第 7 章

深入Elasticsearch集群

上一章介绍了更多Elasticsearch的数据分析能力，使用聚合和切面为索引中的数据赋予意义，还通过使用Elasticsearch建议器为应用程序引入了拼写检查和自动完成功能。通过使用预匹配器，我们创建了警报功能，并且使用附件功能把二进制文件编入索引。我们索引和搜索地理空间数据，并使用卷动API有效获取大量的结果。最后，使用词条查找机制，加快了提取词条列表的查询过程。在本章，你将学习以下内容：

- 理解节点的发现机制、配置和调优；
- 恢复和时光之门模块的控制；
- 为高查询和高索引用例准备Elasticsearch；
- 使用索引模板和动态映射。

7.1 节点发现

启动一个Elasticsearch节点时，该节点会开始寻找具有相同集群名字并且可见的主节点。如果找到主节点，该节点加入一个已经组成了的集群；如果没有找到，该节点成为主节点（如果配置允许）。形成集群和寻找节点的过程称为发现。负责发现的模块有两个主要目的：选出一个主节点和发现集群中的新节点。本节将讨论如何配置和优化发现模块。

7.1.1 发现的类型

默认在没有安装额外插件的情况下，Elasticsearch允许使用zen发现，它提供了多播和单播发现。在计算机网络术语中，多播（http://en.wikipedia.org/wiki/Multicast）是指在单个传输中将消息传递到一组计算机中。单播（http://en.wikipedia.org/wiki/Unicast）指的是一次只通过网络传输单条消息到单个主机上。

当使用多播发现时，Elasticsearch试图找到所有能够接收和响应多播消息的节点。如果使用单播方法，你需要提供集群中的至少一部分主机，节点会尝试连接到它们。

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

选择多播还是单播，首先你应该知道你的网络能否处理多播消息。如果它可以，使用多播将更简单。如果你的网络不能处理多播，请使用单播发现。另一个使用单播发现的原因是安全：不想任何节点误加入你的集群中。所以，如果要运行多个集群，或者开发人员的计算机和集群在同一个网络中，使用单播是更好的选择。

如果你使用的是Linux操作系统，并希望检查你的网络是否支持多播，请对你的网络接口（通常是eth0）使用`ifconfig`命令。如果你的网络支持多播，你会从前面命令的响应中看到`MULTICAST`属性。

7.1.2　主节点

我们已经看到，发现的主要目的之一就是要选择一个主节点，它将查看整个集群。主节点会检查所有其他节点，看它们是否有响应（其他节点也ping主节点）。主节点还将接受想加入群集的新节点。如果不知什么缘故，主节点跟集群断开连接了，其余节点将从中选择一个新的主节点。所有这些过程都基于我们提供的配置自动完成。

1. 配置主节点和数据节点

默认情况下，Elasticsearch允许节点同时成为主节点和数据节点。但在特定情况下，你可能希望有只保存数据的工作节点，以及只处理请求和管理集群的主节点。一种情形是当你需要处理大量的数据，这时数据节点应该尽可能地高性能。为了把节点设置成只保存数据，需要告诉Elasticsearch，不希望这些节点成为主节点。为此，在elasticsearch.yml配置文件中添加如下属性：

```
node.master: false
node.data: true
```

为设置节点不保存数据，而只做主节点，我们希望通知Elasticsearch不希望这些节点保存数据。为此，在elasticsearch.yml配置文件中添加如下属性：

```
node.master: true
node.data: false
```

请注意，如果不设置的话，`node.master`和`node.data`默认都为`true`。但我们倾向于在配置中包含它，这样更清晰。

2. 主节点选取的配置

想象一下你创建了一个包含10个节点的集群。一切工作正常，直到有一天你的网络出现故障，有三个节点从集群中断开连接，但它们仍然能互相看见对方。由于zen发现机制和主节点选取的过程，断开的三个节点中选出了一个新的主节点，你最终有了两个名字相同的集群，各自都有一个主节点。这样的情况称为脑裂（split-brain），你必须尽可能避免。当脑裂发生时，你有两个（或更多）不会互相连接的集群，直到网络（或其他任何）问题得到修复。

