# 一般查询

/\*\*

\* 使用QueryBuilder

\* termQuery("key", obj) 完全匹配

\* termsQuery("key", obj1, obj2..) 一次匹配多个值

\* matchQuery("key", Obj) 单个匹配, field不支持通配符, 前缀具高级特性

\* multiMatchQuery("text", "field1", "field2"..); 匹配多个字段, field有通配符忒行

\* matchAllQuery(); 匹配所有文件

\*/

/\*\*

## 范围内查询 （数字或单词）

\*/

public void testRangeQuery() {

QueryBuilder queryBuilder = QueryBuilders.rangeQuery("user")

.from("kimchy")

.to("wenbronk")

.includeLower(true) // 包含上界

.includeUpper(true); // 包含下届

searchFunction(queryBuilder);

}

# 聚合

ES中的聚合被分为两大类。

    1、Metrics, Metrics 是简单的对过滤出来的数据集进行avg,max等操作，是一个单一的数值。

2、ucket, Bucket 你则可以理解为将过滤出来的数据集按条件分成多个小数据集，然后Metrics会分别作用在这些小数据集上。

聚合在ELK里面是一个非常重要的概念

## date\_histogram 时间柱状图

参考csdn博客：<https://blog.csdn.net/opera95/article/details/78614244> 很好很详细

Date histogram的用法与histogram差不多，只不过区间上支持了日期的表达式。

### 用法

**public** **static** **void** dateHistogramQuery(Client client) {

SearchResponse res = **null**;

AggregationBuilder agg = AggregationBuilders.*dateHistogram*("agg").field("publish\_date")

.interval(DateHistogramInterval.***YEAR***).

minDocCount(1);

res = client.prepareSearch("search\_test").setTypes("article")

.setSearchType(SearchType.***DFS\_QUERY\_THEN\_FETCH***)

.addAggregation(agg)

.setFrom(0)

.setSize(0).execute().actionGet();

System.***out***.println(res);

// on shutdown

client.close();

## 最有用的嵌套组合查询

**public** **static** **void** booQuery(Client client) {//最有用的嵌套查询

SearchResponse res = **null**;

QueryBuilder qb = QueryBuilders.*boolQuery*()

.should(QueryBuilders.*termQuery*("title", "02"))

// .mustNot(QueryBuilders.termQuery("title", "article"))

.should(QueryBuilders.*termQuery*("title", "relevance"));

// .filter(QueryBuilders.termQuery("title", "article"));

res = client.prepareSearch("search\_test").setTypes("article").setSearchType(SearchType.***DFS\_QUERY\_THEN\_FETCH***)

.setQuery(qb).setFrom(0).setSize(10).execute().actionGet();

**for** (SearchHit hit : res.getHits().getHits()) {

System.***out***.println(hit.getSourceAsString());

}

/\*\*

\* 组合查询

\* must(QueryBuilders) : AND

\* mustNot(QueryBuilders): NOT

\* should: : OR

\*/

# 具体学习描述

所有的elasticsearch操作都是通过 Client对象进行的。

操作会存储在client中 然后以批处理的方式执行  
-所有相关API 都基于java API

# 如何在ElasticSearch中批量提交索引 ？

（2）如何使用高级查询（包括，检索，排序，过滤，分页） ？   
（3）如何组合多个查询 ？   
（4）如何使用翻页深度查询 ？   
（5）如何使用基本的聚合查询 ？ 

（一）首先，我们思考下，为什么要使用批量添加，这个毫无疑问，因为效率问题，举个在生活中的例子，假如我们有50个人，要去美国旅游，不使用批处理的方式是，给每一个人派一架飞机送到美国，那么这就需要50次飞机的来回往来，假如使用了批处理，现在的情况就是一个飞机坐50个人，只需一次即可把所有人都送到美国，效率可想而知，生活也有很多实际的例子，大家可以自己想想。   
  
在原生的lucene中，以及solr中，这个批处理方式，实质是控制commit的时机，比如多少个提交一次，或者超过ranbuffersize的大小后自动提交，es封装了lucene的api提供bulk的方式来批量添加，原理也是，聚集一定的数量doc，然后发送一次添加请求。   
  
