# EGO门店服务

## EGO门店服务

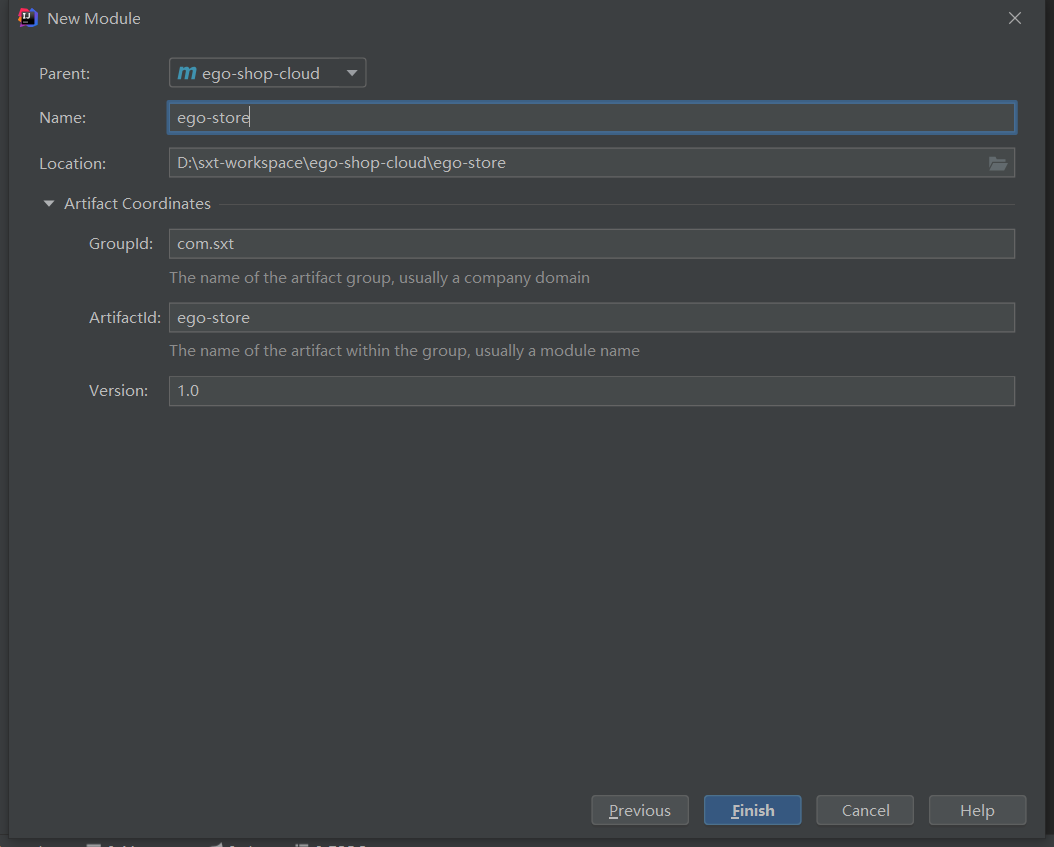
功能定位：门店位置，店铺热搜，主页轮播图，公告，配送地址，快递信息等

集成模块：搜索服务

## EGO门店服务

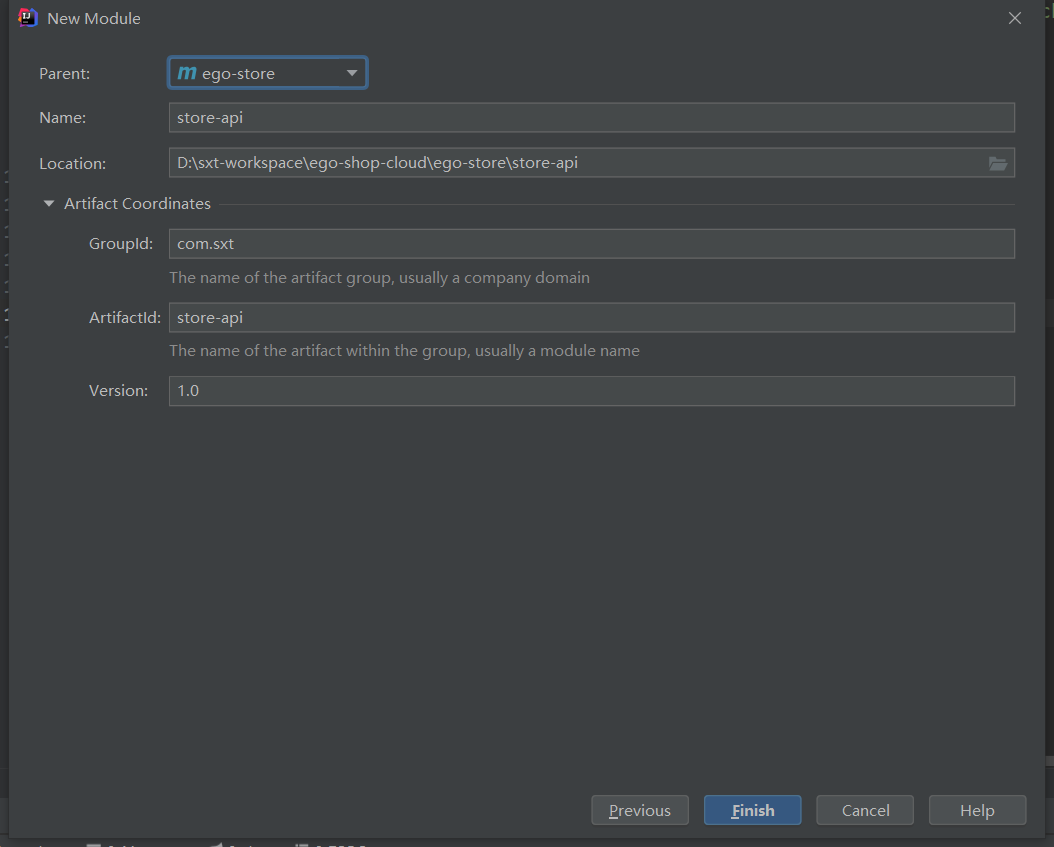
### 新建ego-store父模块

#### 在ego-shop-cloud下面新建一个maven模块



### 在ego-store下面新建一个store-api模块

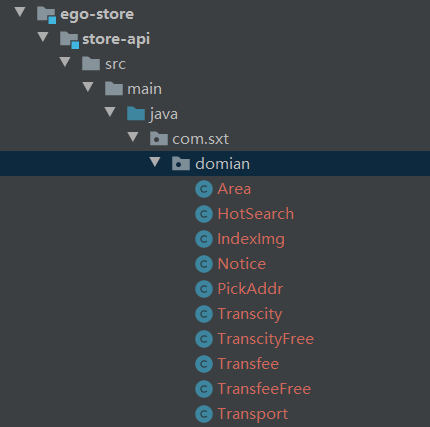
#### 新建项目



#### 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-store</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>store-api</artifactId>  <description>ego商城门店服务api</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>product-api</artifactId>  <version>1.0</version>  <exclusions>  <exclusion>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-elasticsearch</artifactId>  </exclusion>  </exclusions>  </dependency>  <!-- 需要web和openFeign远程调用-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-openfeign</artifactId>  </dependency>  </dependencies>  </project> |

#### 逆向生成这几张表



#### 注意**修改**时间类型和Byte类型

#### 注意**修改IndexImg实体类**

|  |
| --- |
| @TableField(exist = false)  @ApiModelProperty("轮播图关联的商品图片")  private String pic;  @TableField(exist = false)  @ApiModelProperty("轮播图关联的商品的名称")  private String prodName; |

#### 注意**修改Transfee实体类**

|  |
| --- |