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

为了防止脑裂发生，Elasticsearch提供了discovery.zen.minimum_master_nodes属性。该属性定义的是为了形成一个集群，有主节点资格并互相连接的节点的最小数目。现在回到我们的集群，如果把discovery.zen.minimum_master_nodes属性设置为所有可用节点个数加1的50%（在我们的例子中即是6），将只会有一个集群。为什么呢？因为如果网络正常，我们将有10个节点，多于6个，这些节点将成为一个集群。3个节点断开连接后，第一个集群仍然在运行。然而，因为只有3个断开连接，小于6个，它们将无法选出一个新的主节点，只能等待重新连回原来的集群。

7.1.3 设置集群名

如果我们不在elasticsearch.yml文件设置cluster.name属性，Elasticsearch将使用默认值：elasticsearch。这不见得很好，因此建议你设置cluster.name属性为你想要的名字。如果你想在一个网络中运行多个集群，也必须设置cluster.name属性，否则，将导致不同集群的所有节点都连接在一起。

1. 配置多播

多播是zen发现的默认方法。除了常见的设置（很快就会讨论到）外，我们还可以控制以下四个属性。

- discovery.zen.ping.multicast.group：用于多播请求的群组地址，默认为224.2.2.4。
- discovery.zen.ping.multicast.port：用于多播通信的端口号，默认为54328。
- discovery.zen.ping.multicast.ttl：多播请求被认为有效的时间，默认为3秒。
- discovery.zen.ping.multicast.address：Elasticsearch应该绑定的地址。默认为null，意味着Elasticsearch将尝试绑定到操作系统可见的所有网络接口。

要禁用多播，应在elasticsearch.yml文件中添加discovery.zen.ping.multicast.enabled属性，并且把值设为false。

2. 配置单播

因为单播的工作方式，我们需要指定至少一个接收单播消息的主机。为此，在elasticsearch.yml文件中添加discovery.zen.ping.unicast.hosts属性。基本上，我们应该在discovery.zen.ping.unicast.hosts属性中指定所有形成集群的主机。指定所有主机不是必须的，只需要提供足够多的主机以保证最少有一个能工作。如果希望指定192.168.2.1、192.168.2.2和192.168.2.3主机，可以用下面的方法来设置上述属性：

```
discovery.zen.ping.unicast.hosts: 192.168.2.1:9300, 192.168.2.2:9300,
192.168.2.3:9300
```

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

你也可以定义一个Elasticsearch可以使用的端口范围，例如，从9300到9399端口，指定以下命令行：

```
discovery.zen.ping.unicast.hosts: 192.168.2.1:[9300-9399],
192.168.2.2:[9300-9399], 192.168.2.3:[9300-9399]
```

请注意，主机之间用逗号隔开，并指定了预计用于单播消息的端口。

使用单播时，总是设置discovery.zen.ping.multicast.enabled为false。

7.1.4　节点的ping设置

除了刚刚讨论的设置，还可以控制或改变默认的ping设置。ping是一个节点间发送的信号，用来检测它们是否还在运行以及可以响应。主节点会ping集群中的其他节点，其他节点也会ping主节点。可以设置下面的属性。

- discovery.zen.fd.ping_interval：该属性默认为1s（1秒钟），指定了节点互相ping的时间间隔。
- discovery.zen.fd.ping_timeout：该属性默认为30s（30秒钟），指定了节点发送ping信息后等待响应的时间，超过此时间则认为对方节点无响应。
- discovery.zen.fd.ping_retries：该属性默认为3，指定了重试次数，超过此次数则认为对方节点已停止工作。

