天猫商城

### 一 开发环境配置

#### linux系统安装

在VMware中安装linux系统，本项目使用的是CentOS-6.9-x86\_64-bin-DVD1。

#### linux软件源配置

##### 1.2.1 备份

|  |
| --- |
| mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.backup |

##### 1.2.2 下载

下载centos-base.repo到/etc/yum.reposd/

|  |
| --- |
| wget -O /etc/yum.repos.d/CentOS-Base.repo <http://mirrors.aliyun.com/repo/Centos-6.repo> |

##### 1.2.3 运行

|  |
| --- |
| yum makecache |

#### linux下安装JDK

##### 下载JDK

|  |
| --- |
| wget jdk地址 |

##### 删除自带JDK

|  |
| --- |
| java –version  rpm –qa | grep jdk  sudo yum remove “系统自带的JDK” |

##### 安装JDK

|  |
| --- |
| 赋予JDK权限  sudo chmod 777 jdk-8u162-linux-x64.rpm  解压JDK，解压后默认在/usr/java目录下  sudo rpm –ivh jdk-8u162-linux-x64.rpm  配置环境变量  sudo vim /etc/profile  在文件最下方添加环境变量  export JAVA\_HOME=/usr/java/jdk-8u162-linux-x64  export CLASSPATH=.:$JAVA\_HOME/jre/lib/rt.jar:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar  export PATH=$PATH:$JAVA\_HOME/bin  使环境变量生效  source /etc/profile |

#### linux下安装tomcat

##### 1.4.1 下载tomcat

|  |
| --- |
| wget tomcat地址 |

##### 1.4.2 解压tomcat

|  |
| --- |
| tar –zxvf apche-tomcat-7.0.85.tar.gz |

##### 1.4.3 安装tomcat

|  |
| --- |
| // 复制解压的tomcat到指定文件夹  cp –rf /root/apache-tomcat-7.0.85 /developer  //配置环境变量  sudo vim /etc/profile  // 在文件最下方添加环境变量  export CATALINA\_HOME=/developer/apache-tomcat-7.0.85  // 使环境变量生效  source /etc/profile |

##### 1.4.4 配置UTF-8字符集

|  |
| --- |
| sudo vim /developer/config/server.xml  // 在8080端口配置的末尾添加字符集  Encoding=”UTF-8” |

#### linux下安装maven

##### 1.5.1 下载maven

|  |
| --- |
| wget maven地址 |

##### 1.5.2 解压maven

|  |
| --- |
| tar –zxvf apach-maven-3.5.3-bin.tar.gz |

##### 1.5.3 安装maven

|  |
| --- |
| // 复制解压的maven到指定文件夹  cp –rf /root/apache-maven-3.5.3/developer  //配置环境变量  sudo vim /etc/profile  // 在文件最下方添加环境变量  export CATALINA\_HOME=/developer/apache-maven-3.5.3  export PATH=$PATH:$JAVA\_HOME/bin:$MAVEN\_HOME/bin  // 使环境变量生效  source /etc/profile |

#### linux下安装vsftpd

##### 1.6.1 安装vsftpd

|  |
| --- |
| yum –y install vsftpd  // 验证是否安装成功，配置文件在/etc/vsftpdvsftpd.conf  rpm –qa|grep vsftpd |

##### 1.6.2 创建虚拟用户

|  |
| --- |
| // 创建文件夹  mkdir /ftpfile  // 添加匿名用户  useradd ftpuser –d /ftpfile –s /sbin/nologin  // 修改ftpfile权限  chown –R ftpuser.ftpuser /ftpfile  // 重置ftpuser密码，密码为ftpuser  passwd ftpuser |

##### 1.6.3 配置vsftpd

|  |
| --- |
| vim /etc/vsftpd/vsftpd.conf  // 添加信息  anonymous\_enable=NO  ftpd\_banner=Welcome to tmall FTP service.  local\_root=/ftpfile  use\_localtime=yes  chroot\_list\_enable=YES  chroot\_lst\_file=/etc/vsftpd/chroot\_list  pasv\_min\_port=61001  pasv\_max\_port=62000  // 防火墙配置  -A INPUT -p TCP --dport 61001:62000 -j ACCEPT  -A OUTPUT -p TCP --sport 61001:62000 -j ACCEPT  -A INPUT -p TCP --dport 20 -j ACCEPT  -A OUTPUT -p TCP --sport 20 -j ACCEPT  -A INPUT -p TCP --dport 21 -j ACCEPT  -A OUTPUT -p TCP --sport 21 -j ACCEPT  // 使匿名用户可以创建文件  vim /etc/selinux/config  // 添加  SELINUX=disable  #SELINUX=enforcing  // 让命令生效  setenforce 0  // 重启ftp  service vsftpd restart  // 连接ftp  ftp 192.168.178.130 |

#### window下安装ftpserver

##### 1.7.1 安装配置ftpserver

下载解压缩FTPServer，并配置好用户名密码

#### linux下安装nginx

##### 1.8.1 安装nginx依赖

|  |
| --- |
| // 安装nginx的依赖  yum install gcc-c++  yum install pcre pcre-devel  yum install zlib zlib-devel  yum install openssl openssl-devel |

##### 1.8.2 下载nginx

|  |
| --- |
| wget ngin地址 |

##### 1.8.3 解压nginx

|  |
| --- |
| tar –zxvf nginx-1.12.2.tar.gz |

##### 1.8.4 安装nginx

|  |
| --- |
| 进入nginx目录执行./configure  make  make install  find –name nginx  nginx默认装载/usr/local |

##### 1.8.5 nginx常用命令

|  |
| --- |
| // 测试配置文件  安装路径下/nginx/sbin/nginx –t  // 启动命令  安装路径下/nginx/sbin/nginx  // 停止命令  安装路径下/nginx/sbin/nginx –stop 或者 nginx –s quit  // 重启命令  安装路径下/nginx/sbin/nginx –s reload  // 查看进程命令  ps –ef | grep nginx  // 平滑重启  kill –HUP nginx主进程号  // 增加防火墙访问权限  vim /etc/sysconfig/iptables  -A INPUT –p tcp –m state –state NEW –m tcp –dport 80 –j ACCEPT  Service iptables restart |

#### linux下nginx域名解析配置

##### 1.9.1 配置hosts文件

|  |
| --- |
| //修改hosts，window和linux都要修改  vim /etc/hosts  192.168.178.130 www.laughing.com  192.168.178.139 image.laughing.com  192.168.178.139 s.laughing.com |

##### 1.9.2 配置nginx域名

|  |
| --- |
| // 进入nginx目录  cd /usr/local/nginx/conf  mkdir vhost  vim nginx.conf  添加include vhost/\*.conf;  // 在vshost下新建文件  vim www.laughing.com.conf  server {  listen 80;  autoindex on;  server\_name www.laughing.com;  access\_log /usr/local/nginx/logs/access.log combined;  index index.html index.htm index.jsp index.php;  #error\_page 404 /404.html;  if ( $query\_string ~\* ".\*[\;'\<\>].\*" ){  return 404;  }  location / {  proxy\_pass http://127.0.0.1:8080;  add\_header Access-Control-Allow-Origin \*;  }  }  //  vim image.laughing.com.conf  server {  listen 80;  autoindex off;  server\_name image.laughing.com;  access\_log /usr/local/nginx/logs/access.log combined;  index index.html index.htm index.jsp index.php;  #error\_page 404 /404.html;  if ( $query\_string ~\* ".\*[\;'\<\>].\*" ){  return 404;  }  location ~ /(mmall\_fe|mmall\_admin\_fe)/dist/view/\* {  deny all;  }  location / {  root /ftpfile/;  add\_header Access-Control-Allow-Origin \*;  }  }  // 访问测试  www.laughing.com  image.laughing.com |

#### windows下安装nginx

##### 1.10.1 下载解压nginx

在nginx官网下载稳定版本，放在本地任意位置并解压

##### 1.10.2配置nginx

（1）在nginx目录conf文件夹下的ngin.conf中http结束标签后添加include vhost/\*.conf;

（2）在conf文件夹下创建vhost文件夹

（3）新建image.laughing.com.conf文件

|  |
| --- |
| server {  listen 80;  autoindex off;  server\_name image.laughing.com;  access\_log c:/access.log combined;  index index.html index.htm index.jsp index.php;  #error\_page 404 /404.html;  if ( $query\_string ~\* ".\*[\;'\<\>].\*" ){  return 404;  }  location ~ /(mmall\_fe|mmall\_admin\_fe)/dist/view/\* {  deny all;  }  location / {  root F:\开发\Share\img;  add\_header Access-Control-Allow-Origin \*;  }  } |

（5）测试

在cmd窗口执行nginx –t

（6）新建tomcat.laughing.com.conf文件

|  |
| --- |
| server {  listen 80;  autoindex on;  server\_name tomcat.laughing.com;  access\_log c:/access.log combined;  index index.html index.htm index.jsp index.php;  #error\_page 404 /404.html;  if ( $query\_string ~\* ".\*[\;'\<\>].\*" ){  return 404;  }  location / {  proxy\_pass http://127.0.0.1:8080;  add\_header Access-Control-Allow-Origin \*;  }  } |

（7）修改hosts文件

|  |
| --- |
| 127.0.0.1 image.laughing.com  127.0.0.1 tomcat.laughing.com |

（8）重启nginx

|  |
| --- |
| nginx -s reload |

（9）访问域名

|  |
| --- |
| tomcat.laughing.com  image.laughing.com |

#### linux下安装mysql

##### 1.11.1 安装mysql

|  |
| --- |
| yum –y install mysql-server  rpm –qa|grep mysql-server |

##### 1.11.2 配置字符集

|  |
| --- |
| default-character-set=utf8  character-set-server=utf8 |

##### 1.11.3 配置mysql自启动

|  |
| --- |
| chkconfig mysqld on  chkconfig –list mysqld  2-5位为on状态 |

##### 1.11.4 配置防火墙

|  |
| --- |
| vim /etc/sysconfig/iptables  -A INPUT –p tcp –m tcp –dport 3306 –j ACCEPT  service iptables restart |