| @TableField(exist = false)  @ApiModelProperty("收费的城市的中间表")  private List<Area> cityList= Collections.emptyList(); |

#### 注意**修改TransfeeFree实体类**

|  |
| --- |
| @TableField(exist = false)  @ApiModelProperty("免费的城市的中间表")  private List<Area> freeCityList = Collections.emptyList(); |

#### 注意**修改Transport实体类**

|  |
| --- |
| @TableField(exist = false)  @ApiModelProperty("所有的收费模板")  private List<Transfee> transfees= Collections.emptyList();  @TableField(exist = false)  @ApiModelProperty("所有免费的模板")  private List<TransfeeFree> transfeeFrees= Collections.emptyList(); |

#### 创建城市区域的常量类

|  |
| --- |
| public class AreaConstant {  //所有城市的key  public static final String ALL\_AREA = "'all\_area'";  } |

#### 创建轮播图的常量类

|  |
| --- |
| public class IndexImgConstant {  public static final String INDEX\_ALL\_IMGS = "'index\_all\_img'";  } |

#### 创建轮播图的远程调用

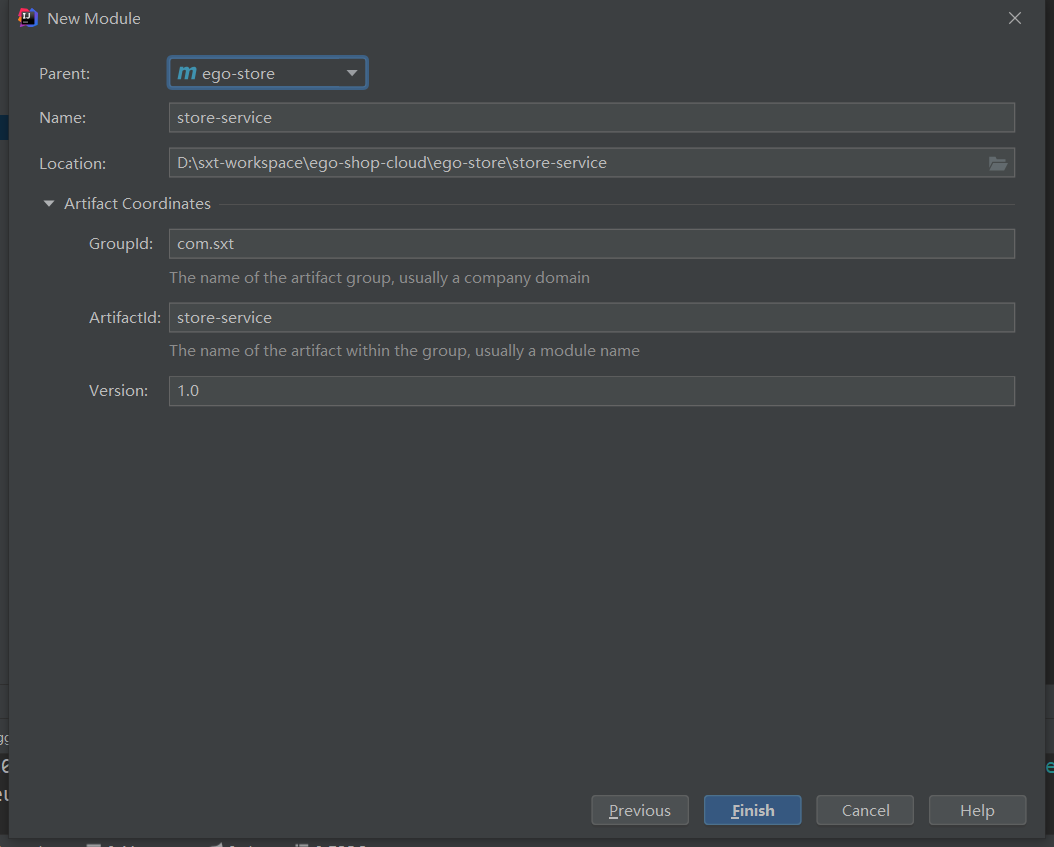
|  |
| --- |
| @FeignClient(value = "product-service", fallback = IndexImgFeignHystrix.class)  public interface IndexImgFeign {  /\*\*  \* 查询单个商品  \*  \* @param id  \* @return  \*/  @GetMapping("/prod/prod/info/{id}")  Prod getProdById(@PathVariable("id") Long id);  } |

