**分布式文档存储**

在上一章，我们看到了将数据放入索引然后检索它们的所有方法。不过我们有意略过了许多关于数据是如何在集群中分布和获取的相关技术细节。这种使用和细节分离是刻意为之的——你不需要知道数据在Elasticsearch如何分布它就会很好的工作。

这一章我们深入这些内部细节来帮助你更好的理解数据是如何在分布式系统中存储的。

注意：

下面的信息只是出于兴趣阅读，你不必为了使用Elasticsearch而弄懂和记住所有的细节。讨论的这些选项只提供给高级用户。

阅读这一部分只是让你了解下系统如何工作，并让你知道这些信息以备以后参考，所以不要被细节吓到。

4.1 路由

**路由文档到分片**

当你索引一个文档，它被存储在单独一个主分片上。Elasticsearch是如何知道文档属于哪个分片的呢？当你创建一个新文档，它是如何知道是应该存储在分片1还是分片2上的呢？

进程不能是随机的，因为我们将来要检索文档。事实上，它根据一个简单的算法决定：

shard = hash(routing) % number\_of\_primary\_shards

routing值是一个任意字符串，它默认是\_id但也可以自定义。这个routing字符串通过哈希函数生成一个数字，然后除以主切片的数量得到一个**余数(remainder)**，余数的范围永远是0到number\_of\_primary\_shards - 1，这个数字就是特定文档所在的分片。

这也解释了为什么主分片的数量只能在创建索引时定义且不能修改（me：与memcache的一致性哈希不同，memcache集群可以随意的撤掉和加入节点而只会影响到很少的数据）：如果主分片的数量在未来改变了，所有先前的路由值就失效了，文档也就永远找不到了。

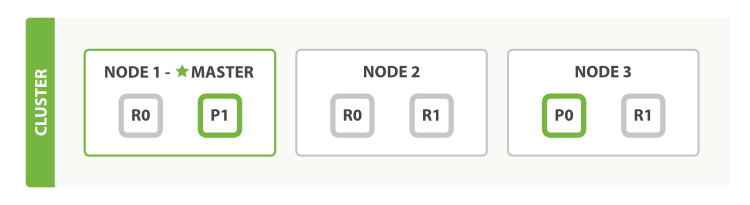
有时用户认为固定数量的主分片会让之后的扩展变得很困难。现实中，有些技术会在你需要的时候让扩展变得容易。我们将在《扩展》章节讨论。

所有的文档API（get、index、delete、bulk、update、mget）都接收一个routing参数，它用来自定义文档到分片的映射。自定义路由值可以确保所有相关文档——例如属于同一个人的文档——被保存在同一分片上。我们将在《扩展》章节说明你为什么需要这么做。

## 4.2 分片交互

## 主分片和复制分片如何交互

为了阐述意图，我们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（me:P0和P1）。每个主分片有两个复制分片（me：R0和R1）。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：



我们能够发送请求给集群中任意一个节点。每个节点都有能力处理任意请求。每个节点都知道任意文档所在的节点(me：通过上一节中的算法)，所以也可以将请求转发到需要的节点。下面的例子中，我们将发送所有请求给Node 1，这个节点我们将会称之为**请求节点(requesting node)**

### 提示：

当我们发送请求，最好的做法是循环通过所有节点请求，这样可以平衡负载。

4.3 新建、索引和删除

## 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。



下面我们罗列在主分片和复制分片上成功新建、索引或删除一个文档必要的顺序步骤：

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

客户端接收到成功响应的时候，文档的修改已经被应用于主分片和所有的复制分片。你的修改生效了。

有很多可选的请求参数允许你更改这一过程。你可能想牺牲一些安全来提高性能。这一选项很少使用因为Elasticsearch已经足够快，不过为了内容的完整我们将做一些阐述。

### replication

复制默认的值是sync。这将导致主分片得到复制分片的成功响应后才返回。

如果你设置replication为async，请求在主分片上被执行后就会返回给客户端。它依旧会转发请求给复制节点，但你将不知道复制节点成功与否。

上面的这个选项不建议使用。默认的sync复制允许Elasticsearch强制反馈传输。async复制可能会因为在不等待其它分片就绪的情况下发送过多的请求而使Elasticsearch过载。

