8012b061833013bfeed0",
8       "nickname": "小黄",
9       "content": "讲的真好",
10      "target_id": "5ea7d9e29117178057ca6905",
11      "target_type": "article",
12      "__v": 0
13    },
14    "replyList": []
15  }
16 }
```

更新评论

routes/comment.js

```
1 const jwtAuth = require('koa-jwt');
2 const secretKey =
  global.config.security.secretKey
3 router.put('/comment/:_id', jwtAuth({secret: secretKey}), CommentController.updateCommentById)
```

controller/comment.js

```
1 // 更新评论
2 static async updateCommentById(ctx,next){
3     const _id = ctx.params._id;
4     const comment = await
5     CommentModel.findByIdAndUpdate({
6     _id},ctx.request.body);
7     if(!comment){
8         throw new global.errs.NotFound('没有找到
9     相关评论')
10    }
11    ctx.body = res.success('更新评论成功');
12 }
```

测试接口

更新评论

PUT http://localhost:3000/api/v1/comment/5ea8012b061833013bfeed0 Params Send Save

Authorization Headers (2) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

Key	Value	Description
nickname	小黑	
content	哈哈哈哈哈	
New key		Description

Body Cookies Headers (4) Test Results Status: 200 OK Time: 6s

Pretty Raw Preview JSON Save Response

```
{
  "msg": "更新评论成功",
  "errorCode": 0,
  "code": 200
}
```

删除评论

routes/comment.js

```
1 // 删除评论接口
2 router.delete('/comment/:_id', jwtAuth({ secret:
  secretKey })),
  CommentController.deleteCommentById)
```

controller/comment.js

```
1 // 删除评论

2 static async deleteCommentById(ctx, next){
3     const _id = ctx.params._id;
4     const comment = await
CommentModel.findOneAndDelete({_id});
5     if(!comment){
6         throw new global.errs.NotFound('没有找到相
关评论')
7     }
8     ctx.body = res.success('删除评论成功');
9 }
```

测试接口

删除评论

DELETE Params

Authorization Headers (2) Body Pre-request Script Tests

	Key	Value	Description	...	Bulk Edit	Pin
<input checked="" type="checkbox"/>	Content-Type	application/x-www-form-urlencoded				
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...				
	New key	Value	Description			

Body Cookies Headers (4) Test Results Status: 200 OK Time

Pretty Raw Preview JSON

```
1 {
2   "msg": "删除评论成功",
3   "errorCode": 0,
4   "code": 200
5 }
```

回复接口

创建回复

新建 `routes/reply.js`

```
1 | const ReplyController =  
  | require('../controller/reply')  
2 | module.exports = (router) => {  
3 |   // 创建回复  
4 |   router.post('/reply', ReplyController.createReply);  
  |  
5 | }
```

注册路由

app.js

```
1 | const reply = require('./routes/reply.js')  
2 | reply(router)
```

controller/reply.js

```
1 | const ReplyModel =  
  | require('../models/ReplyModel')  
2 | const CommentModel =  
  | require('../models/CommentModel')  
3 | const { replyValidator } =  
  | require('../validators/reply');  
4 | const { Resolve } = require('../lib/helper')  
5 | const res = new Resolve();  
6 | class ReplyController {
```

```

7 // 创建回复
8 static async createReply(ctx, next) {
9     replyValidator(ctx);
10    const { comment_id } = ctx.request.body;
11    const comment = await
CommentModel.findById({_id:comment_id});
12    if(!comment){
13        throw new global.errs.NotFound('没有找到相
关评论');
14    }
15    const reply = await
ReplyModel.create(ctx.request.body);
16    // 要想调用get方法 必须将数据转成json
17    reply.toJSON({getters:true});
18    ctx.body = res.json(reply);
19 }
20 }
21 module.exports = ReplyController;

```

校验参数

validators/reply.js

```

1 function replyValidator(ctx) {
2     ctx.validateBody('nickname')
3     .required('评论人名字 nickname不能为空')
4     .isString()

```