如果你遇到网络问题，或者知道你的节点需要更多的时间来等待ping响应，可以把上面那些参数调整为对你的部署有利的值。

7.2　时光之门与恢复模块

除了我们的索引和索引里面的数据，Elasticsearch还需要保存类型映射和索引级别的设置等元数据。此信息需要被持久化到别的地方，这样就可以在群集恢复时读取。这就是为什么Elasticsearch引入了时光之门模块。你可以把它当做一个集群数据和元数据的安全的避风港。你每次启动群集，所有所需数据都从时光之门读取，当你更改你的集群，它使用时光之门模块保存。

7.2.1　时光之门

为了设置我们希望使用的时光之门类型，需要在elasticsearch.yml配置文件中添加gateway.type属性，并设置为local。目前，Elasticsearch推荐使用本地时光之门类型（gateway.type设为local），这也是默认值。过去有其他时光之门类型（比如fs、hdfs和s3），但已被弃用，

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

将在未来的版本中删除。因此跳过对它们的讨论。默认的本地时光之门类型在本地文件系统中存储索引和它们的元数据。跟其他时光之门类型相比，这种的写操作不是异步的。所以，每当写操作成功，都可以确保数据已经被写入了gateway（也就是说，它被索引或存储在事务日志中）。

7.2.2 恢复控制

除了选择时光之门类型以外，Elasticsearch允许配置何时启动最初的恢复过程。恢复是初始化所有分片和副本的过程，从事务日志中读取所有数据，并应用到分片上。基本上，这是启动Elasticsearch所需的一个过程。

假设有一个由10个Elasticsearch节点组成的集群。应该通知Elasticsearch我们的节点数目，设置gateway.expected_nodes属性为10。我们告知Elasticsearch有资格来保存数据且可以被选为主节点的期望节点数目。集群中节点的数目等于gateway.expected_nodes属性值时，Elasticsearch将立即开始恢复过程。

也可以在8个节点之后开始恢复，设置gateway.recover_after_nodes属性为8。可以将它设置为任何我们想要的值，但应该把它设置为一个值以确保集群状态快照的最新版本可用，一般在大多数节点可用时开始恢复。

然而，还有一件事：我们希望gateway在集群形成后的10分钟以后开始恢复，因此设置gateway.recover_after_time属性为10m。这个属性告诉gateway模块，在gateway.recover_after_nodes属性指定数目的节点形成集群之后，需要等待多长时间再开始恢复。如果我们知道网络很慢，想让节点之间的通信变得稳定时，可能需要这样做。

上述属性值都应该设置在elasticsearch.yml配置文件中，如果想设置上面的值，则最终在文件中得到下面的片段：

```
gateway.recover_after_nodes: 8
gateway.recover_after_time: 10m
gateway.expected_nodes: 10
```

额外的gateway恢复选项

除了提到的选项，Elasticsearch还允许做一些控制。额外的选项如下。

- gateway.recover_after_master_nodes：这个属性跟gateway_recover_after_nodes属性类似。它指定了多少个有资格成为主节点的节点在集群中出现时才开始启动恢复，而不是指定所有节点。
- gateway.recover_after_data_nodes：这个属性也跟gateway_recover_after_nodes属性类似。它指定了多少个数据节点在集群中出现时才开始启动恢复。

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

- `gateway.expected_master_nodes`：这个属性跟`gateway.expected_nodes`属性类似，它指定了希望多少个有资格成为主节点的节点出现，而不是所有的节点。
- `gateway.expected_data_nodes`：这个属性也跟`gateway.expected_nodes`属性类似，它指定了你期望出现在集群中的数据节点的个数。

7.3　为高查询和高索引吞吐量准备 Elasticsearch 集群

直到现在，我们谈的主要是Elasticsearch在处理查询和索引数据方面不同的功能。在这里，我们想简要谈谈为高查询和高索引吞吐量准备Elasticsearch集群。本节的开头先提到一些还没谈到的Elasticsearch功能，它们对优化集群很重要。我们知道，这是一个非常简要的介绍，但我们会试着只介绍那些我们认为重要的内容。之后，会就如何调整这些功能给出一般性建议，以及注意事项。希望通过阅读这一节，你能够在调优集群时知道要注意的事项。

