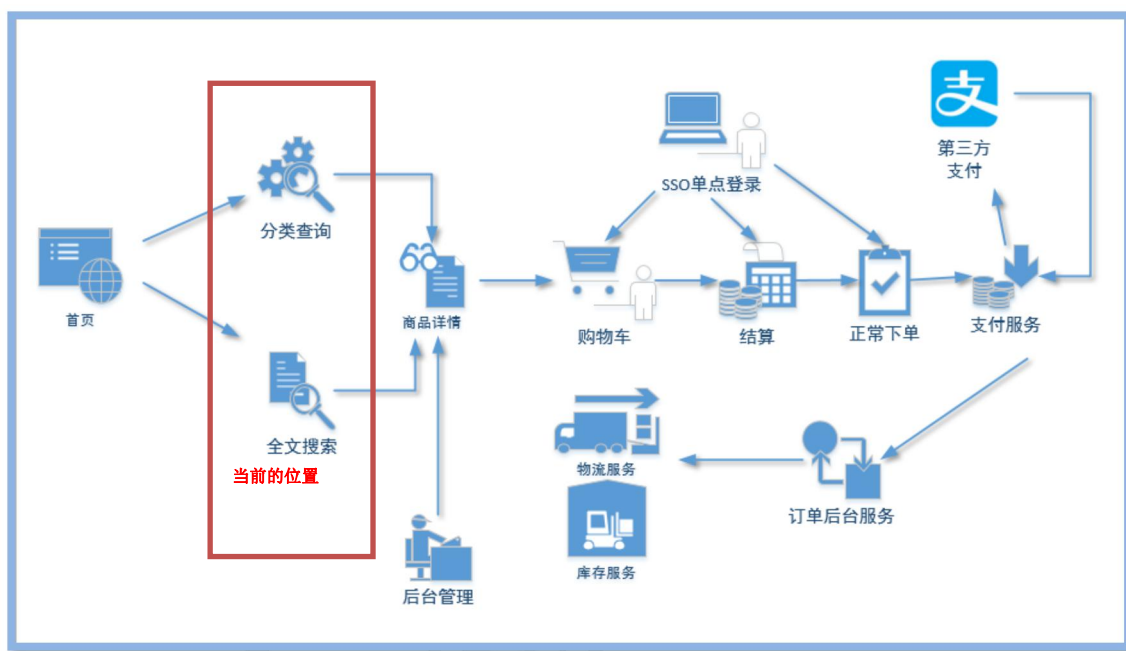


全文搜索

版本: V 1.0

www.atguigu.com



一、搜索

什么是搜索，计算机根据用户输入的关键词进行匹配，从已有的数据库中摘录出相关的记录反馈给用户。

常见的全网搜索引擎，像百度、谷歌这样的。但是除此以外，搜索技术在垂直领域也有广泛的使用，比如淘宝、京东搜索商品，万芳、知网搜索期刊，csdn 中搜索问题贴。也都是基于海量数据的搜索。

1 如何处理搜索

1.1 用传统关系性数据库



弊端：

- 1、对于传统的关系性数据库对于关键词的查询，只能逐字逐行的匹配，性能非常差。
- 2、匹配方式不合理，比如搜索“小密手机”，如果用 like 进行匹配，根本匹配不到。但是考虑使用者的用户体验的话，除了完全匹配的记录，还应该显示一部分近似匹配的记录，至少应该匹配到“手机”。

1.2 专业全文索引是怎么处理的

全文搜索引擎目前主流的索引技术就是倒排索引的方式。

传统的保存数据的方式都是

记录→单词

而倒排索引的保存数据的方式是

单词→记录

例如

搜索“红海行动”

但是数据库中保存的数据如图：

id	标题
1	红海行动影评
2	红海事件始末
3	湄公河行动导演
4

那么搜索引擎是如何能将两者匹配上的呢？

基于分词技术构建倒排索引：

首先每个记录保存数据时，都不会直接存入数据库。系统先会对数据进行分词，然后以倒排索引结构保存。如下：

分词	ids
红海	1,2
行动	1,3
影评	1
事件	2
始末	2
湄公河	3
导演	3

然后等到用户搜索的时候，会把搜索的关键词也进行分词，会把“红海行动”分词分成：红海和行动两个词。

这样的话，先用红海进行匹配，得到 id=1 和 id=2 的记录编号，再用行动匹配可以迅速

定位 id 为 1,3 的记录。

那么全文索引通常，还会根据匹配程度进行打分，显然 1 号记录能匹配的次数更多。所以显示的时候以评分进行排序的话，1 号记录会排到最前面。而 2、3 号记录也可以匹配到。

二 全文检索工具 elasticsearch

1 lucene 与 elasticsearch

咱们之前讲的处理分词，构建倒排索引，等等，都是这个叫 lucene 的做的。那么能不能说这个 lucene 就是搜索引擎呢？

还不能。lucene 只是一个提供全文搜索功能类库的核心工具包，而真正使用它还需要一个完善的服务框架搭建起来的应用。

好比 lucene 是类似于 jdk，而搜索引擎软件就是 tomcat 的。

目前市面上流行的搜索引擎软件，主流的就两款，elasticsearch 和 solr，这两款都是基于 lucene 的搭建的，可以独立部署启动的搜索引擎服务软件。由于内核相同，所以两者除了服务器安装、部署、管理、集群以外，对于数据的操作，修改、添加、保存、查询等等都十分类似。就好像都是支持 sql 语言的两种数据库软件。只要学会其中一个另一个很容易上手。

从实际企业使用情况来看，elasticSearch 的市场份额逐步在取代 solr，国内百度、京东、新浪都是基于 elasticSearch 实现的搜索功能。国外就更多了 像维基百科、GitHub、Stack Overflow 等等也都是基于 ES 的

2 elasticSearch 的使用场景

- 1、为用户提供按关键字查询的全文搜索功能。
- 2、著名的 ELK 框架(ElasticSearch,Logstash,Kibana)，实现企业海量日志的处理分析的解决方案。大数据领域的重要一份子。

3 elasticSearch 的安装

详见《elasticSearch 的安装手册》

4 elasticsearch 的基本概念

cluster	整个 elasticsearch 默认就是集群状态，整个集群是一份完整、互备的数据。
node	集群中的一个节点，一般只一个进程就是一个 node
shard	分片，即使是一个节点中的数据也会通过 hash 算法，分成多个片存放，默认是 5 片。
index	相当于 rdbms 的 database，对于用户来说是一个逻辑数据库，虽然物理上会被分多个 shard 存放，也可能存放在多个 node 中。
type	类似于 rdbms 的 table，但是与其说像 table，其实更像面向对象中的 class，同一 Json 的格式的数据集合。
document	类似于 rdbms 的 row、面向对象里的 object
field	相当于字段、属性

5 利用 kibana 学习 elasticsearch restful api (DSL)

5.1 es 中保存的数据结构

```
public class Movie {  
    String id;  
    String name;  
}
```

5

更多 [Java](#) - [大数据](#) - [前端](#) - [python](#) 人工智能资料下载，可访问百度：[尚硅谷官网](#)

```
Double doubanScore;
List<Actor> actorList;
}

public class Actor{
    String id;
    String name;
}
```

这两个对象如果放在关系型数据库保存，会被拆成 2 张表，但是 elasticsearch 是用一个 json 来表示一个 document。

所以他保存到 es 中应该是：

```
{
  "id": "1",
  "name": "operation red sea",
  "doubanScore": "8.5",
  "actorList": [
    {"id": "1", "name": "zhangyi"},
    {"id": "2", "name": "haiqing"},
    {"id": "3", "name": "zhanghanyu"}
  ]
}
```

5.2 对数据的操作

5.2.1 查看 es 中有哪些索引

```
GET /_cat/indices?v
```

es 中会默认存在一个名为.kibana 的索引

表头的含义

health	green(集群完整) yellow(单点正常、集群不完整) red(单点不正常)
status	是否能使用
index	索引名
uuid	索引统一编号
pri	主节点几个
rep	从节点几个

docs.count	文档数
docs.deleted	文档被删了多少
store.size	整体占空间大小
pri.store.size	主节点占

5.2.2 增加一个索引

```
PUT /movie_index
```

5.2.3 删除一个索引

ES 是不删除也不修改任何数据

```
DELETE /movie_index
```

5.2.4 新增文档

1、格式 PUT /index/type/id

```
PUT /movie_index/movie/1
{
  "id":1,
  "name":"operation red sea",
  "doubanScore":8.5,
  "actorList":[
    {"id":1,"name":"zhang yi"},
    {"id":2,"name":"hai qing"},
    {"id":3,"name":"zhang han yu"}
  ]
}
PUT /movie_index/movie/2
{
  "id":2,
  "name":"operation meigong river",
  "doubanScore":8.0,
  "actorList":[
    {"id":3,"name":"zhang han yu"}
  ]
}
PUT /movie_index/movie/3
{
```

```
{
  "id":3,
  "name":"incident red sea",
  "doubanScore":5.0,
  "actorList":[
    {"id":4,"name":"zhang chen"}
  ]
}
```

如果之前没建过 index 或者 type，es 会自动创建。

5.2.5 直接用 id 查找

```
GET movie_index/movie/1
```

5.2.6 修改—整体替换

和新增没有区别

```
PUT /movie_index/movie/3
{
  "id":"3",
  "name":"incident red sea",
  "doubanScore":"5.0",
  "actorList":[
    {"id":"1","name":"zhang chen"}
  ]
}
```

