目录

[一、 一般查询 2](#_Toc512091644)

[（一） matchAllQuery(client) 2](#_Toc512091645)

[（二） matchQuery(client); 3](#_Toc512091646)

[（三） multiMatchQuery(client); 3](#_Toc512091647)

[（四） wildcardQuery()模糊查询 3](#_Toc512091648)

[（五） commonTermQuery(client); 3](#_Toc512091649)

[（六） termQuery(client); 4](#_Toc512091650)

[（七） testPrefixQuery前缀 4](#_Toc512091651)

[（八） *rangeQuery*(client); 范围查询 4](#_Toc512091652)

[1、 两种写法 5](#_Toc512091653)

[（九） nested query 5](#_Toc512091654)

[（十） 其他查询 6](#_Toc512091655)

[二、 聚合查询AggsQueryTest 7](#_Toc512091656)

[（一） avgQuery(client); 7](#_Toc512091657)

[（二） minQuery(client); 8](#_Toc512091658)

[（三） maxQuery(client) 8](#_Toc512091659)

[（四） valueCountQuery(client); //统计个数 8](#_Toc512091660)

[值计算聚合 8](#_Toc512091661)

[（五） extendedStatsQuery(client);//统计聚合(一堆) 8](#_Toc512091662)

[（六） percentileQuery(client) 9](#_Toc512091663)

[（七） percentileRankQuery(client);//百分比 9](#_Toc512091664)

[（八） *rangeQuery*(client)//范围 9](#_Toc512091665)

[（九） histogramQuery(client);//柱状图聚合 10](#_Toc512091666)

[（十） dateHistogramQuery(client);// 按日期间隔分组 10](#_Toc512091667)

[（十一） 获取聚合里面的结果 10](#_Toc512091668)

[（十二） 嵌套的聚合 10](#_Toc512091669)

[（十三） 反转嵌套 10](#_Toc512091670)

[三、 二级分组的例子： 10](#_Toc512091671)

[四、 嵌套查询 11](#_Toc512091672)

[（一） constantScoreQuery(client); 11](#_Toc512091673)

[（二） *booQuery*(client)（最常用） 12](#_Toc512091674)

[1、 经典案例 12](#_Toc512091675)

[（三） disMaxQuery(client); 13](#_Toc512091676)

[五、 本案例数据导入 14](#_Toc512091677)

# 一般查询

## matchAllQuery(client)

matchAllQuery()方法用来匹配全部文档

|  |
| --- |
| public static void matchAllQuery(Client client ) {  SearchResponse res = null;  QueryBuilder qb = QueryBuilders.matchAllQuery();    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());  } |

for有选择的打印

|  |
| --- |
| 1. **for** (SearchHit searchHit : searchHits) { 2. String name = (String) searchHit.getSource().get("name"); 3. String birth = (String) searchHit.getSource().get("birth"); 4. String interest = (String) searchHit.getSource().get("interest"); 5. System.out.println("-------------" + (++i) + "------------"); 6. System.out.println(name); 7. System.out.println(birth); 8. System.out.println(interest); 9. } |

## matchQuery(client);

不能写为matchQuery("name", "to\*")

matchQuery("filedname","value")匹配单个字段，匹配字段名为filedname,值为value的文档

|  |
| --- |
| QueryBuilder qb = QueryBuilders.matchQuery("title", "article"); |

## multiMatchQuery(client);

多个字段匹配某一个值

1. QueryBuilder queryBuilder = QueryBuilders.multiMatchQuery("music",
2. "name", "interest");//搜索name中或interest中包含有music的文档（必须与music一致）

## **wildcardQuery()模糊查询**

模糊查询，?匹配单个字符，\*匹配多个字符

**[java]** [view plain](https://blog.csdn.net/lom9357bye/article/details/52852533) [copy](https://blog.csdn.net/lom9357bye/article/details/52852533)

1. WildcardQueryBuilder queryBuilder = QueryBuilders.wildcardQuery("name",
2. "\*jack\*");//搜索名字中含有jack文档（name中只要包含jack即可）

## commonTermQuery(client);

一种略高级的查询，充分考虑了stop-word的低优先级，提高了查询精确性。

将terms分为了两种：more-importent（low-frequency） and less important（high-frequency）。less-important比如stop-words，eg：the and。

QueryBuilder qb = QueryBuilders

.commonTermsQuery("title","article");

## termQuery(client);

\* termQuery("key", obj) 完全匹配

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

QueryBuilder qb = QueryBuilders

.*termQuery*("title","article");

