# Elastic search基础

# 一 什么是es？

Elasticsearch是一个实时分布式搜索和分析引擎。基于Apache Lucene(TM)，隐藏lucene复杂性，提供restful服务。

特点：

1. 分布式的实时文件存储，每个字段都被索引并可被搜索
2. 分布式的实时分析搜索引擎
3. 可以扩展到上百台服务器，处理PB级结构化或非结构化数据

## 业界场景

维基百科使用Elasticsearch提供全文搜索并高亮关键字、Github使用Elasticsearch检索1300亿行的代码。

## 1.2 es监控工具

Marvel是Elasticsearch的管理和监控工具

## 1.3分布式特性

es自动完成功能：

1. 文档分区到不同的容器或者分片**(shards)**中，它们可以存在于一个或多个节点

中。

1. 将分片均匀的分配到各个节点，对索引和搜索做负载均衡。
2. 冗余每一个分片，防止硬件故障造成的数据丢失。
3. 将集群中任意一个节点上的请求路由到相应数据所在的节点。
4. 无论是增加节点，还是移除节点，分片都可以做到无缝的扩展和迁移。

# 二 分布式集群

主节点(master):: 临时管理集群级别的一些变更，例如新建或

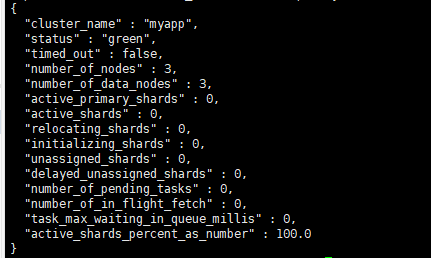
删除索引、增加或移除节点等

## 2.1 集群健康

集群健康分为三种状态:green、yellow或red。

查看es集群状态：

GET /\_cluster/health



|  |  |
| --- | --- |
| 颜色 | 意义 |
| green | **所有主要分片和复制分片都可用** |
| yellow | 所有主分片可用，部分复制分片可用 |
| red | 部分主分片可用 |

## 2.2 添加索引

**索引**：用来指向一个或者多个分片(shards)的”逻辑命名空间(logic namespace)”

### 2.2.1 分片

**分片**: 是一个最小级别的工作单元，只保存索引中所有数据的一部分。分片本质上来说就是一个lucene实例。

分片类型：主分片或复制分片，复制分片是主分片的副本，防止故障导致的数据丢失，同时提供读请求，比如搜索或者从别的分片取回文档。

创建分片

PUT /blogs

{

"settings" : {

"number\_of\_shards" : 3,

"number\_of\_replicas" : 1

}

}

分配3个主分片和一个复制分片，curl具体命令

curl -H "Content-Type: application/json" -XPUT ip地址:9200 -d ‘{

"settings" : {

"number\_of\_shards" : 3,

"number\_of\_replicas" : 1

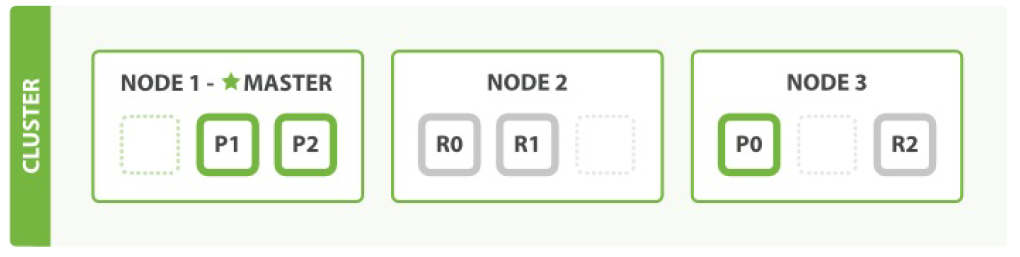
}

}’

注意 大括号前面或后面有引号。

## 2.3 横向扩展

如果集群扩展到三个节点(之前是1个)，那么此时三个分片分布如下:



复制分片的数量在集群中的节点数量发生变化时也能进行改变，数据冗余越多，能处理搜索吞吐量就越大。

PUT /blogs/\_settings

{

"number\_of\_replicas" : 2

}

可以将冗余分片增加到两个

## 2.4 集群增减节点对分片影响

杀掉一个节点的es进程，会发生如下事情

1. 如果节点是主节点，那么集群会重新选举主节点
2. 由于关掉的节点A上有主分片，导致其他节点上对应复制分片没有主分片，这时候，主节点会让节点B、C上对应复制分片有一个提升为主分片，这时候状态是yellow，因为复制分片数量要求是2，这时候集群所有主分片复制分片只有一个，

