# 一 项目初步搭建

## 1.1 项目目录



## 1.2 环境

**"dependencies"**: {  
 **"body-parser"**: **"^1.18.2"**,  
 **"cookies"**: **"^0.7.1"**,  
 **"express"**: **"^4.16.2"**,  
 **"markdown"**: **"^0.5.0"**,  
 **"mongoose"**: **"^4.13.9"**,  
 **"swig"**: **"^1.4.2"**}

## 1.3 express项目启动

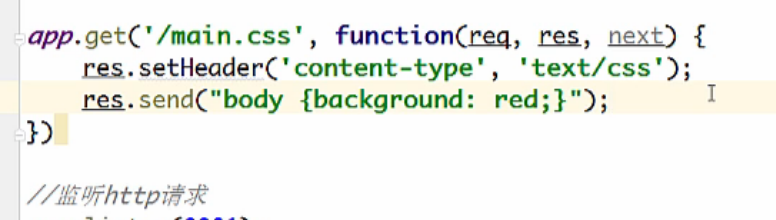


## 1.4 配置模板引擎

*//定义模板引擎，第一个参数是模板引擎名称也是模板文件后缀，第二个参数是解析方式*app.engine('html',swig.renderFile);  
*//注册模板引擎，第一恶参数固定，第二个参数和app.engin中的名称一样*app.set('view engine','html');  
*//swig模板为了提升性能有缓存机制，所以开发时修改html页面无法再加载，那么需要取消限制*swig.setDefaults({cache:false});  
  
app.get('/',(req,res,next)=>{  
 *// res.send('Hi');  
 //读取指定文件，并解析，返回给客户端  
 //参数一：相对于views目录下的html文件 参数二：传递给模板使用的数据* res.render('index')  
});

## 1.5 静态文件管理

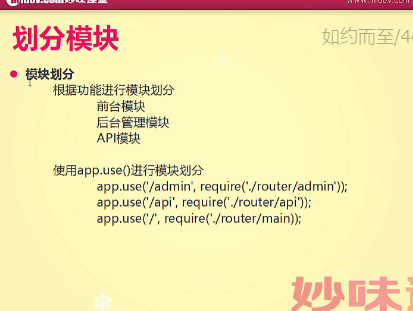
静态文件的请求写起来非常麻烦，如下所示



这样的请求过于复杂，express提供了新的静态文件管理中间件：

*//静态资源管理：当用户访问以/public开始，直接返回对应\_\_dirname+...*app.use('/public',express.static(\_\_dirname + '/public'));

## 1.6 模块划分



在不同模块下分别写下不同的业务逻辑，如下案例：

let express = require('express');  
let router = express.Router();  
router.get('/user',(req,res,next)=>{ *//这里的user其实是/admin/user* res.send('user');  
});  
module.exports = router;

## 1.7 MongoDB

### 1.7.1启动MongoDB

安装后进入bin文件夹： C:\ProgramFiles\MongoDB\Server\3.4\bin>

执行下列命令： mongod --dbpath=N:\blogdb --port=27018

### 1.7.2 app连接MongoDB

4.11版本前连接方式：

mongoose.connect('mongodb://localhost:27018/ruyueshuodb',err=>{  
 if(err){  
 console.log('数据库连接失败');  
 }else {  
 console.log('数据库连接成功');  
 app.listen(8081);  
 }  
});

4.11版本以及后连接方式

*//必须额外加上这句，否则有警告*

mongoose.Promise = global.Promise;

let promiseDB = mongoose.connect('mongodb://localhost:27018/ruyueshuodb',{  
 useMongoClient: true,  
});  
promiseDB.then(function (db) {  
 if(db){  
 console.log('数据库连接成功');  
 app.listen(8081);  
 }else {  
 console.log('数据库连接失败');  
 }  
});

### 1.7.3 数据模型

先创建数据，在schemas文件下创建用户数据文件：userT.js

let mongoose = require('mongoose');  
  
module.exports = new mongoose.Schema({  
 username: String, *//用户名* password: String, *//密码* isAdmin: { *//是否是管理员* type: Boolean,  
 default: false  
 }  
});

在模型文件夹下创建模型userM.js：

let mongoose = require('mongoose');  
let userSchema = require('../schemas/userT');  
module.exports = mongoose.model('userM',userTSchema);

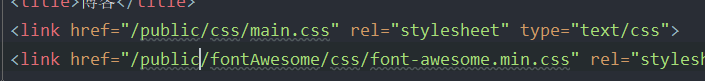
# 二 首页与用户注册

## 2.0 更改渲染界面

将首页静态文件放置在views文件夹下，更改路由与渲染：

let express = require('express');  
let router = express.Router();  
router.get('/',(req,res,next)=>{  
 *// res.send('main');* res.render('main/index');  
});  
module.exports = router;

