回顾：

1）集群状态

如何快速了解集群的健康状况？green、yellow、red？

green：每个索引的primary shard和replica shard都是active状态的
yellow：每个索引的primary shard都是active状态的，但是部分replica shard不是active状态，处于不可用的状态
red：不是所有索引的primary shard都是active状态的，部分索引有数据丢失了

集群什么情况会处于一个yellow状态？

假设现在就一台linux服务器，就启动了一个es进程，相当于就只有一个node。现在es中有一个index，就是kibana自己内置建立的index。由于默认的配置是给每个index分配1个primary shard和1个replica shard，而且primary shard和replica shard不能在同一台机器上（为了容错）。现在kibana自己建立的index是1个primary shard和1个replica shard。当前就一个node，所以只有1个primary shard被分配了和启动了，但是一个replica shard没有第二台机器去启动。

测试：启动第二个es进程，就会在es集群中有2个node，然后那1个replica shard就会自动分配过去，然后cluster status就会变成green状态。

2）不同节点介绍

主节点：node.master：true
数据节点：node.data：true

-1. 客户端节点

当主节点和数据节点配置都设置为false的时候，该节点只能处理路由请求，处理搜索，分发索引操作等，从本质上来说该客户节点表现为智能负载平衡器。

独立的客户端节点在一个比较大的集群中是非常有用的，他协调主节点和数据节点，客户端节点加入集群可以得到集群的状态，根据集群的状态可以直接路由请求。

-2. 数据节点

数据节点主要是存储索引数据的节点，主要对文档进行增删改查操作，聚合操作等。数据节点对cpu，内存，io要求较高， 在优化的时候需要监控数据节点的状态，当资源不够的时候，需要在集群中添加新的节点。

-3. 主节点

主资格节点的主要职责是和集群操作相关的内容，如创建或删除索引，跟踪哪些节点是群集的一部分，并决定哪些分片分配给相关的节点。稳定的主节点对集群的健康是非常重要的，默认情况下任何一个集群中的节点都有可能被选为主节点，索引数据和搜索查询等操作会占用大量的cpu，内存，io资源，为了确保一个集群的稳定，分离主节点和数据节点是一个比较好的选择。

在一个生产集群中我们可以对这些节点的职责进行划分，建议集群中设置3台以上的节点作为master节点，这些节点只负责成为主节点，维护整个集群的状态。再根据数据量设置一批data节点，这些节点只负责存储数据，后期提供建立索引和查询索引的服务，这样的话如果用户请求比较频繁，这些节点的压力也会比较大，所以在集群中建议再设置一批client节点(node.master: false node.data: false)，这些节点只负责处理用户请求，实现请求转发，负载均衡等功能

一、ElasticSearch文档分值_score计算底层原理

1）boolean model

根据用户的query条件，先过滤出包含指定term的doc

```
query "hello world" --> hello / world / hello & world

bool --> must/must not/should --> 过滤 --> 包含 / 不包含 / 可能包含

doc --> 不打分数 --> 正或反 true or false --> 为了减少后续要计算的doc的数量，提升性能
```

2)relevance score算法，简单来说，就是计算出，一个索引中的文本，与搜索文本，他们之间的关联匹配程度

Elasticsearch使用的是 term frequency/inverse document frequency算法，简称为TF/IDF算法

Term frequency：搜索文本中的各个词条在field文本中出现了多少次，出现次数越多，就越相关

```
搜索请求：hello world


doc1：hello you，and world is very good

doc2：hello，how are you

```

Inverse document frequency：搜索文本中的各个词条在整个索引的所有文档中出现了多少次，出现的次数越多，就越不相关

```
搜索请求：hello world


doc1：hello，tuling is very good

doc2：hi world，how are you

```

比如说，在index中有1万条document，hello这个单词在所有的document中，一共出现了1000次；world这个单词在所有的document中，一共出现了100次

