第 1 章 Elasticsearch集群入门

欢迎来到Elasticsearch的奇妙世界，它是优秀的全文检索和分析引擎。不管你对Elasticsearch和全文检索有没有经验，都不要紧。我们希望你可以通过这本书，学习并扩展Elasticsearch的知识。由于这本书也是为初学者准备的，我们决定先简单介绍一般性的全文检索概念，接着再简要概述Elasticsearch。

我们要做的第一件事就是安装Elasticsearch。与许多应用相同，你从安装和配置着手，并经常忘记这些步骤的重要性。我们会尽量引导你完成这些步骤，从而使你更容易记住要点。此外，我们将告诉你如何用最简单的方法来索引和检索数据，而不用陷入太多细节。读完本章，你将学到以下内容：

- 全文检索；
- 了解Apache Lucene；
- 文本分析；
- 学习Elasticsearch的基本概念；
- 安装和配置Elasticsearch；
- 使用Elasticsearch REST API来操纵数据；
- 使用基本的URI请求来搜索。

1.1 全文检索

在全文检索只为一小部分工程师所知的时代，我们大多数人使用SQL数据库来执行搜索操作。当然它至少在一定程度上没什么问题。然而，当你越钻越深，就会看到这种方法的局限，如缺乏扩展性、不够灵活、缺乏语言分析（当然SQL数据库的全文检索对此有所作为）等。于是Apache Lucene（http://lucene.apache.org）出现了，它的目标是提供一个全文检索的功能库。它非常快速，可扩展，并提供不同语言的分析能力。

1.1.1 Lucene词汇表和架构

深入介绍分析处理的细节之前，我们先介绍一下Apache Lucene的词汇表和整体架构，下面

是这个库的基本概念。

- 文档（document）：索引和搜索时使用的主要数据载体，包含一个或多个存有数据的字段。
- 字段（field）：文档的一部分，包含名称和值两部分。
- 词（term）：一个搜索单元，表示文本中的一个词。
- 标记（token）：表示在字段文本中出现的词，由这个词的文本、开始和结束偏移量以及类型组成。

Apache Lucene将所有信息写到一个称为倒排索引（inverted index）的结构中。不同于关系型数据库中表的处理方式，倒排索引建立索引中词和文档之间的映射。你可以把倒排索引看成这样一种数据结构，其中的数据是面向词而不是面向文档的。来看一个简单的例子。我们有一些文档，只有它们的标题字段需要被索引，它们看起来如下所示：

- Elasticsearch Server 1.0 (document 1)；
- Mastering Elasticsearch (document 2)；
- Apache Solr 4 Cookbook (document 3)。

那么，简化版的索引可以看成是这样的：

词	计　数	文　档
1.0	1	<1>
4	1	<3>
Apache	1	<3>
Cookbook	1	<3>
Elasticsearch	2	<1>,<2>
Mastering	1	<2>
Server	1	<1>
Solr	1	<3>

每一个词指向包含它的文档编号。这样就可以执行一种非常高效且快速的搜索，比如基于词的查询。此外，每个词有一个计数，告诉Lucene该词出现的频率。

当然，Lucene实际创建的索引要比这个复杂得多，也先进得多，它创建的额外文件包含了词向量（term vector）、文档值（doc value）等信息。然而，到现在为止，你需要知道的是数据怎么组织，而不是具体怎么存储。

每个索引分为多个“写一次，读多次”（write once and read many time）的段（segment）。建立索引时，一个段写入磁盘后就不能再更新。因此，被删除文档的信息存储在一个单独的文件中，但该段自身不被更新。

然而，多个段可以通过段合并（segments merge）合并在一起。当强制段合并或者Lucene决定合并时，这些小段就会由Lucene合并成更大的一些段。合并需要I/O。然而一些信息需要清除，

因为在合并时，不再需要的信息将被删除（例如，被删除的文档）。除此之外，检索大段比检索存有相同数据的多个小段速度更快。这是因为在一般情况下，搜索只需将查询词与那些被编入索引的词相匹配。通过多个小段寻找和合并结果，显然会比让一个大段直接提供结果慢得多。