增加一个节点es进程后，集群会将复制分片分配到这个节点上达到保持复制分片数量。

# 三 数据

## 文档

Elasticsearch是一个分布式的文档(document)存储引擎。它可以实时存储并检索复杂数据结构——序列化的JSON文档。

### 文档元数据

保存文档信息，三个必须的元数据节点是：

|  |  |
| --- | --- |
| 节点 | 说明 |
| \_index | 文档存储的地方 |
| \_type | 文档代表的对象的类 |
| \_id | 文档的唯一标志 |

### \_index

索引(index)类似于关系型数据库里的“数据库”，用于存储和索引关联数据。

数据被存储和索引在分片(shards)中，索引知识一个把一个或多个分片分组在一起的逻辑空间。

## \_type

类型用来表示多个事物具有相同特征，在es中，相同类型的文档具有相同的数据结构，存储在一个索引下。每个类型(type)都有自己的映射或者结构定义。类型的映射告诉es不同的文档如何被索引。

### \_id

Id仅是一个字符串，用来在es中唯一标识一个文档。

## 索引一个文档

使用自己的ID建立文档：

PUT /{index}/{type}/{id}

{

"field": "value",

...

}

例如我们的索引叫做 “website” ，类型叫做 “blog” ，我们选择的ID是 “123” ，那么这个索引

请求就像这样：

PUT /website/blog/123

{

"title": "My first blog entry",

"text": "Just trying this out...",

"date": "2014/01/01"

}

Elasticsearch的响应：

{

"\_index": "website",

"\_type": "blog",

"\_id": "123",

"\_version": 1,

"created": true

}

\_version:每当文档发生变化，\_version都会增加

如果在添加文档没有指定id，那么es会给我们随机提供提个唯一id，长度22个字符，或者叫UUIDs。

## 检索一个文档

使用GET请求，向es发起请求：

**GET /website/blog/123?pretty**

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 1,

"found" : true,

"\_source" : {

"title": "My first blog entry",

"text": "Just trying this out...",

"date": "2014/01/01"

}

}

发现增加了\_source元数据字段。

pretty: 代表JSON文档以好看的方式打印出来。

### 3.3.1 检索文档一部分

GET /website/blog/123?\_source=title,text

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 1,

"exists" : true,

"\_source" : {

"title": "My first blog entry" ,

"text": "Just trying this out..."

}

}

只返回\_source字段不反悔其他的元数据

GET /website/blog/123/\_source

{

"title": "My first blog entry",

"text": "Just trying this out...",

"date": "2014/01/01"

}

## 检查文档是否存在

使用HEAD方法代替GET即可。

curl -i -XHEAD <http://localhost:9200/website/blog/123>

若文档存在，将返回200 OK状态

HTTP/1.1 200 OK

Content-Type: text/plain; charset=UTF-8

Content-Length: 0

若不存在

HTTP/1.1 404 Not Found

Content-Type: text/plain; charset=UTF-8

Content-Length: 0

## 更新整个文档

PUT /website/blog/123

{

title": "My first blog entry",

"text": "I am starting to get the hang of this...",

"date": "2014/01/02"

}

响应

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 2,

"created": false <1>

}

可以看到es将\_version增加了。created标志为false，是因为同索引、同类型已存在同ID文档。

Update API过程：

1. 从旧文档中检索JSON
2. 修改它
3. 删除旧文档
4. 索引新文档

## 创建新文档

\_index、\_type、\_id三者唯一确定衣蛾文档。最简单方式通过POST方法让ES自动生成唯一ID:

**POST /website/blog/**

**{ ... }**

**其次，自己指定id**

PUT /website/blog/123/\_create

{ ... }

如果成功，则响应状态码是201 created.若失败，返回409 Conflict状态码。

## 删除文档

使用DELETE方法：

DELETE /website/blog/123

如果成功，es返回200 OK状态码和以下相应体。

{

"found" : true,

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 3

}

如果文档未找到，将得到404 Not Found状态码，响应体是:

{

"found" : false,

"\_index" : "website",

"\_type" : "blog",

"\_id" : "123",

"\_version" : 4

}

尽管found为false，\_version还是增加了，主要是为了确保多节点之间不同操作能够保持正确顺序。

## 并发控制

Es并发问题

Es在并发修改的时候可能会出现数据不一致的问题，比如两个web端同时使用一个es，同时库存减一，最后发现es中库存剩余没有减2，反而只少了一个。

**悲观并发控制**:

将访问es行资源加锁，只有获取锁的线程可以修改这行数据。

**乐观并发控制**:

读写过程中进行检测，数据发生变化没有，如果发生变化，更新操作失败。可以重新尝试更新，刷新数据或者直接反馈给用户。

Es提供了\_version参数，这个参数代表document的版本号，可用于控制并发读写的乐观控制。

### 3.8.1 内部版本控制

Es提供了乐观写，

PUT /website/blog/1?version=1 <1>

{

"title": "My first blog entry",

"text": "Starting to get the hang of this..."