#### 创建轮播图的远程调用的熔断

|  |
| --- |
| @Component  @Slf4j  public class IndexImgFeignHystrix implements IndexImgFeign {  /\*\*  \* 查询单个商品  \*  \* @param id  \* @return  \*/  @Override  public Prod getProdById(Long id) {  log.info("远程调用查询单个商品失败");  return null;  }  } |

### 在ego-store下面新建一个store-service模块

#### 新建项目



#### 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-store</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>store-service</artifactId>  <description>ego商城门店服务service</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>store-api</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>ego-common</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>  <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  @EnableCaching  @EnableFeignClients  public class StoreServiceApplication {  public static void main(String[] args) {  SpringApplication.run(StoreServiceApplication.class, args);  }  } |

#### 添加bootstrap.yml配置文件

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

#### 添加远端store-service/store-service-dev.yml文件

|  |
| --- |
| server:  port: ${APP\_PORT:8084}  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: '\*'  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  mybatis-plus:  mapper-locations: classpath:mapper/\*\*/\*.xml  configuration:  log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  swagger2:  base-package: com.sxt.controller  name: sxt  url: https://gitee.com/smiledouble  email: 775610843@qq.com  title: 商家店铺管理系统接口  description: 商家店铺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RedisCacheConfig

|  |
| --- |
| @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AreaController

|  |
| --- |
| @RestController  @RequestMapping("/admin/area")  @Api(tags = "区域管理")  public class AreaController {  @Autowired  private AreaService areaService;  @GetMapping("/list")  @PreAuthorize("hasAuthority('admin:area:list')")  @ApiOperation("查询地址的列表")  public ResponseEntity<List<Area>> list() {  List<Area> list = areaService.list();  return ResponseEntity.ok(list);  }  @GetMapping("/listByPid")  @PreAuthorize("hasAuthority('admin:area:list')")  @ApiOperation("通过父区域id查询子区域")  public ResponseEntity<List<Area>> listByPid(@RequestParam(name = "pid", defaultValue = "0") Long pId) {  List<Area> areas = areaService.listByPid(pId);  return ResponseEntity.ok(areas);  }  @PostMapping  @PreAuthorize("hasAuthority('admin:area:save')")  @ApiOperation("新增一个Area")  public ResponseEntity<Void> save(@RequestBody @Validated Area entity) {  areaService.save(entity);  return ResponseEntity.ok().build();  }  @PutMapping  @PreAuthorize("hasAuthority('admin:area:update')")  @ApiOperation("根据id修改Area")  public ResponseEntity<Void> update(@RequestBody @Validated Area entity) {  areaService.updateById(entity);  return ResponseEntity.ok().build();  }  @DeleteMapping("/{id}")  @PreAuthorize("hasAuthority('admin:area:delete')")  @ApiOperation("删除Area")  public ResponseEntity<Void> delete(@PathVariable("id") Long id) {  areaService.removeById(id);  return ResponseEntity.ok().build();  }  @GetMapping("/info/{id}")  @PreAuthorize("hasAuthority('admin:area:info')")  @ApiOperation("查询Area详情")  public ResponseEntity<Area> info(@PathVariable("id") Long id) {  Area entity = areaService.getById(id);  return ResponseEntity.ok(entity);  }  } |

#### 修改AreaService

|  |
| --- |
| public interface AreaService extends IService<Area> {  /\*\*  \* 通过父id查询子  \*  \* @param pId  \* @return  \*/  List<Area> listByPid(Long pId);  } |

#### 修改AreaServiceImpl

|  |
| --- |
| @Service  @CacheConfig(cacheNames = "com.sxt.service.impl.AreaServiceImpl")  @Slf4j  public class AreaServiceImpl extends ServiceImpl<AreaMapper, Area> implements AreaService {  @Autowired  private AreaMapper areaMapper;  /\*\*  \* 全查询区域信息  \*  \* @return  \*/  @Override  @Cacheable(key = AreaConstant.ALL\_AREA)  public List<Area> list() {  return super.list();  }  /\*\*  \* 通过父id查询子  \*  \* @param pId  \* @return  \*/  @Override  public List<Area> listByPid(Long pId) {  List<Area> areas = areaMapper.selectList(new LambdaQueryWrapper<Area>()  .eq(Area::getParentId, pId)  );  if (ObjectUtils.isEmpty(areas)) {  return Collections.emptyList();  }  return areas;  }  /\*\*  \* 新增一个地区 清理缓存  \*  \* @param entity  \* @return  \*/  @Override  @CacheEvict(key = AreaConstant.ALL\_AREA)  public boolean save(Area entity) {  log.info("新增一个区域{}", JSONUtil.toJsonStr(entity));  if (entity.getParentId() == null || entity.getParentId().equals(0)) {  entity.setLevel(1);  } else {  //查询到父id 设置等级+1  Area parent = super.getById(entity.getParentId());  entity.setLevel(parent.getLevel() + 1);  }  return super.save(entity);  }  /\*\*  \* 删除一个区域  \*  \* @param id  \* @return  \*/  @Override  @Caching(  evict = {  @CacheEvict(key = "#id"),  @CacheEvict(key = AreaConstant.ALL\_AREA)  }  )  public boolean removeById(Serializable id) {  log.info("删除一个区域id为{}", id);  //先判断他下面有没有子节点  Integer count = areaMapper.selectCount(new LambdaQueryWrapper<Area>()  .eq(Area::getParentId, id)  );  if (count != null && count > 0) {  throw new IllegalArgumentException("删除的区域有子节点，不能删除,要删除的id为" + id);  }  return super.removeById(id);  }  /\*\*  \* 更新一个区域  \*  \* @param entity  \* @return  \*/  @Override  @Caching(  evict = {  @CacheEvict(key = "entity.areaId"),  @CacheEvict(key = AreaConstant.ALL\_AREA)  }  )  public boolean updateById(Area entity) {  return super.updateById(entity);  }  /\*\*  \* 查询放入缓存中  \*  \* @param id  \* @return  \*/  @Override  @Cacheable(key = "#id")  public Area getById(Serializable id) {  return super.getById(id);  }  } |