```
5     .trim()
6   ctx.validateBody('content')
7     .required('评论内容 content不能为空')
8     .isString()
9     .trim()
10  ctx.validateBody('comment_id')
11    .required('评论人 comment_id不能为空')
12    .isString()
13    .trim()
14  }
15  module.exports = {
16    replyValidator
17  }
```

测试接口

创建回复

POST http://localhost:3000/api/v1/reply Params Send Save

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

Key	Value	Description
nickname	小鲜花	
content	赞👍	
comment_id	5ea8eb5dd21fe010427e51ac	
New key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK Time:

Pretty Raw Preview JSON Save Re

```
1 {
2   "code": 200,
3   "msg": "success",
4   "errorCode": 0,
5   "data": {
6     "_id": "5ea9187d1509b611f49adece",
7     "nickname": "小鲜花",
8     "content": "赞👍",
9     "comment_id": "5ea8eb5dd21fe010427e51ac",
10    "createAt": "2020-04-29",
11    "__v": 0,
12    "id": "5ea9187d1509b611f49adece"
13  }
14 }
```

回复列表

routes/reply.js

```
1 // 获取评论的回复列表
2 router.get('/reply', ReplyController.getReplyList
)
```

controller/reply.js

```

1 // 获取该评论的回复列表
2 static async getReplyList(ctx,next){
3     const comment_id = ctx.query.comment_id;
4     const replyList = await ReplyModel.find({
comment_id}).sort({'_id':-1});
5     ctx.status = 200;
6     ctx.body = res.json(replyList);d
7 }

```

测试接口

获取该评论的回复列表 Examples (0) ▾

GET Params Send ▾ Save ▾

Authorization Headers Body Pre-request Script Tests Code

Type ▾

Body Cookies Headers (4) Test Results Status: 200 OK Time: 48 ms

Pretty Raw Preview Save Response

```

1 {
2   "code": 200,
3   "msg": "success",
4   "errorCode": 0,
5   "data": [
6     {
7       "_id": "5ea9187d1509b611f49adece",
8       "nickname": "小鲜花",
9       "content": "赞👍",
10      "comment_id": "5ea8eb5dd21fe010427e51ac",
11      "createAt": "2020-04-29",
12      "__v": 0,
13      "id": "5ea9187d1509b611f49adece"
14    },
15    {
16      "_id": "5ea8ebedf05812107010e9cd",
17      "nickname": "小树林",
18      "content": "赞同",
19      "comment_id": "5ea8eb5dd21fe010427e51ac",
20      "createAt": "2020-04-29",
21      "__v": 0,
22      "id": "5ea8ebedf05812107010e9cd"
23    }
24  ]

```


回复详情

routes/reply.js

```
1 | router.get('/reply/:_id', ReplyController.getReplyDetailById)
```

controller/reply.js

```
1 | // 获取该评论的回复详情
2 | static async getReplyDetailById(ctx, next){
3 |     const _id = ctx.params._id;
4 |     const replyDetail = await
ReplyModel.findById({_id});
5 |     if(!replyDetail){
6 |         throw new global.errs.NotFound('没有相关
评论的回复详情信息')
7 |     }
8 |     ctx.body = res.json(replyDetail);
9 | }
```

测试接口

获取回复详情 Example

GET Params Send Save

Authorization Headers Body Pre-request Script Tests

Type

Body Cookies Headers (4) Test Results Status: 200 OK Time: 10

Pretty Raw Preview Save Resp

```
1 {
2   "code": 200,
3   "msg": "success",
4   "errorCode": 0,
5   "data": {
6     "_id": "5ea9187d1509b611f49adece",
7     "nickname": "小鲜花",
8     "content": "赞👍",
9     "comment_id": "5ea8eb5dd21fe010427e51ac",
10    "createAt": "2020-04-29",
11    "_v": 0,
12    "id": "5ea9187d1509b611f49adece"
13  }
14 }
```

更新回复

routes/reply.js

