link: null
title: 珠峰架构师成长计划
description: config\plugin.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=69 sentences=237, words=1196

1. 初始化项目

```
egg-init zfchat-api --type=simple
```

2. 支持socket.io

```
cnpm i egg-socket.io -S
```

- namespace: 通过配置的方式定义namespace(命名空间)
- middleware: 对每一次socket连接/断开、每一次消息/数据传递进行预处理
- controller: 响应socket.io的event事件
- router: 统一了socket.io的event与框架路由的处理配置方式

config\plugin.js

```
exports.io = {
    enable:true,
    package:'egg-socket.io'
}
```

app\router.js

```
;

module.exports = app => {

  const { router, controller, io } = app;
  router.get('/', controller.home.index);

  io.route('addMessage', io.controller.room.addMessage);

  io.route('getAllMessages', io.controller.room.getAllMessages);

  router.post('/login', controller.user.login);

  router.post('/validate', controller.user.validate);

  router.post('/createRoom', controller.rooms.createRoom);

  router.get('/getAllRooms', controller.rooms.getAllRooms);
};
```

app\io\controller\room.js

```
const { Controller } = require('egg');
class RoomController extends Controller {
    async addMessage() {
        let { ctx, app } = this;

        let message = ctx.args[0];
        let doc = await ctx.model.Message.create(message);
        doc = await ctx.model.Message.findById(doc._id).populate('user');

        app.io.emit('messageAdded', doc.toJSON());
    }
    async getAllMessages() {
        let { ctx, app } = this;
        let room = ctx.args[0];
        let messages = await ctx.model.Message.find({ room }).populate('user').sort({ createAt: -1 }).limit(20);
        ctx.socket.emit('allMessages', messages.reverse());
    }
}
module.exports = RoomController;
```

```
  config.io = {
    namespace: {
      "/": {
        connectionMiddleware: ["connect"],
        packetMiddleware:[]
      }
    }
  }
```

- namespace: 通过配置的方式定义namespace(命名空间)
- middleware: 对每一次socket连接/断开、每一次消息/数据传递进行预处理
- controller: 响应socket.io的event事件
- router: 统一了socket.io的event与框架路由的处理配置方式

```
const SYSTEM = {
    name: '系统',
    email: 'admin@126.com',
    avatar: 'http://www.gravatar.com/avatar/1e6fd8e56879c84999cd481255530592'
}
module.exports = app => {
    return async function (ctx, next) {
        const { app, socket, query: { token, room } } = ctx;
        if (token && token != '') {
            const user = app.jwt.verify(token, app.config.jwt.secret);
            if (user) {
                const id = socket.id;
                const nsp = app.io;
                await ctx.model.User.findByIdAndUpdate(user._id, { $set: { online: true, room } });
                socket.join(room);
                socket.broadcast.to(room).emit('online', {
                    user: SYSTEM,
                    content: `用户${user.name}加入聊天室`
                });
                await next();
                await ctx.model.User.findByIdAndUpdate(user._id, { $set: { online: false, room: null } });
                socket.leave(room);
                socket.broadcast.to(room).emit('offline', {
                    user: SYSTEM,
                    content: `用户${user.name}离开聊天室`
                });
            } else {
                socket.emit('needLogin');
            }

        } else {
            socket.emit('needLogin');
        }
    }
}
```

框架是以 Cluster 方式启动的，而 socket.io 协议实现需要 sticky 特性支持，否则在多进程模式下无法正常工作。 修改 package.json 中 npm scripts 脚本：

```
{
  "scripts": {
    "dev": "egg-bin dev --sticky",
    "start": "egg-scripts start --sticky"
  }
}
```

server\server.js

```
var express = require('express');
var app = express();
var path = require('path');
var server = require('http').createServer(app);
app.use(express.static(path.join(__dirname, 'public')));
var io = require('socket.io')(server);
var port = process.env.PORT || 3000;

server.listen(port, () => {
    console.log('Server listening at port %d', port);
});
```

server\public\index.html

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Documenttitle>
head>
<body>
    <button onclick="send()">发送button>
    <script src="/socket.io/socket.io.js">script>
    <script>
        let socket = io('http://localhost:7001/');
        socket.on('connect', function () {
            console.log('连接成功');
            socket.emit('getAllMessages');
        });
        socket.on('messageAdded', function (message) {
            console.log(message);
        });
        socket.on('allMessages', function (messages) {
            console.log(messages);
        });
        socket.on('online', function (message) {
            console.log(message);
        });
        socket.on('offline', function (message) {
            console.log(message);
        });
        function send() {
            socket.emit('addMessage', { content: '你好' });
        }
    script>
body>

html>
```

3.支持mongoose

```
cnpm i egg-mongoose --save
```

config\plugin.js

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

config\config.default.js

