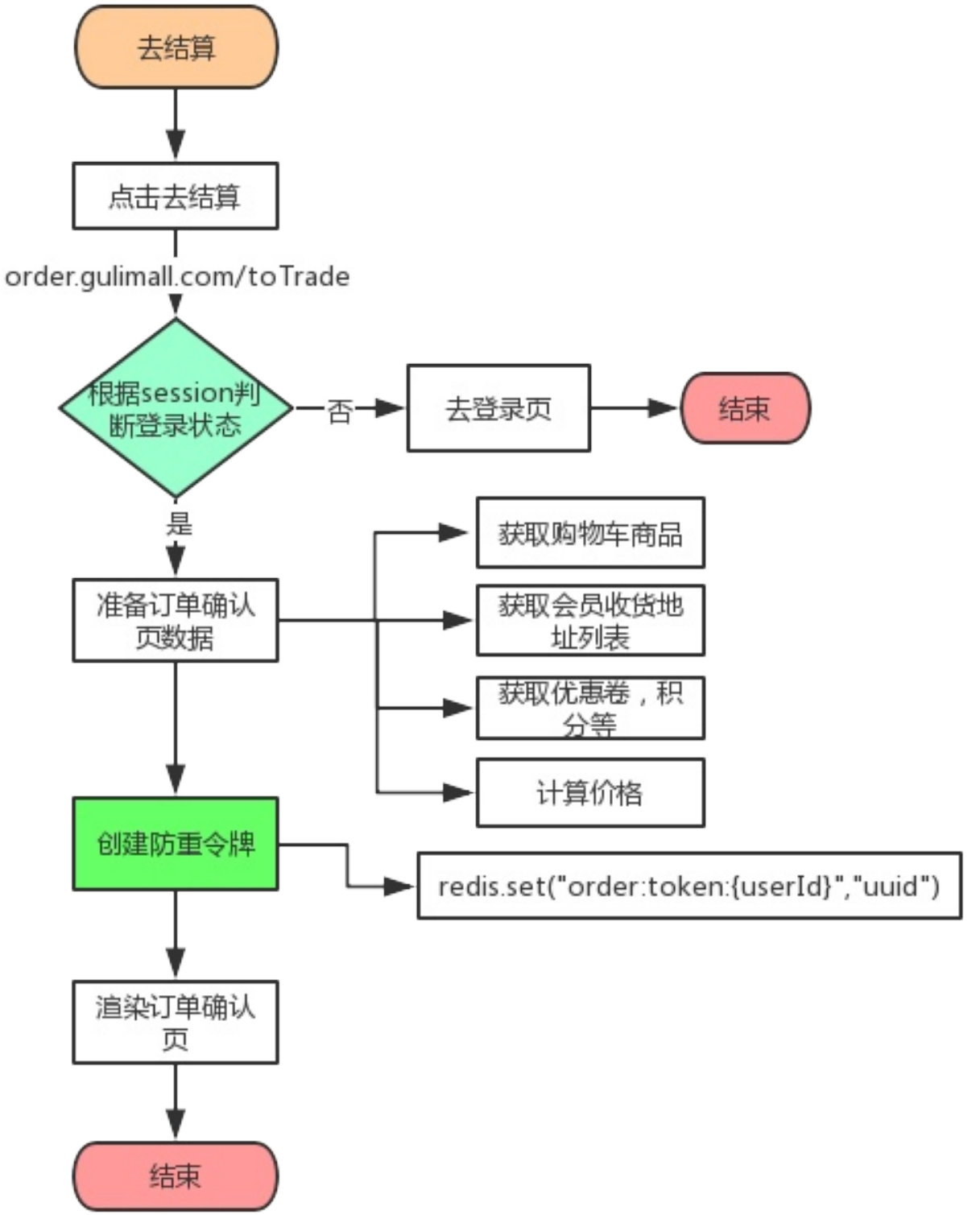
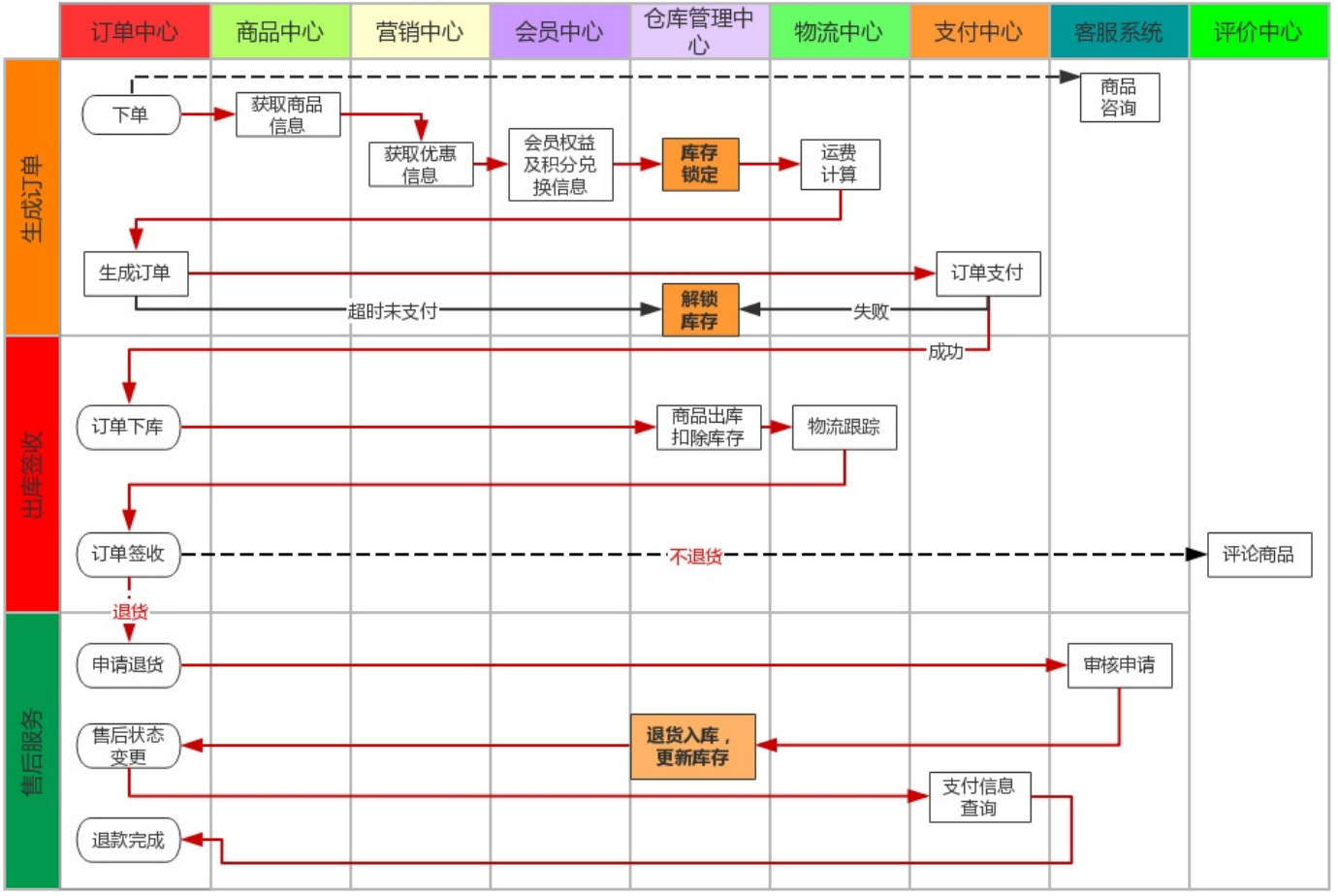
Gulimall

