几个关键的名次:

（1）索引（index）

一个索引就是一个拥有几分相似特征的文档的集合。比如说，你可以有一个客户数据的索引，另一个产品目录的索引，还有一个订单数据的索引。一个索引由一个名字来标识（必须全部是小写字母的），并且当我们要对对应于这个索引中的文档进行索引、搜索、更新和删除的时候，都要使用到这个名字。索引类似于关系型数据库中Database的概念。在一个集群中，如果你想，可以定义任意多的索引。

（2）类型（type）

在一个索引中，你可以定义一种或多种类型。一个类型是你的索引的一个逻辑上的分类/分区，其语义完全由你来定。通常，会为具有一组共同字段的文档定义一个类型。比如说，我们假设你运营一个博客平台并且将你所有的数据存储到一个索引中。在这个索引中，你可以为用户数据定义一个类型，为博客数据定义另一个类型，当然，也可以为评论数据定义另一个类型。类型类似于关系型数据库中Table的概念。

（3）文档（document）

一个文档是一个可被索引的基础信息单元。比如，你可以拥有某一个客户的文档，某一个产品的一个文档，当然，也可以拥有某个订单的一个文档。文档以JSON（JavaScript Object Notation）格式来表示，而JSON是一个到处存在的互联网数据交互格式。

在一个index/type里面，只要你想，你可以存储任意多的文档。注意，尽管一个文档，物理上存在于一个索引之中，文档必须被索引/赋予一个索引的type。文档类似于关系型数据库中Record的概念。实际上一个文档除了用户定义的数据外，还包括\_index、\_type和\_id字段。

**索引（Index）相关API**

1. 创建一个新的索引。

curl -XPUT "192.168.1.129:9200/index\_test"-----------------index\_test为索引名称

使用默认配置创建一个索引。

curl -XPUT "192.168.1.129:9200/index\_test" -d ' # 注意这里的'号  
{  
 "settings": {  
 "index": {  
 "number\_of\_replicas": "1", # 设置复制数  
 "number\_of\_shards": "5" # 设置主分片数  
 }  
 },  
 "mappings": { # 创建mapping  
 "test\_type": { # 在index中创建一个新的type(相当于table)  
 "properties": {  
 "name": { # 创建一个字段（string类型数据，使用普通索引）  
 "type": "string",  
 "index": "not\_analyzed"  
 },  
 "age": {  
 "type": "integer"  
 }  
 }  
 }  
 }  
}'

创建索引的同时，创建settings和mapping。

（2）删除一个索引

curl -XDELETE "192.168.1.129:9200/index\_test"

**映射（Mapping）相关API**

（1）创建索引的mapping

curl -XPUT 'localhost:9200/index\_test/\_mapping/test\_type' -d '   
{  
 "test\_type": { # 注意，这里的test\_type与url上的test\_type名保存一致  
 "properties": {  
 "name": {  
 "type": "string",  
 "index": "not\_analyzed"  
 },  
 "age": {  
 "type": "integer"  
 }  
 }  
 }  
 }'

如果不想单独创建mapping，可以使用上一节的方法（创建索引时创建mappings）。

另外一种创建mapping的方式(加载.json文件的方式)

步骤1 创建一个扩展名为test\_type.json的文件名，其中type\_test就是mapping所对应的type名。

步骤2 在test\_type.json中输入mapping信息。假设你的mapping如下：

{  
 "test\_type": { # 注意，这里的test\_type与json文件名必须一致  
 "properties": {  
 "name": {  
 "type": "string",  
 "index": "not\_analyzed"  
 },  
 "age": {  
 "type": "integer"  
 }  
 }  
 }  
 }

步骤3 在$ES\_HOME/config/路径下创建mappings/index\_test子目录，这里的index\_test目录名必须与我们要建立的索引名一致。将test\_type.json文件拷贝到index\_tes目录下。

步骤4 创建index\_test索引。操作如下：

curl -XPUT "192.168.1.129:9200/index\_test" # 注意，这里的索引名必须与mappings下新建的index\_test目录名一致

这样我们就创建了一个新的索引，并且使用了test\_type.json所定义的mapping作为索引的mapping。就是这么简单方便！

（2）删除mapping。

curl -XDELETE 'localhost:9200/index\_test/\_mapping/test\_type'

（3）查看索引的mapping。

curl -XGET 'localhost:9200/index\_test/\_mapping/test\_type'