#### 添加HotSearchController

|  |
| --- |
| @RestController  @RequestMapping("admin/hotSearch")  public class HotSearchController {  @Autowired  private HotSearchService hotSearchService;  @GetMapping("/page")  @PreAuthorize("hasAuthority('admin:hotSearch:page')")  @ApiOperation("热搜的分页查询")  public ResponseEntity<IPage<HotSearch>> findByPage(Page<HotSearch> page, HotSearch hotSearch) {  IPage<HotSearch> pageData = hotSearchService.findByPage(page, hotSearch);  return ResponseEntity.ok(pageData);  }  @GetMapping("/info/{id}")  @PreAuthorize("hasAuthority('admin:hotSearch:info') ")  @ApiOperation("热搜的回显")  public ResponseEntity<HotSearch> findById(@PathVariable("id") Long id) {  HotSearch byId = hotSearchService.getById(id);  return ResponseEntity.ok(byId);  }  @DeleteMapping("/{id}")  @PreAuthorize("hasAuthority('admin:hotSearch:delete')")  @ApiOperation("删除热搜")  public ResponseEntity<Void> delete(@PathVariable("id") Long id) {  hotSearchService.removeById(id);  return ResponseEntity.ok().build();  }  @PostMapping  @PreAuthorize("hasAuthority('admin:hotSearch:save')")  @ApiOperation("新增一个热搜")  public ResponseEntity<Void> add(@RequestBody @Validated HotSearch hotSearch) {  hotSearch.setRecDate(new Date());  hotSearchService.save(hotSearch);  return ResponseEntity.ok().build();  }  @PutMapping  @PreAuthorize("hasAuthority('admin:hotSearch:update')")  @ApiOperation("新增一个热搜")  public ResponseEntity<Void> update(@RequestBody @Validated HotSearch hotSearch) {  hotSearchService.updateById(hotSearch);  return ResponseEntity.ok().build();  }  } |

#### 修改HotSearchService

|  |
| --- |
| public interface HotSearchService extends IService<HotSearch> {  /\*\*  \* 分页查询热搜  \*  \* @param page  \* @param hotSearch  \* @return  \*/  IPage<HotSearch> findByPage(Page<HotSearch> page, HotSearch hotSearch);  } |

#### 修改HotSearchServiceImpl

|  |
| --- |
| @Service  @CacheConfig(cacheNames = "com.sxt.service.impl.HotSearchServiceImpl")  public class HotSearchServiceImpl extends ServiceImpl<HotSearchMapper, HotSearch> implements HotSearchService {  @Autowired  private HotSearchMapper hotSearchMapper;  /\*\*  \* 分页查询热搜  \*  \* @param page  \* @param hotSearch  \* @return  \*/  @Override  public IPage<HotSearch> findByPage(Page<HotSearch> page, HotSearch hotSearch) {  page.addOrder(OrderItem.desc("rec\_date"));  return hotSearchMapper.selectPage(page, new LambdaQueryWrapper<HotSearch>().  like(StringUtils.hasText(hotSearch.getTitle()), HotSearch::getTitle, hotSearch.getTitle()).  like(StringUtils.hasText(hotSearch.getContent()), HotSearch::getContent, hotSearch.getContent()).  eq(hotSearch.getStatus() != null, HotSearch::getStatus, hotSearch.getStatus())  );  }  @Cacheable(key = "#id")  @Override  public HotSearch getById(Serializable id) {  return super.getById(id);  }  @CacheEvict(key = "#entity.hotSearchId")  @Override  public boolean updateById(HotSearch entity) {  return super.updateById(entity);  }  @CacheEvict(key = "#id")  @Override  public boolean removeById(Serializable id) {  return super.removeById(id);  }  } |

#### 添加IndexImgController

|  |
| --- |
| @RestController  @RequestMapping("/admin/indexImg")  @Api(tags = "首页轮播图管理")  public class IndexImgController {  @Autowired  private IndexImgService indexImgService;  @GetMapping("/page")  @PreAuthorize("hasAuthority('admin:indexImg:page')")  @ApiOperation("分页查询IndexImg")  public ResponseEntity<IPage<IndexImg>> page(Page<IndexImg> page, IndexImg condition) {  IPage<IndexImg> pageDate = indexImgService.findByPage(page, condition);  return ResponseEntity.ok(pageDate);  }  @PostMapping  @PreAuthorize("hasAuthority('admin:indexImg:save')")  @ApiOperation("新增一个IndexImg")  public ResponseEntity<Void> save(@RequestBody @Validated IndexImg entity) {  indexImgService.save(entity);  return ResponseEntity.ok().build();  }  @PutMapping  @PreAuthorize("hasAuthority('admin:indexImg:update')")  @ApiOperation("根据id修改IndexImg")  public ResponseEntity<Void> update(@RequestBody @Validated IndexImg entity) {  indexImgService.updateById(entity);  return ResponseEntity.ok().build();  }  @DeleteMapping("/{id}")  @PreAuthorize("hasAuthority('admin:indexImg:delete')")  @ApiOperation("删除IndexImg")  public ResponseEntity<Void> delete(@PathVariable("id") Long id) {  indexImgService.removeById(id);  return ResponseEntity.ok().build();  }  /\*\*  \* 轮播图的加载  \*/  @GetMapping("/indexImgs")  public ResponseEntity<List<IndexImgVo>> loadIndexImgs() {  List<IndexImgVo> indexImgList = indexImgService.loadIndexImgs();  if (indexImgList == null || indexImgList.isEmpty()) {  return ResponseEntity.ok(Collections.EMPTY\_LIST);  }  return ResponseEntity.ok(indexImgList);  }  /\*\*  \* 查询轮播图的具体信息  \*  \* @param id  \* @return  \*/  @GetMapping("/info/{id}")  @PreAuthorize("hasAuthority('admin:indexImg:info')")  @ApiOperation("查询IndexImg详情")  public ResponseEntity<IndexImg> info(@PathVariable("id") Long id) {  IndexImg entity = indexImgService.getInfo(id);  return ResponseEntity.ok(entity);  }  } |