##### 1.11.5 创建mysql帐号

|  |
| --- |
| insert into mysql.user(Host,User,Password) values("localhost","laughing",password("laughing"));  select user,host from mysql.user;  // 删除匿名用户  Delete from mysql.user where user=’’;  Flush privileges; |

##### 1.11.6 创建数据库

|  |
| --- |
| create database `tmall` default character set utf8 collate utf8\_general\_ci;  // 查看权限  select \* from mysql.user \G  // 赋予本地用户所有权限  grant select,delete,create on tmall.\* to laughing@’localhost’ identified by ‘laughing’  // 赋予帐号外网所有权限  grant all privileges on tmall.\* to laughing@’%’ identified by ‘laughing’ with grant option |

#### linux下安装git

##### 1.12.1 下载git

|  |
| --- |
| wget https://github.com/git/git/archive/v2.8.0.tar.gz |

##### 1.12.2 安装依赖

|  |
| --- |
| yum -y install zlib-devel openssl-devel cpio expat-devel gettext-devel curl-devel perl-ExtUtils-CBuilder perl-ExtUtils-MakeMaker |

##### 1.12.3 解压git

|  |
| --- |
| tar –zxvf v2.8.0.tar.gz |

##### 1.12.4 编译git

|  |
| --- |
| cd git-2.8.0  make prefix=/usr/local all |

##### 1.12.5 安装git

|  |
| --- |
| make prefix=/usr/local install  git --version |

##### 1.12.6 配置私钥

|  |
| --- |
| ssh-keygen –t rsa –C “425984393@qq.com”  ssh-agent bash  ssh-add ~/.ssh/id\_rsa（/前面是数字1左边的波浪线）  cat ~/.ssh/id\_rsa.pub  复制key的内容到github |

### 二 项目初始化

#### 2.1 配置IDEA

##### 2.1.1 配置JDK

Configure->Project Defaults->Project Structure

##### 2.1.2 配置Maven

Configure->Project Defaults->Settings

#### 2.2 git初始化

##### 2.2.1 创建README.md文件

在项目根目录下创建README.md文件

##### 2.2.2 配置.gitignore文件

在项目根目录下创建.gitignore文件

|  |
| --- |
| \*.class  #package file \*.war \*.ear  #kdiff3 ignore \*.orig  #maven ignore target/  #eclipse ignore .settings/ .project .classpath  #idea .idea/ /idea/ \*.ipr \*.iml \*.iws  #temp file \*.log \*.cache \*.diff \*.patch \*.tmp  #system ignore .DS\_Store Thumbs.db |

##### 2.2.3 提交到git仓库

|  |
| --- |
| git add .  git status  git commit -m "first commit init project"  git remote add origin [git@github.com:Lee-Shawn/tmall.git](mailto:git@github.com:Lee-Shawn/tmall.git)  git branch  git pull  git push -u -f origin master |

##### 2.2.4 创建分支

|  |
| --- |
| // 查看当前分支  git branch –r  // 创建新分支v1.0  git checkout -b v1.0 origin/master  // 推送到远程  git push origin HEAD -u |

##### 2.2.5 合并分支到master

|  |
| --- |
| git checkout master  git pull origin master  git merge v1.0 |

#### 2.3 配置pom.xml

##### 2.3.1 配置maven的pom.xml文件

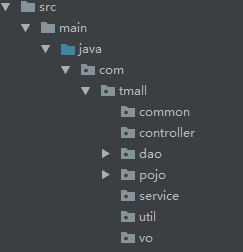
在pom.xml中配置项目所需要的依赖

|  |
| --- |
| <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/maven-v4\_0\_0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com</groupId>  <artifactId>tmall</artifactId>  <packaging>war</packaging>  <version>1.0-SNAPSHOT</version>  <name>tmall Maven Webapp</name>  <url>http://maven.apache.org</url>   <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <maven.compiler.encoding>UTF-8</maven.compiler.encoding>   <org.springframework.version>4.3.10.RELEASE</org.springframework.version>  <org.mybatis.version>3.4.1</org.mybatis.version>  <org.mybatis.spring.version>1.3.0</org.mybatis.spring.version>  </properties>   <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>   <dependency>  <groupId>org.apache.tomcat</groupId>  <artifactId>tomcat-servlet-api</artifactId>  <version>7.0.64</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>${org.springframework.version}</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-oxm</artifactId>  <version>${org.springframework.version}</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${org.springframework.version}</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</artifactId>  <version>${org.springframework.version}</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>${org.springframework.version}</version>  </dependency>   <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>1.8.1</version>  </dependency>   <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>${org.mybatis.spring.version}</version>  </dependency>  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>${org.mybatis.version}</version>  </dependency>   <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjrt</artifactId>  <version>1.8.13</version>  </dependency>   <dependency>  <groupId>org.codehaus.jackson</groupId>  <artifactId>jackson-mapper-asl</artifactId>  <version>1.9.13</version>  </dependency>   <dependency>  <groupId>commons-dbcp</groupId>  <artifactId>commons-dbcp</artifactId>  <version>1.4</version>  <!--<scope>runtime</scope>-->  </dependency>   <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-classic</artifactId>  <version>1.1.8</version>  <scope>compile</scope>  </dependency>  <dependency>  <groupId>ch.qos.logback</groupId>  <artifactId>logback-core</artifactId>  <version>1.1.8</version>  <scope>compile</scope>  </dependency>   <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.38</version>  </dependency>   <dependency>  <groupId>com.google.guava</groupId>  <artifactId>guava</artifactId>  <version>20.0</version>  </dependency>   <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  <version>3.1</version>  </dependency>   <dependency>  <groupId>commons-collections</groupId>  <artifactId>commons-collections</artifactId>  <version>3.2.1</version>  </dependency>   <dependency>  <groupId>joda-time</groupId>  <artifactId>joda-time</artifactId>  <version>2.3</version>  </dependency>   <!-- id加密解密 -->  <dependency>  <groupId>org.hashids</groupId>  <artifactId>hashids</artifactId>  <version>1.0.1</version>  </dependency>   <!-- ftpclient -->  <dependency>  <groupId>commons-net</groupId>  <artifactId>commons-net</artifactId>  <version>3.1</version>  </dependency>   <!-- file upload -->   <!-- https://mvnrepository.com/artifact/commons-fileupload/commons-fileupload -->  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  <version>1.2.2</version>  </dependency>   <dependency>  <groupId>commons-io</groupId>  <artifactId>commons-io</artifactId>  <version>2.0.1</version>  </dependency>   <!-- mybatis pager -->   <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  <version>4.1.0</version>  </dependency>   <dependency>  <groupId>com.github.miemiedev</groupId>  <artifactId>mybatis-paginator</artifactId>  <version>1.2.17</version>  </dependency>   <dependency>  <groupId>com.github.jsqlparser</groupId>  <artifactId>jsqlparser</artifactId>  <version>0.9.4</version>  </dependency>   <!-- alipay -->  <dependency>  <groupId>commons-codec</groupId>  <artifactId>commons-codec</artifactId>  <version>1.10</version>  </dependency>  <dependency>  <groupId>commons-configuration</groupId>  <artifactId>commons-configuration</artifactId>  <version>1.10</version>  </dependency>  <dependency>  <groupId>commons-lang</groupId>  <artifactId>commons-lang</artifactId>  <version>2.6</version>  </dependency>  <dependency>  <groupId>commons-logging</groupId>  <artifactId>commons-logging</artifactId>  <version>1.1.1</version>  </dependency>  <dependency>  <groupId>com.google.zxing</groupId>  <artifactId>core</artifactId>  <version>2.1</version>  </dependency>  <dependency>  <groupId>com.google.code.gson</groupId>  <artifactId>gson</artifactId>  <version>2.3.1</version>  </dependency>  <dependency>  <groupId>org.hamcrest</groupId>  <artifactId>hamcrest-core</artifactId>  <version>1.3</version>  </dependency>   <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.9.0</version>  </dependency>   </dependencies>   <build>  <finalName>tmall</finalName>  <plugins>  <plugin>  <groupId>org.mybatis.generator</groupId>  <artifactId>mybatis-generator-maven-plugin</artifactId>  <version>1.3.2</version>  <configuration>  <verbose>true</verbose>  <overwrite>true</overwrite>  </configuration>  </plugin>   <!-- geelynote maven的核心插件之-complier插件默认只支持编译Java 1.4，因此需要加上支持高版本jre的配置，在pom.xml里面加上 增加编译插件 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  <compilerArguments>  <extdirs>${project.basedir}/src/main/webapp/WEB-INF/lib</extdirs>  </compilerArguments>  </configuration>  </plugin>  </plugins>   </build> </project> |

#### 2.4 项目包结构初始化

##### 2.4.1 新建包结构

（1）在src文件夹先新建main/java/com/tmall，然后再分别建各个功能文件夹。



（2）在src文件夹下新建与main同级的test文件夹。

（3）在java文件夹上右键选择make directory as将其设为Resource Root，在test文件夹上右键设为Test Resource Root。

#### 2.5 配置mybaits-generator

##### 2.5.1 配置pom.xml

在pom.xml中配置依赖，引入jar包

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.mybatis.generator</groupId>  <artifactId>mybatis-generator-maven-plugin</artifactId>  <version>1.3.2</version>  <configuration>  <verbose>true</verbose>  <overwrite>true</overwrite>  </configuration>  </plugin>  </plugins> </build> |