}

<1>代表只希望文档的\_version是1时更新文档才会有效。

如果请求成功，响应体中\_version会增加到2.

{

"\_index": "website",

"\_type": "blog",

"\_id": "1",

"\_version": 2

"created": false

}

### 3.8.2 外部版本控制

由外部程序自己制定版本号，使用方式:

PUT /website/blog/2?version=5&version\_type=external

{

"title": "My first external blog entry",

"text": "Starting to get the hang of this..."

}

增加文档时后面添加 version=版本号&version\_type=external

## 文档局部更新update

Es文档是不可变的，不能被修改只能被替换，es在使用update API处理数据时也是经过了检索-》修改-》重建索引的过程。

使用以下请求为博客

添加一个 tags 字段和一个 views 字段：

POST /website/blog/1/\_update

{

"doc" : {

"tags" : [ "testing" ],

"views": 0

}

}

更新阶段发生冲突如何解决？

Es重试机制，es提供了如下请求方式，乐观失败重试：

POST /website/pageviews/1/\_update?retry\_on\_conflict=5 <1>

{

"script" : "ctx.\_source.views+=1",

"upsert": {

"views": 0

}

}

## 检索多个文档

POST /\_mget

{

"docs" : [

{

"\_index" : "website",

"\_type" : "blog",

"\_id" : 2

},

{

"\_index" : "website",

"\_type" : "pageviews",

"\_id" : 1,

"\_source": "views"

}

]

}

## 3.11 批量更新

使用bulk命令时，REST API以\_bulk结尾,批量操作写在json文件中，官网给出的语法格式：

action\_and\_meta\_data\n

optional\_source\n

action\_and\_meta\_data\n

optional\_source\n

....

action\_and\_meta\_data\n

optional\_source\n

命令操作类型占一行 数据另起一行

如 create 格式 案例

{"index":{"\_id":"17"}} // 需要插入的数据类型 action

{"name":"cddd","id":17} // 带插入的action 对应的数据内容

1. **更新中的所有子请求异步，不互相干扰，其中一个子请求失败，整个请求会标志位失败。**
2. **批量更新请求会占用节点内存空间，需要把握好数据量大小，不能太大。一个好的批次可以保持在5-15MB之间。**

# 四 分布式增删改查

## 路由

文档分布式储存，多个节点可以为一个请求提供服务。Es提供了路由机制用于分发请求。这一功能就是路由。

文档存储到哪一个主机,是根据一个简单的算法决定的:

**shard 等于 hash(routing) % number\_of\_primary\_shards**

routing值是一个任意字符串,吗，默认是\_id也可以自定义。Routing字符串通过哈希函数生成一个数字，然后除以主切片数量得到一个余数。余数决定文档存储的主机位置。

Es规定主分片数量在创建索引时定义且不能修改，这是因为如果主分片数量发生改变，那么一个请求过来就找不到原来的分片位置了。

## 主分片和路由分片之间的交互

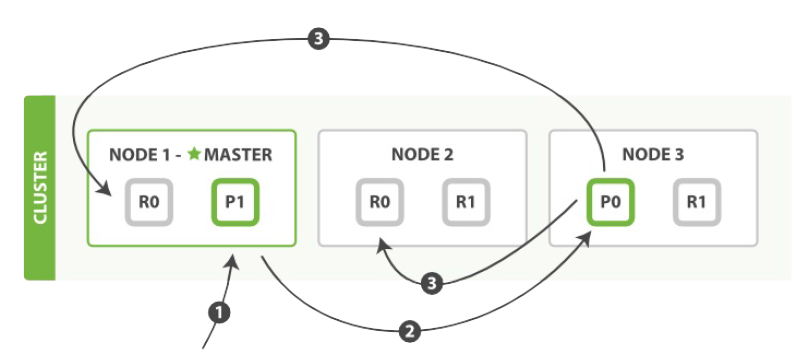
假设有三个节点的集群。它包含一个叫做 bblogs 的索引并拥有两个主分片。每个主分片有两个复制分片。相同的分片不会放在同一个节点上，所以我们的集群是这样的：



上面三个节点都可以提供服务。

### 4.2.1 新建、索引和删除文档

新建、索引和删除请求都是写(write)操作，它们必须在主分片上成功完成才能复制到相关的复制分片上。



新建、索引和删除请求途径的步骤：

1. 客户端给 Node 1 发送新建、索引或删除请求。
2. 节点使用文档\_id确定文档属于分片0.它转发请求到Node3，分片0位于这个节点上。
3. Node3在主分片上执行请求，如果成功，将转发请求到相应的位于Node1和Node2的复制节点上。当所有复制节点报告成功，Node3报告成功到请求的节点，请求的节点再报告给客户端。

