

# 一、根据数据库说明建立数据库和表

## 二、工程编码

### A. 工程初始化及注册接口实现

#### 1. 创建清单

```
npm init -y
```

#### 2. 安装express

```
cnpm install express --save
```

#### 3. 编写工程入口文件 app.js

```
// 1. 引入express模块
const express = require('express')

// 2. 调用express函数得到app对象
const app = express()

// 测试路由
app.get('/', (req, res) => {
  res.send('ok')
})

// 3. 监听端口
app.listen(3008)

console.log('.....')
```

#### 4. 配置用户模块的路由（只处理登录和注册）

##### 4.1 新建router目录-> 新建user.js文件：

```
// 1. 引入express模块
const express = require('express')

// 2. 通过express的Router方法得到路由对象
const router = express.Router()

// 3. 配置路由项
// 3.1 注册接口
router.post('/reg', (req, res) => {
  res.send('reg user')
})

// 3.2 登录接口
```

```
router.post('/login', (req, res) => {
  res.send('login user')
})

// 暴露router
module.exports = router
```

## 4.2 向app.js中引入以上路由并注册

```
// 4. 引入userRouter路由
const userRouter = require('./router/user')
// 5. 注册userRouter路由并配置前缀
app.use('/api', userRouter)
```

## 4.3 通过postman测试以上接口是否正常

## 5. 安装mysql包

```
cnpm install mysql --save
```

## 6. 创建mysql的配置文件

db/db.js文件

```
// 1. 引入mysql模块
const mysql = require('mysql')
// 2. 创建连接
const db = mysql.createPool({
  host: 'localhost',
  username: 'root',
  password: 'mysql',
  database: 'mydb01'
})

// 3. 导出db对象
module.exports = db
```

## 7.创建user的路由对应的处理函数模块

router\_handle/user.js

```
// 1. 定义注册用户的处理函数
let regUser = (req, res) => {
  res.send('reg user')
}

// 3. 定义登录的处理函数
let login = (req, res) => {
  res.send('login user')
}

// 2. 导出处理函数
```

```
module.exports = {
  regUser,
  login
}
```

## 8. 修改原路由配置

router/user.js

```
// 3. 配置路由项
// 3.1 注册接口
router.post('/reg', userHandle.regUser) // 4.1 将对应的位置换成引入的处理函数

// 3.2 登录接口
router.post('/login', userHandle.login) // 4.2 将对应的位置换成引入的处理函数
```

## 9. 修改app.js，使其支持post请求中body数据的解析

```
// 6. 设置body中数据的接收
app.use(express.urlencoded({ extended: true }));
```

## 10. 创建工具文件

tools/tools.js

```
const crypto = require('crypto'); // crypto是node.js内置加密模块
function cryptPwd(password) {
  let md5 = crypto.createHash('md5');
  return md5.update(password).digest('hex');
}

module.exports = {
  cryptPwd
}
```

## 11. 实现注册的处理函数（一）

```
const {cryptPwd} = require('../tools/tools.js')
const db = require('../db/db.js')

// 1. 定义注册用户的处理函数
let regUser = (req, res) => {
  // 4. 获取用户名密码信息
  const user = req.body
  console.log(user)
  // 4.1 对密码进行加密
  const pwd = cryptPwd(user.password)
  console.log(pwd)
  // 5. 编写sql语句，执行
  const sql = 'insert into t_user(username, password) values(?,?)'
  db.query(sql, [user.username, pwd], (err, results) => {
    if(err) return res.send({status: 1, message: err})
    if(results.affectedRows !== 1) return res.send({status: 1, message: '注册失败'})
  })
}
```

```

        // 注册成功
        res.send({
            status: 0,
            message: '注册成功'
        })
    })
}

// 3. 定义登录的处理函数
let login = (req, res) => {
    res.send('login user')
}

// 2. 导出处理函数
module.exports = {
    regUser,
    login
}

```

## 12. 加入校验用户名是否已占用的功能

```

const { cryptPwd } = require('../tools/tools.js')
const db = require('../db/db.js')

// 1. 定义注册用户的处理函数
let regUser = (req, res) => {
    // 4. 获取用户名密码信息
    const user = req.body
    console.log(user)
    // 6. 检查用户名是否已占用
    const checkSql = 'select * from t_user where username=?'
    db.query(checkSql, user.username, (err, results) => {
        if (err) return res.send({ status: 1, message: err })
        if (results.length > 0) return res.send({ status: 1, message: '用户名已占
用' })
        // 用户名可用
        // 4.1 对密码进行加密
        const pwd = cryptPwd(user.password)
        console.log(pwd)
        // 5. 编写插入sql语句, 执行
        const sql = 'insert into t_user(username, password) values(?,?)'
        db.query(sql, [user.username, pwd], (err, results) => {
            if (err) return res.send({ status: 1, message: err })
            if (results.affectedRows !== 1) return res.send({ status: 1, message:
'注册失败' })
            // 注册成功
            res.send({
                status: 0,
                message: '注册成功'
            })
        })
    })
}

