> 主讲老师：Fox
> 有道云笔记地址：https://note.youdao.com/s/ICVDmP4d

1. 聚合的概述

Elasticsearch除搜索以外，提供了针对ES 数据进行统计分析的功能。聚合(aggregations)可以让我们极其方便的实现对数据的统计、分析、运算。例如：

- 什么品牌的手机最受欢迎?
- 这些手机的平均价格、最高价格、最低价格?
- 这些手机每月的销售情况如何?

使用场景

聚合查询可以用于各种场景，比如商业智能、数据挖掘、日志分析等等。

- 电商平台的销售分析：统计每个地区的销售额、每个用户的消费总额、每个产品的销售量等，以便更好地了解销售情况和趋势。
- 社交媒体的用户行为分析：统计每个用户的发布次数、转发次数、评论次数等，以便更好地了解用户行为和趋势，同时可以将数据按照地区、时间、话题等维度进行分析。
- 物流企业的运输分析：统计每个区域的运输量、每个车辆的运输次数、每个司机的行驶里程等，以便更好地了解运输情况和优化运输效率。
- 金融企业的交易分析：统计每个客户的交易总额、每个产品的销售量、每个交易员的业绩等，以便更好地了解交易情况和优化业务流程。
- 智能家居的设备监控分析：统计每个设备的使用次数、每个家庭的能源消耗量、每个时间段的设备使用率等，以便更好地了解用户需求和优化设备效能。

基本语法

聚合查询的语法结构与其他查询相似，通常包含以下部分：

- 查询条件：指定需要聚合的文档，可以使用标准的 Elasticsearch 查询语法，如 term、match、range 等等。
- 聚合函数：指定要执行的聚合操作，如 sum、avg、min、max、terms、date_histogram 等等。每个聚合命令都会生成一个聚合结果。
- 聚合嵌套：聚合命令可以嵌套，以便更细粒度地分析数据。

```
GET <index_name>/_search
{
  "aggs": {
    "<aggs_name>": { // 聚合名称需要自己定义
      "<agg_type>": {
        "field": "<field_name>"
      }
    }
  }
}
```

- aggs_name：聚合函数的名称
- agg_type：聚合种类，比如是桶聚合（terms）或者是指标聚合（avg、sum、min、max等）
- field_name：字段名称或者叫域名。

2. 聚合的分类

- Metric Aggregation：一些数学运算，可以对文档字段进行统计分析，类比Mysql中的 min(), max(), sum() 操作。

```
SELECT MIN(price), MAX(price) FROM products
#Metric聚合的DSL类比实现：
{
    "aggs":{
        "avg_price":{
            "avg":{
                "field":"price"
            }
        }
    }
}
```

- Bucket Aggregation： 一些满足特定条件的文档的集合放置到一个桶里，每一个桶关联一个key，类比Mysql中的group by操作。

```
ELECT size COUNT(*) FROM products GROUP BY size
#bucket聚合的DSL类比实现：
{
 "aggs": {
    "by_size": {
      "terms": {
        "field": "size"
      }
   }
}
```