Field-length norm：field长度，field越长，相关度越弱

搜索请求：hello world

```
doc1：{ "title"："hello article"，"content"："...... N个单词" }
```

```
doc2: { "title": "my article", "content": "...... N个单词，hi world" }
```

hello world在整个index中出现的次数是一样多的

doc1更相关，title field更短

2、分析一个document上的_score是如何被计算出来的

```
GET /es_db/_doc/1/_explain
{
 "query": {
 "match": {
 "remark": "java developer"
 }
 }
}
```

二、分词器工作流程

切分词语，normalization

给你一段句子，然后将这段句子拆分成一个一个的单个的单词，同时对每个单词进行normalization（时态转换，单复数转换），分词器
recall，召回率：搜索的时候，增加能够搜索到的结果的数量

```
character filter：在一段文本进行分词之前，先进行预处理，比如说最常见的就是，过滤html标签（<span>hello<span> --> hello），& --> and（I&you --> I and you）

tokenizer：分词，hello you and me --> hello，you，and，me

token filter：lowercase，stop word，synonymom，liked --> like，Tom --> tom，a/the/an --> 干掉，small --> little

```

一个分词器，很重要，将一段文本进行各种处理，最后处理好的结果才会拿去建立倒排索引

2、内置分词器的介绍

```
Set the shape to semi-transparent by calling set_trans(5)

standard analyzer: set, the, shape, to, semi, transparent, by, calling, set_trans, 5（默认的是standard）

simple analyzer: set, the, shape, to, semi, transparent, by, calling, set, trans

whitespace analyzer: Set, the, shape, to, semi-transparent, by, calling, set_trans(5)

stop analyzer:移除停用词，比如a the it等等

测试：
POST _analyze
{
"analyzer":"standard",
"text":"Set the shape to semi-transparent by calling set_trans(5)"
}
```

3、定制分词器

1）默认的分词器

standard

standard tokenizer：以单词边界进行切分

standard token filter：什么都不做

lowercase token filter：将所有字母转换为小写

stop token filer（默认被禁用）：移除停用词，比如a the it等等

2）修改分词器的设置

启用english停用词token filter

```
PUT /my_index
{
 "settings": {
 "analysis": {
 "analyzer": {
 "es_std": {
 "type": "standard",
 "stopwords": "_english_"
 }
  }
  }
  }
}

GET /my_index/_analyze
{
  "analyzer": "standard",
  "text": "a dog is in the house"
}

GET /my_index/_analyze
{
  "analyzer": "es_std",
  "text":"a dog is in the house"
}

3、定制化自己的分词器

PUT /my_index
{
"settings": {
"analysis": {
"char_filter": {
"&_to_and": {
"type": "mapping",
"mappings": ["&=> and"]
}
},
"filter": {
"my_stopwords": {
```

```
"type": "stop",
"stopwords": ["the", "a"]
}
},
"analyzer": {
"my_analyzer": {
"type": "custom",
"char_filter": ["html_strip", "&_to_and"],
"tokenizer": "standard",
"filter": ["lowercase", "my_stopwords"]
}
}
}
}
}

GET /my_index/_analyze
{
"text": "tom&jerry are a friend in the house, <a>, HAHA!!",
"analyzer": "my_analyzer"
}

PUT /my_index/_mapping/my_type
{
"properties": {
"content": {
"type": "text",
"analyzer": "my_analyzer"
}
}
}
```

3）ik分词器详解

ik配置文件地址：es/plugins/ik/config目录

IKAnalyzer.cfg.xml：用来配置自定义词库

main.dic：ik原生内置的中文词库，总共有27万多条，只要是这些单词，都会被分在一起

quantifier.dic：放了一些单位相关的词
suffix.dic：放了一些后缀
surname.dic：中国的姓氏
stopword.dic：英文停用词

ik原生最重要的两个配置文件

main.dic：包含了原生的中文词语，会按照这个里面的词语去分词
stopword.dic：包含了英文的停用词

停用词，stopword

a the and at but

一般，像停用词，会在分词的时候，直接被干掉，不会建立在倒排索引中

4）IK分词器自定义词库

（1）自己建立词库：每年都会涌现一些特殊的流行词，网红，蓝瘦香菇，喊麦，鬼畜，一般不会在ik的原生词典里

自己补充自己的最新的词语，到ik的词库里面去

IKAnalyzer.cfg.xml：ext_dict，custom/mydict.dic

补充自己的词语，然后需要重启es，才能生效

（2）自己建立停用词库：比如了，的，啥，么，我们可能并不想去建立索引，让人家搜索

custom/ext_stopword.dic，已经有了常用的中文停用词，可以补充自己的停用词，然后重启es

```
IK分词器源码下载：https://github.com/medcl/elasticsearch-analysis-ik/tree
```

