记账本项目添加会话控制

# 我们之前给记账本程序添加了api接口功能，但是这个程序现在是非常不安全，因为没有权限控制，没有对我们的数据做任何保护，任何人都可以新增删除数据，我们现在来完善功能，添加会话控制

# 先把上一节课的项目复制一份，改名accounts-mongoose-resful-session-control，然后做下面的处理

# 一、添加用户注册功能

## 1.需要安装express-session库和connect-mongo库

|  |
| --- |
|  |

## 2.我们在views文件夹里面新建一个reg.ejs,内容如下

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>用户注册</title>      <script src="https://www.itxst.com/package/jquery-3.5.1/jquery.min.js"></script>      <script src="https://www.itxst.com/package/bootstrap-4.5.0/js/bootstrap.min.js"></script>      <link href="https://www.itxst.com/package/bootstrap-4.5.0/css/bootstrap.css" rel="stylesheet">      <script src="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/js/bootstrap-datepicker.min.js"></script>      <script src="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/locales/bootstrap-datepicker.zh-CN.min.js"></script>      <link href="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/css/bootstrap-datepicker.min.css" rel="stylesheet">  </head>  <body>      <div class="container">          <div class="row">              <div class=" col-xs-12 col-lg-8 col-lg-offset-2">                  <h2>用户注册</h2>          <hr>              <form action="/account" method="post">                  <div class="form-group">                      <label for="username">用户名</label>  <br/>                      <input type="text" name="username" class="form-control">                  </div>                  <div class="form-group">                      <label for="password">密码</label> <br/>                      <input  class="form-control"  name="password">                  </div>                  <div class="form-group">                      <button type="submit" class="btn btn-primary btn-block">用户注册</button>                  </div>                  </form>                </div>          </div>      </div>            <!-- <script>          $("#inputDate").datepicker({              language: 'zh-CN', //语言              autoclose: true, //选择后自动关闭              clearBtn: true,//清除按钮              format: "yyyy-mm-dd"//日期格式          });      </script> -->  </body>  </html> |

## 3.把routes/web/index.js复制一份，改名auth.js,然后只保留框架把逻辑代码删除，添加一条路由规则

|  |
| --- |
| var express = require('express');  var router = express.Router();  router.get('/reg',(req,res)=>{    res.render('reg')  })  module.exports = router; |

## 4.需要在app.js里面全局注册这个路由

|  |
| --- |
| var createError = require('http-errors');  var express = require('express');  var path = require('path');  var cookieParser = require('cookie-parser');  var logger = require('morgan');  var indexRouter = require('./routes/web/index');  //导入account.js接口路由文件  let accountRouter = require('./routes/api/account')  //导入注册路由文件  let authRouter = require('./routes/web/auth')  var app = express();  // view engine setup  app.set('views', path.join(\_\_dirname, 'views'));  app.set('view engine', 'ejs');  app.use(logger('dev'));  app.use(express.json());  app.use(express.urlencoded({ extended: false }));  app.use(cookieParser());  app.use(express.static(path.join(\_\_dirname, 'public')));  app.use('/', indexRouter);  app.use('/', authRouter);  //注册api路由规则  app.use('/api', accountRouter);  // catch 404 and forward to error handler  app.use(function(req, res, next) {    next(createError(404));  });  // error handler  app.use(function(err, req, res, next) {    // set locals, only providing error in development    res.locals.message = err.message;    res.locals.error = req.app.get('env') === 'development' ? err : {};    // render the error page    res.status(err.status || 500);    res.render('error');  });  module.exports = app; |

### 测试一下，是可以匹配到的

|  |
| --- |
|  |

## 5.为了方便以后查找文件，我们在views文件夹下面新建一个auth文件夹，把reg.ejs移动到里面，此时需要调整一下引用代码

|  |
| --- |
|  |

### 测试一下，是可以匹配到的

|  |
| --- |
|  |

## 6.把views/auth/reg.ejs完善一下，代码如下

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>用户注册</title>      <script src="https://www.itxst.com/package/jquery-3.5.1/jquery.min.js"></script>      <script src="https://www.itxst.com/package/bootstrap-4.5.0/js/bootstrap.min.js"></script>      <link href="https://www.itxst.com/package/bootstrap-4.5.0/css/bootstrap.css" rel="stylesheet">      <script src="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/js/bootstrap-datepicker.min.js"></script>      <script src="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/locales/bootstrap-datepicker.zh-CN.min.js"></script>      <link href="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/css/bootstrap-datepicker.min.css" rel="stylesheet">  </head>  <body>      <div class="container">          <div class="row">              <div class=" col-xs-12 col-lg-8 col-lg-offset-2">                  <h2>用户注册</h2>          <hr>              <form action="/reg" method="post">                  <div class="form-group">                      <label for="username">用户名</label>  <br/>                      <input type="text" name="username" id="username" class="form-control">                  </div>                  <div class="form-group">                      <label for="password">密码</label> <br/>                      <input  class="form-control" id="password" name="password">                  </div>                  <div class="form-group">                      <button type="submit" class="btn btn-primary btn-block">用户注册</button>                  </div>                  </form>                </div>          </div>      </div>            <!-- <script>          $("#inputDate").datepicker({              language: 'zh-CN', //语言              autoclose: true, //选择后自动关闭              clearBtn: true,//清除按钮              format: "yyyy-mm-dd"//日期格式          });      </script> -->  </body>  </html> |

## 7.在routes/web/auth.js里面添加post方法的/reg路由处理，我们先在控制台输出看看能否获取到表单数据

|  |
| --- |
| var express = require('express');  var router = express.Router();  //用户注册表单页面路由  router.get('/reg',(req,res)=>{    res.render('auth/reg')  })  //用户注册数据处理路由  router.post('/reg',(req,res)=>{     //获取表单数据     console.log(req.body);    res.send("ok")  })  module.exports = router; |

### 测试一下，是可以拿到数据的

|  |
| --- |
|  |

## 8.我们需要一个用户数据模型对应users集合，我们在model文件夹里面新建一个userModel.js文件，内容如下

### model/userModel.js

|  |
| --- |
| let mongoose = require('mongoose')  let userSchema = new mongoose.Schema({      username:{      type:String,      required:true,      unique:true      },     password:{      type:String,      required:true,      }  })  //创建模型对象,bookModel就是对应books集合，她可以完成数据的增删改查  let userModel = mongoose.model("users",userSchema)  //导出模型对象  module.exports = userModel |

## 9.完善注册的post请求，代码如下

### routes/web/auth.js

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入用户模型文件  let userModel = require('../../model/userModel')  //用户注册表单页面路由  router.get('/reg',(req,res)=>{    res.render('auth/reg')  })  //用户注册数据处理路由  router.post('/reg',(req,res)=>{    //表单验证     //获取表单数据    //  console.log(req.body);      userModel.create({       ...req.body    }).then(data=>{      res.render('success',{msg:'注册成功',url:'/login'})    }).catch(err=>{      res.send("注册失败，请稍后再试")      return    })  })  module.exports = router; |

