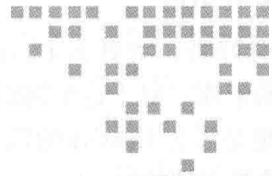


第 1 章 Chapter 1

ElasticSearch 简介

我们希望读者通过阅读本书能获取和拓展关于 ElasticSearch 的基本知识，并假设读者已经知道如何使用 ElasticSearch 进行单次或批量索引创建，如何发送请求检索感兴趣的文档，如何使用过滤器缩减检索返回文档的数量，以及使用切面 / 聚合（faceting/aggregation）机制来计算数据的一些统计量。不过，在接触 ElasticSearch 提供的各种令人激动的功能之前，仍然希望读者能对 Apache Lucene 有一个快速了解，因为 ElasticSearch 使用开源全文检索库 Lucene 进行索引和搜索，此外，我们还希望读者能了解 ElasticSearch 的一些基础概念，以及为了加快学习进程，牢记这些基础知识，当然，这并不难掌握。同时，我们也需要确保读者能按 ElasticSearch 所需要的那样正确理解 Lucene。本章主要涵盖以下内容：

- Apache Lucene 是什么。
- Lucene 的整体架构。
- 文本分析过程是如何实现的。
- Apache Lucene 的查询语言及其使用方法。
- ElasticSearch 的基本概念。
- ElasticSearch 内部是如何通信的。

1.1 Apache Lucene 简介

为了全面理解 ElasticSearch 的工作原理，尤其是索引和查询处理环节，对 Apache Lucene 的理解显得至关重要。揭开 ElasticSearch 神秘的面纱，你会发现它在内部不仅使用 Apache Lucene 创建索引，同时也使用 Apache Lucene 进行搜索。因此，在接下来的内容中，我们将展示 Apache Lucene 的基本概念，特别是针对那些从未使用过 Lucene 的读者们。

1.1.1 熟悉 Lucene

读者也许会好奇，为什么 ElasticSearch 的创始人决定使用 Apache Lucene 而不是开发自己的全文检索库。对于这个问题，笔者并不是很确定，毕竟我们不是这个项目的创始人，但我们猜想是因为 Lucene 的以下特点而得到了创始人的青睐：成熟、高性能、可扩展、轻量级以及强大的功能。Lucene 内核可以创建为独立的 Java 库文件并且不依赖第三方代码，用户可以使用它提供的各种所见即所得的全文检索功能进行索引和搜索操作。当然，Lucene 还有很多扩展，它们提供了各种各样的功能，如多语言处理、拼写检查、高亮显示等。如果不需要这些额外的特性，可以下载单个的 Lucene 内核库文件，直接在应用程序中使用。

1.1.2 Lucene 的总体架构

尽管我们可以直接探讨 Apache Lucene 架构的细节，但有些概念还是需要提前了解，以便更好地理解 Lucene 的架构，它们包括：

- **文档**（document）：索引与搜索的主要数据载体，它包含一个或多个字段，存放将要写入索引或将从索引搜索出来的数据。
- **字段**（field）：文档的一个片段，它包括两个部分：字段的名称和内容。
- **词项**（term）：搜索时的一个单位，代表文本中的某个词。
- **词条**（token）：词项在字段中的一次出现，包括词项的文本、开始和结束的位移以及类型。

Apache Lucene 将写入索引的所有信息组织成一种名为倒排索引（inverted index）的结构。该结构是一种将词项映射到文档的数据结构，其工作方式与传统的关系数据库不同，你大可以认为倒排索引是面向词项而不是面向文档的。接下来我们看看简单的倒排索引是什么样的。例如，我们有一些只包含 title 字段的文档，如下所示：

- ElasticSearch Server（文档 1）
- MasteringElasticSearch（文档 2）
- Apache solr 4 Cookbook（文档 3）

而索引后的结构示意图如下所示。

词项	计数	文档
4	1	<3>
Apache	1	<3>
Cookbook	1	<3>
ElasticSearch	2	<1><2>
Mastering	1	<1>
Server	1	<1>
Solr	1	<3>

