# EGO-SHOP后台和商品管理

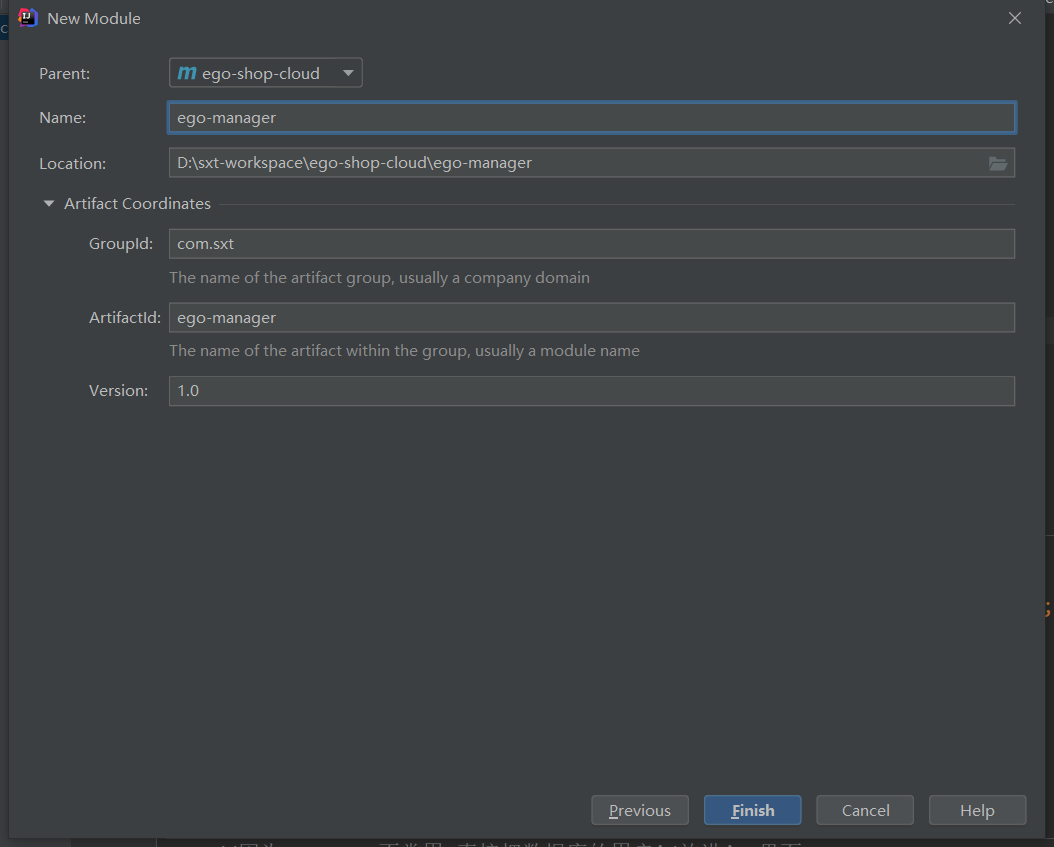
## ego-manager后台管理系统的搭建

功能定位：后台用户的curd，权限的分配，日志管理，登录管理等，前台轮播图管理，

其他模块集成：商品模块，订单模块，店铺模块等

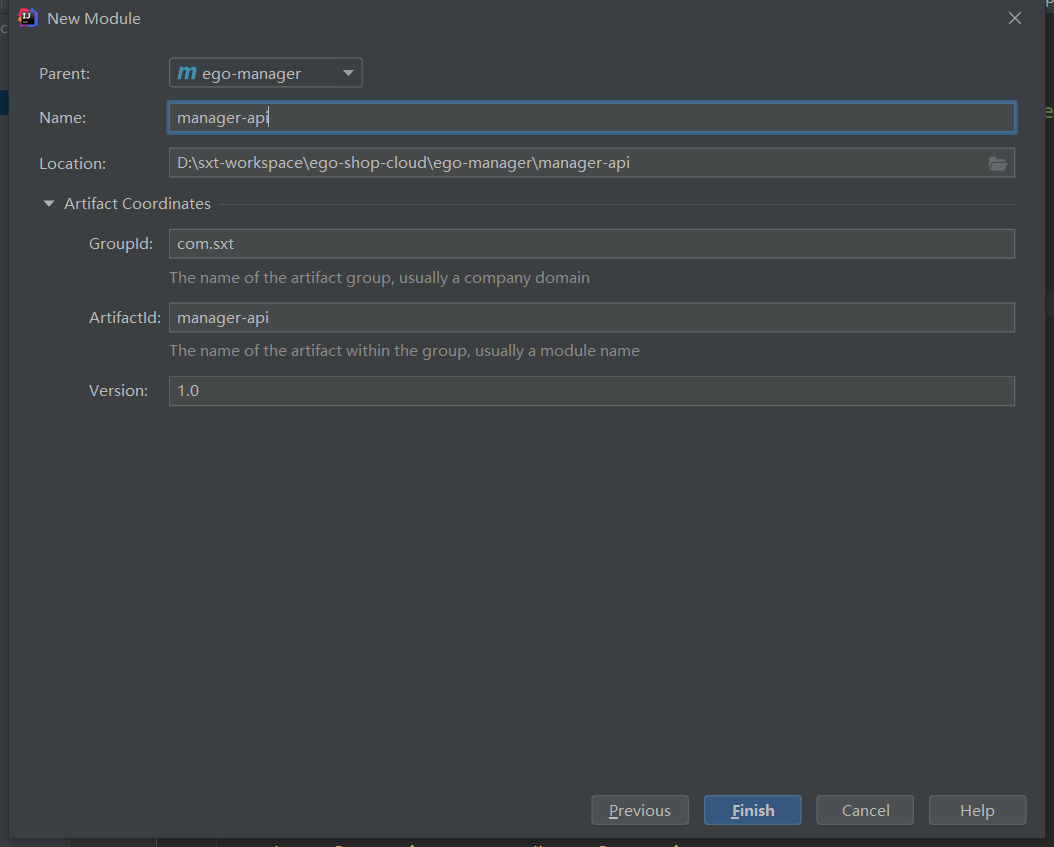
### ego-manager父模块搭建

#### 在ego-sho-cloud下面新建一个maven模块



### manager-api模块搭建

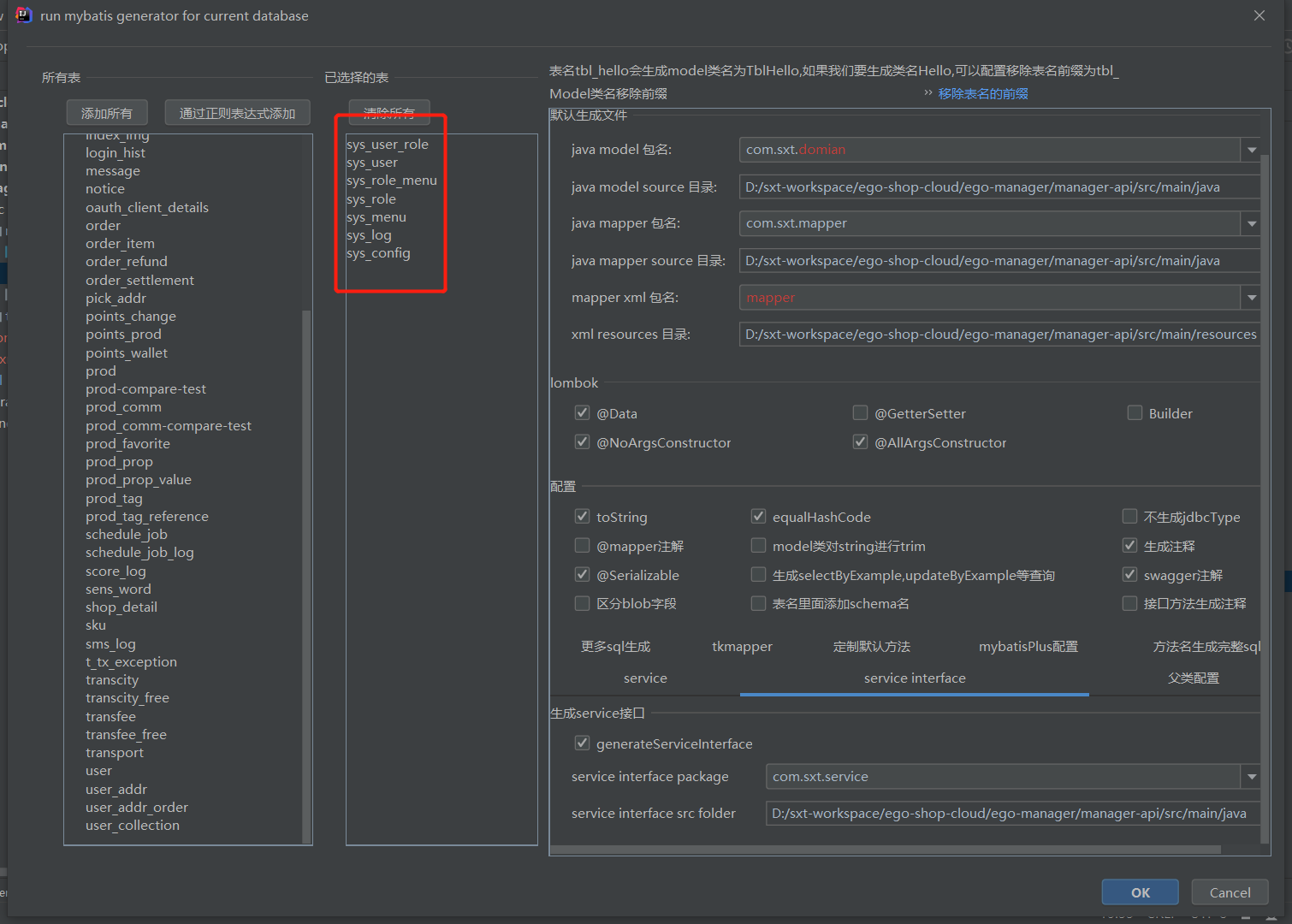
#### 在ego-manager下面新建一个maven模块



#### 修改pom.xml文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <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/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>ego-manager</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>manager-api</artifactId>  <description>ego商城后台管理系统</description>  <dependencies>  <!-- 直接依赖common -->  <dependency>  <groupId>com.sxt</groupId>  <artifactId>ego-common</artifactId>  <version>1.0</version>  </dependency>  </dependencies>  </project> |

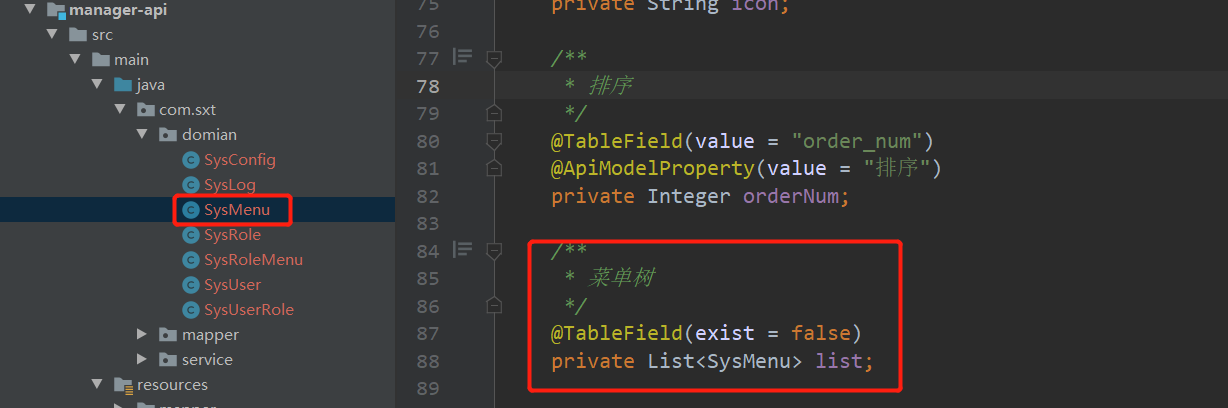
#### 逆向生成实体类



#### 将Date类型添@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss")

SysLog，SysRole，SysUser

#### 在SysMenu类中添加树菜单的List集合



#### 创建常量类ManagerConstant

|  |
| --- |
| public class ManagerConstant {  //菜单数据缓存的key前缀  public static final String MENU = "menu";  } |

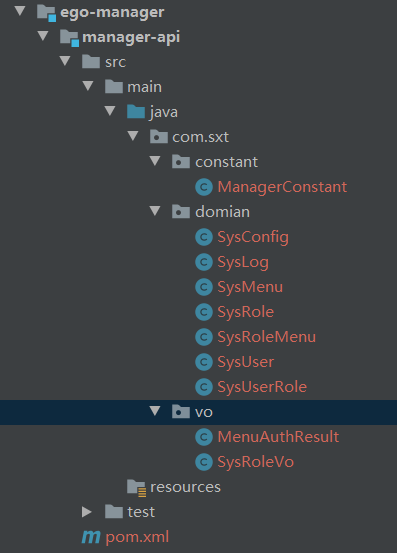
#### 创建菜单和权限的组合类MenuAuthResult

|  |
| --- |
| @Data  public class MenuAuthResult {  @ApiModelProperty("菜单数据")  private List<SysMenu> menuList;  @ApiModelProperty("用户的权限数据")  private List<String> authorities;  } |

