Elasticsearch是准实时搜索平台。这意味着有轻微的延迟（通常为1秒）就可以从入库建索引文件到已经进行关键字搜索。

虽然在一个索引中有多分文档，但这些文档的结构是一致的，并在第一次存储的时候指定。

一个复制的分片不会存储在同一个节点中

每个Elasticsearch分片是一个Lucene的索引。有文档，你可以在一个单一的Lucene索引中存储的最大值为lucene-5843，极限是2147483519（= integer.max\_value - 128）个文档。你可以使用\_cat/shards api监控碎片的大小。

## Elasticsearch索引问题

在elasticsearch中存储数据的行为叫做索引，不过在索引之前，我们需要明确数据应该存储在哪里。在elasticsearch中，文档归属于一个类型，而这些类型存在于索引中，可以类比传统关系型数据库来说明：

Relational DB -> Databases -> Tables -> Rows -> Columns

Elasticsearch -> Indices -> Types -> Documents -> Fields

Elasticsearch集群可以包含多个索引（indices）（数据库），每个索引可以包含多个类型（types）（表），每一个类型包含多个文档（行），每个文档包含多个字段（Field）（列）。

### 索引含义的区别

索引（名词），一个索引（index）就像是传统关系数据库中的**数据库**，它是相关文档存储的地方，index的复数是**indices** 或**indexes**。

索引（动词） **「索引一个文档」**表示把一个文档存储到**索引（名词）**里，以便它可以被检索或者查询。这很像SQL中的INSERT关键字，差别是，如果文档已经存在，新的文档将覆盖旧的文档。

倒排索引 传统数据库为特定列增加一个索引，例如B-Tree索引来加速检索。Elasticsearch和Lucene使用一种叫做**倒排索引(inverted index)**的数据结构来达到相同目的。

默认情况下，文档中的所有字段都会被索引（拥有一个倒排索引），只有这样他们才是可被搜索的。

### 创建索引

文档通过index API被索引——使数据可以被存储和搜索。首先需要确定文档所在，文档通过其\_index、\_type、\_id唯一确定，可以自己提供一个\_id，或者也可使用index API为我们生成一个。如果你的文档有自然的标识符（例如user\_account字段或者其他值表示文档），你就可以提供自己的\_id，使用这种形式的index API：

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

}

响应指出请求的索引已经被成功创建，这个索引中包含\_index、\_type和\_id元数据，以及一个新元素：\_version。

Elasticsearch中每个文档都有版本号，每当文档变化（包括删除）都会使\_version增加。在《版本控制》章节中我们将探讨如何使用\_version号确保你程序的一部分不会覆盖掉另一部分所做的更改。

#### 自增ID

如果我们的数据没有自然ID，我们可以让Elasticsearch自动为我们生成。请求结构发生了变化：PUT方法——“在这个URL中存储文档”变成了POST方法——"在这个类型下存储文档"。（译者注：原来是把文档存储到某个ID对应的空间，现在是把这个文档添加到某个\_type下）。

URL现在只包含\_index和\_type两个字段：

POST /website/blog/

{

"title": "My second blog entry",

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

"date": "2014/01/01"

}

响应内容与刚才类似，只有\_id字段变成了自动生成的值：

{

"\_index": "website",

"\_type": "blog",

"\_id": "wM0OSFhDQXGZAWDf0-drSA",

"\_version": 1,

"created": true

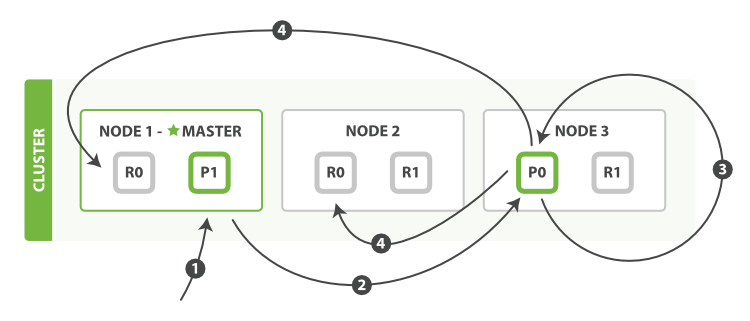
}

自动生成的ID有22个字符长，URL-safe, Base64-encoded string universally unique identifiers, 或者叫 UUIDs。