- Pipeline Aggregation：对其他的聚合结果进行二次聚合

示例数据

```
DELETE /employees
#创建索引库
PUT /employees
{
  "mappings": {
    "properties": {
      "age":{
        "type": "integer"
      },
      "gender":{
        "type": "keyword"
      },
      "job":{
         "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 50
            }
          }
      },
      "name":{
        "type": "keyword"
      },
      "salary":{
        "type": "integer"
      }
    }
  }
}

PUT /employees/_bulk
{ "index" : {  "_id" : "1" } }
{ "name" : "Emma","age":32,"job":"Product Manager","gender":"female","salary":35000 }
{ "index" : {  "_id" : "2" } }
{ "name" : "Underwood","age":41,"job":"Dev Manager","gender":"male","salary": 50000}
{ "index" : {  "_id" : "3" } }
{ "name" : "Tran","age":25,"job":"Web Designer","gender":"male","salary":18000 }
{ "index" : {  "_id" : "4" } }
```

```
{ "name" : "Rivera","age":26,"job":"Web Designer","gender":"female","salary": 22000}
{ "index" : {  "_id" : "5" } }
{ "name" : "Rose","age":25,"job":"QA","gender":"female","salary":18000 }
{ "index" : {  "_id" : "6" } }
{ "name" : "Lucy","age":31,"job":"QA","gender":"female","salary": 25000}
{ "index" : {  "_id" : "7" } }
{ "name" : "Byrd","age":27,"job":"QA","gender":"male","salary":20000 }
{ "index" : {  "_id" : "8" } }
{ "name" : "Foster","age":27,"job":"Java Programmer","gender":"male","salary": 20000}
{ "index" : {  "_id" : "9" } }
{ "name" : "Gregory","age":32,"job":"Java Programmer","gender":"male","salary":22000 }
{ "index" : {  "_id" : "10" } }
{ "name" : "Bryant","age":20,"job":"Java Programmer","gender":"male","salary": 9000}
{ "index" : {  "_id" : "11" } }
{ "name" : "Jenny","age":36,"job":"Java Programmer","gender":"female","salary":38000 }
{ "index" : {  "_id" : "12" } }
{ "name" : "Mcdonald","age":31,"job":"Java Programmer","gender":"male","salary": 32000}
{ "index" : {  "_id" : "13" } }
{ "name" : "Jonthna","age":30,"job":"Java Programmer","gender":"female","salary":30000
}
{ "index" : {  "_id" : "14" } }
{ "name" : "Marshall","age":32,"job":"Javascript Programmer","gender":"male","salary":
25000}
{ "index" : {  "_id" : "15" } }
{ "name" : "King","age":33,"job":"Java Programmer","gender":"male","salary":28000 }
{ "index" : {  "_id" : "16" } }
{ "name" : "Mccarthy","age":21,"job":"Javascript Programmer","gender":"male","salary":
16000}
{ "index" : {  "_id" : "17" } }
{ "name" : "Goodwin","age":25,"job":"Javascript Programmer","gender":"male","salary":
16000}
{ "index" : {  "_id" : "18" } }
{ "name" : "Catherine","age":29,"job":"Javascript
Programmer","gender":"female","salary": 20000}
{ "index" : {  "_id" : "19" } }
{ "name" : "Boone","age":30,"job":"DBA","gender":"male","salary": 30000}
{ "index" : {  "_id" : "20" } }
{ "name" : "Kathy","age":29,"job":"DBA","gender":"female","salary": 20000}
```

指标聚合（Metric Aggregation）

- 单值分析：只输出一个分析结果
  - min, max, avg, sum
  - Cardinality（类似distinct Count)
- 多值分析:输出多个分析结果
  - stats（统计）, extended stats
  - percentile （百分位）, percentile rank
  - top hits(排在前面的示例)

查询员工的最低最高和平均工资

```
#多个 Metric 聚合，找到最低最高和平均工资
POST /employees/_search
{
  "size": 0,
  "aggs": {
    "max_salary": {
      "max": {
        "field": "salary"
      }
    },
    "min_salary": {
      "min": {
        "field": "salary"
      }
    },
    "avg_salary": {
      "avg": {
        "field": "salary"
      }
    }
  }
}
```

对salary进行统计

```
# 一个聚合，输出多值
POST /employees/_search
{
  "size": 0,
  "aggs": {
    "stats_salary": {
      "stats": {
        "field":"salary"
      }
    }
  }
}
```

cardinate对搜索结果去重

```
POST /employees/_search
{
  "size": 0,
  "aggs": {
    "cardinate": {
      "cardinality": {
        "field": "job.keyword"
      }
    }
  }
}
```

桶聚合（Bucket Aggregation）

按照一定的规则，将文档分配到不同的桶中，从而达到分类的目的。ES提供的一些常见的 Bucket Aggregation。

- Terms，需要字段支持filedata

    - keyword 默认支持fielddata
    - text需要在Mapping 中开启fielddata，会按照分词后的结果进行分桶
- 数字类型
    - Range / Data Range
    - Histogram（直方图） / Date Histogram
- 支持嵌套: 也就在桶里再做分桶

桶聚合可以用于各种场景，例如：

- 对数据进行分组统计，比如按照地区、年龄段、性别等字段进行分组统计。
- 对时间序列数据进行时间段分析，比如按照每小时、每天、每月、每季度、每年等时间段进行分析。
- 对各种标签信息分类，并统计其数量。

获取job的分类信息

```
# 对keword 进行聚合
GET /employees/_search
{
  "size": 0,
  "aggs": {
    "jobs": {
      "terms": {
        "field":"job.keyword"
      }
    }
  }
}
```