// QueryBuilder qb = QueryBuilders

// .termsQuery("title","article","relevence");

## testPrefixQuery前缀

参考网址：https://www.cnblogs.com/wenbronk/p/6432990.html

/\*\*

\* 前缀查询

\*/

@Test

public void testPrefixQuery() {

QueryBuilder queryBuilder = QueryBuilders.matchQuery("user", "kimchy");

searchFunction(queryBuilder);

}

## *rangeQuery*(client); 范围查询

// 闭区间 QueryBuilder query = QueryBuilders.rangeQuery("age").from(10).to(20); // 开区间 QueryBuilder query = QueryBuilders.rangeQuery("age").gt(10).lt(20);

### 两种写法

QueryBuilder qb = QueryBuilders

.rangeQuery("like")

.gte(5)

.lt(7);

// QueryBuilder qb = QueryBuilders

// .rangeQuery("like")

// .from(5)

// .to(7)

// .includeLower(true) // 包含上届

// .includeUpper(false); // 包含下届

## nested query

在关系查询中，存在一对多和多对一的关系。因为就会出现两种查询情况。

在解释查询关系之前，需要理解一下Relationship Name，如文档中contact和account的关系  ，一个Account会有多个contact，一个Contact也会有多个Account，但是最终归结的关系为Account对contact的关系为一对多。也就是说 在contact上保存有对account'的引用，这个引用的名称就是RelationshipName（区别于field name），类似于外键的名称。

下面介绍两种查询

1、多对一的查询。

      salesforce 中特有的\_\_r模式，直接关联到parent上，如contact上存有对account的引用，那么我可以直接关联出account上的相关字段。

**[sql]** [view plain](https://blog.csdn.net/gaofly89/article/details/44854927) [copy](https://blog.csdn.net/gaofly89/article/details/44854927)

1. **select** id,**name** ,account.**name**,account.id **from** contact

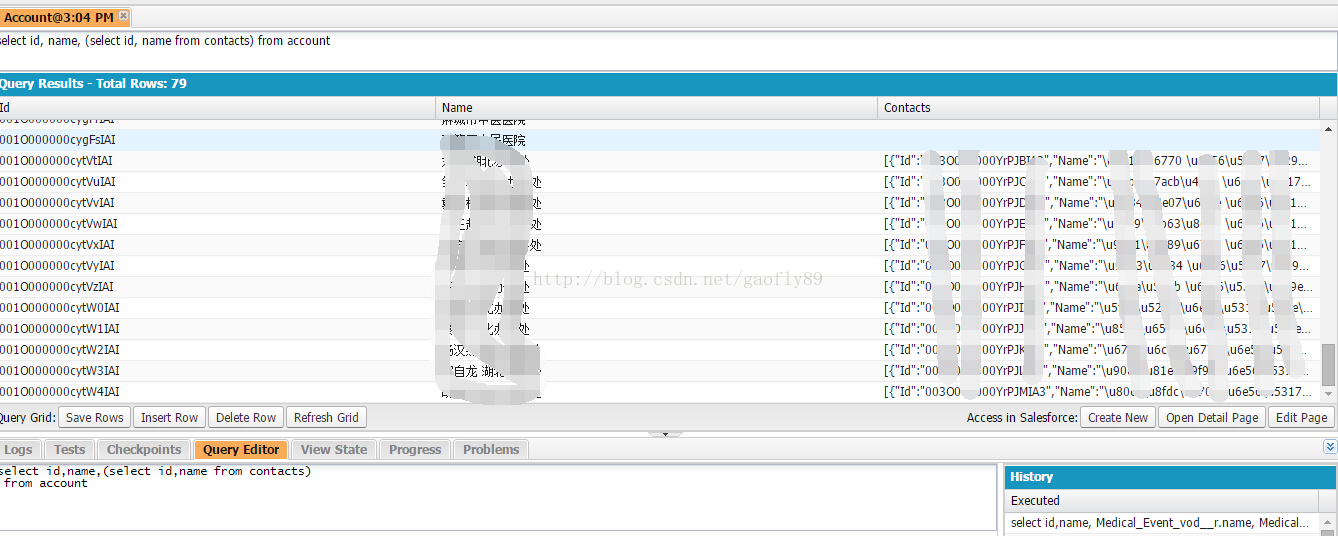
2、一对多的查询

     嵌入式查询(nested query),这种方式适合在父的一端查询相关子的记录。如：我想查找到负责这个account的全部contact。

**[sql]** [view plain](https://blog.csdn.net/gaofly89/article/details/44854927) [copy](https://blog.csdn.net/gaofly89/article/details/44854927)

