# ES 5.x 常用查询接口(transport)使用文档

版本修订日志

|  |  |  |
| --- | --- | --- |
| 修订时间 | 修订人 | 修订内容 |
| 2018年06月05日 | zqz | 创建 (版本V1) |
|  |  |  |
|  |  |  |
|  |  |  |

# 使用方法(JDK1.8 +)：

## 初始化 EsSearchClient

获取client 相当于 操作mysql中的 connection连接，这是使用查询/聚合接口的第一步，

工具类对外提供了四种初始化方法,分别如下：

#### EsSearchClient(String index, String type)

|  |  |
| --- | --- |
| 参数解释 | 调用方式 |
| Index：索引名/别名  type: 类型名 | EsSearchClient client = newEsSearchClient("index\_name", "type\_name") |

#### EsSearchClient(String indexName)

|  |  |
| --- | --- |
| 参数解释 | 调用方式 |
| indexName：索引名/别名 | EsSearchClient client = new EsSearchClient("index\_name") |

#### EsSearchClient( String[] indices, String[] types)

|  |  |
| --- | --- |
| 参数解释 | 调用方式 |
| indices：索引名/别名数组  types: 类型名数组 | EsSearchClient client = new EsSearchClient(**new** String[]{"index\_name"}, **new** String[]{"type\_name"}) |

#### EsSearchClient( String[] indices)

|  |  |
| --- | --- |
| 参数解释 | 调用方式 |
| indices：索引名/别名数组 | EsSearchClientclient=new EsSearchClient(**new**String[]{"index\_name"}) |

## 查询接口

查询相当于对全量数据进行过滤筛选，提供以下接口共使用

### （1）精准/精确查询

精准或精确查询相当于sql语句中的 where auto\_id = 1 或 userName = “test”,字段一般不进行分词；接口对外提供三种方式，如下：

#### addPrimitiveTermQuery(String field, Object term, QueryTypeEnum operator)

参数解释：

Field：要进行精确查询的字段名称，比如auto\_id userName;

Term: 匹配的值，此值不进行分词；

Operator：查询条件拼接方式，枚举类型（后面再次出现不再说明）：

|  |  |
| --- | --- |
| ***枚举类型*** | ***含义*** |
| ***MUST*** | ***一定为此值 add*** |
| ***MUST\_NOT*** | ***一定不能是此值 not*** |
| ***SHOULD*** | ***相当于 or*** |

#### addPrimitiveTermQuery(String field, String[] terms, QueryTypeEnum operator)

terms: 只要field 匹配到数组中的一个，即算为命中

#### **addPrimitiveTermQuery(String field, List<String> termList, QueryTypeEnum operator)**

termList: 只要field 匹配到链表中的一个，即算为命中

### （2）范围查询

类似于 sql 中的 > 、>=、 <和 <= ，进行范围限定，对外提供接口为

#### addRangeQuery

(String field, Object startTerm, RangeCommon startOperate, Object endTerm,RangeCommon endOperate, QueryTypeEnum operator)

参数解释：

Field：进行范围限定的字段名称

startTerm：起始值

endTerm：结束值

RangeCommon 为枚举类型， 1. GT： > 2. GTE：>= 3. LT：< 4. LTE：<=

|  |  |
| --- | --- |
| ***枚举类型*** | ***含义*** |
| ***GT*** | ***>*** |
| ***GTE*** | ***>=*** |
| ***LT*** | ***<*** |
| ***LTE*** | ***<=*** |

Operator：参考以上介绍

备注：若字段为日期类型，则要注意字段的格式，使起始值类型与 mapping类型一致。

### （3）关键词查询

精确查询搜索条件太严格，使用场景有限，所以催生出 关键词查询，只要字段值term列表中包含 关键词分词后的term即可，具体结合方式取决于KeywordsCombine 的配置

#### addKeywordsQuery(String field, String keywords, QueryTypeEnum operator, KeywordsCombine combine)

参数解释：

Field：字段名称

Keywords：关键词

KeywordsCombine ：枚举类 关键词分词后，关键词之间拼接的方式 ***AND***, ***OR***

|  |  |
| --- | --- |
| ***枚举类型*** | ***含义*** |
| ***AND*** | ***关键词经过分词后，所有的term 都要命中，才能返回该文档*** |
| ***OR*** | ***关键词经过分词后，所有的term中命中一个即返回该文档*** |