正如你所见，每个词项指向该词项所出现过的文档数。这种索引组织方式支持快速有效的搜索操作，例如基于词项的查询。除了词项本身以外，每个词项还有一个与之关联的计数（即文档频率），用来告诉 Lucene 这个词项在多少个文档中出现过。

当然，实际中 Lucene 创建的索引更为复杂，也更先进，因为索引中还存储了很多其他信息，如**词向量**（为单个字段创建的小索引，存储该字段中所有的词条）、各字段的原始信息、文档删除标记等。然而，你只需知道 Lucene 索引中数据是如何组织的即可，而不用知道到底存储了哪些东西。

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，段经创建就不会再被修改。例如，文档被删除以后，删除信息被单独保存在一个文件中，而段本身并没有修改。

多个段会在一个叫作**段合并**（segments merge）的阶段被合并在一起，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。段合并非常耗 I/O，且合并期间有些不再使用的信息也将被清理掉，例如，被删除的文档。对于容纳相同数据的索引，段的数量较少时，搜索速度更快。尽管如此，还是需要强调一下：因为段合并非常耗 I/O，请不要强制进行段合并，你只需要仔细配置段合并策略，剩余的事情 Lucene 会自行搞定。

如果你想知道段由哪些文件组成以及每个文件都存储了什么信息，请参考 Apache Lucene 的官方文档：http://lucene.apache.org/core/4_5_0/core/org/apache/lucene/codecs/lucene45/package-summary.html。

1.1.3　分析你的数据

读者也许会好奇，文档中的数据是如何转化为倒排索引的，而查询串又是如何转换为可用于搜索的词项的？这个转换过程称为分析（analysis）。

文本分析由**分析器**来执行，而分析器由分词器（tokenizer）、过滤器（filter）和字符映射器组成（character mapper）。

Lucene 的**分词器**用来将文本切割成词条，其中携带各种额外信息的词项，这些信息包括：词项在原始文本中的位置、词项的长度。分词器的工作成果称为词条流，因为这些词条被一个接一个地推送给过滤器处理。

除了分词器，过滤器也是 Lucene 分析器的组成部分，过滤器数额可选，可以为 0 个、1 个或多个，用于处理词条流中的词条。例如，它可以移除、修改词条流中的词条，甚至可以创造新的词条。Lucene 提供了很多现成的过滤器，你也可以根据需要实现新的过滤器。以下是一些过滤器的例子：

- **小写过滤器**：将所有词条转化为小写。
- **ASCII 过滤器**：移除词条中所有非 ASCII 字符。

- **同义词过滤器**：根据同义词规则，将一个词条转化为另一个词条。
- **多语言词干还原过滤器**：将词条的文本部分归约到它们的词根形式，即词干还原。

当分析器中有多个过滤器时，会逐个处理，理论上可以有无限多个过滤器。

最后我们介绍字符映射器，它用于调用分词器之前的文本预处理操作，如 HTML 文本的去标签处理。

索引与查询

也许读者会好奇，Lucene 以及所有基于 Lucene 的软件，是如何控制索引和查询操作的。在索引期，Lucene 会使用你选择的分析器来处理文档中的内容，并可以对不同的字段使用不同的分析器，例如，文档的 title 字段与 description 字段就可以使用不同的分析器。

在检索时，如果使用了某个查询分析器（query parser），那么查询串就会被分析。当然，你也可以选择不进行查询分析。有一点需要牢记，ElasticSearch 中有些查询会被分析，而有些则不会。例如，前缀查询（prefix query）不会被分析，而匹配查询（match query）会被分析。

你还应该记住，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。例如，如果在索引期使用了词干还原与小写转换，那么在查询期也应该对查询串做相同的处理，否则查询可能不会返回任何结果。

1.1.4 Lucene 查询语言

ElasticSearch 提供的一些查询类型（query type）支持 Apache Lucene 的查询解析语法，因此，我们应该深入了解 Lucene 的查询语言并加以描述。

理解基本概念

在 Lucene 中，一个查询通常被分割为词项与操作符。Lucene 中的词项可以是单个词，也可以是一个短语（用双引号括起来的一组词）。如果设置了查询分析过程，那么预先选定的分析器将会对查询中的所有词项进行处理。