```
exports.mongoose = {
  client: {
    url: 'mongodb://127.0.0.1/zfchat',
    options: {},
  },
};
```

```
module.exports = app => {
    const mongoose = app.mongoose;
    const Schema = mongoose.Schema;
    const ObjectId = Schema.Types.ObjectId;
    const MessageSchema = new Schema({
        content: String,
        user: {
            type: ObjectId,
            ref: 'User'
        },
        room: {
            type: ObjectId,
            ref: 'Room'
        },
        createAt: { type: Date, default: Date.now }
    });
    return mongoose.model('Message', MessageSchema);
}
```

4. egg-jwt

```
cnpm i egg-jwt -S
```

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

```
config.jwt = {
   secret: 'zfpx'
}
config.security = {
   csrf: false
}
```

5. egg-cors

```
cnpm i egg-cors -S
```

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

```
config.security={
    domainWhiteList:['http://localhost:3000'],
    csrf:false
  }
```

6. 实现用户登录

```
module.exports = app => {
    let mongoose = app.mongoose;
    let Schema = mongoose.Schema;
    let ObjectId = Schema.Types.ObjectId;
    let UserSchema = new Schema({
        name: String,
        email: String,
        avatar: String,
        online: { type: Boolean, default: false },
        room: {
            type: ObjectId,
            ref: 'Room'
        },
        createAt: { type: Date, default: Date.now }
    });
    return mongoose.model('User', UserSchema);
}
```

app\router.js

```
router.post('/login', controller.user.login);
router.post('/validate', controller.user.validate);
```

app\controller\base.js

```
let { Controller } = require('egg');
class BaseController extends Controller {
    async success(data) {
        this.ctx.body = {
            code: 0,
            data
        }
    }
    async error(error) {
        this.ctx.body = {
            code: 1,
            error
        }
    }
}
module.exports = BaseController;
```

```
cnpm i gravatar -S
```

\app\controller\user.js

```
let BaseController = require('./base');
let gravatar = require('gravatar')

class UserController extends BaseController {
    async login() {
        const { app, ctx } = this;
        let user = ctx.request.body;
        let doc = await ctx.model.User.findOne({ email: user.email });
        if (!doc) {
            user.name = user.email.split('@')[0];
            user.avatar = gravatar.url(user.email);
            doc = await ctx.model.User.create(user);
        }
        let token = app.jwt.sign(doc.toJSON(), app.config.jwt.secret);
        this.success(token);
    }

    async validate() {
        const { app, ctx } = this;
        let { token } = ctx.request.body;
        try {
            let user = app.jwt.verify(token, app.config.jwt.secret);
            this.success(user);
        } catch (error) {
            this.error('用户验证失败');
        }
    }
}
module.exports = UserController;
```

```
{"code":0,"data":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6IjFAMS5jb20iLCJpYXQiOjE1MjcyNTY0NzF9.2tVGRzHLh75oDKbdc7WHciM80JeW3y2Pqu9WKdjeGGU"}
```

7.实现房间管理功能

```
module.exports = app => {
    let mongoose = app.mongoose;
    let Schema = mongoose.Schema;
    let RoomSchema = new Schema({
        name: String,
        createAt: { type: Date, default: Date.now }
    });
    return mongoose.model('Room', RoomSchema);
}
```

```
router.post('/addRoom', controller.rooms.addRoom);
router.post('/getAllRooms', controller.rooms.getAllRooms);
```

app\controller\rooms.js

```
let BaseController = require('./base');
class RoomsController extends BaseController {
    async createRoom() {
        const { ctx, app } = this;
        let room = ctx.request.body;
        let doc = await ctx.model.Room.findOne({ name: room.name });
        if (doc) {
            this.error('房间已经存在!');
        } else {
            doc = await ctx.model.Room.create(room);
            this.success(doc.toJSON());
        }
    }
    async getAllRooms() {
        const { ctx, app } = this;
        let rooms = await ctx.model.Room.find();
        rooms = rooms.map(room => room.toJSON());
        for (let i = 0; i < rooms.length; i++) {
            let users = await ctx.model.User.find({ room: rooms[i]._id });
            rooms[i].users = users.map(user => user.toJSON())
        }
        this.success(rooms);
    }
}
module.exports = RoomsController;
```

参考