### 测试一下，发现可以保存数据到数据库了

|  |
| --- |
|  |

## 10.其实我们需要对用户的密码进行加密，需要一个md5库，使用npm i md5安装

|  |
| --- |
|  |

## 11.安装完成后我们在auth.js里面导入使用，在保存数据之前对用户的密码进行md5加密

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入md5加密库用来加密用户密码  let md5 = require('md5')  //导入用户模型文件  let userModel = require('../../model/userModel')  //用户注册表单页面路由  router.get('/reg',(req,res)=>{    res.render('auth/reg')  })  //用户注册数据处理路由  router.post('/reg',(req,res)=>{    //表单验证     //获取表单数据    //  console.log(req.body);      userModel.create({       ...req.body,       password:md5(req.body.password)    }).then(data=>{      res.render('success',{msg:'注册成功',url:'/login'})    }).catch(err=>{      res.send("注册失败，请稍后再试")      return    })  })  module.exports = router; |

### 测试一下，

|  |
| --- |
|  |
|  |

### 查看一下数据库

|  |
| --- |
|  |

#### 发现密码被加密了

### 需要注意，既然我们在用户注册的时候对密码做了不可逆的md5加密，那么当用户登录的时候我们也需要对用户密码先进行md5加密，然后再和数据库里面的密码比较否则用户可能无法登录，即使他的密码是正确的

# 二、添加用户登录功能

## 1.在auth.js里面添加一个get路由用来渲染登录页面

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入md5加密库用来加密用户密码  let md5 = require('md5')  //导入用户模型文件  let userModel = require('../../model/userModel')  //用户注册表单页面路由  router.get('/reg',(req,res)=>{    res.render('auth/reg')  })  //用户注册数据处理路由  router.post('/reg',(req,res)=>{    //表单验证     //获取表单数据    //  console.log(req.body);      userModel.create({       ...req.body,       password:md5(req.body.password)    }).then(data=>{      res.render('success',{msg:'注册成功',url:'/login'})    }).catch(err=>{      res.send("注册失败，请稍后再试")      return    })  })  //用户登录页面路由  router.get('/login',(req,res)=>{    res.render('auth/login')  })  module.exports = router; |

## 2.在views/auth文件夹里面把reg.ejs复制一份改名login.ejs,修改一下路由和标题

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>用户登录</title>      <script src="https://www.itxst.com/package/jquery-3.5.1/jquery.min.js"></script>      <script src="https://www.itxst.com/package/bootstrap-4.5.0/js/bootstrap.min.js"></script>      <link href="https://www.itxst.com/package/bootstrap-4.5.0/css/bootstrap.css" rel="stylesheet">      <script src="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/js/bootstrap-datepicker.min.js"></script>      <script src="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/locales/bootstrap-datepicker.zh-CN.min.js"></script>      <link href="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/css/bootstrap-datepicker.min.css" rel="stylesheet">  </head>  <body>      <div class="container">          <div class="row">              <div class=" col-xs-12 col-lg-8 col-lg-offset-2">                  <h2>用户登录</h2>          <hr>              <form action="/login" method="post">                  <div class="form-group">                      <label for="username">用户名</label>  <br/>                      <input type="text" name="username" id="username" class="form-control">                  </div>                  <div class="form-group">                      <label for="password">密码</label> <br/>                      <input  class="form-control" id="password" name="password">                  </div>                  <div class="form-group">                      <button type="submit" class="btn btn-primary btn-block">登录</button>                  </div>                  </form>                </div>          </div>      </div>            <!-- <script>          $("#inputDate").datepicker({              language: 'zh-CN', //语言              autoclose: true, //选择后自动关闭              clearBtn: true,//清除按钮              format: "yyyy-mm-dd"//日期格式          });      </script> -->  </body>  </html> |

## 3.在auth.js文件里面添加对登录数据进行处理的post路由

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入md5加密库用来加密用户密码  let md5 = require('md5')  //导入用户模型文件  let userModel = require('../../model/userModel')  //用户注册表单页面路由  router.get('/reg',(req,res)=>{    res.render('auth/reg')  })  //用户注册数据处理路由  router.post('/reg',(req,res)=>{    //表单验证,以后慢慢完善，如不允许有两个名字相同的用户     //获取表单数据    //  console.log(req.body);      userModel.create({       ...req.body,       password:md5(req.body.password)    }).then(data=>{      res.render('success',{msg:'注册成功',url:'/login'})    }).catch(err=>{      res.status(500).send("注册失败，请稍后再试")      return    })  })  //用户登录页面路由  router.get('/login',(req,res)=>{    res.render('auth/login')  })  //登录处理路由  router.post('/login',(req,res)=>{    //获取用户名和密码    let {username,password} = req.body    let pwd = md5(password) //对密码进行md5加密    //根据用户输入的信息到数据库里面查询改用户    userModel.findOne({username:username,password:pwd})       .then(data=>{        // console.log(data);         //需要判断如果没有查询到用户data是null，说明可能是用户名或者密码错误         if(!data){           return res.send("用户名或者密码错误！！！")          }          res.render('success',{msg:"登录成功",url:"/account"})         }).catch(err=>{        res.status(500).send("登录失败，请稍后再试")        return      })    // res.render('auth/login')  })  module.exports = router; |

# 三，写入session

## 用户登录成功后服务器需要把用户信息写入session并且把session id以cookie的形式发给浏览器，我们来实现这个功能

## 1.需要安装express-session和connect-mongo两个包，然后需要在app.js里面注册他们，我们来注册一下

|  |
| --- |
| var createError = require('http-errors');  var express = require('express');  var path = require('path');  var cookieParser = require('cookie-parser');  var logger = require('morgan');  var indexRouter = require('./routes/web/index');  //导入account.js接口路由文件  let accountRouter = require('./routes/api/account')  //导入注册路由文件  let authRouter = require('./routes/web/auth')  //导入express-session  let session = require('express-session')  //导入connect-mongo  let MongoStore = require('connect-mongo')  var app = express();  // view engine setup  app.set('views', path.join(\_\_dirname, 'views'));  app.set('view engine', 'ejs');  app.use(logger('dev'));  app.use(express.json());  app.use(express.urlencoded({ extended: false }));  app.use(cookieParser());  app.use(express.static(path.join(\_\_dirname, 'public')));  let {DBHOST,DBPORT,DBNAME} = require('./config/config')  let dbUrl = `mongodb://${DBHOST}:${DBPORT}/${DBNAME}`  //设置并且全局注册express-session中间件  app.use(session({    name:'sid',//设置cookie的name，默认是connect.sid    secret:"atguigu",//参与加密的字符串，又叫做签名    saveUninitialized:false,//是否为每一次请求都设置一个cookie用来存储session的id    resave:true,//是否在每一次请求时重新保存session    store:MongoStore.create({        mongoUrl:dbUrl //数据库连接配置    }),    cookie:{        httpOnly:true, //开启后，前端无法通过js操作        maxAge:1000\*60\*60\*24\*7 //控制session id的过期时间和cookie的过期时间    }  }))  app.use('/', indexRouter);  app.use('/', authRouter);  //注册api路由规则  app.use('/api', accountRouter);  // catch 404 and forward to error handler  app.use(function(req, res, next) {    next(createError(404));  });  // error handler  app.use(function(err, req, res, next) {    // set locals, only providing error in development    res.locals.message = err.message;    res.locals.error = req.app.get('env') === 'development' ? err : {};    // render the error page    res.status(err.status || 500);    res.render('error');  });  module.exports = app; |