#### 创建角色和权限的vo

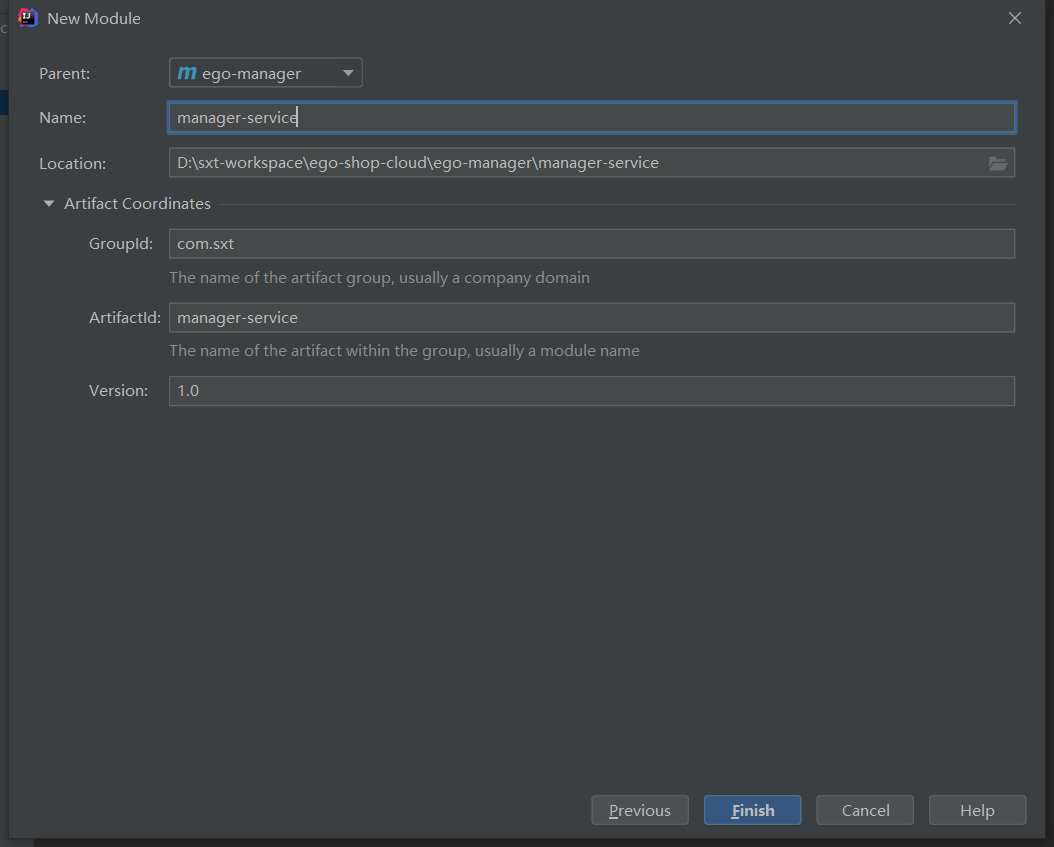
|  |
| --- |
| @Data  public class SysRoleVo extends SysRole {  //添加角色时权限的vo  private List<String> menuIdList;  } |

#### 包和类的截图



### manager-service模块搭建

#### 在ego-manager下面新建一个maven模块



#### 修改pom.xml文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <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/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>ego-manager</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>manager-service</artifactId>  <description>ego商城后台管理系统service</description>  <dependencies>  <!--依赖manager-api-->  <dependency>  <groupId>com.sxt</groupId>  <artifactId>manager-api</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-bus-amqp</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-config-client</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis-reactive</artifactId>  </dependency>  <!-- spring的切面编程-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-aop</artifactId>  </dependency>  <!-- fastDFS的依赖-->  <dependency>  <groupId>com.github.tobato</groupId>  <artifactId>fastdfs-client</artifactId>  <version>${fastDFS.version}</version>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  <!-- 引入druid starter-->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid-spring-boot-starter</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

#### 添加启动类

|  |
| --- |
| @SpringBootApplication  @EnableEurekaClient  public class ManagerServiceApplication {  public static void main(String[] args) {  SpringApplication.run(ManagerServiceApplication.class, args);  }  } |

#### 添加bootstrap.yml配置文件

|  |
| --- |
| spring:  application:  name: manager-service  cloud:  config:  discovery:  service-id: config-server  enabled: true  name: manager-service  label: master  profile: dev  eureka:  client:  service-url:  defaultZone: ${EUREKA\_SERVER:http://admin:admin@localhost:8761/eureka/}  instance:  hostname: ${APP\_HOST:localhost}  prefer-ip-address: true  instance-id: ${spring.application.name}:${server.port}  lease-renewal-interval-in-seconds: 10  lease-expiration-duration-in-seconds: 30 |

#### 添加远端的manager-service/manager-service-dev.yml

|  |
| --- |
| server:  port: ${APP\_PORT:8081}  spring:  datasource:  druid:  url: jdbc:mysql://localhost:3306/ego-shop?useSSL=false&serverTimezone=GMT%2B8  username: root  password: root  driver-class-name: com.mysql.cj.jdbc.Driver  max-active: 30  min-idle: 3  initial-size: 3  validation-query: select 1  max-wait: 5000  stat-view-servlet:  enabled: true  login-username: admin  login-password: admin  allow:  deny:  url-pattern:  - /druid/\*  rabbitmq:  host: 192.168.226.129  password: admin  username: admin  port: 5672  redis:  host: 192.168.226.129  port: 6380  password: cxs1013??  database: 0  management:  endpoints:  web:  exposure:  include: '\*'  mybatis-plus:  mapper-locations: classpath:mapper/\*\*/\*.xml  configuration:  log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  fdfs: #fastDfs的配置  so-timeout: 1501  connect-timeout: 601  tracker-list:  - 192.168.226.129:22122  resources:  url: http://192.168.226.129  swagger2:  base-package: com.sxt.controller  name: sxt  url: https://gitee.com/smiledouble  email: 775610843@qq.com  title: 后台管理系统接口  description: 实现后台管理系统的所有接口  license: cxs  license-url: https://gitee.com/smiledouble  terms-of-service-url: https://gitee.com/smiledouble |

#### 添加日志的切面注解Log

|  |
| --- |
| @Documented  @Target(ElementType.METHOD)  @Retention(RetentionPolicy.RUNTIME)  public @interface Log {  //日志的切面注解  String operation() default "";  } |

#### 添加日志的切面

|  |
| --- |
| @Aspect  @Component  @EnableAspectJAutoProxy(proxyTargetClass = true)  public class LogAspect {  @Autowired  private SysUserService sysUserService;  @Autowired  private SysLogService sysLogService;  /\*\*  \* 日志的界面注解  \*  \* @return  \*/  @Around(value = "@annotation(com.sxt.anno.Log)")  public Object logAspect(ProceedingJoinPoint joinPoint) {  //拿到方法名 拿到用户id 拿到参数 拿到操作名称  MethodSignature signature = (MethodSignature) joinPoint.getSignature();  Method method = signature.getMethod();  //拿到方法名称  String methodName = method.getName();  //拿到具体的注解  Log log = method.getAnnotation(Log.class);  //拿到操作名称  String operation = log.operation();  //拿到ip  ServletRequestAttributes requestAttributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();  HttpServletRequest request = requestAttributes.getRequest();  String ip = request.getRemoteAddr();  System.out.println(ip);  //方法执行时间  Object proceed = null;  Long time = null;  try {  long startTime = System.currentTimeMillis();  proceed = joinPoint.proceed();  long endTime = System.currentTimeMillis();  time = endTime - startTime;  } catch (Throwable throwable) {  throwable.printStackTrace();  }  String userId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  //查询sysUser  SysUser sysUser = sysUserService.getById(Long.valueOf(userId));  //组装sysLog对象  SysLog sysLog = new SysLog();  sysLog.setCreateDate(new Date());  sysLog.setMethod(methodName);  sysLog.setOperation(operation);  sysLog.setUsername(sysUser.getUsername());  sysLog.setIp(ip);  sysLog.setParams(joinPoint.getArgs() == null ? "" : JSON.toJSONString(joinPoint.getArgs()));  sysLog.setTime(time);  sysLogService.save(sysLog);  return proceed;  }  } |

#### 添加LogController

|  |
| --- |
| @RestController  @RequestMapping("/sys/log")  @Api(tags = "日志的管理接口")  public class LogController {  @Autowired  private SysLogService sysLogService;  @GetMapping("/page")  @ApiOperation("分页查询")  public ResponseEntity<IPage<SysLog>> page(Page<SysLog> page, SysLog sysLog) {  //设置排序  page.addOrder(OrderItem.desc("create\_date"));  //模糊查询  IPage<SysLog> page1 = sysLogService.page(page, new LambdaQueryWrapper<SysLog>()  .like(StringUtils.hasText(sysLog.getUsername()), SysLog::getUsername, sysLog.getUsername())  .like(StringUtils.hasText(sysLog.getOperation()), SysLog::getOperation, sysLog.getOperation())  );  return ResponseEntity.ok(page1);  }  } |

#### 添加MenuController

|  |
| --- |
| @RestController  @RequestMapping("/sys/menu")  @Api(tags = "菜单的管理接口")  public class MenuController {  @Autowired  private SysMenuService sysMenuService;  @Autowired  private StringRedisTemplate redisTemplate;  /\*\*  \* 加载用户的菜单和权限数据  \*  \* @return  \*/  @GetMapping("/nav")  @ApiOperation("加载后台菜单和用户所有的权限")  public ResponseEntity<MenuAuthResult> loadMenuAndAuth() {  //得到当前用户id  String userId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  //得到该用户的权限  Collection<? extends GrantedAuthority> authorities = SecurityContextHolder.getContext().getAuthentication().getAuthorities();  List<SysMenu> sysMenus = sysMenuService.loadMenuAndAuth(Long.valueOf(userId));  MenuAuthResult menuAuthResult = new MenuAuthResult();  menuAuthResult.setMenuList(sysMenus);  ArrayList<String> auths = new ArrayList<>(authorities.size() \* 2);  authorities.forEach(k -> auths.add(k.getAuthority()));  menuAuthResult.setAuthorities(auths);  return ResponseEntity.ok(menuAuthResult);  }  @GetMapping("/table")  @ApiOperation("加载所有的菜单列表")  @PreAuthorize("hasAuthority('sys:menu:list')")  public ResponseEntity<List<SysMenu>> loadAllMenuList() {  List<SysMenu> sysMenus = null;  if (redisTemplate.hasKey(ManagerConstant.MENU)) {  String menuStr = redisTemplate.opsForValue().get(ManagerConstant.MENU);  sysMenus = JSON.parseArray(menuStr, SysMenu.class);  } else {  sysMenus = sysMenuService.list();  String menuStr = JSON.toJSONString(sysMenus);  redisTemplate.opsForValue().set(ManagerConstant.MENU, menuStr, Duration.ofDays(1));  }  return ResponseEntity.ok(sysMenus);  }  @GetMapping("/list")  @ApiOperation("查询所有父菜单")  @PreAuthorize("hasAuthority('sys:menu:list')")  public ResponseEntity<List<SysMenu>> loadParentMenuList() {  //查询所有父菜单，type不等于2的 not equals  List<SysMenu> sysMenus = sysMenuService.list(new LambdaQueryWrapper<SysMenu>()  .ne(SysMenu::getType, 2)  );  return ResponseEntity.ok(sysMenus);  }  @ApiOperation("新增一个菜单")  @PostMapping  public ResponseEntity<Void> add(@RequestBody @Validated SysMenu sysMenu) {  sysMenuService.save(sysMenu);  return ResponseEntity.ok().build();  }  @ApiOperation("删除一个菜单")  @PreAuthorize("hasAuthority('sys:menu:delete')")  @DeleteMapping("/{id}")  public ResponseEntity<Void> delete(@PathVariable("id") Long id) {  sysMenuService.removeById(id);  return ResponseEntity.ok().build();  }  @ApiOperation("菜单数据的回显")  @PreAuthorize("hasAuthority('sys:menu:info')")  @GetMapping("/info/{id}")  public ResponseEntity<SysMenu> findById(@PathVariable("id") Long id) {  SysMenu byId = sysMenuService.getById(id);  return ResponseEntity.ok(byId);  }  @PutMapping  @ApiOperation("修改菜单数据")  @PreAuthorize("hasAuthority('sys:menu:update')")  public ResponseEntity<Void> update(@RequestBody @Validated SysMenu sysMenu) {  sysMenuService.updateById(sysMenu);  return ResponseEntity.ok().build();  }  } |

#### 修改SysMenuService

|  |
| --- |
| public interface SysMenuService extends IService<SysMenu> {  /\*\*  \* 根基用户id得到他的菜单  \*  \* @param username  \* @return  \*/  List<SysMenu> loadMenuAndAuth(Long userId);  } |

