聚合

结构：

```
{
    "aggregations" : {                          --可以写成aggs
        "<aggregation_name>" : {
            "<aggregation_type>" : {
                <aggregation_body>
            }
            [,"meta" : {  [<meta_data_body>] } ]?
            [,"aggregations" : { [<sub_aggregation>]+ } ]?
        }
        [,"<aggregation_name_2>" : { ... } ]*
    }
}
```

平均值 avg

1. 例子：

```
//统计grade字段的平均值
{
    "aggs":{
        "agg_name":{
            "avg":{
                "field":"grade"
            }
        }
    }
}
```

2. avg 中可以添加的属性：
    - `script` :可以添加脚本
    - `missing` :将缺少聚合字段的记录设置一个默认值，如：{"missing":10},设置缺少该字段的记录，统计该字段时值为10；

取最大值Max

1. 例子：

```
{
    "aggs" : {
        "max_price" : { "max" : { "field" : "price" } }
    }
}
```

2. max 中可以添加的属性同上；

去最小值min

1. 例子：

```
{
    "aggs" : {
        "min_price" : { "min" : { "field" : "price" } }
    }
}
```

2. min 可添加属性同上；

总和sum

1. 例子：

```
{
    "aggs" : {
        "intraday_return" : { "sum" : { "field" : "change" } }
    }
}
```

2. sum 可添加属性同上

value_count统计某个字段有值的文档数

```
{
    "aggs":{
        "age_count":{
            "value_count":{
                "field":"age"
            }

        }
    }
}
```

top_hits 抓取某顺序下排最前面的几条数据

1. 例子：

```
//根据创建时间倒序找出最前面的两条数据
{
     "aggs":{
         "agg-day":{
             "top_hits":{
                 "from":0,
                 "size":2,
                 "sort":[{
                     "CREATE_TIME": "desc"
                 }
                 ],
                 "_source":{
                     "include":["NAME","SEX","CREATE_TIME"]
                 }
             }
         }
     }
 }
```

2. top_hits 中主要用的参数：
   - from :排序后记录下标；
   - size :抓取多少条数据；
   - sort :排序的方式；
   - _source :对查询返回字段进行设置；
   - 在search body中高亮等参数均可设置；

terms Query,根据特殊字段进行统计，类似mysql里面的select count(1).....group by field

1. 例子：

```
{
    "aggs" : {
        "genders" : {
            "terms" : { "field" : "gender" }
        }
    }
}
```

2. terms 可以添加的参数有：
   - size： 限制只返回几个桶，设置为0的时候，则取值 Integer.MAX_VALUE；
   - shard_size： 限制每个分片返回几个桶给主节点,设置为0的时候，则取值 Integer.MAX_VALUE；

- `script`： 脚本

3. 讲解 `size` 与 `shard_size` 两个参数的作用：实际terms统计的原理是，es每个分片分别对自己存储的文档先进行一次group by（分桶）操作，根据 `shard_size` 的值，返回多少个桶结果给主节点，然后主节点对每个分片的结果进行合并，并根据 `size` 的值返回多少个桶；
   - 如shard_size=5，size=5，有N、M、I三个分片，统计一个公司表中每个部门的人数，即group by deptment；首先N、M、I三个分片各自选出自己中桶数（即人数）最高的5个部门，然后发给主节点，主节点再将这三个分片的部门的统计结果进行**合并与排序**，返回5个最多人数的部门；
   - 但是由于ES分片的原理，导致这个过程中会有问题；假设有ABCDEFG共7个部门，A部门一共有50个人，在N分片中存了40个人的数据、M分片中存了10个人的数据（即**ES分片存储时，并不是一个分片存一个部门所有人的信息**），这时候可能出现，N分片中A部门进入了前5个人数最多的排名中，但是在M分片中没有，这时候汇总到主节点时，就会导致最终比较时N分片仅有40人；所以使用size和shard_size时需要注意；
   - 由于上述问题，如果想统计完整个数，则可以不加size和shard_size两个属性，但是注意不要分太多桶，因为要进行汇总和排序操作，数据太多可能CPU承受不了

基数聚合

`cardinality` :查询一个字段可以分为多少个桶（有多少种取值）；

```
//查询性别字段基数
{
    "aggs" : {
        "author_count" : {
            "cardinality" : {
                "field" : "gender"
            }
        }
    }
}
//如果记录是将性别分为：1为男，2为女，则该聚合就返回值value：2，因为gender就只有两个值；
```

嵌套聚合

先根据最外层定义的聚合方式对文档集进行聚合，然后再执行子聚合的聚合方式；与sql的嵌套反过来（sql是先执行子查询，再根据子查询结果获取外层查询结果）；

如：对一组商品，按照tag字段进行分组，并计算每组的平均价格,并按平均价降序排序

```
{
    "aggs":{
        "tag_agg":{
            "terms":{
                //根据tag字段进行分组并按平均价降序排序
                "field":"tag",
                //排序可以指向一个聚合结果
                "order":{"price_aggs":"desc"}
            },
            "aggs":{
                //平均价聚合
                "price_aggs":{
                    "avg":{
                        "field":"price"
                    }
                }
            }
        }
    }
}
```

由上面可以看到，**求平均值的聚合在里层**，说明ES先执行外层聚合，即先根据tag分组结果：

```
{

  "took": 3,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 5,
    "max_score": 0,
    "hits": [........]//聚合用的文档集合
  },
  "aggregations": {
    "group_by_tags": {
      "doc_count_error_upper_bound":0,
      "sum_other_doc_count": 0,
    //桶聚合的返回
      "buckets": [
        {
            //tag字段取值之一
          "key":"hanghaishi",
          //该桶中的文档数
          "doc_count": 1,
          "avg_price": {
            //平均数
            "value": 2200000000
          }
        },
        {
          "key": "qingxin",
          "doc_count": 1,
          "avg_price": {
            "value": 40
          }
        },
        {
          "key": "meibai",
          "doc_count": 1,
          "avg_price": {
            "value": 30
          }
        },
        {
          "key": "fangzhu",
          "doc_count": 2,
          "avg_price": {
            "value": 27.5
          }
        }
```

```
        ]
      }
    }
  }
```

查询

aggs 关键字内没有 query 作为关键字， query 和 aggs （最外层）是同级的，并且是对查询的结果进行聚合；

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

https://www.elastic.co/guide/en/elasticsearch/reference/2.1/search-aggregations-metrics-stats-aggregation.html#search-aggregations-metrics-stats-aggregation

举例：
https://blog.csdn.net/lianxiaobao/article/details/79128122