1.1.2 输入数据分析

当然，问题是，传入文档中的数据怎样转化成倒排索引，查询文本怎样变成可被搜索的词？这个数据转化的过程被称为分析。你可能希望某些字段经语言分析器处理，使得car和cars在索引中被视为同一个。另外，你可能希望另一些字段只用空格或者小写划分。

分析的工作由分析器完成，它由一个分词器（tokenizer）和零个或多个标记过滤器（token filter）组成，也可以有零个或多个字符映射器（character mapper）。

Lucene中的分词器把文本分割成多个标记，基本就是词加上一些额外信息，比如该词在原始文本中的位置和长度。分词器的处理结果称为标记流（token stream），它是一个接一个的标记，准备被过滤器处理。

除了分词器，Lucene分析器包含零个或多个标记过滤器，用来处理标记流中的标记。下面是一些过滤器的例子。

- 小写过滤器（lowercase filter）：把所有的标记变成小写。
- 同义词过滤器（synonyms filter）：基于基本的同义词规则，把一个标记换成另一个同义的标记。
- 多语言词干提取过滤器（multiple language stemming filter）：减少标记（实际上是标记中的文本部分），得到词根或者基本形式，即词干。

过滤器是一个接一个处理的。所以我们通过使用多个过滤器，几乎可以达到无限的分析可能性。

最后，字符映射器对未经分析的文本起作用，它们在分词器之前工作。因此，我们可以很容易地从文本的整体部分去除HTML标签而无需担心它们被标记。

索引和查询

我们可能想知道当使用Lucene和所有建立在它之上的软件时，上述所有功能对索引和查询的影响。建立索引时，Lucene会使用你选择的分析器来处理你的文档内容。当然，不同的字段可以使用不同的分析器，所以文档的名称字段可以和汇总字段做不同的分析。如果我们愿意，也可以不分析字段。

查询时，查询将被分析。但是，你也可以选择不分析。记住这一点很关键，因为一些Elasticsearch查询被分析，一些则不然。例如，前缀和词查询不被分析，匹配查询则被分析。可以在被分析查询和不被分析查询两者中选择非常有用。有时，你可能希望查询一个未经分析

的字段，而有时你则希望有全文搜索的分析。如果我们查询LightRed这个词，标准分析器分析这个查询后，会去查询light和red；如果我们使用不经分析的查询类型，则会明确地查询LightRed这个词。

关于索引和查询分析，你应该记住的是，索引应该和查询词匹配。如果它们不匹配，Lucene不会返回所需文档。比如，你在建立索引时使用了词干提取和小写，那你应该保证查询中的词也必须是词干和小写，否则你的查询不会返回任何结果。重要的是在索引和查询分析时，对所用标记过滤器保持相同的顺序，这样被分析出来的词是一样的。

1.1.3　评分和查询相关性

另外还有件现在还没提到的事，就是评分（scoring）。什么是文档的得分？得分是根据文档和查询的匹配度用计分公式计算的结果。默认情况下，Apache Lucene使用TF/IDF（term frequency/inverse document frequency，词频/逆向文档频率）评分机制，这是一种计算文档在我们查询上下文中相关度的算法。当然，它不是唯一可用的算法，2.2节将介绍其他算法。

如果你想阅读更多关于Apache Lucene TF/IDF评分公式的内容，请访问Apache Lucene Javadocs中的`TFIDFSimilarity`类，网址：http://lucene.apache.org/core/4_6_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html。

请记住，Elasticsearch和Lucene计算的分数值越高，意味着文档越相关。一些参数（比如boost）、不同的查询类型（3.3节将讨论这些查询类型）、不同的评分算法，都会影响得分的计算。

如果您想更深入地了解Apache Lucene评分是如何工作的、默认算法是什么、分数是如何计算的，请参阅我们的书*Mastering ElasticSearch*，Packt出版。

1.2　Elasticsearch 基础

Elasticsearch是由Shay Banon发起的一个开源搜索服务器项目，2010年2月发布。迄今，该项目已发展成为搜索和数据分析解决方案领域的主要一员，广泛应用于声名卓著或鲜为人知的搜索应用程序。此外，由于其分布式性质和实时功能，许多人把它作为文档数据库。

1.2.1　数据架构的主要概念

让我们过一遍Elasticsearch的基本概念。如果你已经熟悉Elasticsearch架构，可以跳过本节。如果你不熟悉这种架构，请考虑阅读本节。我们将提到本书其余部分会用到的关键字。

