**Elasticsearch官网：https://www.elastic.co/cn/products/elasticsearch**

如上所述，Elasticsearch具备以下特点：

分布式，无需人工搭建集群（solr就需要人为配置，使用Zookeeper作为注册中心）

Restful风格，一切API都遵循Rest原则，容易上手

近实时搜索，数据更新在Elasticsearch中几乎是完全同步的。

**使用容器安装 Elasticsearch**

拉取最新版本的镜像

docker pull elasticsearch:7.4.1

ElasticSearch的默认端口是9200，我们把宿主环境9200端口映射到Docker容器中的9200端口，就可以访问到Docker容器中的ElasticSearch服务了，同时我们把这个容器命名为es。

docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" elasticsearch:7.4.1

**2.3 配置跨域**

**2.3.1 进入容器**

由于要进行配置，因此需要进入容器当中修改相应的配置信息。

docker exec -it es /bin/bash

# 显示文件ls结果如下：LICENSE.txt  README.textile  config  lib   modulesNOTICE.txt   bin             data    logs  plugins

# 进入配置文件夹 cd config

# 显示文件ls结果如下：elasticsearch.keystore  ingest-geoip  log4j2.properties  roles.yml  users\_roleselasticsearch.yml       jvm.options   role\_mapping.yml   users

# 修改配置文件

vi elasticsearch.yml

# 加入跨域配置

http.cors.enabled: true
http.cors.allow-origin: "\*"

重启docker

docker restart es

输入<http://xxx:9200>

{
"name" : "3bf71114192c",
"cluster\_name" : "docker-cluster",
"cluster\_uuid" : "GDfLLhu2S4q-qRl9nwsibA",
"version" : {
"number" : "7.9.0",
"build\_flavor" : "default",
"build\_type" : "docker",
"build\_hash" : "a479a2a7fce0389512d6a9361301708b92dff667",
"build\_date" : "2020-08-11T21:36:48.204330Z",
"build\_snapshot" : false,
"lucene\_version" : "8.6.0",
"minimum\_wire\_compatibility\_version" : "6.8.0",
"minimum\_index\_compatibility\_version" : "6.0.0-beta1"
},
"tagline" : "You Know, for Search"
}

9300是用tcp访问，可以使用，但是无法使用浏览器访问

**三、Docker 部署 ElasticSearch-Head（可不装）**

为什么要安装ElasticSearch-Head呢，原因是需要有一个管理界面进行查看ElasticSearch相关信息

拉取镜像：

docker pull mobz/elasticsearch-head:5

运行镜像：

docker run -d --name es\_admin -p 9100:9100 mobz/elasticsearch-head:5

**四. 因为ElasticSearch-Head页过于面简单，所以直接使用官网的Kibana**

拉取镜像（与es一致，否则出现不兼容问题）

docker pull kibana:7.4.1

**2、编辑kibana.yml配置文件**

kibana.yml配置文件放在宿主机/data/elk/目录下，内容如下：

# Default Kibana configuration for docker target
server.name: kibana
server.host: "0"
elasticsearch.hosts: [ "http://188.188.3.19:9200" ]
xpack.monitoring.ui.container.elasticsearch.enabled: true

启动容器

docker run -d --restart=always --log-drivdocker run -d -p 5601:5601 --link elasticsearch -e "ELASTICSEARCH\_URL=http://188.188.3.19:9200" kibana:7.4.1

**出现** Kibana server is not ready yet ，多刷新几次

**docker安装ik中文分词器**

安装 ik（下载对应版本的ik）

<https://github.com/medcl/elasticsearch-analysis-ik/releases>

进入/root/ik目录解压，如果命令不存在则先安装（yum install unzip）

unzip /root/ik/elasticsearch-analysis-ik-7.4.1.zip

将ik文件夹复制到指定的docker里面

docker cp /root/ik/ elasticsearch:/usr/share/elasticsearch/plugins/ik

重启：

docker restart elasticsearch

**操作索引：**

Elasticsearch也是基于Lucene的全文检索库，本质也是存储数据，很多概念与MySQL类似的。

对比关系：

索引（indices）--------------------------------Databases 数据库

​ 类型（type）-----------------------------Table 数据表

​ 文档（Document）----------------Row 行

​ 字段（Field）-------------------Columns 列

详细说明：

|  |  |
| --- | --- |
| 概念 | 说明 |
| 索引库（indices) | indices是index的复数，代表许多的索引， |
| 类型（type） | 类型是模拟mysql中的table概念，一个索引库下可以有不同类型的索引，比如商品索引，订单索引，其数据格式不同。不过这会导致索引库混乱，因此未来版本中会移除这个概念 |
| 文档（document） | 存入索引库原始的数据。比如每一条商品信息，就是一个文档 |
| 字段（field） | 文档中的属性 |
| 映射配置（mappings） | 字段的数据类型、属性、是否索引、是否存储等特性 |