5.2.7 修改—某个字段

```
POST movie_index/movie/3/_update
{
  "doc": {
    "doubanScore":"7.0"
  }
}
```


5.2.8 删除一个 document

```
DELETE movie_index/movie/3
```

5.2.9 搜索 type 全部数据

```
GET movie_index/movie/_search
```

结果

```
{
  "took": 2,    //耗时时间 毫秒
  "timed_out": false, //是否超时
  "_shards": {
    "total": 5,    //发送给全部 5 个分片
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 3,    //命中 3 条数据
    "max_score": 1,    //最大评分
    "hits": [    // 结果
      {
        "_index": "movie_index",
        "_type": "movie",
        "_id": 2,
        "_score": 1,
        "_source": {
          "id": "2",
          "name": "operation meigong river",
          "doubanScore": 8.0,
          "actorList": [
            {
              "id": "1",
              "name": "zhang han yu"
            }
          ]
        }
      },
      .....
      .....
    ]
  }
}
```

5.2.10 按条件查询(全部)

```
GET movie_index/movie/_search
{
  "query":{
    "match_all": {}
  }
}
```

5.2.11 按分词查询

```
GET movie_index/movie/_search
{
  "query":{
    "match": {"name":"red"}
  }
}
```

注意结果的评分

5.2.12 按分词子属性查询

```
GET movie_index/movie/_search
{
  "query":{
    "match": {"actorList.name":"zhang"}
  }
}
```

5.2.13 match phrase

```
GET movie_index/movie/_search
{
  "query":{
    "match_phrase": {"name":"operation red"}
  }
}
```

按短语查询，不再利用分词技术，直接用短语在原始数据中匹配

5.2.14 fuzzy 查询

```
GET movie_index/movie/_search
{
  "query":{
    "fuzzy":{"name":"rad"}
  }
}
```

校正匹配分词，当一个单词都无法准确匹配，es 通过一种算法对非常接近的单词也给与一定的评分，能够查询出来，但是消耗更多的性能。

5.2.15 过滤--查询后过滤

```
GET movie_index/movie/_search
{
  "query":{
    "match":{"name":"red"}
  },
  "post_filter":{
    "term":{
      "actorList.id": 3
    }
  }
}
```

5.2.16 过滤--查询前过滤（推荐）

```
GET movie_index/movie/_search
{
  "query":{
    "bool":{
      "filter":[ {"term":{ "actorList.id": "1" }},
                  {"term":{ "actorList.id": "3" }}
    ],
    "must":{"match":{"name":"red"}}
  }
}
```

5.2.17 过滤--按范围过滤

```
GET movie_index/movie/_search
{
  "query": {
    "bool": {
      "filter": {
        "range": {
          "doubanScore": {"gte": 8}
        }
      }
    }
  }
}
```

关于范围操作符:

gt	大于
lt	小于
gte	大于等于
lte	小于等于

5.2.18 排序

```
GET movie_index/movie/_search
{
  "query": {
    "match": {"name": "red sea"}
  },
  "sort": [
    {
      "doubanScore": {
        "order": "desc"
      }
    }
  ]
}
```

5.2.19 分页查询

```
GET movie_index/movie/_search
{
  "query": { "match_all": {} },
  "from": 1,
  "size": 1
}
```

5.2.20 指定查询的字段

```
GET movie_index/movie/_search
{
  "query": { "match_all": {} },
  "_source": ["name", "doubanScore"]
}
```

5.2.21 高亮

```
GET movie_index/movie/_search
{
  "query": {
    "match": { "name": "red sea" }
  },
  "highlight": {
    "fields": { "name": {} }
  }
}
```

5.2.22 聚合

取出每个演员共参演了多少部电影

```
GET movie_index/movie/_search
{
  "aggs": {
    "groupby_actor": {
      "terms": {
        "field": "actorList.name.keyword"
      }
    }
  }
}
```

```
}  
}  
}  
}
```

每个演员参演电影的平均分是多少，并按评分排序

```
GET movie_index/movie/_search  
{  
  "aggs": {  
    "groupby_actor_id": {  
      "terms": {  
        "field": "actorList.name.keyword",  
        "order": {  
          "avg_score": "desc"  
        }  
      },  
      "aggs": {  
        "avg_score": {  
          "avg": {  
            "field": "doubanScore"  
          }  
        }  
      }  
    }  
  }  
}
```

5.3 关于 mapping

之前说 type 可以理解为 table，那每个字段的数据类型是如何定义的呢

查看 mapping

```
GET movie_index/_mapping/movie
```

实际上每个 type 中的字段是什么数据类型，由 mapping 定义。

但是如果没有设定 mapping 系统会自动，根据一条数据的格式来推断出应该的数据格式。

- true/false → boolean
- 1020 → long
- 20.1 → double

- “2018-02-01” → date
- “hello world” → text + keyword

默认只有 text 会进行分词，keyword 是不会分词的字符串。

mapping 除了自动定义，还可以手动定义，但是只能对新加的、没有数据的字段进行定义。一旦有了数据就无法再做修改了。

注意：虽然每个 Field 的数据放在不同的 type 下，但是同一个名字的 Field 在一个 index 下只能有一种 mapping 定义。

5.4 中文分词

elasticsearch本身自带的中文分词，就是单纯把中文一个字一个字的分开，根本没有词汇的概念。但是实际应用中，用户都是以词汇为条件，进行查询匹配的，如果能够把文章以词汇为单位切分开，那么与用户的查询条件能够更贴切的匹配上，查询速度也更加快速。

分词器下载网址：<https://github.com/medcl/elasticsearch-analysis-ik>

5.4.1 安装

下载好的 zip 包，请解压后放到 /usr/share/elasticsearch/plugins/ik

```
总用量 0
drwxrwxrwx. 4 root root 223 3月 18 01:12 ik
[root@jack plugins]# pwd
/usr/share/elasticsearch/plugins
[root@jack plugins]#
```

然后重启 es

5.4.2 测试使用

使用默认

```
GET movie_index/_analyze
{
```

```
"text": "我是中国人"
}
```

请观察结果

使用分词器

```
GET movie_index/_analyze
{
  "analyzer": "ik_smart",
  "text": "我是中国人"
}
```

请观察结果

另外一个分词器

ik_max_word

```
GET movie_index/_analyze
{
  "analyzer": "ik_max_word",
  "text": "我是中国人"
}
```

请观察结果

能够看出不同的分词器，分词有明显的区别，所以以后定义一个 **type** 不能再使用默认的 **mapping** 了，要手工建立 **mapping**，因为要选择分词器。

5.4.3 基于中文分词搭建索引

1、建立 mapping

```
PUT movie_chn
{
  "mappings": {
    "movie": {
      "properties": {
        "id": {
          "type": "long"
        }
      }
    }
  }
}
```



```
    },
    "name":{
      "type": "text"
      , "analyzer": "ik_smart"
    },
    "doubanScore":{
      "type": "double"
    },
    "actorList":{
      "properties": {
        "id":{
          "type":"long"
        },
        "name":{
          "type":"keyword"
        }
      }
    }
  }
}
```

插入数据

```
PUT /movie_chn/movie/1
{ "id":1,
  "name":"红海行动",
  "doubanScore":8.5,
  "actorList":[
    {"id":1,"name":"张译"},
    {"id":2,"name":"海清"},
    {"id":3,"name":"张涵予"}
  ]
}

PUT /movie_chn/movie/2
{
  "id":2,
  "name":"湄公河行动",
  "doubanScore":8.0,
  "actorList":[
    {"id":3,"name":"张涵予"}
  ]
}

PUT /movie_chn/movie/3
{
  "id":3,
  "name":"红海事件",
  "doubanScore":5.0,
  "actorList":[
    {"id":4,"name":"张晨"}
  ]
}
```

```
}  
}
```

查询测试

```
GET /movie_chn/movie/_search  
{  
  "query": {  
    "match": {  
      "name": "红海战役"  
    }  
  }  
}  
  
GET /movie_chn/movie/_search  
{  
  "query": {  
    "term": {  
      "actorList.name": "张译"  
    }  
  }  
}
```

5.4.4 自定义词库

修改/usr/share/elasticsearch/plugins/ik/config/中的 IKAnalyzer.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">  
<properties>  
  <comment>IK Analyzer 扩展配置</comment>  
  <!--用户可以在这里配置自己的扩展字典 -->  
  <entry key="ext_dict"></entry>  
  <!--用户可以在这里配置自己的扩展停止词字典-->  
  <entry key="ext_stopwords"></entry>  
  <!--用户可以在这里配置远程扩展字典 -->  
  <entry key="remote_ext_dict">http://192.168.67.163/fenci/myword.txt</entry>  
  <!--用户可以在这里配置远程扩展停止词字典-->  
  <!-- <entry key="remote_ext_stopwords">words_location</entry> -->  
</properties>
```

按照标红的路径利用 nginx 发布静态资源

在 nginx.conf 中配置