#### 修改SysMenuServiceImpl

|  |
| --- |
| @Service  public class SysMenuServiceImpl extends ServiceImpl<SysMenuMapper, SysMenu> implements SysMenuService {  @Autowired  private SysMenuMapper sysMenuMapper;  @Autowired  private StringRedisTemplate redisTemplate;  /\*\*  \* 根基用户id得到他的菜单  \* 通过type筛选出菜单数据  \* 组装成树结构  \*  \* @param userId@return  \*/  @Override  public List<SysMenu> loadMenuAndAuth(Long userId) {  //先看redis有没有  List<SysMenu> userMenus = Collections.EMPTY\_LIST;  if (redisTemplate.hasKey(ManagerConstant.MENU + userId)) {  String userMenu = redisTemplate.opsForValue().get(ManagerConstant.MENU + ":" + userId);  userMenus = JSON.parseArray(userMenu, SysMenu.class);  } else {  //通过id查询对应的所有菜单  userMenus = sysMenuMapper.selectUserMenus(userId);  String jsonString = JSON.toJSONString(userMenus);  redisTemplate.opsForValue().set(ManagerConstant.MENU + ":" + userId, jsonString, Duration.ofDays(1));  }  //组装成树结构  List<SysMenu> tree = transTree(userMenus);  return tree;  }  /\*\*  \* 组装成树结构  \*  \* @param sysMenus  \* @return  \*/  private List<SysMenu> transTree(List<SysMenu> sysMenus) {  //找到所有的父亲节点 parentId = 0的  List<SysMenu> root = sysMenus.stream()  .filter(sysMenu -> sysMenu.getParentId().equals(0L))  .collect(Collectors.toList());  //循环root节点  root.forEach(r -> {  //新建一个子集合  List<SysMenu> subNodes = new ArrayList<>();  //循环所有的菜单  sysMenus.forEach((sysMenu) -> {  if (sysMenu.getParentId().equals(r.getMenuId())) {  subNodes.add(sysMenu);  }  });  //往根节点添加子节点  r.setList(subNodes);  });  return root;  }  }  $2a$10$V0ODEZSeDvin.UcwL3jL0.BEDdqyYpRbj0BBbLwFY5yw/yE8de64y |

#### 修改SysMenuMapper

|  |
| --- |
| public interface SysMenuMapper extends BaseMapper<SysMenu> {  /\*\*  \* 查询用户id对应的所有菜单  \*  \* @param userId  \* @return  \*/  @Select(" SELECT t1.\* FROM `sys\_menu` t1 left join sys\_role\_menu t2 on(t1.menu\_id=t2.menu\_id) join sys\_user\_role t3 on(t2.role\_id = t3.role\_id) where t3.user\_id = #{userId} and (t1.type = 0 or t1.type = 1) ")  List<SysMenu> selectUserMenus(Long userId);  } |

#### 添加RoleController

|  |
| --- |
| @RestController  @RequestMapping("/sys/role")  @Api(tags = "角色的管理接口")  public class RoleController {  @Autowired  private SysRoleService sysRoleService;  @ApiOperation("全查询角色列表")  @GetMapping("/list")  public ResponseEntity<List<SysRole>> list() {  List<SysRole> list = sysRoleService.list();  return ResponseEntity.ok(list);  }  /\*\*  \* 根据名称来做分页查询  \*  \* @param page  \* @param sysRole  \* @return  \*/  @GetMapping("/page")  @PreAuthorize("hasAuthority('sys:role:page')")  @ApiOperation("分页查询角色数据")  public ResponseEntity<IPage<SysRole>> page(Page<SysRole> page, SysRole sysRole) {  IPage<SysRole> roleIPage = sysRoleService.page(page, new LambdaQueryWrapper<SysRole>()  .like(StringUtils.hasText(sysRole.getRoleName()), SysRole::getRoleName, sysRole.getRoleName())  );  return ResponseEntity.ok(roleIPage);  }  @DeleteMapping  @ApiOperation("删除角色")  @PreAuthorize("hasAuthority('sys:role:delete')")  public ResponseEntity<Void> delete(@RequestBody List<Long> ids) {  //注意，如果有用户正在使用某些角色，是不能删除的，这里不做详细代码了  sysRoleService.removeByIds(ids);  return ResponseEntity.ok().build();  }  @PostMapping  @ApiModelProperty("新增一个角色")  @PreAuthorize("hasAuthority('sys:role:save')")  public ResponseEntity<Void> add(@RequestBody @Validated SysRoleVo sysRoleVo) {  //获取当前用户  String userId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  //新增角色表，修改角色和权限的中间表  sysRoleService.saveRoleAndMenu(sysRoleVo, Long.valueOf(userId));  return ResponseEntity.ok().build();  }  @ApiModelProperty("查询一个角色")  @GetMapping("/info/{id}")  @PreAuthorize("hasAuthority('sys:role:info') ")  public ResponseEntity<SysRole> findById(@PathVariable("id") Long id) {  SysRole byId = sysRoleService.getById(id);  return ResponseEntity.ok(byId);  }  @PutMapping  @ApiModelProperty("修改角色数据")  @PreAuthorize("hasAuthority('sys:role:update')")  public ResponseEntity<Void> update(@RequestBody @Validated SysRoleVo sysRoleVo) {  String userId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  //先设置更新操作人是谁  sysRoleVo.setCreateUserId(Long.valueOf(userId));  //更新角色  sysRoleService.updateByRoleId(sysRoleVo);  return ResponseEntity.ok().build();  }  } |

#### 修改SysRoleService

|  |
| --- |
| public interface SysRoleService extends IService<SysRole> {  /\*\*  \* 新增角色和权限  \*  \* @param sysRoleVo  \* @param userId  \*/  void saveRoleAndMenu(SysRoleVo sysRoleVo, Long userId);  /\*\*  \* 更新角色  \*  \* @param sysRoleVo  \* @return  \*/  boolean updateByRoleId(SysRoleVo sysRoleVo);  } |

#### 修改SysRoleServiceImpl

|  |
| --- |
| @Service  @Slf4j  public class SysRoleServiceImpl extends ServiceImpl<SysRoleMapper, SysRole> implements SysRoleService {  @Autowired  private SysRoleMapper sysRoleMapper;  @Autowired  private SysRoleMenuMapper sysRoleMenuMapper;  /\*\*  \* 新增角色和菜单  \*  \* @param sysRoleVo  \* @param userId  \*/  @Override  @Transactional  public void saveRoleAndMenu(SysRoleVo sysRoleVo, Long userId) {  log.info("新增角色开始，操作人Id%s", userId);  SysRole sysRole = new SysRole();  //对象拷贝  BeanUtil.copyProperties(sysRoleVo, sysRole, true);  sysRole.setCreateUserId(userId);  sysRole.setCreateTime(new Date());  sysRoleMapper.insert(sysRole);  log.info("新增角色后插入角色和菜单中间表");  //得到菜单和权限列表  List<String> menuIdList = sysRoleVo.getMenuIdList();  for (String s : menuIdList) {  SysRoleMenu sysRoleMenu = new SysRoleMenu();  sysRoleMenu.setRoleId(sysRole.getRoleId());  sysRoleMenu.setMenuId(Long.valueOf(s));  sysRoleMenuMapper.insert(sysRoleMenu);  }  }  /\*\*  \* 更新角色  \* 1.先更新角色  \* 2.更新权限 （1.删除旧的值，2.插入新的值）  \*  \* @param sysRoleVo  \* @return  \*/  @Override  @Transactional  public boolean updateByRoleId(SysRoleVo sysRoleVo) {  //1  log.info("修改id为%s的角色", sysRoleVo.getRoleId());  sysRoleVo.setCreateTime(new Date());  SysRole sysRole = new SysRole();  //对象拷贝  BeanUtil.copyProperties(sysRoleVo, sysRole, true);  int flag = sysRoleMapper.updateById(sysRole);  if (flag > 0) {  //2.1删除旧的权限  sysRoleMenuMapper.delete(new LambdaQueryWrapper<SysRoleMenu>()  .eq(SysRoleMenu::getRoleId, sysRole.getRoleId())  );  //2.2新增新的  @NotEmpty List<String> menuIdList = sysRoleVo.getMenuIdList();  for (String menuId : menuIdList) {  SysRoleMenu sysRoleMenu = new SysRoleMenu();  sysRoleMenu.setRoleId(sysRole.getRoleId());  sysRoleMenu.setMenuId(Long.valueOf(menuId));  sysRoleMenuMapper.insert(sysRoleMenu);  }  }  return true;  }  } |

#### 添加FileController

|  |
| --- |
| @RestController  @RequestMapping("/admin/file")  @Api(tags = "文件上传管理接口")  public class FileController {  @Autowired  private FastFileStorageClient fastFileStorageClient;  @Value("${resources.url}")  private String serverAddress;  /\*\*  \* @param file  \* @return  \*/  @PostMapping("/upload/element")  @ApiOperation("文件上传")  public ResponseEntity<String> fileUpload(@RequestParam("file") MultipartFile file) {  String filePath = null;  try {  filePath = fastFileStorageClient.uploadFile(file.getInputStream(), file.getSize(), "jpg", null).getFullPath();  } catch (IOException e) {  e.printStackTrace();  }  return ResponseEntity.ok().body(serverAddress + "/" + filePath);  }  } |

#### 添加SysUserController

|  |
| --- |
| @RestController  @RequestMapping("/sys/user")  @Api(tags = "后台用户的管理接口")  public class SysUserController {  @Autowired  private SysUserService sysUserService;  @GetMapping("/info")  public ResponseEntity<SysUser> getCurrentUser() {  // 获得用户id  String userId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  SysUser sysUser = sysUserService.getById(Long.valueOf(userId));  return ResponseEntity.ok(sysUser);  }  /\*\*  \* 分页查询  \*/  @GetMapping("/page")  @ApiOperation("分页的查询")  @PreAuthorize("hasAuthority('sys:user:page') ")  public ResponseEntity<IPage<SysUser>> findByPage(Page<SysUser> page, SysUser sysUser) {  IPage<SysUser> sysUserIPage = sysUserService.page(page, new LambdaQueryWrapper<SysUser>()  .like(StringUtils.hasText(sysUser.getUsername()), SysUser::getUsername, sysUser.getUsername())  );  return ResponseEntity.ok(sysUserIPage);  }  /\*\*  \* 删除一个用户  \*  \* @param id  \* @return  \*/  @DeleteMapping("/{id}")  @ApiOperation("删除一个用户")  @PreAuthorize("hasAuthority('sys:user:delete')")  @Log(operation = "删除一个用户")  public ResponseEntity<Void> delete(@PathVariable("id") Long id) {  sysUserService.removeById(id);  return ResponseEntity.ok().build();  }  /\*\*  \* 删除多个用户  \*/  @DeleteMapping  @ApiOperation("删除多个值")  @Log(operation = "删除多个用户")  public ResponseEntity<Void> delete(@RequestBody List<Long> ids) {  sysUserService.removeByIds(ids);  return ResponseEntity.ok().build();  }  /\*\*  \* 回显  \*/  @GetMapping("/info/{id}")  @ApiOperation("用户的回显")  @PreAuthorize("hasAuthority('sys:user:info')")  public ResponseEntity<SysUser> findById(@PathVariable("id") Long id) {  SysUser sysUser = sysUserService.getById(id);  return ResponseEntity.ok(sysUser);  }  /\*\*  \* 新增用户  \*  \* @param sysUser  \* @return  \*/  @PostMapping  @ApiOperation("新增用户")  @PreAuthorize("hasAuthority('sys:user:save')")  @Log(operation = "新增一个用户")  public ResponseEntity<Void> add(@RequestBody SysUser sysUser) {  sysUser.setPassword(new BCryptPasswordEncoder().encode(sysUser.getPassword()));  //时间  sysUser.setCreateTime(new Date());  //创建人  String createUserId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  sysUser.setCreateUserId(Long.valueOf(createUserId));  sysUserService.save(sysUser);  return ResponseEntity.ok().build();  }  /\*\*  \* 修改用户  \*  \* @param sysUser  \* @return  \*/  @PutMapping  @ApiOperation("修改用户")  @PreAuthorize("hasAuthority('sys:user:update')")  @Log(operation = "修改一个用户")  public ResponseEntity<Void> update(@RequestBody @Validated SysUser sysUser) {  sysUser.setPassword(new BCryptPasswordEncoder().encode(sysUser.getPassword()));  sysUserService.updateById(sysUser);  return ResponseEntity.ok().build();  }  } |