1. 索引

索引（index）是Elasticsearch对逻辑数据的逻辑存储，所以它可以分为更小的部分。你可以把索引看成关系型数据库的表。然而，索引的结构是为快速有效的全文索引准备的，特别是它不存储原始值。如果你知道MongoDB，可以把Elasticsearch的索引看成MongoDB里的一个集合。如果你熟悉CouchDB，可以把索引看成CouchDB数据库索引。Elasticsearch可以把索引存放在一台机器或者分散在多台服务器上，每个索引有一或多个分片（shard），每个分片可以有多个副本（replica）。

2. 文档

存储在Elasticsearch中的主要实体叫文档（document）。用关系型数据库来类比的话，一个文档相当于数据库表中的一行记录。当比较Elasticsearch中的文档和MongoDB中的文档，你会发现两者都可以有不同的结构，但Elasticsearch的文档中，相同字段必须有相同类型。这意味着，所有包含`title`字段的文档，`title`字段类型都必须一样，比如`string`。

文档由多个字段组成，每个字段可能多次出现在一个文档里，这样的字段叫多值字段（multivalued）。每个字段有类型，如文本、数值、日期等。字段类型也可以是复杂类型，一个字段包含其他子文档或者数组。字段类型在Elasticsearch中很重要，因为它给出了各种操作（如分析或排序）如何被执行的信息。幸好，这可以自动确定，然而，我们仍然建议使用映射。与关系型数据库不同，文档不需要有固定的结构，每个文档可以有不同的字段，此外，在程序开发期间，不必确定有哪些字段。当然，可以用模式强行规定文档结构。从客户端的角度看，文档是一个JSON对象（关于JSON格式的更多内容，参见http://en.wikipedia.org/wiki/JSON）。每个文档存储在一个索引中并有一个Elasticsearch自动生成的唯一标识符和文档类型。文档需要有对应文档类型的唯一标识符，这意味着在一个索引中，两个不同类型的文档可以有相同的唯一标识符。

3. 文档类型

在Elasticsearch中，一个索引对象可以存储很多不同用途的对象。例如，一个博客应用程序可以保存文章和评论。文档类型让我们轻易地区分单个索引中的不同对象。每个文档可以有不同的结构，但在实际部署中，将文件按类型区分对数据操作有很大帮助。当然，需要记住一个限制，不同的文档类型不能为相同的属性设置不同的类型。例如，在同一索引中的所有文档类型中，一个叫`title`的字段必须具有相同的类型。

4. 映射

在有关全文搜索基础知识部分，我们提到了分析的过程：为建索引和搜索准备输入文本。文档中的每个字段都必须根据不同类型做相应的分析。举例来说，对数值字段和从网页抓取的文本字段有不同的分析，比如前者的数字不应该按字母顺序排序，后者的第一步是忽略HTML标签，因为它们是无用的信息噪音。Elasticsearch在映射中存储有关字段的信息。每一个文档类型都有自己的映射，即使我们没有明确定义。

1.2.2　Elasticsearch主要概念

现在，我们已经知道Elasticsearch把数据存储在一个或多个索引上，每个索引包含各种类型的文档。我们也知道了每个文档有很多字段，映射定义了Elasticsearch如何对待这些字段。但还有更多，从一开始，Elasticsearch就被设计为能处理数以亿计的文档和每秒数以百计的搜索请求的分布式解决方案。这归功于几个重要的概念，我们现在将更详细地描述。

1. 节点和集群

Elasticsearch可以作为一个独立的单个搜索服务器。不过，为了能够处理大型数据集，实现容错和高可用性，Elasticsearch可以运行在许多互相合作的服务器上。这些服务器称为集群（cluster），形成集群的每个服务器称为节点（node）。

2. 分片

当有大量的文档时，由于内存的限制、硬盘能力、处理能力不足、无法足够快地响应客户端请求等，一个节点可能不够。在这种情况下，数据可以分为较小的称为分片（shard）的部分（其中每个分片都是一个独立的Apache Lucene索引）。每个分片可以放在不同的服务器上，因此，数据可以在集群的节点中传播。当你查询的索引分布在多个分片上时，Elasticsearch会把查询发送给每个相关的分片，并将结果合并在一起，而应用程序并不知道分片的存在。此外，多个分片可以加快索引。