5）IK热更新

每次都是在es的扩展词典中，手动添加新词语，很坑

（1）每次添加完，都要重启es才能生效，非常麻烦

（2）es是分布式的，可能有数百个节点，你不能每次都一个一个节点上面去修改

es不停机，直接我们在外部某个地方添加新的词语，es中立即热加载到这些新词语

IKAnalyzer.cfg.xml

```
<properties>
 <comment>IK Analyzer 扩展配置</comment>
 <!--用户可以在这里配置自己的扩展字典 -->
 <entry key="ext_dict">location</entry>
 <!--用户可以在这里配置自己的扩展停止词字典-->
 <entry key="ext_stopwords">location</entry>
 <!--用户可以在这里配置远程扩展字典 -->
 <entry key="remote_ext_dict">words_location</entry>
 <!--用户可以在这里配置远程扩展停止词字典-->
  <entry key="remote_ext_stopwords">words_location</entry>
</properties>

```

三. 高亮显示

在搜索中，经常需要对搜索关键字做高亮显示，高亮显示也有其常用的参数，在这个案例中做一些常用参数的介绍。

现在搜索cars索引中remark字段中包含“大众”的document。并对“XX关键字”做高亮显示，高亮效果使用html标签，并设定字体为红色。如果remark数据过长，则只显示前20个字符。

```
PUT /news_website
{
 "mappings": {

 "properties": {
 "title": {
 "type": "text",
```

```
"analyzer": "ik_max_word"
},
 "content": {
 "type": "text",
 "analyzer": "ik_max_word"
 }
 }
 }

}


PUT /news_website
{
 "settings" : {
 "index" : {
 "analysis.analyzer.default.type": "ik_max_word"
 }
 }
}




PUT /news_website/_doc/1
{
 "title": "这是我写的第一篇文章",
 "content": "大家好，这是我写的第一篇文章，特别喜欢这个文章门户网站！！！"
}

GET /news_website/_doc/_search
{
 "query": {
 "match": {
 "title": "文章"
 }
 },
 "highlight": {
 "fields": {
 "title": {}
 }
 }
}
```

```
{
 "took" : 458,
 "timed_out" : false,
 "_shards" : {
 "total" : 1,
 "successful" : 1,
 "skipped" : 0,
 "failed" : 0
 },
 "hits" : {
 "total" : {
 "value" : 1,
 "relation" : "eq"
 },
 "max_score" : 0.2876821,
 "hits" : [
 {
 "_index" : "news_website",
 "_type" : "_doc",
 "_id" : "1",
 "_score" : 0.2876821,
 "_source" : {
 "title" : "我的第一篇文章",
 "content" : "大家好，这是我写的第一篇文章，特别喜欢这个文章门户网站！！！"
 },
 "highlight" : {
 "title" : [
 "我的第一篇<em>文章</em>"
 ]
 }
 }
 ]
 }
}
```

表现，会变成红色，所以说你的指定的field中，如果包含了那个搜索词的话，就会在那个field的文本中，对搜索词进行红色的高亮显示

```
GET /news_website/_doc/_search
{
 "query": {
 "bool": {
```

```
"should": [
{
"match": {
"title": "文章"
}
},
{
 "match": {
 "content": "文章"
 }
 }
 ]
 }
 },
 "highlight": {
 "fields": {
 "title": {},
 "content": {}
 }
 }
}
```