#### 修改SysUserServiceImpl

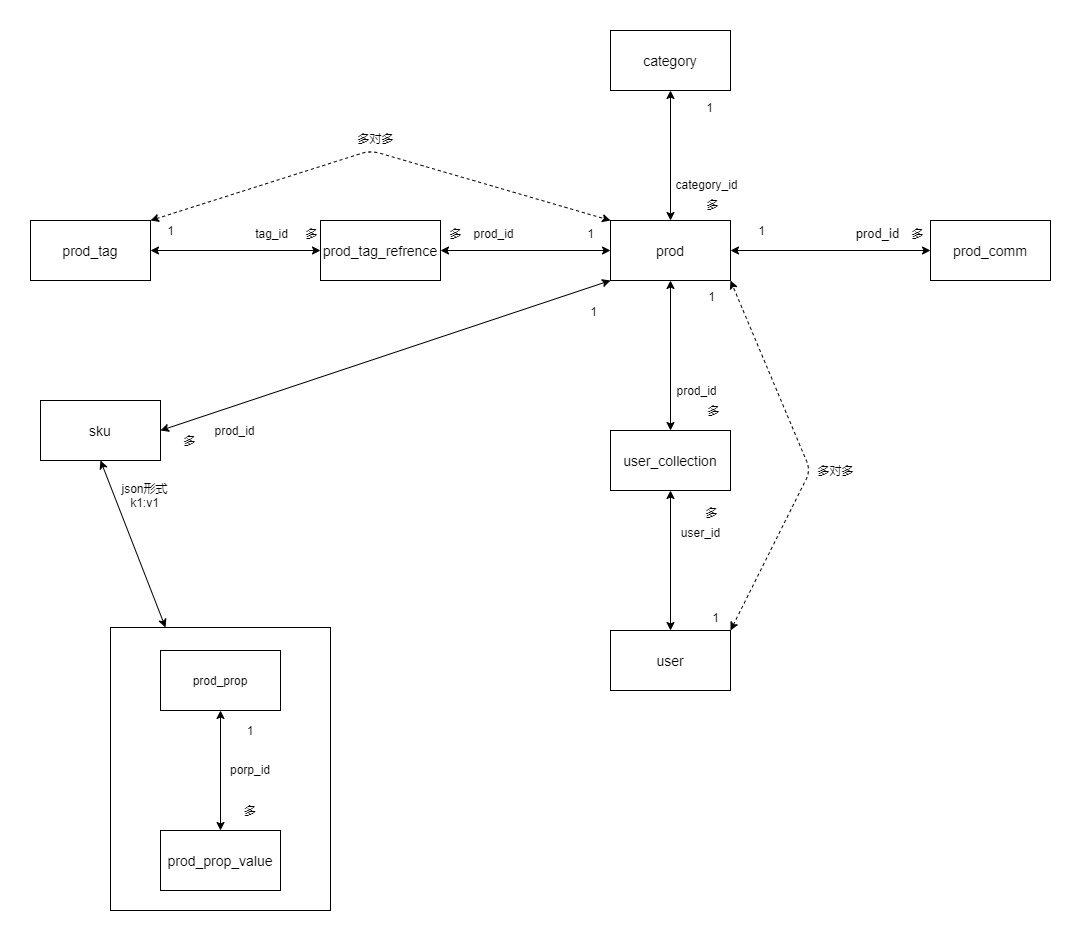
|  |
| --- |
| @Service  @Slf4j  public class SysUserServiceImpl extends ServiceImpl<SysUserMapper, SysUser> implements SysUserService {  @Autowired  private StringRedisTemplate redisTemplate;  @Autowired  private SysUserMapper sysUserMapper;  @Autowired  private SysUserRoleService sysUserRoleService;  @Override  public SysUser getById(Serializable id) {  SysUser sysUser = null;  String sysUserStr = redisTemplate.opsForValue().get(SysUserConstant.SYS\_USER\_PREFIX + id);  if (StringUtils.isEmpty(sysUserStr)) {  //缓存无  sysUser = super.getById(id);  //放缓存  redisTemplate.opsForValue().set(SysUserConstant.SYS\_USER\_PREFIX + id, JSON.toJSONString(sysUser));  } else {  //缓存有  sysUser = JSON.parseObject(sysUserStr, SysUser.class);  }  return sysUser;  }  /\*\*  \* 分页查询系统用户  \*  \* @param page  \* @param sysUser  \* @return  \*/  @Override  public IPage<SysUser> getSysUserByPage(Page<SysUser> page, SysUser sysUser) {  page.addOrder(OrderItem.desc("create\_time"));  //根据用户名称查询  IPage<SysUser> sysUserIPage = sysUserMapper.selectPage(page, new LambdaQueryWrapper<SysUser>()  .like(StringUtils.hasText(sysUser.getUsername()), SysUser::getUsername, sysUser.getUsername())  );  return sysUserIPage;  }  @Override  @Transactional  public boolean save(SysUser entity) {  boolean save = super.save(entity);  if (save) {  List<Long> roleIdList = entity.getRoleIdList();  //写中间表  List<SysUserRole> sysUserRoles = new ArrayList<>(roleIdList.size() \* 2);  roleIdList.forEach(roleId -> {  SysUserRole sysUserRole = new SysUserRole();  sysUserRole.setUserId(entity.getUserId());  sysUserRole.setRoleId(roleId);  sysUserRoles.add(sysUserRole);  //最好不要在循环里面操作数据库  // sysUserRoleMapper.insert(sysUserRole);  });  //统一操作数据库 xml形式 foreach  sysUserRoleService.saveBatch(sysUserRoles);  }  return save;  }  } |

## ego-product商品服务的搭建

功能定位：商品curd

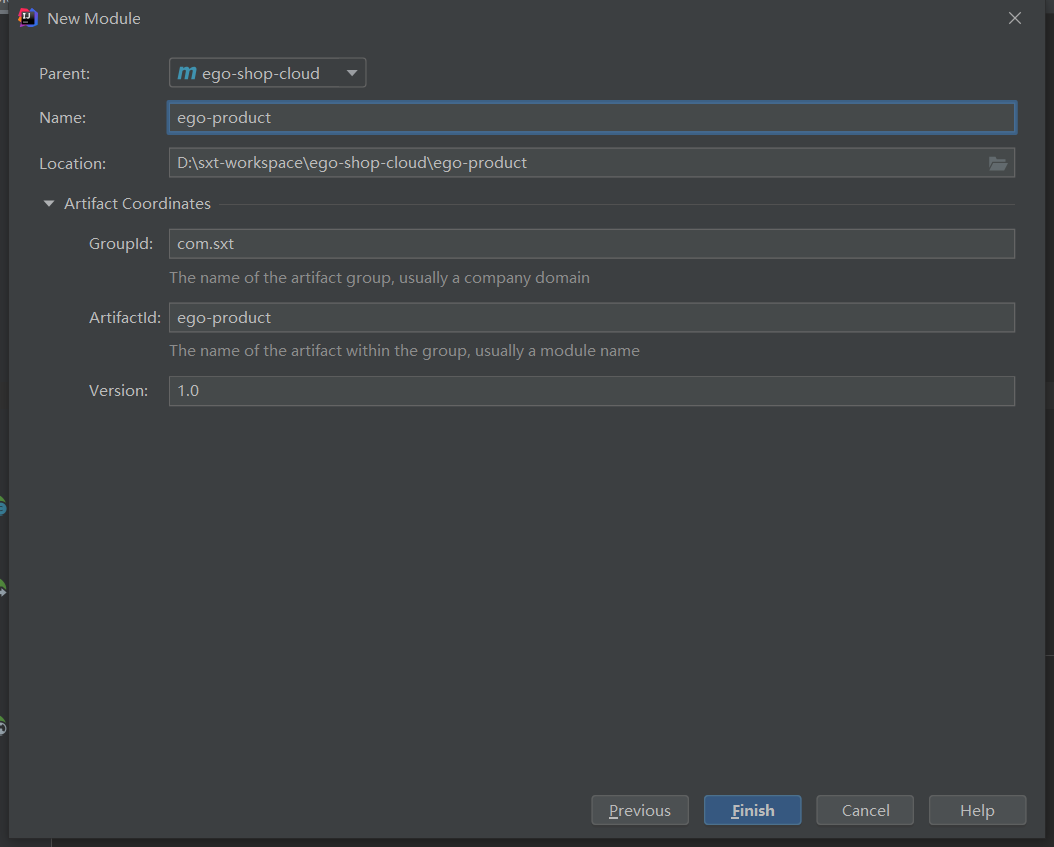
其他模块集成：导入模块等 导入es

### 商品表关系图



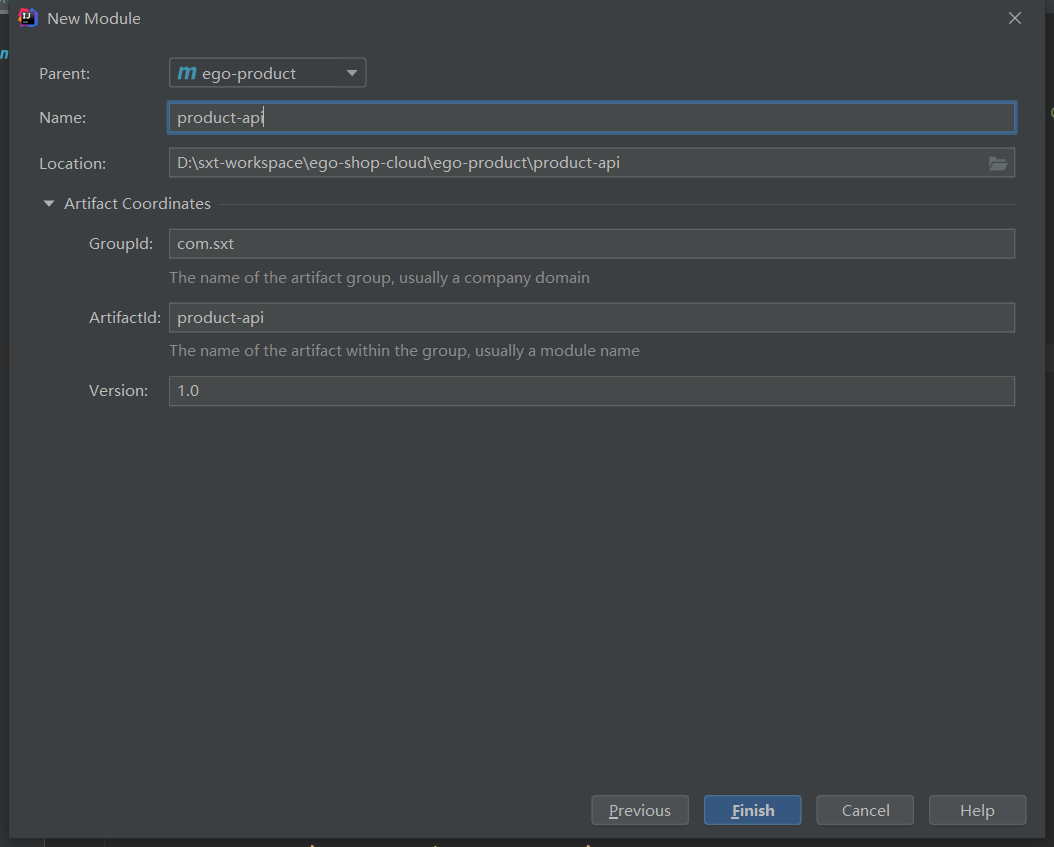
### ego-product父模块搭建

#### 在ego-sho-cloud下面新建一个maven模块



### product-api模块搭建

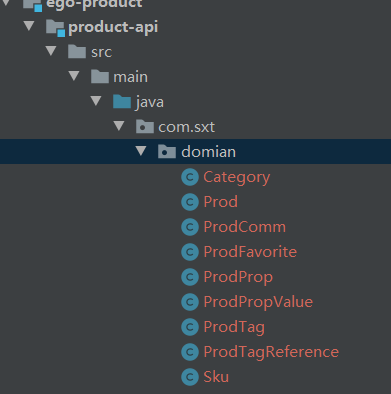
#### 在ego-product下面新建一个maven模块



#### 修改pom.xml文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <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/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>ego-product</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>product-api</artifactId>  <description>ego商城的商品服务api</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>ego-common</artifactId>  <version>1.0</version>  </dependency>  <!--es的依赖 启动导入商品和增量导入-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-elasticsearch</artifactId>  </dependency>  </dependencies>  </project> |

#### 逆向生成实体类



#### 注意实体类的类型

Date类型添加注解

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss", timezone = "GMT+8")

Byte类型改为Integer类型

#### 修改Prod实体类(添加字段属性)

|  |
| --- |
| @ApiModelProperty("商品里面的所有sku")  @TableField(exist = false)  private List<Sku> skuList;  @ApiModelProperty("商品里面所有的标签id")  @TableField(exist = false)  private List<Long> tagList;  @ApiModelProperty("商品的好评数量")  @TableField(exist = false)  private Long praiseNumber;  @ApiModelProperty("商品的好评率")  @TableField(exist = false)  private BigDecimal positiveRating;  @ApiModelProperty("商品的物流模型")  @TableField(exist = false)  private DeliveryModeVo deliveryModeVo;  //配送信息的json  @Data  public static class DeliveryModeVo implements Serializable {  @ApiModelProperty("商家配送")  boolean hasShopDelivery;  @ApiModelProperty("用户自提")  boolean hasUserPickUp;  } |

#### 修改ProdComm实体类(添加字段属性)

|  |
| --- |
| @TableField(exist = false)  @ApiModelProperty(value = "商品的名称")  private String prodName; |

#### 修改ProdProp实体类(添加字段属性)

|  |
| --- |
| @TableField(exist = false)  @ApiModelProperty("商品属性的具体值")  private List<ProdPropValue> prodPropValues; |

#### 创建常量ProductConstant

|  |
| --- |
| public class ProductConstant {  //所有分类 key  public static final String ALL\_CATEGORIES = "'all\_categories'";  //所有父分类 key  public static final String SUB\_CATEGORIES = "'sub\_categories'";  } |

#### 创建评论ProdCommVo

|  |
| --- |
| @Data  public class ProdCommVo implements Serializable {  @ApiModelProperty("商品的id")  private Long prodId;  @ApiModelProperty("商品的好评数")  private Long praiseNumber;  @ApiModelProperty("商品的好评率")  private BigDecimal positiveRating;  @ApiModelProperty("总的评论数")  private Long number;  @ApiModelProperty("中评论数")  private Long secondaryNumber;  @ApiModelProperty("有图的评论")  private Long picNumber;  @ApiModelProperty("差评的数")  private Long negativeNumber;  } |