##### 2.5.2 配置generator.xml

在resources目录下新建generator.xml文件，并进行配置

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE generatorConfiguration  PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">  <generatorConfiguration>  <!--导入属性配置-->  <properties resource="datasource.properties"></properties>   <!--指定特定数据库的jdbc驱动jar包的位置-->  <classPathEntry location="${db.driverLocation}"/>   <context id="default" targetRuntime="MyBatis3">   <!-- optional，旨在创建class时，对注释进行控制 -->  <commentGenerator>  <property name="suppressDate" value="true"/>  <property name="suppressAllComments" value="true"/>  </commentGenerator>   <!--jdbc的数据库连接 -->  <jdbcConnection  driverClass="${db.driverClassName}"  connectionURL="${db.url}"  userId="${db.username}"  password="${db.password}">  </jdbcConnection>   <!-- 非必需，类型处理器，在数据库类型和java类型之间的转换控制-->  <javaTypeResolver>  <property name="forceBigDecimals" value="false"/>  </javaTypeResolver>    <!-- Model模型生成器,用来生成含有主键key的类，记录类 以及查询Example类  targetPackage 指定生成的model生成所在的包名  targetProject 指定在该项目下所在的路径  -->  <!--<javaModelGenerator targetPackage="com.tmall.pojo" targetProject=".\src\main\java">-->  <javaModelGenerator targetPackage="com.tmall.pojo" targetProject="./src/main/java">  <!-- 是否允许子包，即targetPackage.schemaName.tableName -->  <property name="enableSubPackages" value="false"/>  <!-- 是否对model添加 构造函数 -->  <property name="constructorBased" value="true"/>  <!-- 是否对类CHAR类型的列的数据进行trim操作 -->  <property name="trimStrings" value="true"/>  <!-- 建立的Model对象是否 不可改变 即生成的Model对象不会有 setter方法，只有构造方法 -->  <property name="immutable" value="false"/>  </javaModelGenerator>   <!--mapper映射文件生成所在的目录 为每一个数据库的表生成对应的SqlMap文件 -->  <!--<sqlMapGenerator targetPackage="mappers" targetProject=".\src\main\resources">-->  <sqlMapGenerator targetPackage="mappers" targetProject="./src/main/resources">  <property name="enableSubPackages" value="false"/>  </sqlMapGenerator>   <!-- 客户端代码，生成易于使用的针对Model对象和XML配置文件 的代码  type="ANNOTATEDMAPPER",生成Java Model 和基于注解的Mapper对象  type="MIXEDMAPPER",生成基于注解的Java Model 和相应的Mapper对象  type="XMLMAPPER",生成SQLMap XML文件和独立的Mapper接口  -->   <!-- targetPackage：mapper接口dao生成的位置 -->  <!--<javaClientGenerator type="XMLMAPPER" targetPackage="com.tmall.dao" targetProject=".\src\main\java">-->  <javaClientGenerator type="XMLMAPPER" targetPackage="com.tmall.dao" targetProject="./src/main/java">  <!-- enableSubPackages:是否让schema作为包的后缀 -->  <property name="enableSubPackages" value="false" />  </javaClientGenerator>    <table tableName="tmall\_shipping" domainObjectName="Shipping" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"></table>  <table tableName="tmall\_cart" domainObjectName="Cart" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"></table>  <table tableName="tmall\_cart\_item" domainObjectName="CartItem" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"></table>  <table tableName="tmall\_category" domainObjectName="Category" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"></table>  <table tableName="tmall\_order" domainObjectName="Order" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"></table>  <table tableName="tmall\_order\_item" domainObjectName="OrderItem" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"></table>  <table tableName="tmall\_pay\_info" domainObjectName="PayInfo" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"></table>  <table tableName="tmall\_product" domainObjectName="Product" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false">  <columnOverride column="detail" jdbcType="VARCHAR" />  <columnOverride column="sub\_images" jdbcType="VARCHAR" />  </table>  <table tableName="tmall\_user" domainObjectName="User" enableCountByExample="false" enableUpdateByExample="false" enableDeleteByExample="false" enableSelectByExample="false" selectByExampleQueryId="false"></table>    <!-- geelynote mybatis插件的搭建 -->  </context> </generatorConfiguration> |

##### 2.5.3 配置datasource.properties

在resources目录下新建datesource.properties文件并配置

|  |
| --- |
| db.driverLocation=D:/PrograSoftware/Maven/repository/mysql/mysql-connector-java/5.1.38/mysql-connector-java-5.1.38.jar db.driverClassName=com.mysql.jdbc.Driver db.url=jdbc:mysql://localhost:3306/test\_tmall?characterEncoding=utf-8 db.username=root db.password=root   db.initialSize = 20 db.maxActive = 50 db.maxIdle = 20 db.minIdle = 10 db.maxWait = 10 db.defaultAutoCommit = true db.minEvictableIdleTimeMillis = 3600000 |

##### 2.5.4 执行脚本生成文件

点击idea右侧maven project，选择mybatis-generator，双击执行，生成pojo和dao相关类以及xml文件

#### 2.6 配置mybatis-plugin

##### 2.6.1 下载安装mybatis plugin插件

File->settings->plugin->browse repositories，搜索mybatis plugin并安装

#### 2.7 配置项目开发环境

##### 2.7.1 配置spring环境

（1）在resources目录下新建applicationContext.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:aop="http://www.springframework.org/schema/aop"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop.xsd">   <!-- 注解扫描 -->  <context:component-scan base-package="com.tmall" annotation-config="true"/>   <!-- AOP -->  <!--<context:annotation-config/>-->  <aop:aspectj-autoproxy/>   <!-- 引入外部文件 -->  <import resource="applicationContext-datasource.xml"/>  </beans> |

（2）在resources目录下新建applicationContext-datasource.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx.xsd">   <context:component-scan base-package="com.tmall" annotation-config="true"/>   <bean id="propertyConfigurer"  class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">  <property name="order" value="2"/>  <property name="ignoreUnresolvablePlaceholders" value="true"/>  <property name="locations">  <list>  <value>classpath:datasource.properties</value>  </list>  </property>  <property name="fileEncoding" value="utf-8"/>  </bean>    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">  <property name="driverClassName" value="${db.driverClassName}"/>  <property name="url" value="${db.url}"/>  <property name="username" value="${db.username}"/>  <property name="password" value="${db.password}"/>  <!-- 连接池启动时的初始值 -->  <property name="initialSize" value="${db.initialSize}"/>  <!-- 连接池的最大值 -->  <property name="maxActive" value="${db.maxActive}"/>  <!-- 最大空闲值.当经过一个高峰时间后，连接池可以慢慢将已经用不到的连接慢慢释放一部分，一直减少到maxIdle为止 -->  <property name="maxIdle" value="${db.maxIdle}"/>  <!-- 最小空闲值.当空闲的连接数少于阀值时，连接池就会预申请去一些连接，以免洪峰来时来不及申请 -->  <property name="minIdle" value="${db.minIdle}"/>  <!-- 最大建立连接等待时间。如果超过此时间将接到异常。设为－1表示无限制 -->  <property name="maxWait" value="${db.maxWait}"/>  <!--#给出一条简单的sql语句进行验证 -->  <!--<property name="validationQuery" value="select getdate()" />-->  <property name="defaultAutoCommit" value="${db.defaultAutoCommit}"/>  <!-- 回收被遗弃的（一般是忘了释放的）数据库连接到连接池中 -->  <!--<property name="removeAbandoned" value="true" />-->  <!-- 数据库连接过多长时间不用将被视为被遗弃而收回连接池中 -->  <!--<property name="removeAbandonedTimeout" value="120" />-->  <!-- #连接的超时时间，默认为半小时。 -->  <property name="minEvictableIdleTimeMillis" value="${db.minEvictableIdleTimeMillis}"/>   <!--# 失效检查线程运行时间间隔，要小于MySQL默认-->  <property name="timeBetweenEvictionRunsMillis" value="40000"/>  <!--# 检查连接是否有效-->  <property name="testWhileIdle" value="true"/>  <!--# 检查连接有效性的SQL语句-->  <property name="validationQuery" value="SELECT 1 FROM dual"/>  </bean>   <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  <property name="dataSource" ref="dataSource"/>  <property name="mapperLocations" value="classpath\*:mappers/\*Mapper.xml"></property>   <!-- 分页插件 -->  <property name="plugins">  <array>  <bean class="com.github.pagehelper.PageHelper">  <property name="properties">  <value>  dialect=mysql  </value>  </property>  </bean>  </array>  </property>   </bean>   <bean name="mapperScannerConfigurer" class="org.mybatis.spring.mapper.MapperScannerConfigurer">  <property name="basePackage" value="com.tmall.dao"/>  </bean>   <!-- 使用@Transactional进行声明式事务管理需要声明下面这行 -->  <tx:annotation-driven transaction-manager="transactionManager" proxy-target-class="true" />  <!-- 事务管理 -->  <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  <property name="dataSource" ref="dataSource"/>  <property name="rollbackOnCommitFailure" value="true"/>  </bean>  </beans> |

##### 2.7.2 配置spring-mvc环境

（1）在WEB-INF目录下新建dispatcher-servlet.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc.xsd">   <context:component-scan base-package="com.tmall" annotation-config="true"/>   <mvc:annotation-driven>  <mvc:message-converters>  <bean class="org.springframework.http.converter.StringHttpMessageConverter">  <property name="supportedMediaTypes">  <list>  <value>text/plain;charset=UTF-8</value>  <value>text/html;charset=UTF-8</value>  </list>  </property>  </bean>  <bean class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter">  <property name="supportedMediaTypes">  <list>  <value>application/json;charset=UTF-8</value>  </list>  </property>  </bean>  </mvc:message-converters>  </mvc:annotation-driven>     <!-- 文件上传 -->  <bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">  <property name="maxUploadSize" value="10485760"/> <!-- 10m -->  <property name="maxInMemorySize" value="4096" />  <property name="defaultEncoding" value="UTF-8"></property>  </bean>   </beans> |

（2）在WEB-INF目录下编辑web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://java.sun.com/xml/ns/javaee"  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5">   <display-name>Archetype Created Web Application</display-name>   <filter>  <filter-name>characterEncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  <init-param>  <param-name>forceEncoding</param-name>  <param-value>true</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>characterEncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>   <!-- web容器启动和关闭监听 -->  <listener>  <listener-class>org.springframework.web.context.request.RequestContextListener</listener-class>  </listener>   <!-- web容器和spring容器整合监听 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>  classpath:applicationContext.xml  </param-value>  </context-param>   <!-- spring mvc -->  <servlet>  <servlet-name>dispatcher</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <!-- 容器启动时初始化servlet -->  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>dispatcher</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping>  </web-app> |

