分布式文　存

在前面的章　，我　介　了如何索引和　　数据，不　我　忽略了很多底　的技　　，例如文件是如何分布到集群的，又是如何从集群中　取的。　Elasticsearch　本意就是　藏　些底　　，　我　好　注在　　中，所以其　　不必了解　　深入也无妨。

在　个章　中，我　将深入探索　些核心的技　　，　能　助　更好地理解数据如何被存　到　个分布式系　中。

> 注意
>
> 　个章　包含了一些高　　，上面也提到　，就算　不　住和理解所有的　　然能正常使用 Elasticsearch。如果　有　趣的　，　个章　可以作　　的　外　趣　物，　展　的知　面。
>
> 如果　在　　个章　的　候感到很吃力，也不用担心。　　个章　　只是用来告 Elasticsearch　　是如何工作的，　　将来在工作中如果　需要用到　个章　提供的知　，可以再回　　来翻　。

路由一个文　到一个分片中

当索引一个文　的　候，文　会被存　到一个主分片中。　Elasticsearch　如何知道一个文　　存放到　个分片中　？当我　　建文　　，它如何决定　个文　　当被存　在分片 1　是分片 2 中　？

首先　肯定不会是随机的，否　将来要　取文　的　候我　就不知道从何　　了。　　上，　个　程是根据下面　个公式决定的：

```
shard = hash(routing) % number_of_primary_shards
```

`routing`　是一个可　　，　　是文　的　`_id`　，也可以　置成一个自定　的　。　`routing`　通　　hash 函数生成一个数字，然后　个数字再除以 `number_of_primary_shards`　（主分片的数量）后得到　余数　。　个分布在 `0` 到 `number_of_primary_shards-1` 之　的余数，就是我　所　求的文　所在分片的位置。

　就解　了　什　我　要在　建索引的　候就　定好主分片的数量　　并且永　不会改　　个数量：因　如果数量　化了，那　所有之前路由的　都会无效，文　也再也　不到了。

NOTE	可能　得由于　Elasticsearch　主分片数量是固定的会使索引　以　行　容。　　上当　需要　有很多技巧可以　松　　容。我　将会在[scale]一章中提到更多有　水平　展的内容。

所有的文　API（`get`　、　`index`　、　`delete`　、　`bulk`　、　`update` 以及 `mget`　）都接受一个叫做 `routing` 的路由参数　　，通　　个参数我　可以自定　文　到分片的映射。一个自定　的路由参数可以用来保所有相　的文　——例如所有属于同一个用　的文　——都被存　到同一个分片中。我　也会在[scale]一章中　　什　会有　　一　需求。

主分片和副本分片如何交互

了 明目的，我 假 有一个集群由三个 点 成。 它包含一个叫 blogs 的索引，有 个主分片，个主分片有 个副本分片。相同分片的副本不会放在同一 点，所以我 的集群看起来像 有三个点和一个索引的集群。

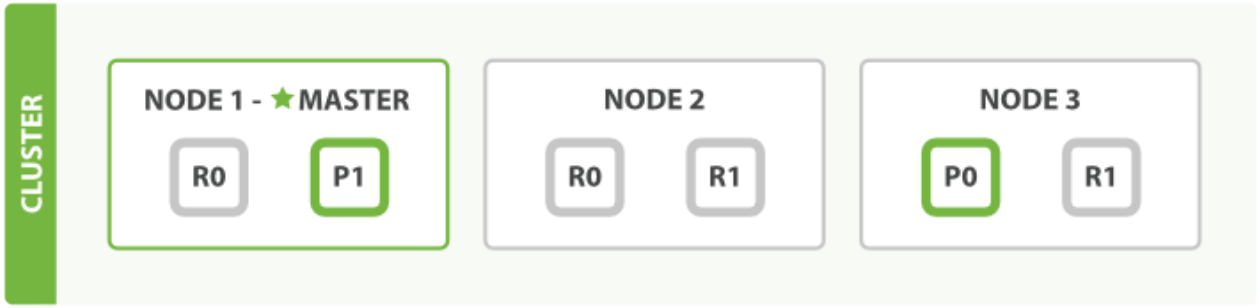

Figure 1. 有三个 点和一个索引的集群

我 可以 送 求到集群中的任一 点。 个 点都有能力 理任意 求。 个点都知道集群中任一文 位置，所以可以直接将 求 到需要的 点上。 在下面的例子中，将所有的求 送到 Node 1，我 将其称 点*(coordinating node)*。

TIP 当 送 求的 候， 了 展 ，更好的做法是 集群中所有的 点。

新建、索引和 除文

新建、索引和 除 求都是 写 操作， 必 在主分片上面完成之后才能被 制到相 的副本分片，如下所示 新建、索引和 除 个文 .