### 4.2.2 replication

复制默认值是sync，代表主分片要得到所有复制分片的响应才会返回。

如果设置replication为async，请求在主分片上被执行后就会立即响应给客户端，主分片依旧会把请求转发给复制节点，但是客户端是不清楚复制节点执行请求的成功与失败的。async

复制可能会让其他机器未就绪的情况下发送过多请求而使elasticsearch负载。

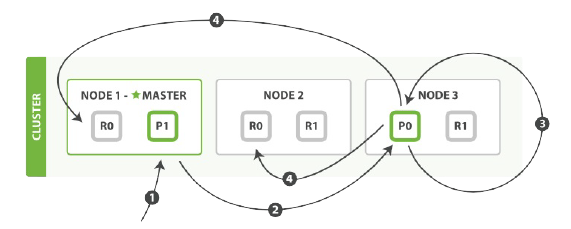
### 4.3.3 一致性

主分片尝试写入时规定数量或者过半的分片可用。防止数据被写入到错的网络分区。规定数量计算公式：

Int(primary+number\_of\_replicas)/2)+1

分片不足时，es会等待更多分片出现。默认等待一分钟。

## 4.4局部更新文档



下面给出必要的顺序步骤:

1. 客户端给任意节点发送请求
2. 如果这个节点上没有目的主分片。将请求转发到目的主分片所在主机。如果有则不用转发
3. 目的主分片所在主机修改\_source字段JSON，然后在主分片上重建索引。如果有其他进程修改了文档。它以retry\_on\_conflict设置的次数重复步骤3，都未成功则放弃。
4. 如果成功则更新文档，同时转发文档的新版本到其他复制分片所在主机，来重建索引。当所有复制节点报告成功，主分片所在节点返回成功给请求节点。请求节点返回给客户端。

# 五 搜索

## 空搜索

GET /\_search

响应内容：

{

"hits" : {

"total" : 14,

"hits" : [

{

"\_index": "us",

"\_type": "tweet",

"\_id": "7",

"\_score": 1,

"\_source": {

"date": "2014-09-17",

"name": "John Smith",

"tweet": "The Query DSL is really powerful and flexible",

"user\_id": 2

}

},

... 9 RESULTS REMOVED ...

],

"max\_score" : 1

},

"took" : 4,

"\_shards" : {

"failed" : 0,

"successful" : 10,

"total" : 10

},

"timed\_out" : false

}

### hits

响应中最重要的是hits。包含total字段表示匹配到的文档总数，页包含了搜索匹配到的文档。

Hits数组中每个结果都包含\_index、\_type和文档\_id字段，被加入到\_source字段中意味着在搜索结果中可以直接使用全部文档。

每个文档中有\_score字段，这是相关性得分(relevance score)。衡量了文档与查询的匹配程度。

### took

took代表整个搜索花费的毫秒数。

### shards

\_shards节点代表参与查询分片数量，有多少是成功的，多少是失败的。

### timeout

查询超时与否，一般，搜索请求不会超时。

## 分页

Es使用from和size参数：

from参数： 跳过开始的结果数，默认0

size: 结果数，默认10

使用方式:

GET /\_search?size=5

GET /\_search?size=5&from=5

GET /\_search?size=5&from=10

# 六 浅析Lucene

## 6.1 段

每个索引由多个段(segment)组成，每个段只会被创建一次但会被查询多次。索引期间，段经创建后不会再被修改。例如，文档被删除后，删除信息被单独保存在一个文件中，而段本身并没有修改。

多个段会在段合并阶段被合并在一起，而且要么强制执行，要么由Lucene的内在机制决定在某个时刻执行，合并后段的数量更少，但是更大。段合并非常消耗I/O，且合并期间不再使用的信息也会被清理掉。

需要强调的内容是，段合并非常消耗I/O，不要强制进行段合并，需要仔细配置段合并策略。

## 6.2 文本分析

文档数据转化为倒排索引，查询串转换为可用于搜索的词项，是需要一个转换过程的，这个过程称之为分析。

文本分析由分析器执行，而分析器包括分词器、过滤器和字符映射器组成。

### 6.2.1 Lucene分词器

Lucene分词器用来将文本切割为词条，其中携带各种额外信息词项，这些信息包括:

1. 词项在原始文本的位置，
2. 词项的长度

分词器的输出称之为词条流，这些词会被过滤器进行处理。

## 6.2.2 过滤器

过滤器主要为了移除、修改词条流中的词条，甚至可以创造新的词条。

现成过滤器包括:

1. 小写过滤器:将所有词条转化为小写。
2. ASCII过滤器: 移除词条中所有非ASCII字符。
3. 同义词过滤器: 根据同义词规则，将一个词条转换为另一个词条
4. 多语言词干还原过滤器: 将词条的文本部分规约到他们的词根形式

### 6.2.3 索引和查询

1. 查询串不是必须被分析的，例如，前缀查询不会被分析，而匹配查询会被分析。

2. 索引期与检索期的文本分析要采用同样的分析器，只有查询分析出来的词项与索引中词项能匹配上，才会返回预期的文档集。

### 6.2.4 Lucene查询语言

ElasticSearch提供了查询类型支持Apache Lucene的查询解析语法。

查询中可包含布尔操作符，用于连接多个词项，使之构成从句:

1. AND:文档需要匹配AND前后两个词项
2. OR：文档需要匹配任意一个OR前后词项
3. NOT: 文档必须不匹配NOT后面的词项。
4. +:必须包含+操作符后面词项的文档才会被认为与从句匹配。例如,查找哪些必须包含lucene，但是apache可出现不可出现的文档，可执行查询: +lucene apache
5. -: 与从句匹配的文档不能出现-操作符后的词项。例如，查找哪些包含lucene但不包含elasticsearch的文档，可执行查询:+lucene-elasticsearch

## 6.3 ElasticSearch基本概念

### 6.3.1 索引

Es索引类似于关系型数据库中的数据库，但由于可用性机制，es会有多个副本。

### 6.3.2 文档

文档相当于数据库中的行，由字段构成，每个字段有其字段名以及一个或者多个字段值。ES文档最终存储到Lucene文档了。

### 6.3.3 映射

映射决定如何将文本切割为词条，哪些应该被过滤掉，或哪些附加处理是必须调用的。

### 6.3.4 类型

ES中每个文档都有与之对应的类型定义，允许一个索引中存储多种文档类型，相当于关系型数据库中的表。

### 6.3.5 分片

分片由Lucene索引构成，副本分片可以缓解但分片的压力，也是该分片具有高可用特性。

### 6.3.6 网关

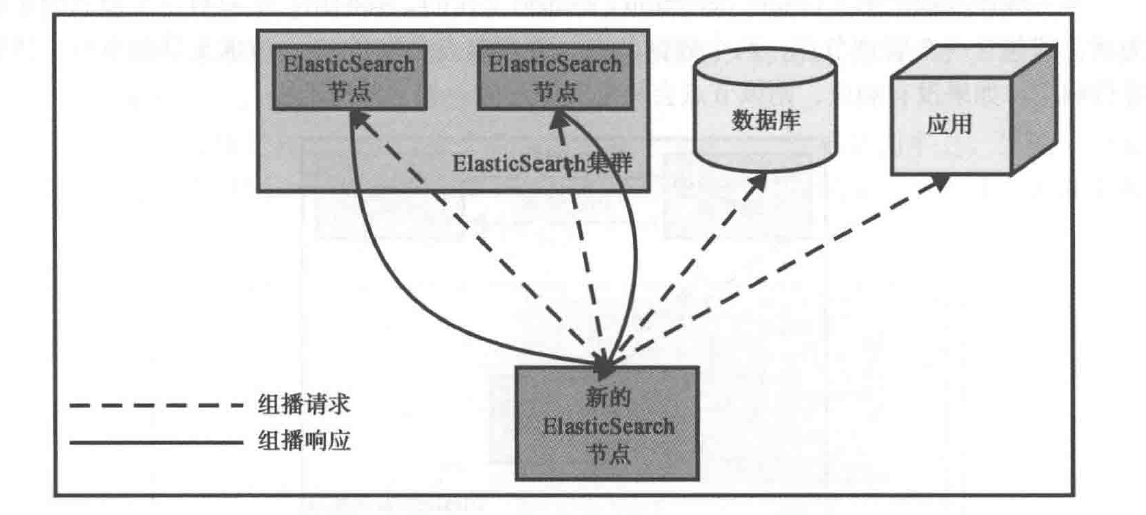
用于收集集群状态，并持久化。

## 6.4 ElasticSearch工作流程

主要介绍es如何进行工作

### 6.4.1 启动过程

1. 当ElasticSearch节点启动时，它使用广播技术来发现同一个集群中的其他节点，并与它们进行链接。



2 集群有节点被选举为管理节点。该节点负责集群状态管理以及集群拓扑发生变化时做出反应，分发索引分片到集群相应节点上。

1. 管理节点读取集群状态信息，并在必要时进行恢复处理。在这个节点，管理节点会检查所有索引分片，并决定哪些分片用于主分片。整个集群进入黄色状态。

### 6.4.2 故障检测

集群管理节点监控所有可用节点，检查它们是否正在工作。如果任何节点在预定义的超时时间内没有响应，则认为该节点已经断开。并会将其身上的分片进行转移。对于丢失的主分片，一个主分片会从其副本分片中选择。