highlight中的field，必须跟query中的field一一对齐的

2、常用的highlight介绍

plain highlight，lucene highlight，默认

posting highlight，index_options=offsets

（1）性能比plain highlight要高，因为不需要重新对高亮文本进行分词
（2）对磁盘的消耗更少

```
DELETE news_website
PUT /news_website
{
 "mappings": {
 "properties": {
 "title": {
 "type": "text",
 "analyzer": "ik_max_word"
 },
```

```
 "content": {
 "type": "text",
 "analyzer": "ik_max_word",
 "index_options": "offsets"
 }
 }
 }
}

PUT /news_website/_doc/1
{
 "title": "我的第一篇文章",
 "content": "大家好，这是我写的第一篇文章，特别喜欢这个文章门户网站！！！"
}

GET /news_website/_doc/_search
{
 "query": {
 "match": {
 "content": "文章"
 }
 },
 "highlight": {
 "fields": {
 "content": {}
 }
 }
}

fast vector highlight

index-time term vector设置在mapping中，就会用fast verctor highlight

（1）对大field而言（大于1mb），性能更高

delete /news_website

PUT /news_website
{
 "mappings": {
 "properties": {
 "title": {
 "type": "text",
```

```
 "analyzer": "ik_max_word"
 },
 "content": {
 "type": "text",
 "analyzer": "ik_max_word",
 "term_vector" : "with_positions_offsets"
 }
 }
 }
}

强制使用某种highlighter，比如对于开启了term vector的field而言，可以强制使用plain highlight

GET /news_website/_doc/_search
{
 "query": {
 "match": {
 "content": "文章"
 }
 },
 "highlight": {
 "fields": {
 "content": {
 "type": "plain"
 }
 }
 }
}

总结一下，其实可以根据你的实际情况去考虑，一般情况下，用plain highlight也就足够了，不需要做其他额外的设置
如果对高亮的性能要求很高，可以尝试启用posting highlight
如果field的值特别大，超过了1M，那么可以用fast vector highlight

3、设置高亮html标签，默认是<em>标签

GET /news_website/_doc/_search
{
 "query": {
 "match": {
 "content": "文章"
 }
 },
```

```
 "highlight": {
 "pre_tags": ["<span color='red'>"],
 "post_tags": ["</span>"],
 "fields": {
 "content": {
 "type": "plain"
 }
 }
 }
}

4、高亮片段fragment的设置

GET /_search
{
 "query" : {
 "match": { "content": "文章" }
 },
 "highlight" : {
 "fields" : {
 "content" : {"fragment_size" : 150, "number_of_fragments" : 3 }
 }
 }
}

fragment_size：你一个Field的值，比如有长度是1万，但是你不可能在页面上显示这么长啊。。。设置要显示出来的fragment文本判断的长度，默认是100
number_of_fragments：你可能你的高亮的fragment文本片段有多个片段，你可以指定就显示几个片段
```

四、 聚合搜索技术深入

1.bucket和metric概念简介

bucket就是一个聚合搜索时的数据分组。如：销售部门有员工张三和李四，开发部门有员工王五和赵六。那么根据部门分组聚合得到结果就是两个bucket。销售部门bucket中有张三和李四，

开发部门 bucket中有王五和赵六。

metric就是对一个bucket数据执行的统计分析。如上述案例中，开发部门有2个员工，销售部门有2个员工，这就是metric。

metric有多种统计，如：求和，最大值，最小值，平均值等。

```
用一个大家容易理解的SQL语法来解释，如：select count(*) from table group by colum
n。那么group by column分组后的每组数据就是bucket。对每个分组执行的count(*)就是metric。
```