聚合可配置属性有：

- field：指定聚合字段
- size：指定聚合结果数量
- order：指定聚合结果排序方式

默认情况下，Bucket聚合会统计Bucket内的文档数量，记为_count，并且按照_count降序排序。我们可以指定order属性，自定义聚合的排序方式：

```
GET /employees/_search
{
  "size": 0,
  "aggs": {
    "jobs": {
      "terms": {
        "field":"job.keyword",
         "size": 10,
        "order": {
          "_count": "desc"
        }
      }
    }
  }
}
```

限定聚合范围

```
#只对salary在10000元以上的文档聚合
GET /employees/_search
{
  "query": {
    "range": {
      "salary": {
        "gte": 10000
      }
    }
  },
  "size": 0,
  "aggs": {
    "jobs": {
      "terms": {
        "field":"job.keyword",
         "size": 10,
        "order": {
          "_count": "desc"
        }
      }
    }
  }
}
```

注意：对 Text 字段进行 terms 聚合查询，会失败抛出异常

```
POST /employees/_search
{
  "size": 0,
  "aggs": {
    "jobs": {
      "terms": {
        "field":"job"
      }
    }
  }
}
```

解决办法：对 Text 字段打开 fielddata，支持terms aggregation

```
PUT /employees/_mapping
{
  "properties" : {
    "job":{
       "type":  "text",
       "fielddata": true
    }
  }
}

# 对 Text 字段进行分词，分词后的terms
POST /employees/_search
{
  "size": 0,
  "aggs": {
    "jobs": {
      "terms": {
        "field":"job"
      }
    }
  }
}
```

对job.keyword 和 job 进行 terms 聚合，分桶的总数并不一样

```
POST /employees/_search
{
  "size": 0,
  "aggs": {
    "cardinate": {
      "cardinality": {
        "field": "job"
      }
    }
  }
}
```

Range & Histogram聚合

- 按照数字的范围，进行分桶
- 在Range Aggregation中，可以自定义Key

Range 示例：按照工资的 Range 分桶

```
Salary Range分桶，可以自己定义 key
POST employees/_search
{
  "size": 0,
  "aggs": {
    "salary_range": {
      "range": {
        "field":"salary",
        "ranges":[
          {
            "to":10000
          },
          {
            "from":10000,
            "to":20000
          },
          {
            "key":">20000",
            "from":20000
          }
        ]
      }
    }
  }
}
```

Histogram示例：按照工资的间隔分桶

```
#工资0到10万，以 5000一个区间进行分桶
POST employees/_search
{
  "size": 0,
  "aggs": {
    "salary_histrogram": {
      "histogram": {
        "field":"salary",
        "interval":5000,
        "extended_bounds":{
          "min":0,
          "max":100000
        }
      }
    }
  }
}
```

top_hits应用场景: 当获取分桶后，桶内最匹配的顶部文档列表

```
# 指定size，不同工种中，年纪最大的3个员工的具体信息
POST /employees/_search
{
  "size": 0,
  "aggs": {
    "jobs": {
      "terms": {
        "field":"job.keyword"
      },
      "aggs":{
        "old_employee":{
          "top_hits":{
            "size":3,
            "sort":[
              {
                "age":{
                  "order":"desc"
                }
              }
            ]
          }
        }
      }
    }
  }
}

```

嵌套聚合示例

```
# 嵌套聚合1，按照工作类型分桶，并统计工资信息
POST employees/_search
{
  "size": 0,
  "aggs": {
    "Job_salary_stats": {
      "terms": {
        "field": "job.keyword"
      },
      "aggs": {
        "salary": {
          "stats": {
            "field": "salary"
          }
        }
      }
    }
  }
}

# 多次嵌套。根据工作类型分桶，然后按照性别分桶，计算工资的统计信息
POST employees/_search
{
  "size": 0,
  "aggs": {
    "Job_gender_stats": {
      "terms": {
        "field": "job.keyword"
      },
      "aggs": {
        "gender_stats": {
          "terms": {
            "field": "gender"
          },
          "aggs": {
            "salary_stats": {
              "stats": {
                "field": "salary"
              }
```

```
            }
          }
        }
      }
    }
  }
}
```

