link: null
title: 珠峰架构师成长计划
description: app/controller/user.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=70 sentences=274, words=1309

1.初始化项目

```
egg-init --type simple chat-api
```

2. 实现后端的登录接口

```
module.exports = app => {
    const mongoose = app.mongoose;
    const Schema = mongoose.Schema;

    const UserSchema = new Schema({
      email: { type: String  }
    });

    return mongoose.model('User', UserSchema);
  }
```

app/controller/user.js

```
const Controller = require('./base');

class UserController extends Controller {
  async login() {
      const {ctx,app}=this;
      let body = ctx.request.body;
      let oldUser = await ctx.model.User.findOne({email: body.email});
      if (oldUser) {
          ctx.session.user=oldUser;
          this.success(oldUser);
      } else {
          let newUser=new ctx.model.User(body);
          await newUser.save();
          ctx.session.user=newUser;
          this.success(newUser);
      }
  }
}

module.exports = UserController;
```

app/controller/base.js

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

module.exports = BaseController;
```

3. 房间功能

```
module.exports = app => {
  const { router, controller } = app;
  router.post('/login',controller.user.login);
  router.get('/rooms',controller.rooms.list);
  router.post('/rooms',controller.rooms.create);
};
```

config/config.default.js

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

  config.security={
    csrf: false,
    domainWhiteList:['http://127.0.0.1:8000']
  }
```

config/plugin.js

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

app/controller/rooms.js

```
const Controller = require('./base');

class RoomController extends Controller {
  async list() {
      const {ctx,app}=this;
      let {keyword=''}=ctx.query;
      let query={};
      if (keyword) {
          query['name']=new RegExp(keyword);
      }
      let list = await app.model.Room.find(query);
      this.success(list);
  }
  async create() {
      const {ctx,app}=this;
      let body=ctx.request.body;
      let oldRoom = await app.model.Room.findOne({name: body.name});
      if (oldRoom) {
          this.error('房间名已经存在');
      } else {
          let newRoom=new app.model.Room(body);
          await newRoom.save();
          this.success(newRoom);
      }
  }
}

module.exports = RoomController;
```

app/model/room.js

```
module.exports = app => {
    const mongoose = app.mongoose;
    const Schema = mongoose.Schema;

    const RoomSchema = new Schema({
      name: { type: String  }
    });

    return mongoose.model('Room', RoomSchema);
  }
```

4.聊天功能

```
const Controller = require('./base');
const gravatar=require('gravatar');
class UserController extends Controller {
  async login() {
      const {ctx,app}=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=new ctx.model.User(user);
            await doc.save();
      }
      let token = app.jwt.sign(doc.toJSON(),app.config.jwt.secret);
      this.success(token);
  }
}

module.exports = UserController;
```

app/model/room.js

```
module.exports = app => {
    const mongoose = app.mongoose;
    const Schema = mongoose.Schema;
    const ObjectId=Schema.Types.ObjectId;
    const RoomSchema = new Schema({
        name: {type: String},
        createAt: {type: Date,default: Date.now},
        creator:{type:ObjectId,ref:'User'}
    });

    return mongoose.model('Room', RoomSchema);
}
```

app/model/user.js

```
module.exports = app => {
    const mongoose = app.mongoose;
    const Schema = mongoose.Schema;
    const ObjectId=Schema.Types.ObjectId;
    const UserSchema=new Schema({
        name: String,
        avatar:String,
        email: {type: String},
        room: {type: ObjectId,ref: 'Room'},
        socket:{type:String},
        online:{type:Boolean,default:false}
    });

    return mongoose.model('User', UserSchema);
  }
```

app/router.js

```
module.exports = app => {
  const { router, controller,io } = app;
  router.post('/login',controller.user.login);
  router.get('/rooms',controller.rooms.list);
  router.post('/rooms',controller.rooms.create);
  io.route('getRoom',io.controller.messages.getRoom);
  io.route('addMessage',io.controller.messages.addMessage);
};
```