3. 副本

为了提高查询吞吐量或实现高可用性，可以使用分片副本。副本（replica）只是一个分片的精确复制，每个分片可以有零个或多个副本。换句话说，Elasticsearch可以有许多相同的分片，其中之一被自动选择去更改索引操作。这种特殊的分片称为主分片（primary shard），其余称为副本分片（replica shard）。在主分片丢失时，例如该分片数据所在服务器不可用，集群将副本提升为新的主分片。

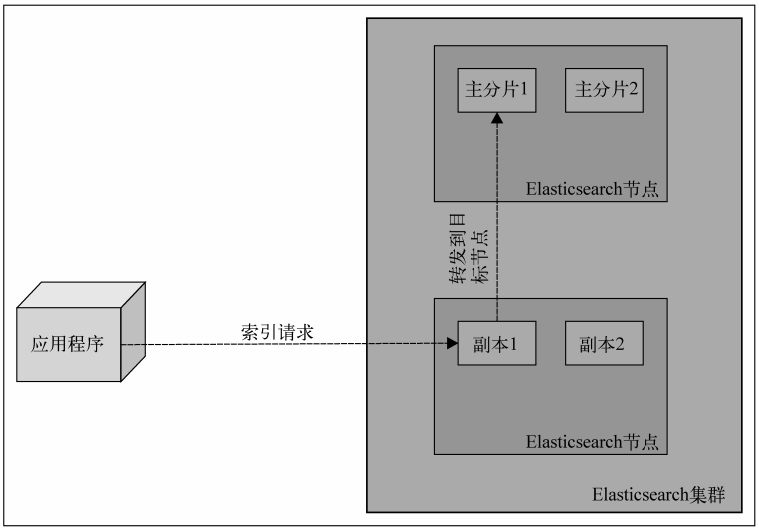
4. 时光之门

Elasticsearch处理许多节点。集群的状态由时光之门控制。默认情况下，每个节点都在本地存储这些信息，并且在节点中同步。我们将在7.2节详细讨论时光之门模块。

1.2.3　索引建立和搜索

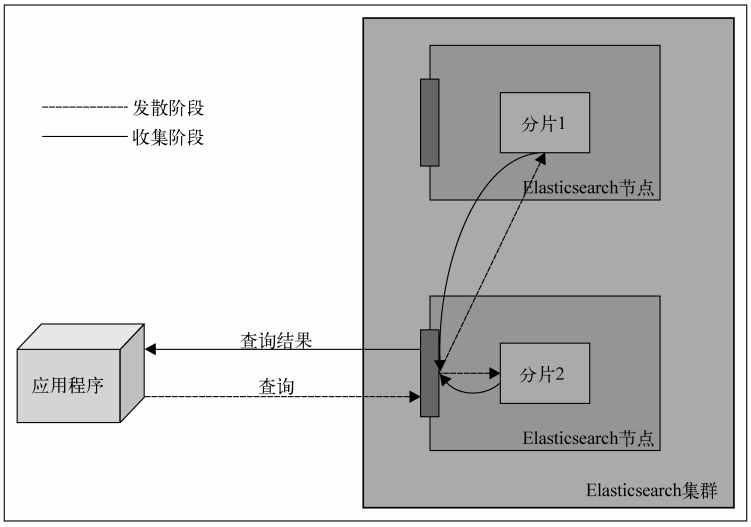
你可能会问实际上如何把所有的索引、分片和副本绑在单个环境里。理论上，当你必须知道你的文档在哪，哪台服务器、哪个分片上时，从集群获取数据非常困难。更为困难的是当一个搜索需要返回的文档分布在集群中不同节点的不同分片上时。这确实是一个复杂的问题，好在我们

不需要关心，它由Elasticsearch本身自动处理。来看看下图：



发送一个新的文档给集群时，你指定一个目标索引并发送给它的任意一个节点。这个节点知道目标索引有多少分片，并且能够确定哪个分片应该用来存储你的文档。可以更改Elasticsearch的这个行为。2.6.3节将对此进行讨论。现在你需要记住的重要信息是，Elasticsearch使用文档的唯一标识符来计算文档应该被放到哪个分片中。索引请求发送到一个节点后，该节点会转发文档到持有相关分片的目标节点中。