管道聚合（Pipeline Aggregation）

支持对聚合分析的结果，再次进行聚合分析。

Pipeline 的分析结果会输出到原结果中，根据位置的不同，分为两类：

- Sibling - 结果和现有分析结果同级
  - Max，min，Avg & Sum Bucket
  - Stats，Extended Status Bucket
  - Percentiles Bucket
- Parent -结果内嵌到现有的聚合分析结果之中
  - Derivative(求导)
  - Cumultive Sum(累计求和)
  - Moving Function(移动平均值)

min_bucket示例

在员工数最多的工种里，找出平均工资最低的工种

```
# 平均工资最低的工种
POST employees/_search
{
  "size": 0,
  "aggs": {
    "jobs": {
      "terms": {
        "field": "job.keyword",
        "size": 10
      },
      "aggs": {
        "avg_salary": {
          "avg": {
            "field": "salary"
          }
        }
      }
    },
    "min_salary_by_job":{
      "min_bucket": {
        "buckets_path": "jobs>avg_salary"
      }
    }
  }
}
```

- min_salary_by_job结果和jobs的聚合同级
- min_bucket求之前结果的最小值
- 通过bucket_path关键字指定路径

Stats示例

```
# 平均工资的统计分析
POST employees/_search
{
  "size": 0,
  "aggs": {
    "jobs": {
      "terms": {
        "field": "job.keyword",
        "size": 10
      },
      "aggs": {
        "avg_salary": {
          "avg": {
            "field": "salary"
          }
        }
      }
    },
    "stats_salary_by_job":{
      "stats_bucket": {
        "buckets_path": "jobs>avg_salary"
      }
    }
  }
}
```

percentiles示例

```
# 平均工资的百分位数
POST employees/_search
{
  "size": 0,
  "aggs": {
    "jobs": {
      "terms": {
        "field": "job.keyword",
        "size": 10
      },
      "aggs": {
        "avg_salary": {
          "avg": {
            "field": "salary"
          }
        }
      }
    },
    "percentiles_salary_by_job":{
      "percentiles_bucket": {
        "buckets_path": "jobs>avg_salary"
      }
    }
  }
}

```

Cumulative_sum示例

```
#Cumulative_sum   累计求和
POST employees/_search
{
  "size": 0,
  "aggs": {
    "age": {
      "histogram": {
        "field": "age",
        "min_doc_count": 0,
        "interval": 1
      },
      "aggs": {
        "avg_salary": {
          "avg": {
            "field": "salary"
          }
        },
        "cumulative_salary":{
          "cumulative_sum": {
            "buckets_path": "avg_salary"
          }
        }
      }
    }
  }
}

```

聚合的作用范围

ES聚合分析的默认作用范围是query的查询结果集，同时ES还支持以下方式改变聚合的作用范围：

- Filter
- Post Filter
- Global

```
#Query
POST employees/_search
{
  "size": 0,
  "query": {
    "range": {
      "age": {
        "gte": 20
      }
    }
  },
  "aggs": {
    "jobs": {
      "terms": {
        "field":"job.keyword"

      }
    }
  }
}

#Filter
POST employees/_search
{
  "size": 0,
  "aggs": {
    "older_person": {
      "filter":{
        "range":{
          "age":{
            "from":35
          }
        }
      },
      "aggs":{
         "jobs":{
           "terms": {
        "field":"job.keyword"
      }
```

```
      }
    }},
    "all_jobs": {
      "terms": {
        "field":"job.keyword"

      }
    }
  }
}


#Post field. 一条语句，找出所有的job类型。还能找到聚合后符合条件的结果
POST employees/_search
{
  "aggs": {
    "jobs": {
      "terms": {
        "field": "job.keyword"
      }
    }
  },
  "post_filter": {
    "match": {
      "job.keyword": "Dev Manager"
    }
  }
}


#global
# 使用global聚合来计算所有匹配查询的文档（即所有年龄大于或等于40岁的员工）的平均薪资。
#global聚合的特点是它会考虑查询范围内的所有文档，而不仅仅是某个特定分组或桶中的文档。
POST employees/_search
{
  "size": 0,
  "query": {
    "range": {
      "age": {
```

```
        "gte": 40
      }
    }
  },
  "aggs": {
    "jobs": {
      "terms": {
        "field":"job.keyword"
      }
    },

    "all":{
      "global":{},
      "aggs":{
        "salary_avg":{
          "avg":{
            "field":"salary"
          }
        }
      }
    }
  }
}
```