```
server {  
    listen 80;  
    server_name 192.168.67.163;  
    location /fenci/ {  
        root es;  
    }  
}
```

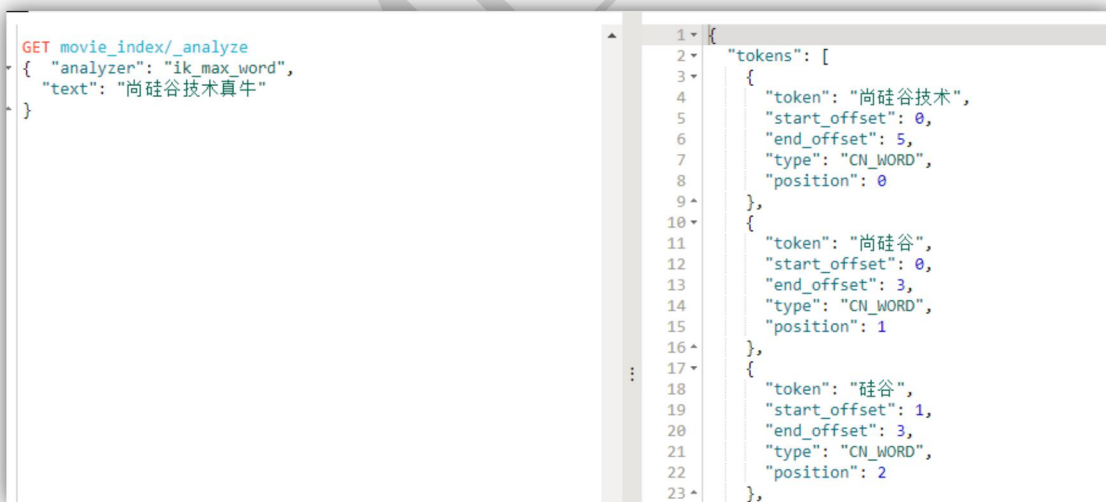
并且在 /usr/local/nginx/ 下建 /es/fenci/ 目录，目录下加 myword.txt

myword.txt 中编写关键词，每一行代表一个词。



然后重启 es 服务器，重启 nginx。

在 kibana 中测试分词效果

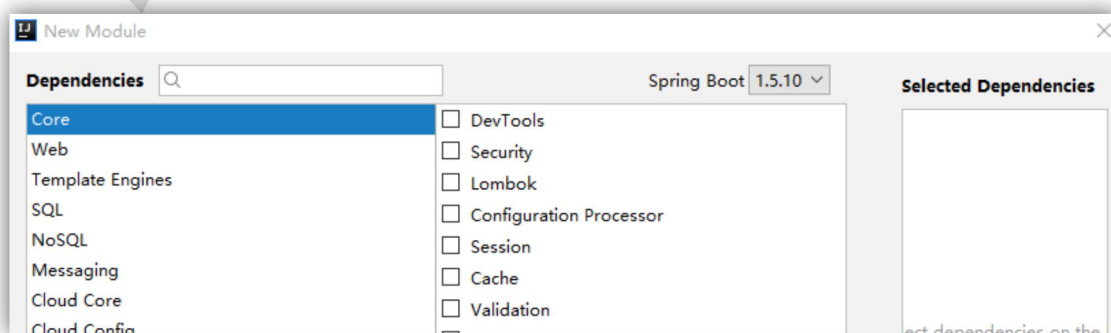
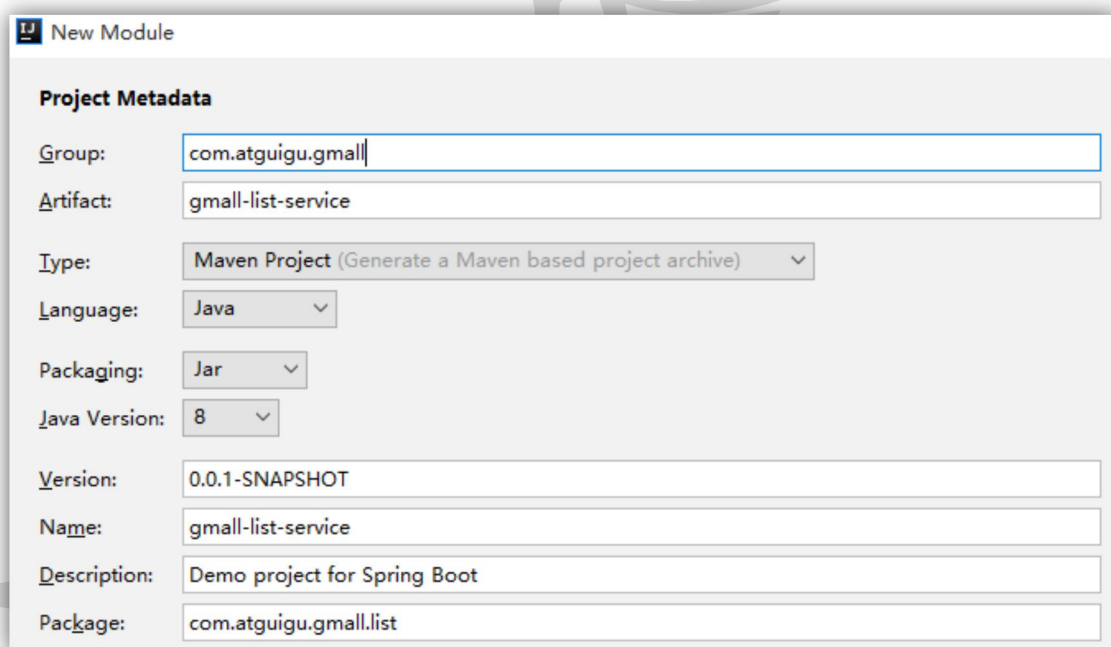


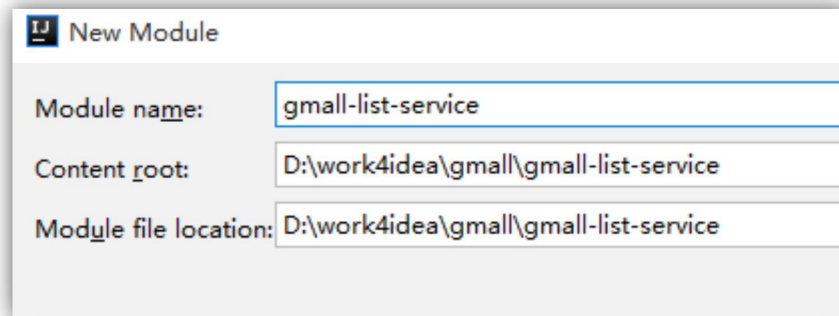
更新完成后，es 只会对新增的数据用新词分词。历史数据是不会重新分词的。如果想要历史数据重新分词。需要执行：

```
POST movies_index_chn/_update_by_query?conflicts=proceed
```

三 Java 程序中的应用

1 、搭建模块





pom.xml

```
<?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">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.atguigu.gmall</groupId>
  <artifactId>gmall-list-service</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>gmall-list-service</name>
  <description>Demo project for Spring Boot</description>

  <parent>
    <groupId>com.atguigu.gmall</groupId>
    <artifactId>gmall-parent</artifactId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <dependencies>

    <dependency>
      <groupId>com.atguigu.gmall</groupId>
      <artifactId>gmall-interface</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>

    <dependency>
      <groupId>com.atguigu.gmall</groupId>
      <artifactId>gmall-service-util</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>

  </dependencies>

  <build>
    <plugins>
```

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>

</project>
```

elasticsearch 的 版本号，请提取到 gmall-parent 中

2、关于 es 的 java 客户端的选择

目前市面上有两类客户端

一类是 `TransportClient` 为代表的 ES 原生客户端，不能执行原生 dsl 语句必须使用它的 Java api 方法。

另外一种是以 Rest Api 为主的 `RestClient`，最典型的的就是 `Jest`。这种客户端可以直接使用 dsl 语句拼成的字符串，直接传给服务端，然后返回 json 字符串再解析。

两种方式各有优劣，但是最近 elasticsearch 官网，宣布计划在 7.0 以后的版本中废除 `TransportClient`。以 `RestClient` 为主。



WARNING

We plan on deprecating the `TransportClient` in Elasticsearch 7.0 and removing it completely in 8.0. Instead, you should be using the [Java High Level REST Client](#), which executes HTTP requests rather than serialized Java requests. The [migration guide](#) describes all the steps needed to migrate.