##### 2.7.3 配置logback环境

在resources目录下新建logback.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <configuration scan="true" scanPeriod="60 seconds" debug="false">  <appender name="console" class="ch.qos.logback.core.ConsoleAppender">  <encoding>UTF-8</encoding>  <encoder>  <pattern>[%d{HH:mm:ss.SSS}][%p][%c{40}][%t] %m%n</pattern>  </encoder>  <filter class="ch.qos.logback.classic.filter.ThresholdFilter">  <level>DEBUG</level>  </filter>  </appender>   <appender name="tmall" class="ch.qos.logback.core.rolling.RollingFileAppender">  <!--<File>d:/tmalllog/tmall.log</File>-->  <File>/PrograSoftware/Tomcat/apache-tomcat-7.0.69/logs/mmall.log</File>  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  <fileNamePattern>/PrograSoftware/Tomcat/apache-tomcat-7.0.69/logs/tmall.log.%d{yyyy-MM-dd}.gz</fileNamePattern>  <append>true</append>  <maxHistory>10</maxHistory>  </rollingPolicy>  <encoder>  <pattern>[%d{HH:mm:ss.SSS}][%p][%c{40}][%t] %m%n</pattern>  </encoder>  </appender>    <appender name="error" class="ch.qos.logback.core.rolling.RollingFileAppender">  <!--<File>d:/mmalllog/error.log</File>-->  <File>/PrograSoftware/Tomcat/apache-tomcat-7.0.69/logs/error.log</File>  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  <fileNamePattern>/PrograSoftware/Tomcat/apache-tomcat-7.0.69/logs/error.log.%d{yyyy-MM-dd}.gz</fileNamePattern>  <!--<fileNamePattern>d:/mmalllog/error.log.%d{yyyy-MM-dd}.gz</fileNamePattern>-->  <append>true</append>  <maxHistory>10</maxHistory>  </rollingPolicy>  <encoder>  <pattern>[%d{HH:mm:ss.SSS}][%p][%c{40}][%t] %m%n</pattern>  </encoder>  <filter class="ch.qos.logback.classic.filter.LevelFilter">  <level>ERROR</level>  <onMatch>ACCEPT</onMatch>  <onMismatch>DENY</onMismatch>  </filter>  </appender>   <logger name="com.tmall" additivity="false" level="INFO" >  <appender-ref ref="tmall" />  <appender-ref ref="console"/>  </logger>     <!-- geelynote mybatis log 日志 -->   <logger name="com.tmall.dao" level="DEBUG"/>   <!--<logger name="com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate" level="DEBUG" >-->  <!--<appender-ref ref="console"/>-->  <!--</logger>-->   <!--<logger name="java.sql.Connection" level="DEBUG">-->  <!--<appender-ref ref="console"/>-->  <!--</logger>-->  <!--<logger name="java.sql.Statement" level="DEBUG">-->  <!--<appender-ref ref="console"/>-->  <!--</logger>-->   <!--<logger name="java.sql.PreparedStatement" level="DEBUG">-->  <!--<appender-ref ref="console"/>-->  <!--</logger>-->    <root level="DEBUG">  <appender-ref ref="console"/>  <appender-ref ref="error"/>  </root>  </configuration> |

##### 2.7.4 配置IDEA注入和自动编译环境

（1）IDEA注入配置

file->settings->editor->inspections->spring->spring core->wutowiring for bean class

（2）IDEA自动编译

file->setting->build,execution,deployment->complier->build project automaticlly

### 三 数据库表设计

### 四 用户模块

#### 4.1 用户登录

##### 4.1.1 公共类

（1）服务器响应类，将结果序列化之后返回给前端

|  |
| --- |
| package com.tmall.common;  import org.codehaus.jackson.annotate.JsonIgnore; import org.codehaus.jackson.map.annotate.JsonSerialize;  import java.io.Serializable;  */\*\*  \* 服务器响应类，将结果序列化返回  \** ***@param*** <*T*>  *\*/*  // 保证序列化json的时候，如果是null对象，key也会消失@JsonSerialize(include = JsonSerialize.Inclusion.*NON\_NULL*) public class ServerResponse<T> implements Serializable {   private int status;   private String msg;   private T data;   private ServerResponse(int status) {  this.status = status;  }   private ServerResponse(int status, T data) {  this.status = status;  this.data = data;  }   private ServerResponse(int status, String msg, T data) {  this.status = status;  this.msg = msg;  this.data = data;  }   private ServerResponse(int status, String msg) {  this.status = status;  this.msg = msg;  }   @JsonIgnore// 使之不在json序列化结果中  public boolean isSuccess() {  return this.status == ResponseCode.*SUCCESS*.getCode();  }   public int getStatus() {  return status;  }   public String getMsg() {  return msg;  }   public T getData() {  return data;  }   public static <T> ServerResponse<T> createBySuccess() {  return new ServerResponse<T>(ResponseCode.*SUCCESS*.getCode());  }   public static <T> ServerResponse<T> createBySuccessMessage(String msg) {  return new ServerResponse<T>(ResponseCode.*SUCCESS*.getCode(), msg);  }   public static <T> ServerResponse<T> createBySuccess(T data) {  return new ServerResponse<T>(ResponseCode.*SUCCESS*.getCode(), data);  }   public static <T> ServerResponse<T> createBySuccess(String msg, T data) {  return new ServerResponse<T>(ResponseCode.*SUCCESS*.getCode(), msg, data);  }   public static <T> ServerResponse<T> createByError() {  return new ServerResponse<T>(ResponseCode.*ERROR*.getCode(), ResponseCode.*ERROR*.getDesc());  }   public static <T> ServerResponse<T> createByErrorMessage(String errorMessage) {  return new ServerResponse<T>(ResponseCode.*ERROR*.getCode(), errorMessage);  }   public static <T> ServerResponse<T> createByErrorCodeMessage(int errorCode, String errorMessage) {  return new ServerResponse<T>(errorCode, errorMessage);  } } |

（2）枚举类，列出所有的状态

|  |
| --- |
| package com.tmall.common;  */\*\*  \* 枚举类，定义各种状态  \*/* public enum ResponseCode {   *SUCCESS*(0, "SUCCESS"),  *ERROR*(1, "ERROR"),  *NEED\_LOGIN*(10, "NEED\_LOGIN"),  *ILLEGAL\_ARGUMENT*(2, "ILLEGAL\_ARGUMENT");   private final int code;   private final String desc;   ResponseCode(int code, String desc) {  this.code = code;  this.desc = desc;  }   public int getCode() {  return code;  }   public String getDesc() {  return desc;  } } |

（3）常量类，把通用的字段定义为常量

|  |
| --- |
| package com.tmall.common;  /\*\*  \*常量类  \*/ public class Const {  */\*\*  \* 当前用户  \*/*  public static final String *CURRENT\_USER* = "currentUser";  } |

##### 4.1.2 DAO层

数据访问层，封装对数据库的访问

|  |
| --- |
| package com.tmall.dao;  import com.tmall.pojo.User; import org.apache.ibatis.annotations.Param;  */\*\*  \* 用户表  \*/* public interface UserMapper {   */\*\*  \* 根据主键删除用户  \** ***@param*** *id  \** ***@return*** *\*/* int deleteByPrimaryKey(Integer id);   */\*\*  \* 插入用户表所有字段  \** ***@param*** *record  \** ***@return*** *\*/* int insert(User record);   */\*\*  \* 插入用户表非空字段  \** ***@param*** *record  \** ***@return*** *\*/* int insertSelective(User record);   */\*\*  \* 根据主键查询用户  \** ***@param*** *id  \** ***@return*** *\*/* User selectByPrimaryKey(Integer id);   */\*\*  \* 更新用户表非空字段  \** ***@param*** *record  \** ***@return*** *\*/* int updateByPrimaryKeySelective(User record);   */\*\*  \* 更新用户表所有字段  \** ***@param*** *record  \** ***@return*** *\*/* int updateByPrimaryKey(User record);   */\*\*  \* 校验用户名是否存在  \** ***@param*** *username  \** ***@return*** *\*/* int checkUsername(String username);   */\*\*  \* 校验登录时用户名和密码是否正确  \** ***@param*** *username  \** ***@param*** *password  \** ***@return*** *\*/* User selectLogin(@Param("username") String username, @Param("password") String password);  } |

##### 4.1.3 数据库

在UserMapper.xml文件中编写数据库语句

|  |
| --- |
| <select id="checkUsername" resultType="int" parameterType="string">  SELECT *count*(1)  FROM tmall\_user  WHERE username = #{username} </select>  <select id="selectLogin" resultMap="BaseResultMap" parameterType="map">  SELECT <include refid="Base\_Column\_List" />  FROM tmall\_user  WHERE username = #{username}  and password = #{password} </select> |

##### 4.1.4 Service层

Service层用于处理各种业务逻辑

（1）接口类

|  |
| --- |
| package com.tmall.service;  import com.tmall.common.ServerResponse; import com.tmall.pojo.User;  */\*\*  \* 用户接口  \*/* public interface IUserService {   */\*\*  \* 用户登录  \** ***@param*** *username  \** ***@param*** *password  \** ***@return*** *\*/* ServerResponse<User> login(String username, String password); } |