### consistency

默认主分片在尝试写入时需要**规定数量(quorum)**或过半的分片（可以是主节点或复制节点）可用。这是防止数据被写入到错的网络分区。规定的数量计算公式如下：

1

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

consistency允许的值为one（只有一个主分片），all（所有主分片和复制分片）或者默认的quorum或过半分片。

注意number\_of\_replicas是在索引中的设置，用来定义复制分片的数量，而不是现在活动的复制节点的数量。如果你定义了索引有3个复制节点，那规定数量是：

int( (primary + 3 replicas) / 2 ) + 1 = 3

但如果你只有2个节点，那你的活动分片不够规定数量，也就不能索引或删除任何文档。

1

### timeout

当分片副本不足时会怎样？Elasticsearch会等待更多的分片出现。默认等待一分钟。如果需要，你可以设置timeout参数让它终止的更早：100表示100毫秒，30s表示30秒。

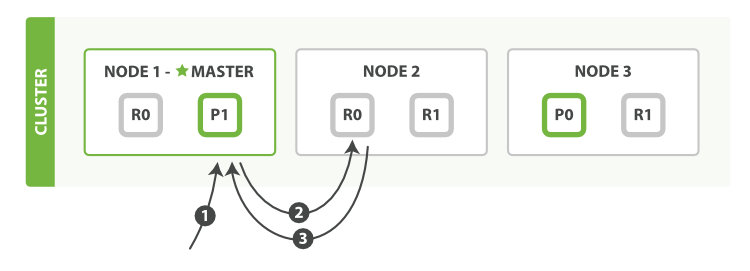
注意：

新索引默认有1个复制分片，这意味着为了满足quorum的要求**需要**两个活动的分片。当然，这个默认设置将阻止我们在单一节点集群中进行操作。为了避开这个问题，规定数量只有在number\_of\_replicas大于一时才生效。

4.4检索

**检索文档**

文档能够从主分片或任意一个复制分片被检索（me：写入的时候只能是主分片）。



下面我们罗列在主分片或复制分片上检索一个文档必要的顺序步骤：

1. 客户端给Node 1发送get请求。
2. 节点使用文档的\_id确定文档属于分片0。分片0对应的复制分片在三个节点上都有。此时，它转发请求到Node 2。
3. Node 2返回文档(document)给Node 1然后返回给客户端。

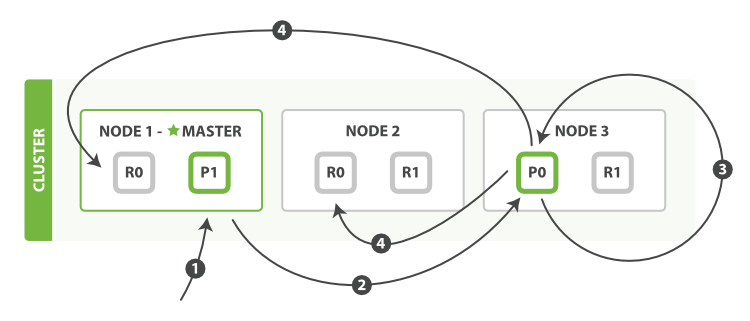
对于读请求，为了平衡负载，请求节点会为每个请求选择不同的分片——它会循环所有分片副本。

可能的情况是，一个被索引的文档已经存在于主分片上却还没来得及同步到复制分片上。这时复制分片会报告文档未找到，主分片会成功返回文档。一旦索引请求成功返回给用户，文档则在主分片和复制分片都是可用的（me：否则可能是复制分片还未将主分片的数据同步过来）。