商城业务


去结算
点击去结算
order.gulimall.com/toTrade
根据session判断登录状态
否
去登录页
结束
是
准备订单确认页数据
获取购物车商品
获取会员收货地址列表
获取优惠卷，积分等
计算价格
创建防重令牌
redis.set("order:token:{userId}","uuid")
渲染订单确认页
结束


订单中心
商品中心
营销中心
会员中心
仓库管理中心
物流中心
支付中心
客服系统
评价中心
生成订单
出库签收
售后服务
下单
获取商品信息
获取优惠信息
会员权益及积分兑换信息
库存锁定
运费计算
商品咨询
生成订单
订单支付
超时未支付
解锁库存
失败
成功
订单下库
商品出库扣除库存
物流跟踪
订单签收
不退货
评论商品
退货
申请退货
审核申请
售后状态变更
退货入库，更新库存
支付信息查询
退款完成

一、商品上架

上架的商品才可以在网站展示。
上架的商品需要可以被检索。

1、商品 Mapping

分析：商品上架在 es 中是存 sku 还是 spu？
1）、检索的时候输入名字，是需要按照 sku 的 title 进行全文检索的
2）、检索使用商品规格，规格是 spu 的公共属性，每个 spu 是一样的
3）、按照分类 id 进去的都是直接列出 spu 的，还可以切换。
4）、我们如果将 sku 的全量信息保存到 es 中（包括 spu 属性）就太多量字段了。
5）、我们如果将 spu 以及他包含的 sku 信息保存到 es 中，也可以方便检索。但是 sku 属于 spu 的级联对象，在 es 中需要 nested 模型，这种性能差点。
6）、但是存储与检索我们必须性能折中。
7）、如果我们分拆存储，spu 和 attr 一个索引，sku 单独一个索引可能涉及的**问题**。
检索商品的名字，如“手机”，对应的 spu 有很多，我们要分析出这些 spu 的所有关联属性，再做一次查询，就必须将所有 spu_id 都发出去。假设有 1 万个数据，数据传输一次就 10000*4=4MB；并发情况下假设 1000 检索请求，那就是 4GB 的数据，，传输阻塞时间会很长，业务更加无法继续。
所以，我们如下设计，这样才是文档区别于关系型数据库的地方，宽表设计，不能去考虑数据库范式。
1）、PUT product

```
product 的 mapping
{
  "mappings": {
    "properties": {
      "skuId": {
        "type": "long"
      },
      "spuId": {
        "type": "keyword"
      },
      "skuTitle": {
        "type": "text",
        "analyzer": "ik_smart"
      },
      "skuPrice": {
        "type": "keyword"
      },
      "skuImg": {
        "type": "keyword",
```

```
        "index": false,
        "doc_values": false
      },
      "saleCount": {
        "type": "long"
      },
      "hasStock": {
        "type": "boolean"
      },
      "hotScore": {
        "type": "long"
      },
      "brandId": {
        "type": "long"
      },
      "catalogId": {
        "type": "long"
      },
      "brandName": {
        "type": "keyword",
        "index": false,
        "doc_values": false
      },
      "brandImg": {
        "type": "keyword",
        "index": false,
        "doc_values": false
      },
      "catalogName": {
        "type": "keyword",
        "index": false,
        "doc_values": false
      },
      "attrs": {
        "type": "nested",
        "properties": {
          "attrId": {
            "type": "long"
          },
          "attrName": {
            "type": "keyword",
            "index": false,
            "doc_values": false
          },
```

```
            "attrValue": {
              "type": "keyword"
            }
          }
        }
      }
    }
}
```

index：
默认 true，如果为 false，表示该字段不会被索引，但是检索结果里面有，但字段本身不能当做检索条件。

doc_values：
默认 true，设置为 false，表示不可以做排序、聚合以及脚本操作，这样更节省磁盘空间。
还可以通过设定 doc_values 为 true，index 为 false 来让字段不能被搜索但可以用于排序、聚合以及脚本操作：

2、上架细节

上架是将后台的商品放在 es 中可以提供检索和查询功能
1）、hasStock：代表是否有库存。默认上架的商品都有库存。如果库存无货的时候才需要更新一下 es
2）、库存补上以后，也需要重新更新一下 es
3）、hotScore 是热度值，我们只模拟使用点击率更新热度。点击率增加到一定程度才更新热度值。
4）、下架就是从 es 中移除检索项，以及修改 mysql 状态

商品上架步骤：
1）、先在 es 中按照之前的 mapping 信息，建立 product 索引。
2）、点击上架，查询出所有 sku 的信息，保存到 es 中
3）、es 保存成功返回，更新数据库的上架状态信息。

3、数据一致性

1）、商品无库存的时候需要更新 es 的库存信息
2）、商品有库存也要更新 es 的信息

二、商品检索

1、检索业务分析

商品检索三个入口：

1）、选择分类进入商品检索

2）、输入检索关键字展示检索页

3）、选择筛选条件进入

检索条件&排序条件

- 全文检索：skuTitle
- 排序：saleCount、hotScore、skuPrice
- 过滤：hasStock、skuPrice 区间、brandId、catalogId、attrs
- 聚合：attrs

完整的 url 参数

keyword=小米&**sort**=saleCount_desc/asc&**hasStock**=0/1&**skuPrice**=400_1900&**brandId**=1&**catalogId**=1&**attrs**=1_3G:4G:5G&**attrs**=2_骁龙 845&**attrs**=4_高清屏