对前台静态文件路径进行一系列调整：如静态文件引入、登录注册的显示隐藏问题



处理前台界面展示问题：

let $loginBox = $('#loginBox');  
let $registerBox = $('#registerBox');  
*//点击切换注册/登录 显示与隐藏*$loginBox.find('a').on('click',function () {  
 $loginBox.hide();  
 $registerBox.show();  
});  
$registerBox.find('a').on('click',function () {  
 $loginBox.show();  
 $registerBox.hide();  
});

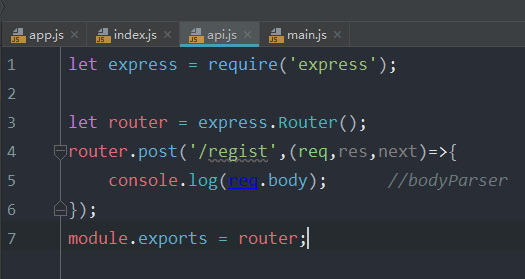
## 2.1 注册请求发送与接收

*//注册*$registerBox.find('button').on('click',function () {  
 $.ajax({  
 type: 'post',  
 url: 'http://localhost:8081/api/regist',  
 data: {  
 username: $registerBox.find('[name="username"]').val(),  
 password: $registerBox.find('[name="password"]').val(),  
 repassword: $registerBox.find('[name="repassword"]').val()  
 },  
 dataType: 'json',  
 success: function (result) {  
 console.log(result);  
 }  
 });

这时候我们需要在入口文件配置bodyparse来处理post请求发送的数据：

*//配置bodyparser*app.use(bodyParse.urlencoded({extended: true}));

后台接收测试：



## 2.2 后台处理注册post请求

let express = require('express');  
let UserM = require('../models/userM');  
  
let router = express.Router();  
let responseData;  
router.use(function (req,res,next) {  
 responseData = {  
 code: 0,  
 msg: '恭喜您，注册成功！'  
 };  
 next();  
});  
  
router.post('/regist',(req,res,next)=>{  
 *// console.log(req.body); //bodyParser* /\*  
 用户名不能为空 密码不能为空 两次输入密码必须一致 用户是否已经注册  
 \*/  
 let username = req.body.username;  
 let password = req.body.password;  
 let repassword = req.body.repassword;  
  
 if(username == '' || password == ''){  
 responseData.code = 1;  
 responseData.msg = '用户名或密码不能为空';  
 res.json(responseData);  
 return;  
 }  
 if(password != repassword){  
 responseData.code = 2;  
 responseData.msg = '密码输入不一致';  
 res.json(responseData);  
 return;  
 }  
 UserM.findOne({  
 username: username  
 }).then(function (info) {  
 if(info) {  
 responseData.code = 3;  
 responseData.msg = '用户名已存在';  
 res.json(responseData);  
 return;  
 }  
 let user = new UserM({  
 username: username,  
 password: password  
 });  
 return user.save();  
 }).then(function (newInfo) {  
 console.log('新用户注册：' + username);  
 res.json(responseData);  
 });  
});  
module.exports = router;

## 2.3 前台接收信息展示修改

success: function (result) {  
 $registerBox.find('.colWarning').html(result.msg);  
 if(!result.code){ *//注册成功* setTimeout(function () {  
 $loginBox.show();  
 $registerBox.hide();  
 },1000);  
 }  
}

# 三 登录

## 3.1 登录基本实现

*//登录*$loginBox.find('button').on('click',function () {  
 $.ajax({  
 type: 'post',  
 url: 'http://localhost:8081/api/login',  
 data: {  
 username: $loginBox.find('[name="username"]').val(),  
 password: $loginBox.find('[name="password"]').val()  
 },  
 dataType: 'json',  
 success: function (result) {  
 console.log(result);  
 $loginBox.find('.colWarning').html(result.msg);  
 if(result.code == 5){ *//注册成功* setTimeout(function () {  
 $loginBox.hide();  
 $registerBox.hide();  
 $userInfoBox.show();  
 $userInfoBox.find('.username').html(result.userInfo.username);  
  
 },1000);  
 }  
 }  
 });  
});

登录的后台实现：

*//登录*router.post('/login',function (req,res,next) {  
 let username = req.body.username;  
 let password = req.body.password;  
 if(username == '' || password == ''){  
 responseData.code = 1;  
 responseData.msg = '用户名或者密码不能为空';  
 res.json(responseData);  
 return;  
 }  
 UserM.findOne({  
 username: username,  
 password: password  
 }).then(function (info) {  
 if(!info){  
 responseData.code = 4;  
 responseData.msg = '用户名或密码错误';  
 res.json(responseData);  
 return;  
 }  
 responseData.code = 5;  
 responseData.msg = '登录成功';  
 responseData.userInfo = {  
 username: info.username,  
 id: info.\_id  
 };  
 res.json(responseData);  
 return;  
 });