1. **select** id,**name**,(**select** id,**name** **from** contacts)
2. **from** account

查询结果如图：



这样就会关联出所以的contact数据，contact部分的展示形式json串。**注意contacts不是对象名称，是Relationshipname**

## 其他查询

QueryBuilder qb = QueryBuilders.*existsQuery*("str");

//QueryBuilder qb = QueryBuilders.prefixQuery("name", "prefix");

//QueryBuilder qb = QueryBuilders.regexpQuery("user", "k.\*y");

正则表达式

|  |
| --- |
| /\*\*  \* 模糊查询  \* 不能用通配符, 不知道干啥用  \*/  //QueryBuilder qb = QueryBuilders.fuzzyQuery("name", "kimzhy"); |

//QueryBuilder qb = QueryBuilders.typeQuery("my\_type");

|  |
| --- |
| /\*\*  \* 只查询一个id的  \* QueryBuilders.idsQuery(String...type).ids(Collection<String> ids)  \*/  //QueryBuilder qb = QueryBuilders.idsQuery("my\_type","type2").addIds("1","2","5"); |

# 聚合查询AggsQueryTest

## avgQuery(client);

public static void avgQuery(Client client ) {

SearchResponse res = null;

AvgBuilder agg = AggregationBuilders

.avg("avg\_num")

.field("like");

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

.setTypes("article")

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

.addAggregation(agg)

.setFrom(0)

.setSize(10)

.execute().actionGet();

System.out.println(res);

// on shutdown

client.close();

}

## minQuery(client);

MinBuilder agg = AggregationBuilders

.min("min\_num")

.field("like");

## maxQuery(client)

MaxBuilder agg = AggregationBuilders

.max("max\_num")

.field("like");

## valueCountQuery(client); //统计个数

### 值计算聚合

SearchResponse res = null;

ExtendedStatsBuilder agg = AggregationBuilders

.extendedStats("extended\_stats\_num")

.field("like");

## extendedStatsQuery(client);//统计聚合(一堆)

返回聚合分析后所有指标，比Stats多三个统计结果：平方和、方差、标准差

|  |  |
| --- | --- |
| 1  2  3  4  5 | {      "aggs" : {      "grades\_stats" : { "extended\_stats" : { "field" : "grade" } }      }  } |

ExtendedStatsBuilder agg = AggregationBuilders.*extendedStats*("extended\_stats\_num").field("like");

## percentileQuery(client)

PercentilesBuilder agg = AggregationBuilders

.percentiles("percentile\_num")

.field("like")

.percentiles(95,99,99.9);

## percentileRankQuery(client);//百分比

PercentileRanksBuilder agg = AggregationBuilders

.percentileRanks("percentile\_rank\_num")

.field("like")

.percentiles(3,5);

## *rangeQuery*(client)//范围

AggregationBuilder agg =

AggregationBuilders

.range("agg")

.field("like")

.addUnboundedTo(3)

.addRange(3, 5)

.addUnboundedFrom(5);

## histogramQuery(client);//柱状图聚合

## dateHistogramQuery(client);// 按日期间隔分组

## 获取聚合里面的结果

TopHitsBuilder thb= AggregationBuilders.topHits("top\_result");

## 嵌套的聚合

NestedBuilder nb= AggregationBuilders.nested("negsted\_path").path("quests");

## 反转嵌套

AggregationBuilders.reverseNested("res\_negsted").path("kps ");

# 二级分组的例子：

上面这些基本就是常用的聚合查询了，在嵌套（nested）下面的子聚合查询就是嵌套查询了，除了嵌套查询，其他的聚合查询也可以无限级添加子查询

举个例子

SearchRequestBuilder **search** = client.prepareSearch("index").setTypes("type");

TermsBuilder **one**= AggregationBuilders.terms("group\_name").field("name");

TermsBuilder **two**= AggregationBuilders.terms("group\_age").field("age");

**one**.subAggregation(**two**)

**search**.addAggregation(**one**);

Terms terms= **search**.get().getAggregations().get("group\_name");

**for**(Terms.Bucket name\_buk:terms.getBuckets()){

*//一级分组的内容*

Terms terms\_age= name\_buk.getAggregations().get("group\_age");