7.3.1　过滤器缓存

过滤器缓存负责缓存查询中使用到的过滤器。你可以从缓存中飞快地获取信息。如果设置得当，它将有效地提高查询速度，尤其是那些包含已经执行过的过滤器的查询。

Elasticsearch包含两种类型的过滤器缓存：节点过滤器缓存（默认）和索引过滤器缓存。节点过滤器缓存被分配在节点上的所有索引共享，可以配置成使用特定大小的内存，或分配给Elasticsearch总内存的百分比。为了设置这个值，应该包含`indices.cache.filter.size`这个节点属性值，并设置成需要的大小或百分比。

第二种过滤器缓存基于索引级别。一般来说，你应该使用节点级别的过滤器缓存，因为很难预测每个索引的最终缓存大小，通常也不知道最终节点上会有多少索引。我们将省略对索引级别过滤器缓存的进一步解释，关于它的更多信息可以在官方文档找到，或者我们的书*Mastering ElasticSearch*。

7.3.2　字段数据缓存和断路器

字段数据缓存是Elasticsearch缓存的一部分，主要用于当查询对字段执行排序或切面时。Elasticsearch把用于该字段的数据加载到内存，以便基于每个文档快速访问这些值。构建这些字段数据缓存是昂贵的，所以最好有足够的内存，以便缓存中的数据一旦加载就留在缓存中。

你也可以把字段配置成使用doc值，而不是字段数据缓存。doc值在2.2节中讨论过。

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

允许用于字段数据缓存的内存大小可以用indices.fielddata.cache.size属性来控制。可以把它设置为绝对值（例如2 GB）或者分配给Elasticsearch实例的内存百分比（例如40%）。请注意，这些值是节点级别，而不是索引级别的。为其他条目丢弃部分缓存会导致查询性能变差，所以建议要有足够的物理内存。此外请记住，默认情况下，字段数据缓存的大小是无限的，所以如果我们不小心，会导致集群的内存爆炸。

我们还可以控制字段数据缓存的过期时间，同样，默认情况下字段数据缓存永远不过期。可以使用indices.fielddata.cache.expire属性来控制，将其设置为最大的不活动时间。例如，将它设置为10m将导致缓存不活动10分钟后过期。记住重建字段数据缓存是非常昂贵的，一般情况下，你不应该设置过期时间。

断路器

字段数据断路器（field data circuit breaker）允许估计一个字段加载到缓存所需的内存。利用它，可以通过抛出异常来防止一些字段加载到内存中。Elasticsearch有两个属性来控制断路器的行为。第一个是indices.fielddata.breaker.limit属性，默认值为80%，可以使用集群的更新设置API来动态地修改它。这意味着，当查询导致加载字段的值所需的内存超过了Elasticsearch进程中可用堆内存的80%时，将引发一个异常。第二个属性是indices.fielddata.breaker.overhead，默认为1.03，它定义了用来与原始估计相乘的一个常量。

7.3.3 存储模块

Elasticsearch中的存储模块负责控制如何写入索引数据。我们的索引可以完全存储在内存或者一个持久化磁盘中。纯内存的索引极快但不稳定，而基于磁盘的索引慢一些，但可容忍故障。

利用index.store.type属性，可以指定使用哪种存储类型，可用的选项包括下面这些。

- simplefs：这是基于磁盘的存储，使用随机文件来访问索引文件。它对并发访问的性能不够好，因此不建议在生产环境使用。
- niofs：这是第二个基于磁盘的索引存储，使用Java NIO类来访问索引文件。它在高并发环境提供了非常好的性能，但不建议在Windows平台使用，因为Java在这个平台的实现有bug。
- mmapfs：这是另一个基于磁盘的存储，它在内存中映射索引文件（对于mmap，请参阅http://en.wikipedia.org/wiki/Mmap）。这是64位系统下的默认存储，因为为索引文件提供了操作系统级别的缓存，因此它的读操作更快。你需要确保有足够数量的虚拟地址空间，但在64位系统下，这不是问题。
- memory：这将把索引存在内存中。请记住你需要足够的物理内存来存储所有的文档，否则Elasticsearch将失败。