排序

指定order，按照count和key进行排序：

- 默认情况，按照count降序排序
- 指定size，就能返回相应的桶

```
#排序 order
#count and key
POST employees/_search
{
  "size": 0,
  "query": {
    "range": {
      "age": {
        "gte": 20
      }
    }
  },
  "aggs": {
    "jobs": {
      "terms": {
        "field":"job.keyword",
        "order":[
          {"_count":"asc"},
          {"_key":"desc"}
          ]

      }
    }
  }
}


#排序 order
#count and key
POST employees/_search
{
  "size": 0,
  "aggs": {
    "jobs": {
      "terms": {
        "field":"job.keyword",
        "order":[  {
            "avg_salary":"desc"
          }]
```

```
      },
    "aggs": {
      "avg_salary": {
        "avg": {
          "field":"salary"
        }
      }
    }
    }
  }
}


#排序 order
#count and key
POST employees/_search
{
  "size": 0,
  "aggs": {
    "jobs": {
      "terms": {
        "field":"job.keyword",
        "order":[  {
            "stats_salary.min":"desc"
          }]
      },
    "aggs": {
      "stats_salary": {
        "stats": {
          "field":"salary"
        }
      }
    }
    }
  }
}
```

3. ES聚合分析不精准原因分析

ElasticSearch在对海量数据进行聚合分析的时候会损失搜索的精准度来满足实时性的需求。

Terms聚合分析的执行流程：

不精准的原因： 数据分散到多个分片，聚合是每个分片的取 Top X，导致结果不精准。ES 可以不每个分片Top X，而是全量聚合，但势必这会有很大的性能问题。

思考：如何提高聚合精确度?

方案1：设置主分片为1

注意7.x版本已经默认为1。

适用场景：数据量小的小集群规模业务场景。

方案2：调大 shard_size 值

设置 shard_size 为比较大的值，官方推荐：size*1.5+10。shard_size 值越大，结果越趋近于精准聚合结果值。此外，还可以通过show_term_doc_count_error参数显示最差情况下的错误值，用于辅助确定 shard_size 大小。

- size：是聚合结果的返回值，客户期望返回聚合排名前三，size值就是 3。
- shard_size: 每个分片上聚合的数据条数。shard_size 原则上要大于等于 size

适用场景：数据量大、分片数多的集群业务场景。

测试： 使用kibana的测试数据

```
DELETE my_flights
PUT my_flights
{
  "settings": {
    "number_of_shards": 20
  },
  "mappings" : {
      "properties" : {
        "AvgTicketPrice" : {
          "type" : "float"
        },
        "Cancelled" : {
          "type" : "boolean"
        },
        "Carrier" : {
          "type" : "keyword"
        },
        "Dest" : {
          "type" : "keyword"
        },
        "DestAirportID" : {
          "type" : "keyword"
        },
        "DestCityName" : {
          "type" : "keyword"
        },
        "DestCountry" : {
          "type" : "keyword"
        },
        "DestLocation" : {
          "type" : "geo_point"
        },
        "DestRegion" : {
          "type" : "keyword"
        },
        "DestWeather" : {
          "type" : "keyword"
        },
        "DistanceKilometers" : {
```

```
        "type" : "float"
      },
      "DistanceMiles" : {
        "type" : "float"
      },
      "FlightDelay" : {
        "type" : "boolean"
      },
      "FlightDelayMin" : {
        "type" : "integer"
      },
      "FlightDelayType" : {
        "type" : "keyword"
      },
      "FlightNum" : {
        "type" : "keyword"
      },
      "FlightTimeHour" : {
        "type" : "keyword"
      },
      "FlightTimeMin" : {
        "type" : "float"
      },
      "Origin" : {
        "type" : "keyword"
      },
      "OriginAirportID" : {
        "type" : "keyword"
      },
      "OriginCityName" : {
        "type" : "keyword"
      },
      "OriginCountry" : {
        "type" : "keyword"
      },
      "OriginLocation" : {
        "type" : "geo_point"
      },
      "OriginRegion" : {
        "type" : "keyword"
```

```
        },
        "OriginWeather" : {
          "type" : "keyword"
        },
        "dayOfWeek" : {
          "type" : "integer"
        },
        "timestamp" : {
          "type" : "date"
        }
      }
    }
}

POST _reindex
{
  "source": {
    "index": "kibana_sample_data_flights"
  },
  "dest": {
    "index": "my_flights"
  }
}

GET my_flights/_count
GET kibana_sample_data_flights/_search
{
  "size": 0,
  "aggs": {
    "weather": {
      "terms": {
        "field":"OriginWeather",
        "size":5,
        "show_term_doc_count_error":true
      }
    }
  }
}

GET my_flights/_search
```

```
{
  "size": 0,
  "aggs": {
    "weather": {
      "terms": {
        "field":"OriginWeather",
        "size":5,
        "shard_size":10,
        "show_term_doc_count_error":true
      }
    }
  }
}
```