现在来看看关于执行搜索请求的图：



尝试用文档标识符来获取文档时，发送查询到一个节点，该节点使用同样的路由算法来决定持有文档的节点和分片，然后转发查询，获取结果，并把结果发送给你。另一方面，查询过程更为复杂。除非使用了路由，查询将直接转发到单个分片，否则，收到查询请求的节点会把查询转发给保存了属于给定索引的分片的所有节点，并要求与查询匹配的文档的最少信息（默认情况下是标识符和得分）。这个过程称为*发散阶段*（scatter phase）。收到这些信息后，该聚合节点（收到客户端请求的节点）对结果排序，并发送第2个请求来获取结果列表所需的文档（除了标识符和得分以外的所有信息）。这个阶段称为*收集阶段*（gather phase）。这个阶段执行完毕后，结果返回到客户端。

现在问题来了，在前面描述的过程中，副本扮演了什么角色呢？在建立索引时，副本只作为额外的位置来存储数据。当执行查询时，默认情况下，Elasticsearch会尽量平衡分片和它的副本之间的负载，使它们承受均衡的压力。此外，记住我们可以改变该行为。3.2节将对此进行讨论。

1.3　安装并配置集群

有几个安装Elasticsearch所需的步骤，接下来几节将详细说明。

1.3.1　安装Java

为了建立Elasticsearch，第一步是确保正确安装Java SE环境。Elasticsearch需要Java 6或更高版本。你可以从http://www.oracle.com/technetwork/java/javase/downloads/index.html下载。如果你想，也可以使用OpenJDK（http://openjdk.java.net/）。当然你可以使用Java 6，但它已经没有补丁的支持，所以建议安装Java 7。

1.3.2　安装Elasticsearch

从http://www.elasticsearch.org/download/下载，解压。选择最新的稳定版本，安装完毕。

写这本书时，我们用的是Elasticsearch 1.0.0 GA。这意味着，我们已经跳过一些被标记为过时（deprecated）的属性的描述，它们已经或将在未来的Elasticsearch版本中被移除。

与Elasticsearch交互的主要接口是基于HTTP协议和REST的。这意味着你甚至可以使用Web浏览器来完成基本的查询和请求，但对于更复杂的情况，你需要额外的命令行工具，比如cURL。如果你使用Linux或OS X命令，cURL已经可用了。如果你使用Windows，可以从http://curl.haxx.se/download.html下载。

1.3.3 在Linux上用二进制包安装Elasticsearch

安装Elasticsearch的另一个方法是使用提供的二进制包，RPM或DEB，视你的Linux发行版而定。这些二进制包可以在http://www.elasticsearch.org/download/找到。

1. 使用RPM包安装Elasticsearch

下载RPM包后，你只需执行如下命令：

```
sudo yum elasticsearch-1.0.0.noarch.rpm
```

就这么简单。如果一切顺利，Elasticsearch应该安装好了，配置文件应该在/etc/sysconfig/elasticsearch中。如你的操作系统基于红帽，应该可以使用/etc/init.d/elasticsearch下的`init`脚本。如果你的操作系统是SUSE Linux，可以使用/bin下的systemctl文件来启动和停止Elasticsearch服务。

2. 使用DEB包安装Elasticsearch

下载DEB包后，只需执行如下命令：

```
sudo dpkg -i elasticsearch-1.0.0.deb
```

就这么简单。如果一切顺利，Elasticsearch应该安装成功，配置文件存在/etc/elasticsearch/elasticsearch.yml。/etc/init.d/elasticsearch下的`init`脚本可以用来启动和停止Elasticsearch。此外，/etc/default/elasticsearch下的文件包含了环境设置。

1.3.4 目录布局

现在，到新创建的目录中。应该可以看到下面的目录结构：

目　录	描　述
bin	运行Elasticsearch实例和插件管理所需的脚本
config	配置文件所在的目录
lib	Elasticsearch使用的库

Elasticsearch启动后，会创建如下目录（如果目录不存在）：

目　录	描　述
data	Elasticsearch使用的所有数据的存储位置
logs	关于事件和错误记录的文件
plugins	存储所安装插件的地方
work	Elasticsearch使用的临时文件