#### 修改IndexImgService

|  |
| --- |
| public interface IndexImgService extends IService<IndexImg> {  /\*\*  \* 分页查询轮播图  \*  \* @param page  \* @param condition  \* @return  \*/  IPage<IndexImg> findByPage(Page<IndexImg> page, IndexImg condition);  /\*\*  \* 轮播图的加载  \*  \* @return  \*/  List<IndexImgVo> loadIndexImgs();  /\*\*  \* 轮播图和商品的对应信息  \*  \* @param id  \* @return  \*/  IndexImg getInfo(Long id);  } |

#### 修改IndexImgServiceImpl

|  |
| --- |
| @Service  @Slf4j  @CacheConfig(cacheNames = "com.sxt.service.impl.IndexImgServiceImpl")  public class IndexImgServiceImpl extends ServiceImpl<IndexImgMapper, IndexImg> implements IndexImgService {  @Autowired  private IndexImgMapper indexImgMapper;  @Autowired  private IndexImgFeign indexImgFeign;  /\*\*  \* 分页查询轮播图  \*  \* @param page  \* @param condition  \* @return  \*/  @Override  public IPage<IndexImg> findByPage(Page<IndexImg> page, IndexImg condition) {  page.addOrder(OrderItem.desc("upload\_time"));  IPage<IndexImg> indexImgIPage = indexImgMapper.selectPage(page, new LambdaQueryWrapper<IndexImg>()  .eq(condition.getStatus() != null, IndexImg::getStatus, condition.getStatus())  );  return indexImgIPage;  }  @CacheEvict(key = IndexImgConstant.INDEX\_ALL\_IMGS)  @Override  public boolean save(IndexImg entity) {  entity.setUploadTime(new Date());  return super.save(entity);  }  @Cacheable(key = "#id")  @Override  public IndexImg getById(Serializable id) {  return super.getById(id);  }  @Caching(evict = {@CacheEvict(key = "#id"), @CacheEvict(key = IndexImgConstant.INDEX\_ALL\_IMGS)})  @Override  public boolean removeById(Serializable id) {  return super.removeById(id);  }  @Caching(evict = {@CacheEvict(key = "#entity.imgId"), @CacheEvict(key = IndexImgConstant.INDEX\_ALL\_IMGS)})  @Override  public boolean updateById(IndexImg entity) {  return super.updateById(entity);  }  /\*\*  \* 轮播图的加载  \*  \* @return  \*/  @Override  @Cacheable(key = IndexImgConstant.INDEX\_ALL\_IMGS)  public List<IndexImgVo> loadIndexImgs() {  List<IndexImg> indexImgs = indexImgMapper.selectList(new LambdaQueryWrapper<IndexImg>()  .eq(IndexImg::getStatus, 1)  .orderByAsc(IndexImg::getSeq) //前端根据这个排序  );  //转换对象  List<IndexImgVo> indexImgVos = new ArrayList<>();  if (indexImgs != null && indexImgs.size() > 0) {  indexImgs.forEach(indexImg -> {  IndexImgVo indexImgVo = new IndexImgVo();  indexImgVo.setImgUrl(indexImg.getImgUrl());  indexImgVo.setRelation(indexImg.getRelation());  indexImgVos.add(indexImgVo);  });  }  return indexImgVos;  }  /\*\*  \* 轮播图和商品的对应信息  \*  \* @param id  \* @return  \*/  @Override  public IndexImg getInfo(Long id) {  //根据轮播图id查询商品  IndexImg indexImg = indexImgMapper.selectById(id);  if (ObjectUtils.isEmpty(indexImg)) {  throw new IllegalArgumentException("查询的轮播图id不存在");  }  if (indexImg.getRelation() == null) {  return indexImg;  }  //远程调用商品的服务 查询商品  Prod prod = indexImgFeign.getProdById(indexImg.getRelation());  indexImg.setProdName(prod.getProdName());  indexImg.setPic(prod.getPic());  return indexImg;  }  } |

#### 添加NoticeController

|  |
| --- |
| @RestController  @RequestMapping("/shop/notice")  public class NoticeController {  @Autowired  private NoticeService noticeService;  @GetMapping({"/page"})  @PreAuthorize("hasAuthority('shop:notice:page')")  @ApiOperation("分页查询公告")  public ResponseEntity<IPage<Notice>> findByPage(Page<Notice> page, Notice notice) {  IPage<Notice> pageData = this.noticeService.findByPage(page, notice);  return ResponseEntity.ok(pageData);  }  @GetMapping({"/info/{id}"})  @PreAuthorize("hasAuthority('shop:notice:info')")  @ApiOperation("公告的回显")  public ResponseEntity<Notice> findById(@PathVariable("id") Long id) {  Notice notice = noticeService.getById(id);  return ResponseEntity.ok(notice);  }  @DeleteMapping({"/{id}"})  @PreAuthorize("hasAuthority('shop:notice:delete')")  @ApiOperation("删除公告")  public ResponseEntity<Void> delete(@PathVariable("id") Long id) {  this.noticeService.removeById(id);  return ResponseEntity.ok().build();  }  @PutMapping  @PreAuthorize("hasAuthority('shop:notice:update')")  @ApiOperation("修改公告")  public ResponseEntity<Void> update(@RequestBody @Validated Notice notice) {  this.noticeService.updateById(notice);  return ResponseEntity.ok().build();  }  @PostMapping  @PreAuthorize("hasAuthority('shop:notice:save')")  @ApiOperation("新增公告")  public ResponseEntity<Void> save(@RequestBody @Validated Notice notice) {  this.noticeService.save(notice);  return ResponseEntity.ok().build();  }  } |