4.5局部更新

## 局部更新文档

update API 结合了之前提到的读和写的模式。



下面我们罗列执行局部更新必要的顺序步骤：

1. 客户端给Node 1发送更新请求。
2. 它转发请求到主分片所在节点Node 3。
3. Node 3从主分片检索出文档，修改\_source字段的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Node 3成功更新文档，它同时转发文档的新版本到Node 1和Node 2上的复制节点以重建索引。当所有复制节点报告成功，Node 3返回成功给请求节点，然后返回给客户端。

update API还接受《新建、索引和删除》章节提到的routing、replication、consistency和timout参数。

1

### 基于文档的复制

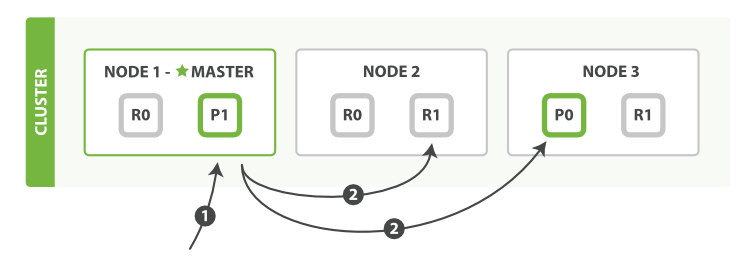
当主分片转发更改给复制分片时，并不是转发更新请求，而是转发整个文档的新版本。记住这些修改转发到复制节点是异步的，它们并不能保证到达的顺序与发送相同。如果Elasticsearch转发的仅仅是修改请求，修改的顺序可能是错误的，那得到的就是个损坏的文档（me：转发修改后的整个文档使得文档不容易因为异步的原因而损坏）。

4.5 批量请求

**多文档模式**

mget和bulk API与单独的文档类似。差别是请求节点知道每个文档所在的分片。它把多文档请求拆成**每个分片**的对文档请求，然后转发每个参与的节点。

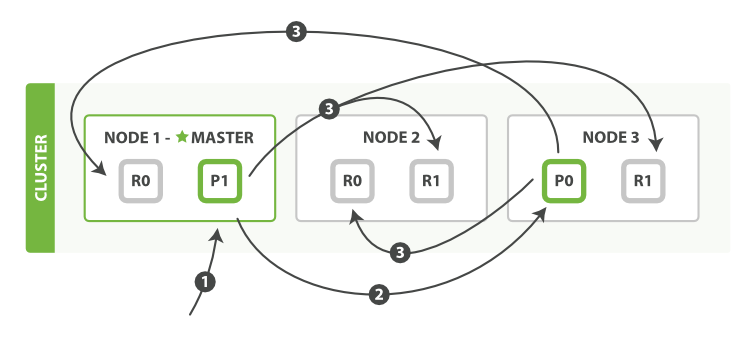
一旦接收到每个节点的应答，然后整理这些响应组合为一个单独的响应，最后返回给客户端。



下面我们将罗列通过一个mget请求检索多个文档的顺序步骤：

1. 客户端向Node 1发送mget请求。
2. Node 1为每个分片构建一个多条数据检索请求（me：需要耗时），然后转发到这些请求所需的主分片或复制分片上。当所有回复被接收，Node 1构建响应并返回给客户端。

routing 参数可以被docs中的每个文档设置。



下面我们将罗列使用一个bulk执行多个create、index、delete和update请求的顺序步骤：

1. 客户端向Node 1发送bulk请求。
2. Node 1为每个分片构建批量请求，然后转发到这些请求所需的主分片上。
3. 主分片一个接一个的按序执行操作。当一个操作执行完，主分片转发新文档（或者删除部分）给对应的复制节点，然后执行下一个操作。一旦所有复制节点报告所有操作已成功完成，节点就报告success给请求节点，后者(请求节点)整理响应并返回给客户端。

bulk API还可以在最上层使用replication和consistency参数，routing参数则在每个请求的元数据中使用。

4．7 批量格式

**为什么是奇怪的格式？**

当我们在《批量》一章中学习了批量请求后，你可能会问：“为什么bulk API需要带换行符的奇怪格式，而不是像mget API一样使用JSON数组？”