2、检索语句构建

1、请求参数模型

```
@Data
public class SearchParam {

    private String keyword;//页面传递过来的全文匹配关键字  v
    private Long catalog3Id;//三级分类id   v

    /**
     *   sort=saleCount_asc/desc
     *   sort=skuPrice_asc/desc
     *   sort=hotScore_asc/desc
     */
    private String sort;//排序条件 v

    /**
     *过滤条件
     *  hasStock(是否有货)、skuPrice 区间、brandId、catalog3Id、attrs
     *  hasStock=0/1
     *  skuPrice=1_500/_500/500_
     *  brandId=1
     *  attrs=2_5 存:6 寸
     *
     */
    private Integer hasStock;//是否只显示有货  v 0（无库存）1（有库存）
    private String skuPrice;//价格区间查询  v
    private List<Long> brandId;//按照品牌进行查询，可以多选  v
    private List<String> attrs;//按照属性进行筛选  v
    private Integer pageNum = 1;//页码

    private String _queryString;//原生的所有查询条件
}
```

2、构建参数

```
private SearchRequest buildSearchRequrest(SearchParam param) {
    SearchSourceBuilder sourceBuilder = new SearchSourceBuilder();//构建DSL 语句的

    /**
     * 查询：过滤（按照属性，分类，品牌，价格区间，库存）
     */
```

```
    //1、构建bool - query
    BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
    //1.1、must-模糊匹配,
    if (!StringUtils.isEmpty(param.getKeyword())) {
        boolQuery.must(QueryBuilders.matchQuery("skuTitle",
param.getKeyword()));
    }
    //1.2、bool - filter - 按照三级分类id查询
    if (param.getCatalog3Id() != null) {
        boolQuery.filter(QueryBuilders.termQuery("catalogId",
param.getCatalog3Id()));
    }
    //1.2、bool - filter - 按照品牌id查询
    if (param.getBrandId() != null && param.getBrandId().size() > 0) {
        boolQuery.filter(QueryBuilders.termsQuery("brandId",
param.getBrandId()));
    }
    //1.2、bool - filter - 按照所有指定的属性进行查询
    if (param.getAttrs() != null && param.getAttrs().size() > 0) {
        for (String attrStr : param.getAttrs()) {
            //attrs=1_5寸:8寸&attrs=2_16G:8G
            BoolQueryBuilder nestedboolQuery = QueryBuilders.boolQuery();
            //attr = 1_5寸:8寸
            String[] s = attrStr.split("_");
            String attrId = s[0]; //检索的属性id
            String[] attrValues = s[1].split(":"); //这个属性的检索用的值
            nestedboolQuery.must(QueryBuilders.termQuery("attrs.attrId",
attrId));

nestedboolQuery.must(QueryBuilders.termsQuery("attrs.attrValue",
attrValues));
            //每一个必须都得生成一个nested查询
            NestedQueryBuilder nestedQuery =
QueryBuilders.nestedQuery("attrs", nestedboolQuery, ScoreMode.None);
            boolQuery.filter(nestedQuery);
        }

    }

    //1.2、bool - filter - 按照库存是否有进行查询
    if(param.getHasStock() != null){
        boolQuery.filter(QueryBuilders.termQuery("hasStock",
param.getHasStock() == 1));
```

```
    }


    //1.2、bool - filter - 按照价格区间
    if (!StringUtils.isEmpty(param.getSkuPrice())) {
        //1_500/_500/500_
        /**
         * "range": {
         *             "skuPrice": {
         *               "gte": 0,
         *               "lte": 6000
         *             }
         *           }
         */
        RangeQueryBuilder rangeQuery =
QueryBuilders.rangeQuery("skuPrice");

        String[] s = param.getSkuPrice().split("_");
        if (s.length == 2) {
            //区间
            rangeQuery.gte(s[0]).lte(s[1]);
        } else if (s.length == 1) {
            if (param.getSkuPrice().startsWith("_")) {
                rangeQuery.lte(s[0]);
            }

            if (param.getSkuPrice().endsWith("_")) {
                rangeQuery.gte(s[0]);
            }
        }

        boolQuery.filter(rangeQuery);
    }

    //把以前的所有条件都拿来进行封装
    sourceBuilder.query(boolQuery);


    /**
     * 排序，分页，高亮，
     */

    //2.1、排序
```

```
    if (!StringUtils.isEmpty(param.getSort())) {
        String sort = param.getSort();
        //sort=hotScore_asc/desc
        String[] s = sort.split("_");
        SortOrder order = s[1].equalsIgnoreCase("asc") ? SortOrder.ASC :
SortOrder.DESC;
        sourceBuilder.sort(s[0], order);
    }
    //2.2、分页 pageSize:5
    //  pageNum:1  from:0  size:5  [0,1,2,3,4]
    // pageNum:2  from:5   size:5
    //from = (pageNum-1)*size
    sourceBuilder.from((param.getPageNum() - 1) *
EsConstant.PRODUCT_PAGESIZE);
    sourceBuilder.size(EsConstant.PRODUCT_PAGESIZE);

    //2.3、高亮
    if (!StringUtils.isEmpty(param.getKeyword())) {
        HighlightBuilder builder = new HighlightBuilder();
        builder.field("skuTitle");
        builder.preTags("<b style='color:red'>");
        builder.postTags("</b>");
        sourceBuilder.highlighter(builder);
    }


    /**
     * 聚合分析
     */
    //1、品牌聚合
    TermsAggregationBuilder brand_agg =
AggregationBuilders.terms("brand_agg");
    brand_agg.field("brandId").size(50);
    //品牌聚合的子聚合

brand_agg.subAggregation(AggregationBuilders.terms("brand_name_agg").fi
eld("brandName").size(1));

brand_agg.subAggregation(AggregationBuilders.terms("brand_img_agg").fie
ld("brandImg").size(1));
    //TODO 1、聚合brand
    sourceBuilder.aggregation(brand_agg);
    //2、分类聚合 catalog_agg
```

```
    TermsAggregationBuilder catalog_agg =
AggregationBuilders.terms("catalog_agg").field("catalogId").size(20);

catalog_agg.subAggregation(AggregationBuilders.terms("catalog_name_agg"
).field("catalogName").size(1));
    //TODO 2、聚合catalog
    sourceBuilder.aggregation(catalog_agg);
    //3、属性聚合 attr_agg
    NestedAggregationBuilder attr_agg =
AggregationBuilders.nested("attr_agg", "attrs");

    //聚合出当前所有的attrId
    TermsAggregationBuilder attr_id_agg =
AggregationBuilders.terms("attr_id_agg").field("attrs.attrId");
    //聚合分析出当前attr_id对应的名字

attr_id_agg.subAggregation(AggregationBuilders.terms("attr_name_agg").f
ield("attrs.attrName").size(1));
    //聚合分析出当前attr_id对应的所有可能的属性值attrValue

attr_id_agg.subAggregation(AggregationBuilders.terms("attr_value_agg").
field("attrs.attrValue").size(50));
    attr_agg.subAggregation(attr_id_agg);
    //TODO 3、聚合attr
    sourceBuilder.aggregation(attr_agg);

    String s = sourceBuilder.toString();
    System.out.println("构建的DSL" + s);

    SearchRequest searchRequest = new SearchRequest(new
String[]{EsConstant.PRODUCT_INDEX}, sourceBuilder);
    return searchRequest;
}
```