**文档（document）相关API**

（1）新增一个文档。

curl -XPUT '192.168.1.129:9200/index\_test/test\_type/1?pretty' -d ' # 这里的pretty参数的作用是使得返回的json显示地更加好看。1是文档的id值（唯一键）。  
{  
 "name": "zhangsan",  
 "age" : "12"  
}'

（2）更新一个文档

curl -XPOST '192.168.1.129:9200/index\_test/test\_type/1?pretty' -d ' # 这里的1必须是索引中已经存在id，否则就会变成新增文档操作  
{  
 "name": "lisi",  
 "age" : "12"  
}'

（3）删除一个文档

curl -XDELETE '192.168.1.129:9200/index\_test/test\_type/1?pretty' # 这里的1必须是索引中已经存在id

（4）查询单个文档

curl -XGET '192.168.1.129:9200/index\_test/test\_type/1?pretty'

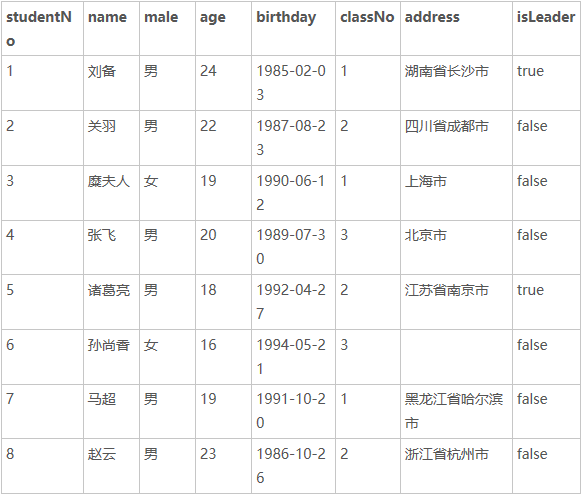
**Mapping映射**



上面的表结构所对应的mapping如下

{  
 "student": {  
 "properties": {  
 "studentNo": {  
 "type": "string",  
 "index": "not\_analyzed"  
 },  
 "name": {  
 "type": "string",  
 "index": "not\_analyzed"  
 },  
 "male": {  
 "type": "string",  
 "index": "not\_analyzed"  
 },  
 "age": {  
 "type": "integer"  
 },  
 "birthday": {  
 "type": "date",  
 "format": "yyyy-MM-dd"  
 },  
 "address": {  
 "type": "string",  
 "index": "not\_analyzed"  
 },  
 "classNo": {  
 "type": "string",  
 "index": "not\_analyzed "  
 },  
 "isLeader": {  
 "type": "boolean"  
 }  
 }  
 }  
}

索引中保存的数据如下，下面介绍的所有API都将基于这个数据表。



**查询API**

**Query和Filter**

ES为用户提供两类查询API，一类是在查询阶段就进行条件过滤的query查询，另一类是在query查询出来的数据基础上再进行过滤的filter查询。这两类查询的区别是：

* query方法会计算查询条件与待查询数据之间的相关性，计算结果写入一个score字段，类似于搜索引擎。filter仅仅做字符串匹配，不会计算相关性，类似于一般的数据查询，所以filter得查询速度比query快。
* filter查询出来的数据会自动被缓存，而query不能。

query和filter可以单独使用，也可以相互嵌套使用，非常灵活。

**Query查询**

下面的情况下适合使用query查询：

* 需要进行全文搜索。
* 查询结果依赖于相关性，即需要计算查询串和数据的相关性。

（1）Match All Query

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "query": {  
 "match\_all": {}  
 }  
}'

查询结果如下。其他所有的查询都是返回这种格式的数据

{  
 "took": 156, // 查询耗时（毫秒）  
 "timed\_out": false, // 是否超时  
 "\_shards": {  
 "total": 5, // 总共查询的分片数  
 "successful": 5, // 查询成功的分片数  
 "failed": 0 // 查询失败的分片数  
 },  
 "hits": {  
 "total": 8, // 本次查询的记录数  
 "max\_score": 1, // 查询所有数据中的最大score  
 "hits": [ // 数据列表  
 {  
 "\_index": "student", // 数据所属的索引名  
 "\_type": "student", // 数据所属的type  
 "\_id": "4", // 数据的id值  
 "\_score": 1, // 该记录的score  
 "\_source": { // ES将原始数据保存到\_source字段中  
 "studentNo": "4",  
 "name": "张飞",  
 "male": "男",  
 "age": "20",  
 "birthday": "1989-07-30",  
 "classNo": "3",  
 "isLeader": "F"  
 }  
 },  
 {  
 …… // 其他的数据格式相同，就不列出来了  
 }  
 ]  
 }  
}