为了回答这个问题，我们需要简单的介绍一下背景：

批量中每个引用的文档属于不同的主分片，每个分片可能被分布于集群中的某个节点上。这意味着批量中的每个**操作(action)**需要被转发到对应的分片和节点上。

如果每个单独的请求被包装到JSON数组中，那意味着我们需要：

* 解析JSON为数组（包括文档数据，可能非常大）
* 检查每个请求决定应该到哪个分片上
* 为每个分片创建一个请求的数组
* 序列化这些数组为内部传输格式
* 发送请求到每个分片

这可行，但需要大量的RAM来承载本质上相同的数据，还要创建更多的数据结构使得JVM花更多的时间执行垃圾回收。

取而代之的，Elasticsearch则是从网络缓冲区中一行一行的直接读取数据。它使用换行符识别和解析**action/metadata**行，以决定哪些分片来处理这个请求。

这些行请求直接转发到对应的分片上。这些没有冗余复制，没有多余的数据结构。整个请求过程使用最小的内存在进行。

5 搜索

## 搜索——基本的工具

到目前为止，我们已经学会了如何使用elasticsearch作为一个简单的NoSQL风格的分布式文件存储器——我们可以将一个JSON文档扔给Elasticsearch，也可以根据ID检索它们。但Elasticsearch真正强大之处在于可以从混乱的数据中找出有意义的信息——从大数据到全面的信息。

这也是为什么我们使用结构化的JSON文档，而不是无结构的二进制数据。Elasticsearch不只会**存储(store)**文档，也会**索引(indexes)**文档内容来使之可以被搜索。

2

**每个文档里的字段都会被索引并被查询**。而且不仅如此。在简单查询时，Elasticsearch可以使用**所有**的索引，以非常快的速度返回结果。这让你永远不必考虑传统数据库的一些东西。

A search can be: **搜索(search)**可以：

* 在类似于gender或者age这样的字段上使用结构化查询，join\_date这样的字段上使用排序，就像SQL的结构化查询一样。
* 全文检索，可以使用所有字段来匹配关键字，然后按照**关联性(relevance)**排序返回结果。
* 或者结合以上两条。

很多搜索都是开箱即用的，为了充分挖掘Elasticsearch的潜力，你需要理解以下三个概念：

| **概念** | **解释** |
| --- | --- |
| **映射(Mapping)** | 数据在每个字段中的解释说明 |
| **分析(Analysis)** | 全文是如何处理的可以被搜索的 |
| **领域特定语言查询(Query DSL)** | Elasticsearch使用的灵活的、强大的查询语言 |

以上提到的每个点都是一个巨大的话题，我们将在《深入搜索》一章阐述它们。本章节我们将介绍这三点的一些基本概念——仅仅帮助你大致了解搜索是如何工作的。

我们将使用最简单的形式开始介绍search API.

### 测试数据

本章节测试用的数据可以在这里被找到<https://gist.github.com/clintongormley/8579281>

1

你可以把这些命令复制到终端中执行以便可以实践本章的例子。

5.1 空搜素

# 空搜索

最基本的搜索API表单是**空搜索(empty search)**，它没有指定任何的查询条件，只返回集群索引中的所有文档：

GET /\_search

响应内容（为了编辑简洁）类似于这样：

{

"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

响应中最重要的部分是hits，它包含了total字段来表示匹配到的文档总数，hits数组还包含了匹配到的前10条数据。

hits数组中的每个结果都包含\_index、\_type和文档的\_id字段，被加入到\_source字段中这意味着在搜索结果中我们将可以直接使用全部文档。这不像其他搜索引擎只返回文档ID，需要你单独去获取文档。

5

每个节点都有一个\_score字段，这是**相关性得分(relevance score)，它衡量了文档与查询的匹配程度。**默认的，返回的结果中关联性最大的文档排在首位；这意味着，它是按照\_score降序排列的。这种情况下，我们没有指定任何查询，所以所有文档的相关性是一样的，因此所有结果的\_score都是取得一个中间值1