3、结果提取封装

1、响应数据模型

```
@Data
public class SearchResult {

    //查询到的所有商品信息
    private List<SkuEsModel> products;

    /**
     * 以下是分页信息
     */
    private Integer pageNum;//当前页码
    private Long total;//总记录数
    private Integer totalPages;//总页码
    private List<Integer> pageNavs;

    private List<BrandVo> brands;//当前查询到的结果，所有涉及到的品牌
    private List<CatalogVo> catalogs;//当前查询到的结果，所有涉及到的所有分
类
    private List<AttrVo> attrs;//当前查询到的结果，所有涉及到的所有属性

    //==========以上是返回给页面的所有信息============


    //面包屑导航数据
    private List<NavVo> navs = new ArrayList<>();
    private List<Long> attrIds = new ArrayList<>();


    @Data
    public static class NavVo{
        private String navName;
        private String navValue;
        private String link;
    }



    @Data
    public static class BrandVo{
```

```
        private Long brandId;
        private String brandName;

        private String brandImg;
    }

    @Data
    public static class CatalogVo{
        private Long catalogId;
        private String catalogName;

    }

    @Data
    public static class AttrVo{
        private Long attrId;
        private String attrName;

        private List<String> attrValue;
    }
}
```

2、响应结果封装

```
   private SearchResult buildSearchResult(SearchResponse response,
SearchParam param) {

        SearchResult result = new SearchResult();
        //1、返回的所有查询到的商品
        SearchHits hits = response.getHits();
        List<SkuEsModel> esModels = new ArrayList<>();
        if (hits.getHits() != null && hits.getHits().length > 0) {
            for (SearchHit hit : hits.getHits()) {
                String sourceAsString = hit.getSourceAsString();
                SkuEsModel esModel = JSON.parseObject(sourceAsString,
SkuEsModel.class);
                if(!StringUtils.isEmpty(param.getKeyword())){
                    HighlightField skuTitle =
hit.getHighlightFields().get("skuTitle");
                    String string = skuTitle.getFragments()[0].string();
                    esModel.setSkuTitle(string);
```

```
                }
                esModels.add(esModel);
            }
            ;
        }
        result.setProducts(esModels);



//        //2、当前所有商品涉及到的所有属性信息
        List<SearchResult.AttrVo> attrVos = new ArrayList<>();
        ParsedNested attr_agg =
response.getAggregations().get("attr_agg");
        ParsedLongTerms attr_id_agg =
attr_agg.getAggregations().get("attr_id_agg");
        for (Terms.Bucket bucket : attr_id_agg.getBuckets()) {
            SearchResult.AttrVo attrVo = new SearchResult.AttrVo();
            //1、得到属性的id
            long attrId = bucket.getKeyAsNumber().longValue();
            //2、得到属性的名字
            String attrName = ((ParsedStringTerms)
bucket.getAggregations().get("attr_name_agg")).getBuckets().get(0).getK
eyAsString();

            //3、得到属性的所有值
            List<String> attrValues = ((ParsedStringTerms)
bucket.getAggregations().get("attr_value_agg")).getBuckets().stream().m
ap(item -> {
                String keyAsString = ((Terms.Bucket)
item).getKeyAsString();
                return keyAsString;
            }).collect(Collectors.toList());

            attrVo.setAttrId(attrId);
            attrVo.setAttrName(attrName);
            attrVo.setAttrValue(attrValues);



            attrVos.add(attrVo);
        }
```

```
        result.setAttrs(attrVos);
//        //3、当前所有商品涉及到的所有品牌信息
        List<SearchResult.BrandVo> brandVos = new ArrayList<>();
        ParsedLongTerms brand_agg =
response.getAggregations().get("brand_agg");
        for (Terms.Bucket bucket : brand_agg.getBuckets()) {
            SearchResult.BrandVo brandVo = new SearchResult.BrandVo();
            //1、得到品牌的id
            long brandId = bucket.getKeyAsNumber().longValue();
            //2、得到品牌的名
            String brandName = ((ParsedStringTerms)
bucket.getAggregations().get("brand_name_agg")).getBuckets().get(0).get
KeyAsString();
            //3、得到品牌的图片
            String brandImg = ((ParsedStringTerms)
bucket.getAggregations().get("brand_img_agg")).getBuckets().get(0).getK
eyAsString();
            brandVo.setBrandId(brandId);
            brandVo.setBrandName(brandName);
            brandVo.setBrandImg(brandImg);
            brandVos.add(brandVo);
        }

        result.setBrands(brandVos);
//        //4、当前所有商品涉及到的所有分类信息
        ParsedLongTerms catalog_agg =
response.getAggregations().get("catalog_agg");
        List<SearchResult.CatalogVo> catalogVos = new ArrayList<>();
        List<? extends Terms.Bucket> buckets = catalog_agg.getBuckets();
        for (Terms.Bucket bucket : buckets) {
            SearchResult.CatalogVo catalogVo = new
SearchResult.CatalogVo();
            //得到分类id
            String keyAsString = bucket.getKeyAsString();
            catalogVo.setCatalogId(Long.parseLong(keyAsString));

            //得到分类名
            ParsedStringTerms catalog_name_agg =
bucket.getAggregations().get("catalog_name_agg");
            String catalog_name =
catalog_name_agg.getBuckets().get(0).getKeyAsString();
            catalogVo.setCatalogName(catalog_name);
            catalogVos.add(catalogVo);
```

```
        }
        result.setCatalogs(catalogVos);
//        ========以上从聚合信息中获取======


//        //5、分页信息-页码
        result.setPageNum(param.getPageNum());
//        //5、分页信息-总记录树
        long total = hits.getTotalHits().value;
        result.setTotal(total);
//        //5、分页信息-总页码-计算  11/2 = 5 .. 1
        int totalPages = (int) total % EsConstant.PRODUCT_PAGESIZE == 0 ?
(int) total / EsConstant.PRODUCT_PAGESIZE : ((int) total /
EsConstant.PRODUCT_PAGESIZE + 1);
        result.setTotalPages(totalPages);


        List<Integer> pageNavs = new ArrayList<>();
        for (int i=1;i<=totalPages;i++){
            pageNavs.add(i);
        }
        result.setPageNavs(pageNavs);


        //6、构建面包屑导航功能
        if(param.getAttrs()!=null && param.getAttrs().size()>0){
            List<SearchResult.NavVo> collect =
param.getAttrs().stream().map(attr -> {
                //1、分析每个attrs 传过来的查询参数值。
                SearchResult.NavVo navVo = new SearchResult.NavVo();
//              attrs=2_5 存:6 寸
                String[] s = attr.split("_");
                navVo.setNavValue(s[1]);
                R r = productFeignService.attrInfo(Long.parseLong(s[0]));
                result.getAttrIds().add(Long.parseLong(s[0]));
                if(r.getCode() == 0){
                    AttrResponseVo data = r.getData("attr", new
TypeReference<AttrResponseVo>() {
                    });
                    navVo.setNavName( data.getAttrName());
                }else{
                    navVo.setNavName(s[0]);
                }
```

```
//
                //2、取消了这个面包屑以后，我们要跳转到那个地方.将请求地址的url
里面的当前置空
                //拿到所有的查询条件，去掉当前。
                //attrs=  15_海思（Hisilicon）
                String replace = replaceQueryString(param, attr,"attrs");

navVo.setLink("http://search.gulimall.com/list.html?"+replace);

                return navVo;
            }).collect(Collectors.toList());



            result.setNavs(collect);
        }

        //品牌，分类
        if(param.getBrandId()!=null && param.getBrandId().size()>0){
            List<SearchResult.NavVo> navs = result.getNavs();
            SearchResult.NavVo navVo = new SearchResult.NavVo();

            navVo.setNavName("品牌");
            //TODO 远程查询所有品牌
            R r = productFeignService.brandsInfo(param.getBrandId());
            if(r.getCode() == 0){
                List<BrandVo> brand = r.getData("brand", new
TypeReference<List<BrandVo>>() {
                });
                StringBuffer buffer = new StringBuffer();
                String replace = "";
                for (BrandVo brandVo : brand) {
                    buffer.append(brandVo.getBrandName()+";");
                    replace = replaceQueryString(param,
brandVo.getBrandId()+"","brandId");
                }
                navVo.setNavValue(buffer.toString());

navVo.setLink("http://search.gulimall.com/list.html?"+replace);
            }

            navs.add(navVo);
        }
```

```
        //TODO 分类：不需要导航取消









        return result;
    }

    private String replaceQueryString(SearchParam param, String
value,String key) {
        String encode = null;
        try {
            encode = URLEncoder.encode(value, "UTF-8");
            encode = encode.replace("+","%20");//浏览器对空格编码和java 不一
样
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return param.get_queryString().replace("&"+key+"=" + encode, "");
    }
```