查询时，你会发现无论数据量有多大，每次最多只能查到10条数据。这是因为ES服务端默认对查询结果做了分页处理，每页默认的大小为10。如果想自己指定查询的数据，可使用from和size字段，并且按指定的字段排序。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'  
{  
 "query": {  
 "match\_all": {}  
 },  
 "from": 2, // 从2条记录开始取  
 "size": 4, // 取4条数据  
 "sort": {  
 "studentNo": { // 按studentNo字段升序  
 "order": "asc"// 降序为desc  
 }  
 }   
}  
'

注意：不要把from设得过大（超过10000），否则会导致ES服务端因频繁GC而无法正常提供服务。其实实际项目中也没有谁会翻那么多页，但是为了ES的可用性，务必要对分页查询的页码做一定的限制。

（2）term query

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d

'{  
 "query": {  
 "term": {  
 "name": "诸葛亮"  
 }  
 }  
}'

（3）Bool Query

Bool（布尔）查询是一种复合型查询，它可以结合多个其他的查询条件。主要有3类逻辑查询：

* must：查询结果必须符合该查询条件（列表）。
* should：类似于in的查询条件。如果bool查询中不包含must查询，那么should默认表示必须符合查询列表中的一个或多个查询条件。
* must\_not：查询结果必须不符合查询条件（列表）。

查找2班的班干部，查询结果为学号为5的记录。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "query": {  
 "bool": {  
 "must": [  
 {  
 "term": {  
 "classNo": "2"  
 }  
 },  
 {  
 "term": {  
 "isLeader": "true"  
 }  
 }  
 ]  
 }  
 }  
}'

（4）Ids Query

id字段查询。查询数据id值为1和2的同学，由于id的值与studentNo相同，故查询结果为学号为1和2的学生。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "query": {  
 "ids": {  
 "type": "student",  
 "values": [  
 "1",  
 "2"  
 ]  
 }  
 }  
}'

（5）Prefix Query

前缀查询。查找姓【赵】的同学，查询结果是学号为8的赵云。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "query": {  
 "prefix": {  
 "name": "赵"  
 }  
 }  
}'

（6）Range Query

范围查询，针对date和number类型的数据。查找年龄到18~20岁的同学，查询结果是学号为3、4、5、7的记录。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "query": {  
 "range": {  
 "age": {  
 "gte": "18", // 表示>=  
 "lte": "20" // 表示<=  
 }  
 }  
 }  
}'

实际上，对于date类型的数据，ES中以其时间戳（长整形）的形式存放的。

（7）Terms Query

多词语查询，查找符合词语列表的数据。如果要查询的字段索引为not\_analyzed类型，则terms查询非常类似于关系型数据库中的in查询。下面查找学号为1，3的学生。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "query": {  
 "terms": {  
 "studentNo": [  
 "1",  
 "3"  
 ]  
 }  
 }  
}'

（8）Wildcard Query

通配符查询，是简化的正则表达式查询，包括下面两类通配符：

* \* 代表任意（包括0个）多个字符
* ? 代表任意一个字符

查找名字的最后一个字是“亮”的同学，查询结果是学号为5的诸葛亮。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "query": {  
 "wildcard": {  
 "name": "\*亮"  
 }  
 }  
}'

（9）Regexp Query同学

正则表达式查询，这是最灵活的字符串类型字段查询方式。查找家住长沙市的学生，查询结果为学号为1的学生。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "query": {  
 "regexp": {  
 "address": ".\*长沙市.\*" // 这里的.号表示任意一个字符  
 }  
 }  
}'

**Filter查询**

下面的情况下适合使用filter查询：

* yes/no的二元查询
* 针对精确值进行查询

filter和query的查询方式有不少是重叠的，所以本节仅仅介绍API的调用，一些通用的注意的事项就不再重复了

（1）Term Filter

词语查询，如果是对未分词的字段进行查询，则表示精确查询。查找名为“诸葛亮”的学生，查询结果为学号为5的记录。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "filter": {   
 "term": {  
 "name": "诸葛亮",  
 "\_cache" : true // 与query主要是这里的区别，可以设置数据缓存  
 }  
 }  
}'

filter查询方式都可以通过设置\_cache为true来缓存数据。如果下一次恰好以相同的查询条件进行查询并且该缓存没有过期，就可以直接从缓存中读取数据，这样就大大加快的查询速度