1.3.5　配置Elasticsearch

Elasticsearch很容易入门，这是它越来越流行的原因之一，当然，不是全部原因。因为已为简单的环境配置了合理的默认值和自动设置，我们可以跳过配置，不需改变我们的配置文件中的任意一行而直接走到下一章。然而，为了真正理解Elasticsearch，学习一些可用的设置还是值得的。

现在，来探讨Elasticsearch的tar.gz存档提供的默认目录和文件的布局。整个配置位于config目录下，可以看到两个文件：elasticsearch.yml（或elasticsearch.json，如果有的话会被使用）和logging.yml。第一个文件负责设置服务器的默认配置值。重要的是，因为一些配置值可以在运行时更改，也可作为集群状态的一部分被保留，所以这个文件中的值可能不准确。有两个值不能在运行时更改，分别是`cluster.name`和`node.name`。

`cluster.name`属性保存集群的名字，不同的集群用名字来区分，配置成相同集群名字的各个节点形成一个集群。

`node.name`是实例（该节点）的名字，可以不定义此参数，这时，Elasticsearch自动选择一个唯一的名称。注意，此名称是每次启动时选择的，所以在每次重启后名称可能都不一样。在很长的时间区间或者重启过后，需要在API中提及具体实例名称，或者用监控工具查看节点，自定义一个名称还是很有帮助的。给你的节点想一个描述性的名字吧。

文件中的其他参数有很好的注释，所以建议你看看。不要担心不理解那些解释。希望在读完下面几章后，一切都变得清晰起来。

记住，大多数在elasticsearch.yml文件中设置的参数都可以用Elasticsearch REST API来覆盖。8.8节将介绍这些API。

第2个文件（logging.yml）定义了多少信息写入系统日志，定义了日志文件，并定期创建新文件。只有在调整监控、备份方案或系统调试时，才需要修改。然而如果想有一份更详细的日志，就需要相应调整。

我们保留这些配置文件不动。配置的一个重要部分是调整你的操作系统。在建立索引时，尤其是有很多分片和副本的情况下，Elasticsearch将创建很多文件。所以，系统不能限制打开的文件描述符小于32 000个。在Linux上，一般在/etc/security/limits.conf中修改，当前的值可以用`ulimit`命令来查看。如果达到极限，Elasticsearch将无法创建新的文件，所以合并会失败，索引会失败，新的索引无法创建。

下一组设定关联到单个Elasticsearch实例的Java虚拟机（JVM）的堆内存限制。对小型部署来说，默认的内存限制（1024 M）就足够了，但对于大型项目不够。如果你在日志文件中发现

OutOfMemoryError异常的条目，把ES_HEAP_SIZE变量设置到大于1024。当选择分配给JVM的合适内存大小时，记住，通常不应该分配超过50%的系统总内存。不过，所有的规则都有例外，稍后将更详细地讨论，但你应该经常监控JVM堆的使用量，需要时调整。

1.3.6 运行Elasticsearch

运行刚刚下载并解压的ZIP包，转到bin目录，然后根据不同的操作系统，运行如下命令。

- Linux或OS X：./elasticsearch。
- Windows：elasticsearch.bat。

恭喜你！现在把Elasticsearch启动并运行起来了。它在工作时一般使用2个端口号：第1个是使用HTTP协议与REST API通信的端口，第2个是传输模块（transport module），是用来在集群内以及Java客户端和集群之间通信的端口。HTTP API的默认端口号是9200，所以可以在浏览器中打开http://127.0.0.1:9200来检查搜索是否就绪，浏览器将显示类似下面这样的代码片段：

```
{
  "status" : 200,
  "name" : "es_server",
  "version" : {
    "number" : "1.0.0",
    "build_hash" : "a46900e9c72c0a623d71b54016357d5f94c8ea32",
    "build_timestamp" : "2014-02-12T16:18:34Z",
    "build_snapshot" : false,
    "lucene_version" : "4.6"
  },
  "tagline" : "You Know, for Search"
}
```

输出是JSON结构的。如果你还不熟悉JSON（JavaScript Object Notation），请花几分钟阅读http://en.wikipedia.org/wiki/JSON上的这篇文章。

Elasticsearch很聪明。如果默认端口不可用，引擎将绑定到下一个可用端口，你可以在启动时的控制台上找到如下相关信息：

```
[2013-11-16 11:56:12,101][INFO ][http] [Red Lotus]
  bound_address {inet[/0:0:0:0:0:0:0:0%0:9200]},
    publish_address {inet[/192.168.1.101:9200]}
```