所以在官方的 `RestClient` 基础上，进行了简单包装的 `Jest` 客户端，就成了首选，而且该客户端也与 `springboot` 完美集成。

3、导入 Jest 依赖

```
<dependency>
```

```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-elasticsearch</artifactId>
</dependency>

<!-- https://mvnrepository.com/artifact/io.searchbox/jest -->
<dependency>
  <groupId>io.searchbox</groupId>
  <artifactId>jest</artifactId>
  <version>5.3.3</version>
</dependency>

<!-- https://mvnrepository.com/artifact/net.java.dev.jna/jna -->
<dependency>
  <groupId>net.java.dev.jna</groupId>
  <artifactId>jna</artifactId>
  <version>4.5.1</version>
</dependency>
```

其中 `jest` 和 `jna` 请将版本号，部分纳入 `gmall-parent` 中管理。
`spring-boot-starter-data-elasticsearch` 不用管理版本号，其版本跟随 `springboot` 的 1.5.10 大版本号。

4 、在测试类中测试 ES

`application.properties` 中加入

```
spring.elasticsearch.jest.uris=http://192.168.67.163:9200
```

测试类

```
@Autowired
JestClient jestClient;

@Test
public void testEs() throws IOException {
    String query="{\n" +
        "  \"query\": {\n" +
        "    \"match\": {\n" +
        "      \"actorList.name\": \"张译\"\n" +
        "    }\n" +
        "  }\n" +
        "}";
    Search search = new Search.Builder(query).addIndex("movie_chn").addType("movie").build();
    SearchResult result = jestClient.execute(search);
}
```

```
List<SearchResult.Hit<HashMap, Void>> hits = result.getHits(HashMap.class);

for (SearchResult.Hit<HashMap, Void> hit : hits) {
    HashMap source = hit.source;
    System.err.println("source = " + source);
}
}
```

打印结果：

```
source = {doubanScore=8.5, es_metadata_id=1, name=红海行动, actorList=[{id=1.0, name=张译}, {id=2.0, na
```

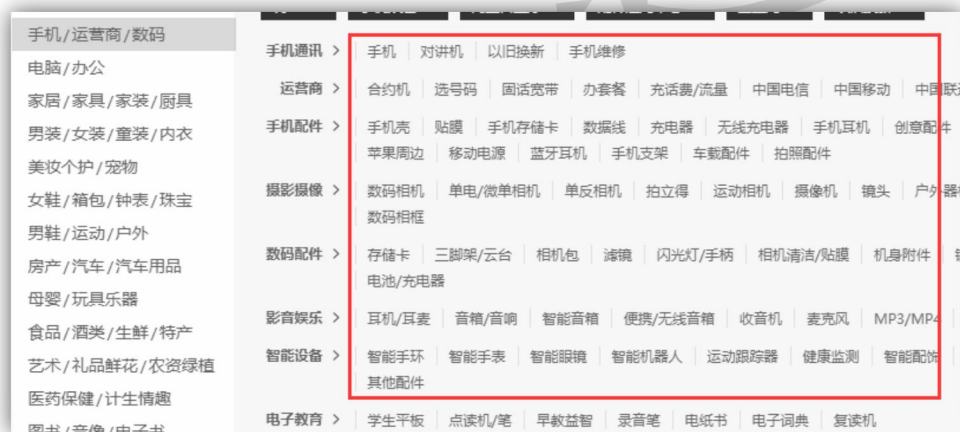
以上技术方面的准备就做好了。下面回到咱们电商的业务

三、利用 elasticSearch 开发电商的搜索列表功能

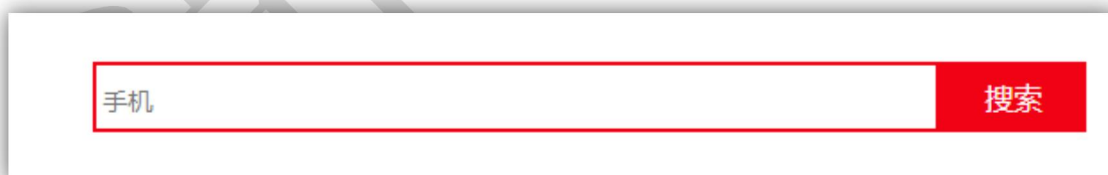
1、功能简介

1.1 入口： 两个

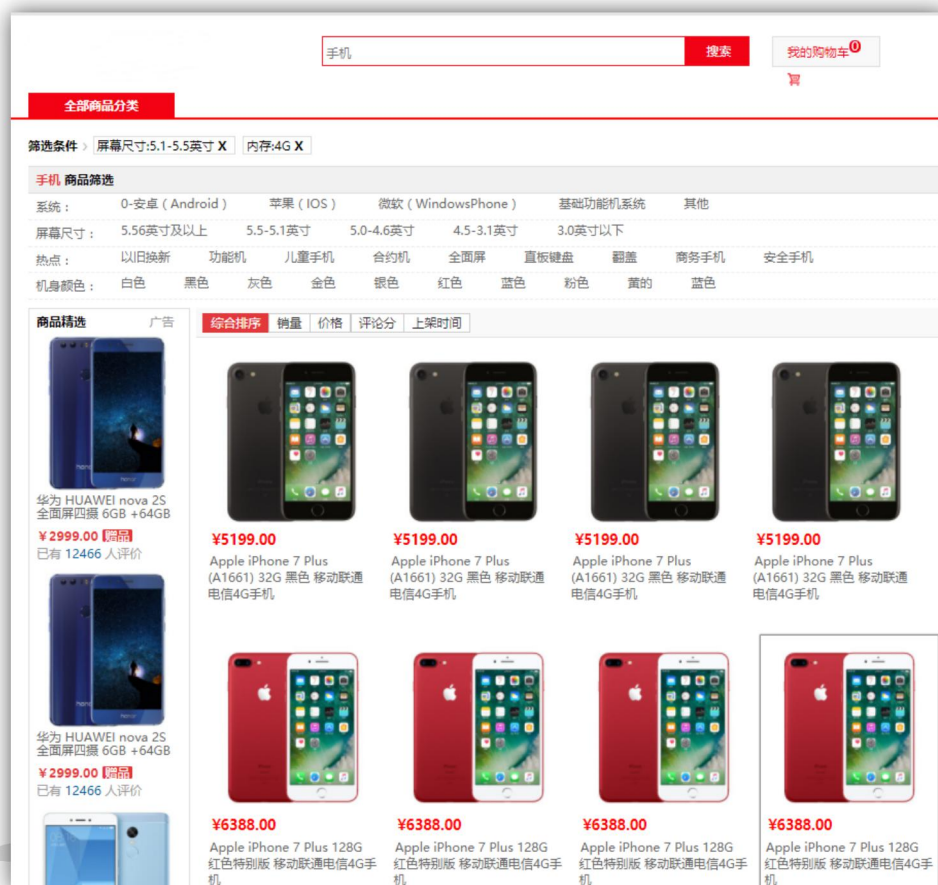
首页的分类



搜索栏



列表展示页面



2 根据业务搭建数据结构

这时我们要思考三个问题：

- 1、哪些字段需要分词
- 2、我们用哪些字段进行过滤
- 3、哪些字段我们需要通过搜索显示出来。

需要分词的字段	sku 名称 sku 描述	分词、定义分词器
有可能用于过滤的字段	平台属性、三级分类、价格	要索引
其他需要显示的字段	skuld 图片路径	不索引

根据以上制定出如下结构：

```
PUT gmall
{
  "mappings": {
    "SkuInfo": {
      "properties": {
        "id": {
          "type": "keyword",
          "index": false
        },
        "price": {
          "type": "double"
        },
        "skuName": {
          "type": "text",
          "analyzer": "ik_max_word"
        },
        "skuDesc": {
          "type": "text",
          "analyzer": "ik_smart"
        },
        "catalog3Id": {
          "type": "keyword"
        },
        "skuDefaultImg": {
          "type": "keyword",
          "index": false
        },
        "skuAttrValueList": {
          "properties": {
            "valueId": {
              "type": "keyword"
            }
          }
        }
      }
    }
  }
}
```

```
}
```

3 sku 数据保存到 ES

思路:

回顾一下, es 数据保存的 dsl

```
PUT /movie_index/movie/1
{ "id":1,
  "name":"operation red sea",
  "doubanScore":8.5,
  "actorList":[
    {"id":1,"name":"zhang yi"},
    {"id":2,"name":"hai qing"},
    {"id":3,"name":"zhang han yu"}
  ]
}
```

es 存储数据是以 json 格式保存的, 那么如果一个 javaBean 的结构刚好跟要求的 json 格式吻合, 我们就可以直接把 javaBean 序列化为 json 保持到 es 中, 所以我们要制作一个与 es 中 json 格式一致的 javaBean.

3.1 JavaBean

skuLsInfo

```
public class SkuLsInfo implements Serializable {

    String id;

    BigDecimal price;

    String skuName;

    String skuDesc;

    String catalog3Id;
```

```
String skuDefaultImg;

Long hotScore;

List<SkuLsAttrValue> skuAttrValueList;
}
```

SkuLsAttrValue

```
public class SkuLsAttrValue implements Serializable {

    String valueId;
}
```

3.2 保存 sku 数据的业务实现类

在 gmall-list-service 模块中增加业务实现类 listServiceImpl

```
public static final String index_name_gmall="gmall";

public static final String type_name_gmall="SkuInfo";

public void saveSkuInfo(SkuLsInfo skuLsInfo){
    Index index=new
    Index.Builder(skuLsInfo).index(index_name_gmall).type(type_name_gmall).id(skuLsInfo.getId()).
    build();
    try {
        jestClient.execute(index);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

自行添加 gmall-inteface 中增加接口方法

3.3 在后台管理的 sku 保存中，调用该方法

```
private void sendSkuToList(SkuInfo skuInfo){

    SkuLsInfo skuLsInfo=new SkuLsInfo();

    try {
        BeanUtils.copyProperties(skuLsInfo, skuInfo);
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (InvocationTargetException e) {
        e.printStackTrace();
    }
    listService.saveSkuInfo(skuLsInfo);
}

public void saveSkuInfo(SkuInfo skuInfo){

.....