（2）Bool Filter

查找2班的班干部，查询结果为学号为5的记录。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "filter": {  
 "bool": {  
 "must": [  
 {  
 "term": {  
 "classNo": "2"  
 }  
 },  
 {  
 "term": {  
 "isLeader": "true"  
 }  
 }  
 ]  
 }  
 }  
}'

（3）And Filter

And逻辑连接查询，连接1个或1个以上查询条件。它与bool查询中的must查询非常相似。实际上，and查询可以转化为对应的bool查询。查找2班的班干部，查询结果为学号为5的学生。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "filter": {  
 "and": [  
 {  
 "term": {  
 "classNo": "2"  
 }  
 },  
 {  
 "term": {  
 "isLeader": "true"  
 }  
 }  
 ]  
 }  
}'

（4）Or Filter

Or连接查询，表示逻辑或。。查找2班或者是班干部的学生名单，查询结果为学号为1、2、5、8的学生。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "filter": {  
 "or": [  
 {  
 "term": {  
 "classNo": "2"  
 }  
 },  
 {  
 "term": {  
 "isLeader": "true"  
 }  
 }  
 ]  
 }  
}'

（5）Exists Filter

存在查询，查询指定字段至少包含一个非null值的数据。如果字段索引为not\_analyzed类型，则查询sql中的is not null查询方式。查询地址存在学生，查询结果为除了6之外的所有学生。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "filter": {  
 "exists": {  
 "field": "address"  
 }  
 }  
}'

（6）Missing Filter

缺失值查询，与Exists查询正好相反。查询地址不存在的学生，查询结果为学号为6的学生。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "filter": {  
 "missing": {  
 "field": "address"  
 }  
 }  
}'

（7）Prefix Filter

前缀查询。查找姓【赵】的同学，查询结果是学号为8的赵云。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "filter": {  
 "prefix": {  
 "name": "赵"  
 }  
 }  
}'

（8）Range Filter

范围查询，针对date和number类型的数据。查找年龄到18~20岁的同学，查询结果是学号为3、4、5、7的记录。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "filter": {  
 "range": {  
 "age": {  
 "gte": "18",  
 "lte": "20"  
 }  
 }  
 }  
}'

（9）Terms Filter

多词语查询，查找符合词语列表的数据。如果要查询的字段索引为not\_analyzed类型，则terms查询非常类似于关系型数据库中的in查询。下面查找学号为1，3的学生。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "filter": {  
 "terms": {  
 "studentNo": [  
 "1",  
 "3"  
 ]  
 }  
 }  
}'

（10）Regexp Filter

正则表达式查询，是最灵活的字符串类型字段查询方式。查找家住长沙市的学生，查询结果为学号为1的学生。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "filter": {  
 "regexp": {  
 "address": ".\*长沙市.\*"  
 }  
 }  
}'

**聚合API**

本章的主要内容有：

1. metric API的使用
2. bucketing API的使用
3. 两类API的嵌套使用

ES中的聚合上可以分为下面两类：

1. **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
2. **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既可以作用在整个数据集上，也可以作为bucketing的子聚合作用在每一个“桶”中的数据集上。当然，我们可以把整个数据集合看做一个大“桶”，所有的数据都分配到这个大“桶”中。

ES中的聚合API的调用格式如下：

"aggregations" : { // 表示聚合操作，可以使用aggs替代  
 "<aggregation\_name>" : { // 聚合名，可以是任意的字符串。用做响应的key，便于快速取得正确的响应数据。  
 "<aggregation\_type>" : { // 聚合类别，就是各种类型的聚合，如min等  
 <aggregation\_body> // 聚合体，不同的聚合有不同的body  
 }  
 [,"aggregations" : { [<sub\_aggregation>]+ } ]? // 嵌套的子聚合，可以有0或多个  
 }  
 [,"<aggregation\_name\_2>" : { ... } ]\* // 另外的聚合，可以有0或多个  
}