注意[http]的片段。Elasticsearch使用了一些端口完成各种任务。我们所使用的接口是由HTTP模块处理的。

现在，将使用cURL程序。例如，要检查集群健康度，会使用以下命令：

```
curl -XGET http://127.0.0.1:9200/_cluster/health?pretty
```

参数-X是一个请求方法，默认值是GET（所以在上面的例子中，可以忽略此参数）。暂时不要担心GET这个值，本章的后面将更详细地描述它。

作为一个标准，API返回的JSON对象信息里，换行符是被省略的，在请求中加上pretty参数是强制Elasticsearch在响应中加上换行符，使之更可读。你可以试着去掉pretty参数运行上面的请求，看看有什么不同。

Elasticsearch在中小型应用程序中非常有用，但它的初衷是建成大型集群。所以，现在来建立由两个节点组成的大的集群。解压Elasticsearch到另一个目录，然后运行第二个实例。我们会在日志中看到如下内容：

```
[2013-11-16 11:55:16,767][INFO ][cluster.service]
[Stane, Obadiah] detected_master [Martha Johansson]
[vswsFRWTSjOa_fy7uPuOMA]
[inet[/192.168.1.19:9300]], added {[Martha Johansson]
[vswsFRWTSjOa_fy7uPuOMA]
[inet[/192.168.1.19:9300]],}, reason: zen-disco-receive(from master
[[Martha Johansson][vswsFRWTSjOa_fy7uPuOMA]
[inet[/192.168.1.19:9300]]])
```

这意味着我们的第二个实例（名字为Stane,Obadiah）检测到了之前运行的实例（名字为Martha Johansson）。这里，Elasticsearch自动形成了一个新的双节点集群。

请注意，在某些系统上，防火墙软件默认自动打开，可能导致节点无法找到其他节点。

1.3.7　关掉Elasticsearch

尽管我们期望集群或节点完美地一直运行下去，但仍可能需要正确地重启或者关闭，比如，为了维护。下面是三种可以关闭Elasticsearch的方法。

- 如果节点是连接到控制台，按下Ctrl+C。
- 第二种选择是通过发送TERM信号杀掉服务器进程（参考Linux上的kill命令和Windows上的任务管理器）。
- 第三种方法是使用REST API。

现在着重介绍第三种方法。可以执行以下命令来关掉整个集群：

```
curl -XPOST http://localhost:9200/_cluster/nodes/_shutdown
```

为关闭单一节点，假如节点标识符是BlrmMvBdSKiCeYGsiHijdg，可以执行下面的命令：

```
curl -XPOST
http://localhost:9200/_cluster/nodes/BlrmMvBdSKiCeYGsiHijdg/_shutdown
```

节点的标识符可以在日志中看到，或者使用_cluster/nodes API，命令如下：

```
curl -XGET http://localhost:9200/_cluster/nodes/
```

1.3.8 Elasticsearch作为系统服务运行

Elasticsearch 1.0可以作为服务运行在基于Linux的系统和基于Windows的系统上。

1. 在Linux上运行系统服务

如果是从提供的二进制包安装的Elasticsearch，你已经完成了，什么都不用担心。但是，如果你刚刚下载归档文件，解压到所选择的目录，就需要做一些额外的工作。为了将Elasticsearch安装成一个Linux系统服务，将使用Elasticsearch service wrapper，你可以从https://github.com/elasticsearch/elasticsearch-servicewrapper下载。

来看看使用Elasticsearch service wrapper建立Elasticsearch Linux服务的步骤。首先，执行以下命令来下载这个wrapper：

```
curl -L http://github.com/elasticsearch/elasticsearch-
  servicewrapper/tarball/master  | tar -xz
```

假设Elasticsearch已经安装在/usr/local/share/elasticsearch下，执行如下命令来移动所需的wrapper文件：

```
sudo mv *servicewrapper*/service/usr/local/share/elasticsearch/bin/
```

执行如下命令来移除剩余的文件

```
rm -Rf *servicewrapper*
```

最后，通过执行install命令来安装服务：

```
sudo /usr/local/share/elasticsearch/bin/service/elasticsearch install
```