三、商品详情

1、详情数据

```
// 1. 获取sku的基本信息     0.5s

// 2. 获取sku的图片信息     0.5s

// 3. 获取sku的促销信息     1s

// 4. 获取spu的所有销售属性   1s

// 5. 获取规格参数组及组下的规格参数    1.5s

// 6. spu详情       1s
```

2、查询详情

```
@Override
public SkuItemVo item(Long skuId) throws ExecutionException, InterruptedException {
    SkuItemVo skuItemVo  = new SkuItemVo();

    CompletableFuture<SkuInfoEntity> infoFuture = CompletableFuture.supplyAsync(() ->
{
        //1、sku 基本信息获取  pms_sku_info
        SkuInfoEntity info = getById(skuId);
        skuItemVo.setInfo(info);
        return info;
    }, executor);

    CompletableFuture<Void> saleAttrFuture = infoFuture.thenAcceptAsync((res) -> {
        //3、获取 spu 的销售属性组合。
        List<SkuItemSaleAttrVo> saleAttrVos =
skuSaleAttrValueService.getSaleAttrsBySpuId(res.getSpuId());
        skuItemVo.setSaleAttr(saleAttrVos);
    }, executor);

    CompletableFuture<Void> descFuture = infoFuture.thenAcceptAsync(res -> {
        //4、获取 spu 的介绍  pms_spu_info_desc
        SpuInfoDescEntity spuInfoDescEntity =
spuInfoDescService.getById(res.getSpuId());
        skuItemVo.setDesp(spuInfoDescEntity);
    }, executor);

    CompletableFuture<Void> baseAttrFuture = infoFuture.thenAcceptAsync(res -> {
        //5、获取 spu 的规格参数信息。
        List<SpuItemAttrGroupVo> attrGroupVos =
attrGroupService.getAttrGroupWithAttrsBySpuId(res.getSpuId(), res.getCatalogId());
        skuItemVo.setGroupAttrs(attrGroupVos);
    }, executor);



    //2、sku 的图片信息  pms_sku_images
    CompletableFuture<Void> imageFuture = CompletableFuture.runAsync(() -> {
        List<SkuImagesEntity> images = imagesService.getImagesBySkuId(skuId);
        skuItemVo.setImages(images);
    }, executor);
```

```
    //等到所有任务都完成
CompletableFuture.allOf(saleAttrFuture,descFuture,baseAttrFuture,imageFuture).get()
;

    return skuItemVo;
}
```

3、sku 组合切换

页面遍历

```
<div class="box-attr clear" th:each="attr:${item.saleAttr}">
                        <dl>
                            <dt>选择[[${attr.attrName}]]</dt>
                            <dd th:each="vals:${attr.attrValues}"
                                >
                                <a class="sku_attr_value"
th:attr="skus=${vals.skuIds},class=${#lists.contains(#strings.listSplit(val
s.skuIds,','), item.info.skuId.toString())?'sku_attr_value
checked':'sku_attr_value'}"  >
                                    [[${vals.attrValue}]]
<!--                                 <img
src="/static/item/img/59ddfcb1Nc3edb8f1.jpg" /> -->
                                </a>
                            </dd>
                        </dl>
                    </div>
```

动态切换

```
$(".sku_attr_value").click(function(){
   //1、点击的元素先添加上自定义的属性。为了识别我们是刚才被点击的
   var skus = new Array();
   $(this).addClass("clicked");
   var curr = $(this).attr("skus").split(",");
   //当前被点击的所有 sku 组合数组放进去
   skus.push(curr);
   //去掉同一行的所有的 checked

$(this).parent().parent().find(".sku_attr_value").removeClass("checked");

   $("a[class='sku_attr_value checked']").each(function(){
```

```
        skus.push($(this).attr("skus").split(","));
    });
    console.log(skus);

    //2、取出他们的交集，得到skuId
    var filterEle = skus[0];
    for(var i = 1;i<skus.length;i++){
        filterEle = $(filterEle).filter(skus[i]);
    }

    console.log(filterEle[0]);
    location.href = "http://item.gulimall.com/"+filterEle[0]+".html";

    //4、跳转
});
```