#### 前台评论对象ProdCommSxt

|  |
| --- |
| @Data  public class ProdCommSxt implements Serializable {  private Long prodId;  private Date recTime;  private String pic;  private String nickName;  private Integer score;  private String content;  private String pics;  private String replyContent;  } |

#### 创建ProductEs(es实体类)

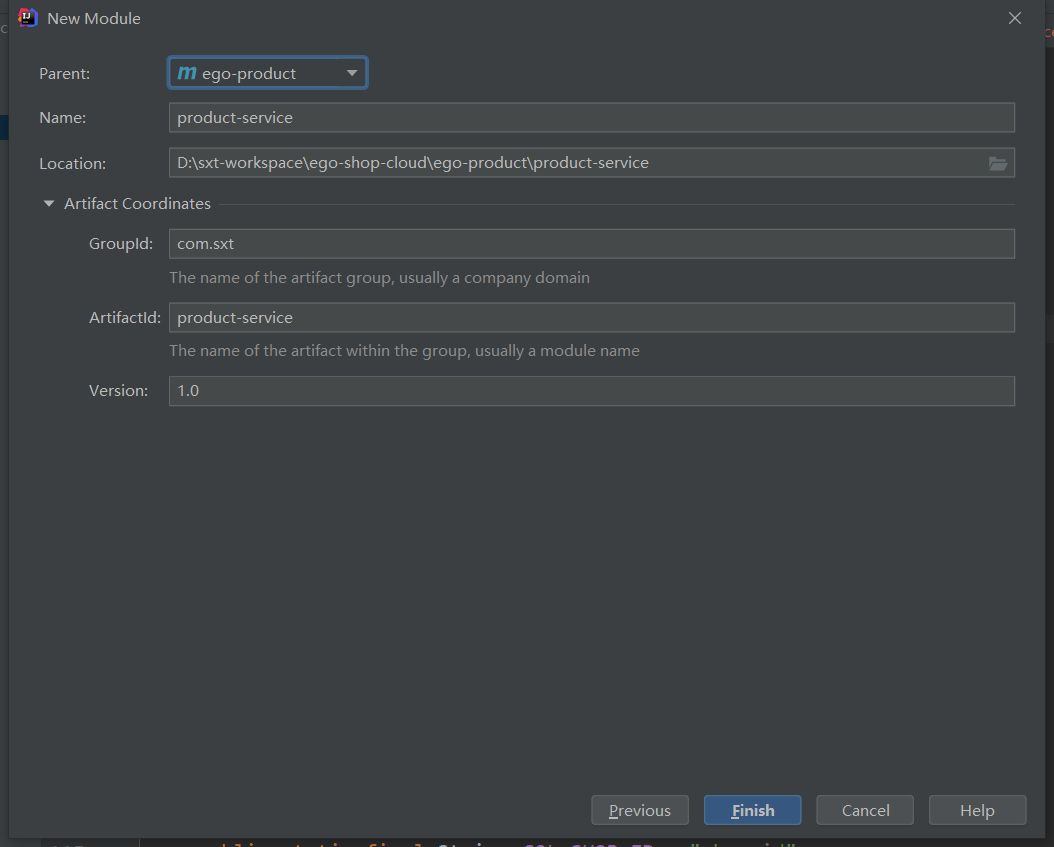
|  |
| --- |
| @Data  @AllArgsConstructor  @NoArgsConstructor  @Document(indexName = "product\_index", createIndex = true, shards = 2, replicas = 1, refreshInterval = "1s")  @Mapping  public class ProductEs {  @Id  @Field  @ApiModelProperty("商品的id")  private Long prodId;  @ApiModelProperty("商品的名称")  @Field(type = FieldType.Text, analyzer = "ik\_max\_word", searchAnalyzer = "ik\_smart")  private String prodName;  @ApiModelProperty("商品的价格")  @Field(type = FieldType.Double)  private BigDecimal price;  @ApiModelProperty("商品的销量")  @Field(type = FieldType.Long)  private Long soldNum;  @ApiModelProperty("商品的买点")  @Field(type = FieldType.Text, analyzer = "ik\_max\_word", searchAnalyzer = "ik\_smart")  private String brief;  @ApiModelProperty("商品的主图")  @Field(type = FieldType.Text)  private String pic;  @ApiModelProperty("商品的状态")  @Field(type = FieldType.Integer)  private Integer status;  @ApiModelProperty("商品的库存")  @Field(type = FieldType.Integer)  private Long totalStocks;  @ApiModelProperty("商品的分类id")  @Field(type = FieldType.Long)  private Long categoryId;  @ApiModelProperty("商品的标签")  @Field(type = FieldType.Text)  private List<Long> tagList;  @ApiModelProperty("商品的好评数")  @Field(type = FieldType.Long)  private Long praiseNumber;  @ApiModelProperty("商品的好评率")  @Field(type = FieldType.Double)  private BigDecimal positiveRating;  } |

#### 新增ProductDao

|  |
| --- |
| @Repository  public interface ProductDao extends ElasticsearchRepository<ProductEs, Long> {  } |

### product-service模块搭建

#### 在ego-product下面新建一个maven模块



#### 修改pom.xml文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <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/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>ego-product</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>product-service</artifactId>  <description>ego商城商品服务service</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>product-api</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-bus-amqp</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-config-client</artifactId>  </dependency>  <!-- fastDFS的依赖-->  <dependency>  <groupId>com.github.tobato</groupId>  <artifactId>fastdfs-client</artifactId>  <version>${fastDFS.version}</version>  </dependency>  <!-- spring的切面编程-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-aop</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis-reactive</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  <!-- 引入druid starter-->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid-spring-boot-starter</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

#### 创建bootstrap.yml配置文件

|  |
| --- |
| spring:  application:  name: product-service  cloud:  config:  discovery:  enabled: true  service-id: config-server  profile: dev  name: product-service  label: master  eureka:  client:  service-url:  defaultZone: ${EUREKA\_SERVER:http://admin:admin@localhost:8761/eureka}  instance:  hostname: ${APP\_HOST:localhost}  prefer-ip-address: true  instance-id: ${spring.application.name}:${server.port}  lease-expiration-duration-in-seconds: 30  lease-renewal-interval-in-seconds: 10 |

#### 创建远端product-service/product-service.yml

|  |
| --- |
| server:  port: ${APP\_PORT:8082}  spring:  datasource:  druid:  url: jdbc:mysql://localhost:3306/ego-shop?useSSL=false&serverTimezone=GMT%2B8  username: root  password: root  driver-class-name: com.mysql.cj.jdbc.Driver  max-active: 30  min-idle: 3  initial-size: 3  validation-query: select 1  max-wait: 5000  stat-view-servlet:  enabled: true  login-username: admin  login-password: admin  allow:  deny:  url-pattern:  - /druid/\*  mvc:  format:  date: yyyy-MM-dd HH:mm:ss  rabbitmq:  host: 192.168.226.129  password: admin  username: admin  port: 5672  listener:  simple:  acknowledge-mode: manual #手动签收消息 默认是自动的  redis:  host: 192.168.226.129  port: 6380  password: cxs1013??  database: 0  elasticsearch:  rest:  uris: http://192.168.226.129:9200  esimport: #每次导入的数量  size: 20  management:  endpoints:  web:  exposure:  include: '\*'  mybatis-plus:  mapper-locations: classpath:mapper/\*\*/\*.xml  configuration:  log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  fdfs:  so-timeout: 1501  connect-timeout: 601  tracker-list: 192.168.226.129:22122  feign:  hystrix:  enabled: true  ribbon:  ReadTimeout: 5000  ConnectTimeout: 5000  hystrix: #hystrix的全局控制  command:  default: #default是全局控制，也可以换成单个方法控制，把default换成方法名即可  fallback:  isolation:  semaphore:  maxConcurrentRequests: 1000  circuitBreaker:  enabled: true #开启断路器  requestVolumeThreshold: 20 #失败次数（阀值）  sleepWindowInMilliseconds: 30000 #窗口时间  errorThresholdPercentage: 60 #失败率  execution:  isolation:  strategy: SEMAPHORE #使用信号量隔离方式  thread:  timeoutInMilliseconds: 5000  semaphore:  maxConcurrentRequests: 1000 #最大并发量  swagger2:  base-package: com.sxt.controller  name: sxt  url: https://gitee.com/smiledouble  email: 775610843@qq.com  title: 商品管理系统接口  description: 实现商品管理系统的所有接口  license: cxs  license-url: https://gitee.com/smiledouble  terms-of-service-url: https://gitee.com/smiledouble |

#### 添加启动类

|  |
| --- |
| @SpringBootApplication  @EnableEurekaClient  @EnableCaching //开启缓存，默认托管到redis中  @EnableFeignClients  @EnableScheduling //开启定时任务 增量导入  public class ProductServiceApplication {  public static void main(String[] args) {  SpringApplication.run(ProductServiceApplication.class, args);  }  } |

#### 添加redis的key序列化配置RedisCacheConfig（主要是在redis里面可以看得懂）

|  |
| --- |
| @Configuration  public class RedisCacheConfig {  //使用了cache的注解，需要手动设置序列化  private CacheProperties cacheProperties;  /\*\*  \* 该配置类被构造时，它里面的参数，将有Spring 的ioc 容器提供  \*  \* @param cacheProperties  \*/  public RedisCacheConfig(CacheProperties cacheProperties) {  this.cacheProperties = cacheProperties;  }  /\*\*  \* 定义Redis缓存的序列化的形式  \* 在这个RedisSerializer类里面提交了3 种常用的序列化形式  \* 1 java jdk的序列化 (默认的形式)  \* 2 json jackson （我们可以手动切换）  \* 3 String String的序列化(主要用在字符串的存储，不是对象) 对key 一般使用的是String的序列化  \*/  @Bean  public RedisCacheConfiguration redisCacheConfiguration() {  CacheProperties.Redis redisProperties = this.cacheProperties.getRedis();  RedisCacheConfiguration config = RedisCacheConfiguration  .defaultCacheConfig();  config = config.serializeValuesWith(  RedisSerializationContext.SerializationPair.fromSerializer(RedisSerializer.json()));  // 缓存的过期时间，一般我们在配置文件里面设置  if (redisProperties.getTimeToLive() != null) {  config = config.entryTtl(redisProperties.getTimeToLive());  }  // 是否给缓存添加前缀  if (redisProperties.getKeyPrefix() != null) {  config = config.prefixKeysWith(redisProperties.getKeyPrefix());  }  // 空值的缓存  if (!redisProperties.isCacheNullValues()) {  config = config.disableCachingNullValues();  }  //禁用key  if (!redisProperties.isUseKeyPrefix()) {  config = config.disableKeyPrefix();  }  return config;  }  } |

#### 添加CategoryController(商品分类管理)

|  |
| --- |
| @RestController  @RequestMapping("/prod/category")  @Api(tags = "商品分类管理")  public class CategoryController {  @Autowired  private CategoryService categoryService;  @GetMapping({"/table"})  @PreAuthorize("hasAuthority('prod:category:page')")  @ApiOperation("分类全查询")  public ResponseEntity<List<Category>> list() {  List<Category> list = this.categoryService.list();  return ResponseEntity.ok(list);  }  @DeleteMapping("{id}")  @ApiOperation("删除商品分类")  @PreAuthorize("hasAuthority('prod:category:delete')")  public ResponseEntity<Void> deleteCateGory(@PathVariable Long id) {  categoryService.removeById(id);  return ResponseEntity.ok().build();  }  @GetMapping("/info/{id}")  @ApiOperation("获得一个分类对象")  @PreAuthorize("hasAuthority('prod:category:info')")  public ResponseEntity<Category> info(@PathVariable Long id) {  Category category = categoryService.getById(id);  return ResponseEntity.ok().body(category);  }  @PostMapping  @PreAuthorize("hasAuthority('prod:category:save')")  @ApiOperation("新增一个分类")  public ResponseEntity<Void> save(@RequestBody @Validated Category category) {  this.categoryService.save(category);  return ResponseEntity.ok().build();  }  @PutMapping  @ApiOperation("更新分类")  @PreAuthorize("hasAuthority('prod:category:update')")  public ResponseEntity<Void> updateCateGory(@RequestBody @Validated Category category) {  categoryService.updateById(category);  return ResponseEntity.ok().build();  }  @GetMapping({"listCategory"})  @PreAuthorize("hasAuthority('prod:category:page')")  @ApiOperation("列出所有分类的父亲")  public ResponseEntity<List<Category>> listAllParent() {  List<Category> categories = this.categoryService.listAllParent();  return ResponseEntity.ok(categories);  }  } |

#### 修改CategoryService

|  |
| --- |
| public interface CategoryService extends IService<Category> {  /\*\*  \* 查询所有分类的父节点  \*  \* @return  \*/  List<Category> listAllParent();  } |