## 6.5 检索数据

### 6.5.1 索引数据

创建索引的四种方式:

1. 使用索引API
2. 使用bulk API
3. 使用UDP bulk API
4. 使用插件发送数据，成为河流

### 6.5.2 查询数据

查询API占据ElasticSearch API大部分内容，使用查询DSL(基于JSON的可用于构建复杂查询语言).，能够做如下事情:

1. 使用各种查询类型。比如:简单的词项查询、短语查询、范围查询、布尔查询、模糊查询、区间查询、通配符查询、空间查询。
2. 组合查询
3. 文档过滤
4. 相似查询

查询操作过程：

查询分为两个阶段：分散阶段和合并阶段。

分散阶段将查询分发到包含相关文档的多个分片中执行查询

合并阶段则从众多分片中手机返回结果。然后对它们进行合并、排序、后续处理，然后返回给客户端。

# 七 查询DSL进阶

主要包括Lucene评分公式工作原理。

## 7.1 文档得分要素

ElasticSearch为了计算文档得分，需要考虑以下因子:

1. 文档权重:索引期赋予某个文档的权重值
2. 字段权重:查询器赋予某个字段的权重值
3. 协调因子：基于文档中词项命中个数的协调因子，文档命中了查询中的词项越多，得分越高
4. 逆文档频率：告诉评分公式词项在文档中的罕见程度。逆文档频率越低，词项越罕见。
5. 长度范数：每个字段的基于词项个数的归一化因子(索引期计算出来并存储在索引中)。一个字段包含的词项越多，该因子的权重越低。意味着Lucene评分公式更喜欢包含更少词项的字段。
6. 词频： 基于词项的因子，表示一个词项在文档出现了多少次。词频越高，文档得分越高
7. 查询范数：基于查询的归一化因子，等于查询中词项的权重平方和。

### 7.2 Lucene实际评分公式



得分公式是关于查询数q和文档d的函数，

评分基本原则:

1. 越多罕见的词项被匹配，文档得分越高
2. 文档字段越短，文档得分越高
3. 权重越高（索引期和查询期）,文档得分越高

## 7.3 二次评分

重新计算查询返回的文档中指定个数文档的得分。这意味着es会截取查询返回文档的前N个。并使用预定义的二次评分来重新计算得分。

## 7.4 排序

包括排序和多值排序

# 八 es性能优化实践

ES 性能调优

ES 的默认配置，是综合了数据可靠性、写入速度、搜索实时性等因素。实际使用时，我们需要根据公司要求，进行偏向性的优化。

## 8.1 写优化

假设我们的应用场景要求是，每秒 300 万的写入速度，每条 500 字节左右。

针对这种对于搜索性能要求不高，但是对写入要求较高的场景，我们需要尽可能的选择恰当写优化策略。

综合来说，可以考虑以下几个方面来提升写索引的性能：

1. 加大 Translog Flush ，目的是降低 Iops、Writeblock。
2. 增加 Index Refresh 间隔，目的是减少 Segment Merge 的次数。
3. 调整 Bulk 线程池和队列。
4. 优化节点间的任务分布。
5. 优化 Lucene 层的索引建立，目的是降低 CPU 及 IO。

### 8.1.1 批量提交

ES 提供了 Bulk API 支持批量操作，当我们有大量的写任务时，可以使用 Bulk 来进行批量写入。

每次提交的数据量为多少时，能达到最优的性能，主要受到文件大小、网络情况、数据类型、集群状态等因素影响。

**通用的策略如下：**Bulk 默认设置批量提交的数据量不能超过 100M。数据条数一般是根据文档的大小和服务器性能而定的，但是单次批处理的数据大小应从 5MB～15MB 逐渐增加，当性能没有提升时，把这个数据量作为最大值。

$ vi request

$ cat request

{ "index" : { "\_index" : "chandler","\_type": "test", "\_id" : "1" } }

{ "name" : "钱丁君","age": "18" }

$ curl -s -H "Content-Type: application/json" -XPOST localhost:9200/\_bulk --data-binary @request; echo

{"took":214,"errors":false,"items":[{"index":{"\_index":"chandler","\_type":"test","\_id":"1","\_version":1,"result":"created","\_shards":{"total":2,"successful":1,"failed":0},"\_seq\_no":0,"\_primary\_term":1,"status":201}}]}