  
（二）只要我们使用了全文检索，我们的业务就会有各种各样的api操作，包括，任意维度的字段查询，过滤掉某些无效的信息，然后根据某个字段排序，再取topN的结果集返回，使用数据库的小伙伴们，相信大家都不陌生，在es中，这些操作都是支持的，而且还非常高效，它能满足我们大部分的需求   
  
  
（三）在es中，我们可以查询多个index，以及多个type，这一点是非常灵活地，我们，我们可以一次组装两个毫无关系的查询，发送到es服务端进行检索，然后获取结果。   
  
  
（四）es中，通过了scorll的方式，支持深度分页查询，在数据库里，我们使用的是一个cursor游标来记录读取的偏移量，同样的在es中也支持，这样的查询方式，它通过一个scrollid记录了上一次查询的状态，能轻而易举的实现深度翻页，本质上是对了Lucene的SearchAfter的封装。   
  
（五）es中，也提供了对聚合函数的支持，比如一些max，min，avg，count，sum等支持，除此之外还支持group，facet等操作，这些功能，在电商中应用非常广泛，基于lucene的solr和es都有很好的支持。 

# 一般

### 12.2.1  匹配查询

查看[匹配查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-match-query.htm) 

QueryBuilder qb = matchQuery(

"name", [注释1]

"kimchy elasticsearch" [注释2]

);

[注释1]：属性  
[注释2]：文本

### 12.2.2  多匹配查询

查看 [多匹配查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-multi-match-query.htm) 

QueryBuilder qb = multiMatchQuery(

"kimchy elasticsearch", [注释1]

"user", "message" [注释2]

);

[注释1]：文本  
[注释2]：属性列表

### 12.2.3  一般条件查询

查看[一般条件查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-common-terms-query.htm) 

QueryBuilder qb = commonTermsQuery("name", [注释1]

"kimchy"); [注释2]

[注释1]：属性  
[注释2]：值

### 12.2.4  查询字符串查询

查看[查询字符串查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-query-string-query.htm) 

QueryBuilder qb = queryStringQuery("+kimchy -elasticsearch"); [注释1]

[注释1]：文本

### 12.2.5  简单的查询字符串查询

查看[简单的查询字符串查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-simple-query-string-query.htm) 

QueryBuilder qb = simpleQueryStringQuery("+kimchy -elasticsearch"); [注释1]

[注释1]：文本

## 12.3  条件等级查询

虽然全文搜索将在执行之前分析查询字符串，但是条件等级查询对存储在反向索引中的确切条件进行操作。  
这些查询通常用于结构化数据，如数字，日期和枚举，而不是全文本字段。 或者，它们允许您绘制低级查询，以前的分析过程。  
此组中的查询为：

### 12.3.1  词汇查询

查找包含在指定字段中指定的确切词汇的文档。  
查看[词汇查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-term-query.htm) 

QueryBuilder qb = termQuery(

"name", [注释1]

"kimchy" [注释1]

);

[注释1]：属性  
[注释2]：文本

### 12.3.2  多词汇查询

查找包含指定字段中指定的所有确切词汇的文档。  
查看[多词汇查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-terms-query.htm) 

QueryBuilder qb = termsQuery("tags", [注释1]

"blue", "pill"); [注释2]

[注释1]：属性  
[注释2]：多个要匹配的文本词汇

### 12.3.3  范围查询

查找指定字段包含指定范围内的值（日期，数字或字符串）的文档。  
查看[范围查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-range-query.htm) 

QueryBuilder qb = rangeQuery("price") [注释1]

.from(5) [注释2]

.to(10) [注释3]

.includeLower(true) [注释4]

.includeUpper(false); [注释5]

[注释1]：属性  
[注释2]：起始值  
[注释3]：结束值  
[注释4]：是否包含起始值，false：不包含，true：包含  
[注释5]：是否包含结束值，false：不包含，true：包含

// A simplified form using gte, gt, lt or lte

QueryBuilder qb = rangeQuery("age") [注释1]