```
1 const jwtAuth = require('koa-jwt');
2 // 更新单个回复
3 router.put('/reply/:_id', jwtAuth({secret: global.
  config.security.secretKey})
  , ReplyController.updateReplyById)
```

controller/reply.js

```
1 static async updateReplyById(ctx,next){
2     const _id = ctx.params._id;
3     const reply = await
ReplyModel.findByIdAndUpdate({_id},ctx.request.b
ody)
4     if(!reply){
5         throw new global.errs.NotFound('没有找到
相关评论信息')
6     }
7     ctx.status = 200;
8     ctx.body = res.success('更新评论成功');
9 }
```

测试接口

更新回复

PUT Params

Authorization Headers (2) **Body** Pre-request Script Tests

☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary

	Key	Value	Description	..
<input checked="" type="checkbox"/>	nickname	黑色妖姬		
<input checked="" type="checkbox"/>	content	哈哈哈哈哈		
<input checked="" type="checkbox"/>	comment_id	5ea9242dd793541394903564		
	New key	Value	Description	

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview JSON

```
1 {
2   "msg": "更新回复成功",
3   "errorCode": 0,
4   "code": 200
5 }
```

删除回复

routes/reply.js

```
1 router.delete('/reply/:_id', jwtAuth({secret: global.config.security.secretKey}), ReplyController.deleteReplyById);
```

controller/reply.js

```
1 static async deleteReplyById(ctx,next){
2     const _id = ctx.params._id;

3     const reply = await
ReplyModel.findByIdAndDelete({_id})
4     if(!reply){
5         throw new global.errs.NotFound('没有找到
相关评论信息')
6     }
7     ctx.status = 200;
8     ctx.body = res.success('删除评论成功');
9 }
```

测试接口

删除回复

DELETE

http://localhost:3000/api/v1/reply/5ea9187d1509b611f49adece

Params

Send

Authorization

Headers (2)

Body

Pre-request Script

Tests

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	Content-Type	application/x-www-form-urlencoded			
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...			
	New key	Value	Description		

Body

Cookies

Headers (4)

Test Results

Status: 200 OK

Pretty

Raw

Preview

JSON

Save

```
1 {
2   "msg": "删除回复成功",
3   "errorCode": 0,
4   "code": 200
5 }
```

广告接口

创建广告

新建 routes/advertise.js

```

1 | const AdvertiseController =
   | require('../controller/advertise')
2 | module.exports = (router) => {
   |   // 创建广告
3 |   router.post('/advertise', AdvertiseController.c
   | reateAdvertise);
4 | }
5 |
6 |

```

注册路由

app.js

```

1 | const advertise = require('../routes/advertise')
2 | advertise(router)

```

controller/advertise.js

```

1 | const AdvertiseModel =
   | require('../models/AdvertiseModel')
2 | const { advertiseValidator } =
   | require('../validators/advertise');
3 | const { Resolve } = require('../lib/helper')
4 | const res = new Resolve();
5 | class ReplyController {
6 |   // 创建广告
7 |   static async createAdvertise(ctx, next) {

```

```
8     advertiseValidator(ctx);
9     const { title } = ctx.request.body
10     let hasAdvertise = await
    AdvertiseModel.findOne({ title })
11     if (hasAdvertise) {
12         throw new global.errs.Existing('广告已存
    在')
13     }
14     let advertise = await
    AdvertiseModel.create(ctx.request.body)
15     ctx.body = res.json(advertise)
16 }
17 }
18 module.exports = ReplyController;
```

校验参数

validators/advertise.js


```
1 function advertiseValidator(ctx) {
2   ctx.validateBody('title')
3     .required('广告标题 title不能为空')
4     .isString()
5     .trim()
6   ctx.validateBody('link')
7     .required('广告链接 link不能为空')
8     .isString()
9     .trim()
10 }
11 module.exports = {
12   advertiseValidator
13 }
```

测试接口

创建广告

POST http://localhost:3000/api/v1/advertise Params Send

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

Key	Value	Description
<input checked="" type="checkbox"/> title	Vue官网	
<input checked="" type="checkbox"/> link	https://cn.vuejs.org/	
New key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview JSON