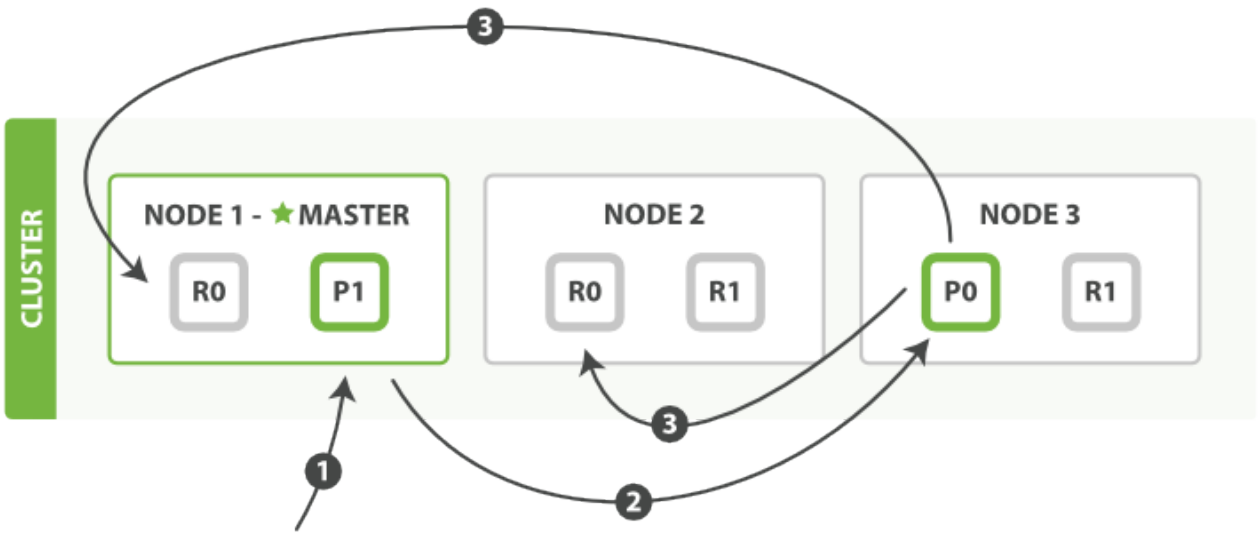

Figure 2. 新建、索引和 除 个文

以下是在主副分片和任何副本分片上面 成功新建，索引和 除文 所需要的 序：

1. 客 端向 Node 1 送新建、索引或者 除 求。
2. 点使用文 的 _id 定文 属于分片 0 。 求会被 到 Node 3，因 分片 0 的主分片目前被分配在

Node 3 上。

3. Node 3 在主分片上面 行 求。如果成功了，它将 求并行 到 Node 1 和 Node 2 的副本分片上。一旦所有的副本分片都 告成功，Node 3 将向 点 告成功， 点向客 端 告成功。

在客 端收到成功 ，文 更已 在主分片和所有副本分片 行完成， 更是安全的。

有一些可 的 求参数允 影 个 程，可能以数据安全 代 提升性能。 些 很少使用，因 Elasticsearch已 很快，但是 了完整起 ，在 里 述如下：

consistency

consistency，即一致性。在 置下，即使 是在 行一个_写_操作之前，主分片都会要求 必 要有 定数量*(quorum)*（或者 法，也即必 要有大多数）的分片副本 于活 可用状 ，才会去 行_写_操作(其中分片副本可以是主分片或者副本分片)。 是 了避免在 生 分区故障（network partition）的 候 行_写_操作， 而 致数据不一致。_ 定数量_即：

```
int( (primary + number_of_replicas) / 2 ) + 1
```

consistency 参数的 可以 one （只要主分片状 ok 就允 行_写_操作）,all（必 要主分片和所有副本分片的状 没 才允 行_写_操作）， 或 quorum 。 quorum ，即大多数的分片副本状 没 就允 行_写_操作。

注意， 定数量 的 算公式中 number_of_replicas 指的是在索引 置中的 定副本分片数，而不是指当前 理活 状 的副本分片数。如果 的索引 置中指定了当前索引 有三个副本分片，那 定数量的 算 果即：

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

如果此 只 个 点，那 于活 状 的分片副本数量就 不到 定数量，也因此 将无法索引和 除任何文 。

timeout

如果没有足 的副本分片会 生什 ？ Elasticsearch会等待，希望更多的分片出 。 情况下，它最多等待1分 。 如果 需要， 可以使用 timeout 参数 使它更早 止： 100 100秒，30s 是30秒。

NOTE

新索引 有 1 个副本分片， 意味着 足 定数量 需要 个活 的分片副本。但是， 些 的 置会阻止我 在 一 点上做任何事情。 了避免 个 ，要求只有当 number_of_replicas 大于1的 候， 定数量才会 行。

取回一个文

可以从主分片或者从其它任意副本分片 索文 ，如下 所示 取回 个文 .