## 2.在auth.js里面用户登录成功的代码里面添加保存session的代码

### routes/web/auth.js

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入md5加密库用来加密用户密码  let md5 = require('md5')  //导入用户模型文件  let userModel = require('../../model/userModel')  //用户注册表单页面路由  router.get('/reg',(req,res)=>{    res.render('auth/reg')  })  //用户注册数据处理路由  router.post('/reg',(req,res)=>{    //表单验证,以后慢慢完善，如不允许有两个名字相同的用户     //获取表单数据    //  console.log(req.body);      userModel.create({       ...req.body,       password:md5(req.body.password)    }).then(data=>{      res.render('success',{msg:'注册成功',url:'/login'})    }).catch(err=>{      res.status(500).send("注册失败，请稍后再试")      return    })  })  //用户登录页面路由  router.get('/login',(req,res)=>{    res.render('auth/login')  })  //登录处理路由  router.post('/login',(req,res)=>{    //获取用户名和密码    let {username,password} = req.body    let pwd = md5(password) //对密码进行md5加密    //根据用户输入的信息到数据库里面查询改用户    userModel.findOne({username:username,password:pwd})       .then(data=>{        // console.log(data);         //需要判断如果没有查询到用户data是null，说明可能是用户名或者密码错误         if(!data){           return res.send("用户名或者密码错误！！！")          }          //写入session          req.session.username = data.username          req.session.\_is = data.\_id          res.render('success',{msg:"登录成功",url:"/account"})         }).catch(err=>{        res.status(500).send("登录失败，请稍后再试")        return      })    // res.render('auth/login')  })  module.exports = router; |

### 测试一下，成功

|  |
| --- |
|  |
|  |
|  |

# 四、用户登录检测

## 虽然我们已经做了注册登录以及session写入的功能，但是这些功能并没有响应到获取账单列表新建账单和删除账单的功能我们没有登录仍然可以查看新增和删除，这是不对的，我们先来做登录检测功能

## 1.我们先对获取账单列表这个路由进行完善，打开index.js，先定义一个中间件函数，用来检测用户是否登录，如果是，就调用next函数放行，如果没有登录，就重定向到登录页面

|  |
| --- |
| var express = require('express');  var router = express.Router();  //引入模型  let accountModel = require('../../model/accountModel')  //导入moment库  let moment = require('moment')  //定义一个检测用户登录的中间件  let checkLoginMiddleware = (req,res,next)=>{      //判断用户是否登录，如果没有就把引导到登录页面    if(!req.session.username){      return res.redirect('/login')    }    //如果用户登录了，就放行    next()  }  /\* GET home page. \*/  // router.get('/', function(req, res, next) {  //   res.render('index', { title: 'Express' });  // });  //显示账本列表的路由，从数据库读取所有文档数据渲染到页面,需要添加用户登录检测功能  router.get('/account', function(req, res, next) {      //获取所有账单信息,按时间倒序排列    accountModel.find().sort({time:-1}).then(data=>{      res.render('list',{data:data,moment:moment});    }).catch(err=>{      res.status(500).send("读取数据失败")      return    })    });  //添加记录的路由  router.get('/account/create', function(req, res, next) {    res.render('create');  });  //接收表单数据的路由  router.post('/account', function(req, res) {    //express的脚手架帮我们添加了解析表单数据的中间件功能，所以这里我们可以直接使用req.body    // console.log(req.body);    //用mongodb保存数据    accountModel.create({        ...req.body,        time:moment(req.body.time).toDate()      })     .then(data=>{       console.log(data);       let msg = "数据添加成功"       let url = '/account'       res.render('success',{msg,url})     }).catch(err=>{       console.log(err)       res.status(500).send("插入数据失败")       return     })    });  //删除记录的路由  router.get('/account/:id', function(req, res, next) {    let id = req.params.id    // console.log(id)    //删除记录    accountModel.deleteOne({\_id:id}).then(data=>{        console.log(data);        let msg = "数据删除成功"        let url = '/account'        res.render('success',{msg,url}) //复用上面的success页面    }).catch(err=>{      console.log(err)       res.status(500).send("删除数据失败")       return    })    });  module.exports = router; |

## 2把这个中间件放到所有需要验证登录的路由函数的第二个参数设置

### index.js

|  |
| --- |
| var express = require('express');  var router = express.Router();  //引入模型  let accountModel = require('../../model/accountModel')  //导入moment库  let moment = require('moment')  //定义一个检测用户登录的中间件  let checkLoginMiddleware = (req,res,next)=>{      //判断用户是否登录，如果没有就把引导到登录页面    if(!req.session.username){      return res.redirect('/login')    }    //如果用户登录了，就放行    next()  }  /\* GET home page. \*/  // router.get('/', function(req, res, next) {  //   res.render('index', { title: 'Express' });  // });  //显示账本列表的路由，从数据库读取所有文档数据渲染到页面,需要添加用户登录检测功能  router.get('/account',checkLoginMiddleware, function(req, res, next) {      //获取所有账单信息,按时间倒序排列    accountModel.find().sort({time:-1}).then(data=>{      res.render('list',{data:data,moment:moment});    }).catch(err=>{      res.status(500).send("读取数据失败")      return    })    });  //添加记录的路由  router.get('/account/create',checkLoginMiddleware, function(req, res, next) {    res.render('create');  });  //接收表单数据的路由  router.post('/account',checkLoginMiddleware, function(req, res) {    //express的脚手架帮我们添加了解析表单数据的中间件功能，所以这里我们可以直接使用req.body    // console.log(req.body);    //用mongodb保存数据    accountModel.create({        ...req.body,        time:moment(req.body.time).toDate()      })     .then(data=>{       console.log(data);       let msg = "数据添加成功"       let url = '/account'       res.render('success',{msg,url})     }).catch(err=>{       console.log(err)       res.status(500).send("插入数据失败")       return     })    });  //删除记录的路由  router.get('/account/:id',checkLoginMiddleware ,function(req, res, next) {    let id = req.params.id    // console.log(id)    //删除记录    accountModel.deleteOne({\_id:id}).then(data=>{        console.log(data);        let msg = "数据删除成功"        let url = '/account'        res.render('success',{msg,url}) //复用上面的success页面    }).catch(err=>{      console.log(err)       res.status(500).send("删除数据失败")       return    })    });  module.exports = router; |