关键 sql

```
SELECT
    ssav.`attr_id` attr_id,
    ssav.`attr_name` attr_name,
    ssav.`attr_value`,
    GROUP_CONCAT(DISTINCT info.`sku_id`) sku_ids
  FROM `pms_sku_info` info
LEFT JOIN `pms_sku_sale_attr_value` ssav ON
ssav.`sku_id`=info.`sku_id`
WHERE info.`spu_id`=#{spuId}
GROUP BY ssav.`attr_id`,ssav.`attr_name`,ssav.`attr_value`
```

attr_id	attr_name	attr_value	sku_ids
9	颜色	亮黑色	3,4
9	颜色	星河银	1,2
9	颜色	罗兰紫	7,8
9	颜色	翡冷翠	5,6
12	版本	8GB+128GB	2,4,6,8
12	版本	8GB+256GB	1,3,5,7

四、购物车

1、购物车需求

1）、需求描述：

- 用户可以在**登录状态**下将商品添加到购物车**【用户购物车/在线购物车】**
  - **放入数据库**
  - **mongodb**
  - **放入 redis（采用）**
    登录以后，会将**临时购物车**的数据全部合并过来，并清空**临时购物车；**

- 用户可以在**未登录状态**下将商品添加到购物车**【游客购物车/离线购物车/临时购物车】**
  - 放入 localstorage（客户端存储，后台不存）
  - cookie
  - WebSQL
  - **放入 redis（采用）**
    浏览器即使关闭，下次进入，**临时购物车**数据都在

- 用户可以使用购物车一起结算下单
- 给购物车**添加商品**
- 用户可以**查询自己的购物车**
- 用户可以在购物车中**修改购买商品的数量**。
- 用户可以在购物车中**删除商品**。
- **选中不选中商品**
- 在购物车中展示商品优惠信息
- 提示购物车商品价格变化

2）、数据结构

因此每一个购物项信息，都是一个对象，基本字段包括：

```
{
    skuId: 2131241,
    check: true,
    title: "Apple iphone.....",
    defaultImage: "...",
    price: 4999,
    count: 1,
    totalPrice: 4999,
    skuSaleVO: {...}
}
```

另外，购物车中不止一条数据，因此最终会是对象的数组。即：

```
[
    {...},{...},{...}
]
```

Redis 有 5 种不同数据结构，这里选择哪一种比较合适呢？Map<String, List<String>>

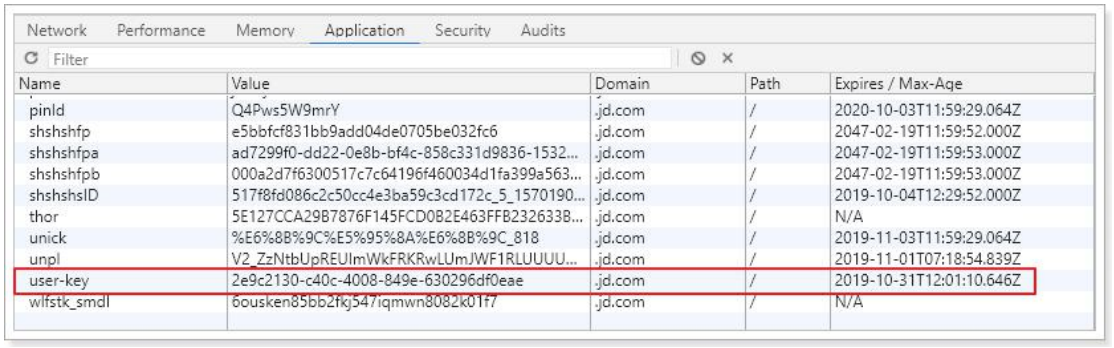
- 首先不同用户应该有独立的购物车，因此购物车应该以用户的作为 key 来存储，Value 是用户的所有购物车信息。这样看来基本的`k-v`结构就可以了。
- 但是，我们对购物车中的商品进行增、删、改操作，基本都需要根据商品 id 进行判断，为了方便后期处理，我们的购物车也应该是`k-v`结构，key 是商品 id，value 才是这个商品的购物车信息。

综上所述，我们的购物车结构是一个双层 Map：Map<String,Map<String,String>>

- 第一层 Map，Key 是用户 id
- 第二层 Map，Key 是购物车中商品 id，值是购物项数据

3）、流程

参照京东



user-key 是随机生成的 id，不管有没有登录都会有这个 cookie 信息。

两个功能：新增商品到购物车、查询购物车。
新增商品：判断是否登录
- 是：则添加商品到后台 Redis 中，把 user 的唯一标识符作为 key。
- 否：则添加商品到后台 redis 中，使用随机生成的 user-key 作为 key。

查询购物车列表：判断是否登录

- 否：直接根据 user-key 查询 redis 中数据并展示
- 是：已登录，则需要先根据 user-key 查询 redis 是否有数据。
  - 有：需要提交到后台添加到 redis，合并数据，而后查询。
  - 否：直接去后台查询 redis，而后返回。

2、临时购物车

```
/**
 * 获取到我们要操作的购物车
 * @return
 */
private BoundHashOperations<String, Object, Object> getCartOps() {
    UserInfoTo userInfoTo = CartInterceptor.threadLocal.get();
    String cartKey = "";
    if (userInfoTo.getUserId() != null) {
        //gulimall:cart:1
        cartKey = CART_PREFIX + userInfoTo.getUserId();
    } else {
        cartKey = CART_PREFIX + userInfoTo.getUserKey();
    }

    BoundHashOperations<String, Object, Object> operations =
redisTemplate.boundHashOps(cartKey);
    return operations;
}
```

3、登录购物车

```
@Override
public Cart getCart() throws ExecutionException, InterruptedException {

    Cart cart = new Cart();
    UserInfoTo userInfoTo = CartInterceptor.threadLocal.get();
    if(userInfoTo.getUserId()!=null){
        //1、登录
        String cartKey =CART_PREFIX+ userInfoTo.getUserId();
        //2、如果临时购物车的数据还没有进行合并【合并购物车】
        String tempCartKey = CART_PREFIX + userInfoTo.getUserKey();
        List<CartItem> tempCartItems = getCartItems(tempCartKey);
        if(tempCartItems!=null){
            //临时购物车有数据，需要合并
            for (CartItem item : tempCartItems) {
                addToCart(item.getSkuId(),item.getCount());
            }
            //清除临时购物车的数据
            clearCart(tempCartKey);
        }

        //3、获取登录后的购物车的数据【包含合并过来的临时购物车的数据，和登录后的购物车的数据】
        List<CartItem> cartItems = getCartItems(cartKey);
        cart.setItems(cartItems);

    }else{
        //2、没登录
        String cartKey =CART_PREFIX+ userInfoTo.getUserKey();
        //获取临时购物车的所有购物项
        List<CartItem> cartItems = getCartItems(cartKey);
        cart.setItems(cartItems);

    }
    return cart;
}
```