## 3.2 登录的cookie

### 3.2.1 cookie的配置

*//cookie中间件 全局可以访问*app.use(function (req,res,next) {  
 req.cookies = new Cookies(req,res);  
 *//定义一个全局对象，让浏览器能够随处访问到cookie* req.userInfo = {};  
 if(req.cookies.get('userInfo')){  
 try{  
 req.userInfo = JSON.parse(req.cookies.get('userInfo'));  
 }catch (e){  
  
 }  
 }  
 next();  
});

### 3.2.2 登录成功后设置cookie

req.cookies.set('userInfo',JSON.stringify({  
 \_id: info.\_id,  
 username: info.username,  
 password: info.password  
}));

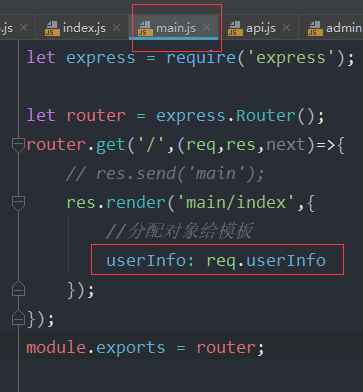
### 3.2.3 登录成功的前台代码

success: function (result) {  
 console.log(result);  
 $loginBox.find('.colWarning').html(result.msg);  
 if(result.code == 5){ *//登录成功  
 // setTimeout(function () {  
 // $loginBox.hide();  
 // $registerBox.hide();  
 // $userInfoBox.show();  
 // $userInfoBox.find('.username').html(result.userInfo.username);  
 //  
 // },1000);  
 //点击登录，一旦成功服务器就会返回cookie信息，浏览器获得cookie信息，只用刷新页面即可* window.location.reload();  
 *//但是由于更改了模板的渲染，需要在模板中设置动态数据信息* }  
}

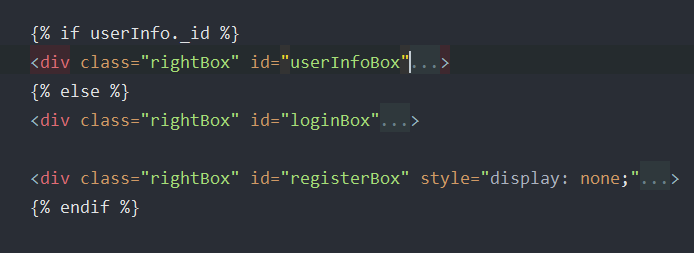
我们不再需要之前的代码逻辑，新的登录成功后，后台都会返回cookie给前台，前台收到cookie后，使用cookie内的信息进行页面的渲染、改变即可。

所以我们只需要刷新页面。

但是此时前台的渲染就需要更改，首先我们需要在后台传递给前台一个渲染数据：



这个req.userInfo本身是个自定义的空对象，但是一旦有了cookie，就会将cookie内的数据赋值给这个对象，前台模板通过该对象进行取值。





## 3.3 退出

有了cookie后，新的退出方式：

*//退出*$('#logoutBtn').on('click',function () {  
 $.ajax({  
 url: 'http://localhost:8081/api/logout',  
 success: function (result) {  
 if(result.code == 6){  
 window.location.reload();  
 }  
 }  
 });  
});

退出后台：

*//退出*router.get('/logout',function (req,res) {  
 req.cookies.set('userInfo',null);  
 responseData.code = 6;  
 res.json(responseData);  
});

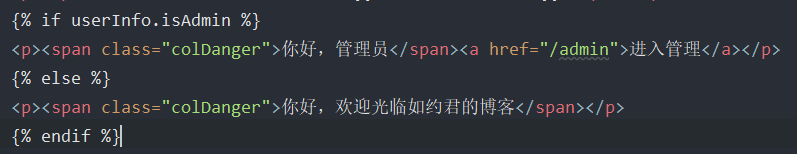
# 四 普通用户与管理员用户

是否是管理员用户，需要实时验证，不能使用cookie。这里我们手动在数据库中添加一个管理员用户，即 isAdmin字段为true。

首先需要将是否是管理员字段的值保存：

*//cookie中间件*app.use(function (req,res,next) {  
 req.cookies = new Cookies(req,res);  
 *//定义一个全局对象，让浏览器能够随处访问到cookie* req.userInfo = {};  
 if(req.cookies.get('userInfo')){  
 try{  
 req.userInfo = JSON.parse(req.cookies.get('userInfo'));  
 *//判断用户是否是管理员* UserM.findById(req.userInfo.\_id).then(function (userInfo) {  
 req.userInfo.isAdmin = Boolean(userInfo.isAdmin);  
 next();  
 })  
 }catch (e){  
 next();  
 }  
 }else {  
 next();  
 }  
});

前台展示情况修改：



最后书写管理员页面控制：

router.use(function (req,res,next) {  
 if(!req.userInfo.isAdmin){  
 res.send('对不起，只有管理员才可以进入管理页面');  
 return;  
 }  
 next();  
});  
  
router.get('/',(req,res,next)=>{  
 res.send('后台首页');  
});