### 测试一下，我们把数据库里面是sessions集合删除，然后在浏览器里面输入/account,或者/account/create就会重定向到login

|  |
| --- |
|  |

### 然后我们输入用户名密码点击登录

|  |
| --- |
|  |

### 就会跳转到登录成功页面，

|  |
| --- |
|  |

### 点击“点击跳转”就会跳转到指定列表页面

|  |
| --- |
|  |

## 3.功能是完成了但是我们需要把这个中间件单独抽取出来放到一个文件中，方便以后复用，我们在项目根目录下面新建一个middle文件夹，然后新建一个checkLoginMiddleware.js文件

|  |
| --- |
|  |

## 4.在index.js中导入这个中间件

|  |
| --- |
| var express = require('express');  var router = express.Router();  //引入模型  let accountModel = require('../../model/accountModel')  //导入moment库  let moment = require('moment')  //导入测用户登录的中间件  let checkLoginMiddleware = require('../../middleware/checkLoginMiddleware')  /\* GET home page. \*/  // router.get('/', function(req, res, next) {  //   res.render('index', { title: 'Express' });  // });  //显示账本列表的路由，从数据库读取所有文档数据渲染到页面,需要添加用户登录检测功能  router.get('/account',checkLoginMiddleware, function(req, res, next) {      //获取所有账单信息,按时间倒序排列    accountModel.find().sort({time:-1}).then(data=>{      res.render('list',{data:data,moment:moment});    }).catch(err=>{      res.status(500).send("读取数据失败")      return    })    });  //添加记录的路由  router.get('/account/create',checkLoginMiddleware, function(req, res, next) {    res.render('create');  });  //接收表单数据的路由  router.post('/account',checkLoginMiddleware, function(req, res) {    //express的脚手架帮我们添加了解析表单数据的中间件功能，所以这里我们可以直接使用req.body    // console.log(req.body);    //用mongodb保存数据    accountModel.create({        ...req.body,        time:moment(req.body.time).toDate()      })     .then(data=>{       console.log(data);       let msg = "数据添加成功"       let url = '/account'       res.render('success',{msg,url})     }).catch(err=>{       console.log(err)       res.status(500).send("插入数据失败")       return     })    });  //删除记录的路由  router.get('/account/:id',checkLoginMiddleware ,function(req, res, next) {    let id = req.params.id    // console.log(id)    //删除记录    accountModel.deleteOne({\_id:id}).then(data=>{        console.log(data);        let msg = "数据删除成功"        let url = '/account'        res.render('success',{msg,url}) //复用上面的success页面    }).catch(err=>{      console.log(err)       res.status(500).send("删除数据失败")       return    })    });  module.exports = router; |

# 五、实现退出登录功能

## 1.打开routes/web/auth.js,在里面添加退出登录的路由处理，其实比较简单，就是销毁session，然后重定向到登录页面

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入md5加密库用来加密用户密码  let md5 = require('md5')  //导入用户模型文件  let userModel = require('../../model/userModel')  //用户注册表单页面路由  router.get('/reg',(req,res)=>{    res.render('auth/reg')  })  //用户注册数据处理路由  router.post('/reg',(req,res)=>{    //表单验证,以后慢慢完善，如不允许有两个名字相同的用户     //获取表单数据    //  console.log(req.body);      userModel.create({       ...req.body,       password:md5(req.body.password)    }).then(data=>{      res.render('success',{msg:'注册成功',url:'/login'})    }).catch(err=>{      res.status(500).send("注册失败，请稍后再试")      return    })  })  //用户登录页面路由  router.get('/login',(req,res)=>{    res.render('auth/login')  })  //登录处理路由  router.post('/login',(req,res)=>{    //获取用户名和密码    let {username,password} = req.body    let pwd = md5(password) //对密码进行md5加密    //根据用户输入的信息到数据库里面查询改用户    userModel.findOne({username:username,password:pwd})       .then(data=>{        // console.log(data);         //需要判断如果没有查询到用户data是null，说明可能是用户名或者密码错误         if(!data){           return res.send("用户名或者密码错误！！！")          }          //写入session          req.session.username = data.username          req.session.\_is = data.\_id          res.render('success',{msg:"登录成功",url:"/account"})         }).catch(err=>{        res.status(500).send("登录失败，请稍后再试")        return      })    // res.render('auth/login')  })  //退出登录的路由处理  router.get('/logout',(req,res)=>{    //销毁session    req.session.destroy(()=>{      //需要渲染成功页面      res.render('success',{msg:'退出登录成功',url:"/login"})    })  })  module.exports = router; |

### 测试：我们在浏览器里面输入/logout回车，效果如下

|  |
| --- |
|  |
|  |

### 点击跳转按钮，就会进入登录页面

|  |
| --- |
|  |

## 2.虽然是成功了，但是以后一般不会直接在浏览器中输入/logout，我们需要修改一下账单列表，给他添加一个退出超链接，文件打开views/list.ejs,给他添加一个超链接，注意我跟老师的样式有点不一样

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>账本列表</title>      <script src="https://www.itxst.com/package/jquery-3.5.1/jquery.min.js"></script>      <script src="https://www.itxst.com/package/bootstrap-4.5.0/js/bootstrap.min.js"></script>      <link href="https://www.itxst.com/package/bootstrap-4.5.0/css/bootstrap.css" rel="stylesheet">      <script src="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/js/bootstrap-datepicker.min.js"></script>      <script src="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/locales/bootstrap-datepicker.zh-CN.min.js"></script>      <link href="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/css/bootstrap-datepicker.min.css" rel="stylesheet">      <style>          .red{              color:red;              border: 1px solid red;              background-color: #b1a2a2;              border-radius: 7px;              padding: 2px;          }          .blue{              color:blue;              border: 1px solid rgb(135, 235, 157);              border-radius: 7px;              background-color: lightblue;              padding: 2px;          }      </style>  </head>  <body>      <div class="container">              <h2>记账本</h2>                <hr>              <% data.forEach(item =>{%>                  <div>                      <ul class="list-group">                          <%  let actstatus = item.type===1?'list-group-item-success':'list-group-item-danger' %>                          <!-- <li class="list-group-item <%=actstatus%>"><%=item.time %></li> -->                          <li class="list-group-item <%=actstatus%>"><%=moment(item.time).format('YYYY-MM-DD') %></li>                          <li class="list-group-item">事项：<%=item.title %></li>                          <%  let myclass = item.type===1?'blue':'red' %>                          <li class="list-group-item" >类型：<span class="<%=myclass%>"><%=item.type===1?"收入":"支出" %></span></li>                          <li class="list-group-item">金额：<strong><%=item.account %></strong></li>                          <li class="list-group-item">备注：<%=item.remarks %></li>                          <li class="list-group-item">                              <a class="btn btn-primary" href="/account/create">添加</a>                              <a class="btn btn-warning delBtn" href="/account/<%=item.\_id%>">删除</a>                              <a class="btn btn-danger " href="/logout">退出</a>                          </li>                      </ul>                    </div>                <%})%>      </div>   <script>      //获取所有item的删除链接      let delBtns = document.querySelectorAll('.delBtn')      //给他们绑定事件      delBtns.forEach(btn =>{         btn.addEventListener("click",function(e) {              if(confirm("删除数据?")){                  return true              } else {                 e.preventDefault()              }         })      })   </script>  </body>  </html> |