```

```
// 3. 定义登录的处理函数
let login = (req, res) => {
  res.send('login user')
}

// 2. 导出处理函数
module.exports = {
  regUser,
  login
}
```

## 13. 校验注册时传递的数据

### 13.1 安装两个包

```
cnpm install joi @escook/express-joi --save
```

### 13.2 定义注册时的校验规则

schema/user.js

```
const Joi = require('joi')

// 定义注册时的规则
const regUserSchema = {
  body: {
    username: Joi.string().pattern(/^[a-zA-Z0-9\u4e00-\u9fa5]{2,10}$/).required(),
    password: Joi.string().min(3).max(6).required()
  }
}

// 导出规则
module.exports = {
  regUserSchema
}
```

### 13.3 在router/user.js路由中局部注册express-joi中间件

```
// 1. 引入express模块
const express = require('express')

// 2. 通过express的Router方法得到路由对象
const router = express.Router()

// 4. 引入user对应的处理函数模块
const userHandle = require('../router_handle/user')
// 5. 导入express-joi中间件
const expressJoi = require('@escook/express-joi')
// 6. 导入所需的schema
const {regUserSchema} = require('../schema/user')
```

```
// 3. 配置路由项
// 3.1 注册接口
router.post('/reg', expressJoi(regUserSchema), userHandle.regUser) // 4.1 将对应
的位置换成引入的处理函数

// 3.2 登录接口
router.post('/login', userHandle.login) // 4.2 将对应的位置换成引入的处理函数

// 暴露router
module.exports = router
```

## 13.4 设置错误处理的全局中间件 app.js

```
// 6. 配置错误中间件
const Joi = require('joi')
app.use((err, req, res, next) => {
  // 6.1 Joi 参数校验失败
  if (err instanceof Joi.ValidationError) {
    return res.send({
      status: 1,
      message: err.message
    })
  }
  // 6.2 未知错误
  res.send({
    status: 1,
    message: err.message
  })
})
})
```

## B. 登录接口实现

### 1. 安装jsonwebtoken包

```
cnpm install jsonwebtoken --save
```

用于登陆成功后产生token (令牌)

### 2. 修改login方法, 实现登录功能

router\_handle/user.js

```
const { cryptPwd } = require('../tools/tools.js')
const db = require('../db/db.js')
const jwt = require('jsonwebtoken')

// 1. 定义注册用户的处理函数
let regUser = (req, res) => {
  // 4. 获取用户名密码信息
  const user = req.body
  console.log(user)
  // 6. 检查用户名是否已占用
  const checkSql = 'select * from t_user where username=?'
  db.query(checkSql, user.username, (err, results) => {
    if (err) return res.send({ status: 1, message: err })
  })
}
```

```

    if (results.length > 0) return res.send({ status: 1, message: '用户名已占
用' })
    // 用户名可用
    // 4.1 对密码进行加密
    const pwd = cryptPwd(user.password)
    console.log(pwd)
    // 5. 编写插入sql语句, 执行
    const sql = 'insert into t_user(username, password) values(?,?)'
    db.query(sql, [user.username, pwd], (err, results) => {
        if (err) return res.send({ status: 1, message: err })
        if (results.affectedRows !== 1) return res.send({ status: 1, message:
'注册失败' })
        // 注册成功
        res.send({
            status: 0,
            message: '注册成功'
        })
    })
})

}

// 3. 定义登录的处理函数
let login = (req, res) => {
    const user = req.body
    // 1) 编写sql语句
    const sql = 'select * from t_user where username=? and password=md5(?)'
    // 2) 执行sql
    db.query(sql, [user.username, user.password], (err, results) => {
        if (err) return res.send({ status: 1, message: err })
        if(results.length === 0) return res.send({status: 1, message: '用户名或密码
有误'})
        // 用户名密码都正确
        // 产生令牌
        console.log(results[0].id, results[0].username)
        const token = jwt.sign({id: results[0].id, username:
results[0].username}, 'love you', { expiresIn: '2h' })
        console.log(token)
        return res.send({
            status: 0,
            message: '登录成功',
            token: 'bearer ' + token
        })
    })
})

}

// 2. 导出处理函数
module.exports = {
    regUser,
    login
}

```

### 3. 编写登录时的数据校验规则

schema/user.js

自己完成

### 4. router/user.js路由中局部注册express-joi中间件实现对登录时的数据校验

自己完成

### 5. 安装express-jwt包

```
cnpm install express-jwt --save
```

这个包是一个负责解析token的中间件

### 6. 配置express-jwt中间件（在注册路由中间件之前）

app.js

```
// 7. 引入express-jwt对路由进行鉴权
const { expressjwt: jwt } = require("express-jwt");
app.use(jwt({
  secret: 'love you',
  algorithms: ['HS256'],
}).unless({ path: [/^\/api\/] })))
```