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

7.3.4　索引缓冲和刷新率

当说到索引时，Elasticsearch允许你为索引设置最大的内存数。indices.memory.index_buffer_size属性可以控制在一个节点上，所有索引的分片共拥有的最大内存大小（或者最大堆内存的百分比）。例如，把该属性设置为20%，将告诉Elasticsearch提供最大堆大小20%的内存给索引缓冲。

此外，还有个indices.memory.min_shard_index_buffer_size属性，默认为4mb，允许为每个分片设置最小索引缓冲。

索引刷新率

有关索引的最后一件事是index.refresh_interval属性，它指定在索引上，默认为1s（1秒钟），指定了索引搜索器对象刷新的频率，基本上意味着数据视图刷新的频率。刷新率越低，文档对搜索操作可视的时间越短，也意味着Elasticsearch将需要利用更多资源来刷新索引视图，因此索引和搜索操作将会变慢。

对庞大的批量索引，例如，当对数据重建索引时，建议在索引阶段把index.refresh_interval属性设为-1。

7.3.5　线程池的配置

Elasticsearch使用多个池来控制线程的处理，以及控制用户请求占用多少内存消耗。

Java虚拟机允许应用程序使用多线程并行地运行程序的多个分支。有关Java线程的更多信息，请参阅http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html。

我们尤其感兴趣的是Elasticsearch公开的如下线程池类型。

- cache：这是无限制的线程池，为每个传入的请求创建一个线程。
- fixed：这是一个有着固定大小的线程池，大小由size属性指定，允许你指定一个队列（使用queue_size属性指定）用来保存请求，直到有一个空闲的线程来执行请求。如果Elasticsearch无法把请求放到队列中（队列满了），该请求将被拒绝。

有很多线程池（可以使用type属性指定要配置的线程类型），然而，对于性能来说，最重要的是下面几个。

- index：此线程池用于索引和删除操作。它的类型默认为fixed，size默认为可用处理器的数量，队列的size默认为300。

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

- search：此线程池用于搜索和计数请求。它的类型默认为fixed，size默认为可用处理器的数量乘以3，队列的size默认为1000。
- suggest：此线程池用于建议器请求。它的类型默认为fixed，size默认为可用处理器的数量，队列的size默认为1000。
- get：此线程池用于实时的GET请求。它的类型默认为fixed，size默认为可用处理器的数量，队列的size默认为1000。
- bulk：你可以猜到，此线程池用于批量操作。它的类型默认为fixed，size默认为可用处理器的数量，队列的size默认为50。
- percolate：此线程池用于预匹配器操作。它的类型默认为fixed，size默认为可用处理器的数量，队列的size默认为1000。

举个例子，把用于索引操作的线程池配置成fixed类型，大小为100，队列大小为500，我们将在elasticsearch.yml配置文件中设置如下属性：

```
threadpool.index.type: fixed
threadpool.index.size: 100
threadpool.index.queue_size: 500
```

记住，线程池的配置可以使用集群的更新API来更新，如下所示：

```
curl -XPUT 'localhost:9200/_cluster/settings' -d '{
  "transient" : {
    "threadpool.index.type" : "fixed",
    "threadpool.index.size" : 100,
    "threadpool.index.queue_size" : 500
  }
}'
```

7.3.6 结合起来，一些通用建议

现在，我们知道了Elasticsearch所公开的缓存和缓冲区，可以尝试结合这些知识来配置一个高索引和查询吞吐量的集群。接下来的两个小节将讨论在设置集群时，什么可以在默认配置中更改，什么是要注意的。