### 测试：我们登录进入账单列表，然后点击退出按钮

|  |
| --- |
|  |
|  |
|  |

# 六、解决跨站请求伪造CSRF

## 为了防止一些坏人向我们发送非法退出请求，我们把auth.js中的退出的路由的请求方法由get改为post

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入md5加密库用来加密用户密码  let md5 = require('md5')  //导入用户模型文件  let userModel = require('../../model/userModel')  //用户注册表单页面路由  router.get('/reg',(req,res)=>{    res.render('auth/reg')  })  //用户注册数据处理路由  router.post('/reg',(req,res)=>{    //表单验证,以后慢慢完善，如不允许有两个名字相同的用户     //获取表单数据    //  console.log(req.body);      userModel.create({       ...req.body,       password:md5(req.body.password)    }).then(data=>{      res.render('success',{msg:'注册成功',url:'/login'})    }).catch(err=>{      res.status(500).send("注册失败，请稍后再试")      return    })  })  //用户登录页面路由  router.get('/login',(req,res)=>{    res.render('auth/login')  })  //登录处理路由  router.post('/login',(req,res)=>{    //获取用户名和密码    let {username,password} = req.body    let pwd = md5(password) //对密码进行md5加密    //根据用户输入的信息到数据库里面查询改用户    userModel.findOne({username:username,password:pwd})       .then(data=>{        // console.log(data);         //需要判断如果没有查询到用户data是null，说明可能是用户名或者密码错误         if(!data){           return res.send("用户名或者密码错误！！！")          }          //写入session          req.session.username = data.username          req.session.\_is = data.\_id          res.render('success',{msg:"登录成功",url:"/account"})         }).catch(err=>{        res.status(500).send("登录失败，请稍后再试")        return      })    // res.render('auth/login')  })  //退出登录的路由处理，使用post方法可以防止跨站请求伪造  router.post('/logout',(req,res)=>{    //销毁session    req.session.destroy(()=>{      //需要渲染成功页面      res.render('success',{msg:'退出登录成功',url:"/login"})    })  })  module.exports = router; |

## 然后需要修改一些list.ejs的退出超链接，把它放入一个form表单中，然后改为button

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>账本列表</title>      <script src="https://www.itxst.com/package/jquery-3.5.1/jquery.min.js"></script>      <script src="https://www.itxst.com/package/bootstrap-4.5.0/js/bootstrap.min.js"></script>      <link href="https://www.itxst.com/package/bootstrap-4.5.0/css/bootstrap.css" rel="stylesheet">      <script src="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/js/bootstrap-datepicker.min.js"></script>      <script src="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/locales/bootstrap-datepicker.zh-CN.min.js"></script>      <link href="https://www.itxst.com/package/bootstrap-datepicker-1.9.0/css/bootstrap-datepicker.min.css" rel="stylesheet">      <style>          .red{              color:red;              border: 1px solid red;              background-color: #b1a2a2;              border-radius: 7px;              padding: 2px;          }          .blue{              color:blue;              border: 1px solid rgb(135, 235, 157);              border-radius: 7px;              background-color: lightblue;              padding: 2px;          }      </style>  </head>  <body>      <div class="container">              <h2>记账本</h2>                <hr>              <% data.forEach(item =>{%>                  <div>                      <ul class="list-group">                          <%  let actstatus = item.type===1?'list-group-item-success':'list-group-item-danger' %>                          <!-- <li class="list-group-item <%=actstatus%>"><%=item.time %></li> -->                          <li class="list-group-item <%=actstatus%>"><%=moment(item.time).format('YYYY-MM-DD') %></li>                          <li class="list-group-item">事项：<%=item.title %></li>                          <%  let myclass = item.type===1?'blue':'red' %>                          <li class="list-group-item" >类型：<span class="<%=myclass%>"><%=item.type===1?"收入":"支出" %></span></li>                          <li class="list-group-item">金额：<strong><%=item.account %></strong></li>                          <li class="list-group-item">备注：<%=item.remarks %></li>                          <li class="list-group-item">                              <a class="btn btn-primary" href="/account/create">添加</a>                              <a class="btn btn-warning delBtn" href="/account/<%=item.\_id%>">删除</a>                          </li>                          <li  class="list-group-item">                                <form action="/logout" method="post">                                   <button class="btn btn-danger ">退出</button>                              </form>                          </li>                      </ul>                    </div>                <%})%>      </div>   <script>      //获取所有item的删除链接      let delBtns = document.querySelectorAll('.delBtn')      //给他们绑定事件      delBtns.forEach(btn =>{         btn.addEventListener("click",function(e) {              if(confirm("删除数据?")){                  return true              } else {                 e.preventDefault()              }         })      })   </script>  </body>  </html> |

### 测试一下

|  |
| --- |
|  |
|  |
|  |

## 也就是说如果不是获取数据的请求最好不要使用get方法

# 七、添加首页和404页面

## 我们虽然已经完成了很多功能，但是我们并没有编写首页功能，如果在我们的浏览器里面输入：localhost:3000/回车，就会单独一个错误

|  |
| --- |
|  |

## 1、我们打开index.js添加首页的路由规则，

|  |
| --- |
| var express = require('express');  var router = express.Router();  //引入模型  let accountModel = require('../../model/accountModel')  //导入moment库  let moment = require('moment')  //导入检测用户登录的中间件  let checkLoginMiddleware = require('../../middleware/checkLoginMiddleware')  //添加首页的路由规则  router.get('/', function(req, res, next) {    //只需要重定向到/account路由   res.redirect("/account")  });  //显示账本列表的路由，从数据库读取所有文档数据渲染到页面,需要添加用户登录检测功能  router.get('/account',checkLoginMiddleware, function(req, res, next) {      //获取所有账单信息,按时间倒序排列    accountModel.find().sort({time:-1}).then(data=>{      res.render('list',{data:data,moment:moment});    }).catch(err=>{      res.status(500).send("读取数据失败")      return    })    });  //添加记录的路由  router.get('/account/create',checkLoginMiddleware, function(req, res, next) {    res.render('create');  });  //接收表单数据的路由  router.post('/account',checkLoginMiddleware, function(req, res) {    //express的脚手架帮我们添加了解析表单数据的中间件功能，所以这里我们可以直接使用req.body    // console.log(req.body);    //用mongodb保存数据    accountModel.create({        ...req.body,        time:moment(req.body.time).toDate()      })     .then(data=>{       console.log(data);       let msg = "数据添加成功"       let url = '/account'       res.render('success',{msg,url})     }).catch(err=>{       console.log(err)       res.status(500).send("插入数据失败")       return     })    });  //删除记录的路由  router.get('/account/:id',checkLoginMiddleware ,function(req, res, next) {    let id = req.params.id    // console.log(id)    //删除记录    accountModel.deleteOne({\_id:id}).then(data=>{        console.log(data);        let msg = "数据删除成功"        let url = '/account'        res.render('success',{msg,url}) //复用上面的success页面    }).catch(err=>{      console.log(err)       res.status(500).send("删除数据失败")       return    })    });  module.exports = router; |