sendSkuToList( skuInfo);    //在保存 sku 最后保存到 es
}
```

这时在界面中保存 sku 是可以，自动向 es 发送数据的。

4 查询数据的后台方法

4.1 分析

首先先观察功能页面，咱们一共要用什么查询条件，查询出什么数据？

查询条件：

- 1、关键字
- 2、可以通过分类进入列表页面

3、属性值

4、分页页码

查询结果：

- 1 sku 的列表(关键字高亮显示)
- 2 这些 sku 涉及了哪些属性和属性值
- 3 命中个数，用于分页

基于以上

4.2 编写 DSL 语句：

```
GET mall/SkuInfo/_search
{
  "query": {
    "bool": {
      "filter": [
        {"terms": {"skuAttrValueList.valueld": ["46","45"]}},
        {"term": {"catalog3ld": "61"}}
      ],
      "must": [
        {"match": {"skuName": "128"} }
      ]
    }
  },
  "highlight": {
    "fields": {"skuName": {}}
  },
  "from": 3,
  "size": 1,
  "sort": {"hotScore": {"order": "desc"}},
  "aggs": {
    "groupby_attr": {
      "terms": {
        "field": "skuAttrValueList.valueld"
      }
    }
  }
}
```

4.3 制作传入参数的类

```
public class SkuLsParams implements Serializable {
```

31

```
String keyword;

String catalog3Id;

String[] valuelid;

int pageNo=1;

int pageSize=20;

}
```

4.4 返回结果的类

```
public class SkuLsResult implements Serializable {

    List<SkuLsInfo> skuLsInfoList;

    long total;

    long totalPages;

    List<String> attrValuelidList;

}
```

4.5 基于这个 DSL 查询编写 Java 代码

```
public SkuLsResult search(SkuLsParams skuLsParams){

    String query=makeQueryStringForSearch(skuLsParams);

    Search search=
    Search.Builder(query).addIndex(index_name_gmall).addType(type_name_gmall).build(); new
    SearchResult searchResult=null;
    try {
        searchResult = jestClient.execute(search);
    } catch (IOException e) {
        e.printStackTrace();
    }

    SkuLsResult skuLsResult = makeResultForSearch(skuLsParams, searchResult);

    return skuLsResult;

}
```


4.5.1 构造查询 DSL

查询的过程很简单，但是要构造查询的 `query` 这个字符串有点麻烦，主要是这个 `Json` 串中的数据都是动态的。要拼接这个字符串，需要各种循环判断，处理标点符号等等。操作麻烦，可读性差。

但是 `jest` 这个客户端包，提供了一组 `builder` 工具。这个工具可以比较方便的帮程序员组合复杂的查询 `Json`。

```
private String makeQueryStringForSearch(SkuLsParams skuLsParams){
    SearchSourceBuilder searchSourceBuilder=new SearchSourceBuilder();

    BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();
    if(skuLsParams.getKeyword()!=null){
        MatchQueryBuilder matchQueryBuilder=new
MatchQueryBuilder("skuName",skuLsParams.getKeyword());
boolQueryBuilder.must(matchQueryBuilder);

        HighlightBuilder highlightBuilder=new HighlightBuilder();
        highlightBuilder.field("skuName");
        highlightBuilder.preTags("<span style='color:red'>");
        highlightBuilder.postTags("</span>");
        searchSourceBuilder.highlight(highlightBuilder);

        TermsBuilder groupby_attr =
AggregationBuilders.terms("groupby_attr").field("skuAttrValueList.valuelid");
searchSourceBuilder.aggregation(groupby_attr);
    }
    if(skuLsParams.getCatalog3Id()!=null){
        QueryBuilder termQueryBuilder=new
TermQueryBuilder("catalog3Id",skuLsParams.getCatalog3Id());
boolQueryBuilder.filter(termQueryBuilder);
    }
    if(skuLsParams.getValuelid()!=null&&skuLsParams.getValuelid().length>=0){
        for (int i = 0; i < skuLsParams.getValuelid().length; i++) {
            String valuelid = skuLsParams.getValuelid()[i];
            QueryBuilder termQueryBuilder=new
TermsQueryBuilder("skuAttrValueList.valuelid",valuelid);
boolQueryBuilder.filter(termQueryBuilder);
        }
    }
    searchSourceBuilder.query(boolQueryBuilder);
}
```

```
int from =(skuLsParams.getPageNo()-1)*skuLsParams.getPageSize();
searchSourceBuilder.from(from);
searchSourceBuilder.size(skuLsParams.getPageSize());

searchSourceBuilder.sort("hotScore",SortOrder.DESC);

String query = searchSourceBuilder.toString();

System.err.println("query = " + query);
return query;
}
```

4.5.2 处理返回值

思路：所有的返回值其实都在这个 `searchResult` 中

```
searchResult = jestClient.execute(search);
```

它的结构其实可以在 kibana 中观察一下：

命中的结果



```
1 {
2   "took": 247,
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": 2,
12    "max_score": null,
13    "hits": [
14      {
15        "_index": "gmall",
16        "_type": "SkuInfo",
17        "_id": "14",
18        "_score": null,
19        "_source": {
20          "id": "14",
21          "spuId": "24",
22          "price": 2888,
23          "skuName": "小米6 全网通 64G 亮蓝色 移动联通电信4G手机 双卡双待",
24          "skuDesc": "小米6 全网通 64G 亮蓝色 移动联通电信4G手机 双卡双待"
```

高亮显示

```
    "highlight": {  
      "skuName": [  
        "<span style='color:red'>小米</span>6  
全网通 6GB+128GB 亮蓝色 移动联通电信4G手机  
双卡双待"  
      ]  
    }
```

分组统计结果:

```
"aggregations": {  
  "groupby_attr": {  
    "doc_count_error_upper_bound": 0,  
    "sum_other_doc_count": 0,  
    "buckets": [  
      {  
        "key": "41",  
        "doc_count": 3  
      },  
      {  
        "key": "46",  
        "doc_count": 2  
      },  
      {  
        "key": "45",  
        "doc_count": 1  
      }  
    ]  
  }  
}
```

针对这三个部分来解析 searchResult

```
private SkuLsResult makeResultForSearch(SkuLsParams skuLsParams, SearchResult  
searchResult){  
    SkuLsResult skuLsResult=new SkuLsResult();  
    List<SkuLsInfo> skuLsInfoList=new ArrayList<>(skuLsParams.getPageSize());  
  
    //获取 sku 列表  
    List<SearchResult.Hit<SkuLsInfo, Void>> hits =  
searchResult.getHits(SkuLsInfo.class);  
    for (SearchResult.Hit<SkuLsInfo, Void> hit : hits) {  
        SkuLsInfo skuLsInfo = hit.source;  
        if(hit.highlight!=null&&hit.highlight.size()>0){  
            List<String> list = hit.highlight.get("skuName");  
            //把带有高亮标签的字符串替换 skuName  
            String skuNameHl = list.get(0);  
            skuLsInfo.setSkuName(skuNameHl);  
        }  
        skuLsInfoList.add(skuLsInfo);  
    }  
}
```

```
}
skuLsResult.setSkuLsInfoList(skuLsInfoList);
skuLsResult.setTotal(searchResult.getTotal());

//取记录个数并计算出总页数
long totalPage= (searchResult.getTotal() + skuLsParams.getPageSize() -1) /
skuLsParams.getPageSize();
skuLsResult.setTotalPages( totalPage);

//取出涉及的属性值 id
List<String> attrValueIdList=new ArrayList<>();
MetricAggregation aggregations = searchResult.getAggregations();
TermsAggregation groupby_attr = aggregations.getTermsAggregation("groupby_attr");
if(groupby_attr!=null){
    List<TermsAggregation.Entry> buckets = groupby_attr.getBuckets();
    for (TermsAggregation.Entry bucket : buckets) {
        attrValueIdList.add( bucket.getKey() );
    }
    skuLsResult.setAttrValueIdList(attrValueIdList);
}

