# 公共模块优化，商品搜索

## ego-common模块的优化

### 新增全局异常处理

|  |
| --- |
| @RestControllerAdvice  @Slf4j  public class WebLogError {  /\*\*  \* 授权异常  \*  \* @return  \*/  @ExceptionHandler({AuthenticationException.class})  public ResponseEntity<String> authException() {  return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("你的登录过期，请重新登录");  }  /\*\*  \* 参数异常  \*  \* @param e  \* @return  \*/  @ExceptionHandler({IllegalArgumentException.class})  public ResponseEntity<String> paramException(IllegalArgumentException e) {  return ResponseEntity.badRequest().body(e.getMessage());  }  /\*\*  \* 运行时异常  \*  \* @param e  \* @return  \*/  @ExceptionHandler({RuntimeException.class})  public ResponseEntity<String> runtimeException(RuntimeException e) {  log.error("运行异常", e);  return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("服务器维修中!");  }  /\*\*  \* 方法参数校验异常  \*  \* @param bindException  \* @return  \*/  @ExceptionHandler({MethodArgumentNotValidException.class})  public ResponseEntity<String> methodArgumentNotValidException(MethodArgumentNotValidException bindException) {  BindingResult bindingResult = bindException.getBindingResult();  System.out.println(bindingResult);  StringBuilder sb = new StringBuilder("数据校验失败，原因是:");  List<FieldError> fieldErrors = bindingResult.getFieldErrors();  for (FieldError fieldError : fieldErrors) {  sb.append("字段:" + fieldError.getField() + "," + fieldError.getDefaultMessage() + "!");  }  return ResponseEntity.badRequest().body(sb.toString());  }  } |