### 这样子如果用户没有登录，就会显示登录页面

|  |
| --- |
|  |

## 2.我们来制作404路由响应，其实在app.js已经由有了，我们只需要把它改进一下，在这里渲染一个我们自己的404模板文件

### app.js

|  |
| --- |
| var createError = require('http-errors');  var express = require('express');  var path = require('path');  var cookieParser = require('cookie-parser');  var logger = require('morgan');  var indexRouter = require('./routes/web/index');  //导入account.js接口路由文件  let accountRouter = require('./routes/api/account')  //导入注册路由文件  let authRouter = require('./routes/web/auth')  //导入express-session  let session = require('express-session')  //导入connect-mongo  let MongoStore = require('connect-mongo')  var app = express();  // view engine setup  app.set('views', path.join(\_\_dirname, 'views'));  app.set('view engine', 'ejs');  app.use(logger('dev'));  app.use(express.json());  app.use(express.urlencoded({ extended: false }));  app.use(cookieParser());  app.use(express.static(path.join(\_\_dirname, 'public')));  let {DBHOST,DBPORT,DBNAME} = require('./config/config')  let dbUrl = `mongodb://${DBHOST}:${DBPORT}/${DBNAME}`  //设置并且全局注册express-session中间件  app.use(session({    name:'sid',//设置cookie的name，默认是connect.sid    secret:"atguigu",//参与加密的字符串，又叫做签名    saveUninitialized:false,//是否为每一次请求都设置一个cookie用来存储session的id    resave:true,//是否在每一次请求时重新保存session    store:MongoStore.create({        mongoUrl:dbUrl //数据库连接配置    }),    cookie:{        httpOnly:true, //开启后，前端无法通过js操作        maxAge:1000\*60\*60\*24\*7 //控制session id的过期时间和cookie的过期时间    }  }))  app.use('/', indexRouter);  app.use('/', authRouter);  //注册api路由规则  app.use('/api', accountRouter);  // catch 404 and forward to error handler  app.use(function(req, res, next) {    // next(createError(404));    res.render('404')  });  // error handler  app.use(function(err, req, res, next) {    // set locals, only providing error in development    res.locals.message = err.message;    res.locals.error = req.app.get('env') === 'development' ? err : {};    // render the error page    res.status(err.status || 500);    res.render('error');  });  module.exports = app; |

## 3.然后我们在views文件夹下面新建一个404.ejs文件

|  |
| --- |
|  |

### 注意：如果你想的话，可以把你的404页面配置为公益404，可以参考网上的资料

# 8.会话控制，token介绍

## 1.）token的概念和作用

|  |
| --- |
|  |

## 2）.token的规则流程

|  |
| --- |
|  |
|  |

## 3）token的特点

|  |
| --- |
|  |

# 9.会话控制-JWT

|  |
| --- |
|  |

## 安装：npm i jsonwebtoken

## 使用方法

|  |
| --- |
|  |

### 注意：token是会过期的，过期了就会校验失败

### 小案例

|  |
| --- |
| let jwt = require('jsonwebtoken')  //创建  let token = jwt.sign(      {username:'xiaoming'},      'atguigu',      {expiresIn:60}   )  //  解析  jwt.verify(token,'atguigu',(err,data)=>{      if(err){          console.log('校验失败');          return      }      console.log(data);  }) |

### 测试

|  |
| --- |
|  |

# 10.完善记账本项目-登录响应token

## 1.我们上面已经基本把web端完成了，但是我们的api接口还不完善，还有可以直接访问。我们把上面的项目复制一份，改名：accounts-mongoose-resful-session-token，这个可能就是我们的最终版本

|  |
| --- |
|  |

## 2、把routes/web/auth.js复制一份到routes/api下面

|  |
| --- |
|  |

## 3.打开routes/api/auth.js,把除了两个post请求以外的所有路由删除

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入md5加密库用来加密用户密码  let md5 = require('md5')  //导入用户模型文件  let userModel = require('../../model/userModel')  //登录处理路由  router.post('/login',(req,res)=>{    //获取用户名和密码    let {username,password} = req.body    let pwd = md5(password) //对密码进行md5加密    //根据用户输入的信息到数据库里面查询改用户    userModel.findOne({username:username,password:pwd})       .then(data=>{        // console.log(data);         //需要判断如果没有查询到用户data是null，说明可能是用户名或者密码错误         if(!data){           return res.send("用户名或者密码错误！！！")          }          //写入session          req.session.username = data.username          req.session.\_is = data.\_id          res.render('success',{msg:"登录成功",url:"/account"})         }).catch(err=>{        res.status(500).send("登录失败，请稍后再试")        return      })  })  //退出登录的路由处理，使用post方法可以防止跨站请求伪造  router.post('/logout',(req,res)=>{    //销毁session    req.session.destroy(()=>{      //需要渲染成功页面      res.render('success',{msg:'退出登录成功',url:"/login"})    })  })  module.exports = router; |

## 4.修改代码提交token响应功能

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入md5加密库用来加密用户密码  let md5 = require('md5')  //导入用户模型文件  let userModel = require('../../model/userModel')  //导入jwt  let jwt = require('jsonwebtoken')  //登录处理路由  router.post('/login',(req,res)=>{    //获取用户名和密码    let {username,password} = req.body    let pwd = md5(password) //对密码进行md5加密    //根据用户输入的信息到数据库里面查询改用户    userModel.findOne({username:username,password:pwd})       .then(data=>{        // console.log(data);         //需要判断如果没有查询到用户data是null，说明可能是用户名或者密码错误         if(!data){           return  res.json({            code:'2002',            msg:"用户名或者密码错误！！！",            data:null          })        }          //创建token         let token = jwt.sign(            {              username:data.username,              \_id:data.\_id            },            "atguigu",            {expiresIn:60\*60\*24\*7} //设置7天有效，他的单位是秒         )          //响应token          res.json({            code:'0000',            msg:'登录成功' ,            data:token          })          res.render('success',{msg:"登录成功",url:"/account"})         }).catch(err=>{        res.json({          code:'2001',          msg:"数据库读取失败",          data:null        })        return      })  })  //退出登录的路由处理，使用post方法可以防止跨站请求伪造  router.post('/logout',(req,res)=>{    //销毁session    req.session.destroy(()=>{      //需要渲染成功页面      res.render('success',{msg:'退出登录成功',url:"/login"})    })  })  module.exports = router; |