return skuLsResult;
}
```

测试后台程序...

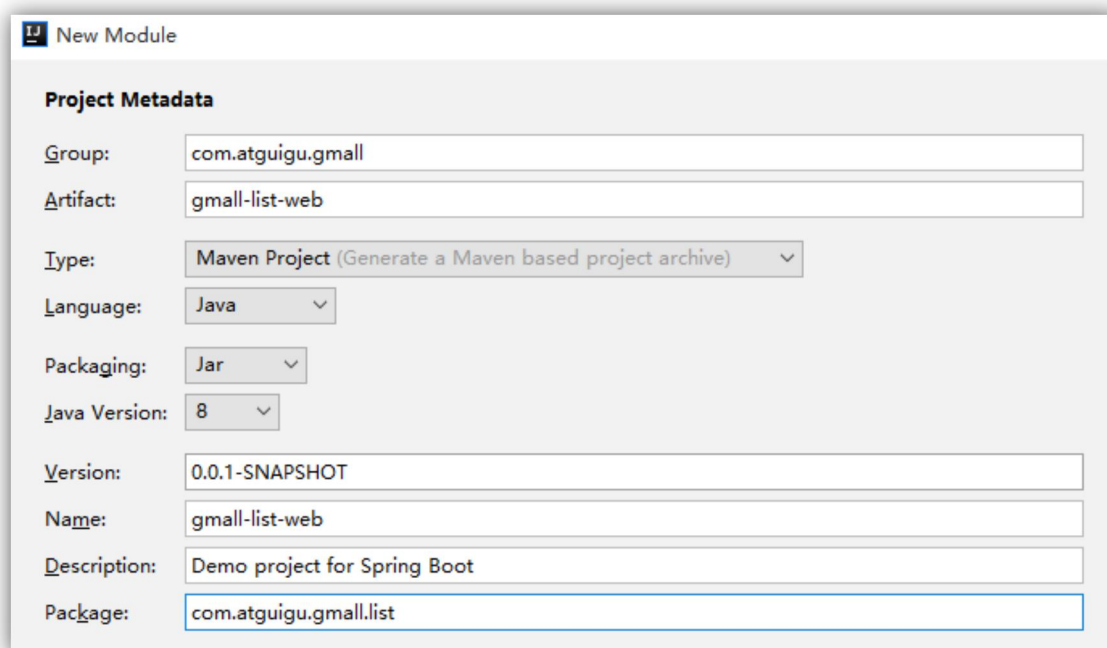
完成了 sku 列表数据获取的方法。回到页面功能上来。

5 检索的页面

检索功能

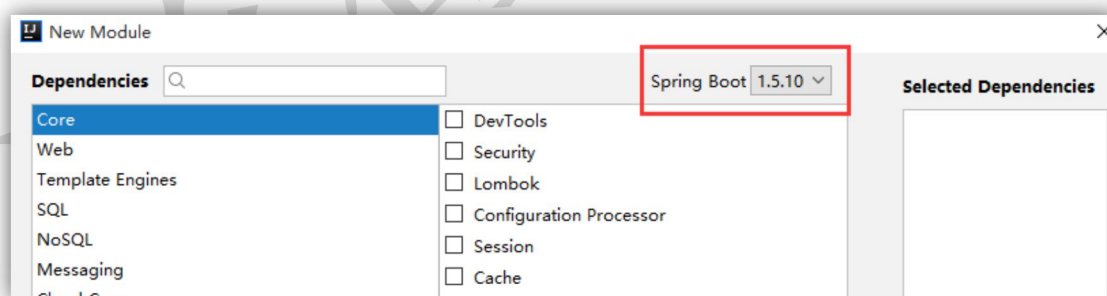


5.1 创建 gmall-list-web 模块



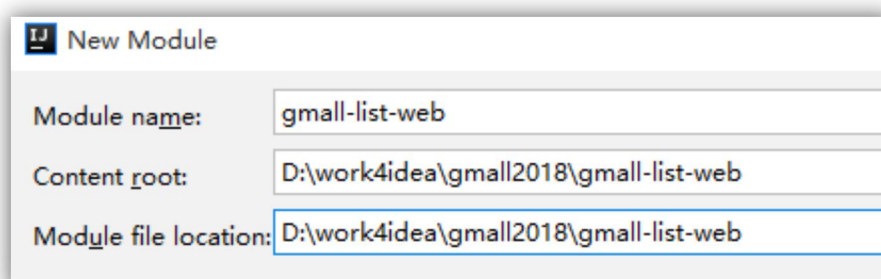
The 'New Module' dialog box in IntelliJ IDEA. The 'Project Metadata' section is filled out with the following values:

- Group: com.atguigu.gmall
- Artifact: gmall-list-web
- Type: Maven Project (Generate a Maven based project archive)
- Language: Java
- Packaging: Jar
- Java Version: 8
- Version: 0.0.1-SNAPSHOT
- Name: gmall-list-web
- Description: Demo project for Spring Boot
- Package: com.atguigu.gmall.list



The 'New Module' dialog box showing the 'Dependencies' section. The 'Core' dependency is selected. The 'Spring Boot' version is set to 1.5.10. The 'Selected Dependencies' section is empty.

Dependencies	Selected Dependencies
<input checked="" type="checkbox"/> Core	
<input type="checkbox"/> Web	
<input type="checkbox"/> Template Engines	
<input type="checkbox"/> SQL	
<input type="checkbox"/> NoSQL	
<input type="checkbox"/> Messaging	
<input type="checkbox"/> Cloud Core	
<input type="checkbox"/> DevTools	
<input type="checkbox"/> Security	
<input type="checkbox"/> Lombok	
<input type="checkbox"/> Configuration Processor	
<input type="checkbox"/> Session	
<input type="checkbox"/> Cache	



The 'New Module' dialog box showing the 'Module name' section. The 'Module name' is gmall-list-web. The 'Content root' and 'Module file location' are both D:\work4idea\gmall2018\gmall-list-web.

Module name	Content root	Module file location
gmall-list-web	D:\work4idea\gmall2018\gmall-list-web	D:\work4idea\gmall2018\gmall-list-web

5.1.1 pom.xml

```
<parent>
  <groupId>com.atguigu.gmall</groupId>
  <artifactId>gmall-parent</artifactId>
  <version>1.0-SNAPSHOT</version>
</parent>

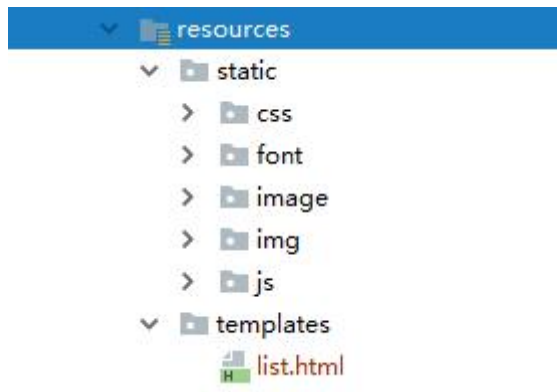
<dependencies>

<dependency>
  <groupId>com.atguigu.gmall</groupId>
  <artifactId>gmall-interface</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>

<dependency>
  <groupId>com.atguigu.gmall</groupId>
  <artifactId>gmall-web-util</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
</dependencies>
```

5.1.2 静态网页及资源文件

拷贝静态文件到 resources 目录下，手工建立 static 和 templates 目录



手工替换一下目录

把 list.html 的路径中 "../static/" 替换成 "/"

5.1.3 application.properties

```
server.port=8085

spring.thymeleaf.cache=false

spring.thymeleaf.mode=LEGACYHTML5
```

5.1.4 nginx 配置

host 文件

```
# gmall
192.168.67.163 item.gmall.com list.gmall.com manage.gmall.com resource.gmall.com
```

nginx.conf

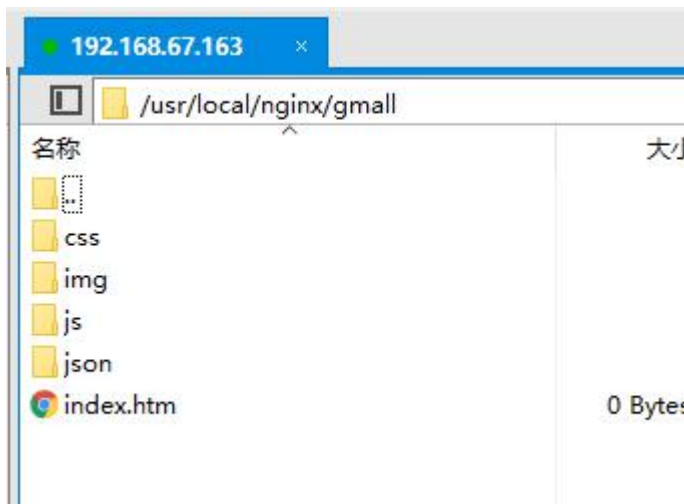
```
upstream list.gmall.com{
    server 192.168.67.1:8085;
}
server {
    listen 80;
    server_name list.gmall.com;
    location / {
        proxy_pass http:// list.gmall.com;
        proxy_set_header X-forwarded-for $proxy_add_x_forwarded_for;
```



```
}
}
```

5.1.5 放置 nginx 首页静态资源

用 xftp 拷贝首页资源到 nginx/gmall 目录下



修改 nginx.conf 配置文件

```
server {
    listen      80;
    server_name www.gmall.com;
    location / {
        root    gmall;
        index   index.html index.htm;
    }
}
```

host 文件

```
# gmall
192.168.67.163    item.gmall.com    list.gmall.com    manage.gmall.com    www.gmall.com
resource.gmall.com
```

5.2 sku 列表功能



首先是根据关键字、属性值、分类 Id、页码查询 sku 列表。

5.2.1 ListController

```
@RequestMapping("list.html")
public String getList( SkuLsParams skuLsParams, Model model){

    skuLsParams.setPageSize(4);
    //根据参数返回 sku 列表
    SkuLsResult skuLsResult = listService.search(skuLsParams);
    model.addAttribute("skuLsInfoList",skuLsResult.getSkuLsInfoList());
    return "list";
}
```

5.2.2 页面 html 渲染

```
<div style="width:215px" th:each="skuLsInfo:${skuLsInfoList}" >
    <p class="da">
        <a href="#"
            th:onclick="'javascript:item(\'+${skuLsInfo.id}+\')'">
```

```

        
    </a>
</p>

<p class="tab_R">
    <span th:text="'¥'+${#numbers.formatDecimal(skuLsInfo.price,1,2)}">¥5199.00</span>
</p>

<a href="#" title=""
th:onclick="'javascript:item(\''+${skuLsInfo.id}+'\'')'" class="tab_JE"
th:utext="${skuLsInfo.skuName}" >
    Apple iPhone 7 Plus (A1661) 32G 黑色 移动联通电信 4G 手机
</a>
</div>

```

要注意的是其中 skuName 中因为关键字标签所以必须要用 utext 否则标签会被转义。

5.2.3 搜索栏相关 html