五、订单

1、订单中心

电商系统涉及到 3 流，分别时信息流，资金流，物流，而订单系统作为中枢将三者有机的集合起来。
订单模块是电商系统的枢纽，在订单这个环节上需求获取多个模块的数据和信息，同时对这些信息进行加工处理后流向下个环节，这一系列就构成了订单的信息流通。

1、订单构成

1、用户信息
用户信息包括用户账号、用户等级、用户的收货地址、收货人、收货人电话等组成，用户账户需要绑定手机号码，但是用户绑定的手机号码不一定是收货信息上的电话。用户可以添加多个收货信息，用户等级信息可以用来和促销系统进行匹配，获取商品折扣，同时用户等级还可以获取积分的奖励等

2、订单基础信息
订单基础信息是订单流转的核心，其包括订单类型、父/子订单、订单编号、订单状态、订单流转的时间等。
（1）订单类型包括实体商品订单和虚拟订单商品等，这个根据商城商品和服务类型进行区分。

（2）同时订单都需要做父子订单处理，之前在初创公司一直只有一个订单，没有做父子订单处理后期需要进行拆单的时候就比较麻烦，尤其是多商户商场，和不同仓库商品的时候，父子订单就是为后期做拆单准备的。
（3）订单编号不多说了，需要强调的一点是父子订单都需要有订单编号，需要完善的时候可以对订单编号的每个字段进行统一定义和诠释。
（4）订单状态记录订单每次流转过程，后面会对订单状态进行单独的说明。
（5）订单流转时间需要记录下单时间，支付时间，发货时间，结束时间/关闭时间等等

3、商品信息
商品信息从商品库中获取商品的 SKU 信息、图片、名称、属性规格、商品单价、商户信息等，从用户下单行为记录的用户下单数量，商品合计价格等。

4. 优惠信息
优惠信息记录用户参与的优惠活动，包括优惠促销活动，比如满减、满赠、秒杀等，用户使用的优惠券信息，优惠券满足条件的优惠券需要默认展示出来，具体方式已在之前的优惠券篇章做过详细介绍，另外还虚拟币抵扣信息等进行记录。

为什么把优惠信息单独拿出来而不放在支付信息里面呢？
因为优惠信息只是记录用户使用的条目，而支付信息需要加入数据进行计算，所以做为区分。

5. 支付信息
（1）支付流水单号，这个流水单号是在唤起网关支付后支付通道返回给电商业务平台的支付流水号，财务通过订单号和流水单号与支付通道进行对账使用。
（2）支付方式用户使用的支付方式，比如微信支付、支付宝支付、钱包支付、快捷支付等。支付方式有时候可能有两个——余额支付+第三方支付。
（3）商品总金额，每个商品加总后的金额；运费，物流产生的费用；优惠总金额，包括促销活动的优惠金额，优惠券优惠金额，虚拟积分或者虚拟币抵扣的金额，会员折扣的金额等之和；实付金额，用户实际需要付款的金额。

用户实付金额=商品总金额+运费-优惠总金额

6. 物流信息
物流信息包括配送方式，物流公司，物流单号，物流状态，物流状态可以通过第三方接口来获取和向用户展示物流每个状态节点。

2、订单状态

1. 待付款
用户提交订单后，订单进行预下单，目前主流电商网站都会唤起支付，便于用户快速完成支付，需要注意的是待付款状态下可以对库存进行锁定，锁定库存需要配置支付超时时间，超时后将自动取消订单，订单变更关闭状态。

2. 已付款/待发货
用户完成订单支付，订单系统需要记录支付时间，支付流水单号便于对账，订单下放到 WMS

系统，仓库进行调拨，配货，分拣，出库等操作。

3. 待收货/已发货

仓储将商品出库后，订单进入物流环节，订单系统需要同步物流信息，便于用户实时知悉物品物流状态

4. 已完成

用户确认收货后，订单交易完成。后续支付侧进行结算，如果订单存在问题进入售后状态

5. 已取消

付款之前取消订单。包括超时未付款或用户商户取消订单都会产生这种订单状态。

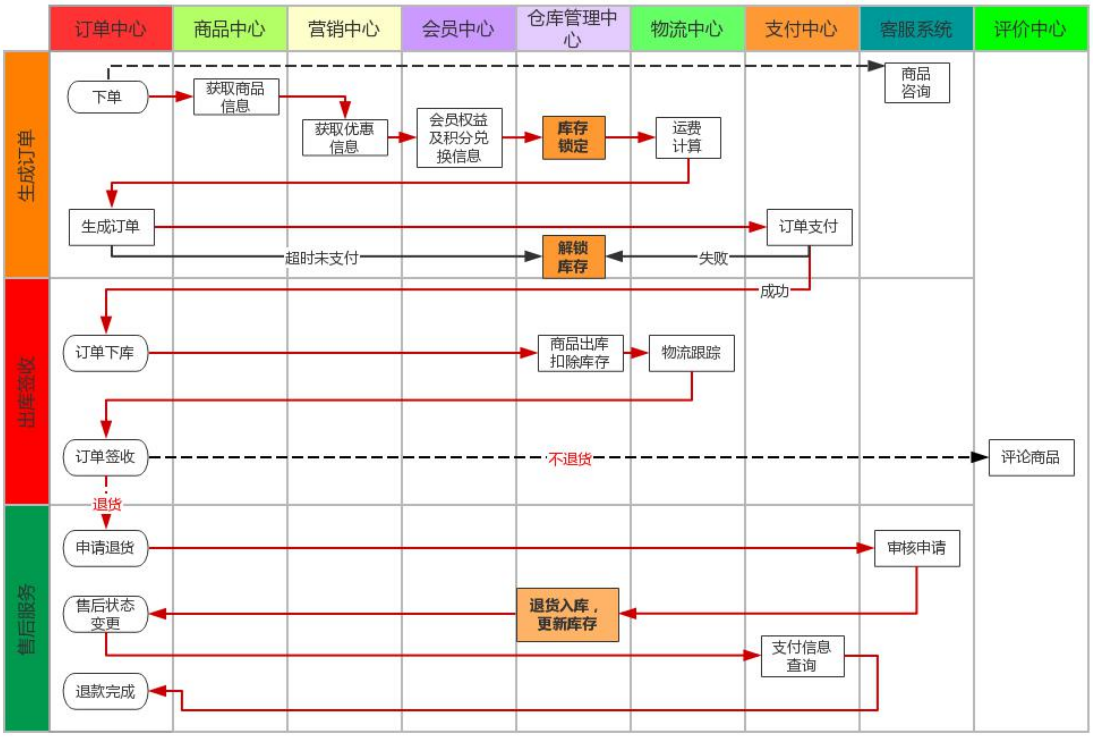
6. 售后中

用户在付款后申请退款，或商家发货后用户申请退换货。

售后也同样存在各种状态，当发起售后申请后生成售后订单，售后订单状态为待审核，等待商家审核，商家审核通过后订单状态变更为待退货，等待用户将商品寄回，商家收货后订单状态更新为待退款状态，退款到用户原账户后订单状态更新为售后成功。