$ curl -XGET localhost:9200/chandler/test/1?pretty

{

"\_index" : "chandler",

"\_type" : "test",

"\_id" : "1",

"\_version" : 1,

"found" : true,

"\_source" : {

"name" : "钱丁君",

"age" : "18"

}

}

Bulk 不支持 Gget 操作，因为没什么用处。

### 8.1.2 优化存储设备

ES 是一种密集使用磁盘的应用，在段合并的时候会频繁操作磁盘，所以对磁盘要求较高，当磁盘速度提升之后，集群的整体性能会大幅度提高。

磁盘的选择，提供以下几点建议：

1. **使用固态硬盘（Solid State Disk）替代机械硬盘。**SSD 与机械磁盘相比，具有高效的读写速度和稳定性。
2. **使用 RAID 0。**RAID 0 条带化存储，可以提升磁盘读写效率。
3. 在 ES 的服务器上挂载多块硬盘。使用多块硬盘同时进行读写操作提升效率，在配置文件 ES 中设置多个存储路径，如下所示：

避免使用 NFS（Network File System）等远程存储设备，网络的延迟对性能的影响是很大的。

### 8.1.3 合理使用合并

Lucene 以段的形式存储数据。当有新的数据写入索引时，Lucene 就会自动创建一个新的段。

随着数据量的变化，段的数量会越来越多，消耗的多文件句柄数及 CPU 就越多，查询效率就会下降。

由于 Lucene 段合并的计算量庞大，会消耗大量的 I/O，所以 ES 默认采用较保守的策略，让后台定期进行段合并，如下所述：

1. 索引写入效率下降:

当段合并的速度落后于索引写入的速度时，ES 会把索引的线程数量减少到 1。这样可以避免出现堆积的段数量爆发，同时在日志中打印出“now throttling indexing”INFO 级别的“警告”信息。

1. 提升段合并速度

ES 默认对段合并的速度是 20m/s，如果使用了 SSD，我们可以通过以下的命令将这个合并的速度增加到 100m/s。

PUT /\_cluster/settings

{

"persistent" : {

"indices.store.throttle.max\_bytes\_per\_sec" : "100mb"

}

}

### 8.1.4 减少Refresh次数

Lucene 在新增数据时，采用了延迟写入的策略，默认情况下索引的 refresh\_interval 为 1 秒。 Lucene 将待写入的数据先写到内存中，超过 1 秒（默认）时就会触发一次 Refresh，然后 Refresh 会把内存中的的数据刷新到操作系统的文件缓存系统中。 如果我们对搜索的实效性要求不高，可以将 Refresh 周期延长，例如 30 秒。这样还可以有效地减少段刷新次数，但这同时意味着需要消耗更多的Heap内存。

index.refresh\_interval:30s

### 8.1.5 加大Flush设置

Flush 的主要目的是把文件缓存系统中的段持久化到硬盘，当 Translog 的数据量达到 512MB 或者 30 分钟时，会触发一次 Flush。 index.translog.flush\_threshold\_size 参数的默认值是 512MB，我们进行修改。

增加参数值意味着文件缓存系统中可能需要存储更多的数据，所以我们需要为操作系统的文件缓存系统留下足够的空间。

### 8.1.6 减少副本数量

ES 为了保证集群的可用性，提供了 Replicas（副本）支持，然而每个副本也会执行分析、索引及可能的合并过程，所以 Replicas 的数量会严重影响写索引的效率。

当写索引时，需要把写入的数据都同步到副本节点，副本节点越多，写索引的效率就越慢。

如果我们需要大批量进行写入操作，可以先禁止 Replica 复制，设置 index.number\_of\_replicas: 0 关闭副本。在写入完成后，Replica 修改回正常的状态。

## 8.2 读优化

8.2.1 避免大结果集和深翻

在上一篇讲到了集群中的查询流程，例如，要查询从 from 开始的 size 条数据，则需要在每个分片中查询打分排名在前面的 from+size 条数据。

协同节点将收集到的n×(from+size)条数据聚合，再进行一次排序，然后从 from+size 开始返回 size 条数据。

当 from、size 或者 n 中有一个值很大的时候，需要参加排序的数量也会增长，这样的查询会消耗很多 CPU 资源，从而导致效率的降低。

为了提升查询效率，ES 提供了 Scroll 和 Scroll-Scan 这两种查询模式。

###### ****Scroll：****是为检索大量的结果而设计的。例如，我们需要查询 1～100 页的数据，每页 100 条数据。

如果使用 Search 查询：每次都需要在每个分片上查询得分最高的 from+100 条数据，然后协同节点把收集到的 n×(from+100)条数据聚合起来再进行一次排序。