在讨论Elasticsearch特定配置相关的所有事情之前，应该记住，必须给予Elasticsearch足够的内存，而且是物理内存。一般来说，运行Elasticsearch的JVM进程不应该超过可用内存的50%或60%，这样做是因为要留一些可用内存给操作系统以及操作系统的I/O缓存。

然而，需要记住，50%到60%不一定总是对的。你可以想象一个有256 GB内存的节点，在节点上有个总共30 GB的索引，在这种情况下，即使分配多于60%的物理内存给Elasticsearch，也会给操作系统留下足够的内存。另外，把Xmx和Xms参数设置为相同的值以避免JVM堆的大小调整，也是个好主意。

当优化你的系统时，记得有一个在相同环境可以反复跑的性能测试。一旦你做出更改，你需要看到它是如何影响整体性能的。此外，Elasticsearch可扩展，正因为此，有时最好在单机上

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

做一个简单的性能测试，看看性能如何，可以从中得到什么。这样的一些观察是进一步调优的好起点。

在继续之前，注意我们不能给你高索引高查询吞吐量的秘方，因为每个部署都是不同的。因此，我们将只讨论你在调优时应该注意什么。如果你对这样的用例感兴趣，可以访问博客http://blog.sematext.com，有些作者会写一些关于性能测试的文章。

1. 选择正确的存储

当然，除了已经谈过的物理内存外，应该选择正确的存储实现。一般来说，如果运行的是64位操作系统，你应该选择`mmapfs`。如果没有运行64位操作系统，为UNIX系统选择`niofs`，为Windows系统选择`simplefs`。如果你可以容忍一个易失的存储，但希望它非常快，可以看看`memory`存储，它会给你最好的索引访问性能，但需要足够的内存来处理所有索引文件、索引和查询。

2. 索引刷新率

应该注意的第二件事是索引刷新率。我们知道刷新率指定文档多快可以对搜索操作可见。等式非常简单：刷新率越快，查询越慢，索引吞吐量越低。如果我们允许有一个较慢的刷新率，如10s或30s，设置它是不错的。这使得Elasticsearch承受的压力更少，因为内部对象重新打开的频率更低，因此，将有更多的资源用于索引和查询。

3. 优化线程池

强烈建议调整默认线程池，尤其是查询操作。在性能测试之后，你通常看到集群上的查询不堪重负，这时应该开始拒绝请求。我们认为在大多数情况下，最好是立刻拒绝该请求，而不是把它放到队列并强制应用程序等待很长时间请求处理。我们真的很想给你一个准确的数字，但这仍然在很大程度上取决于你的部署，给不了通用的建议。

4. 优化合并过程

合并过程同样在很大程度上取决于你的用例，以及若干因素，例如是否正在索引、你添加了多少数据以及做这些操作的频率。一般来说，记住查询多个段跟查询数量更少的段相比更慢。但是，想要段的数目更少，你需要付出更多的合并代价。

2.5节讨论过段合并。我们还提到了调节，它允许限制I/O操作。

通常来说，如果你想查询更快，应该以索引中更少的段为目标。如果想索引更快，应该有更多的段。如果你想兼具两者，就要找到两者之间的黄金点，让合并不会太频繁但又不会导致大量的段。使用并行合并调度器并调整默认调节值，使I/O子系统不会被合并淹没。

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

5. 字段数据缓存和断路器

默认情况下，Elasticsearch中的字段数据缓存是无限的。这很危险，尤其在很多字段上使用切面或排序时。如果字段基数很高，你可能会遇到更多的麻烦，麻烦的意思是说你可能会内存不足。

我们有两个不同因子可以调节，来确保不会遇到内存不足错误。首先，可以限制字段数据缓存的大小。其次是断路器，通过它可以很容易地配置成在加载过多数据时抛出一个异常。两者结合可以确保我们不会遇到内存问题。

然而，也应该记住，当数据字段缓存的大小不足以处理切面或排序请求时，Elasticsearch将从中移除数据。这将影响查询性能，因为加载字段数据信息是低效的。不过，我们认为宁愿让查询慢，也不能由于内存不足错误而导致集群不工作。

