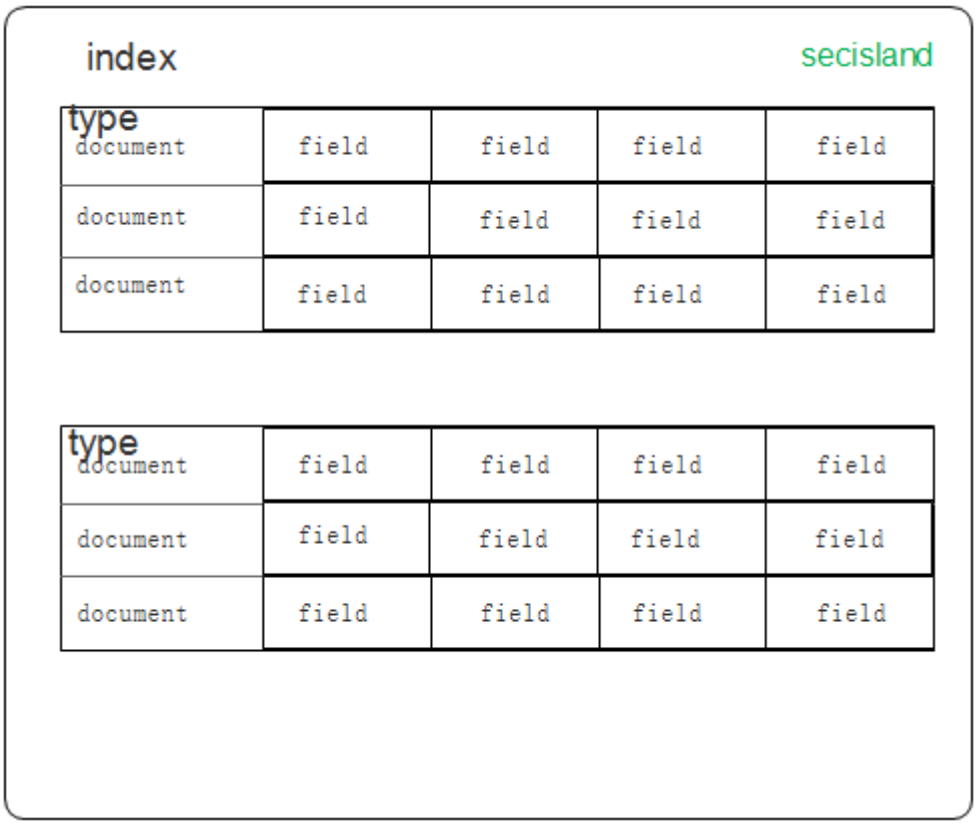
es深入搜索语句

相关的搜索语句，以及对比总结

0.了解es的结构和这货到底是啥



ES的定义

Elasticsearch 是一个兼有搜索引擎和NoSQL数据库功能的开源系统，
基于Java/Lucene构建，可以用于全文搜索，结构化搜索以及近实时分析。

可以说Lucene是当今最先进，最高效的全功能开源搜索引擎框架。 说明： Lucene：只是一个框架，要充分利用它的功能，需要使用JAVA，并且在程序中集成Lucene，学习成本高，Lucene确实非常复杂。 Elasticsearch 是 面向文档型数据库，这意味着它存储的是整个对象或者 文档，它不但会存储它们，还会为他们建立索引，这样你就可以搜索他们了

归纳定义2

直观感受：
是个开源系统
具有全局搜索引擎
具有面向文档型的数据库，能实现nosql数据库功能

实际效果基本就是lucene的简化版，简化了lucene的学习成本

1.es查询需要了解的一些前提部分

1.1（1）请求方式的大分类

请求参数

eg:

```
#获得按年龄排序的每页页数为2的第二的数据
GET /zs1/_doc/_search?q=*&sort=age:desc&size=2&from=2
GET /zs1/_doc/_search?q=*&sort=age:desc
```

请求体

eg:

```
GET /zs1/_doc/_search
{
  "query":{
    "match":{
      "title":"王五"
    }
  },
  "highlight":{
    "fields":{
    "title":{}
  }
  }
}
```