## 5.在app.js中注册路由文件

|  |
| --- |
| var createError = require('http-errors');  var express = require('express');  var path = require('path');  var cookieParser = require('cookie-parser');  var logger = require('morgan');  var indexRouter = require('./routes/web/index');  //导入account.js接口路由文件  let accountRouter = require('./routes/api/account')  //导入注册路由文件  let authRouter = require('./routes/web/auth')  //导入token路由文件  let authApiRouter = require('./routes/api/auth')  //导入express-session  let session = require('express-session')  //导入connect-mongo  let MongoStore = require('connect-mongo')  var app = express();  // view engine setup  app.set('views', path.join(\_\_dirname, 'views'));  app.set('view engine', 'ejs');  app.use(logger('dev'));  app.use(express.json());  app.use(express.urlencoded({ extended: false }));  app.use(cookieParser());  app.use(express.static(path.join(\_\_dirname, 'public')));  let {DBHOST,DBPORT,DBNAME} = require('./config/config')  let dbUrl = `mongodb://${DBHOST}:${DBPORT}/${DBNAME}`  //设置并且全局注册express-session中间件  app.use(session({    name:'sid',//设置cookie的name，默认是connect.sid    secret:"atguigu",//参与加密的字符串，又叫做签名    saveUninitialized:false,//是否为每一次请求都设置一个cookie用来存储session的id    resave:true,//是否在每一次请求时重新保存session    store:MongoStore.create({        mongoUrl:dbUrl //数据库连接配置    }),    cookie:{        httpOnly:true, //开启后，前端无法通过js操作        maxAge:1000\*60\*60\*24\*7 //控制session id的过期时间和cookie的过期时间    }  }))  app.use('/', indexRouter);  app.use('/', authRouter);  //注册api路由规则  app.use('/api', accountRouter);  //注册token响应路由规则  app.use('/api', authApiRouter);  // catch 404 and forward to error handler  app.use(function(req, res, next) {    // next(createError(404));    res.render('404')  });  // error handler  app.use(function(err, req, res, next) {    // set locals, only providing error in development    res.locals.message = err.message;    res.locals.error = req.app.get('env') === 'development' ? err : {};    // render the error page    res.status(err.status || 500);    res.render('error');  });  module.exports = app; |

### 测试，成功返回token

|  |
| --- |
|  |

### 如果密码错，返回错误信息

|  |
| --- |
|  |

# 11.完善记账本api之token检验

## 1.我们用记账本列表路由来做token校验，打开api/account.js，进入/account的get请求方法里面，问题是token保存在哪里。这个是有服务器决定的，一般在请求头，而且名字不是固定的，有时候是token，有时候是tk还有时候 是user key。

|  |
| --- |
|  |

### 测试：没有登录的时候，会报错

|  |
| --- |
|  |

### 然后我们登录成功，再来测试，需要把成功的token复制过来粘贴到请求头的token参数的值那里然后就请求成功

|  |
| --- |
|  |

### 虽然成功，但是只有一个接口有，其他接口没有，怎么办？

## 2.把这个功能封装成一个中间件

|  |
| --- |
| // 定义token验证中间件  let checkTokenMiddleware = (req,res,next)=>{     //获取token    let token = req.get('token')    //判断token是否有值，需要处理没有值的情况    if(!token){        return res.json({          code:'2003',msg:'token缺失',data:null        })    }     //token校验     jwt.verify(token,'atguigu',(err,data)=>{      if(err){        return res.json({          code:'2004',msg:'token校验失败',data:null        })      }      //token校验成功，就放行      next()    })  } |

### 然后在需要使用这个中间件的地方把它放在第二个参数的位置

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入jwt  let jwt = require('jsonwebtoken')  //引入模型  let accountModel = require('../../model/accountModel')  //导入moment库  let moment = require('moment')  // console.log(moment('2023-02-23').toDate());//ok  // 定义token验证中间件  let checkTokenMiddleware = (req,res,next)=>{     //获取token    let token = req.get('token')    //判断token是否有值，需要处理没有值的情况    if(!token){        return res.json({          code:'2003',msg:'token缺失',data:null        })    }     //token校验     jwt.verify(token,'atguigu',(err,data)=>{      if(err){        return res.json({          code:'2004',msg:'token校验失败',data:null        })      }      //token校验成功，就放行      next()    })  }  //显示账本列表的路由，从数据库读取所有文档数据渲染到页面  router.get('/account',checkTokenMiddleware, function(req, res, next) {    //获取所有账单信息,按时间倒序排列      accountModel.find().sort({time:-1}).then(data=>{        //api接口是返回json数据给客户端的，json数据有以下属性        //1.code，通常是0000或者20000，建议使用0000注意这里不是http状态码        //2.msg也就是信息        //3.data也就是响应数据        res.json({          code:'0000',          msg:'读取成功',          data:data        })      }).catch(err=>{        res.json({            code:'1001',//json响应失败的状态码是1xxxx，如1001            msg:'读取失败',            data:null        })        return      })    });  //新增账单的api  router.post('/account',checkTokenMiddleware, function(req, res) {      //用mongodb保存数据    accountModel.create({        ...req.body,        time:moment(req.body.time).toDate()      })     .then(data=>{      res.json({          code:'0000',          msg:'插入数据成功',          data:data        })     }).catch(err=>{      res.json({          code:'1002',//json响应失败的状态码是1xxxx，如1001          msg:'插入数据失败',          data:null      })      return     })    });  //修改账单的api  router.patch('/account/:id',checkTokenMiddleware,function(req,res,next){        let {id} = req.params        accountModel.updateOne({\_id:id},req.body)        .then(data=>{          //这里是默认的处理方式，他会返回一个统计对象，如果你不想要，就需要在下面做处理，获取你需要的数据          // res.json({          //   code:'0000',          //   msg:'更新数据成功',          //   data:data          // })         //由于我们需要更新后的数据，所以这里需要根据id查询数据,然后把结果返回           accountModel.findById(id).then(data=>{              res.json({                code:'0000',                msg:'更新数据成功',                data:data              })           })        }).catch(err=>{          res.json({            code:'1005',            msg:'更新数据失败',            data:null          })        })  })  //删除记录的api  router.delete('/account/:id', checkTokenMiddleware,function(req, res, next) {    let id = req.params.id    // console.log(id)    //删除记录    accountModel.deleteOne({\_id:id}).then(data=>{      res.json({          code:'0000',          msg:'删除数据成功',          data:{}        })    }).catch(err=>{      res.json({          code:'1003',//json响应失败的状态码是1xxxx，如1001          msg:'删除数据失败',          data:null      })      return    })    });  //获取一条账单的api接口  router.get('/account/:id',checkTokenMiddleware,function(req,res,next) {    let id = req.params.id    accountModel.findById({\_id:id}).then(data=>{              res.json({                code:'0000',                msg:'查询数据成功',                data:data            })         }).catch(err=>{          res.json({            code:'1004',            msg:'查询数据失败',            data:null        })      })  })  module.exports = router; |

### 测试一下，成功

|  |
| --- |
|  |

### 然后我们需要把这个中间件抽取出来粘贴到一个文件然后放到middlware文件夹下面

### middleware/checkTokenMiddleware.js

|  |
| --- |
| // 定义token验证中间件  //导入jwt  let jwt = require('jsonwebtoken')  module.exports = (req,res,next)=>{      //获取token     let token = req.get('token')     //判断token是否有值，需要处理没有值的情况     if(!token){         return res.json({           code:'2003',msg:'token缺失',data:null         })     }      //token校验      jwt.verify(token,'atguigu',(err,data)=>{       if(err){         return res.json({           code:'2004',msg:'token校验失败',data:null         })       }       //token校验成功，就放行       next()     })   } |