### 如何修改数据

#### 局部更新

Update API结合了之前提到的读和写的模式



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。

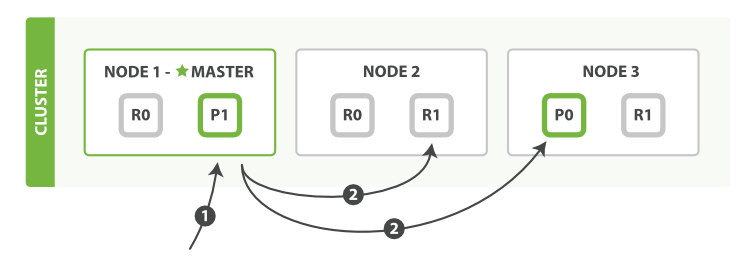
如果Node 3成功更新文档，它同时转发文档的新版本到Node 1和Node 2上的复制节点以重建索引。当所有复制节点报告成功，Node 3返回成功给请求节点，然后返回给客户端

### 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。

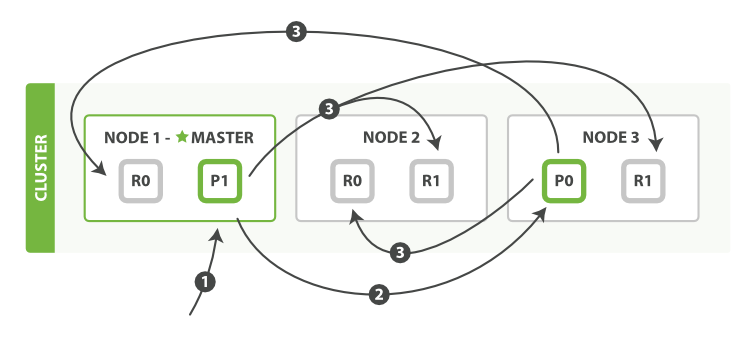
一旦接收到每个节点的应答，然后整理这些响应组合为一个单独的响应，最后返回给客户端。



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，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。

### 批量格式

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。

## 存储的问题

Elasticsearch采用混合mmap fs /nio fs为默认的方式存储索引。默认的操作系统mmap设置可能是太低，这可能会导致内存溢出异常。在Linux系统中，您可以用root账户通过以下命令来增加限制：

sysctl -w vm.max\_map\_count=262144

  如果要想使设置永久生效，可以设置/etc/sysctl.conf中的vm.max\_map\_count值。

  注意：如果使用安装包(.deb, .rpm)来进行安装，此值会自动进行设置。

#### 文件系统的存储类型

  有不同的文件系统的有不同存储类型。这个参数一般会自动选择：Windows 64bit操作系统一般是mmapfs，在Windows 32位操作系统一般是simplefs，Elasticsearch默认为（niofs/mmapfs混合方式）。这个参数可以配置使所有的索引生效，配置文件为config/elasticsearch.yml。

配置项：index.store.type: niofs

  也可以在建索引的时候指定类型：

PUT localhost:9200/my\_index{

  "settings": {

    "index.store.type": "niofs"

  }

}

注意：这个设置有可能在未来被删除。

可选择的参数

Simple FS（简单文件系统）  
   Simplefs类型是一个简单的实现随机访问文件的文件存储系统（映射到Lucene SimpleFsDirectory的）。该实现的并发性能较差（多线程是个瓶颈）。当你需要将索引持久化时，最好使用niofs。  
NIO FS（NIO文件系统）  
   niofs类型是通过NIO将分片索引文件写到文件系统上（映射到Lucene NIOFSDirectory）。它允许多线程同时读取文件。不建议在Windows系统上使用，由于JAVA在实现上有一个错误。  
MMap FS（内存映射文件系统）  
   mmapfs类型通过映射文件到内存中（MMAP）存储分片索引到文件系统上（映射到Lucene MMapDirectory）。内存映射的过程中将划分出与被映射文件大小一样的虚拟内存空间。使用这个类之前，请确保您有足够的虚拟地址空间。

default\_fs

  默认类型是混合NiO FS和mmapfs，系统会自动选择操作系统匹配最好的文件系统。目前只有Lucene的term dictionary和doc values采用内存映射的方式来减少对操作系统的影响。所有其他的方式都使用Lucene niofsdirectory打开。