6. 索引的内存缓冲区

请记住，用于索引缓冲区的可用内存越多（`indices.memory.index_buffer_size`属性），Elasticsearch可以在内存中保存的文档也越多。但是，我们当然不想Elasticsearch占用100%的可用内存。默认情况下，该属性被设置为10%，但如果真的需要更高的索引比例，你可以提高这个百分比。我们见过一些关注数据索引的集群，把该属性设为30%，确实是有帮助的。

7. 优化事务日志

我们还没讨论到，但Elasticsearch有个内部模块称为translog（http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/index-modules-translog.html）。它是分片上的结构，为预写日志（http://en.wikipedia.org/wiki/Write-ahead_logging）服务。

基本上，它允许Elasticsearch为GET操作公开最新的更新，确保数据持久性，并优化对Lucene索引的写入。

默认情况下，Elasticsearch在事务日志中保存最多5000次操作，同时最大不超过200 mb。但如果想要更高的索引吞吐量，又可以承担数据对搜索操作不可见的时间更长，就可以提高这个默认值。通过`index.translog.flush_threshold_ops`和`index.translog.flush_threshold_size`属性（两者都是索引上的设置，可以用Elasticsearch API实时更新），可以设置保存在事务日志中的最大操作数和最大的大小。我们见过一些部署把这个属性值设成默认值的10倍。

要记住一件事，如果发生故障，对于有更大事务日志的分片，其初始化当然也更慢，因为Elasticsearch需要在分片可用之前处理事务日志中的所有信息。

8. 牢记于心

当然，前面提到的因素并非全部。你应该监视Elasticsearch集群并作出相应的反应。如果你

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

看到索引中的段开始增长，而你不想这样，请调整合并政策。当你看到合并使用过多的I/O资源，并影响了整体性能，请调整你的调节。只是要记住，调整不是一次性的；数据会增加，查询数目也会增加，你要不断适配它。

7.4　模板和动态模板

在2.2节中，我们学过映射、如何创建它们以及类型确定机制是如何工作的。现在将进入更高级的主题，如何为新的索引动态地创建映射以及如何在模板中应用一些逻辑。

7.4.1　模板

我们在前面看到过，索引配置，特别是映射，可以是很复杂的野兽。如果有可能定义一个或多个映射，用在每个新创建的索引上，而不需要每次创建索引时都发送它们，那就太好了。Elasticsearch创作者预见到这点，并实现了一个叫索引模板（index templates）的功能。每个模板定义了一个模式，用来比较新创建索引的名称。当两者匹配，在模板中定义的值复制到索引的结构定义中。当多个模板匹配新创建索引的名称时，所有模板都会被应用，后应用的模板中的值将覆盖先应用的模板中定义的值。这非常方便，因为可以在通用模板中定义一些常用设置，然后在专有模板中修改它们。此外，还有个`order`参数，可以强制所需模板的顺序。你可以把模板想象成一个动态映射，但它不是应用到文档中的类型，而是应用到索引。

1. 模板的一个例子

来看一个真实的模板例子。假设要创建许多索引，并且不希望在索引中存储源文档，以便让索引更小，而且也不需要任何副本。可以创建一个模板来满足这个需求，通过使用Elasticsearch的REST API，发送如下命令：

```
curl -XPUT http://localhost:9200/_template/main_template?pretty -d '{
  "template" : "*",
  "order" : 1,
  "settings" : {
    "index.number_of_replicas" : 0
  },
  "mappings" : {
    "_default_" : {
      "_source" : {
        "enabled" : false
      }
    }
  }
}'
```

从现在开始，所有创建的索引都将没有副本，也没有存储源文档。这是因为`template`参数值设成了`*`，意味着匹配所有索引名。注意例子中的`_default_`类型名字，这是一个特殊的类型