2. 准备案例数据

```
PUT /cars
{
"mappings": {
"properties": {
"price": {
"type": "long"
},
"color": {
"type": "keyword"
},
"brand": {
"type": "keyword"
},
"model": {
"type": "keyword"
},
"sold_date": {
"type": "date"
},
"remark" : {
"type" : "text",
"analyzer" : "ik_max_word"
}
}
}
}
```

```
POST /cars/_bulk
{ "index": {}}
{ "price" : 258000, "color" : "金色", "brand":"大众", "model" : "大众迈腾", "sol
d_date" : "2021-10-28","remark" : "大众中档车" }
{ "index": {}}
{ "price" : 123000, "color" : "金色", "brand":"大众", "model" : "大众速腾", "sol
d_date" : "2021-11-05","remark" : "大众神车" }
{ "index": {}}
{ "price" : 239800, "color" : "白色", "brand":"标志", "model" : "标志508", "sold
_date" : "2021-05-18","remark" : "标志品牌全球上市车型" }
{ "index": {}}
```

```
{ "price" : 148800, "color" : "白色", "brand":"标志", "model" : "标志408", "sold_date" : "2021-07-02","remark" : "比较大的紧凑型车" }
{ "index": {}}
{ "price" : 1998000, "color" : "黑色", "brand":"大众", "model" : "大众辉腾", "sold_date" : "2021-08-19","remark" : "大众最让人肝疼的车" }
{ "index": {}}
{ "price" : 218000, "color" : "红色", "brand":"奥迪", "model" : "奥迪A4", "sold_date" : "2021-11-05","remark" : "小资车型" }
{ "index": {}}
{ "price" : 489000, "color" : "黑色", "brand":"奥迪", "model" : "奥迪A6", "sold_date" : "2022-01-01","remark" : "政府专用？" }
{ "index": {}}
{ "price" : 1899000, "color" : "黑色", "brand":"奥迪", "model" : "奥迪A 8", "sold_date" : "2022-02-12","remark" : "很贵的大A6。。。" }
```

五. 聚合操作案例

1、根据color分组统计销售数量

只执行聚合分组，不做复杂的聚合统计。在ES中最基础的聚合为terms，相当于SQL中的count。

在ES中默认为分组数据做排序，使用的是doc_count数据执行降序排列。可以使用_key元数据，根据分组后的字段数据执行不同的排序方案，也可以根据_count元数据，根据分组后的统计值执行不同的排序方案。

```
GET /cars/_search
{
"aggs": {
"group_by_color": {
"terms": {
"field": "color",
"order": {
"_count": "desc"
}
}
}
}
}
```

2、统计不同color车辆的平均价格

本案例先根据color执行聚合分组，在此分组的基础上，对组内数据执行聚合统计，这个组内数据的聚合统计就是metric。同样可以执行排序，因为组内有聚合统

计，且对统计数据给予了命名avg_by_price，所以可以根据这个聚合统计数据字段名执行排序逻辑。

```
GET /cars/_search
{
"aggs": {
"group_by_color": {
"terms": {
"field": "color",
"order": {
"avg_by_price": "asc"
}
},
"aggs": {
"avg_by_price": {
"avg": {
"field": "price"
}
}
}
}
}
```

size可以设置为0，表示不返回ES中的文档，只返回ES聚合之后的数据，提高查询速度，当然如果你需要这些文档的话，也可以按照实际情况进行设置

```
GET /cars/_search
{
"size" : 0,
"aggs": {
"group_by_color": {
"terms": {
"field": "color"
},
"aggs": {
"group_by_brand" : {
"terms": {
"field": "brand",
"order": {
"avg_by_price": "desc"
}
},
"aggs": {
```

```
"avg_by_price": {
"avg": {
"field": "price"
}
}
}
}
}
}
}
}
```

3、统计不同color不同brand中车辆的平均价格

先根据color聚合分组，在组内根据brand再次聚合分组，这种操作可以称为下钻分析。

Aggs如果定义比较多，则会感觉语法格式混乱，aggs语法格式，有一个相对固定的结构，简单定义：aggs可以嵌套定义，可以水平定义。

嵌套定义称为下钻分析。水平定义就是平铺多个分组方式。

```
GET /index_name/type_name/_search
{
"aggs" : {
"定义分组名称（最外层）": {
"分组策略如：terms、avg、sum" : {
"field" : "根据哪一个字段分组",
"其他参数" : ""
},
"aggs" : {
 "分组名称1" : {},
 "分组名称2" : {}
}
}
}
}
```

```
GET /cars/_search
{
"aggs": {
"group_by_color": {
"terms": {
"field": "color",
```

```
"order": {
"avg_by_price_color": "asc"
}
 },
 "aggs": {
 "avg_by_price_color" : {
 "avg": {
 "field": "price"
 }
 },
 "group_by_brand" : {
 "terms": {
 "field": "brand",
 "order": {
 "avg_by_price_brand": "desc"
 }
 },
 "aggs": {
 "avg_by_price_brand": {
 "avg": {
 "field": "price"
 }
 }
 }
 }
 }
 }
 }
```