### 然后在api/account.js里面导入这个中间件

|  |
| --- |
| var express = require('express');  var router = express.Router();  //引入模型  let accountModel = require('../../model/accountModel')  //导入moment库  let moment = require('moment')  // console.log(moment('2023-02-23').toDate());//ok  //导入token验证中间件  let checkTokenMiddleware = require('../../middleware/checkTokenMiddleware')  //显示账本列表的路由，从数据库读取所有文档数据渲染到页面  router.get('/account',checkTokenMiddleware, function(req, res, next) {    //获取所有账单信息,按时间倒序排列      accountModel.find().sort({time:-1}).then(data=>{        //api接口是返回json数据给客户端的，json数据有以下属性        //1.code，通常是0000或者20000，建议使用0000注意这里不是http状态码        //2.msg也就是信息        //3.data也就是响应数据        res.json({          code:'0000',          msg:'读取成功',          data:data        })      }).catch(err=>{        res.json({            code:'1001',//json响应失败的状态码是1xxxx，如1001            msg:'读取失败',            data:null        })        return      })    });  //新增账单的api  router.post('/account',checkTokenMiddleware, function(req, res) {      //用mongodb保存数据    accountModel.create({        ...req.body,        time:moment(req.body.time).toDate()      })     .then(data=>{      res.json({          code:'0000',          msg:'插入数据成功',          data:data        })     }).catch(err=>{      res.json({          code:'1002',//json响应失败的状态码是1xxxx，如1001          msg:'插入数据失败',          data:null      })      return     })    });  //修改(更新)账单的api  router.patch('/account/:id',checkTokenMiddleware,function(req,res,next){        let {id} = req.params        accountModel.updateOne({\_id:id},req.body)        .then(data=>{          //这里是默认的处理方式，他会返回一个统计对象，如果你不想要，就需要在下面做处理，获取你需要的数据          // res.json({          //   code:'0000',          //   msg:'更新数据成功',          //   data:data          // })         //由于我们需要更新后的数据，所以这里需要根据id查询数据,然后把结果返回           accountModel.findById(id).then(data=>{              res.json({                code:'0000',                msg:'更新数据成功',                data:data              })           })        }).catch(err=>{          res.json({            code:'1005',            msg:'更新数据失败',            data:null          })        })  })  //删除记录的api  router.delete('/account/:id', checkTokenMiddleware,function(req, res, next) {    let id = req.params.id    // console.log(id)    //删除记录    accountModel.deleteOne({\_id:id}).then(data=>{      res.json({          code:'0000',          msg:'删除数据成功',          data:{}        })    }).catch(err=>{      res.json({          code:'1003',//json响应失败的状态码是1xxxx，如1001          msg:'删除数据失败',          data:null      })      return    })    });  //获取一条账单的api接口  router.get('/account/:id',checkTokenMiddleware,function(req,res,next) {    let id = req.params.id    accountModel.findById({\_id:id}).then(data=>{              res.json({                code:'0000',                msg:'查询数据成功',                data:data            })         }).catch(err=>{          res.json({            code:'1004',            msg:'查询数据失败',            data:null        })      })  })  module.exports = router; |

### 测试一下，成功

|  |
| --- |
|  |

### 虽然成功，不过用户不会这么干，怎么优化？

## 3.完善token功能

### 1）把加密的密钥，抽取出来，放到config/config.js文件夹中

|  |
| --- |
| //配置文件  module.exports = {      DBHOST:'127.0.0.1',      DBPORT:27017,      DBNAME:"myaccount",      secret:'atguigu'  } |

### 2）在auth.js中导入密钥

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入md5加密库用来加密用户密码  let md5 = require('md5')  //导入用户模型文件  let userModel = require('../../model/userModel')  //导入jwt  let jwt = require('jsonwebtoken')  //从配置文件中获取加密密钥  let {secret} = require('../../config/config')  //登录处理路由  router.post('/login',(req,res)=>{    //获取用户名和密码    let {username,password} = req.body    let pwd = md5(password) //对密码进行md5加密    //根据用户输入的信息到数据库里面查询改用户    userModel.findOne({username:username,password:pwd})       .then(data=>{        // console.log(data);         //需要判断如果没有查询到用户data是null，说明可能是用户名或者密码错误         if(!data){           return  res.json({            code:'2002',            msg:"用户名或者密码错误！！！",            data:null          })        }          //创建token         let token = jwt.sign(            {              username:data.username,              \_id:data.\_id            },            secret,            {expiresIn:60\*60\*24\*7} //设置7天有效，他的单位是秒         )          //响应token          res.json({            code:'0000',            msg:'登录成功' ,            data:token          })          res.render('success',{msg:"登录成功",url:"/account"})         }).catch(err=>{        res.json({          code:'2001',          msg:"数据库读取失败",          data:null        })        return      })  })  //退出登录的路由处理，使用post方法可以防止跨站请求伪造  router.post('/logout',(req,res)=>{    //销毁session    req.session.destroy(()=>{      //需要渲染成功页面      res.render('success',{msg:'退出登录成功',url:"/login"})    })  })  module.exports = router; |

### 3）在checkTokenMiddleware.js中导入密钥

|  |
| --- |
| //导入jwt  let jwt = require('jsonwebtoken')  //从配置文件中获取密钥  let {secret} = require('../config/config')  // 定义token验证中间件  module.exports = (req,res,next)=>{      //获取token     let token = req.get('token')     //判断token是否有值，需要处理没有值的情况     if(!token){         return res.json({           code:'2003',msg:'token缺失',data:null         })     }      //token校验      jwt.verify(token,secret,(err,data)=>{       if(err){         return res.json({           code:'2004',msg:'token校验失败',data:null         })       }       //token校验成功，就放行       next()     })   } |

#### 测试一下，工作正常

|  |
| --- |
|  |

### 4）需要进一步完善中间件文件，我们修改一下middleware/checkTokenMiddleware.js文件

|  |
| --- |
| //导入jwt  let jwt = require('jsonwebtoken')  //从配置文件中获取密钥  let {secret} = require('../config/config')  // 定义token验证中间件  module.exports = (req,res,next)=>{      //获取token     let token = req.get('token')     //判断token是否有值，需要处理没有值的情况     if(!token){         return res.json({           code:'2003',msg:'token缺失',data:null         })     }      //token校验      jwt.verify(token,secret,(err,data)=>{       if(err){         return res.json({           code:'2004',msg:'token校验失败',data:null         })       }        //保存用户信息        req.user = data       //token校验成功，就放行       next()     })   } |

# 12完善记账本之本地域名配置

## 什么是本地域名

|  |
| --- |
|  |
|  |

## 配置表单域名，如我们把本地域名配置为：[www.kennychoi.com](http://www.kennychoi.com)

|  |
| --- |
|  |

### 然后我们来测试一下，前提是：服务要启动起来，成功了

|  |
| --- |
|  |

### 注意：hosts文件有一个作用就是可以用来屏蔽广告或者防止一些软件更新，只需要把127.0.0.1 和需要屏蔽的广告网站地址关联起来即可

|  |
| --- |
|  |
|  |