```

<!-- 搜索导航 -->
<div class="header_sous">
    <div class="logo">
        <a href="#"></a>
    </div>
    <div class="header_form">
        <input id="keyword" name="keyword" type="text" placeholder="手机" />
        <a href="#" onclick="searchList()">搜索</a>
    </div>
    <div class="header_ico">
        <div class="header_gw">
            <span><a href="#">我的购物车</a></span>
            
            <span>0</span>
        </div>
        <div class="header_ko">
            <p>购物车中还没有商品，赶紧选购吧！</p>
        </div>
    </div>

```

```
</div>
</div>
```

5.2.4 js 代码

```
function searchList(){
    var keyword = $("#keyword").val();
    window.location.href="/list.html?keyword="+keyword;
}

function item(skuid) {
    window.location.href="http://item.gmall.com/"+skuid+".html";
}
```

这时可以看到列表效果了。

5.3 页面功能—提供可供选择的属性列表

5.3.1 思路:

这个列表有两种情况

- 1、如果是通过首页的 3 级分类点击进入的，要按照分类 id 查询对应的属性和属性值列表。
- 2、如果是直接用搜索栏输入文字进入的，要根据 sku 的查询结果涉及的属性值(好在我们已经通过 es 的聚合取出来了)，再去查询数据库把文字列表显示出来。

5.3.2 ListController

getList 方法中添加

```
// 根据查询的结果返回属性和属性值列表
List<BaseAttrInfo> attrList=null;
if(skuLsParams.getCatalog3Id()!=null){
    attrList =
    manageService.getAttrList(skuLsParams.getCatalog3Id());
}else {
    List<String> attrValueIdList = skuLsResult.getAttrValueIdList();
    if(attrValueIdList!=null&&attrValueIdList.size()>0){
        attrList = manageService.getAttrList(attrValueIdList);
    }
}
model.addAttribute("attrList",attrList);
```

其中 `manageService.getAttrList(String catalog3Id)` 这个方法我们已经有现成的了。
但是 `manageService.getAttrList(attrValueIdList)` 这个方法我们要新添加。

5.3.3 在 ManageServiceImpl 中增加方法

```
@Override
public List<BaseAttrInfo> getAttrList(List<String> attrValueIdList) {
    String attrValueIds = StringUtils.join(attrValueIdList.toArray(), ",");
    List<BaseAttrInfo> baseAttrInfoList
    = baseAttrInfoMapper.selectAttrInfoListByIds(attrValueIds);
    return baseAttrInfoList;
}
```

5.3.4 BaseAttrInfoMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.atguigu.gmall.manage.mapper.BaseAttrInfoMapper">
    <select id="selectAttrInfoList" parameterType="long" resultMap="attrInfoMap">
        SELECT ba.id,ba.attr_name,ba.catalog3_id,
        bv.id value_id ,bv.value_name, bv.attr_id FROM
        base_attr_info ba INNER JOIN base_attr_value bv ON ba.id =bv.attr_id
        where ba.catalog3_id=#{catalog3Id}
    </select>
    <resultMap id="attrInfoMap" type="com.atguigu.gmall.bean.BaseAttrInfo"
    autoMapping="true">
        <result property="id" column="id" ></result>
        <collection property="attrValueList" ofType="com.atguigu.gmall.bean.BaseAttrValue"
        autoMapping="true">
            <result property="id" column="value_id" ></result>
        </collection>
    </resultMap>

    <select id="selectAttrInfoListByIds" resultMap="attrInfoMap">
        SELECT ba.id,ba.attr_name,ba.catalog3_id,
        bv.id value_id ,bv.value_name, bv.attr_id FROM
        base_attr_info ba INNER JOIN base_attr_value bv ON ba.id =bv.attr_id
        where bv.id in (${attrValueIds})
    </select>
</mapper>
```

注意这里面没有用#{ }是因为 attrValueIds 是两个数字用逗号分开的，所以不能整体套上单引，所以使用\${ }。

5.3.5 BaseAttrInfoMapper.class

```
public interface BaseAttrInfoMapper extends Mapper<BaseAttrInfo> {

    public List<BaseAttrInfo> selectAttrInfoList(long catalog3Id);

    public List<BaseAttrInfo> selectAttrInfoListByIds(@Param("attrValueIds")
String attrValueIds);
}
```

此处必须要用@Param 注解否则\${ }无法识别。

5.3.6 点击属性值的链接

`getAttrList(List<String> attrValueIdList)`方法实现后，还有一个问题就是，点击属性时，要把上次查询的内容也带上，即带上历史参数。

```
private String makeUrlParam(String keyword,String catalog3Id, String[] valueIds,String
excludeValueId ){
    String url="";
    List<String> paramList=new ArrayList<>();
    if(keyword!=null&&keyword.length()>0){
        paramList.add("keyword="+keyword);
    }
    if(catalog3Id!=null){
        paramList.add("catalog3Id="+catalog3Id);
    }

    if(valueIds!=null) {
        for (int i = 0; i < valueIds.length; i++) {
            String valueId = valueIds[i];
            if (!excludeValueId.equals(valueId)) {
                paramList.add("valueId=" + valueId);
            }
        }
    }
    url = StringUtils.join(paramList, "&");
    return url;
}
```

`getList` 方法中增加

```
//历史参数
String[] valueIds=skuLsParams.getValueId();
String catalog3Id = skuLsParams.getCatalog3Id();
String keyword=skuLsParams.getKeyword();

String urlParam = makeUrlParam(keyword,catalog3Id, valueIds, "");
model.addAttribute("urlParam",urlParam);
```

java 部分的代码就完成了。

5.3.7 生成属性列表的 html 部分

```
<div class="GM_selector">
    <!--手机商品筛选-->
    <div class="title">
        <h3><em>商品筛选</em></h3>
```

```

</div>
<div class="GM_nav_logo">

    <div class="GM_pre" th:each="attrInfo:${attrList}">
        <div class="sl_key">
            <span th:text="${attrInfo.attrName}+'<\/span>属性: <\/div>
            <div class="sl_value">
                <ul>
                    <li th:each="attrValue:${attrInfo.attrValueList}"><a
th:href='"/list.html?'+${urlParam}+'&valueId='+${attrValue.id}"
th:text="${attrValue.valueName}">属性值<\/a><\/li>
                <\/ul>
            <\/div>
        <\/div>
    <\/div>
<\/div>

```

完成后



5.4 页面功能--面包屑



面包屑导航是为了能够让用户清楚的知道当前页面的所在位置和筛选条件的功能。但是这个小小的人性化功能却有点麻烦。

- 功能点：
- 1、点击某个属性值的时候对应的那行属性要消失掉不能再次选择。
 - 2、列在上面的属性面包屑，要可以取消掉，恢复到没选择之前。

5.4.1 思路：

- 1 把本应显示的列表与用户已选择的属性值列表用循环交叉判断，如果匹配把本应显示的那个属性去掉。
- 2 已选择的属性值列表，要携带点击跳转的路径，这个路径参数就是咱们上边讲的那个“历史参数”，但是要把自己本身的属性值去掉。

5.4.2 扩展类

增加 BaseAttrValueExt 类，这个类是扩展了原 BaseAttrValue 类为了方便携带参数。

```
public class BaseAttrValueExt extends BaseAttrValue {  
  
    String attrName ;  
  
    String cancelUrlParam;  
  
}
```

5.4.3 controller 中的 getList 方法增加

代码如下

```
//去掉已选择的属性值  
if(skuLsParams.getValueld()!=null&&attrList!=null) {  
    for (int i = 0; valuelds != null && i < valuelds.length; i++) {  
        String selectedValueld = valuelds[i];  
  
        for (Iterator<BaseAttrInfo> iterator = attrList.iterator(); iterator.hasNext(); ) {  
            BaseAttrInfo baseAttrInfo = iterator.next();  
            List<BaseAttrValue> attrValueList = baseAttrInfo.getAttrValueList();  
            for (BaseAttrValue baseAttrValue : attrValueList) {  
                if (selectedValueld.equals(baseAttrValue.getId())) {  
                    BaseAttrValueExt baseAttrValueExt = new BaseAttrValueExt();  
                    baseAttrValueExt.setAttrName(baseAttrInfo.getAttrName());  
                    baseAttrValueExt.setId(baseAttrValue.getId());  
  
                    baseAttrValueExt.setValueName(baseAttrValue.getValueName());  
                    baseAttrValueExt.setAttrId(baseAttrInfo.getId());  
                    String cancelUrlParam = makeUrlParam(keyword,catalog3Id,  
                    valuelds, selectedValueld);  
                    baseAttrValueExt.setCancelUrlParam(cancelUrlParam);  
  
                    selectedValuelist.add(baseAttrValueExt);  
  
                    iterator.remove();// 如果选中了该属性 就从属性列表中去掉  
                }  
            }  
        }  
    }  
}
```

```

    }
  }
}

model.addAttribute("selectedValueList", selectedValuelist);
model.addAttribute("keyword", keyword);
model.addAttribute("attrList", attrList);