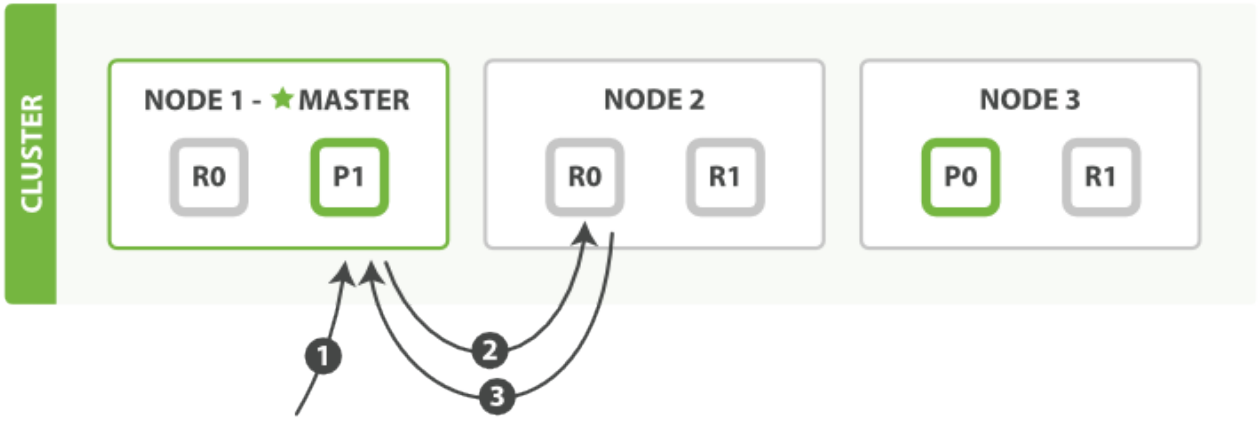

Figure 3. 取回　个文

以下是从主分片或者副本分片　索文　的　　序：

1、客　端向 Node 1　送　取　求。

2、　点使用文　的　_id　来　定文　属于分片　0　。分片　0　的副本分片存在于所有的三个　点上。　在　情况下，它将　求　　到 Node 2 。

3、Node 2 将文　返回　Node 1 ，然后将文　返回　客　端。

在　理　取　求　，　　点在　次　求的　候都会通　　所有的副本分片来　到　均衡。

在文　被　索　，已　被索引的文　可能已　存在于主分片上但是　没有　制到副本分片。　　在　情况下，副本分片可能会　告文　不存在，但是主分片可能成功返回文　。　一旦索引　求成功返回用　，文　在主分片和副本分片都是可用的。