查询中也可以包含布尔操作符，用于连接多个词项，使之构成从句（clause）。有以下这些布尔操作符：

- AND：它的含义是，文档匹配当前从句当且仅当 AND 操作符左右两边的词项都在文档中出现。例如，执行 apache AND lucene 这样的查询，只有同时包含 apache 和 lucene 这两个词项的文档才会返回给用户。
- OR：它的含义是，包含当前从句中任意词项的文档都会被视为与该从句匹配。例如，执行 apache OR lucene 这样的查询，任意包含词项 apache 或词项 lucene 的文档都会返回给用户。
- NOT：它的含义是，与当前从句匹配的文档必须不包含 NOT 操作符后面的词项。

例如，执行 lucene NOT elasticsearch 这样的查询，只有包含词项 lucene 且不包含词项 elasticsearch 的文档才会返回给用户。

此外，我们还可以使用以下操作符：

- +：它的含义是，只有包含 + 操作符后面词项的文档才会被认为是与从句匹配。例如，查找那些必须包含 lucene，但是 apache 可出现可不出现的文档，可执行查询：+lucene apache。
- -：它的含义是，与从句匹配的文档不能出现 - 操作符后的词项。例如，查找那些包含 lucene 但不包含 elasticsearch 的文档，可以执行查询：+lucene-elasticsearch。

如果查询中没有出现前面提到过的任意操作符，那么默认使用 OR 操作符。

另外，你还可以使用圆括号对从句进行分组，以构造更复杂的从句，例如：

```
elasticsearch AND (mastering OR book)
```

在字段中查询

就像 ElasticSearch 的处理方式那样，Lucene 中所有数据都存储在字段（field）中，而字段又是文档的组成单位。为了实现针对某个字段的查询，用户需要提供字段名称，再加上冒号以及将要在该字段中执行查询的从句。例如要查询所有在 title 字段中包含词项 elasticsearch 的文档，可执行以下查询：

```
title:elasticsearch
```

也可以在一个字段中同时使用多个从句，例如，要查找所有在 title 字段中同时包含词项 elasticsearch 和短语 mastering book 的文档，可执行如下查询：

```
title:(+elasticsearch +"mastering book")
```

当然，该查询也可以写成下面这种形式：

```
+title:elasticsearch +title:"mastering book"
```

词项修饰符

除了使用简单词项和从句的常规字段查询以外，Lucene 还允许用户使用修饰符（modifier）修改传入查询对象的词项。毫无疑问，最常见的修饰符就是通配符（wildcard）。Lucene 支持两种通配符：? 和 *。前者匹配任意一个字符，而后者匹配多个字符。

请记住，出于对性能的考虑，通配符不能作为词项的第一个字符出现。

除通配符之外，Lucene 还支持模糊（fuzzy and proximity）查询，通过使用 ~ 字符以及一个紧随其后的整数值。当使用该修饰符修饰一个词项时，意味着我们想搜索那些包含该词项近似词项的文档（所以这种查询称为模糊查询）。~ 字符后的整数值确定了近似词项与原始词项的最大编辑距离。例如，当我们执行查询：writer~2，意味着包含词项 writer 和 writers 的文档都能与查询匹配。

当修饰符~用于短语时，其后的整数值用于告诉 Lucene 词项之间可以接受的最大距离。例如，执行如下查询：

```
title:"mastering elasticsearch"
```

在 title 字段中包含 mastering elasticsearch 的文档被视为与查询匹配，而包含 mastering book elasticsearch 的文档则不匹配。如果执行这个查询：title:"mastering elasticsearch"~2，则这两个文档都被认为与查询匹配。

此外，还可以使用^字符并赋以一个浮点数对词项加权（boosting），以提高该词项的重要程度。如果都被加权，则权重值较大的词项更重要。默认情况下词项权重为 1。可以参考 2.1 节进一步了解什么是权重值，以及它在文档评分中的作用。

我们也可以使用方括号和花括号来构建范围查询。例如，要在一个数值类型的字段上执行一个范围查询，执行如下查询即可：

```
price:[10.00 TO 15.00]
```