（2）接口的实现类

|  |
| --- |
| package com.tmall.service.impl;  import com.tmall.common.ServerResponse; import com.tmall.dao.UserMapper; import com.tmall.pojo.User; import com.tmall.service.IUserService; import org.apache.commons.lang3.StringUtils; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Service;  */\*\*  \* 用户接口实现类  \*/* @Service("iUserService") public class UserServiceImpl implements IUserService {   @Autowired  private UserMapper userMapper;   */\*\*  \* 用户登录  \** ***@param*** *username  \** ***@param*** *password  \** ***@return*** *\*/* @Override  public ServerResponse<User> login(String username, String password) {  int resultCount = userMapper.checkUsername(username);  if (resultCount == 0) {  return ServerResponse.*createByErrorMessage*("用户名不存在");  }  // 密码MD5加密登录 *TODO* User user = userMapper.selectLogin(username, password);  if (user == null) {  return ServerResponse.*createByErrorMessage*("用户名或密码错误");  }  // 密码置空  user.setPassword(StringUtils.*EMPTY*);   return ServerResponse.*createBySuccess*("登录成功", user);  } } |

##### 4.1.5 Controller层

Controller层，负责从视图读取数据，控制用户输入并向模型发送数据

|  |
| --- |
| package com.tmall.controller.portal;  import com.tmall.common.Const; import com.tmall.common.ServerResponse; import com.tmall.pojo.User; import com.tmall.service.IUserService; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Controller; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RequestMethod; import org.springframework.web.bind.annotation.ResponseBody;  import javax.servlet.http.HttpSession;  */\*\*  \* 用户控制  \*/* @Controller @RequestMapping("/user/") public class UserController {   @Autowired  private IUserService iUserService;   */\*\*  \* 登录  \** ***@param*** *username  \** ***@param*** *password  \** ***@param*** *session  \** ***@return*** *\*/* @RequestMapping(value = "login.do", method = RequestMethod.*POST*)  @ResponseBody// 通过Jackson将返回的结果序列化成json  public ServerResponse<User> login(String username, String password, HttpSession session) {  ServerResponse response = iUserService.login(username, password);  if (response.isSuccess()) {  session.setAttribute(Const.*CURRENT\_USER*, response.getData());  }  return response;  }  } |

#### 4.2 用户退出

##### 4.2.1 Controller层

|  |
| --- |
| */\*\*  \* 用户退出  \** ***@param*** *session  \** ***@return*** *\*/* @RequestMapping(value = "logout.do", method = RequestMethod.*POST*) @ResponseBody public ServerResponse<String> logout(HttpSession session) {  session.removeAttribute(Const.*CURRENT\_USER*);  return ServerResponse.*createBySuccess*(); } |

#### 4.3 用户注册

##### 4.3.1 公共类

（1）添加MD5加密工具类

|  |
| --- |
| package com.tmall.util;  import java.security.MessageDigest;  */\*\*  \* MD5加密算法  \*/* public class MD5Util {   private static String byteArrayToHexString(byte b[]) {  StringBuffer resultSb = new StringBuffer();  for (int i = 0; i < b.length; i++)  resultSb.append(*byteToHexString*(b[i]));   return resultSb.toString();  }   private static String byteToHexString(byte b) {  int n = b;  if (n < 0)  n += 256;  int d1 = n / 16;  int d2 = n % 16;  return *hexDigits*[d1] + *hexDigits*[d2];  }   */\*\*  \* 返回大写MD5  \*  \** ***@param*** *origin  \** ***@param*** *charsetname  \** ***@return*** *\*/* private static String MD5Encode(String origin, String charsetname) {  String resultString = null;  try {  resultString = new String(origin);  MessageDigest md = MessageDigest.*getInstance*("MD5");  if (charsetname == null || "".equals(charsetname))  resultString = *byteArrayToHexString*(md.digest(resultString.getBytes()));  else  resultString = *byteArrayToHexString*(md.digest(resultString.getBytes(charsetname)));  } catch (Exception exception) {  }  return resultString.toUpperCase();  }   public static String MD5EncodeUtf8(String origin) {  origin = origin + PropertiesUtil.*getProperty*("password.salt", "");  return *MD5Encode*(origin, "utf-8");  }    private static final String *hexDigits*[] = {"0", "1", "2", "3", "4", "5",  "6", "7", "8", "9", "a", "b", "c", "d", "e", "f"};  } |

（2）添加配置文件读取类

|  |
| --- |
| package com.tmall.util;  import org.apache.commons.lang3.StringUtils; import org.slf4j.Logger; import org.slf4j.LoggerFactory; import java.io.IOException; import java.io.InputStreamReader; import java.util.Properties;  public class PropertiesUtil {   private static Logger *logger* = LoggerFactory.*getLogger*(PropertiesUtil.class);   private static Properties *props*;   static {  String fileName = "tmall.properties";  *props* = new Properties();  try {  *props*.load(new InputStreamReader(PropertiesUtil.class.getClassLoader().getResourceAsStream(fileName),"UTF-8"));  } catch (IOException e) {  *logger*.error("配置文件读取异常",e);  }  }   public static String getProperty(String key){  String value = *props*.getProperty(key.trim());  if(StringUtils.*isBlank*(value)){  return null;  }  return value.trim();  }   public static String getProperty(String key,String defaultValue){  String value = *props*.getProperty(key.trim());  if(StringUtils.*isBlank*(value)){  value = defaultValue;  }  return value.trim();  } } |

（3）在Const类中添加用户角色

|  |
| --- |
| */\*\*  \* 用户角色  \*/* public interface Role {  // 普通用户  int *ROLE\_CUSTOMER* = 0;   // 管理员  int *ROLE\_DMIN* = 1; } |

##### 4.3.2 DAO层

在UserMapper中添加邮箱校验

|  |
| --- |
| *\*\*  \* 校验用户邮箱  \** ***@param*** *email  \** ***@return*** *\*/* int checkEmail(String email); |

##### 4.3.3 数据库

在UserMapper.xml文件中添加数据库语句

|  |
| --- |
| <select id="checkEmail" resultType="int" parameterType="string">  SELECT *count*(1)  FROM tmall\_user  WHERE email = #{email} </select> |

##### 4.3.4 Service层

（1）在IUserService接口中添加注册方法

|  |
| --- |
| */\*\*  \* 用户注册  \** ***@param*** *user  \** ***@return*** *\*/* ServerResponse<String> register(User user); |

（2）在UserServiceImpl中实现接口中的注册方法

|  |
| --- |
| */\*\*  \* 用户注册  \** ***@param*** *user  \** ***@return*** *\*/* public ServerResponse<String> register(User user) {  ServerResponse validResponse = this.checkValid(user.getUsername(), Const.*USERNAME*);  if (!validResponse.isSuccess()) {  return validResponse;  }  validResponse = this.checkValid(user.getEmail(), Const.*EMAIL*);  if (!validResponse.isSuccess()) {  return validResponse;  }  user.setRole(Const.Role.*ROLE\_CUSTOMER*);  // MD5加密  user.setPassword(MD5Util.*MD5EncodeUtf8*(user.getPassword()));  int resultCount = userMapper.insert(user);  if (resultCount == 0) {  return ServerResponse.*createByErrorMessage*("注册失败");  }   return ServerResponse.*createBySuccessMessage*("注册成功"); } |

##### 4.3.5 Controller层

在Controller中调用Servicede的注册方法

|  |
| --- |
| */\*\*  \* 用户注册  \** ***@param*** *user  \** ***@return*** *\*/* @RequestMapping(value = "register.do", method = RequestMethod.*POST*) @ResponseBody public ServerResponse<String> register(User user) {  return iUserService.register(user); } |

#### 4.4 用户名和邮箱校验

##### 4.4.1 公共类

在Const类中添加常量

|  |
| --- |
| */\*\*  \* 邮箱  \*/* public static final String *EMAIL* = "email";  */\*\*  \* 用户名  \*/* public static final String *USERNAME* = "username"; |

##### 4.4.2 Service层

（1）在IUserService接口中添加校验方法

|  |
| --- |
| */\*\*  \* 校验用户名和邮箱  \** ***@param*** *str  \** ***@param*** *type  \** ***@return*** *\*/* ServerResponse<String> checkValid(String str, String type); |

（2）在UserServiceImpl中实现邮箱校验

|  |
| --- |
| */\*\*  \* 校验用户名和邮箱  \** ***@param*** *str  \** ***@param*** *type  \** ***@return*** *\*/* public ServerResponse<String> checkValid(String str, String type) {  if (StringUtils.*isNotBlank*(type)) {  if (Const.*USERNAME*.equals(type)) {  int resultCount = userMapper.checkUsername(str);  if (resultCount > 0) {  return ServerResponse.*createByErrorMessage*("用户名已存在");  }  }  if (Const.*EMAIL*.equals(type)) {  int resultCount = userMapper.checkEmail(str);  if (resultCount > 0) {  return ServerResponse.*createByErrorMessage*("邮箱已被注册");  }  }  } else {  return ServerResponse.*createByErrorMessage*("参数错误");  }   return ServerResponse.*createBySuccessMessage*("校验成功"); } |

##### 4.4.3 Controller层

在Controller中调用Service的校验方法

|  |
| --- |
| */\*\*  \* 校验用户名和邮箱  \** ***@param*** *str  \** ***@param*** *type  \** ***@return*** *\*/* @RequestMapping(value = "check\_valid.do", method = RequestMethod.*POST*) @ResponseBody public ServerResponse<String> checkValid(String str, String type) {  return iUserService.checkValid(str, type); } |

#### 4.5 用户忘记密码

##### 4.5.1 DAO层

在UserMapper类中添加根据用户名查找问题的方法

|  |
| --- |
| */\*\*  \* 根据用户名查找问题  \** ***@param*** *username  \** ***@return*** *\*/* String selectQuestionByUsername(String username); |

##### 4.5.2 数据库

在UserMapper.xml文件中添加查询语句

|  |
| --- |
| <select id="selectQuestionByUsername" resultType="string" parameterType="string">  SELECT question  FROM tmall\_user  WHERE username = #{username} </select> |

##### 4.5.3 Service层

（1）在IUserService接口中添加找回密码问题的方法

|  |
| --- |
| */\*\*  \* 找回密码的提示问题  \** ***@param*** *username  \** ***@return*** *\*/* ServerResponse<String> selectQuestion(String username); |