局部更新文

如 局部更新文　所示，update API　合了先前　明的　取和写入模式。

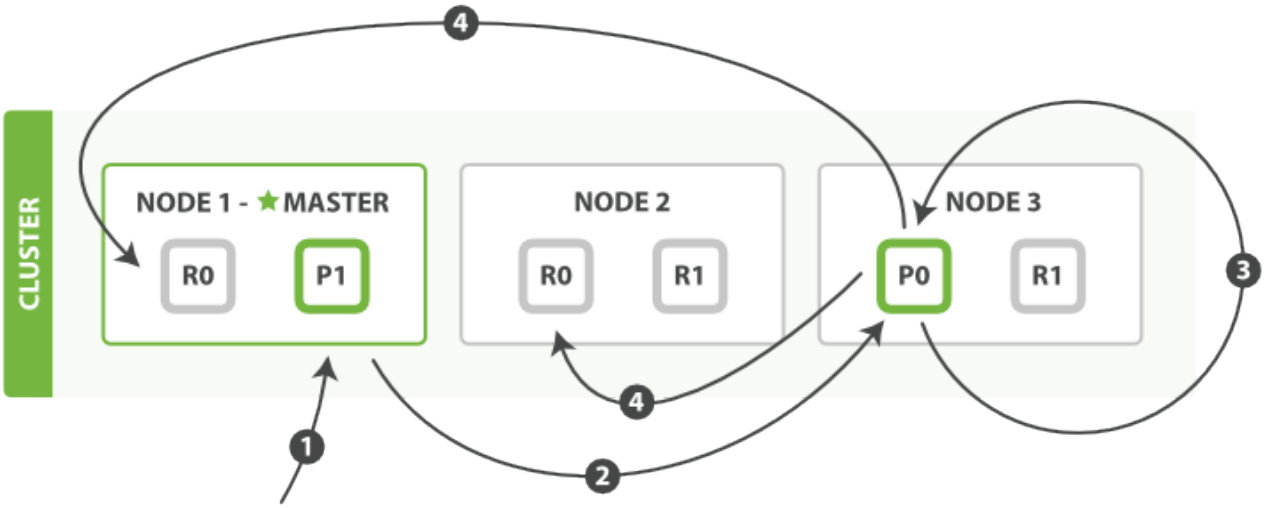

Figure 4. 局部更新文

以下是部分更新一个文　的　　：

1. 客 端向 Node 1 送更新 求。
2. 它将 求 到主分片所在的 Node 3 。
3. Node 3 从主分片 索文 ，修改 _source 字段中的 JSON ，并且 重新索引主分片的文 。 如果文 已 被 一个 程修改，它会重 3 ，超 retry_on_conflict 次后放 。
4. 如果 Node 3 成功地更新文 ，它将新版本的文 并行 到 Node 1 和 Node 2 上的副本分片，重新建立索引。 一旦所有副本分片都返回成功， Node 3 向 点也返回成功， 点向客 端返回成功。

update API 接受在 新建、索引和 除文 章 中介 的 routing 、 replication 、 consistency 和 timeout 参数。

> **基于文 的 制**
>
> 当主分片把更改 到副本分片 ， 它不会 更新 求。 相反，它 完整文 的新版本。 住， 些更改将会 到副本分片，并且不能保 它 以 送它 相同的 序到 。 如果Elasticsearch 更改 求， 可能以 的 序 用更改， 致得到 坏的文 。

多文 模式

mget 和 bulk API 的模式 似于 文 模式。区 在于 点知道 个文 存在于 个分片中。 它将整个多文 求分解成 个分片 的多文 求，并且将 些 求并行 到 个参与 点。

点一旦收到来自 个 点的 答，就将 个 点的 收集整理成 个 ，返回 客 端，如 使用 mget 取回多个文 所示。

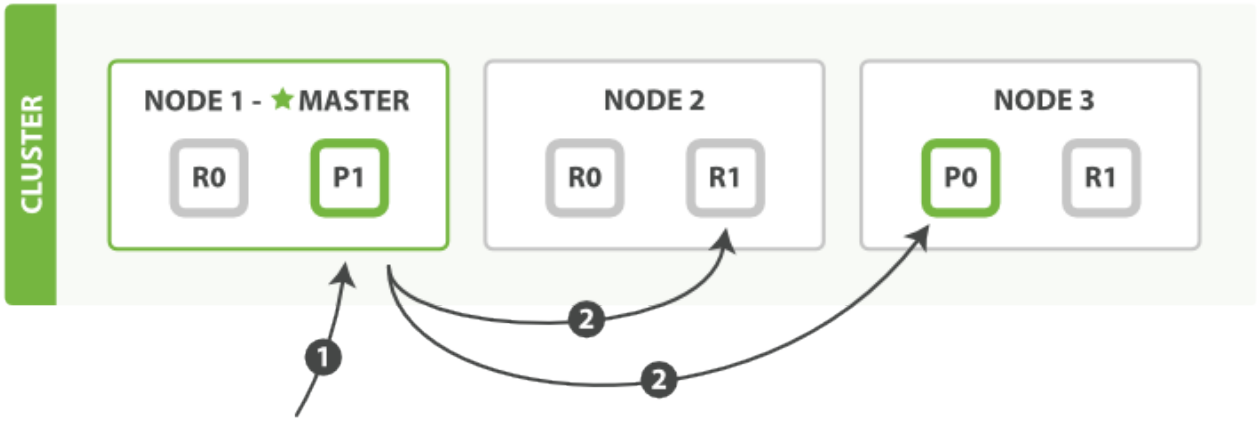

Figure 5. 使用 mget 取回多个文

以下是使用 个 mget 求取回多个文 所需的 序：

1. 客 端向 Node 1 送 mget 求。
2. Node 1 个分片 建多文 取 求，然后并行 些 求到托管在 个所需的主分片或者副本分片的 点上。一旦收到所有答 ， Node 1 建 并将其返回 客 端。