#### 修改CategoryServiceImpl

|  |
| --- |
| @Service  @Slf4j  @CacheConfig(cacheNames = "com.sxt.service.impl.CategoryServiceImpl")  public class CategoryServiceImpl extends ServiceImpl<CategoryMapper, Category> implements CategoryService {  @Autowired  private CategoryMapper categoryMapper;  @Autowired  private ProdMapper prodMapper;  @Autowired  private FastFileStorageClient fastFileStorageClient;  /\*\*  \* 查询所有的商品分类，存放到redis中  \*  \* @return  \*/  @Cacheable(key = ProductConstant.ALL\_CATEGORIES)  @Override  public List<Category> list() {  List<Category> list = super.list();  return list;  }  /\*\*  \* 新增一个分类 清空redis  \*  \* @param entity  \* @return  \*/  @CacheEvict(key = ProductConstant.ALL\_CATEGORIES)  @Override  public boolean save(Category entity) {  log.info("新增一个分类开始,名称为{}", entity.getCategoryName());  //校验  validate(entity);  entity.setRecTime(new Date());  entity.setUpdateTime(new Date());  entity.setStatus(1);  entity.setShopId(1L);  entity.setSeq(1);  return super.save(entity);  }  /\*\*  \* 校验  \*  \* @param entity  \*/  private void validate(Category entity) {  if (entity.getParentId() == null || entity.getParentId().equals(0L)) {  //说明新增的是一个父节点  entity.setParentId(0L);  //分类层级 第一级别  entity.setGrade(1);  } else {  //查询一下父id有没有值  Category parent = this.getById(entity.getParentId());  if (ObjectUtils.isEmpty(parent)) {  throw new IllegalArgumentException("新增分类时选的父节点不能为空");  }  entity.setParentId(parent.getCategoryId());  //这里设置级别  int subNode = parent.getGrade() + 1;  entity.setGrade(subNode);  }  }  /\*\*  \* 删除一个分类 清空缓存  \*  \* @param id  \* @return  \*/  @CacheEvict(key = ProductConstant.ALL\_CATEGORIES)  @Override  public boolean removeById(Serializable id) {  log.info("删除商品分类的id为{}", id);  //判断有没有子分类  Integer count = categoryMapper.selectCount(new LambdaQueryWrapper<Category>()  .eq(Category::getParentId, id)  );  if (count > 0) {  //说明有子分类，不能删除  throw new IllegalArgumentException("该分类有子分类，不能删除");  }  //查询有没有商品再用此分类  Integer count1 = prodMapper.selectCount(new LambdaQueryWrapper<Prod>()  .eq(Prod::getCategoryId, id)  );  if (count1 > 0) {  throw new IllegalArgumentException("有商品处于该分类中，不能删除");  }  return super.removeById(id);  }  /\*\*  \* 更新一个分类 清空缓存  \*  \* @param entity  \* @return  \*/  @CacheEvict(key = ProductConstant.ALL\_CATEGORIES)  @Override  public boolean updateById(Category entity) {  log.info("修改分类开始，被修改的id为{}", entity.getCategoryId());  //需改注意图片，先得到原来的独享  Category oldCateGory = this.getById(entity.getCategoryId());  if (ObjectUtils.isEmpty(oldCateGory)) {  throw new IllegalArgumentException("修改的值不存在");  }  @NotBlank String pic = oldCateGory.getPic();  //看一下图片  if (!pic.equals(entity.getPic())) {  //如果图片被修改了，则从fastDFS里删掉  String group = FastDFSUtil.parseGroup(pic);  try {  fastFileStorageClient.deleteFile(group, pic.replaceFirst(group + "/", ""));  } catch (Exception e) {  log.error("图片删除失败,地址为{}", pic);  }  }  return super.updateById(entity);  }  /\*\*  \* 查询所有分类的父节点  \*  \* @return  \*/  @Cacheable(key = ProductConstant.SUB\_CATEGORIES)  @Override  public List<Category> listAllParent() {  List<Category> parentCategory = categoryMapper.selectList(new LambdaQueryWrapper<Category>()  .eq(Category::getGrade, 1)  );  return parentCategory;  }  } |

#### 创建ProdCommController（商品评论管理）

|  |
| --- |
| @RestController  @RequestMapping("/prod/prodComm")  @Api(tags = "商品评论管理")  public class ProdCommController {  @Autowired  private ProdCommService prodCommService;  @GetMapping("page")  @ApiOperation("分页查询评论")  @PreAuthorize("hasAuthority('prod:prodComm:page')")  public ResponseEntity<IPage<ProdComm>> loadProdCommByPage(Page<ProdComm> page, ProdComm prodComm) {  IPage<ProdComm> prodCommIPage = prodCommService.loadProdCommByPage(page, prodComm);  return ResponseEntity.ok(prodCommIPage);  }  /\*\*  \* 需要返回什么数据  \* 总评多少个  \* 好评多少个  \* 好频率多少  \* 中评多少个  \* 差评多少个  \* 带图的多少个  \*  \* @param prodId  \* @return  \*/  @GetMapping("prodComm/prodCommData")  @ApiOperation("根据商品id查询商品评论总览")  public ResponseEntity<ProdCommVo> getProdCommAll(@RequestParam("prodId") Long prodId) {  ProdCommVo prodCommVo = prodCommService.getProdCommAll(prodId);  return ResponseEntity.ok(prodCommVo);  }  @GetMapping("prodComm/prodCommPageByProd")  @ApiOperation("分页查询前台的评论")  public ResponseEntity<IPage<ProdCommSxt>> prodCommPageByProd(Page<ProdComm> page, ProdComm prodComm) {  IPage<ProdCommSxt> prodCommSxtIPage = prodCommService.prodCommPageByProd(page, prodComm);  return ResponseEntity.ok(prodCommSxtIPage);  }  @GetMapping("{id}")  @ApiOperation("获得一条评论数据")  @PreAuthorize("hasAuthority('prod:prodComm:info')")  public ResponseEntity<ProdCommVo> findById(@PathVariable("id") Long id) {  ProdCommVo byId = prodCommService.getProdCommById(id);  return ResponseEntity.ok().body(byId);  }  @PutMapping  @ApiOperation("评论的修改")  @PreAuthorize("hasAuthority('prod:prodComm:update')")  public ResponseEntity<Void> update(@RequestBody @Validated ProdComm prodComm) {  this.prodCommService.updateById(prodComm);  return ResponseEntity.ok().build();  }  } |

#### 修改ProdCommService

|  |
| --- |
| public interface ProdCommService extends IService<ProdComm> {  /\*\*  \* 分页查询商品的评论  \*  \* @param page  \* @param prodComm  \* @return  \*/  IPage<ProdComm> loadProdCommByPage(Page<ProdComm> page, ProdComm prodComm);  /\*\*  \* 根据id查询评论  \*  \* @param id  \* @return  \*/  ProdCommVo getProdCommById(Long id);  } |

#### 修改ProdCommServiceImpl

|  |
| --- |
| @Service  @Slf4j  public class ProdCommServiceImpl extends ServiceImpl<ProdCommMapper, ProdComm> implements ProdCommService {  @Autowired  private ProdCommMapper prodCommMapper;  @Autowired  private ProdMapper prodMapper;  /\*\*  \* 分页查询商品的评论  \* 商品名称没查询  \*  \* @param page  \* @param prodComm  \* @return  \*/  @Override  public IPage<ProdComm> loadProdCommByPage(Page<ProdComm> page, ProdComm prodComm) {  IPage<ProdComm> prodCommIPage = new Page<>();  page.addOrder(OrderItem.desc("rec\_time"));  //根据商品名称查询prodIds  String prodName = prodComm.getProdName();  List<Object> prodIds = null;  if (!StringUtils.isEmpty(prodName)) {  //如果有名称 查询prodIds  prodIds = prodMapper.selectObjs(new LambdaQueryWrapper<Prod>()  .select(Prod::getProdId)  .like(Prod::getProdName, prodName)  );  if (CollectionUtils.isEmpty(prodIds)) {  prodCommIPage.setRecords(Collections.emptyList());  prodCommIPage.setTotal(0L);  return prodCommIPage;  }  }  //查询所有评论 条件就是 如果有prodIds 就带上 没有就不带  prodCommIPage = prodCommMapper.selectPage(page, new LambdaQueryWrapper<ProdComm>()  .eq(prodComm.getStatus() != null, ProdComm::getStatus, prodComm.getStatus())  .in(!CollectionUtils.isEmpty(prodIds), ProdComm::getProdId, prodIds)  );  List<ProdComm> records = prodCommIPage.getRecords();  if (!CollectionUtils.isEmpty(records)) {  //拿到商品ids 然后查询商品表 然后组装数据 filter 过滤 条件 map 抽取 collect 转换集合  //stream 流 1 过程filter，map，sort，limit， 2终止foreach collect  List<Long> pIds = records.stream().map(ProdComm::getProdId).collect(Collectors.toList());  List<Prod> prods = prodMapper.selectList(new LambdaQueryWrapper<Prod>()  .in(Prod::getProdId, pIds)  );  //在代码里面做组装  records.forEach(r -> {  Prod prod1 = prods.stream()  .filter(prod -> prod.getProdId().equals(r.getProdId()))  .collect(Collectors.toList())  .get(0);  r.setProdName(prod1.getProdName());  });  }  return prodCommIPage;  }  /\*\*  \* 根据id查询评论  \*  \* @param id  \* @return  \*/  @Override  public ProdCommVo getProdCommById(Long id) {  ProdCommVo prodCommVo = new ProdCommVo();  ProdComm prodComm = prodCommMapper.selectOne(new LambdaQueryWrapper<ProdComm>()  .eq(ProdComm::getProdCommId, id)  );  if (ObjectUtils.isEmpty(prodComm)) {  //如果评论为空，则返回空  throw new IllegalArgumentException("评论id为空");  }  Prod prod = prodMapper.selectOne(new LambdaQueryWrapper<Prod>()  .eq(Prod::getProdId, prodComm.getProdId())  );  if (!ObjectUtils.isEmpty(prod)) {  prodCommVo.setProdName(prod.getProdName());  }  BeanUtil.copyProperties(prodComm, prodCommVo, true);  return prodCommVo;  }  } |

#### 创建ProdPropController（商品规格管理）

|  |
| --- |
| @RestController  @RequestMapping("/prod/spec")  @Api(tags = "商品的规格管理")  public class ProdPropController {  @Autowired  private ProdPropService prodPropService;  @GetMapping("/page")  @ApiOperation("分页查询")  @PreAuthorize("hasAuthority('prod:spec:page')")  public ResponseEntity<IPage<ProdProp>> page(Page<ProdProp> page, ProdProp prodProp) {  IPage<ProdProp> prodPropIPage = prodPropService.findByPage(page, prodProp);  return ResponseEntity.ok(prodPropIPage);  }  @DeleteMapping({"/{id}"})  @PreAuthorize("hasAuthority('prod:spec:delete')")  @ApiOperation("删除属性")  public ResponseEntity<Void> delete(@PathVariable("id") Long id) {  this.prodPropService.removeById(id);  return ResponseEntity.ok().build();  }  @PostMapping  @PreAuthorize("hasAuthority('prod:spec:save')")  @ApiOperation("新增一个属性的值")  public ResponseEntity<Void> add(@RequestBody @Validated ProdProp prodProp) {  this.prodPropService.save(prodProp);  return ResponseEntity.ok().build();  }  @PutMapping  @PreAuthorize("hasAuthority('prod:spec:update')")  @ApiOperation("修改一个属性的值")  public ResponseEntity<Void> update(@RequestBody @Validated ProdProp prodProp) {  this.prodPropService.updateById(prodProp);  return ResponseEntity.ok().build();  }  @GetMapping({"/info/{id}"})  @PreAuthorize("hasAuthority('prod:spec:info')")  @ApiOperation("属性值的回显")  public ResponseEntity<ProdProp> findById(@PathVariable("id") Long id) {  ProdProp byId = this.prodPropService.getById(id);  return ResponseEntity.ok(byId);  }  @GetMapping({"/list"})  @PreAuthorize("hasAuthority('prod:spec:info')")  @ApiOperation("属性的回显")  public ResponseEntity<List<ProdProp>> list() {  List<ProdProp> list = this.prodPropService.list();  return ResponseEntity.ok(list);  }  @GetMapping({"/listSpecValue/{id}"})  @PreAuthorize("hasAuthority('prod:spec:info')")  @ApiOperation("属性值的回显")  public ResponseEntity<List<ProdPropValue>> listSpecValue(@PathVariable("id") Long id) {  List<ProdPropValue> prodPropValue = this.prodPropService.listSpecValue(id);  return ResponseEntity.ok(prodPropValue);  }  } |

#### 修改ProdPropService

|  |
| --- |
| public interface ProdPropService extends IService<ProdProp> {  /\*\*  \* 分页查询商品的属性，规格  \*  \* @param page  \* @param prodProp  \* @return  \*/  IPage<ProdProp> findByPage(Page<ProdProp> page, ProdProp prodProp);  /\*\*  \* 属性值的回显  \*  \* @param id  \* @return  \*/  List<ProdPropValue> listSpecValue(Long id);  } |