#### 修改NoticeService

|  |
| --- |
| public interface NoticeService extends IService<Notice> {  /\*\*  \* 分页查询公告  \*  \* @param page  \* @param notice  \* @return  \*/  IPage<Notice> findByPage(Page<Notice> page, Notice notice);  } |

#### 修改NoticeServiceImpl

|  |
| --- |
| @Service  @Slf4j  @CacheConfig(cacheNames = "com.sxt.service.impl.NoticeServiceImpl")  public class NoticeServiceImpl extends ServiceImpl<NoticeMapper, Notice> implements NoticeService {  @Autowired  private NoticeMapper noticeMapper;  /\*\*  \* 分页查询公告  \*  \* @param page  \* @param notice  \* @return  \*/  @Override  public IPage<Notice> findByPage(Page<Notice> page, Notice notice) {  page.addOrder(OrderItem.desc("update\_time"));  IPage<Notice> noticeIPage = noticeMapper.selectPage(page, new LambdaQueryWrapper<Notice>()  .like(StringUtils.hasText(notice.getTitle()), Notice::getTitle, notice.getTitle())  .eq(notice.getStatus() != null, Notice::getStatus, notice.getStatus())  .eq(notice.getIsTop() != null, Notice::getIsTop, notice.getIsTop())  );  return noticeIPage;  }  @Override  @Cacheable(key = "#id")  public Notice getById(Serializable id) {  return super.getById(id);  }  /\*\*  \* 更新操作  \* 不能使用@CachePut注解来操作更新，因为执行完以后会缓存一个boolean  \* 可以使用@CacheEvict,使用beforeInvocation来前置执行  \*  \* @param entity  \* @return  \*/  @Override  @CacheEvict(key = "#entity.id", beforeInvocation = true)  public boolean updateById(Notice entity) {  log.info("更新的公告为", JSONUtil.toJsonStr(entity));  entity.setUpdateTime(new Date());  return super.updateById(entity);  }  /\*\*  \* 删除  \*  \* @param id  \* @return  \*/  @Override  @CacheEvict(key = "#id")  public boolean removeById(Serializable id) {  log.info("删除一个公告,id为{}", id);  return super.removeById(id);  }  } |

#### 添加PickAddrController

|  |
| --- |
| @RestController  @RequestMapping("/shop/pickAddr")  public class PickAddrController {  @Autowired  private PickAddrService pickAddrService;  @GetMapping("/page")  @PreAuthorize("hasAuthority('shop:pickAddr:page')")  @ApiOperation("分页查询PickAddr")  public ResponseEntity<IPage<PickAddr>> page(Page<PickAddr> page, PickAddr condition) {  IPage<PickAddr> pageDate = pickAddrService.findByPage(page, condition);  return ResponseEntity.ok(pageDate);  }  @PostMapping  @PreAuthorize("hasAuthority('shop:pickAddr:save')")  @ApiOperation("新增一个PickAddr")  public ResponseEntity<Void> save(@RequestBody @Validated PickAddr entity) {  pickAddrService.save(entity);  return ResponseEntity.ok().build();  }  @PutMapping  @PreAuthorize("hasAuthority('shop:pickAddr:update')")  @ApiOperation("根据id修改PickAddr")  public ResponseEntity<Void> update(@RequestBody @Validated PickAddr entity) {  pickAddrService.updateById(entity);  return ResponseEntity.ok().build();  }  @DeleteMapping("/{id}")  @PreAuthorize("hasAuthority('shop:pickAddr:delete')")  @ApiOperation("删除PickAddr")  public ResponseEntity<Void> delete(@PathVariable("id") Long id) {  pickAddrService.removeById(id);  return ResponseEntity.ok().build();  }  @GetMapping("/info/{id}")  @PreAuthorize("hasAuthority('shop:pickAddr:info')")  @ApiOperation("查询PickAddr详情")  public ResponseEntity<PickAddr> info(@PathVariable("id") Long id) {  PickAddr entity = pickAddrService.getById(id);  return ResponseEntity.ok(entity);  }  } |

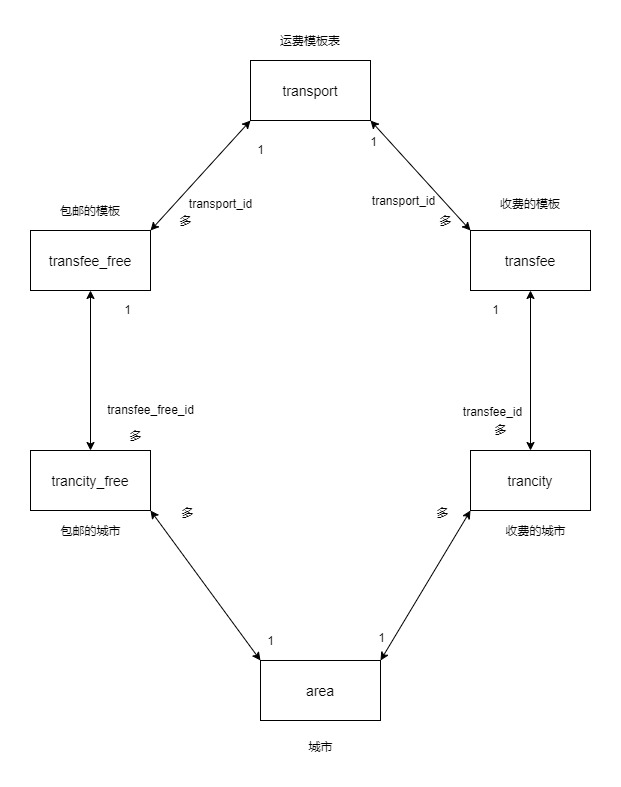
#### 修改PickAddeService

|  |
| --- |
| public interface PickAddrService extends IService<PickAddr> {  /\*\*  \* 分页查询自提点  \*  \* @param page  \* @param condition  \* @return  \*/  IPage<PickAddr> findByPage(Page<PickAddr> page, PickAddr condition);  } |