上述查询所返回文档的 price 字段的值大于等于 10.00 并小于等于 15.00。

当然，我们也可以在字符串类型的字段上执行范围查询，例如：

```
name:[Adam TO Adria]
```

上面查询所返回文档的 name 字段包含按字典顺序介于 Adam 和 Adria 之间（包括 Adam 和 Adria）的词项。

如果想执行范围查询同时又想排除边界值，则可使用花括号作为修饰符。例如，查找 price 字段值大于等于 10.00 但小于 15.00 的文档，可使用如下查询：

```
price:[10.00 TO 15.00}
```

特殊字符处理

很多应用场景中，需要搜索某个特殊字符（这些特殊字符包括+、-、&&、||、!、(，)、{}、[]、^、"、~、*、?、:、\、/），这时先使用反斜杠对这些特殊字符进行转义。例如，搜索 abc"efg 这个词项，需要按如下方式处理：

```
abc\"efg
```

1.2 ElasticSearch 简介

虽然读者可能已经对 ElasticSearch 有所了解，至少已经了解了它的一些核心概念和基本用法。然而，为了全面理解该搜索引擎是如何工作的，我们最好简略地讨论一下它。

ElasticSearch 是一个可用于构建搜索应用的成品软件[⊖]。它最早由 Shay Banon 创建并

⊖ 区别于 Lucene 这种中间件。——译者注

于 2010 年 2 月发布。之后的几年 ElasticSearch 迅速流行开来，成为商业解决方案之外且开源的一个重要选择，也是下载量最多的开源软件之一，每月下载量超过 20 万次。

1.2.1　ElasticSearch 的基本概念

现在，让我们了解一下 ElasticSearch 的基本概念及其特征。

索引

ElasticSearch 将它的数据存储在一个或多个索引（index）中。用 SQL 领域的术语来类比，索引就像数据库，可以向索引写入文档或者从索引中读取文档，并通过在 ElasticSearch 内部使用 Lucene 将数据写入索引或从索引中检索数据。需要注意的是，ElasticSearch 中的索引可能由一个或多个 Lucene 索引构成，具体细节由 ElasticSearch 的索引分片（shard）、复制（replica）机制及其配置决定。

文档

文档（document）是 ElasticSearch 世界中的主要实体（对 Lucene 来说也是如此）。对所有使用 ElasticSearch 的案例来说，它们最终都可以归结为对文档的搜索。文档由字段构成，每个字段有它的字段名以及一个或多个字段值（在这种情况下，该字段被称为是多值的，即文档中有多个同名字段）。文档之间可能有各自不同的字段集合，且文档并没有固定的模式或强制的结构。另外，这些规则也适用于 Lucene 文档。事实上，ElasticSearch 的文档最后都存储为 Lucene 文档了。从客户端的角度来看，文档是一个 JSON 对象（想了解更多关于 JSON 格式细节，请参考 http://en.wikipedia.org/wiki/JSON）。

映射

正如你在 1.1 节所了解的那样，所有文档在写入索引前都需要先进行分析。用户可以设置一些参数，来决定如何将输入文本分割为词条，哪些词条应该被过滤掉，或哪些附加处理是有必要被调用的（如移除 HTML 标签）。此外，ElasticSearch 也提供了各种特性，如排序时所需的字段内容信息。这就是映射（mapping）扮演的角色，存储所有这种元信息。虽然 ElasticSearch 能根据字段值自动检测字段的类型，但有时候（事实上，几乎是所有时候）用户还是想自行配置映射，以避免出现一些令人不愉快的意外。

类型

ElasticSearch 中每个文档都有与之对应的类型（type）定义。这允许用户在一个索引中存储多种文档类型，并为不同文档类型提供不同的映射。

节点

单个的 ElasticSearch 服务实例称为节点（node）。很多时候部署一个 ElasticSearch 节点就足以应付大多数简单的应用，但是考虑到容错性或在数据膨胀到单机无法应付这些状况时，你也许会更倾向于使用多节点的 ElasticSearch 集群。