**for**(Terms.Bucket age\_buk:terms\_age.getBuckets()){

*//二级分组的内容*

System.**out**.println(name\_buk.getKey()+" "+age\_buk.getKey()+" "+age\_buk.getDocCount());

}

}

# 嵌套查询

## constantScoreQuery(client);

/\*\*

\* 包裹查询, 高于设定分数, 不计算相关性

\*/

@Test

public void testConstantScoreQuery() {

QueryBuilder queryBuilder = QueryBuilders.constantScoreQuery(QueryBuilders.termQuery("name", "kimchy")).boost(2.0f);

searchFunction(queryBuilder);

## *booQuery*(client)（最常用）

/\*\*

\* 组合查询

\* must(QueryBuilders) : AND

\* mustNot(QueryBuilders): NOT

\* should: : OR

\*/

**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());

}

### 经典案例

如果需要查询(addr = Beijing) && (sex = false) && (10 < age < 20)的doc：

public static QueryBuilder createQuery() {

BoolQueryBuilder query = QueryBuilders.boolQuery();

// addr = Beijing

query.must(new QueryStringQueryBuilder("Beijing").field("addr"));

// sex = falese

query.must(new QueryStringQueryBuilder("false").field("sex"));

// age ∈ (10, 20)

query.must(new RangeQueryBuilder("age").gt(10).lt(20));

return query;

}

返回结果：

{"pid":168,"age":16,"sex":false,"name":"Tom","addr":"Beijing"}

{"pid":276,"age":19,"sex":false,"name":"Bill","addr":"Beijing"}

{"pid":565,"age":16,"sex":false,"name":"Brown","addr":"Beijing"}

{"pid":73,"age":13,"sex":false,"name":"David","addr":"Beijing"}

作者：唐影若凡  
链接：https://www.jianshu.com/p/a3694b13bf89  
來源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## disMaxQuery(client);

/\*\*

\* disMax查询

\* 对子查询的结果做union, score沿用子查询score的最大值,

\* 广泛用于muti-field查询

\*/

@Test

public void testDisMaxQuery() {

QueryBuilder queryBuilder = QueryBuilders.disMaxQuery()

.add(QueryBuilders.termQuery("user", "kimch")) // 查询条件

.add(QueryBuilders.termQuery("message", "hello"))

.boost(1.3f)

.tieBreaker(0.7f);

searchFunction(queryBuilder);

}

# 本案例数据导入

curl -XPUT 'http://169.254.135.217:9200/search\_test/' -d '{

"settings" : {

"index" : {

"number\_of\_shards" : 3,

"number\_of\_replicas" : 1

}

},

"mappings" : {

"article" : {

"properties" : {

"title" : { "type" : "string"},

"body" : { "type" : "string"},

"like" : { "type" : "long"},

"publish\_date" : { "type" : "date"}

}

}

}

}'

curl -XGET 'http://169.254.135.217:9200/search\_test/\_mapping?pretty'

curl -XGET 'http://169.254.135.217:9200/search\_test/\_mapping/article?pretty'

curl -XHEAD -i 'http://169.254.135.217:9200/search\_test/article'

/search\_test/article/1

{

"title": "What's relevance?",

"body": "atticle body of relevence:Term frequency/inverse document frequency",

"like": "1",

"publish\_date": "2016-03-24"

}

/search\_test/article/2

{

"title": "article 02",

"body": "article 02 atticle body of relevence:Term frequency/inverse document frequency",

"like": "2",

"publish\_date": "2016-05-24"

}

/search\_test/article/3

{

"title": "article 03",

"body": "article 03 atticle body of relevence:Term frequency/inverse document frequency",

"like": "3",

"publish\_date": "2016-07-24"

}

/search\_test/article/4

{

"title": "article 04",

"body": "article 04 atticle body of relevence:Term frequency/inverse document frequency",

"like": "4",

"publish\_date": "2016-09-24"

}

/search\_test/article/5

{

"title": "article 05",

"body": "article 04 atticle body of relevence:Term frequency/inverse document frequency",

"like": "5",

"publish\_date": "2016-11-24"

}

/search\_test/article/6

{

"title": "Quick brown rabbits",

"body": "Brown rabbits are commonly seen.",

"like": "6",

"publish\_date": "2016-12-24"

}

/search\_test/article/7

{

"title": "Keeping pets healthy",

"body": "My quick brown fox eats rabbits on a regular basis.",

"like": "7",

"publish\_date": "2017-11-24"

}