#### addKeywordsQuery(String[] fields, String keywords, QueryTypeEnum operator, KeywordsCombine combine)

参数解释：

fields：字段列名称，只要有一个字段命中，即返回该文档

### （4）短语查询

由于精准/精确查询要求太严格，而关键词查询只是取决于包含与否，并不对term之间的位置有所要求，但是 对于 短语来说是有位置要求的，如 hello world这一短语,当进行 关键词搜索keywords= world hello,也会返回出来因为 关键词查询只进行包含判断，并不进行位置判断，那么在这种情况下 短语查询应运而生。

#### addPhraseQuery(String field, Strin**g phrase,** QueryTypeEnum operator)

参数介绍：

Field：字段名称

Phrase: 短语

#### addPhraseQuery(String field, Strin**g phrase**, int slop，QueryTypeEnum operator)

参数介绍：

Field：字段名称

Phrase: 短语

Slop：跨度，默认值为0，跨度越大辐射的范围越大，返回的文档数可能越多；

如quick brown fox为文档，则quick fox 2个项的slop为1，quick向后移动一次.而fox quick 需要quick向后移动3次，所以slop为3

#### addPhrasePrefixQuery(String field, String phrase, int slop, QueryTypeEnum operator)

短语前缀查询

摘自ES官网：The match\_phrase\_prefix is the same as match\_phrase, except that it allows for prefix matches on the last term in the text.

当搜索项为中文时，几乎无差别，当搜索英文时存在较大差异，比如quick brown fox，

当搜索 quick brown f时 addPhraseQuery不会返回结果，当进行addPhrasePrefixQuery 时会返回结果，因为会对最后一个 term 的前缀进行匹配；

#### addPhrasePrefixQuery(String field, String contextStr,String splitStr, int slop, QueryTypeEnum operator)

多个短语拼接在一起，取交集 返回结果

参数说明：

contentStr : 短语集

splitStr: 短语间拆分格式

比如：contentStr为 中华人民 小康，那么会存在两个短语：中华人民 + 小康，splitStr 则为 “ ” 空格

#### addMultFieldsPhrasePrefixQuery(fields, contextStr, ISIOperator.MUST);

多字段短语匹配，只要满足任意一个字段 就会名中

参数说明：

Fields：短语匹配字段数组，只要命中任意一个数组 ，就会返回

contentStr : 短语集 中间按空格进行拆分

### （5）其他查询（不常用）

prefix，wildcard以及regexp查询基于词条(taken)进行操作，也就是说 是 term 内部的匹配；

若存在term 为 zhang

#### addPrefixQuery(String field, String term, QueryTypeEnum operator)

前缀查询 Eg：termVal 可为 zh/zhan ...

#### addRegexpQuery(String field, String regexpStr, QueryTypeEnum operator)

正则匹配查询 如：\*han\*

#### addWildcardQuery(String field, String term, QueryTypeEnum operator)

通配符查询 Eg：zha?g

### （6）自定义查询接口

以上为常用接口，若是业务比较复杂则需要自己组装查询条件，然后放入到自定义接口里即可；

#### addQuery(QueryBuilder query, QueryTypeEnum operator)

参数介绍：

Query：自定义的查询对象

Operator：查询条件 组装方式；

如：

|  |
| --- |
| BoolQueryBuilder builderSentment = QueryBuilders.*boolQuery*();  **for** (String sentimentVal : sentimentArr) {    QueryBuilder sentQb = QueryBuilders.*termQuery*("sentiment", sentimentVal);  builderSentment.should(sentQb);  }    client.addQuery(builderSentment, ISIOperator.***MUST***); |

### （7）排序接口

### addSortField(String field, SortOrder order)

参数介绍：

Field：排序字段名称

SortOrder：枚举类 ASC 升序，DESC 降序

|  |  |
| --- | --- |
| ***枚举类型*** | ***含义*** |
| ***ASC*** | ***升序*** |
| ***DESC*** | ***降序*** |