集群

当数据量或查询压力超过单机负载时，需要多个节点来协同处理，所有这些节点组成的系统称为集群（cluster）。集群同时也是无间断提供服务的一种解决方案，即便在某些节点因为宕机或执行管理任务（如升级）不可用时。ElasticSearch 几乎无缝集成了集群功能。在我们看来，这是它胜过竞争对手的最主要的优点之一。而且，在 ElasticSearch 中配置一个集群是再容易不过的事了。

分片

正如我们之前提到的那样，集群允许系统存储的数据总量超过单机容量。为了满足这个需求，ElasticSearch 将数据散布到多个物理 Lucene 索引上。这些 Lucene 索引称为分片（shard），而散布这些分片的过程叫作分片处理（sharding）。ElasticSearch 会自动完成分片处理，并且让这些分片呈现出一个大索引的样子。请记住，除了 ElasticSearch 本身自动进行分片处理外，用户为具体的应用进行参数调优也是至关重要的，因为分片的数量在索引创建时就已经配置好，而且之后无法改变，至少对目前的版本是这样的。

副本

分片处理允许用户向 ElasticSearch 集群推送超过单机容量的数据。副本（replica）则解决了访问压力过大时单机无法处理所有请求的问题。思路很简单，即为每个分片创建冗余的副本，处理查询时可以把这些副本用作最初的主分片（primary shard）。请记住，我们并未付出额外的代价。即使某个分片所在的节点宕机，ElasticSearch 也可以使用其副本，从而不会造成数据丢失，而且支持在任意时间点添加或移除副本，所以一旦有需要可随时调整副本的数量。

网关

在 ElasticSearch 的工作过程中，关于集群状态，索引设置的各种信息都会被收集起来，并在网关（gateway）中被持久化。

1.2.2 ElasticSearch 架构背后的关键概念

ElasticSearch 架构遵循了一些设计理念。通常开发团队希望这个搜索引擎产品易于使用和扩展，并能在 ElasticSearch 的各个地方体现出来。从架构的角度出发，ElasticSearch 具有下面这些主要特征：

- 合理的默认配置，使得用户在简单安装以后能直接使用 ElasticSearch 而不需要任何额外的调试，这包括内置的发现（如字段类型检测）和自动配置功能。
- 默认的分布式工作模式。每个节点总是假定自己是某个集群的一部分或将是某个集群的一部分，一旦工作启动节点便会加入某个集群。
- 对等架构（P2P）可以避免**单点故障**（SPOF）。节点会自动连接到集群中的其他节点，进行相互的数据交换和监控操作。这其中就包括索引分片的自动复制。

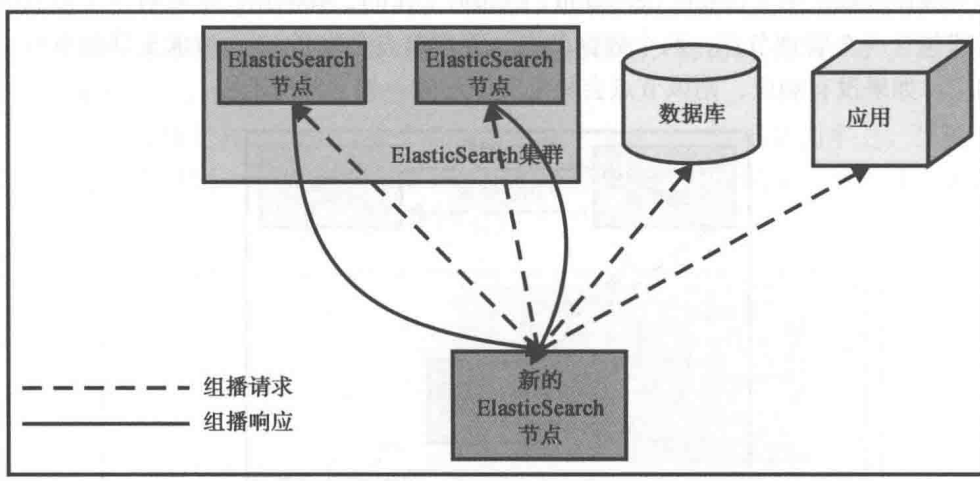
- 易于向集群扩充新节点，不论是从数据容量的角度还是数量角度。
- ElasticSearch 没有对索引中的数据结构强加任何限制，从而允许用户调整现有的数据模型。正如之前描述的那样，ElasticSearch 支持在一个索引中存在多种数据类型，并允许用户调整业务模型，包括处理文档之间的关联（尽管这种功能非常有限）。
- **准实时**（Near Real Time，NRT）搜索和版本同步（versioning）。考虑到 ElasticSearch 的分布式特性，查询延迟和节点之间临时的数据不同步是难以避免的。ElasticSearch 尝试消除这些问题并且提供额外的机制用于版本同步。