#### 修改ProdPropServiceImpl

|  |
| --- |
| @Service  @Slf4j  public class ProdPropServiceImpl extends ServiceImpl<ProdPropMapper, ProdProp> implements ProdPropService {  @Autowired  private ProdPropMapper prodPropMapper;  @Autowired  private ProdPropValueMapper prodPropValueMapper;  /\*\*  \* 分页查询商品的属性，规格  \*  \* @param page  \* @param prodProp  \* @return  \*/  @Override  public IPage<ProdProp> findProdPropByPage(Page<ProdProp> page, ProdProp prodProp) {  //分页查询  IPage<ProdProp> prodPropIPage = prodPropMapper.selectPage(page, new LambdaQueryWrapper<ProdProp>()  .like(StringUtils.hasText(prodProp.getPropName()), ProdProp::getPropName, prodProp.getPropName())  );  List<ProdProp> prodProps = prodPropIPage.getRecords();  if (!CollectionUtils.isEmpty(prodProps)) {  //拿到属性名称ids集合  List<Long> propIds = prodProps.stream().map(ProdProp::getPropId).collect(Collectors.toList());  //查询属性值集合  List<ProdPropValue> prodPropValues = prodPropValueMapper.selectList(new LambdaQueryWrapper<ProdPropValue>()  .in(ProdPropValue::getPropId, propIds)  );  //循环组装数据  prodProps.forEach(prodProp1 -> {  List<ProdPropValue> prodPropValueList = prodPropValues.stream().filter(prodPropValue -> prodProp1.getPropId().equals(prodPropValue.getPropId())).collect(Collectors.toList());  prodProp1.setProdPropValues(prodPropValues);  });  }  return prodPropIPage;  }  /\*\*  \* 属性值的回显  \*  \* @param id  \* @return  \*/  @Override  public List<ProdPropValue> listSpecValue(Long id) {  List<ProdPropValue> prodPropValues = prodPropValueMapper.selectList(new LambdaQueryWrapper<ProdPropValue>()  .eq(ProdPropValue::getPropId, id)  );  return prodPropValues;  }  /\*\*  \* 新增属性和属性值  \*  \* @param entity  \* @return  \*/  @Override  @Transactional  public boolean save(ProdProp entity) {  int insert = prodPropMapper.insert(entity);  List<ProdPropValue> prodPropValues = entity.getProdPropValues();  List<ProdPropValue> prodPropValueList = new ArrayList<>(prodPropValues.size() \* 2);  prodPropValues.forEach(prodPropValue -> {  ProdPropValue prodPropValue1 = new ProdPropValue();  prodPropValue1.setPropId(entity.getPropId());  prodPropValue1.setPropValue(prodPropValue.getPropValue());  prodPropValueList.add(prodPropValue1);  });  prodPropValueService.saveBatch(prodPropValueList);  return insert > 0;  }} |

#### 创建ProdTagController(商品标签分组管理)

|  |
| --- |
| @RestController  @Api(tags = "商品标签分组管理")  @RequestMapping("prod/prodTag")  public class ProdTagController {  @Autowired  private ProdTagService prodTagService;  @GetMapping("/page")  @ApiOperation("分页查询商品的标签")  @PreAuthorize("hasAuthority('prod:prodTag:page')")  public ResponseEntity<IPage<ProdTag>> findByPage(Page<ProdTag> page, ProdTag prodTag) {  IPage<ProdTag> pageData = prodTagService.findByPage(page, prodTag);  return ResponseEntity.ok(pageData);  }  @GetMapping("/info/{id}")  @PreAuthorize("hasAuthority('prod:prodTag:info')")  @ApiOperation("标签数据的回显")  public ResponseEntity<ProdTag> findById(@PathVariable("id") Long id) {  ProdTag prodTag = prodTagService.getById(id);  return ResponseEntity.ok(prodTag);  }  @PostMapping  @ApiOperation("新增一个商品的标签")  @PreAuthorize("hasAuthority('prod:prodTag:save')")  public ResponseEntity<Void> save(@RequestBody @Validated ProdTag prodTag) {  prodTagService.save(prodTag);  return ResponseEntity.ok().build();  }  @DeleteMapping("/{id}")  @PreAuthorize("hasAuthority('prod:prodTag:delete')")  @ApiOperation("删除商品的标签")  public ResponseEntity<Void> delete(@PathVariable("id") Long id) {  prodTagService.removeById(id);  return ResponseEntity.ok().build();  }  @DeleteMapping  @PreAuthorize("hasAuthority('prod:prodTag:delete')")  @ApiOperation("删除多个商品的标签")  public ResponseEntity<Void> delete(@RequestBody List<Long> ids) {  prodTagService.removeByIds(ids);  return ResponseEntity.ok().build();  }  @PutMapping  @ApiOperation("新增一个商品的标签")  @PreAuthorize("hasAuthority('prod:prodTag:save')")  public ResponseEntity<Void> update(@RequestBody @Validated ProdTag prodTag) {  prodTagService.updateById(prodTag);  return ResponseEntity.ok().build();  }  @GetMapping({"/listTagList"})  @ApiOperation("列出所有的商品标签的列表")  @PreAuthorize("hasAuthority('prod:prodTag:info')")  public ResponseEntity<List<ProdTag>> list() {  List<ProdTag> list = this.prodTagService.list();  return ResponseEntity.ok(list);  }  } |

#### 修改ProdTagService

|  |
| --- |
| public interface ProdTagService extends IService<ProdTag> {  /\*\*  \* 分页查询商品标签（分组）  \*  \* @param page  \* @param prodTag  \* @return  \*/  IPage<ProdTag> findByPage(Page<ProdTag> page, ProdTag prodTag);  } |

#### 修改ProdTagServiceImpl

|  |
| --- |
| @Service  @Slf4j  public class ProdTagServiceImpl extends ServiceImpl<ProdTagMapper, ProdTag> implements ProdTagService {  @Autowired  private ProdTagMapper prodTagMapper;  @Autowired  private ProdTagReferenceMapper prodTagReferenceMapper;  /\*\*  \* 分页查询商品标签（分组）  \*  \* @param page  \* @param prodTag  \* @return  \*/  @Override  public IPage<ProdTag> findByPage(Page<ProdTag> page, ProdTag prodTag) {  page.addOrder(OrderItem.desc("update\_time"));  IPage<ProdTag> prodTagIPage = prodTagMapper.selectPage(page, new LambdaQueryWrapper<ProdTag>()  .like(StringUtils.hasText(prodTag.getTitle()), ProdTag::getTitle, prodTag.getTitle())  .eq(prodTag.getStatus() != null, ProdTag::getStatus, prodTag.getStatus())  );  return prodTagIPage;  }  /\*\*  \* 删除商品的标签  \*  \* @param id  \* @return  \*/  @Override  public boolean removeById(Serializable id) {  log.info("删除商品的标签分组id为{}", id);  //查看有没有商品正在用此分组标签  Integer count = prodTagReferenceMapper.selectCount(new LambdaQueryWrapper<ProdTagReference>()  .eq(ProdTagReference::getTagId, id)  );  if (count > 0) {  log.error("有商品正在使用此标签分组，不能删除");  throw new IllegalArgumentException("有商品正在使用此标签分组，不能删除");  }  return super.removeById(id);  }  } |

#### 创建ProdController(商品管理)

|  |
| --- |
| @RestController  @RequestMapping("/prod/prod")  @Api(tags = "商品管理")  public class ProdController {  @Autowired  private ProdService prodService;  @GetMapping("/page")  @ApiOperation("商品的分页查询")  @PreAuthorize("hasAuthority('prod:prod:page')")  public ResponseEntity<IPage<Prod>> page(Page<Prod> page, Prod prod) {  IPage<Prod> page1 = prodService.findByPage(page, prod);  return ResponseEntity.ok(page1);  }  @DeleteMapping("/{id}")  @ApiOperation("删除一个商品")  @PreAuthorize("hasAuthority('prod:prod:delete')")  public ResponseEntity<Void> delete(@PathVariable("id") Long id) {  prodService.removeById(id);  return ResponseEntity.ok().build();  }  @DeleteMapping  @ApiOperation("删除多个商品")  @PreAuthorize("hasAuthority('prod:prod:delete')")  public ResponseEntity<Void> delete(@RequestBody List<Long> list) {  prodService.removeByIds(list);  return ResponseEntity.ok().build();  }  @PostMapping  @ApiOperation("新增一个商品")  @PreAuthorize("hasAuthority('prod:prod:save')")  public ResponseEntity<Void> save(@RequestBody @Validated Prod prod) {  this.prodService.save(prod);  return ResponseEntity.ok().build();  }  @GetMapping({"/info/{id}"})  @ApiOperation("数据的回显")  @PreAuthorize("hasAuthority('prod:prod:info')")  public ResponseEntity<Prod> findById(@PathVariable("id") Long id) {  Prod prod = prodService.getById(id);  return ResponseEntity.ok(prod);  }  @PutMapping  @ApiOperation("修改一个商品")  @PreAuthorize("hasAuthority('prod:prod:update')")  public ResponseEntity<Void> update(@RequestBody @Validated Prod prod) {  this.prodService.updateById(prod);  return ResponseEntity.ok().build();  }  } |

#### 修改ProdService

|  |
| --- |
| public interface ProdService extends IService<Prod> {  /\*\*  \* 分页查询商品  \*  \* @param page  \* @param prod  \* @return  \*/  IPage<Prod> findByPage(Page<Prod> page, Prod prod);  /\*\*  \* 查询时间段内的商品总条数  \*  \* @param t1  \* @param t2  \* @return  \*/  Integer getTotal(Date t1, Date t2);  /\*\*  \* 分页查询商品数据 导入es的  \*  \* @param prodPage  \* @param t1  \* @param t2  \* @return  \*/  IPage<Prod> findProdByPageToEs(IPage<Prod> prodPage, Date t1, Date t2);  } |