**1.1 度量类型（metric）聚合**

最小值查询，作用于number类型字段上。查询2班最小的年龄值。

curl -XPOST "192.168.1.129:9200/student/student/\_search" -d   
'{  
 "query": { // 可以先使用query查询得到需要的数据集  
 "term": {  
 "classNo": "2"  
 }  
 },  
 "aggs": {  
 "min\_age": {  
 "min": {  
 "field": "age"  
 }  
 }  
 }  
}'

查询结果为：

{  
 "took": 19, // 前面部分数据与普通的查询数据相同  
 "timed\_out": false,  
 "\_shards": {  
 "total": 5,  
 "successful": 5,  
 "failed": 0  
 },  
 "hits": {  
 "total": 3,  
 "max\_score": 1.4054651,  
 "hits": [  
 {  
 "\_index": "student",  
 "\_type": "student",  
 "\_id": "2",  
 "\_score": 1.4054651,  
 "\_source": {  
 "studentNo": "2",  
 "name": "关羽",  
 "male": "男",  
 "age": "22",  
 "birthday": "1987-08-23",  
 "classNo": "2",  
 "isLeader": "false"  
 }  
 },  
 {  
 "\_index": "student",  
 "\_type": "student",  
 "\_id": "8",  
 "\_score": 1,  
 "\_source": {  
 "studentNo": "8",  
 "name": "赵云",  
 "male": "男",  
 "age": "23",  
 "birthday": "1986-10-26",  
 "classNo": "2",  
 "isLeader": "false"  
 }  
 },  
 {  
 "\_index": "student",  
 "\_type": "student",  
 "\_id": "5",  
 "\_score": 0.30685282,  
 "\_source": {  
 "studentNo": "5",  
 "name": "诸葛亮",  
 "male": "男",  
 "age": "18",  
 "birthday": "1992-04-27",  
 "classNo": "2",  
 "isLeader": "true"  
 }  
 }  
 ]  
 },  
 "aggregations": { // 聚合结果  
 "min\_age": { // 前面输入的聚合名  
 "value": 18, // 聚合后的数据  
 "value\_as\_string": "18.0"  
 }  
 }  
}

上面的聚合查询有两个要注意的点：

1. 可以通过query先过滤数据
2. 返回的结果会包含聚合操作所作用的数据全集

有时候我们对作用的数据全集并不太敢兴趣，我们仅仅需要最终的聚合结果。可以通过查询类型（search\_type）参数来实现这个需求。下面查询出来的数据量会大大减少，ES内部也会在查询时减少一些耗时的步骤，所以查询效率会提高。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d // 注意这里的search\_type=count  
'{  
 "query": { // 可以先使用query查询得到需要的数据集  
 "term": {  
 "classNo": "2"  
 }  
 },  
 "aggs": {  
 "min\_age": {  
 "min": {  
 "field": "age"  
 }  
 }  
 }  
}'

本次的查询结果为：

{  
...

"aggregations": { // 聚合结果  
 "min\_age": { // 前面输入的聚合名  
 "value": 18, // 聚合后的数据  
 "value\_as\_string": "18.0"  
 }  
 }  
}

（2）Max Aggregation

最大值查询。下面查询2班最大的年龄值，查询结果为23。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d   
'{  
 "query": {  
 "term": {  
 "classNo": "2"  
 }  
 },  
 "aggs": {  
 "max\_age": {  
 "max": {  
 "field": "age"  
 }  
 }  
 }  
}'

（3）Sum Aggregation

数值求和。下面统计查询2班的年龄总和，查询结果为63。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d   
'{  
 "query": {  
 "term": {  
 "classNo": "2"  
 }  
 },  
 "aggs": {  
 "sum\_age": {  
 "sum": {  
 "field": "age"  
 }  
 }  
 }  
}'

（4）Avg Aggregation

计算平均值。下面计算查询2班的年龄平均值，结果为21。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d   
'{  
 "query": {  
 "term": {  
 "classNo": "2"  
 }  
 },  
 "aggs": {  
 "avg\_age": {  
 "avg": {  
 "field": "age"  
 }  
 }  
 }  
}'

（5）Stats Aggregation

统计查询，一次性统计出某个字段上的常用统计值。下面对整个学校的学生进行简单地统计。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d   
'{  
 "aggs": {  
 "stats\_age": {  
 "stats": {  
 "field": "age"  
 }  
 }  
 }  
}'