### 分页设置

#### setStart(int start)

设置起始索引，start = (pageNum-1)\*pageSize;

#### setRow(int rows)

设置结果返回条数Rows = pageSize;

## 三、聚合接口

聚合相当于 sql 中的分组统计

### 对字段值进行聚合统计

#### addMultFieldsStat(String[] fields, int topN, boolean orderByCount, boolean isDesc)

参数介绍：

Fields：聚合字段数组，具有层级关系 数组下标越小，层级越高；

topN: 聚合 范围 类似于 sql中 limit ，比如 top50即排序的前五十条；

orderByCount：是否按统计值进行排序，true 则进行排序；

isDesc： 若orderByCount = true, isDesc=true 则进行降序排列， isDesc=false 时则为升序。

#### **addMultFieldsStat(String[] fields, int topN, boolean orderByCount, boolean isDesc, Long[] minDocCounts, Long[] shardsMinDocCounts)**

参数介绍：

minDocCounts：与之相对应的字段，各分片汇总后最低的count

shardsMinDocCounts：与之相对应的字段，每个分片汇总后最低的count

这两个参数用来限定聚合范围，比如 当每个分片数字段统计值小于50 时，则不纳入聚合集。

### 按时间字段进行统计

比如说 按天/月等进行统计

#### addStatByDate(String dateField, Integer dateType, Integer interval)

参数介绍：

dateField: 统计字段

dateType: 0：day 1: week 2：hour，默认为天

|  |  |
| --- | --- |
| ***dateType数字值*** | ***含义*** |
| ***0*** | ***天（默认值）*** |
| ***1*** | ***周*** |
| ***2*** | ***小时*** |

Interval: 时间跨度 默认为1，

#### addFieldStatByDate(String statField, String dateField, Integer dateType, Integer interval)

这是一个**二维聚合**，第一层为statField字段聚合，第二层为 日期字段统计;

比如场景1：每个班级学生分为男女两个性别，每天上课都要进行打卡，最后求男同学与女同学中打卡量的趋势图或 时间系列?

因为 男女 同学都要进行统计，那么此时应调用此接口statField =sex（性别），dateField = classDate(上课时间)

### 度量聚合 如AVG、SUM、MIN、MAX等

度量聚合相当于 sql中 max() min() avg() 等

#### addMetricAgg(String field, Metric metric)

Field：统计字段

Metric metric： 度量类型，枚举类

|  |  |
| --- | --- |
| ***Metric 数字值*** | ***含义*** |
| ***MIN*** | ***最小值*** |
| ***MAX*** | ***最大值*** |
| ***AVG*** | ***平均值*** |
| ***SUM*** | ***求和*** |

### 自定义聚合接口

以上为常用的聚合接口，当业务场景比较复杂时，需自定义聚合函数，因此对外提供自定义接口；

#### addDefinedAggExpression(AggregationBuilder aggregation)

aggregation：自定义的聚合对象

例子如下：

|  |
| --- |
| **public** **static** **void** main(String[] args) {  // 第一步 建立连接  EsSearchClient client = **new** EsSearchClient("indexName", "short\_text\_info\_temp");  client.reset();  // 返回字段 设置  String [] fields = **new** String[1];  fields[0] = "content";  // 字段 值 范围检索  client.addRangeQuery("insert\_time",  "2017-05-10 19:20:13.000", RangeCommon.***GTE***,  "2018-05-15 23:20:13.000", RangeCommon.***LTE***,  ISIOperator.***MUST***);  // 字段值 匹配  PercentilesAggregationBuilder percentAgg =  AggregationBuilders  .*percentiles*("agg\_similary\_number")  .field("similary\_number")  .percentiles(1.0, 5.0, 10.0, 20.0, 30.0, 75.0, 95.0, 99.0);  // client.addQuery(percentAgg, ISIOperator.MUST);  client.addDefinedAggExpression(percentAgg);  //返回条数设置  client.setRow(0);  // 执行检索  client.execute(fields);  **long** total = client.getTotal();  System.***out***.println("命中："+total);  System.***out***.println("results："+results.size());  Percentiles agg =client.getAggResponse().getAggregations().get("agg\_similary\_number");  // For each entry  **for** (Percentile entry : agg) {  **double** percent = entry.getPercent(); // Percent  **double** value = entry.getValue(); // Value    System.***out***.println("{percent --> value} --- {"+percent+"-->"+value+"}");  }  } |