每次返回 from+1 开始的 100 条数据，并且要重复执行 100 次。

如果使用 Scroll 查询：在各个分片上查询 10000 条数据，协同节点聚合 n×10000 条数据进行合并、排序，并将排名前 10000 的结果快照起来。这样做的好处是减少了查询和排序的次数。

###### ****Scroll-Scan：****Scroll 是先做一次初始化搜索把所有符合搜索条件的结果缓存起来生成一个快照，然后持续地、批量地从快照里拉取数据直到没有数据剩下。

###### 而这时对索引数据的插入、删除、更新都不会影响遍历结果，因此 Scroll 并不适合用来做实时搜索。

###### 

###### 其思路和使用方式与 Scroll 非常相似，但是 Scroll-Scan 关闭了 Scroll 中最耗时的文本相似度计算和排序，使得性能更加高效。

### 8.2.2 选择合适的路由

ES 内在机制决定了拥有相同路由属性的文档，一定会被分配到同一个分片上，无论是主分片还是副本。

查询时可以根据 Routing 信息，直接定位到目标分片，避免查询所有的分片，再经过协调节点二次排序。

### 8.2.3 定期删除

由于在 Lucene 中段具有不变性，每次进行删除操作后不会立即从硬盘中进行实际的删除，而是产生一个 .del 文件记录删除动作。

随着删除操作的增长，.del 文件会越来也多。当我们进行查询操作的时候，被删除的数据还会参与检索中，然后根据 .del 文件进行过滤。.del 文件越多，查询过滤过程越长，进而影响查询的效率。

当机器空闲时，我们可以通过如下命令删除文件，来提升查询的效率：

$ curl -XPOST localhost:9200/chandler/\_forcemerge?only\_expunge\_deletes=true

{"\_shards":{"total":10,"successful":5,"failed":0}}

定期对不再更新的索引做 optimize (ES 2.0 以后更改为 Force Merge API)。

这 Optimze 的实质是对 Segment File 强制做合并，可以节省大量的 Segment Memory。

## 8.3 堆大小设置

ES 默认安装后设置的内存是 1GB，对于任何一个现实业务来说，这个设置都太小了。

如果是通过解压安装的 ES，则在 ES 安装文件中包含一个 jvm.option 文件，添加如下命令来设置 ES 的堆大小：

-Xms10g

-Xmx10g

因为 ES 堆内存的分配需要满足以下两个原则：

* **不要超过物理内存的 50%：**Lucene 的设计目的是把底层 OS 里的数据缓存到内存中。

Lucene 的段是分别存储到单个文件中的，这些文件都是不会变化的，所以很利于缓存，同时操作系统也会把这些段文件缓存起来，以便更快的访问。

如果我们设置的堆内存过大，Lucene 可用的内存将会减少，就会严重影响降低 Lucene 的全文本查询性能。

* **堆内存的大小最好不要超过 32GB：**在 Java 中，所有对象都分配在堆上，然后有一个 Klass Pointer 指针指向它的类元数据。

这个指针在 64 位的操作系统上为 64 位，64 位的操作系统可以使用更多的内存（2^64）。在 32 位的系统上为 32 位，32 位的操作系统的最大寻址空间为 4GB（2^32）。

但是 64 位的指针意味着更大的浪费，因为你的指针本身大了。浪费内存不算，更糟糕的是，更大的指针在主内存和缓存器（例如 LLC, L1等）之间移动数据的时候，会占用更多的带宽。

8.4 关闭swapping

在计算机中运行的程序均需在内存执行，若内存消耗殆尽将导致程序无法进行。为了解决这个问题，操作系统使用一种叫作虚拟内存的技术。

当内存耗尽时，操作系统就会自动把内存中暂时不使用的数据交换到硬盘中，需要使用的时候再从硬盘交换到内存。

如果内存交换到磁盘上需要 10 毫秒，从磁盘交换到内存需要 20 毫秒，那么多的操作时延累加起来，将导致几何级增长。

不难看出 Swapping 对于性能是多么可怕。所以为了使 ES 有更好等性能，强烈建议关闭 Swap。

关闭 Swap 的方式如下：

**①暂时禁用。**如果我们想要在 Linux 服务器上暂时关闭，可以执行如下命令，但在服务器重启后失效：

sudo swapoff -a

**②永久性关闭。**我们可以修改 /etc/sysctl.conf（不同的操作系统路径有可能不同），增加如下参数：

vm.swappiness = 1 //0-100，则表示越倾向于使用虚拟内存。

注意：Swappiness 设置为 1 比设置为 0 要好，因为在一些内核版本，Swappness=0 会引发 OOM（内存溢出）。