#### 修改PickAddeServiceImpl

|  |
| --- |
| @Service  @Slf4j  @CacheConfig(cacheNames = "com.sxt.service.impl.PickAddrServiceImpl")  public class PickAddrServiceImpl extends ServiceImpl<PickAddrMapper, PickAddr> implements PickAddrService {  @Autowired  private PickAddrMapper pickAddrMapper;  /\*\*  \* 分页查询自提点  \*  \* @param page  \* @param condition  \* @return  \*/  @Override  public IPage<PickAddr> findByPage(Page<PickAddr> page, PickAddr condition) {  return pickAddrMapper.selectPage(page, new LambdaQueryWrapper<PickAddr>().  like(StringUtils.hasText(condition.getAddrName()), PickAddr::getAddrName, condition.getAddrName()));  }  @Cacheable(key = "#id")  @Override  public PickAddr getById(Serializable id) {  return super.getById(id);  }  @CacheEvict(key = "#id")  @Override  public boolean removeById(Serializable id) {  return super.removeById(id);  }  @CacheEvict(key = "#entity.addrId")  @Override  public boolean updateById(PickAddr entity) {  return super.updateById(entity);  }  } |

#### 快递运费关系图



#### 添加TransportController

|  |
| --- |
| @RestController  @RequestMapping("/shop/transport")  public class TransportController {  @Autowired  private TransportService transportService;  @GetMapping("/page")  @PreAuthorize("hasAuthority('shop:transport:page') ")  @ApiOperation("分页查询Transport")  public ResponseEntity<IPage<Transport>> page(Page<Transport> page, Transport condition) {  IPage<Transport> pageDate = transportService.findByPage(page, condition);  return ResponseEntity.ok(pageDate);  }  @PostMapping  @PreAuthorize("hasAuthority('shop:transport:save') ")  @ApiOperation("新增一个Transport")  public ResponseEntity<Void> save(@RequestBody @Validated Transport entity) {  transportService.save(entity);  return ResponseEntity.ok().build();  }  @PutMapping  @PreAuthorize("hasAuthority('shop:transport:update') ")  @ApiOperation("根据id修改Transport")  public ResponseEntity<Void> update(@RequestBody @Validated Transport entity) {  transportService.updateById(entity);  return ResponseEntity.ok().build();  }  @DeleteMapping("/{id}")  @PreAuthorize("hasAuthority('shop:transport:delete') ")  @ApiOperation("删除Transport")  public ResponseEntity<Void> delete(@PathVariable("id") Long id) {  transportService.removeById(id);  return ResponseEntity.ok().build();  }  @GetMapping("/info/{id}")  @PreAuthorize("hasAuthority('shop:transport:info') ")  @ApiOperation("查询Transport详情")  public ResponseEntity<Transport> info(@PathVariable("id") Long id) {  Transport entity = transportService.getById(id);  return ResponseEntity.ok(entity);  }  } |

#### 修改TransportService

|  |
| --- |
| public interface TransportService extends IService<Transport> {  /\*\*  \* 分页查询运费模板  \*  \* @param page  \* @param condition  \* @return  \*/  IPage<Transport> findByPage(Page<Transport> page, Transport condition);  } |