查询结果为：

{  
 ... // 次要数据省略

"aggregations": {  
 "stats\_age": {  
 "count": 8, // 含有年龄数据的学生计数  
 "min": 16, // 年龄最小值  
 "max": 24, // 年龄最大值  
 "avg": 20.125, // 年龄平均值  
 "sum": 161, // 年龄总和  
 "min\_as\_string": "16.0",  
 "max\_as\_string": "24.0",  
 "avg\_as\_string": "20.125",  
 "sum\_as\_string": "161.0"  
 }  
 }  
}

（6）Top hits Aggregation

取符合条件的前n条数据记录。下面查询全校年龄排在前2位的学生，仅需返回学生姓名和年龄。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d   
{  
 "aggs": {  
 "top\_age": {  
 "top\_hits": {  
 "sort": [ // 排序  
 {  
 "age": { // 按年龄降序  
 "order": "desc"  
 }  
 }  
 ],  
 "\_source": {  
 "include": [ // 指定返回字段  
 "name",  
 "age"  
 ]  
 },  
 "size": 2 // 取前2条数据  
 }  
 }  
 }  
}

返回结果为：

{  
 ...

"aggregations": {  
 "top\_age": {  
 "hits": {  
 "total": 9,  
 "max\_score": null,  
 "hits": [  
 {  
 "\_index": "student",  
 "\_type": "student",  
 "\_id": "1",  
 "\_score": null,  
 "\_source": {  
 "name": "刘备",  
 "age": "24"  
 },  
 "sort": [  
 24  
 ]  
 },  
 {  
 "\_index": "student",  
 "\_type": "student",  
 "\_id": "8",  
 "\_score": null,  
 "\_source": {  
 "name": "赵云",  
 "age": "23"  
 },  
 "sort": [  
 23  
 ]  
 }  
 ]  
 }  
 }  
 }  
}

**1.2 桶类型（bucketing）聚合**

（1）Terms Aggregation

按照指定的1或多个字段将数据划分成若干个小的区间，计算落在每一个区间上记录数量，并按指定顺序进行排序。下面统计每个班的学生数，并按学生数从大到小排序，取学生数靠前的2个班级。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d   
'{  
 "aggs": {  
 "terms\_classNo": {  
 "terms": {  
 "field": "classNo", // 按照班号进行分组  
 "order": { // 按学生数从大到小排序  
 "\_count": "desc"  
 },  
 "size": 2 // 取前两名  
 }  
 }  
 }  
}'

值得注意的，取得的前2名的学生数实际上是一个近似值，ES的实现方式参见[这里](https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-terms-aggregation.html)。如果想要取得精确值，可以不指定size值，使其进行一次全排序，然后在程序中自行去取前2条记录。当然，这样做会使得ES做大量的排序运算工作，效率比较差。

（2）Range Aggregation

自定义区间范围的聚合，我们可以自己手动地划分区间，ES会根据划分出来的区间将数据分配不同的区间上去。下面将全校学生按照年龄划分为5个区间段：16岁以下、16~18、19~21、22~24、24岁以上，要求统计每一个年龄段内的学生数。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d   
'  
{  
 "aggs": {  
 "range\_age": {  
 "range": {  
 "field": "age",  
 "ranges": [  
 {  
 "to": 15  
 },  
 {  
 "from": "16",  
 "to": "18"  
 },  
 {  
 "from": "19",  
 "to": "21"  
 },

{  
 "from": "22",  
 "to": "24"  
 },  
 {  
 "from": "25"  
 }  
 ]  
 }  
 }  
 }  
}  
'

（3）Date Range Aggregation

时间区间聚合专门针对date类型的字段，它与Range Aggregation的主要区别是其可以使用时间运算表达式。主要包括+（加法）运算、-（减法）运算和/（四舍五入）运算，每种运算都可以作用在不同的时间域上面，下面是一些时间运算表达式示例。

* now+10y：表示从现在开始的第10年。
* now+10M：表示从现在开始的第10个月。
* 1990-01-10||+20y：表示从1990-01-01开始后的第20年，即2010-01-01。

下面查询25年前及更早出生的学生数。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d   
'{  
 "aggs": {  
 "range\_age": {  
 "date\_range": {  
 "field": "birthday",  
 "ranges": [  
 {  
 "to": "now-25y"  
 }  
 ]  
 }  
 }  
 }  
}'