```
1 {
2   "code": 200,
3   "msg": "success",
4   "errorCode": 0,
5   "data": {
6     "_id": "5ea936633231ba18be3b0c80",
7     "title": "Vue官网",
8     "link": "https://cn.vuejs.org/",
9     "createAt": "2020-04-29 16:10:11",
10    "_v": 0,
11    "id": "5ea936633231ba18be3b0c80"
12  }
13 }
```

获取广告列表

routes/advertise.js

```
1 // 获取广告列表
2 router.get('/advertise', AdvertiseController.getAdvertiseList);
```

controller/advertise.js

```
1 // 获取广告列表
2 static async getAdvertiseList(ctx,next){
3     let advertiseList = await
    AdvertiseModel.find();
4     ctx.status = 200;
5     ctx.body = res.json(advertiseList);
6 }
```

测试接口

广告列表

GET Params

Authorization Headers Body Pre-request Script Tests

Type No Auth

Body Cookies Headers (4) Test Results Status: 200 OK Time

Pretty Raw Preview JSON

```
1 {
2   "code": 200,
3   "msg": "success",
4   "errorCode": 0,
5   "data": [
6     {
7       "_id": "5ea936633231ba18be3b0c80",
8       "title": "Vue官网",
9       "link": "https://cn.vuejs.org/",
10      "createAt": "2020-04-29 16:10:11",
11      "__v": 0,
12      "id": "5ea936633231ba18be3b0c80"
13    }
14  ]
15 }
```

获取广告详情

routes/advertise.js

```
1 // 获取广告详情
2 router.get('/advertise/:_id',
  AdvertiseController.getAdvertiseDetailById);
```

controller/advertise.js

```
1 // 获取广告详情
2 static async getAdvertiseDetailById(ctx,next){
3   let _id = ctx.params._id;
4   let advertiseDetail = await
  AdvertiseModel.findById({_id}).lean();
5   if(!advertiseDetail){
6     throw new global.errs.NotFound('此广告不
  存在')
7   }
8   ctx.status = 200;
9   ctx.body = res.json(advertiseDetail);
10 }
```

测试接口

广告详情

GET Params

Authorization Headers Body Pre-request Script Tests

Type

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview JSON

```
1 {
2   "code": 200,
3   "msg": "success",
4   "errorCode": 0,
5   "data": {
6     "_id": "5ea936633231ba18be3b0c80",
7     "title": "Vue 官网",
8     "link": "https://cn.vuejs.org/",
9     "createAt": "2020-04-29T08:10:11.303Z",
10    "__v": 0
11  }
12 }
```

更新广告

routes/advertise.js

```
1 // 更新广告
2 router.put('/advertise/:_id', jwtAuth({ secret:
  global.config.security.secretKey }),
  AdvertiseController.updateAdvertiseById);
```

controller/advertise.js

```
1 // 更新广告
2 static async updateAdvertiseById(ctx,next){
3     let _id = ctx.params._id;
4     let advertise = await
    AdvertiseModel.findByIdAndUpdate({ _id
    },ctx.request.body).lean();
5     if (!advertise) {
6         throw new global.errs.NotFound('此广告不
    存在')
7     }
8     ctx.status = 200;
9     ctx.body = res.success('更新广告成功')
10 }
```

测试接口

更新广告

PUT

http://localhost:3000/api/v1/advertise/5ea936633231ba18be3b0c80

Params

Send

Authorization

Headers (2)

Body

Pre-request Script

Tests

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	Content-Type	application/x-www-form-urlencoded			
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...			
	New key	Value	Description		

Body

Cookies

Headers (4)

Test Results

Status: 200 OK

Pretty

Raw

Preview

JSON

1

{

2

"msg": "更新广告成功",

3

"errorCode": 0,

4

"code": 200

5

}

删除广告