config/config.default.js

```
;

module.exports = appInfo => {
  const config = exports = {};

  config.keys = appInfo.name + '_1533636529759_8753';

  config.middleware=[];

  config.mongoose = {
    client: {
      url: 'mongodb://127.0.0.1/chat',
      options: {},
    },
  };

  config.security={
    csrf: false,
    domainWhiteList:['http://127.0.0.1:8000']
  }
  config.io = {
    init: { wsEngine: 'ws' },
    namespace: {
      '/': {
        connectionMiddleware: ['connect'],
        packetMiddleware: [],
      }
    },
    redis: {
      host: '127.0.0.1',
      port: 6379
    },
  };
  config.jwt = {
    secret: 'zfpx'
  }
  return config;
};
```

config/plugin.js

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

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

package.json

```
"scripts": {
    "start": "egg-scripts start --daemon --title=egg-server-chat-api --sticky",
    "stop": "egg-scripts stop --title=egg-server-chat-api",
    "dev": "egg-bin dev --sticky",
    "debug": "egg-bin debug",
    "test": "npm run lint -- --fix && npm run test-local",
    "test-local": "egg-bin test",
    "cov": "egg-bin cov",
    "lint": "eslint .",
    "ci": "npm run lint && npm run cov",
    "autod": "autod"
}
```

app/io/controller/messages.js

```
const {Controller}=require('egg');
class MessageController extends Controller{
    async addMessage() {
        let {app,ctx}=this;
        let message=ctx.args[0];
        let newMessage=new app.model.Message(message);
        await newMessage.save();
        let doc=await app.model.Message.findById(newMessage._id).populate('user');
        app.io.to(message.room).emit('messageAdded',doc.toJSON());
    }
    async getRoom() {
        let {app,ctx}=this;
        let room=ctx.args[0];
        let users=await app.model.User.find({room,online:true});
        let messages = await app.model.Message.find({room}).sort({createAt:-1}).populate('user').sort({createAt: -1}).limit(20);
        ctx.socket.emit('room',{users,messages:messages.reverse()});
    }
}
module.exports=MessageController;
```

app/io/middleware/connect.js

```
const SYSTEM = {
    name: '系统',
    email: 'admin@126.com',
    avatar: 'http://www.gravatar.com/avatar/1e6fd8e56879c84999cd481255530592'
}
module.exports = app => {
    return async (ctx, next) => {
        const {app,socket,query: {room,token}}=ctx;
        if (token) {
            let decodeUser=app.jwt.verify(token,app.config.jwt.secret);
            if (decodeUser) {
                let oldUser=await app.model.User.findById(decodeUser._id);
                let oldSocket=oldUser.socket;
                 if (oldSocket) {
                    app.io.of('/').adapter.remoteDisconnect(oldSocket, true, err => {
                        app.logger.error(err);
                    });
                }
                oldUser.room=room;
                oldUser.online=true;
                oldUser.socket=socket.id;
                await oldUser.save();
                socket.join(room);
                socket.broadcast.to(room).emit('online',oldUser.toJSON());
                socket.broadcast.to(room).emit('messageAdded',{
                    user: SYSTEM,
                    content:`用户${oldUser.name}加入聊天室`
                });
                await next();
                socket.leave(room);
                socket.broadcast.to(room).emit('offline',oldUser._id);
                socket.broadcast.to(room).emit('messageAdded',{
                    user: SYSTEM,
                    content:`用户${oldUser.name}离开了聊天室`
                });
                oldUser.room=null;
                oldUser.online=null;
                oldUser.socket=null;
                await oldUser.save();
            }else {
                socket.emit('needLogin','你需要先登录');
            }
        } else {
            socket.emit('needLogin','你需要先登录');
        }
    };
};
```

app/model/message.js

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

server/app.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

```
Document

发送

    let socket = io('http://localhost:7001/',{query:{room:'default'}});
    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('message', function (message) {
        console.log(message);
    });
    socket.on('needLogin', function (message) {
        console.log(message);
    });

    function send() {
        socket.emit('addMessage', { content: '你好' });
    }
```

参考