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

名，表明当前规则应适用于所有的文档类型。第二个有趣的事情是order参数。使用如下命令定义第二个模板：

```
curl -XPUT http://localhost:9200/_template/ha_template?pretty -d '{
  "template" : "ha_*",
  "order" : 10,
  "settings" : {
    "index.number_of_replicas" : 5
  }
}'
```

执行上述命令后，除了名字以ha_开头，所有其他新索引将有相同的行为。两个模板都被应用在这些索引中。首先应用order值更小的模板，然后下一个模板覆盖副本的设置。所以，名字以ha_开头的索引将有5个副本，并禁用源文档的存储。

2. 在文件中存储模板

模板也可以存储在文件中。默认情况下，文件应该放在config/templates目录中。例如，ha_template模板应该放在config/templates/ha_template.json文件中，内容如下所示：

```
{
  "ha_template" : {
    "template" : "ha_*",
    "order" : 10,
    "settings" : {
      "index.number_of_replicas" : 5
    }
  }
}
```

注意这个JSON的结构有点不同，它使用模板名字作为主对象的键。另外很重要的是，模板必须放在Elasticsearch的每个实例中。此外，文件中定义的模板不可用在REST API调用中。

7.4.2 动态模板

有时，我们想依赖一个字段名称和类型来定义一个类型。这是动态模板发挥作用的地方。动态模板跟通常的映射相似，但每个模板都定义了它的模式，并将其应用于文档的字段名称。如果一个字段名称与模式匹配，则使用该模板。看看下面的示例：

```
{
  "mappings" : {
    "article" : {
      "dynamic_templates" : [
      {
        "template_test": {
          "match" : "*",
          "mapping" : {
            "index" : "analyzed",
```

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

```
            "fields" : {
              "str": {"type": "{dynamic_type}",
                      "index": "not_analyzed" }
            }
          }
        }
      }]
    }
  }
}
```

在前面的例子中，我们为article类型定义了一个映射。在这个映射中，我们只有一个动态模板，名为template_test。由于match属性设置成了星号，这个模板将应用到输入文档的每个字段上。每个字段都将被视为多字段，由原始字段名字（比如，title）和第二个以str作为后缀的字段（比如，title.str）。第一个字段的类型由Elasticsearch决定（{dynamic_type}类型），第二个字段将是一个字符串（因为string类型）。

1. 匹配模式

我们有以下两种方式定义匹配模式。

- match：如果字段名与模式相匹配的话，则使用该模板（这是我们例子中使用的类型）。
- unmatch：如果字段名与模式不匹配，则使用该模板。

默认情况下，该模式非常简单，并且使用glob模式，可以使用match_pattern=regexp来修改。添加此属性后，可以使用正则表达式提供的所有魔法来匹配或不匹配模式。

有些变种，比如path_match和path_unmatch，可以用来匹配嵌套文档的名字。

2. 字段定义

当写入一个目标字段定义时，可以使用下面的变量。

- {name}：输入文档中找到的原始字段的名字。
- {dynamic_type}：原始文档确定的类型。

请注意，Elasticsearch根据定义的顺序检查模板，并应用第一个匹配的模板。这意味着最通用的模板（比如，使用"match":"*"的那些）应该定义在最后面。

7.5　小结

本章介绍了Elasticsearch的一些功能，如节点发现、模块负责什么，以及如何调优。也学习了恢复和时光之门模块如何设置以配合集群，它们提供了什么配置选项。我们还探讨了一些

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权

Elasticsearch的内部功能，使用这些来为高索引和高查询调优集群。最后，我们使用了模板和动态映射，以便更好地管理动态索引。

下一章关注Elasticsearch的管理能力。我们将学习如何备份集群数据，使用可用的API调用监视我们的集群；讨论使用Elasticsearch API如何控制分片的分配，如何在集群中移动分片；了解什么是索引预热器，以及它们的用处；使用别名。最后，还将学习如何安装和管理Elasticsearch插件，以及用更新设置API能做些什么。

图灵社区会员 打顺顺(lvshun@live.cn) 专享 尊重版权