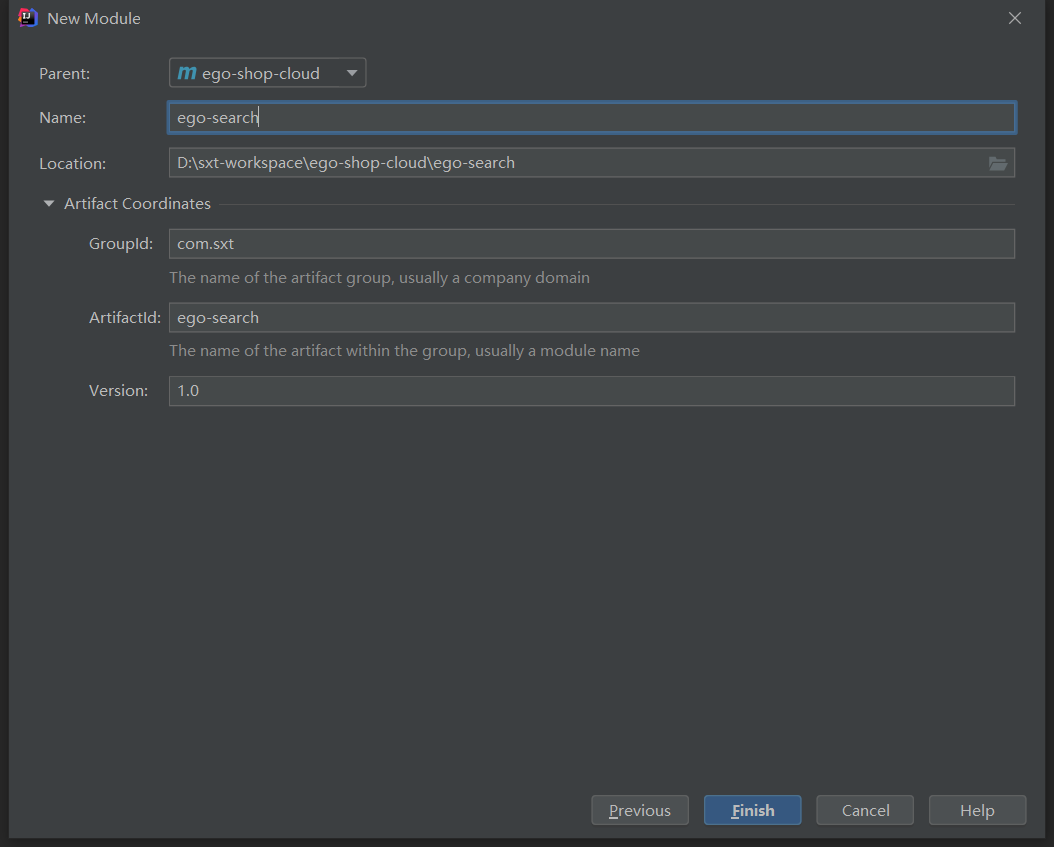
## ego-search搜索模块的搭建

功能定位：商品的查询

其他模块集成：商品模块等

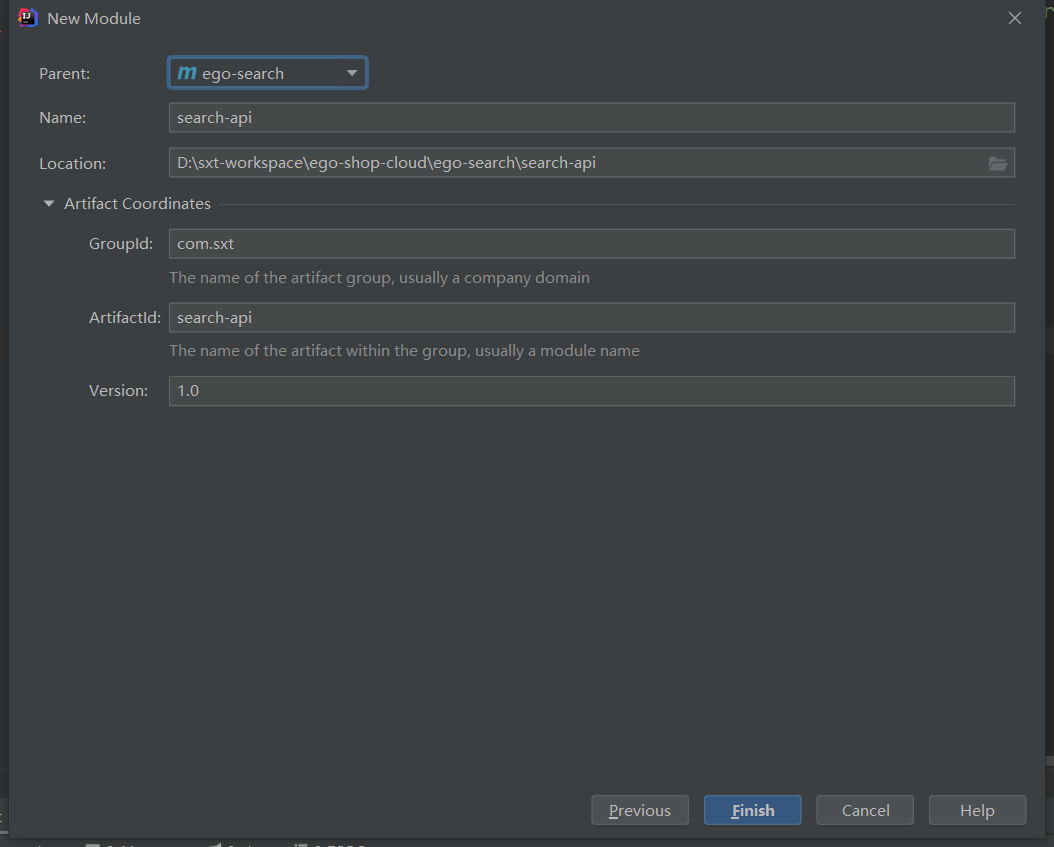
### ego-search父模块搭建

#### 在ego-shop-cloud下面创建一个maven模块



### search-api模块的搭建

#### 在ego-search下面创建一个maven模块

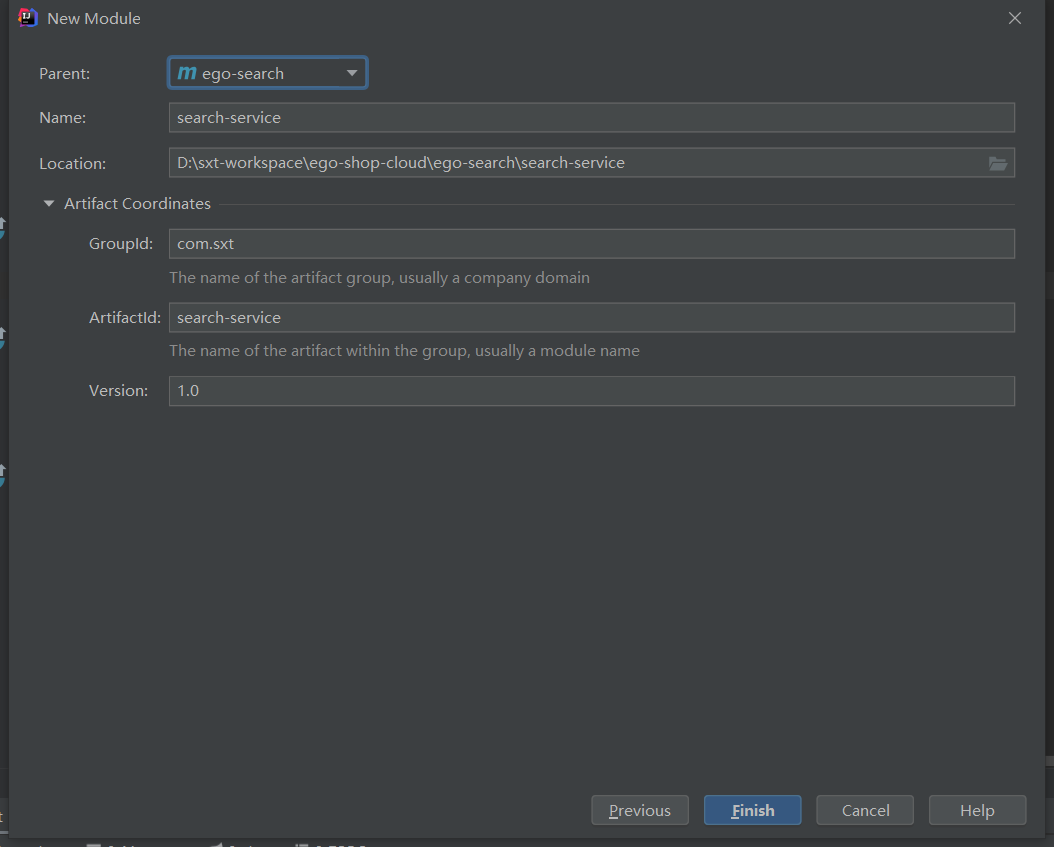


#### 修改pom.xml文件

|  |
| --- |
| <?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-search</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>search-api</artifactId>  <description>ego商城搜索模块api</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>product-api</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>ego-common</artifactId>  <version>1.0</version>  </dependency>    </dependencies>  </project> |

### search-service模块的搭建

#### 在ego-search下面创建一个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-search</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>search-service</artifactId>  <description>ego商城搜索的service</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>search-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.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</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(exclude = DataSourceAutoConfiguration.class)  @EnableEurekaClient  public class SearchServiceApplication {  public static void main(String[] args) {  SpringApplication.run(SearchServiceApplication.class, args);  }  } |

#### 添加bootstrap.yml配置文件

|  |
| --- |
| spring:  application:  name: search-service  cloud:  config:  discovery:  enabled: true  service-id: config-server  profile: dev  name: search-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 |

#### 添加远端search-service/search-service.yml配置文件

|  |
| --- |