（2）在UserServiceImpl类中实现该方法

|  |
| --- |
| */\*\*  \* 找回密码的提示问题  \** ***@param*** *username  \** ***@return*** *\*/* public ServerResponse<String> selectQuestion(String username) {  ServerResponse validResponse = this.checkValid(username, Const.*USERNAME*);  if (validResponse.isSuccess()) {  return ServerResponse.*createByErrorMessage*("用户不存在");  }  String question = userMapper.selectQuestionByUsername(username);  if (StringUtils.*isNotBlank*(question)) {  return ServerResponse.*createBySuccess*(question);  }  return ServerResponse.*createByErrorMessage*("找回密码问题的答案为空"); } |

##### 4.5.4 Controller层

在Controller中调用service

|  |
| --- |
| */\*\*  \* 忘记密码  \** ***@param*** *username  \** ***@return*** *\*/* @RequestMapping(value = "forget\_get\_question.do", method = RequestMethod.*POST*) @ResponseBody public ServerResponse<String> forgetGetQuestion(String username) {  return iUserService.selectQuestion(username); } |

#### 4.6 用户重置密码问题校验

##### 4.6.1 公共类

构建本地缓存

|  |
| --- |
| package *com.tmall.common*;  import *com.google.common.cache.CacheBuilder*; import *com.google.common.cache.CacheLoader*; import *com.google.common.cache.LoadingCache*; import *org.slf4j.Logger*; import *org.slf4j.LoggerFactory*;  import *java.util.concurrent.TimeUnit*;  /\*\*  \* 本地缓存  \*/ public class *TokenCache* {   private static *Logger* logger = *LoggerFactory*.getLogger(*TokenCache*.class);   public static final *String* ***TOKEN\_PREFIX*** = "token\_";   private static *LoadingCache*<*String*, *String*> localCache = *CacheBuilder*.newBuilder()  .initialCapacity(1000).maximumSize(10000).expireAfterAccess(12, *TimeUnit*.***HOURS***)  .build(new *CacheLoader*<*String*, *String*>() {  //默认的数据加载实现,当调用get取值的时候,如果key没有对应的值,就调用这个方法进行加载.  @Override  public *String* load(*String s*) throws *Exception* {  return "null";  }  });   public static void setKey(*String key*, *String value*) {  localCache.put(*key*, *value*);  }   public static *String* getKey(*String key*) {  *String* value = null;  try {  value = localCache.get(*key*);  if ("null".equals(value)) {  return null;  }  return value;  } catch (*Exception e*) {  logger.error("localCache get error", *e*);  }  return null;  }  } |

##### 4.6.2 DAO层

在userMapper中添加校验重置密码问题的答案

|  |
| --- |
| /\*\*  \* 校验提示问题的答案  \* @param username  \* @param question  \* @param answer  \* @return  \*/ int checkAnswer(@Param("username") *String username*, @Param("question") *String question*, @Param("answer") *String answer*); |

##### 4.6.3 数据库

在userMapper.xml中添加查询语句

|  |
| --- |
| <select *id*="checkAnswer" *resultType*="int" *parameterType*="map">  SELECT count(1)  FROM tmall\_user  WHERE username = #{username}  and question = #{question}  and answer = #{answer} </select> |

##### 4.6.4 Service层

（1）在IUserService接口中添加校验重置密码问题答案的方法

|  |
| --- |
| /\*\*  \* 提示问题与答案  \* @param username  \* @param question  \* @param answer  \* @return  \*/ *ServerResponse*<*String*> checkAnswer(*String username*, *String question*, *String answer*); |

（2）在UserServiceImpl类中实现该方法

|  |
| --- |
| /\*\*  \* 问题答案校验  \* @param username  \* @param question  \* @param answer  \* @return  \*/ public *ServerResponse*<*String*> checkAnswer(*String username*, *String question*, *String answer*) {  int resultCount = userMapper.checkAnswer(*username*, *question*, *answer*);  if (resultCount > 0) {  // 说明问题及问题答案是这个用户的并且是正确的  *String* forgetToken = *UUID*.randomUUID().toString();  *TokenCache*.setKey(*TokenCache*.***TOKEN\_PREFIX***+*username*, forgetToken);  return *ServerResponse*.createBySuccess(forgetToken);  }  return *ServerResponse*.createByErrorMessage("问题答案错误"); } |

##### 4.6.5 Controller层

在Controller中调用service

|  |
| --- |
| /\*\*  \* 校验问题答案  \* @param username  \* @param question  \* @param answer  \* @return  \*/ @RequestMapping(value = "forget\_check\_answer.do", method = *RequestMethod*.***POST***) @ResponseBody public *ServerResponse*<*String*> forgetCheckAnswer(*String username*, *String question*, *String answer*) {  return iUserService.checkAnswer(*username*, *question*, *answer*); } |

#### 4.7 用户忘记密码状态下重置密码

##### 4.7.1 DAO层

在userMapper中添加校根据用户名更新密码

|  |
| --- |
| /\*\*  \* 根据用户名更新密码  \* @param *username* \* @param passwordNew  \* @return  \*/ int updatePasswordByUsername(@Param("username") *String username*, @Param("passwordNew") *String passwordNew*); |

##### 4.7.2 数据库

在userMapper.xml中添加更新语句

|  |
| --- |
| <update *id*="updatePasswordByUsername" *parameterType*="map">  UPDATE tmall\_user  SET password = #{passwordNew}, update\_time = now()  WHERE username = #{username} </update> |

##### 4.7.3 Service层

（1）在IUserService接口中添加忘记密码状态的重置密码的方法

|  |
| --- |
| /\*\*  \* 重置密码（忘记密码）  \* @param username  \* @param passwordNew  \* @param forgetToken  \* @return  \*/ *ServerResponse*<*String*> forgetResetPassword(*String username*, *String passwordNew*, *String forgetToken*); |

（2）在UserServiceImpl类中实现该方法

|  |
| --- |
| /\*\*  \* 重置密码（忘记密码）  \* @param username  \* @param passwordNew  \* @param forgetToken  \* @return  \*/ public *ServerResponse*<*String*> forgetResetPassword(*String username*, *String passwordNew*, *String forgetToken*) {  if (*StringUtils*.isBlank(*forgetToken*)) {  return *ServerResponse*.createByErrorMessage("参数错误，token需要传递");  }  *ServerResponse* validResponse = this.checkValid(*username*, *Const*.***USERNAME***);  if (validResponse.isSuccess()) {  return *ServerResponse*.createByErrorMessage("用户不存在");  }  *String* token = *TokenCache*.getKey(*TokenCache*.***TOKEN\_PREFIX***+*username*);  if (*StringUtils*.isBlank(token)) {  return *ServerResponse*.createByErrorMessage("token无效或者过期");  }  if (*StringUtils*.equals(*forgetToken*, token)) {  *String* md5Password = *MD5Util*.MD5EncodeUtf8(*passwordNew*);  int count = userMapper.updatePasswordByUsername(*username*, md5Password);  if (count > 0) {  return *ServerResponse*.createBySuccessMessage("修改密码成功");  }  } else {  return *ServerResponse*.createByErrorMessage("token错误，请重新获取修改密码的token");  }  return *ServerResponse*.createByErrorMessage("修改密码失败"); } |

##### 4.7.4 Controller层

在Controller中调用service

|  |
| --- |
| /\*\*  \* 重置密码  \* @param *username* \* @param passwordNew  \* @param forgetToken  \* @return  \*/ @RequestMapping(value = "forget\_reset\_password.do", method = *RequestMethod*.***POST***) @ResponseBody public *ServerResponse*<*String*> forgetResetPassword(*String username*, *String passwordNew*, *String forgetToken*) {  return iUserService.forgetResetPassword(*username*, *passwordNew*, *forgetToken*); } |

#### 4.8 用户登录状态下重置密码

##### 4.8.1 DAO层

在userMapper中添加校验当前密码方法

|  |
| --- |
| /\*\*  \* 校验当前密码  \* @param password  \* @param userId  \* @return  \*/ int checkPassword(@Param("password") *String password*, @Param("userId") *Integer userId*); |

##### 4.8.2 数据库

在userMapper.xml中添加查询语句

|  |
| --- |
| <select *id*="checkPassword" *parameterType*="map" *resultType*="int">  SELECT count(1)  FROM tmall\_user  WHERE password = #{password}  and id = #{userId} </select> |

##### 4.8.3 Service层

（1）在IUserService接口中添加登录状态的重置密码的方法

|  |
| --- |
| /\*\*  \* 重置密码（登录状态）  \* @param passwordOld  \* @param passwordNew  \* @param user  \* @return  \*/ *ServerResponse*<*String*> resetPassword(*String passwordOld*, *String passwordNew*, *User user*); |

（2）在UserServiceImpl类中实现该方法

|  |
| --- |
| /\*\*  \* 重置密码（登录状态）  \* @param passwordOld  \* @param passwordNew  \* @param user  \* @return  \*/ public *ServerResponse*<*String*> resetPassword(*String passwordOld*, *String passwordNew*, *User user*) {  // 防止横向越权,要校验一下这个用户的旧密码,一定要指定是这个用户.因为我们会查询一个count(1),如果不指定id,那么结果就是true啦count>0;  int resultCount = userMapper.checkPassword(*MD5Util*.MD5EncodeUtf8(*passwordOld*), *user*.getId());  if (resultCount == 0) {  return *ServerResponse*.createByErrorMessage("密码错误");  }  *user*.setPassword(*MD5Util*.MD5EncodeUtf8(*passwordNew*));  int updateCount = userMapper.updateByPrimaryKeySelective(*user*);  if (updateCount > 0) {  return *ServerResponse*.createBySuccessMessage("修改密码成功");  }  return *ServerResponse*.createByErrorMessage("重置密码失败"); } |