可以 docs 数 中 个文 置 routing 参数。

bulk API，如使用 bulk 修改多个文档所示，允许在单个批量请求中执行多个创建、索引、删除和更新请求。

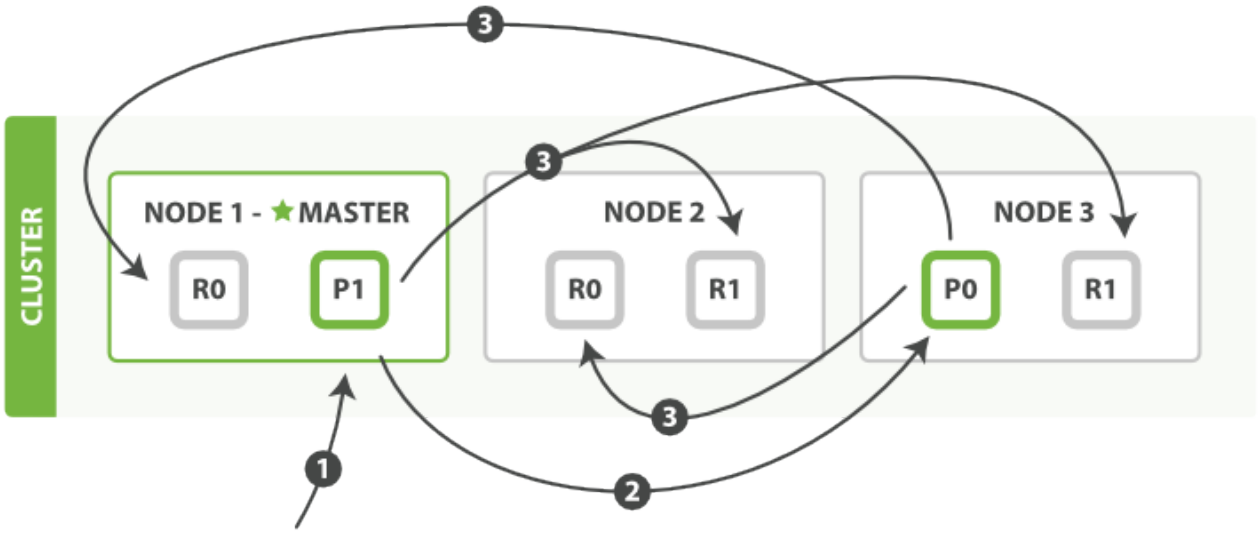

Figure 6. 使用 bulk 修改多个文档

bulk API 按如下顺序执行：

1. 客户端向 Node 1 发送 bulk 请求。
2. Node 1 为每个节点创建一个批量请求，并将这些请求并行转发到每个包含主分片的节点主机。
3. 主分片一个接一个按顺序执行每个操作。当每个操作成功时，主分片并行转发新文档（或删除）到副本分片，然后执行下一个操作。一旦所有的副本分片报告所有操作成功，节点将向协调节点报告成功，协调节点将这些响应收集整理并返回客户端。

bulk API 还可以在整个批量请求的最顶层使用 consistency 参数，以及在每个请求中的元数据中使用 routing 参数。

什么是有趣的格式？

当我们早些时候在[bulk]文档章节了解批量请求时，可能会自己，为什么 bulk API 需要有换行符的有趣格式，而不是将请求包装在 JSON 数组中的请求，例如 mget API？"。

为了回答这一点，我们需要解释一点背景：在批量请求中引用的每个文档可能属于不同的主分片，每个文档可能被分配给集群中的任何节点。这意味着批量请求 bulk 中的每个操作都需要被转到正确节点上的正确分片。

如果每个请求被包装在 JSON 数组中，那就意味着我们需要执行以下操作：

- 将 JSON 解析为数组（包括文档数据，可以非常大）
- 决定对每个请求以及将被转到哪个分片
- 为每个分片创建一个请求数组
- 将这些数组序列化为内部传输格式
- 将请求发送到每个分片

是可行的，但需要大量的 RAM 来存 原本相同的数据的副本，并将 建更多的数据 ，Java虚机（JVM）将不得不花 行 回收。

相反，Elasticsearch可以直接 取被 冲区接收的原始数据。 它使用 行符字符来 和解析小的action/metadata 行来决定 个分片 理 个 求。

些原始 求会被直接 到正 的分片。没有冗余的数据 制，没有浪 的数据 。整个 求尽可能在最小的内存中 理。