link: null
title: 珠峰架构师成长计划
description: app.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=41 sentences=165, words=972

1.搭建JWT后端环境

```
cnpm i express cors jsonwebtoken cookie-parser morgan mongoose bcryptjs -S
cnpm i nodemon -g
```

2.启动服务

app.js

```
const express=require('express');
const cors=require('cors');
const path=require('path');
const logger=require('morgan');
const cookieParser=require('cookie-parser');
const indexRouter=require('./routes/index');
const usersRouter=require('./routes/users');
const app=express();
app.use(cors());
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({extended: false}));
app.use(cookieParser());
app.use('/',indexRouter);
app.use('/users',usersRouter);
app.listen(3000,() => console.log('server started at port 3000'));
```

routes/index.js

```
const express=require('express');
const router=express.Router();
router.get('/',function (req,res) {
    res.send('index');
});
module.exports=router;
```

routes/users.js

```
const express=require('express');
const router=express.Router();
router.get('/',function (req,res) {
    res.send('users');
});
module.exports=router;
```

```
"scripts": {
    "start": "nodemon ./app.js"
  },
```

3.用户接口

config.js

```
module.exports={
    PORT: 3000,
    DB_URL:"mongodb://localhost/userSystem"
}
```

routes/users.js

```
const express=require('express');
const User=require('../models/user');
const router=express.Router();
router.post('/signup',async function (req,res) {
    let user=new User(req.body);
    try {
        await user.save();
        res.json({
            code: 0,
            data:user
        });
    } catch (error) {
        res.json({
            code: 1,
            error
        });
    }

});
module.exports=router;
```

models/index.js

```
const {DB_URL} = require('../config');
const mongoose=require('mongoose');
let connection=mongoose.createConnection(DB_URL);
module.exports=connection;
```

models/user.js

```
const mongoose=require('mongoose');
const connection=require('./index');
const Schema=mongoose.Schema;
const UserSchema=new Schema({
    username: {type: String,unique: true},
    password:{type:String}
},{timestamps:true});
const User=connection.model('User',UserSchema);
module.exports=User;
```

4.密码加密

models/user.js

```
UserSchema.pre('save',function (next) {
    bcrypt.genSalt(10,(err,salt)=>{
        bcrypt.hash(this.password,salt,(err,hash) => {
            this.password=hash;
            next();
        });
    });
});
UserSchema.methods.comparePassword=function (password) {
    return bcrypt.compareSync(password,this.password);
}
```

routes/users.js

```
const express=require('express');
const User=require('../models/user');
const router=express.Router();
router.post('/signup',async function (req,res) {
    let user=new User(req.body);
    try {
        await user.save();
        res.json({
            code: 0,
            data: {
                id: user._id,
                username:user.username
            }
        });
    } catch (error) {
        res.json({
            code: 1,
            error
        });
    }

});
router.post('/signin',async (req,res) => {
    const {username,password}=req.body;
    try {
        const user=await User.findOne({username});
        if (user && user.comparePassword(password)) {
            res.json({
                code: 0,
                data: {
                    id: user._id,
                    username:user.username
                }
            });
        } else {
            res.status(403).json({
                code: 1,
                error:'用户名或密码错误'
            });
        }
    } catch (error) {
        res.status(403).json({
            code: 1,
            error
        });
    }
})
module.exports=router;
```

5.JWT认证

```
module.exports={
    PORT: 3000,
    DB_URL: "mongodb://localhost:27017/userSystem",
    SECRET:"zfpx"
}
```

```
let connection=mongoose.createConnection(DB_URL,{ useNewUrlParser: true })
```

```
const {sign,verify}=require('../utils/jwt');
router.post('/signin',async (req,res) => {
    const {username,password}=req.body;
    try {
        const user=await User.findOne({username});
        if (user && user.comparePassword(password)) {
            res.json({
                code: 0,
                data: {
                    token:sign({username:user.username})
                }
            });
        } else {
            res.status(403).json({
                code: 1,
                error:'用户名或密码错误'
            });
        }
    } catch (error) {
        res.status(403).json({
            code: 1,
            error
        });
    }
})
router.get('/signout',verify,(req,res) => {
    res.json({
        code: 0,
        data:'退出登录成功!'
    });
});
```

```
const jwt=require('jsonwebtoken');
const {SECRET}=require('../config');
const sign=user => {
    return jwt.sign(user,SECRET,{
        expiresIn:10
    });
}
const verify=(req,res,next) => {
    const token=req.headers.authorization;
    if (token) {
        jwt.verify(token,SECRET,(err,data)=>{
            if (err) {
                if (err.name == 'TokenExpiredError') {
                    return res.status(401).json({
                        code: 1,
                        error:'token已经过期!'
                    });
                } else {
                    return res.status(401).json({
                        code: 1,
                        error:'token认证失败!'
                    });
                }
            } else {
                next();
            }
        });
    } else {
        return res.status(401).json({
            code: 1,
            error:'请提供token'
        });
    }
}
module.exports={
    sign,
    verify
}
```

6.管理员发表文章管理

```
const articlesRouter=require('./routes/articles');
app.use('/articles',articlesRouter);
```

```
const mongoose=require('mongoose');
const connection=require('./index');
const Schema=mongoose.Schema;
const ArticleSchema=new Schema({
    title: {type: String},
    content: {type: String}
},{timestamps: true});

const Article=connection.model('Article',ArticleSchema);
module.exports=Article;
```

```
 const Schema=mongoose.Schema;
 const UserSchema=new Schema({
     username: {type: String,unique: true},
-    password:{type:String}
+    password: {type: String},
+    admin:{type:Boolean,default:false}
 },{timestamps: true});
 UserSchema.pre('save',function (next) {
     bcrypt.genSalt(10,(err,salt)=>{
```

```
const express=require('express');
const Article=require('../models/article');
const {verify} = require('../utils/jwt');
const router=express.Router();
router.post('/add',verify(true),async function (req,res) {
    const article=new Article(req.body);
    try {
        await article.save();
        res.json({
            code: 0,
            article
        });
    } catch (error) {
        res.status(500).json({code:1,error});
    }
});
router.get('/list',verify(),async function (req,res) {
    try {
        let articles=await Article.find();
        res.json({
            code: 0,
            articles
        });
    } catch (error) {
        res.status(500).json({code:1,error});
    }
});
module.exports=router;
```

```
                data: {
-                   token:sign({username:user.username})
+                   token:sign({username:user.username,admin:user.admin})
                }
```

```
const verify=(mustAdmin)=>(req,res,next) => {
    const token=req.headers.authorization;
    if (token) {
        jwt.verify(token,SECRET,(err,data)=>{
            if (err) {
                if (err.name == 'TokenExpiredError') {
                    return res.status(401).json({
                        code: 1,
                        error:'token已经过期!'
                    });
                } else {
                    return res.status(401).json({
                        code: 1,
                        error:'token认证失败!'
                    });
                }
            } else {
-               next();
+                if (mustAdmin) {
+                    let {admin}=data;
+                    if (admin) {
+                        next();
+                    } else {
+                        return res.status(401).json({
+                            code: 1,
+                            error:'必须是管理员才能进行此项操作!'
+                        });
+                    }
+                } else {
+                    next();
+                }

            }
        });
    } else {
        return res.status(401).json({
            code: 1,
            error:'请提供token'
        });
    }
}
```