```

这块代码看似多层循环嵌套性能隐患，其实因为单次循环基本不会超过五次，循环中没有网络或者 io 访问，完全在虚拟机中运行，所以即使多层循环嵌套压力也不会太大。

5.4.4 页面代码

```

<div class="GM_ipone">
  <div class="GM_ipone_bar">
    <div class="GM_ipone_one a">
      筛选条件
    </div>
    <i></i>
    <span th:if="{keyword}!=null" th:text="{&quot;'+{keyword}+'&quot;}"></span>
    <a class="select-attr" th:each="selectedValue:{selectedValueList}"
      th:href="/list.html?'+{selectedValue.cancelUrlParam}"
      th:utext="{selectedValue.attrName}+'.'+{selectedValue.valueName} +'<b> X </b>'" > 屏
      幕尺寸:5.1-5.5 英寸
    </a>
  </div>
</div>

```

5.5 页面功能--分页

由于咱们在 es 中查询数据时已经完成了后台分页，这里主要就是把分页结果丢给前台页面展示就好了。

注意每个页面都要带上历史参数。

5.5.1 controller 中的 getList 里加入

```
long totalPages = skuLsResult.getTotalPages();
model.addAttribute("totalPages",totalPages);
model.addAttribute("pageNo",skuLsParams.getPageNo());
```

至此 ListController 就全部完成了

5.5.2 ListController 代码

```
@Controller
public class ListController {

    @Reference
    ManageService manageService;

    @Reference
    ListService listService;

    @RequestMapping("list.html")
    public String getList(    SkuLsParams skuLsParams, Model model){

        skuLsParams.setPageSize(4);
        //根据参数返回 sku 列表
        SkuLsResult skuLsResult = listService.search(skuLsParams);
        model.addAttribute("skuLsInfoList",skuLsResult.getSkuLsInfoList());

        //根据查询的结果返回属性和属性值列表
        List<BaseAttrInfo> attrList=null;
        if(skuLsParams.getCatalog3Id()!=null){
            attrList = manageService.getAttrList(skuLsParams.getCatalog3Id());
        }else {
            List<String> attrValueIdList = skuLsResult.getAttrValueIdList();
            if(attrValueIdList!=null&&attrValueIdList.size()>0){
                attrList = manageService.getAttrList(attrValueIdList);
            }
        }

        model.addAttribute("attrList",attrList);

        //历史参数
        String[] valueIds=skuLsParams.getValueId();
        String catalog3Id = skuLsParams.getCatalog3Id();
        String keyword=skuLsParams.getKeyword();
```

```
String urlParam = makeUrlParam(keyword,catalog3Id, valueIds, "");
model.addAttribute("urlParam",urlParam);

List<BaseAttrValueExt> selectedValuelist =new ArrayList<>();

//去掉已选择的属性值
if(skuLsParams.getValueId()!=null&&attrList!=null) {
    for (int i = 0; valueIds != null && i < valueIds.length; i++) {
        String selectedValueId = valueIds[i];

        for (Iterator<BaseAttrInfo> iterator = attrList.iterator(); iterator.hasNext(); ) {
            BaseAttrInfo baseAttrInfo = iterator.next();
            List<BaseAttrValue> attrValueList = baseAttrInfo.getAttrValueList();
            for (BaseAttrValue baseAttrValue : attrValueList) {
                if (selectedValueId.equals(baseAttrValue.getId())) {
                    BaseAttrValueExt baseAttrValueExt = new BaseAttrValueExt();
                    baseAttrValueExt.setAttrName(baseAttrInfo.getAttrName());
                    baseAttrValueExt.setId(baseAttrValue.getId());

                    baseAttrValueExt.setValueName(baseAttrValue.getValueName());
                    baseAttrValueExt.setAttrId(baseAttrInfo.getId());
                    String cancelUrlParam = makeUrlParam(keyword,catalog3Id,
valueIds, selectedValueId);
                    baseAttrValueExt.setCancelUrlParam(cancelUrlParam);

                    selectedValuelist.add(baseAttrValueExt);

                    iterator.remove();// 如果选中了该属性 就从属性列表中去
掉
                }
            }
        }
    }

    model.addAttribute("selectedValueList", selectedValuelist);
    model.addAttribute("keyword",keyword);
    model.addAttribute("attrList",attrList);

    long totalPages = skuLsResult.getTotalPages();
    model.addAttribute("totalPages",totalPages);
    model.addAttribute("pageNo",skuLsParams.getPageNo());

    //以下是查询商品信息

    return "list";
}
```

```
private String makeUrlParam(String keyword,String catalog3Id, String[] valueIds,String
excludeValueId ){
    String url="";
    List<String> paramList=new ArrayList<>();
    if(keyword!=null&&keyword.length()>0){
        paramList.add("keyword="+keyword);
    }
    if(catalog3Id!=null){
        paramList.add("catalog3Id="+catalog3Id);
    }

    if(valueIds!=null) {
        for (int i = 0; i < valueIds.length; i++) {
            String valueId = valueIds[i];
            if (!excludeValueId.equals(valueId)) {
                paramList.add("valueId=" + valueId);
            }
        }
    }
    url = StringUtils.join(paramList, "&");
    return url;
}
```

5.5.3 分页的页面部分代码

```
<script th:inline="javascript">
/**/

$(function(){
    $(".page_span1").empty();
    var totalPages = [{${totalPages}}];
    var pageNo = [{${pageNo}}];
    var urlParam = [{${urlParam}}];

    //上一页
    var $lastPage = '&lt;a href="/list.html?'+urlParam+'&amp;pageNo='+pageNo-1+'"&gt;&lt; 上一
    页&lt;/a&gt;';
    if(pageNo == 1){
        $lastPage = '&lt;a disabled="true"&gt;&lt; 上一页&lt;/a&gt;';
    }
    $(".page_span1").append($lastPage);
    for(var i = 1; i &lt;= totalPages; i++) {
        if (i==pageNo){</pre></div><div data-bbox="146 894 169 908" data-label="Page-Footer">54</div><div data-bbox="144 912 793 931" data-label="Page-Footer">更多 Java - 大数据 - 前端 - python 人工智能资料下载, 可访问百度: 尚硅谷官网</div>
```

```

        $(".page_span1").append('<a style="border: 0;font-size: 20px;color: red;background: #fff">'+i+'</a>');
    } else if(i==1 || i==totalPage || (i>=(pageNo-2)&&i<=(pageNo+2))){
        $(".page_span1").append('<a href="/list.html?'+urlParam+'&pageNo='+i+'">'+i+'</a>');
    } else if((i==(pageNo+3) && i<totalPage) || (i==(pageNo-3) && i>1)){
        $(".page_span1").append('<a style="border: 0;font-size: 20px;color: #999;background: #fff">...</a>');
    }
}
// 下一页
var $nextPage = '<a href="/list.html?'+urlParam+'&pageNo='+pageNo+1+'">下一
页 ></a>';
if(pageNo == totalPage || totalPage==0){
    $nextPage = '<a disabled="true">下一页 ></a>';
}
$(".page_span1").append($nextPage);

})
/*]]>*/
</script>

```

分页栏由 6 部分组成，通过判断页面决定显示效果。



6 排序

页面结构完成了，考虑一下如何排序，es 查询的 dsl 语句中我们是用了 hotScore 来进行排序的。

但是 hotScore 从何而来，根据业务去定义，也可以扩展更多类型的评分，让用户去选择如何排序。

这里的 hotScore 我们假定以点击量来决定热度。

那么我们每次用户点击，将这个评分+1,不就可以了么。

6.1 问题：

1、 es 大量的写操作会影响 es 性能，因为 es 需要更新索引，而且 es 不是内存数据库，会做相应的 io 操作。

2、而且修改某一个值，在高并发情况下会有冲突，造成更新丢失，需要加锁，而 es 的乐观锁会恶化性能问题。

从业务角度出发，其实我们为商品进行排序所需要的热度评分，并不需要非常精确，大致能比出个高下就可以了。

利用这个特点我们可以稀释掉大量写操作。

6.2 解决思路：

用 redis 做精确计数器，redis 是内存数据库读写性能都非常快，利用 redis 的原子性的自增可以解决并发写操作。

redis 每计 100 次数（可以被 100 整除）我们就更新一次 es ，这样写操作就被稀释了 100 倍，这个倍数可以根据业务情况灵活设定。

代码 在 listServiceImpl 中增加更新操作

6.3 代码：更新 es

```
private void updateHotScore(String skuId, Long hotScore) {
    String updateJson="{\n" +
        "  \"doc\":{\n" +
        "    \"hotScore\":\"+hotScore+\n" +
        "  }\n" +
        "}";

    Update update = new
Update.Builder(updateJson).index("gma11").type("SkuInfo").id(skuId).build()
;
    try {
        jestClient.execute(update);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

6.4 更新 redis 计数器

```
//更新热度评分
@Override
public void incrHotScore(String skuId){
    Jedis jedis = redisUtil.getJedis();
    int timesToEs=100;
    Double hotScore = jedis.zincrby("hotScore", 1, "skuId:" + skuId);
    if(hotScore%timesToEs==0){
        updateHotScore( skuId, Math.round(hotScore) );
    }
}
```

6.5 详情页调用

这个 `incrHotScore` 方法可以在进入详情页面的时候调用。

```
@Controller
public class ItemController {

    @Reference
    ListService listService;

    @Reference
    ManageService manageService;

    @RequestMapping("/{skuId}.html")
    public String getSkuInfo(@PathVariable("skuId") String skuId, Model model){
        .....
        listService.incrHotScore(skuId); //最终应该由异步方式调用
    }
}
```