是不是与Lucene和solr中的概念类似。

另外，在SolrCloud中，有一些集群相关的概念，在Elasticsearch也有类似的：

索引集（Indices，index的复数）：逻辑上的完整索引

分片（shard）：数据拆分后的各个部分

副本（replica）：每个分片的复制

要注意的是：Elasticsearch本身就是分布式的，因此即便你只有一个节点，Elasticsearch默认也会对你的数据进行分片和副本操作，当你向集群添加新数据时，数据也会在新加入的节点中进行平衡。

**1. 创建索引**

**Put /**索引库名
{
"settings": {
"number\_of\_shards": 3, //分片个数
"number\_of\_replicas": 2 //副本个数
}
}

**1.1 删除索引**

DELETE /索引库名

**2. 创建映射字段（7以后的版本不支持type，所以要加上**?include\_type\_name=true**）**

PUT /索引库名/\_mapping/类型名称?include\_type\_name=true
{
"properties": {
"字段名": {
"type": "类型",
"index": true，
"store": true，
"analyzer": "分词器"
}
}
}

**2.1 7.X版本以后写法**

PUT /intell/\_mapping
{
"properties": {
"title":{
"type": "text",
"analyzer": "ik\_max\_word"
},
"images":{
"type": "keyword",
"index": false
},
"price":{
"type": "float"
}
}
}

**3. 新增数据**

POST /索引库名
{
"key":"value"
}

POST /intell/goods/
{
"title":"小米手机",
"images":"http://image.leyou.com/12479122.jpg",
"price":2699.00
}

**3.1 7.X版本以后写法**

POST /intell/\_doc
{
"title":"大米手机",
"images":"http://image.leyou.com/12479122.jpg",
"price":2899.00
}

**4. 查询数据**

GET /intell/\_doc/mnSBI3QBson7zHOLGwD0

**5. 全文检索**

**查询and条件连接**

GET /intell/\_search
{
"query": {
"match": {
"title": {"query": "大米手机","operator": "and"}
}
}
}

**多字段匹配：multi\_match 与 match 类似，不同的是它可以在多个字段中查询**

**GET**

**GET /intell/\_search**
**{**
 **"query": {**
 **"multi\_match": {**
 **"query": "哈哈",**
 **"fields": ["title","subTitle"]**
 **}**
 **}**
**}**

**词条匹配（搜索除了text类型以外的完整词条，不完整的词条text类型也能搜索到）**

GET /intell/\_search
{
"query": {
"term": {
"title": {
"value": "大米手机"
}
}
}
}

**6 结果过滤**

**GET /intell/\_search**
**{**
 **"\_source": ["title"],** 
 **"query": {**
 **"match": {**
 **"title": "大米手机"**
 **}**
 **}**
**}**

排除不想显示的字段

GET /intell/\_search
{
"\_source": {
"excludes": "{field}"
},
"query": {
"match": {
"title": "大米手机"
}
}
}

**7 其他查询**

**模糊查询 fuzzy**

GET /intell/\_search
{
"\_source": {
"excludes": "{field}"
},
"query": {
"fuzzy": {
"title": "a米"
}
}
}

**范围查询 range**

**GET /intell/\_search**
**{**
 **"\_source": {**
 **"excludes": "{field}"**
 **},**
 **"query": {**
 **"range": {**
 **"price": {**
 **"gte": 10,**
 **"lte": 20000**
 **}**
 **}**
 **}**
**}**

**布尔查询 bool（must：与，should：或，must\_not：非）**

GET /intell/\_search
{
"\_source": {
"excludes": "{field}"
},
"query": {
"bool": {
"must": [
{"match": {
"title": "大米"
}},
{
"range": {
"price": {
"gte": 10,
"lte": 20000
}
}
}
]
}
}
}

过滤条件不能影响得分

GET /intell/\_search
{
"\_source": {
"excludes": "{field}"
},
"query": {
"bool": {
"must": [
{"match": {
"title": "大米"
}}
],
"filter": {
"range": {
"price": {
"gte": 10,
"lte": 20000
}
}
}
}
}
}

**排序和分页**

**GET /intell/\_search**
**{**
 **"\_source": {**
 **"excludes": "{field}"**
 **},**
 **"query": {**
 **"bool": {**
 **"must": [**
 **{"match": {**
 **"title": "大米"**
 **}}**
 **],**
 **"filter": {**
 **"range": {**
 **"price": {**
 **"gte": 10,**
 **"lte": 20000**
 **}**
 **}**
 **}**
 **}**
 **},**
 **"sort": [**
 **{**
 **"price": {**
 **"order": "desc"**
 **}**
 **}**
 **],**
 **"from": 0,**
 **"size": 2**
**}**