1.2.3　ElasticSearch 的工作流程

现在，让我们简单地讨论一下 ElasticSearch 是如何工作的。

启动过程

当 ElasticSearch 节点启动时，它使用广播技术（也可配置为单播）来发现同一个集群中的其他节点（这里的关键是配置文件中的集群名称）并与它们连接。读者可以通过下图的描述来了解相关的处理过程：



集群中会有一个节点被选为管理节点（master node）。该节点负责集群的状态管理以及在集群拓扑变化时做出反应，分发索引分片至集群的相应节点上。

请记住，从用户的角度来看，ElasticSearch 中的管理节点并不比其他节点重要，这与其他某些分布式系统不同（如数据库）。实际上，你不需要知道哪个节点是管理节点，所有操作可以发送至任意节点，ElasticSearch 内部会自行处理这些不可思议的事情。如果有需要，任意节点可以并行发送子查询给其他节点，并合并搜索结果，然后返回给用户。所有这些操作并不需要经过管理节点处理（请记住，ElasticSearch 是基于对等架构的）。

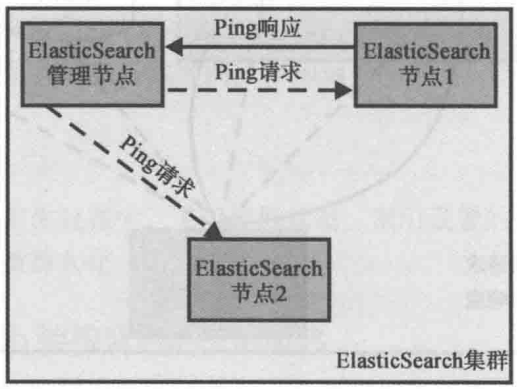
管理节点读取集群的状态信息，并在必要时进行恢复处理。在该阶段，管理节点会检查所有索引分片并决定哪些分片将用于主分片。然后，整个集群进入黄色状态。

这意味着集群可以执行查询，但是系统的吞吐量以及各种可能的状况是未知的（这种状况可以简单理解为所有的主分片已经分配出去了，而副本没有），因而接下来就是要寻找到冗余的分片并用作副本。如果某个主分片的副本数过少，管理节点将决定基于某个主分片创建分片和副本。如果一切顺利，集群将进入绿色状态（这意味着所有主分片和副本均已分配好）。

故障检测

集群正常工作时，管理节点会监控所有可用节点，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，然后开始启动错误处理过程。这意味着要在集群 - 分片之间重新做平衡，因为之前已断开节点上的那些分片不可用了，剩下的节点要肩负起相应的责任。换句话说，对每个丢失的主分片，一个新的主分片将会从原来的主分片的副本中脱颖而出。新分片和副本的放置策略是可配置的，用户可以根据具体需求进行配置。更多信息请参见第 4 章（索引分布架构）的内容。

为了描述故障检测（failure detection）是如何工作的，我们用一个只有三个节点的集群为例，即包含一个管理节点，两个数据节点。管理节点会发送 ping 请求至其他节点，然后等待响应。如下图所示：



与 ElasticSearch 通信

前面已经讨论过 ElasticSearch 是如何构建的了，然而，对普通用户来说，最重要的还是如何向 ElasticSearch 推送数据并构建查询。为了提供这些功能，ElasticSearch 对外公开了一个设计精巧的 API。这个 API 是基于 REST（REST 细节请参考：http://en.wikipedia.org/wiki/Representational_state_transfer）的，并在实践中能轻松整合到任何支持 HTTP 协议的系统中去。