1.1（2）请求的另一种大分类

查询 计算score分数

过滤 不计算score分数

1.2返回的内容的详细参数

took：是查询花费的时间，毫秒单位

time_out：标识查询是否超时

_shards：描述了查询分片的信息，查询了多少个分片、成功的分片数量、失败的分片数量等

hits：搜索的结果，total是全部的满足的文档数目，hits是返回的实际数目（默认是10）

_score是文档的分数信息，与排名相关度有关，参考各大搜索引擎的搜索结果，就容易理解。

2DSL查询语句

查询的通用格式

```
GET /index/type/_search

{

    ... ...

}
```

通用格式内的内容

查询：query

```
"query":{
    ... ...
}

eg：查询所有
"query":{
"match_all":{}
}
```

基于

```
GET /index/type/_search

{

    "query":{
    ... ...
    }

}
```

query内的几种：

1关键字查找match

```
#match这里是比如查的是张顺真帅  能查到这几个字拆分出去的词 ： 注意是词和字都行
张顺真帅 = 查询：张顺 or 真帅 or 张顺真帅 or 张 or 顺 or 真 or 帅
"match":{
    "property":"... ..."
```

```
}
#match_phrase这里是相当于精准查找，只有所有词组都在的时候才行
张顺真帅 = 查询：张顺 and 真帅 or 张顺真帅
"match_phrase":{
    "property":"... ..."
}
##对比上一条，多了可调节因子，少匹配一个也满足
"match_phrase":{
    "property":"... ..."
    "slop":1
}
###了解 multi_match 更灵活的组合匹配方式

#全查
"match_all":{}
```

2关键字查找term

```
#term是代表完全匹配，即不进行分词器分析，文档中必须包含整个搜索的词汇
张顺真帅 = 张顺真帅
"term":{
    "property":"... ..."
}
#terms用户多词条匹配
"term":{
    "property":"... ...", "... ..."
}
```

3关键字查找range

```
#范围
gt = >
gte = >=
lt = <
lte = <=

    "range":{
        "property":{
            "gte":"...",
            "lte":"..."
        }
    }
```

4关键字查找prefix

```
#基于前缀的搜索
"prefix":{
    "property":"..."
}
```

5关键字查找wildcard

```
#基于通配符的查找
"wildcard":{
    "property":"..."
}
```

6关键字查找ids

```
#基础ids 也就是_id进行查找
"ids":[x,x,x]
```

7模糊查找fuzzy

```
"fuzzy":{
    "property":"..."
}
```

8正则regexp

```
"regexp":{
    "property":"正则表达式"
}
```

9多条件查找bool->must or should or must_not;

```
#基于Boolean的查询（多条件查询）
must ： 查询结果必须符合该查询条件（列表）。
should ： 类似于or的查询条件。
must_not ： 查询结果必须不符合查询条件（列表）
```

```
"bool":{
    "must or should or must_not":[
        {
            ...
        },
        {
            ...
        }
    ]
}
```

```
#省略号的内容就是其他关键字的使用，比如单查和模糊查询之类的
```

排序：

排序：sort

```
"sort":{
    "property":"asc or desc"
}
```

分页：from and size

```
**url实现：
GET /zpark/user/_search?q=*&sort=_id:asc&from=第几页&size=每页的文档个数


"query"{
"match_all":{}
},
"sort":{
    "property":"asc or desc"
}
 "from":integer ,
 "size":integer
```

属性高亮：highlight

```
"highlight":{
    "fields":{
        "property":{}#指明高亮的属性
    }
}
```

总结

```
GET /index(索引名-相当于数据库)/type（类型-相当与表，去查document文档，也就是表中的行）/_search(查询操作)
{
    "query or sort or highlight or from or size":{
        "match or match_all or match_phrase or term or range or prifix or wildcard or ids or fuzzy or regexp or bool":{
            "property(属性)": "对应的值"
        }
    }
}


例外
"ids":[x,x,x]

bool:{
    "must or should or must_not":[
    {
        "对应的匹配条件"
    }
]
}
```

过滤：归属于查询的一种

过滤基于的查询结构

过滤缓存位集，但是不像一般查询那样给的分score

过滤用于大范围的数据筛选， 查询用于小范围的精准数据筛选

```
GET /index(索引名-相当于数据库)/type (类型-相当与表，去查document文档，也就是表中的行) /_search(查询操作)
{
    "query":{
        "bool":{
            "must or should or must_not":[
                {
                    "对应的匹配条件"
                }
            ],
            ....
        }
    }
}
```

省略号部分的结构

```
"filter":{
    "...":{
     "property":"..." or [x,x,x]
    }
}
```

再简化

```
"filter":{
    ...
}
```

继续省略号部分的结构

term和terms

同之前的

range

同之前的

ids

同之前的

exists:

这个字段是否存在

```
"exists":{
    "field":"property"
}
```

聚合

使用之前需要了解的东西

聚合提供了功能可以分组并统计你的数据。理解聚合最简单的方式就是可以把它粗略的看做SQL的GROUP BY操作和 SQL的聚合函数。

ES中常用的聚合：

metric（度量）聚合：

度量类型聚合主要针对的number类型的数据，需要ES做比较多的计算工作

bucketing（桶）聚合：

划分不同的“桶”，将数据分配到不同的“桶”里。非常类似sql中的group语句的含义

metric（度量）聚合

具体分类

avg

sum

max

min

stats

通用结构

```
GET or POST /index/type/_search
#以上就是傻屌的标准开头
{
    "aggs or aggregations(这俩都行)": {
        "起个名": {
            "具体的操作: avg or sum or max or min (stats就可以把前面的全显示出来)": {
                "field":"property"
            }
        }
    }
}
```

bucketing（桶）聚合

通用结构

```
GET or POST /index/type/_search
#以上就是傻屌的标准开头
{
    "aggs or aggregations(这俩都行)": {
        "起个名": {

        }
    }
}
```

关于具体的操作

```
"具体的操作: ": {
                ...
            }
```

具体分类

range

```
#通用结构
"range":{
    "field":"property",
    "ranges":[{"from":... ,
               "to":...}
              ,{},{}
             ]
}

eg:
GET /zs1/_doc/_search
{
  "aggs":{
    "平均年龄":{
      "range":{
        "field":"age",
        "ranges":[
          {
            "from":1,
            "to":100
        },{
            "from":101,
            "to":200
        },{
            "from":201,
            "to":300

        }]
```

```
      }
    }
  }
}
```

terms

```
#精准的查找某个属性的某个属性的数量（或者别的因为暂时只用到了size）
"terms":{
    "field":"property",
    "size":integer
}


eg:
GET /zs1/_doc/_search
{
  "aggs":{
    "平均年龄":{
      "terms":{
        "field":"age",
        "size":4
      }
    }
  }
}


result:
"aggregations": {
    "平均年龄": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": 181,
          "doc_count": 3
        },
        {
          "key": 18,
          "doc_count": 2
        },
        {
          "key": 22,
          "doc_count": 1
        }
      ]
    }
  }
```

```
}
```

date_range

```
#某个data类型的属性的某个范围内的document的数目
"date_range":{
    "field":"property",
    "format":"yyyy-MM-dd",
    "ranges":[{
        "from":"now/y +- ny",    #n是任意integer类型 now表示当年1月1号
         "to":"now"
    },{},{}
              ]
}
```

histogram

```
#直方图 interval 表示直方图数据之间的间隔

"histogram":{
    "field":"property",
    "interval":integer
}


eg:
GET /zs1/_doc/_search
{
  "aggs":{
    "直方图":{
      "histogram":{
        "field":"age",
        "interval":100
      }
    }
  }
}

result:

"aggregations": {
    "直方图": {
      "buckets": [
        {
          "key": 0,
          "doc_count": 3

        },
```

```
        {
          "key": 100,
          "doc_count": 3
        }
      ]
    }
  }
}
```

date_histogram

```
#和时间有关的属性的直方图的绘制 interval还是间隔 不过间隔变成了year month day
#如果最后锁定的直方图是在同一年内，那么选择year也是按month分类
#如果是选的month ，但是数据在全一个月内，那么就会自动按天数划分

"date_histogram":{
    "field":"property",
    "format":"yyyy-MM-dd",
    "interval":"year or month or day"
}

eg:
GET /zs1/_doc/_search
{
  "aggs":{
    "和日期相关的直方图":{
      "date_histogram":{
        "field":"created",
        "interval":"year",
        "format":"yyyy-MM-dd"
      }
    }
  }
}


result:
#数据太长截取一部分
 {
          "key_as_string": "2011-01-01",
          "key": 1293840000000,
          "doc_count": 0
        },
        {
          "key_as_string": "2012-01-01",
          "key": 1325376000000,
          "doc_count": 0
        },

        {
```

```
        "key_as_string": "2013-01-01",
        "key": 1356998400000,
        "doc_count": 0
      },
      {
        "key_as_string": "2014-01-01",
        "key": 1388534400000,
        "doc_count": 0
      },
      {
        "key_as_string": "2015-01-01",
        "key": 1420070400000,
        "doc_count": 0
      },
      {
        "key_as_string": "2016-01-01",
        "key": 1451606400000,
        "doc_count": 0
      },
      {
        "key_as_string": "2017-01-01",
        "key": 1483228800000,
        "doc_count": 0
      },
      {
        "key_as_string": "2018-01-01",
        "key": 1514764800000,
        "doc_count": 5
      }
    ]
  }
```