## 执行与结果获取

按照上面的方式拼接查询条件和聚合条件完毕后，则需要执行与获取结果；

### 执行程序

#### client.execute()

这种方式执行，在后面的查询结果获取中只能获取到 \_id(文档Id)

#### client.execute(fields)

这种方式可以设置返回字段，避免从source获取全量, 但是 **fields 中的字段要求 store:true 才行**，否则取值为 null; 如下

String[] fields = new String[3];

fields[0] = "send\_time";

fields[1] = "source\_number\_city";

fields[2] = "content";

client.execute(fields);

##### 注意：

有时查询的索引很多，要区分doc 是属于哪一个 type(表)下的，这是要在返回字段fields 中配置 “\_type”字段，即可返回是那一个表下的。

### 查询结果获取

#### List<String[]> results = client.getResults();

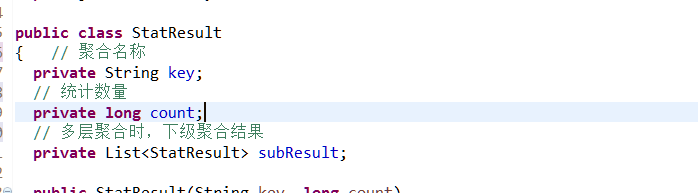
若是初始化client中types 为null,则 String[0] 为 type,若是types 不为空 string[] 顺序则和client.execute(fields) 中 返回字段数组fields 顺序一致；

### 聚合结果获取

#### 字段值聚合结果获取

List<StatResult> agg = client.getMultFieldStatResult();

其中 StatResult 为集合封装结果类



#### 时间字段统计结果获取

和时间字段统计对应，分为两种

1. List<StatResult> agg = client.getStatByDateResult();
2. List<StatResult> agg = client.getFieldStatByDateResult()();

#### 度量聚合结果

MetricResult metricAggResult = client.getMetricAggResult();

String key = metricAggResult.getKey();

**double** value = metricAggResult.getValue();

## 五、简单的例子：

### （1）查询：

|  |
| --- |
| **public** **static** **void** main(String[] args) {    EsSearchClient client = **new** EsSearchClient("indexName");  String[] fields = **new** String[3];//设置返回字段  fields[0] = "send\_time";  fields[1] = "source\_number\_city";  fields[2] = "content";  //client.addPrimitiveTermQuery("\_id", "2206", ISIOperator.MUST);    client.addPhrasePrefixQuery("content", "人民 你好", " ", 0, ISIOperator.***MUST***);  client.addRangeQuery("send\_time", "2017-09-04 11:20:01", RangeCommon.***GT***, "2017-10-04 11:20:01", RangeCommon.***LT***, ISIOperator.***MUST***);  client.setStart(0);//起始索引  client.setRow(10);//默认返回10条  client.execute(fields);  //client.execute();    List<String[]> results = client.getResults();  **for** (String[] strings : results) {  **for** (String string : strings) {    **if**(string.length() > 10){  string = client.getHighLightStr(string);// 根据需求设置高亮  }  System.***out***.println(string);  }  }  } |

### （2）聚合：

|  |
| --- |
| // 第一步 建立连接  String indexname = "indexName";  EsSearchClient client = **new** EsSearchClient(indexname, **null**);  client.reset();    // 1.聚合（聚合 agg 类似于mysql 分组）  // 多层嵌套聚合时， 获取聚合结果要 逐级获取  client.addMultFieldsStat(**new** String[] { "source\_number","source\_number\_province" }, 50, **true**, **true**);  client.setRow(0);//减小不必要的IO  client.execute(**null**);  List<StatResult> agg = client.getMultFieldStatResult();  **for** (StatResult obj : agg) {  String key = obj.getKey();  **long** count = obj.getCount();  String source\_number\_province =obj.getSubResult().get(0).getKey();  System.***out***.println("key:"+key +"--->"+"count:"+count +"source\_number\_province:"+source\_number\_province);  } |