4、统计不同color中的最大和最小价格、总价

```
GET /cars/_search
{
 "aggs": {
 "group_by_color": {
 "terms": {
 "field": "color"
 },
 "aggs": {
 "max_price": {
 "max": {
```

```
"field": "price"
}
},
"min_price" : {
"min": {
"field": "price"
}
},
"sum_price" : {
"sum": {
"field": "price"
}
}
}
}
}
}
```

在常见的业务常见中，聚合分析，最常用的种类就是统计数量，最大，最小，平均，总计等。通常占有聚合业务中的60%以上的比例，小型项目中，甚至占比85%以上。

5、统计不同品牌汽车中价格排名最高的车型

在分组后，可能需要对组内的数据进行排序，并选择其中排名高的数据。那么可以使用s来实现：top_top_hithits中的属性size代表取组内多少条数据（默认为10）；sort代表组内使用什么字段什么规则排序（默认使用_doc的asc规则排序）；_source代表结果中包含document中的那些字段（默认包含全部字段）。

```
GET cars/_search
{
"size" : 0,
"aggs": {
"group_by_brand": {
"terms": {
"field": "brand"
},
"aggs": {
"top_car": {
"top_hits": {
"size": 1,
"sort": [
{
```

```
"price": {
"order": "desc"
}
}
],
"_source": {
"includes": ["model", "price"]
}
}
}
}
}
}
}
```

6、histogram 区间统计

histogram类似terms，也是进行bucket分组操作的，是根据一个field，实现数据区间分组。

如：以100万为一个范围，统计不同范围内车辆的销售量和平均价格。那么使用histogram的聚合的时候，field指定价格字段price。区间范围是100万-interval ：1000000。这个时候ES会将price价格区间划分为： [0, 1000000)，[1000000, 2000000)，[2000000, 3000000)等，依次类推。在划分区间的同时，histogram会类似terms进行数据数量的统计（count），可以通过嵌套aggs对聚合分组后的组内数据做再次聚合分析。

```
GET /cars/_search
{
"aggs": {
"histogram_by_price": {
"histogram": {
"field": "price",
"interval": 1000000
},
"aggs": {
 "avg_by_price": {
 "avg": {
 "field": "price"
 }
 }
 }
```

```
}
}
}
```

7、date_histogram区间分组

date_histogram可以对date类型的field执行区间聚合分组，如每月销量，每年销量等。

如：以月为单位，统计不同月份汽车的销售数量及销售总金额。这个时候可以使用date_histogram实现聚合分组，其中field来指定用于聚合分组的字段，interval指定区间范围（可选值有：year、quarter、month、week、day、hour、minute、second），format指定日期格式化，min_doc_count指定每个区间的最少document（如果不指定，默认为0，当区间范围内没有document时，也会显示bucket分组），extended_bounds指定起始时间和结束时间（如果不指定，默认使用字段中日期最小值所在范围和最大值所在范围为起始和结束时间）。

```
ES7.x之前的语法
GET /cars/_search
{
"aggs": {
"histogram_by_date" : {
"date_histogram": {
"field": "sold_date",
"interval": "month",
"format": "yyyy-MM-dd",
 "min_doc_count": 1,
 "extended_bounds": {
 "min": "2021-01-01",
 "max": "2022-12-31"
 }
 },
 "aggs": {
 "sum_by_price": {
 "sum": {
 "field": "price"
 }
 }
 }
 }
 }
}
```

```
执行后出现
#! Deprecation: [interval] on [date_histogram] is deprecated, use [fixed_interval] or [calendar_interval] in the future.

7.X之后
GET /cars/_search
{
"aggs": {
"histogram_by_date" : {
"date_histogram": {
"field": "sold_date",
"calendar_interval": "month",
"format": "yyyy-MM-dd",
"min_doc_count": 1,
"extended_bounds": {
"min": "2021-01-01",
"max": "2022-12-31"
}
},
"aggs": {
"sum_by_price": {
"sum": {
"field": "price"
}
}
}
}
}
```