2

max\_score指的是所有文档匹配查询中\_score的最大值。

## took

took告诉我们整个搜索请求花费的毫秒数。

## shards

\_shards节点告诉我们参与查询的分片数（total字段），有多少是成功的（successful字段），有多少的是失败的（failed字段）。通常我们不希望分片失败，不过这个有可能发生。如果我们遭受一些重大的故障导致主分片和复制分片都故障，那这个分片的数据将无法响应给搜索请求。这种情况下，Elasticsearch将报告分片failed，但仍将继续返回剩余分片上的结果。

## timeout

time\_out值告诉我们查询超时与否。一般的，搜索请求不会超时。如果响应速度比完整的结果更重要，你可以定义timeout参数为10或者10ms（10毫秒），或者1s（1秒）

GET /\_search?timeout=10ms

Elasticsearch将返回在请求超时前收集到的结果。

超时不是一个断路器（circuit breaker）（译者注：关于断路器的理解请看警告）。

1

## 警告

需要注意的是timeout不会停止执行查询，它仅仅告诉你**目前**顺利返回结果的节点然后关闭连接。在后台，其他分片可能依旧执行查询，尽管结果已经被发送。

使用超时是因为对于你的业务需求（译者注：SLA，Service-Level Agreement服务等级协议，在此我翻译为业务需求）来说非常重要，而不是因为你想中断执行长时间运行的查询。

5.2多索引和多类型

## 多索引和多类别

你注意到空搜索的结果中不同类型的文档——user和tweet——来自于不同的索引——us和gb。

通过限制搜索的不同索引或类型，我们可以在集群中跨**所有**文档搜索。Elasticsearch转发搜索请求到集群中平行的主分片或每个分片的复制分片上，收集结果后选择顶部十个返回给我们。

通常，当然，你可能想搜索一个或几个自定的索引或类型，我们能通过定义URL中的索引或类型达到这个目的，像这样：

#### /\_search

在所有索引的所有类型中搜索

#### /gb/\_search

在索引gb的所有类型中搜索

#### /gb,us/\_search

在索引gb和us的所有类型中搜索

#### /g\*,u\*/\_search

在以g或u开头的索引的所有类型中搜索

#### /gb/user/\_search

在索引gb的类型user中搜索

#### /gb,us/user,tweet/\_search

在索引gb和us的类型为user和tweet中搜索

#### /\_all/user,tweet/\_search

在所有索引的user和tweet中搜索 search types user and tweet in all indices

1

当你搜索包含单一索引时，Elasticsearch转发搜索请求到这个索引的主分片或每个分片的复制分片上，然后聚集每个分片的结果。搜索包含多个索引也是同样的方式——只不过或有更多的分片被关联。

1

## 重要

搜索一个索引有5个主分片和5个索引各有一个分片**事实上是一样的**。

1

接下来，你将看到这些简单的情况如何灵活的扩展以适应你需求的变更。

5.3分页

## 分页

《空搜索》一节告诉我们在集群中有14个文档匹配我们的（空）搜索语句。但是只有10个文档在hits数组中。我们如何看到其他文档？

和SQL使用LIMIT关键字返回只有一页的结果一样，Elasticsearch接受from和size参数：

size: 结果数，默认10

from: 跳过开始的结果数，默认0

如果你想每页显示5个结果，页码从1到3，那请求如下：

GET /\_search?size=5

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

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

应该当心分页太深或者一次请求太多的结果。结果在返回前会被排序。但是记住一个搜索请求常常涉及多个分片。每个分片生成自己排好序的结果，它们接着需要集中起来排序以确保整体排序正确（me：类似mapreduce分类和聚合）。

### 在集群系统中深度分页

为了理解为什么深度分页是有问题的，让我们假设在一个有5个主分片的索引中搜索。当我们请求结果的第一页（结果1到10）时，每个分片产生自己最顶端10个结果然后返回它们给**请求节点(requesting node)**，它再排序这所有的50个结果以选出顶端的10个结果。

现在假设我们请求第1000页——结果10001到10010。工作方式都相同，不同的是每个分片都必须产生顶端的10010个结果。然后请求节点排序这50050个结果并丢弃50040个！