注意排除/api下的路由（其实就是登录和注册），因为这两个路由不需要身份验证就该能够访问。

### 7. 测试token

#### 7.1 添加一个/my下的路由

router/userinfo.js

(get: /my/userinfo)

```
// 1. 引入express模块
const express = require('express')

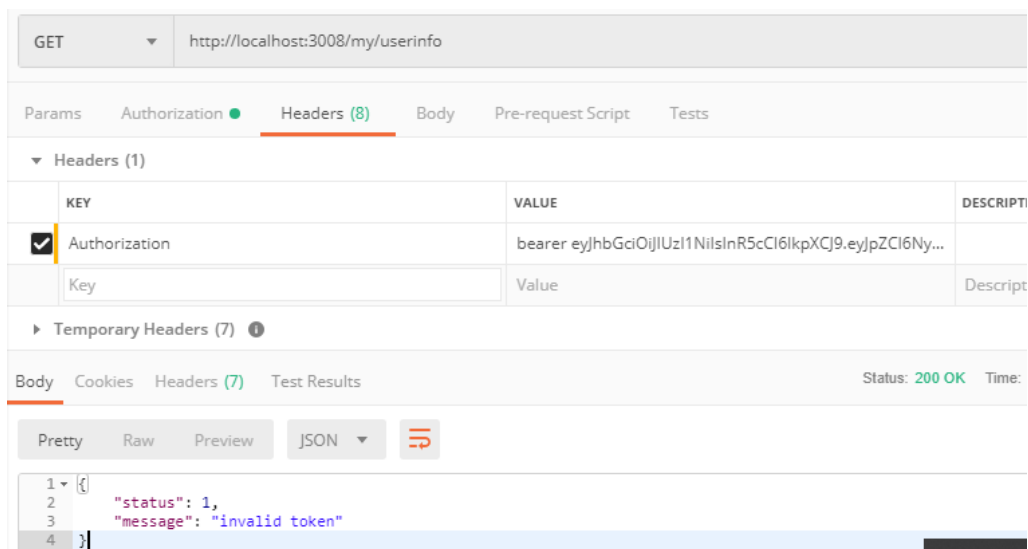
// 2. 通过express的Router方法得到路由对象
const router = express.Router()

router.get('/userinfo', (req, res) => {
  res.send('userinfo.....')
})

// 暴露router
module.exports = router
```



## 7.2 用postman分别访问reg, login, userinfo, 观察权限情况



## 三、个人中心接口开发

### A. 获取用户的基本信息

通过 `req.auth` 可以获取当前登录用户的信息（登录时存入token中的用户数据）

由express-jwt这个中间件自动将登录时存入token中的用户数据并添加一些其他信息放到`req.auth`中有了当前登录用户的信息（如id）就可以方便的查询出用户的基本信息了。

自行完成。

```
// 获取当前登录用户的个人信息
let getUserInfo = (req, res) => {
  // 得到当前登录用户的部分信息（如id）
  console.log(req.auth)
  res.send('。。。')
}

module.exports = {
  getUserInfo
}
```

### B.更新用户的基本信息

自行完成

### C. 重置密码

自行完成

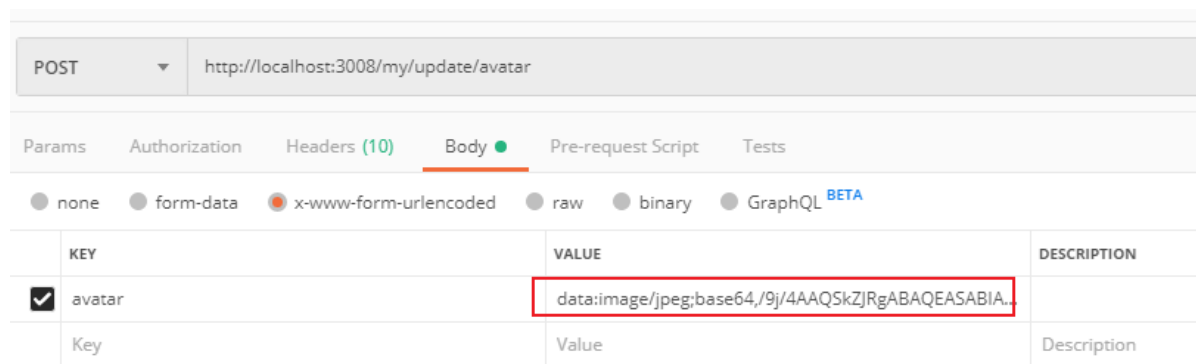
### D. 更换头像

```

router_handle/userinfo.js文件
// 设置头像
let updateAvatar = (req, res) => {
  const {id} = req.auth
  const {avatar} = req.body
  const sql = 'update t_user set headpic=? where id=?'
  db.query(sql, [avatar, id], (err, results) => {
    if (err) return res.send({ status: 1, message: err })
    if (results.affectedRows === 0) return res.send({ status: 1, message: '更新头像失败' })
    return res.send({ status: 0, message: '更新头像成功' })
  })
}