ElasticSearch 假设数据由 URL 携带或者以 JSON（JSON 细节请参考 http://en.wikipedia.org/wiki/JSON。文档的形式由 HTTP 消息体携带。使用 Java 或基于 JVM 语言的用户，

应该了解一下 Java API，它除了 REST API 提供的所有功能以外还有内置的集群发现功能。

值得一提的是，ElasticSearch 在内部也使用 Java API 进行节点间通信。读者可以在第 8 章中了解更多 Java API 的细节，而这里只是简略地了解 Java API 提供了哪些功能。本书假设读者已经使用过这些功能了，只是在此做一点小小的提示。如果用户还没有使用过，强烈建议阅读相关材料，其中《ElasticSearch Server》一书覆盖了所有这些内容。

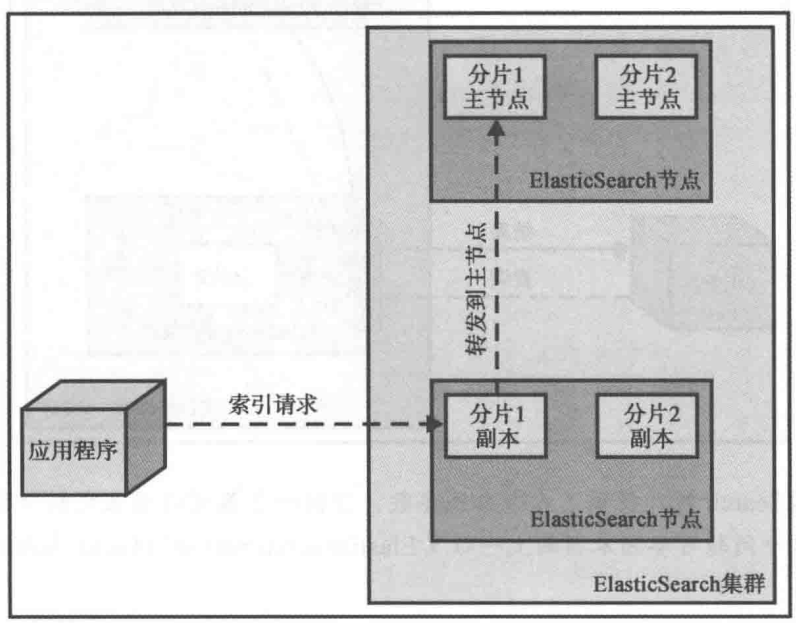
索引数据

ElasticSearch 提供了四种方式来创建索引。最简单的方式是使用索引 API，它允许用户发送一个文档至特定的索引。例如，使用 curl 工具（详见 http://curl.haxx.se/），并用如下命令创建一个文档：

```
curl -XPUT http://localhost:9200/blog/article/1 -d '{"title": "New
version of Elastic Search released!", "content": "...", "tags":
["announce", "elasticsearch", "release"] }'
```

第二种或第三种方式允许用户通过 bulk API 或 UDP bulk API 来一次性发送多个文档至集群。两者的区别在于网络连接方式，前者使用 HTTP 协议，后者使用 UDP 协议，且后者速度快，但是不可靠。第四种方式使用插件发送数据，称为河流（river），河流运行在 ElasticSearch 节点上，能够从外部系统获取数据。

有一件事情需要记住，建索引操作只会发生在主分片上，而不是副本上。当把一个索引请求发送至某节点时，如果该节点没有对应的主分片或者只有副本，那么这个请求会被转发到拥有正确的主分片的节点（如下图所示）。