1

你可以看到在分布式系统中，排序结果的花费随着分页的深入而成倍增长。这也是为什么网络搜索引擎中任何语句不能返回多于1000个结果的原因。

### TIP

在《重建索引》章节我们将阐述如何能高效的检索大量文档

5.4查询字符串

## 简易搜索

search API有两种表单：一种是“简易版”的**查询字符串(query string)**将所有参数通过查询字符串定义，另一种版本使用JSON完整的表示**请求体(request body)**，这种富搜索语言叫做结构化查询语句（DSL）

查询字符串搜索对于在命令行下运行**点对点(ad hoc)**查询特别有用。例如这个语句查询所有类型为tweet并在tweet字段中包含elasticsearch字符的文档：

2

GET /\_all/tweet/\_search?q=tweet:elasticsearch

下一个语句查找name字段中包含"john"和tweet字段包含"mary"的结果。实际的查询只需要：

+name:john +tweet:mary

但是**百分比编码(percent encoding)**（译者注：就是url编码）需要将查询字符串参数变得更加神秘：

GET /\_search?q=%2Bname%3Ajohn+%2Btweet%3Amary

"+"前缀表示语句匹配条件**必须**被满足。类似的"-"前缀表示条件**必须不**被满足。所有条件如果没有+或-表示是可选的——匹配越多，相关的文档就越多。

### \_all字段

返回包含"mary"字符的所有文档的简单搜索：

GET /\_search?q=mary

在前一个例子中，我们搜索tweet或name字段中包含某个字符的结果。然而，这个语句返回的结果在三个不同的字段中包含"mary"：

* 用户的名字是“Mary”
* “Mary”发的六个推文
* 针对“@mary”的一个推文

Elasticsearch是如何设法找到三个不同字段的结果的？

当你索引一个文档，Elasticsearch把所有字符串字段值连接起来放在一个大字符串中，它被索引为一个特殊的字段\_all。例如，当索引这个文档：

{

"tweet": "However did I manage before Elasticsearch?",

"date": "2014-09-14",

"name": "Mary Jones",

"user\_id": 1

}

这好比我们增加了一个叫做\_all的额外字段值：

"However did I manage before Elasticsearch? 2014-09-14 Mary Jones 1"

若没有指定字段，查询字符串搜索（即q=xxx）使用\_all字段搜索。

### TIP

\_all字段对于开始一个新应用时是一个有用的特性。之后，如果你定义字段来代替\_all字段，你的搜索结果将更加可控。当\_all字段不再使用，你可以停用它，这个会在《全字段》章节阐述。

1

### 更复杂的语句

下一个搜索推特的语句：

\_all field

* name字段包含"mary"或"john"
* date晚于2014-09-10
* \_all字段包含"aggregations"或"geo"

+name:(mary john) +date:>2014-09-10 +(aggregations geo)

编码后的查询字符串变得不太容易阅读：

?q=%2Bname%3A(mary+john)+%2Bdate%3A%3E2014-09-10+%2B(aggregations+geo)

就像你上面看到的例子，**简单(lite)**查询字符串搜索惊人的强大。它的查询语法，会在《查询字符串语法》章节阐述。参考文档允许我们简洁明快的表示复杂的查询。这对于命令行下一次性查询或者开发模式下非常有用。

然而，你可以看到简洁带来了隐晦和调试困难。而且它很脆弱——查询字符串中一个细小的语法错误，像-、:、/或"错位就会导致返回错误而不是结果。

最后，查询字符串搜索允许任意用户在索引中任何一个字段上运行潜在的慢查询语句，可能暴露私有信息甚至使你的集群瘫痪。

### TIP

因为这些原因，我们不建议直接暴露查询字符串搜索给用户，除非这些用户对于你的数据和集群可信。

取而代之的，生产环境我们一般依赖全功能的**请求体**搜索API，它能完成前面所有的事情，甚至更多。在了解它们之前，我们首先需要看看数据是如何在Elasticsearch中被索引的。