#### 修改TransportServiceImpl

|  |
| --- |
| @Service  @Slf4j  @CacheConfig(cacheNames = "com.sxt.service.impl.TransportServiceImpl")  public class TransportServiceImpl extends ServiceImpl<TransportMapper, Transport> implements TransportService {  @Autowired  private TransportMapper transportMapper;  @Autowired  private TransfeeMapper transfeeMapper;  @Autowired  private TransfeeFreeMapper transfeeFreeMapper;  @Autowired  private TranscityMapper transcityMapper;  @Autowired  private TranscityFreeMapper transcityFreeMapper;  @Autowired  private AreaMapper areaMapper;  /\*\*  \* 分页查询运费模板  \*  \* @param page  \* @param condition  \* @return  \*/  @Override  public IPage<Transport> findByPage(Page<Transport> page, Transport condition) {  page.addOrder(OrderItem.desc("create\_time"));  return transportMapper.selectPage(page, new LambdaQueryWrapper<Transport>().  like(StringUtils.hasText(condition.getTransName()), Transport::getTransName, condition.getTransName()));  }  @Override  @Cacheable(key = "#id")  @Transactional  public Transport getById(Serializable id) {  log.info("查询id为{}的运费模板", id);  //先查出来 再查对应的收费模板和免费模板  Transport transport = super.getById(id);  //查询收费的  transport.setTransfees(getTransFees(transport.getTransportId()));  //查询免费的  transport.setTransfeeFrees(getTransFeesFree(transport.getTransportId()));  return super.getById(id);  }  /\*\*  \* 查询免费的模板  \*  \* @param transportId  \* @return  \*/  private List<TransfeeFree> getTransFeesFree(Long transportId) {  //查询免费模板  List<TransfeeFree> transfeeFrees = transfeeFreeMapper.selectList(new LambdaQueryWrapper<TransfeeFree>()  .eq(TransfeeFree::getTransportId, transportId)  );  if (ObjectUtils.isEmpty(transfeeFrees)) {  return Collections.emptyList();  }  transfeeFrees.forEach(transfeeFree -> {  //循环查询 免费的城市  List<Object> cityIds = transcityFreeMapper.selectObjs(new LambdaQueryWrapper<TranscityFree>()  .select(TranscityFree::getFreeCityId)  .eq(TranscityFree::getTransfeeFreeId, transfeeFree.getTransfeeFreeId())  );  if (!ObjectUtils.isEmpty(cityIds)) {  //如果城市不等空  List<Area> areas = areaMapper.selectList(new LambdaQueryWrapper<Area>()  .in(Area::getAreaId, cityIds)  );  transfeeFree.setFreeCityList(areas);  }  });  return transfeeFrees;  }  /\*\*  \* 查询所有收费的模板  \*  \* @param transportId  \* @return  \*/  private List<Transfee> getTransFees(Long transportId) {  //先查询收费的模板  List<Transfee> transfees = transfeeMapper.selectList(new LambdaQueryWrapper<Transfee>()  .eq(Transfee::getTransportId, transportId)  );  if (ObjectUtils.isEmpty(transfees)) {  return Collections.emptyList();  }  //循环查询收费模板对应的城市  transfees.forEach(transfee -> {  List<Object> cityIds = transcityMapper.selectObjs(new LambdaQueryWrapper<Transcity>()  .select(Transcity::getCityId)  .eq(Transcity::getTransfeeId, transfee.getTransfeeId())  );  if (!ObjectUtils.isEmpty(cityIds)) {  //如果城市不为空，查询城市  List<Area> areas = areaMapper.selectList(new LambdaQueryWrapper<Area>()  .in(Area::getAreaId, cityIds)  );  //设置对应模板的城市  transfee.setCityList(areas);  }  });  return transfees;  }  @Override  @Transactional  public boolean save(Transport entity) {  //新增一个运费模板  boolean save = super.save(entity);  if (save) {  //处理包邮的  handlerFree(entity);  //处理收费的  handlerFee(entity);  }  return save;  }  @Transactional  @CacheEvict(key = "#entity.transportId")  public boolean updateById(Transport entity) {  boolean flag = removeById(entity.getTransportId());  if (flag) {  save(entity);  return true;  }  return false;  }  @CacheEvict(key = "#id")  @Transactional  @Override  public boolean removeById(Serializable id) {  boolean b = super.removeById(id); // transport  if (b) {  //删除收费的  deleteFee(id);  //删除包邮的  deleteFree(id);  }  return b;  }  private void deleteFee(Serializable id) {  List<Object> transCityIds = transfeeMapper.selectObjs(new LambdaQueryWrapper<Transfee>().  select(Transfee::getTransfeeId).  eq(Transfee::getTransportId, id));  if (transCityIds != null && !transCityIds.isEmpty()) {  // 删除和城市的中间表  transcityMapper.delete(new LambdaQueryWrapper<Transcity>().in(Transcity::getTransfeeId, transCityIds));  }  // 删除自己  transfeeMapper.delete(new LambdaQueryWrapper<Transfee>().  eq(Transfee::getTransportId, id));  }  private void deleteFree(Serializable id) {  List<Object> transCityFreeIds = transfeeFreeMapper.selectObjs(new LambdaQueryWrapper<TransfeeFree>().  select(TransfeeFree::getTransfeeFreeId).  eq(TransfeeFree::getTransportId, id));  if (transCityFreeIds != null && !transCityFreeIds.isEmpty()) {  // 删除和城市的中间表  transcityFreeMapper.delete(new LambdaQueryWrapper<TranscityFree>().in(TranscityFree::getTransfeeFreeId, transCityFreeIds));  }  // 删除自己  transfeeFreeMapper.delete(new LambdaQueryWrapper<TransfeeFree>().  eq(TransfeeFree::getTransportId, id));  }  /\*\*  \* 处理收费的  \*  \* @param entity  \*/  private void handlerFee(Transport entity) {  if (entity.getIsFreeFee().equals(1)) {  //得到收费对象  Transfee transfee = entity.getTransfees().get(0);  //设置id  transfee.setTransportId(entity.getTransportId());  transfeeMapper.insert(transfee);  return;  }  List<Transfee> transfees = entity.getTransfees(); // 所有的收费的情况  if (!transfees.isEmpty()) {  for (Transfee transfee : transfees) {  insertTransFee(entity, transfee);  }  }  }  private void insertTransFee(Transport entity, Transfee transfee) {  transfee.setTransportId(entity.getTransportId());  int insert = transfeeMapper.insert(transfee);  if (insert > 0) {  List<Area> cityList = transfee.getCityList();  if (!cityList.isEmpty()) {  for (Area area : cityList) {  Transcity transcity = new Transcity();  transcity.setCityId(area.getAreaId());  transcity.setTransfeeId(transfee.getTransfeeId());  transcityMapper.insert(transcity);  }  }  }  }  /\*\*  \* 处理免费的  \*  \* @param entity  \*/  private void handlerFree(Transport entity) {  if (entity.getIsFreeFee().equals(1)) { //买家包邮  return;  }  // 条件包邮  List<TransfeeFree> transfeeFrees = entity.getTransfeeFrees();  if (!transfeeFrees.isEmpty()) {  for (TransfeeFree transfeeFree : transfeeFrees) {  insertTransFeeFree(entity, transfeeFree);  }  }  }  private void insertTransFeeFree(Transport entity, TransfeeFree transfeeFree) {  transfeeFree.setTransportId(entity.getTransportId());  int insert = transfeeFreeMapper.insert(transfeeFree);  if (insert > 0) {  List<Area> freeCityList = transfeeFree.getFreeCityList();  if (!freeCityList.isEmpty()) {  for (Area area : freeCityList) {  TranscityFree transcityFree = new TranscityFree();  transcityFree.setFreeCityId(area.getAreaId());  transcityFree.setTransfeeFreeId(transfeeFree.getTransfeeFreeId().longValue());  transcityFreeMapper.insert(transcityFree);  }  }  }  }  } |