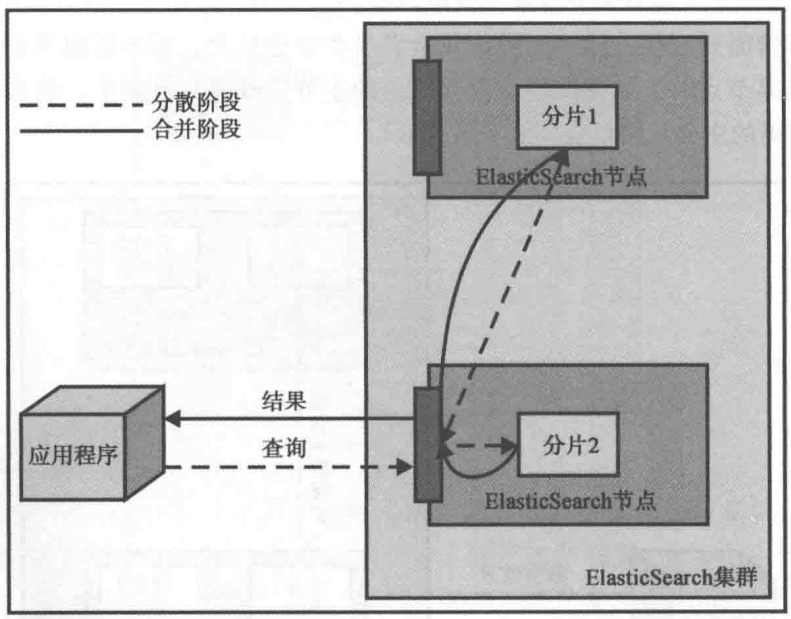


查询数据

查询 API 占据了 ElasticSearch API 的大部分内容。使用查询 DSL（基于 JSON 的可用于构建复杂查询的语言），我们可以做下面这些事情：

- 使用各种查询类型，包括：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、空间查询等。
- 组合简单查询构建复杂查询。
- 文档过滤，在不影响评分的前提下抛弃那些不满足特定查询条件的文档。
- 查找与特定文档相似的文档。
- 查找特定短语的查询建议和拼写检查。
- 使用切面构建动态导航和计算各种统计量。
- 使用预搜索（prospective search）并查找与指定文档匹配的 query 集合。

对于查询操作，读者应该要重点了解：查询并不是一个简单的、单步骤的操作。一般来说，查询分为两个阶段：分散阶段（scatter phase）和合并阶段（gather phase）。分散阶段将 query 分发到包含相关文档的多个分片中去执行查询，合并阶段则从众多分片中收集返回结果，然后对它们进行合并、排序、后续处理，然后返回给客户端。该机制可以由下图描述。



ElasticSearch 对外提供了 6 个系统参数，任何一个都可以用来定制分散 / 合并机制。关于这个问题可参阅本书的上一版《ElasticSearch Server》(Packt 出版社)。

索引配置

前面已经讨论过 ElasticSearch 的自动索引配置以及发现识别文档字段类型和结构的功能。当然，ElasticSearch 也提供了一些功能使得用户能手动配置，例如用户想通过映射来配置自定义的文档结构，或者想设置索引的分片和副本数，抑或定制文本分析过程，种种这些需求都可以通过手动配置解决。

系统管理和监控

ElasticSearch 中系统管理和监控相关的 API 允许用户改变集群的设置，如调节集群发现机制和索引放置策略等。此外，你还可得到关于集群状态信息或每个节点、每个索引的统计信息。集群监控 API 非常全面，我们将在第 5 章学习相关 API 的使用范例。

1.3　小结

在本章中，我们了解了 Apache Lucene 的一般架构，例如它的工作原理，文本分析过程是如何完成的，如何使用 Apache Lucene 查询语言。此外，我们还讨论了 ElasticSearch 的一些基本概念，例如它的基本架构和内部通讯机制。

在下一章，我们将学习 Apache Lucene 的默认评分公式，什么是查询重写过程（query rewrite process）以及它是如何工作的。除此之外，还将讨论 ElasticSearch 的一些功能，例如查询的二次评分（query rescore）、准实时批量获取（multi near real-time get）、批量搜索操作（bulk search operations）。接着将学习到如何使用 update API 来部分地改变文档，如何对数据进行排序，如何使用过滤功能（filterring）来改进查询的性能。最后，我们将了解如何在切面机制中使用过滤器（filters）和作用域（scope）。