**8. 聚合**

**Elasticsearch 中的聚合，包含多种类型，常用的两种，一个叫桶，一个叫度量**

**桶（bucket）**

**桶的作用，是按照某种方式对数据进行分组**

**度量（metrics）**

**分组完成后，我们一般会对组中的数据，进行聚合运算**

Elasticsearch中提供的划分桶的方式有很多：

Date Histogram Aggregation: 根据数据阶梯分组，例如给定阶梯为周，会自动每周分一组

Histogram Aggregation: 根据数据阶梯分组，与日期类似

Terms Aggregation: 根据词条内容分组，词条内容完全匹配的为一组

Range Aggregation: 数值和日期的范围，指定开始和结束，然后按段分组

综合所述，我们发现bucket aggregations 只负责对数据进行分组，并不进行计算，因此往往bucket中往往会嵌套另一种聚合：metrics aggregations 即度量

比较常用的一些度量聚合方式：

- Avg Aggregation：求平均值

- Max Aggregation：求最大值

- Min Aggregation：求最小值

- Percentiles Aggregation：求百分比

- Stats Aggregation：同时返回avg、max、min、sum、count等

- Sum Aggregation：求和

- Top hits Aggregation：求前几

- Value Count Aggregation：求总数

JAVA 代码使用

*/\*\**
 *\** ***@description:***
*\** ***@program:*** *demobase*
 *\** ***@author:*** *Charles*
 *\** ***@created:*** *2020-08-25 16:19*
 *\*\*/*
@RunWith(SpringRunner.class)
@SpringBootTest
public class EsTest {
@Autowired
private ElasticsearchRestTemplate template;
@Autowired
private ItemRepository repository;
@Test
public void test() {
IndexOperations indexOps = template.indexOps(Item.class);
//删除
indexOps.delete();
//创建
indexOps.create();
//映射关系
indexOps.putMapping(indexOps.createMapping(Item.class));
// template.deleteIndex(Item.class);
// //创建索引
// template.createIndex(Item.class);
// //映射关系
// template.putMapping(Item.class);
}
@Test
public void indexList() {
List<Item> list = new ArrayList<>();
list.add(new Item(1L, "小米手机", "手机", "小米", 3299.00, "http://image.leyou.com/13123.jpg"));
list.add(new Item(6L, "小米手机", "手机", "小米", 3299.00, "http://image.leyou.com/13123.jpg"));
list.add(new Item(2L, "坚果手机R1", "手机", "锤子", 3699.00, "http://image.leyou.com/13123.jpg"));
list.add(new Item(3L, "华为META10", "手机", "华为", 4499.00, "http://image.leyou.com/13123.jpg"));
list.add(new Item(4L, "小米Mix2S", "手机", "小米", 4299.00, "http://image.leyou.com/13123.jpg"));
list.add(new Item(5L, "荣耀V10", "手机", "华为", 2799.00, "http://image.leyou.com/13123.jpg"));
// 接收对象集合，实现批量新增
// repository.saveAll(list);
template.save(list);
}
@Test
public void testFind() {
Iterable<Item> all = repository.findAll();
for (Item item : all) {
System.*out*.println("item=" + item);
}
}
@Test
public void testFindBy() {
List<Item> list = repository.findByPriceBetween(2000d, 4000d);
for (Item item : list) {
System.*out*.println("item=" + item);
}
}
@Test
public void testQuery() {
//查询构建器
NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();
//结果过滤
queryBuilder.withSourceFilter(new FetchSourceFilter(new String[]{"id", "title", "price"}, null));
//添加过滤条件
queryBuilder.withQuery(QueryBuilders.*matchQuery*("title", "小米手机"));
//排序
queryBuilder.withSort(SortBuilders.*fieldSort*("price").order(SortOrder.*DESC*));
//分页
queryBuilder.withPageable(PageRequest.*of*(0, 1));
// SearchHits<Item> searchHits = template.search(queryBuilder.build(), Item.class);
// List<Item> list = searchHits.stream().map(SearchHit::getContent).collect(Collectors.toList());
// for (SearchHit<Item> hit : hits) {
// System.out.println("hit=" + hit);
// }
// new PageImpl(list, , searchHits.getTotalHits());
Page<Item> result = repository.search(queryBuilder.build());
System.*out*.println("total:" + result.getTotalElements());
System.*out*.println("totalPages:" + result.getTotalPages());
for (Item item : result) {
System.*out*.println("item=" + item);
}
}
*/\*\**
 *\* 单层聚合*
 *\*/*
@Test
public void testAgg() {
NativeSearchQueryBuilder queryBuilder = new NativeSearchQueryBuilder();
String aggName = "popularBrand";
//聚合
queryBuilder.addAggregation(AggregationBuilders.*terms*("popularBrand").field("brand.keyword"));
SearchHits<Item> search = template.search(queryBuilder.build(), Item.class);
//解析结果
Terms terms = search.getAggregations().get(aggName);
//获取桶
List<? extends Terms.Bucket> buckets = terms.getBuckets();
for (Terms.Bucket bucket : buckets) {
System.*out*.println("bucket.getKeyAsString()" + bucket.getKeyAsString());
System.*out*.println("bucket.getDocCount()" + bucket.getDocCount());
}
}
}