| server:  port: ${APP\_PORT:8083}  spring:  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  elasticsearch:  rest:  uris: http://192.168.226.129:9200  management:  endpoints:  web:  exposure:  include: '\*'  swagger2:  base-package: com.sxt.controller  name: sxt  url: https://gitee.com/smiledouble  email: 775610843@qq.com  title: 商品搜搜系统接口  description: 实现商品搜索系统的所有接口  license: cxs  license-url: https://gitee.com/smiledouble  terms-of-service-url: https://gitee.com/smiledouble  feign:  hystrix:  enabled: false  ribbon:  ReadTimeout: 18000  ConnectTimeout: 18000  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 |

#### 添加搜索的SearchController

|  |
| --- |
| @RestController  @Api(tags = "搜索服务")  public class SearchController {  @Autowired  private SearchService searchService;  @ApiOperation("关键字查询")  @GetMapping("/search/searchProdPage")  public ResponseEntity<Page<ProductEs>> findByKeywords(Page<ProductEs> page, @RequestParam("0") Integer sort, @RequestParam("\*") String prodName) {  Page<ProductEs> productEsPage = searchService.findByKeyWords(page, prodName, sort);  return ResponseEntity.ok().body(productEsPage);  }  @GetMapping("/prod/pageProd")  @ApiOperation("根据分类id查询")  public ResponseEntity<Page<ProductEs>> findByCategoryId(@RequestParam Long cateGoryId) {  Page<ProductEs> page = new Page<>();  page.setSize(20L);  Page<ProductEs> productEsPage = searchService.findByCategoryId(page, cateGoryId, 0);  return ResponseEntity.ok().body(productEsPage);  }  @GetMapping("/prod/prodListByTagId")  @ApiOperation("根据标签id查询")  public ResponseEntity<Page<ProductEs>> findByTagId(Page<ProductEs> page, @RequestParam Long tagId) {  Page<ProductEs> productEsPage = searchService.findByTagId(page, tagId, 0);  return ResponseEntity.ok().body(productEsPage);  }  } |