##### 4.8.4 Controller层

在Controller中调用service

|  |
| --- |
| /\*\*  \* 重置密码  \* @param session  \* @param passwordOld  \* @param passwordNew  \* @return  \*/ @RequestMapping(value = "reset\_password.do", method = *RequestMethod*.***POST***) @ResponseBody public *ServerResponse*<*String*> resetPassword(*HttpSession session*, *String passwordOld*, *String passwordNew*) {  *User* user = (*User*) *session*.getAttribute(*Const*.***CURRENT\_USER***);  if (user == null) {  return *ServerResponse*.createByErrorMessage("用户未登录");  }  return iUserService.resetPassword(*passwordOld*, *passwordNew*, user); } |

#### 4.9 用户信息更新

##### 4.9.1 DAO层

在userMapper中添加校验根据用户ID校验邮箱方法

|  |
| --- |
| /\*\*  \* 根据用户ID校验邮箱  \* @param email  \* @param userId  \* @return  \*/ int checkEmailByUserId(@Param("email") *String email*, @Param("userId") *Integer userId*); |

##### 4.9.2 数据库

在userMapper.xml中添加查询语句

|  |
| --- |
| <select *id*="checkEmailByUserId" *parameterType*="map" *resultType*="int">  SELECT count(1)  FROM tmall\_user  WHERE email = *#{email}* and id != #{userId} </select> |

##### 4.9.3 Service层

（1）在IUserService接口中添加更新用户信息的方法

|  |
| --- |
| /\*\*  \* 更新个人信息  \* @param user  \* @return  \*/ *ServerResponse*<*User*> updateInformation(*User user*); |

（2）在UserServiceImpl类中实现该方法

|  |
| --- |
| /\*\*  \* 更新个人信息  \* @param user  \* @return  \*/ public *ServerResponse*<*User*> updateInformation(*User user*) {  // email也要进行一个校验,校验新的email是不是已经存在,并且存在的email如果相同的话,不能是我们当前的这个用户的  int resultCount = userMapper.checkEmailByUserId(*user*.getEmail(), *user*.getId());  if (resultCount > 0) {  return *ServerResponse*.createByErrorMessage("email已被注册");  }  *User* updateUser = new User();  updateUser.setId(*user*.getId());  updateUser.setEmail(*user*.getEmail());  updateUser.setPhone(*user*.getPhone());  updateUser.setQuestion(*user*.getQuestion());  updateUser.setAnswer(*user*.getAnswer());  int updateCount = userMapper.updateByPrimaryKeySelective(updateUser);  if (updateCount > 0) {  return *ServerResponse*.createBySuccess("更新个人信息成功", updateUser);  }  return *ServerResponse*.createByErrorMessage("更新个人信息失败"); } |

##### 4.9.4 Controller层

在Controller中调用service

|  |
| --- |
| /\*\*  \* 更新个人信息  \* @param session  \* @param user  \* @return  \*/ @RequestMapping(value = "update\_information.do", method = *RequestMethod*.***POST***) @ResponseBody public *ServerResponse*<*User*> updateInformation(*HttpSession session*, *User user*) {  *User* currentUser = (*User*)*session*.getAttribute(*Const*.***CURRENT\_USER***);  if (currentUser == null) {  return *ServerResponse*.createByErrorMessage("用户未登录");  }  *user*.setId(currentUser.getId());  *user*.setUsername(currentUser.getUsername());  *ServerResponse*<*User*> response = iUserService.updateInformation(*user*);  if (response.isSuccess()) {  *session*.setAttribute(*Const*.***CURRENT\_USER***, response.getData());  }  return response; } |

#### 4.10 用户信息获取

##### 4.10.1 Service层

（1）在IUserService接口中添加获取用户信息的方法

|  |
| --- |
| /\*\*  \* 获取用户信息  \* @param userId  \* @return  \*/ *ServerResponse*<*User*> getInformation(*Integer userId*); |

（2）在UserServiceImpl类中实现该方法

|  |
| --- |
| /\*\*  \* 获取用户信息  \* @param userId  \* @return  \*/ public *ServerResponse*<*User*> getInformation(*Integer userId*) {  *User* user = userMapper.selectByPrimaryKey(*userId*);  if (user == null) {  return *ServerResponse*.createByErrorMessage("用户不存在");  }  user.setPassword(*StringUtils*.***EMPTY***);  return *ServerResponse*.createBySuccess(user); } |

##### 4.10.2 Controller层

|  |
| --- |
| /\*\*  \* 获取用户信息  \* @param session  \* @return  \*/ @RequestMapping(value = "get\_information.do", method = *RequestMethod*.***POST***) @ResponseBody public *ServerResponse*<*User*> getInformation(*HttpSession session*) {  *User* currentUser = (*User*)*session*.getAttribute(*Const*.***CURRENT\_USER***);  if (currentUser == null) {  return *ServerResponse*.createByErrorCodeMessage(*ResponseCode*.***NEED\_LOGIN***.getCode(), "未登录，需要登录status=10");  }  return iUserService.getInformation(currentUser.getId()); } |

#### 4.11 管理员登录

##### 4.11.1 Service层

|  |
| --- |
| /\*\*  \* 校验用户是否是管理员  \* @param user  \* @return  \*/ public *ServerResponse* checkAdminRole(*User user*) {  if (*user* != null && *user*.getRole().intValue() == *Const*.*Role*.***ROLE\_ADMIN***) {  return *ServerResponse*.createBySuccess();  }  return *ServerResponse*.createByError(); } |

##### 4.11.2 Controller层

|  |
| --- |
| package *com.tmall.controller.backend*;  import *com.tmall.common.Const*; import *com.tmall.common.ServerResponse*; import *com.tmall.pojo.User*; import *com.tmall.service.IUserService*; import *org.springframework.beans.factory.annotation.*Autowired; import *org.springframework.stereotype.*Controller; import *org.springframework.web.bind.annotation.*RequestMapping; import *org.springframework.web.bind.annotation.RequestMethod*; import *org.springframework.web.bind.annotation.*ResponseBody;  import *javax.servlet.http.HttpSession*;  @Controller @RequestMapping("/manage/user") public class *UserManagerController* {   @Autowired  private *IUserService* iUserService;   @RequestMapping(value="login.do",method = *RequestMethod*.***POST***)  @ResponseBody  public *ServerResponse*<*User*> login(*String username*, *String password*, *HttpSession session*) {  *ServerResponse*<*User*> response = iUserService.login(*username*, *password*);  if (response.isSuccess()) {  *User* user = response.getData();  if (user.getRole() == *Const*.*Role*.***ROLE\_ADMIN***) {  *session*.setAttribute(*Const*.***CURRENT\_USER***, user);  return response;  } else {  return *ServerResponse*.createByErrorMessage("不是管理员，无法登录");  }  }  return response;  }  } |

### 五 分类管理模块

#### 5.1 新增分类

##### 5.1.1 Service层

（1）在ICategoryService接口中添加分类信息的方法

|  |
| --- |
| *ServerResponse* addCategory(*String categoryName*, *Integer parentId*); |

（2）在CategoryServiceImpl方法中实现接口

|  |
| --- |
| /\*\*  \* 添加商品分类  \* @param categoryName  \* @param parentId  \* @return  \*/ public *ServerResponse* addCategory(*String categoryName*, *Integer parentId*) {  if (*parentId* == null || *StringUtils*.isBlank(*categoryName*)) {  return *ServerResponse*.createByErrorMessage("添加品类参数错误");  }   *Category* category = new Category();  category.setName(*categoryName*);  category.setParentId(*parentId*);  // 分类是可用的  category.setStatus(true);   int rowCount = categoryMapper.insert(category);  if (rowCount > 0) {  return *ServerResponse*.createBySuccess("添加品类成功");  }  return *ServerResponse*.createByErrorMessage("添加品类失败"); } |

##### 5.1.2 Controller层

|  |
| --- |
| @Controller @RequestMapping("/manage/category") public class *CategoryManagerController* {   @Autowired  private *IUserService* iUserService;   @Autowired  private *ICategoryService* iCategoryService;   /\*\*  \* 添加分类  \* @param session  \* @param categoryName  \* @param parentId  \* @return  \*/  @RequestMapping("add\_category.do")  @ResponseBody  public *ServerResponse* addCategory(*HttpSession session*, *String categoryName*, @RequestParam(value = "parentId", defaultValue = "0") int *parentId*) {  *User* user = (*User*) *session*.getAttribute(*Const*.***CURRENT\_USER***);  if (user == null) {  return *ServerResponse*.createByErrorCodeMessage(*ResponseCode*.***NEED\_LOGIN***.getCode(), "用户未登录");  }  if (iUserService.checkAdminRole(user).isSuccess()) {  return iCategoryService.addCategory(*categoryName*, *parentId*);  } else {  return *ServerResponse*.createByErrorMessage("无权限操作，需要管理员权限");  }  }  } |

#### 5.2 更新分类

##### 5.2.1 Service层

（1）在ICategoryService接口中更新分类名称的方法

|  |
| --- |
| *ServerResponse* updateCategoryName(*Integer categoryId*, *String categoryName*); |

（2）在CategoryServiceImpl方法中实现接口

|  |
| --- |
| /\*\*  \* 更新分类名称  \* @param categoryId  \* @param categoryName  \* @return  \*/ public *ServerResponse* updateCategoryName(*Integer categoryId*, *String categoryName*) {  if (*categoryId* == null || *StringUtils*.isBlank(*categoryName*)) {  return *ServerResponse*.createByErrorMessage("更新品类参数错误");  }  *Category* category = new Category();  category.setId(*categoryId*);  category.setName(*categoryName*);   int rowCount = categoryMapper.updateByPrimaryKeySelective(category);  if (rowCount > 0) {  return *ServerResponse*.createBySuccess("更新品类成功");  }  return *ServerResponse*.createByErrorMessage("更新品类失败"); } |