#### 修改ProdServiceImpl

|  |
| --- |
| @Service  @Slf4j  public class ProdServiceImpl extends ServiceImpl<ProdMapper, Prod> implements ProdService {  @Autowired  private ProdMapper prodMapper;  @Autowired  private SkuMapper skuMapper;  @Autowired  private ProdTagReferenceMapper prodTagReferenceMapper;  /\*\*  \* 分页查询商品  \*  \* @param page  \* @param prod  \* @return  \*/  @Override  public IPage<Prod> findByPage(Page<Prod> page, Prod prod) {  page.addOrder(OrderItem.desc("create\_time"));  IPage<Prod> prodIPage = prodMapper.selectPage(page, new LambdaQueryWrapper<Prod>()  .like(StringUtils.hasText(prod.getProdName()), Prod::getProdName, prod.getProdName())  .eq(prod.getStatus() != null, Prod::getStatus, prod.getStatus())  .ne(prod.getStatus() == null, Prod::getStatus, -1)  );  return prodIPage;  }  /\*\*  \* 商品的新增  \* 1.商品表  \* 2.sku表要写 list  \* 3.tag-refrence标签的中间表要写  \*  \* @param prod  \*/  @Override  @Transactional  public void addProd(Prod prod) {  prod.setCreateTime(new Date());  prod.setUpdateTime(new Date());  prod.setPutawayTime(prod.getStatus() == 1 ? new Date() : null);  prod.setVersion(1);  prod.setSoldNum(0);  //配送方式  Prod.DeliveryModeVo deliveryModeVo = prod.getDeliveryModeVo();  String deliveryModeStr = JSON.toJSONString(deliveryModeVo);  prod.setDeliveryMode(deliveryModeStr);  //设置prod库存  List<Sku> skuList = prod.getSkuList();  //规约  Integer reduce = skuList.stream().map(Sku::getStocks).reduce(0, Integer::sum);  // List<Integer> 500 900 600  // Integer reduce = skuList.stream().map(Sku::getStocks).reduce(0, (x, y) -> x + y);  System.out.println(reduce);  Integer prodStock = 0;  //流式操作不能使用非原子性操作 atomic  for (Sku sku : skuList) {  prodStock += sku.getStocks();  }  prod.setTotalStocks(prodStock);  //新增商品  int insert = prodMapper.insert(prod);  if (insert > 0) {  //写sku表  handlerSku(prod.getSkuList(), prod.getProdId());  //写tag的中间表  handlerTagReference(prod.getTagList(), prod.getProdId());  }  }  /\*\*  \* 处理sku  \*  \* @param skuList  \* @param prodId  \*/  private void handlerSku(List<Sku> skuList, Long prodId) {  ArrayList<Sku> skus = new ArrayList<>(skuList.size() \* 2);  skuList.forEach(sku -> {  sku.setUpdateTime(new Date());  sku.setRecTime(new Date());  sku.setVersion(1);  sku.setProdId(prodId);  skus.add(sku);  });  skuService.saveBatch(skus);  }  /\*\*  \* 处理商品标签分组的中间表  \*  \* @param tagList  \* @param prodId  \*/  private void handlerTagReference(List<Long> tagList, Long prodId) {  List<ProdTagReference> prodTagReferences = new ArrayList<>(tagList.size() \* 2);  tagList.forEach(t -> {  ProdTagReference prodTagReference = new ProdTagReference();  prodTagReference.setCreateTime(new Date());  prodTagReference.setProdId(prodId);  prodTagReference.setShopId(1L);  prodTagReference.setStatus(Boolean.TRUE);  prodTagReference.setTagId(Long.valueOf(t));  prodTagReferences.add(prodTagReference);  });  prodTagReferenceService.saveBatch(prodTagReferences);  }  /\*\*  \* 查询一个商品  \*  \* @param id  \* @return  \*/  @Override  public Prod getById(Serializable id) {  Prod prod = super.getById(id);  prod.setDeliveryModeVo(JSONUtil.toBean(prod.getDeliveryMode(), Prod.DeliveryModeVo.class));  //查询sku  List<Sku> skus = getProdSkuList(id);  prod.setSkuList(skus);  //查询商品的标签分组关系  List<Long> tags = getTagList(id);  prod.setTagList(tags);  return prod;  }  /\*\*  \* 根据商品的id 查询商品标签的关系  \*  \* @param id  \* @return  \*/  private List<Long> getTagList(Serializable id) {  List<Object> objects = prodTagReferenceMapper.selectObjs(new LambdaQueryWrapper<ProdTagReference>()  .select(ProdTagReference::getTagId)  .eq(ProdTagReference::getProdId, id)  );  if (ObjectUtils.isEmpty(objects)) {  return Collections.emptyList();  }  List<Long> tagIds = new ArrayList<>(objects.size() \* 2);  objects.forEach(k -> tagIds.add((Long) k));  return tagIds;  }  /\*\*  \* 根据商品的id查询商品的sku  \*  \* @param id  \* @return  \*/  private List<Sku> getProdSkuList(Serializable id) {  List<Sku> skus = skuMapper.selectList(new LambdaQueryWrapper<Sku>()  .eq(Sku::getProdId, id)  );  if (ObjectUtils.isEmpty(skus)) {  return Collections.emptyList();  }  return skus;  }  /\*\*  \* 修改一个商品  \*  \* @param entity  \* @return  \*/  @Override  @Transactional  public boolean updateById(Prod entity) {  log.info("修改一个商品为{}", JSONUtil.toJsonStr(entity));  entity.setUpdateTime(new Date());  if (entity.getStatus().equals(1)) {  //说明是上架  entity.setPutawayTime(new Date());  }  //设置版本号  entity.setVersion(entity.getVersion()==null?1 :entity.getVersion()+ 1);  boolean b = super.updateById(entity);  if (b) {  //修改sku  updateSkuList(entity.getProdId(), entity.getSkuList());  //修改tag  updateTagList(entity.getProdId(), entity.getTagList());  }  return super.updateById(entity);  }  /\*\*  \* 修改tag  \*  \* @param prodId  \* @param tagList  \*/  private void updateTagList(Long prodId, List<Long> tagList) {  log.info("更新商品{}的tag关系", prodId);  //把之前的删掉，然后重新写入即可  prodTagReferenceMapper.delete(new LambdaQueryWrapper<ProdTagReference>()  .eq(ProdTagReference::getProdId, prodId)  );  handlerTagList(prodId, tagList);  }  /\*\*  \* 修改sku  \*  \* @param prodId  \* @param skuList  \*/  private void updateSkuList(Long prodId, List<Sku> skuList) {  log.info("更新商品{}的sku", prodId);  //先删掉sku  skuMapper.delete(new LambdaQueryWrapper<Sku>()  .eq(Sku::getProdId, prodId)  );  //如果sku列表不为空，则写进去  handlerSkuList(prodId, skuList);  }  /\*\*  \* 1.删除一个商品  \* 2.删除sku  \* 3.删除标签分组的关系  \*  \* @param id  \* @return  \*/  @Override  public boolean removeById(Serializable id) {  log.info("移除商品{}", id);  //这里做逻辑删除，直接是更新操作  Prod prod = new Prod();  prod.setProdId((Long) id);  prod.setStatus(-1);  int flag = prodMapper.updateById(prod);  //处理sku  removeSkuRealtion(id);  // 和tag\_reference  removeTagRealtion(id);  return flag > 0;  }  /\*\*  \* 处理tag关系的删除  \*  \* @param id  \*/  private void removeTagRealtion(Serializable id) {  log.info("移除商品{}的tag关系", id);  prodTagReferenceMapper.delete(new LambdaQueryWrapper<ProdTagReference>()  .eq(ProdTagReference::getProdId, id)  );  }  /\*\*  \* 处理sku的删除  \*  \* @param id  \*/  private void removeSkuRealtion(Serializable id) {  log.info("移除商品{}的sku", id);  skuMapper.delete(new LambdaQueryWrapper<Sku>()  .eq(Sku::getProdId, id)  );  }  /\*\*  \* 查询时间段内的商品总条数  \*  \* @param t1  \* @param t2  \* @return  \*/  @Override  public Integer getTotal(Date t1, Date t2) {  Integer count = prodMapper.selectCount(new LambdaQueryWrapper<Prod>()  .between(t1 != null && t2 != null, Prod::getUpdateTime, t1, t2)  );  return count;  }  /\*\*  \* 分页查询商品数据 导入es的  \*  \* @param prodPage  \* @param t1  \* @param t2  \* @return  \*/  @Override  public IPage<Prod> findProdByPageToEs(IPage<Prod> prodPage, Date t1, Date t2) {  //得到数据  IPage<Prod> prodIPage = prodMapper.selectPage(prodPage, new LambdaQueryWrapper<Prod>()  .between(t1 != null && t2 != null, Prod::getUpdateTime, t1, t2)  );  //组装数据 还需要标签 还需要好评率 （好评数/总评论 \*100% 取2位小数）  List<Prod> prods = prodIPage.getRecords();  List<Long> prodIds = prods.stream().map(Prod::getProdId).collect(Collectors.toList());  //查询标签  //根据id来查询标签分组  List<ProdTagReference> prodTagReferences = prodTagReferenceService.list(new LambdaQueryWrapper<ProdTagReference>()  .in(ProdTagReference::getProdId, prodIds)  );  prods.forEach(prod -> {  //查到商品对应的标签分组  List<ProdTagReference> tagReferences = prodTagReferences.stream()  .filter(prodTagReference -> prodTagReference.getProdId().equals(prod.getProdId()))  .collect(Collectors.toList());  //拿到商品对应的标签分组  List<Long> tagList = tagReferences.stream()  .map(ProdTagReference::getTagId)  .collect(Collectors.toList());  //设置分组  prod.setTagList(tagList);  });  //查询评论  List<ProdComm> prodCommList = prodCommService.list(new LambdaQueryWrapper<ProdComm>()  .in(ProdComm::getProdId, prodIds)  );  //设置好评率 （好评数/总评论 \*100% 取2位小数）  prods.forEach(prod -> {  List<ProdComm> prodComms = prodCommList.stream()  .filter(prodComm -> prodComm.getProdId().equals(prod.getProdId()))  .collect(Collectors.toList());  int size = prodComms.size();  if (size <= 0) {  //没有评论数 哪来的好评率  prod.setPraiseNumber(0L);  prod.setPositiveRating(new BigDecimal(0));  } else {  //说明有评论  prod.setPraiseNumber(Long.valueOf(size));  //算好评率  List<ProdComm> goodsProdComm = prodComms.stream()  .filter(prodComm -> prodComm.getEvaluate().equals(0))  .collect(Collectors.toList());  int goodsSize = goodsProdComm.size();  if (goodsSize <= 0) {  //没有好评  prod.setPositiveRating(new BigDecimal(0));  } else {  //有好评率  Double a = (double) goodsSize;  Double b = (double) size;  double c = a / b;  prod.setPositiveRating(  new BigDecimal(c)  .multiply(new BigDecimal(100))  .setScale(2, BigDecimal.ROUND\_HALF\_UP)  );  }  }  });  return prodIPage;  }  } |

#### 新增ImportService(商品导入)

1. 三种方式

全量导入 （业务场景：项目一启动就要导入，商品有500w条 代码怎么实现？ 自己分页 自定义导入批量大小 300-500条 ）

增量导入 （业务场景：新增商品 代码实现： 有一个时间窗口 30分钟 updateTime 区间的 就导入 16.05 --- 16.45 ）

快速导入 （立马修改es的数据 下单 买商品 减库存的时候吗 怎么实现 （MQ））

|  |
| --- |
| public interface ImportService {  /\*\*  \* 全量导入  \* 项目一启动就把数据库里面的商品导入到es里面  \*/  void importAll();  /\*\*  \* 增量导入  \* 当管理员新增一个商品，我们隔半个小时后把商品导入  \*/  void importUpdate();  /\*\*  \* 快速导入  \* 商品的信息，价格，库存等变动，需要快速导入更新  \*/  void quickImport();  } |

#### 在ego-common中增加QueueConstant

|  |
| --- |
| public class QueueConstant {  //商品修改的队列  public static final String PROD\_CHANGE\_QUEUE = "prod.change.queue";  //商品修改队列的交换机  public static final String PROD\_CHANGE\_EXCHANGE = "prodChangeEx";  //商品修改的路由key  public static final String PROD\_ROUTER\_KEY = "prodRouterKey";  } |

#### 新增ImportServiceImpl（重点）

|  |
| --- |
| @Service  @Slf4j  public class ImportServiceImpl implements ImportService, CommandLineRunner {  @Autowired  private ProductDao productDao;  @Autowired  private ProdService prodService;  /\*\*  \* 每次导入多少条  \*/  @Value("${esimport.size}")  private Integer size;  private Date t1;  /\*\*  \* 全部导入 用于项目一起动  \*/  @Override  public void importAll() {  //1.查询数量 计算分页 t1 t2  Integer count = prodService.getTotal(null, null);  if (count <= 0) {  log.info("商品数量为0 不导入");  return;  }  //计算总页数  int totalPage = count % size == 0 ? count / size : (count / size) + 1;  for (int i = 1; i <= totalPage; i++) {  //分页查询数据库 导入es  log.info("分页查询数据库i=" + i);  importToEs(i, size, null, null);  }  log.info("导入完成");  //给一个时间戳  t1 = new Date();  }  /\*\*  \* 分页查询商品  \*  \* @param current  \* @param size  \* @param t1  \* @param t2  \*/  private void importToEs(int current, Integer size, Date t1, Date t2) {  //组装分页对象 优化的地方 不让他自己再查一次count了  IPage<Prod> prodPage = new Page<>(current, size, false);  //查询分页数据  IPage<Prod> prodIPage = prodService.findProdByPageToEs(prodPage, t1, t2);  List<Prod> prods = prodIPage.getRecords();  if (CollectionUtils.isEmpty(prods)) {  return;  }  //组装成es对象 导入  ArrayList<ProductEs> productEs = new ArrayList<>(prods.size() \* 2);  prods.forEach(prod -> {  ProductEs productEs1 = new ProductEs();  productEs1.setProdId(prod.getProdId());  productEs1.setProdName(prod.getProdName());  productEs1.setBrief(prod.getBrief());  productEs1.setCategoryId(prod.getCategoryId());  productEs1.setPic(prod.getPic());  productEs1.setTagList(prod.getTagList());  productEs1.setPrice(prod.getPrice());  productEs1.setSoldNum(Long.valueOf(prod.getSoldNum()));  productEs1.setStatus(prod.getStatus());  productEs1.setTotalStocks(Long.valueOf(prod.getTotalStocks()));  productEs1.setPraiseNumber(prod.getPraiseNumber());  productEs1.setPositiveRating(prod.getPositiveRating());  productEs.add(productEs1);  });  productDao.saveAll(productEs);  }  /\*\*  \* 增量导入 用于上新品，隔3分钟导入es  \*/  @Override  @Scheduled(initialDelay = 120 \* 1000, fixedRate = 120 \* 1000)  public void importUpdate() {  Date t2 = new Date();  System.out.println(t1 + "----" + t2);  Integer count = prodService.getTotal(t1, t2);  if (count <= 0) {  log.info("新增或者修改商品数量为0 不导入");  return;  }  //计算总页数  int totalPage = count % size == 0 ? count / size : (count / size) + 1;  for (int i = 1; i <= totalPage; i++) {  //分页查询数据库 导入es  log.info("分页查询数据库i=" + i);  importToEs(i, size, t1, t2);  }  log.info("导入完成");  //时间往后推移  t1 = t2;  }  /\*\*  \* 及时导入 用于用户下单修改了商品信息  \* 放进mq里面  \* 我们监听mq 的一个队列 从而修改es数据  \*/  @Override  public void quickImport() {  }  /\*\*  \* 及时处理es map<Long,Integer>  \* 1,-2  \* 2,-3  \* 3,-3  \*  \* @param message  \* @param channel  \* @param tag  \*/  @RabbitListener(queues = QueueConstant.PROD\_CHANGE\_QUEUE, concurrency = "3-5")  public void handlerProdChange(Message message, Channel channel, @Header(AmqpHeaders.DELIVERY\_TAG) Long tag) {  //拿到消息对象  JSONObject jsonObject = JSON.parseObject(new String(message.getBody()));  ArrayList<ProductEs> productEsArrayList = new ArrayList<>();  //map怎么遍历  jsonObject.forEach((prodId, stock) -> {  //从es得到这一条数据  ProductEs productEs = productDao.findById(Long.valueOf(prodId)).get();  if (ObjectUtils.isEmpty(productEs)) {  throw new IllegalArgumentException("es中没有这个id的商品,id为" + prodId);  }  Long stockL = Long.valueOf(stock.toString());  long realStock = productEs.getTotalStocks() + stockL;  if (realStock < 0) {  throw new IllegalArgumentException("商品数量异常");  }  productEs.setTotalStocks(realStock);  productEsArrayList.add(productEs);  });  //不要在循环里面操作数据库  try {  productDao.saveAll(productEsArrayList);  //签收消息 false 代表不签收 <=tag 的消息  channel.basicAck(tag, false);  } catch (Exception e) {  log.error("更新es失败");  }  }  /\*\*  \* 这个方法可以保证所有的bean都初始化完成后再执行  \*  \* @param args  \* @throws Exception  \*/  @Override  public void run(String... args) throws Exception {  log.info("全量导入开始");  System.out.println(productDao);  if (productDao == null) {  throw new RuntimeException("productDao是空，请查看");  }  importAll();  }  } |