8、_global bucket

在聚合统计数据的时候，有些时候需要对比部分数据和总体数据。

如：统计某品牌车辆平均价格和所有车辆平均价格。global是用于定义一个全局bucket，这个bucket会忽略query的条件，检索所有document进行对应的聚合统计。

```
GET /cars/_search
{
"size" : 0,
"query": {
"match": {
"brand": "大众"
}
},
```

```
"aggs": {
 "volkswagen_of_avg_price": {
 "avg": {
 "field": "price"
 }
 },
 "all_avg_price" : {
 "global": {},
 "aggs": {
 "all_of_price": {
 "avg": {
 "field": "price"
 }
 }
 }
 }
 }
```

9、aggs+order

对聚合统计数据进行排序。

如：统计每个品牌的汽车销量和销售总额，按照销售总额的降序排列。

```
GET /cars/_search
{
 "aggs": {
 "group_of_brand": {
 "terms": {
 "field": "brand",
 "order": {
 "sum_of_price": "desc"
 }
 },
 "aggs": {
 "sum_of_price": {
 "sum": {
 "field": "price"
 }
 }
 }
 }
 }
}
```

如果有多层aggs，执行下钻聚合的时候，也可以根据最内层聚合数据执行排序。

如：统计每个品牌中每种颜色车辆的销售总额，并根据销售总额降序排列。***这就像SQL中的分组排序一样，只能组内数据排序，而不能跨组实现排序。***

```
GET /cars/_search
{
"aggs": {
"group_by_brand": {
"terms": {
"field": "brand"
},
"aggs": {
"group_by_color": {
 "terms": {
 "field": "color",
 "order": {
 "sum_of_price": "desc"
 }
 },
 "aggs": {
 "sum_of_price": {
 "sum": {
 "field": "price"
 }
 }
 }
 }
 }
 }
}
```

10、search+aggs

聚合类似SQL中的group by子句，search类似SQL中的where子句。在ES中是完全可以将search和aggregations整合起来，执行相对更复杂的搜索统计。

如：统计某品牌车辆每个季度的销量和销售额。

```
GET /cars/_search
{
"query": {
```

```
"match": {
"brand": "大众"
}
},
"aggs": {
"histogram_by_date": {
 "date_histogram": {
 "field": "sold_date",
 "calendar_interval": "quarter",
 "min_doc_count": 1
 },
 "aggs": {
 "sum_by_price": {
 "sum": {
 "field": "price"
 }
 }
 }
 }
 }
 }
```

11、filter+aggs

在ES中，filter也可以和aggs组合使用，实现相对复杂的过滤聚合分析。

如：统计10万~50万之间的车辆的平均价格。

```
GET /cars/_search
{
"query": {
"constant_score": {
"filter": {
"range": {
"price": {
"gte": 100000,
"lte": 500000
 }
 }
 }
 }
},
 "aggs": {
 "avg_by_price": {
 "avg": {
```

```
"field": "price"
}
}
}
}
```

12、聚合中使用filter

filter也可以使用在aggs句法中，filter的范围决定了其过滤的范围。

如：统计某品牌汽车最近一年的销售总额。将filter放在aggs内部，代表这个过滤器只对query搜索得到的结果执行filter过滤。如果filter放在aggs外部，过滤器则会过滤所有的数据。

- 12M/M 表示 12 个月。
- 1y/y 表示 1年。
- d 表示天

```
GET /cars/_search
{
"query": {
"match": {
"brand": "大众"
}
},
"aggs": {
"count_last_year": {
 "filter": {
 "range": {
 "sold_date": {
 "gte": "now-12M"
 }
 }
 },
 "aggs": {
 "sum_of_price_last_year": {
 "sum": {
 "field": "price"
 }
 }
 }
 }
 }
}
```

文档：04 ElasticSearch笔记.note

链接：http://note.youdao.com/noteshare?id=cd119779e7328b83437b5d46e168d46a&sub=F0F437FEBC624D7F81EB70B9DD7A2CC4