（4）Histogram Aggregation

直方图聚合，它将某个number类型字段等分成n份，统计落在每一个区间内的记录数。它与前面介绍的Range聚合非常像，只不过Range可以任意划分区间，而Histogram做等间距划分。既然是等间距划分，那么参数里面必然有距离参数，就是interval参数。下面按学生年龄统计各个年龄段内的学生数量，分隔距离为2岁。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d   
'{  
 "aggs": {  
 "histogram\_age": {  
 "histogram": {  
 "field": "age",  
 "interval": 2, // 距离为2  
 "min\_doc\_count": 1 // 只返回记录数量大于等于1的区间  
 }  
 }  
 }  
}'

（5）Date Histogram Aggregation

时间直方图聚合，专门对时间类型的字段做直方图聚合。这种需求是比较常用见得的，我们在统计时，通常就会按照固定的时间断（1个月或1年等）来做统计。下面统计学校中同一年出生的学生数。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d   
'{  
 "aggs": {  
 "data\_histogram\_birthday": {  
 "date\_histogram": {  
 "field": "birthday",  
 "interval": "year", // 按年统计  
 "format": "yyyy" // 返回结果的key的格式  
 }  
 }  
 }  
}'

返回结果如下，可以看到由于上面的”format”: “yyyy”，所以返回的key\_as\_string只返回年的信息。

{  
 "buckets": [  
 {  
 "key\_as\_string": "1985",  
 "key": 473385600000,  
 "doc\_count": 1  
 },  
 {  
 "key\_as\_string": "1986",  
 "key": 504921600000,  
 "doc\_count": 1  
 },  
 {  
 "key\_as\_string": "1987",  
 "key": 536457600000,  
 "doc\_count": 1  
 },  
 {  
 "key\_as\_string": "1989",  
 "key": 599616000000,  
 "doc\_count": 1  
 },  
 {  
 "key\_as\_string": "1990",  
 "key": 631152000000,  
 "doc\_count": 1  
 },  
 {  
 "key\_as\_string": "1991",  
 "key": 662688000000,  
 "doc\_count": 1  
 },  
 {  
 "key\_as\_string": "1992",  
 "key": 694224000000,  
 "doc\_count": 1  
 },  
 {  
 "key\_as\_string": "1994",  
 "key": 757382400000,  
 "doc\_count": 1  
 }  
 ]  
}

（6）Missing Aggregation

值缺损聚合，它是一类单桶聚合，也就是最终只会产生一个“桶”。下面统计学生信息中地址栏缺损的记录数量。由于只有学号为6的孙尚香的地址缺损，所以统计值为1。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d   
'{  
 "aggs": {  
 "missing\_address": {  
 "missing": {  
 "field": "address"  
 }  
 }  
 }  
}'

**1.3 嵌套使用**

前面已经说过，聚合操作是可以嵌套使用的。通过嵌套，可以使得metric类型的聚合操作作用在每一“桶”上。我们可以使用ES的嵌套聚合操作来完成稍微复杂一点的统计功能。下面统计每一个班里最大的年龄值。

curl -XPOST "192.168.1.129:9200/student/student/\_search?search\_type=count" -d  
'{  
 "aggs": {  
 "missing\_address": {  
 "terms": {  
 "field": "classNo"  
 },  
 "aggs": { // 在这里嵌套新的子聚合  
 "max\_age": {  
 "max": { // 使用max聚合  
 "field": "age"  
 }  
 }  
 }  
 }  
 }  
}'

返回结果如下：

{  
 "buckets": [  
 {  
 "key": "1", // key是班级号  
 "doc\_count": 3, // 每个班级内的人数  
 "max\_age": { // 这里是我们指定的子聚合名  
 "value": 24, // 每班的年龄值  
 "value\_as\_string": "24.0"  
 }  
 },  
 {  
 "key": "2",  
 "doc\_count": 3,  
 "max\_age": {  
 "value": 23,  
 "value\_as\_string": "23.0"  
 }  
 },  
 {  
 "key": "3",  
 "doc\_count": 1,  
 "max\_age": {  
 "value": 20,  
 "value\_as\_string": "20.0"  
 }  
 },  
 {  
 "key": "4",  
 "doc\_count": 1,  
 "max\_age": {  
 "value": 16,  
 "value\_as\_string": "16.0"  
 }  
 }  
 ]  
}