##### 5.2.2 Controller层

|  |
| --- |
| /\*\*  \* 更新分类名称  \* @param session  \* @param categoryId  \* @param categoryName  \* @return  \*/ @RequestMapping("set\_category\_name.do") @ResponseBody public *ServerResponse* setCategoryName(*HttpSession session*, *Integer categoryId*, *String categoryName*) {  *User* user = (*User*) *session*.getAttribute(*Const*.***CURRENT\_USER***);  if (user == null) {  return *ServerResponse*.createByErrorCodeMessage(*ResponseCode*.***NEED\_LOGIN***.getCode(), "用户未登录");  }  if (iUserService.checkAdminRole(user).isSuccess()) {  return iCategoryService.updateCategoryName(*categoryId*, *categoryName*);  } else {  return *ServerResponse*.createByErrorMessage("无权限操作，需要管理员权限");  } } |

#### 5.3 新增子分类

##### 5.3.1 DAO层

在categoryMapper中添加根据父结点查询子节点方法

|  |
| --- |
| /\*\*  \* 根据父节点查询子分类  \* @param parentId  \* @return  \*/ *List*<*Category*> selectCategoryChildrenByParentId(*Integer parentId*); |

##### 5.3.2 数据库

在categoryMapper.xml中添加查询SQL

|  |
| --- |
| <select *id*="selectCategoryChildrenByParentId" *parameterType*="int" *resultMap*="BaseResultMap">  SELECT <include *refid*="Base\_Column\_List" />  from tmall\_category  where parent\_id = *#{parentId}* </select> |

##### 5.3.3 实体类

重写equals和hashCode方法

|  |
| --- |
| @Override public boolean equals(*Object o*) {  if (this == *o*) return true;  if (*o* == null || getClass() != *o*.getClass()) return false;   *Category* category = (*Category*) *o*;   return id != null ? id.equals(category.id) : category.id == null; }  @Override public int hashCode() {  return id != null ? id.hashCode() : 0; } |

##### 5.3.4 Service层

（1）在ICategoryService接口中更新分类名称的方法

|  |
| --- |
| *ServerResponse* getChildrenParallelCategory(*Integer categoryId*); |

（2）在CategoryServiceImpl方法中实现接口

|  |
| --- |
| /\*\*  \* 查询分类的子分类  \* @param categoryId  \* @return  \*/ public *ServerResponse*<*List*<*Category*>> getChildrenParallelCategory(*Integer categoryId*) {  *List*<*Category*> categoryList = categoryMapper.selectCategoryChildrenByParentId(*categoryId*);  if (*CollectionUtils*.isEmpty(categoryList)) {  logger.info("未找到当前分类的子分类");  }  return *ServerResponse*.createBySuccess(categoryList); } |

##### 5.3.5 Controller层

|  |
| --- |
| /\*\*  \* 查询子分类  \* @param session  \* @param categoryId  \* @return  \*/ @RequestMapping("get\_category.do") @ResponseBody public *ServerResponse* getChildrenParallelCategory(*HttpSession session*, @RequestParam(value = "categoryId", defaultValue = "0") int *categoryId*) {  *User* user = (*User*) *session*.getAttribute(*Const*.***CURRENT\_USER***);  if (user == null) {  return *ServerResponse*.createByErrorCodeMessage(*ResponseCode*.***NEED\_LOGIN***.getCode(), "用户未登录");  }  if (iUserService.checkAdminRole(user).isSuccess()) {  // 查询子节点的分类信息，并且不递归，保持平级  return iCategoryService.getChildrenParallelCategory(*categoryId*);  } else {  return *ServerResponse*.createByErrorMessage("无权限操作，需要管理员权限");  } } |

#### 5.4 递归更新子分类

##### 5.4.1 Service层

（1）在ICategoryService接口中递归更新分类名称的方法

|  |
| --- |
| *ServerResponse* selectCategoryAndChildrenById(*Integer categoryId*); |

（2）在CategoryServiceImpl方法中实现接口

|  |
| --- |
| /\*\*  \* 根据ID查询子分类  \* @param categoryId  \* @return  \*/ public *ServerResponse* selectCategoryAndChildrenById(*Integer categoryId*) {  *Set*<*Category*> categorySet = *Sets*.newHashSet();  findChildrenCategory(categorySet, *categoryId*);   *List*<*Integer*> categoryIdList = *Lists*.newArrayList();  if (*categoryId* != null) {  for (*Category* categoryItem : categorySet) {  categoryIdList.add(categoryItem.getId());  }  }  return *ServerResponse*.createBySuccess(categoryIdList); }  /\*\*  \* 递归查询子节点  \* @param categorySet  \* @param categoryId  \* @return  \*/ private *Set*<*Category*> findChildrenCategory(*Set*<*Category*> *categorySet*, *Integer categoryId*) {  *Category* category = categoryMapper.selectByPrimaryKey(*categoryId*);  if (category != null) {  *categorySet*.add(category);  }  // 查询子节点  *List*<*Category*> categoryList = categoryMapper.selectCategoryChildrenByParentId(*categoryId*);  for (*Category* categoryItem : categoryList) {  findChildrenCategory(*categorySet*, categoryItem.getId());  }  return *categorySet*; } |

##### 5.4.2 Controller层

|  |
| --- |
| /\*\*  \* 递归查询子节点  \* @param session  \* @param categoryId  \* @return  \*/ @RequestMapping("get\_deep\_category.do") @ResponseBody public *ServerResponse* getCategoryAndDeepChildrenCategory(*HttpSession session*, @RequestParam(value = "categoryId", defaultValue = "0") int *categoryId*) {  *User* user = (*User*) *session*.getAttribute(*Const*.***CURRENT\_USER***);  if (user == null) {  return *ServerResponse*.createByErrorCodeMessage(*ResponseCode*.***NEED\_LOGIN***.getCode(), "用户未登录");  }  if (iUserService.checkAdminRole(user).isSuccess()) {  return iCategoryService.selectCategoryAndChildrenById(*categoryId*);  } else {  return *ServerResponse*.createByErrorMessage("无权限操作，需要管理员权限");  } } |

### 六 商品管理模块

#### 6.1 新增更新产品

##### 6.1.1 Service层

（1）在IProductService接口中新增更新产品的方法

|  |
| --- |
| public interface *IProductService* {   *ServerResponse* saveOrUpdateProduct(*Product product*);  } |

（2）在ProductServiceImpl方法中实现接口

|  |
| --- |
| /\*\*  \* 产品模块  \*/ @Service("iProductService") public class *ProductServiceImpl* implements *IProductService* {   @Autowired  private *ProductMapper* productMapper;   /\*\*  \* 新增或更新产品  \* @param product  \* @return  \*/  public *ServerResponse* saveOrUpdateProduct(*Product product*) {  if (*product* != null) {  if (*StringUtils*.isNotBlank(*product*.getSubImages())) {  *String*[] subImageArray = *product*.getSubImages().split(",");  if (subImageArray.length > 0) {  *product*.setMainImage(subImageArray[0]);  }  }  if (*product*.getId() != null) {  int rowCount = productMapper.updateByPrimaryKey(*product*);  if (rowCount > 0) {  return *ServerResponse*.createBySuccessMessage("更新产品成功");  }  return *ServerResponse*.createByErrorMessage("更新产品失败");  } else {  int rowCount = productMapper.insert(*product*);  if (rowCount > 0) {  return *ServerResponse*.createBySuccessMessage("新增产品成功");  }  return *ServerResponse*.createByErrorMessage("新增产品失败");  }  }  return *ServerResponse*.createByErrorMessage("新增或更新产品参数错误");  }  } |

##### 6.1.2 Controller层

|  |
| --- |
| package *com.tmall.controller.backend*;  import *com.tmall.common.Const*; import *com.tmall.common.ResponseCode*; import *com.tmall.common.ServerResponse*; import *com.tmall.pojo.Product*; import *com.tmall.pojo.User*; import *com.tmall.service.IProductService*; import *com.tmall.service.IUserService*; import *org.springframework.beans.factory.annotation.*Autowired; import *org.springframework.stereotype.*Controller; import *org.springframework.web.bind.annotation.*RequestMapping; import *org.springframework.web.bind.annotation.*ResponseBody;  import *javax.servlet.http.HttpSession*;  @Controller @RequestMapping("/manage/product") public class *ProductManageController* {   @Autowired  private *IUserService* iUserService;   @Autowired  private *IProductService* iProductService;   /\*\*  \* 新增或更新产品  \* @param session  \* @param product  \* @return  \*/  @RequestMapping("save.do")  @ResponseBody  public *ServerResponse* productSave(*HttpSession session*, *Product product*) {  *User* user = (*User*) *session*.getAttribute(*Const*.***CURRENT\_USER***);  if (user == null) {  return *ServerResponse*.createByErrorCodeMessage(*ResponseCode*.***NEED\_LOGIN***.getCode(), "用户未登录");  }  if (iUserService.checkAdminRole(user).isSuccess()) {  return iProductService.saveOrUpdateProduct(*product*);  } else {  return *ServerResponse*.createByErrorMessage("无权限操作");  }  }  } |

#### 6.2 修改产品销售状态

##### 6.2.1 Service层

（1）在IProductService接口中新增更新产品的方法

|  |
| --- |
| *ServerResponse* setSaleStatus(*Integer productId*, *Integer status*); |

（2）在ProductServiceImpl方法中实现接口

|  |
| --- |
| /\*\*  \* 修改产品销售状态  \* @param productId  \* @param status  \* @return  \*/ public *ServerResponse* setSaleStatus(*Integer productId*, *Integer status*) {  if (*productId* == null || *status* == null) {  return *ServerResponse*.createByErrorCodeMessage(*ResponseCode*.***ILLEGAL\_ARGUMENT***.getCode(), *ResponseCode*.***ILLEGAL\_ARGUMENT***.getDesc());  }  *Product* product = new Product();  product.setId(*productId*);  product.setStatus(*status*);  int rowCount = productMapper.updateByPrimaryKeySelective(product);  if (rowCount > 0) {  return *ServerResponse*.createBySuccessMessage("修改产品销售状态成功");  }  return *ServerResponse*.createByErrorMessage("修改产品销售状态失败"); } |