2、订单流程

订单流程是指从订单产生到完成整个流转的过程，从而行程了一套标准流程规则。而不同的产品类型或业务类型在系统中的流程会千差万别，比如上面提到的线上实物订单和虚拟订单的流程，线上实物订单与 O2O 订单等，所以需要根据不同的类型进行构建订单流程。
不管类型如何订单都包括正向流程和逆向流程，对应的场景就是购买商品和退换货流程，正向流程就是一个正常的网购步骤：订单生成–>支付订单–>卖家发货–>确认收货–>交易成功。而每个步骤的背后，订单是如何在多系统之间交互流转的，可概括如下图



1、订单创建与支付

(1) 、订单创建前需要预览订单，选择收货信息等
(2) 、订单创建需要锁定库存，库存有才可创建，否则不能创建
(3) 、订单创建后超时未支付需要解锁库存
(4) 、支付成功后，需要进行拆单，根据商品打包方式，所在仓库，物流等进行拆单
(5) 、支付的每笔流水都需要记录，以待查账
(6) 、订单创建，支付成功等状态都需要给 MQ 发送消息，方便其他系统感知订阅

2、逆向流程

(1) 、修改订单，用户没有提交订单，可以对订单一些信息进行修改，比如配送信息，优惠信息，及其他一些订单可修改范围的内容，此时只需对数据进行变更即可。
(2) 、订单取消，**用户主动取消订单和用户超时未支付**，两种情况下订单都会取消订单，而超时情况是系统自动关闭订单，所以在订单支付的响应机制上面要做支付的

限时处理，尤其是在前面说的下单减库存的情形下面，可以保证快速的释放库存。另外需要需要处理的是促销优惠中使用的优惠券，权益等视平台规则，进行相应补回给用户。

(3) 、退款，在待发货订单状态下取消订单时，分为缺货退款和用户申请退款。如果是全部退款则订单更新为关闭状态，若只是做部分退款则订单仍需进行进行，同时生成一条退款的售后订单，走退款流程。退款金额需原路返回用户的账户。

(4) 、发货后的退款，发生在仓储货物配送，在配送过程中商品遗失，用户拒收，用户收货后对商品不满意，这样情况下用户发起退款的售后诉求后，需要商户进行退款的审核，双方达成一致后，系统更新退款状态，对订单进行退款操作，金额原路返回用户的账户，同时关闭原订单数据。仅退款情况下暂不考虑仓库系统变化。如果发生双方协调不一致情况下，可以申请平台客服介入。在退款订单商户不处理的情况下，系统需要做限期判断，比如 5 天商户不处理，退款单自动变更同意退款。

3、幂等性处理

参照幂等性文档

4、订单业务

1、搭建环境

订单服务引入页面，nginx 配置动静分离，上传静态资源到 nginx。编写 controller 跳转逻辑

2、订单确认页

可以发现订单结算页，包含以下信息：

1. 收货人信息：有更多地址，即有多个收货地址，其中有一个默认收货地址
2. 支付方式：货到付款、在线支付，不需要后台提供
3. 送货清单：配送方式（不做）及商品列表（根据购物车选中的 skuId 到数据库中查询）
4. 发票：不做
5. 优惠：查询用户领取的优惠券（不做）及可用积分（京豆）

OrderConfirmVo

```
@Data
public class OrderConfirmVO {

    // 收货地址，ums_member_receive_address 表
    private List<MemberReceiveAddressEntity> addresses;

    // 购物清单，根据购物车页面传递过来的 skuIds 查询
    private List<OrderItemVO> orderItems;

    // 可用积分，ums_member 表中的 integration 字段
    private Integer bounds;

    // 订单令牌，防止重复提交
    private String orderToken;
}
```

OrderItemVO（参照 Cart 对象）

```
public class OrderItemVo {
```

```
    private Long skuId;
    private Boolean check = true;
    private String title;
    private String image;
    private List<String> skuAttr;
    private BigDecimal price;
    private Integer count;
    private BigDecimal totalPrice;
```

3、创建订单

当用户点击提交订单按钮，应该收集页面数据提交到后台并生成订单数据。

1、数据模型

订单确认页，需要提交的数据：

```
@Data
public class OrderSubmitVO {

    //提交上次订单确认页给你的令牌;
    private String orderToken;

    private BigDecimal apyPrice; // 校验总价格时，拿计算价格和这个价格比较

    private Integer payType;//0-在线支付  1-货到付款

    private String delivery_company; // 配送方式

    // 订单清单可以不用提交，继续从购物车中获取

    // 地址信息，提交地址 id，会员 id 不需要提交
    Private Long addrId;


    // TODO：发票相关信息略
    // TODO：营销信息等
}
```

提交以后，需要响应的数据：

```
@Data
public class OrderSubmitResponseVO {
```

```
    private OrderEntity orderEntity;
    private Integer code;
    // 1-不可重复提交或页面已过期  2-库存不足  3-价格校验不合法  等
}
```

2、防止超卖

数据库 unsigned int 做最后的保证。

4、自动关单

订单超时未支付，需要取消订单

5、解锁库存

订单关闭，需要解锁已经占用的库存

库存锁定成功，订单回滚，保证最终一致性，也需要库存自动解锁

4、5 功能参照消息队列流程完成

六、秒杀

1、秒杀业务

秒杀具有瞬间高并发的特点，针对这一特点，必须要做限流 + 异步 + 缓存（页面静态化）+ 独立部署。

限流方式：

1. 前端限流，一些高并发的网站直接在前端页面开始限流，例如：小米的验证码设计
2. nginx 限流，直接负载部分请求到错误的静态页面：令牌算法 漏斗算法
3. 网关限流，限流的过滤器
4. 代码中使用分布式信号量
5. rabbitmq 限流（能者多劳：chanel.basicQos(1)），保证发挥所有服务器的性能。

2、秒杀流程

见秒杀流程图

3、限流

参照 Alibaba Sentinel