在Terms Aggregation的返回中有两个特殊的数值：

- doc_count_error_upper_bound : 被遗漏的term 分桶，包含的文档，有可能的最大值
- sum_other_doc_count: 除了返回结果 bucket的terms以外，其他 terms 的文档总数（总数-返回的总数)

方案3：将size设置为全量值，来解决精度问题

将size设置为2的32次方减去1也就是分片支持的最大值，来解决精度问题。

原因：1.x版本，size等于 0 代表全部，高版本取消 0 值，所以设置了最大值（大于业务的全量值）。

全量带来的弊端就是：如果分片数据量极大，这样做会耗费巨大的CPU 资源来排序，而且可能会阻塞网络。

适用场景：对聚合精准度要求极高的业务场景，由于性能问题，不推荐使用。

方案4：使用Clickhouse/ Spark 进行精准聚合

适用场景：数据量非常大、聚合精度要求高、响应速度快的业务场景。

4. Elasticsearch 聚合性能优化

插入数据时对索引进行预排序

- Index sorting （索引排序）可用于在插入时对索引进行预排序，而不是在查询时再对索引进行排序，这将提高范围查询（range query）和排序操作的性能。
- 在 Elasticsearch 中创建新索引时，可以配置如何对每个分片内的段进行排序。
- 这是 Elasticsearch 6.X 之后版本才有的特性。

```
PUT /my_index
{
  "settings": {
    "index":{
      "sort.field": "create_time",
      "sort.order": "desc"
    }
  },
  "mappings": {
    "properties": {
      "create_time":{
        "type": "date"
      }
    }
  }
}
```

注意：预排序将增加 Elasticsearch 写入的成本。在某些用户特定场景下，开启索引预排序会导致大约 40%-50% 的写性能下降。也就是说，如果用户场景更关注写性能的业务，开启索引预排序不是一个很好的选择。

使用节点查询缓存

节点查询缓存（Node query cache）可用于有效缓存过滤器（filter）操作的结果。如果多次执行同一 filter 操作，这将很有效，但是即便更改过滤器中的某一个值，也将意味着需要计算新的过滤器结果。你可以执行一个带有过滤查询的搜索请求，Elasticsearch将自动尝试使用节点查询缓存来优化性能。例如，如果你想缓存一个基于特定字段值的过滤查询，你可以发送如下的HTTP请求：

```
GET /your_index/_search
{
  "query": {
    "bool": {
      "filter": {
        "term": {
          "your_field": "your_value"
        }
      }
    }
  }
}
```

使用分片请求缓存

聚合语句中，设置：size：0，就会使用分片请求缓存缓存结果。size = 0 的含义是：只返回聚合结果，不返回查询结果。

```
GET /es_db/_search
{
  "size": 0,
  "aggs": {
    "remark_agg": {
      "terms": {
        "field": "remark.keyword"
      }
    }
  }
}
```

拆分聚合，使聚合并行化

Elasticsearch 查询条件中同时有多个条件聚合，默认情况下聚合不是并行运行的。当为每个聚合提供自己的查询并执行 msearch 时，性能会有显著提升。因此，在 CPU 资源不是瓶颈的前提下，如果想缩短响应时间，可以将多个聚合拆分为多个查询，借助：msearch 实现并行聚合。

```
#常规的多条件聚合实现
GET /employees/_search
{
  "size": 0,
  "aggs": {
    "job_agg": {
      "terms": {
        "field": "job.keyword"
      }
    },
    "max_salary":{
      "max": {
        "field": "salary"
      }
    }
  }
}
# msearch 拆分多个语句的聚合实现
GET _msearch
{"index":"employees"}
{"size":0,"aggs":{"job_agg":{"terms":{"field": "job.keyword"}}}}
{"index":"employees"}
{"size":0,"aggs":{"max_salary":{"max":{"field": "salary"}}}}
```