```

将本地一张小图（如200\*200）通过网站<http://www.json.cn/img2base64/>转成base64格式，在postman中测试：



## 四、文件分类接口

略

## 五、文章管理接口

### A. 发布文章

#### 关于上传文件的操作

1. 安装multer（看官方手册用法）
2. 配置相关路由  
router/article.js中  
post -> /add  
在这个文件中还进行了multer的配置

```

6
7  const multer = require('multer')
8
9  const storage = multer.diskStorage({
10    destination: function(req, file, cb) {
11      // cb(null, '/uploads/images')
12      cb(null, path.join(__dirname, '../uploads/images'))
13    },
14    filename: function(req, file, cb) {
15      console.log(file)
16      cb(null, file.originalname)
17    }
18  })
19
20  const upload = multer({storage})
21
22  const articleHandle = require('../router_handle/article')
23
24  router.post('/add', upload.single('cover_img'), articleHandle.addArticle)
25
26

```

### 3. 编写对应的处理函数

router\_handle/article.js

```

const db = require("../db/db")

let addArticle = (req, res) => {
  const articleInfo = {
    ...req.body, // 从req.body中拿到非文件型数据
    pub_date: new Date().toLocaleString(),
    author_id: req.auth.id, // 获取当前登录用户的id
    is_delete: 0,
    cover_img: 'uploads/images/' + req.file.filename // 上传后文件应该存储的路径
  }
  const sql = 'insert into t_article set ?' // 简化写法, 用于将多字段批量写入
  db.query(sql, articleInfo, (err, results) => {
    if (err) return res.send({ status: 1, message: err })
    if (results.affectedRows === 0) return res.send({ status: 1, message: '发布文章失败' })
    return res.send({ status: 0, message: '发布成功' })
  })
}

module.exports = {
  addArticle
}

```

### 4. 在postman中测试

POST http://localhost:3008/my/article/add

Params Authorization Headers (10) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary GraphQL BETA

KEY	VALUE	DESC
<input checked="" type="checkbox"/> cover_img	2457cc5414aabc69b42806059f857521.jpeg	文件型数据
<input checked="" type="checkbox"/> title	天气不错	非文件 型数据
<input checked="" type="checkbox"/> content	40度	
<input checked="" type="checkbox"/> cate_id	1	
<input checked="" type="checkbox"/> state	草稿	
Key	Value	Desc

Body Cookies Headers (7) Test Results

Status: 200 OK Time:

5. 检查数据库数据是否正常录入以及上传目录有没有资源存在

6. 通过浏览器检测 一下能不能独立访问服务器上的图片

1) 权限问题

配置jwt的排除目录

2) 把服务器上允许客户端直接的访问的资源配置成静态资源，可以不用配置路由就可访问

配置时可以指定虚拟目录

```
// 7. 引入express-jwt对路由进行鉴权
const { expressjwt: jwt } = require("express-jwt");
app.use(jwt({
  secret: 'love you',
  algorithms: ['HS256'],
}).unless({ path: [/^\/api\/$/, /^\/images\/$/] })))

// 配置静态资源（图片等）的访问
// 第一个参数表示客户端访问时的路径
app.use('/images/', express.static('uploads/images/'))
```

服务器端真实目录

## 关于上传后的文件名问题

```
const {v4: uuidv4} = require('uuid')
const storage = multer.diskStorage({
  destination: function(req, file, cb) {
    // cb(null, '/uploads/images')
    cb(null, path.join(__dirname, '../uploads/images'))
  },
  filename: function(req, file, cb) {
    // 生成随机名称
    const tmp = uuidv4().replaceAll('-', '')
    cb(null, tmp + path.extname(file.originalname))
  }
})

const upload = multer({storage})

const articleHandle = require('../router_handle/article')

router.post('/add', upload.single('cover_img'), articleHandle.addArticle)
```

uuid是一种通用唯一识别码，用于生成不会重复的字符串

调用uuidv4()生成唯一码，并去掉各部分之间的-，将作为图片的名称

得到扩展名

数据库存储结果：

id	title	content	cover_img
11	天气不错	40度	/uploads/images/6044bd6cb6d643e8b77a64cd64348491.jpeg
12	天气不错	40度	/uploads/images/bc5e99484b5b41f7957c9b7bfe941c4d.jpeg