#### 添加搜索的接口SearchService

|  |
| --- |
| public interface SearchService {  /\*\*  \* 根据关键字 分页查询  \*  \* @param page  \* @param keywords  \* @param sort  \* @return  \*/  Page<ProductEs> findByKeyWords(Page<ProductEs> page, String keywords, Integer sort);  /\*\*  \* 根据标签分组 分页查询商品  \* 今日热卖的  \* 秒杀的  \*  \* @param page  \* @param tagId  \* @param sort  \* @return  \*/  Page<ProductEs> findByTagId(Page<ProductEs> page, Long tagId, Integer sort);  /\*\*  \* 根据分类id分页查询商品  \*  \* @param page  \* @param categoryId  \* @param sort  \* @return  \*/  Page<ProductEs> findByCategoryId(Page<ProductEs> page, Long categoryId, Integer sort);  /\*\*  \* 根据商品的ids 批量查询商品  \*  \* @param prodIds  \* @return  \*/  List<ProductEs> findByProductIds(List<Long> prodIds);  } |

#### 添加搜索的实现类SearchServiceImpl

|  |
| --- |
| @Service  public class SearchServiceImpl implements SearchService {  @Autowired  private ElasticsearchRestTemplate elasticsearchRestTemplate;  @Autowired  private ProductDao productDao;  /\*\*  \* 根据关键字 分页查询  \*  \* @param page  \* @param keywords  \* @param sort  \* @return  \*/  @Override  public Page<ProductEs> findByKeyWords(Page<ProductEs> page, String keywords, Integer sort) {  MatchQueryBuilder esQuery = QueryBuilders.matchQuery("prodName", keywords);  return query(esQuery, page, sort, keywords);  }  /\*\*  \* 根据标签分组 分页查询商品  \* 今日热卖的  \* 秒杀的  \*  \* @param page  \* @param tagId  \* @param sort  \* @return  \*/  @Override  public Page<ProductEs> findByTagId(Page<ProductEs> page, Long tagId, Integer sort) {  TermQueryBuilder queryBuilder = QueryBuilders.termQuery("tagList", tagId);  return query(queryBuilder, page, sort, null);  }  /\*\*  \* 根据分类id分页查询商品  \*  \* @param page  \* @param categoryId  \* @param sort  \* @return  \*/  @Override  public Page<ProductEs> findByCategoryId(Page<ProductEs> page, Long categoryId, Integer sort) {  TermQueryBuilder queryBuilder = QueryBuilders.termQuery("categoryId", categoryId);  return query(queryBuilder, page, sort, null);  }  /\*\*  \* 根据商品的ids 批量查询商品  \*  \* @param prodIds  \* @return  \*/  @Override  public List<ProductEs> findByProductIds(List<Long> prodIds) {  if (ObjectUtils.isEmpty(prodIds)) {  return Collections.emptyList();  }  List<ProductEs> productByIds = (List<ProductEs>) productDao.findAllById(prodIds);  return productByIds;  }  /\*\*  \* 排序  \*  \* @param sort  \* @return  \*/  private SortBuilder sort(Integer sort) {  switch (sort) {  case 0:  //praiseNumber 综合排序  return SortBuilders.fieldSort("praiseNumber").order(SortOrder.DESC);  case 1:  //soldNum销量  return SortBuilders.fieldSort("soldNum").order(SortOrder.DESC);  case 2:  //prodPrice价格排序  return SortBuilders.fieldSort("prodPrice").order(SortOrder.ASC);  default:  throw new IllegalArgumentException("排序方法不允许");  }  }  /\*\*  \* 查询，封装高亮结果集  \*  \* @param queryBuilder  \* @param page  \* @param sort  \* @return  \*/  private Page<ProductEs> query(AbstractQueryBuilder queryBuilder, Page<ProductEs> page, Integer sort, String keywords) {  int current = (int) page.getCurrent();  int size = (int) page.getSize();  //得到排序  SortBuilder sortBuilder = sort(sort);  //组装查询条件  NativeSearchQueryBuilder searchQueryBuilder = new NativeSearchQueryBuilder()  .withQuery(queryBuilder)  .withPageable(PageRequest.of(current, size))  .withSort(sortBuilder);  HighlightBuilder.Field field = null;  if (keywords != null) {  field = new HighlightBuilder.Field("prodName");  field.preTags("<span style = 'color:red'>");  field.postTags("</span>");  searchQueryBuilder.withHighlightFields(field);  }  //查询拿到结果  SearchHits<ProductEs> searchHits = elasticsearchRestTemplate.search(searchQueryBuilder.build(), ProductEs.class);  //封装高亮  List<SearchHit<ProductEs>> hits = searchHits.getSearchHits();  List<ProductEs> productEs = new ArrayList<>(hits.size() \* 2);  hits.forEach(hit -> {  //得到内容  ProductEs content = hit.getContent();  List<String> prodName = hit.getHighlightField("prodName");  if (prodName != null && prodName.size() > 0) {  //设置高亮的字段  content.setProdName(prodName.get(0));  }  productEs.add(content);  });  page.setTotal(searchHits.getTotalHits());  page.setRecords(productEs);  return page;  }  } |