在这之后，需要创建一个符号链接指向/usr/local/bin/rcelasticsearch下的/usr/local/share/elasticsearch/bin/service/elasticsearch脚本。可通过运行如下命令来实现：

```
sudo ln -s 'readlink  -f
  /usr/local/share/elasticsearch/bin/service/elasticsearch'
    /usr/local/bin/rcelasticsearch
```

就这样。如果你想启动Elasticsearch，执行如下命令：

```
/etc/init.d/elasticsearch start
```

2. 在Windows上运行系统服务

在Windows下把Elasticsearch安装为系统服务非常容易，你只需转到Elasticsearch的安装目录，

到bin子目录下，执行:

```
service.bat install
```

你会被问及操作权限，允许脚本运行，Elasticsearch就被安装成一个Windows服务。

如果你想看看所有被service.bat脚本文件暴露出来的命令，在相同目录下执行：

```
service.bat
```

例如，为了启动Elasticsearch，可执行如下命令：

```
service.bat start
```

1.4　用 REST API 操作数据

Elasticsearch REST API可用于各种任务。有了它，可以管理索引，更改实例参数，检查节点和群集状态，索引数据，搜索数据或者通过GET API检索文档。但是现在，我们将集中在API中的CRUD（create-retrieve-update-delete，增删改查）部分，它让我们能像使用NoSQL数据库一样使用Elasticsearch。

1.4.1　理解Elasticsearch的RESTful API

在一个类REST的架构中，每个请求都指向地址路径所表示的一个具体对象。如果/books/是一个图书馆中图书列表的引用，/books/1则引用ID为1的那本书。注意这些对象可以嵌套，/books/1/chapter/6表示图书馆的第一本书的第6章，等等。我们的API调用有个主题。我们想执行的操作（比如GET或POST操作）怎么样？请求类型就是用来指定这个的。HTTP协议给出了可以在API调用中用作动词的一组相当长的类型。合乎逻辑的选择是，GET用来获得请求对象的当前状态，POST来改变对象的当前状态，PUT创建一个对象，而DELETE销毁对象，另外还有个HEAD请求仅仅用来获取对象的基础信息。

现在来看看在1.3.7节中讨论的如下操作例子，一切都应该更容易理解。

- `GET http://localhost:9000/`：这个命令用来获取Elasticsearch的基本信息。
- `GET http://localhost:9200/_cluster/state/nodes/`：这个命令获取集群中节点的信息。
- `POST http://localhost:9200/_cluster/nodes/_shutdown`：这个命令向集群中所有节点发送一个shutdown请求。

我们现在至少知道了REST的一般概念。你可以在http://en.wikipedia.org/wiki/Representational_state_transfer上阅读更多关于REST的信息。现在，可以继续学习如何使用Elasticsearch API来存储、

读取、修改和删除数据。

1.4.2 在Elasticsearch中存储数据

我们已经讨论过，在Elasticsearch中，所有的数据，即每个文档，都有定义好的索引和类型。每个文档可以包含一个或多个字段来保存数据。首先展示如何使用Elasticsearch为一个简单文档建立索引。

1.4.3 新建文档

现在，尝试索引一些文档。例如，为博客建立某种内容管理系统（CMS）。文章（article）是博客中的一个实体。

使用JSON标记，一个文档可以如下所示的例子来表示：

```
{
  "id": "1",
  "title": "New version of Elasticsearch released!",
  "content": "Version 1.0 released today!",
  "priority": 10,
  "tags": ["announce", "elasticsearch", "release"]
}
```

可以看到，JSON文档包含一组字段，每个字段可以有不同的形式。在以上示例中，我们有数字（`priority`）、文本（`title`）和字符串数组（`tags`）。以下示例将展示其他类型。如本章前面所述，Elasticsearch能猜出这些类型（因为JSON是半类型化的，例如，数字没有放在引号中），并自动定制这些数据在其内部结构中如何存储。

当然，我们希望为示例文档建立索引，并使其可用于搜索。我们将使用一个名为`blog`的索引和名为`article`的类型。为了把示例文档以给定类型、标识符为1建立在索引中，执行以下命令：

```
curl -XPUT http://localhost:9200/blog/article/1 -d '{"title": "New version of
Elasticsearch released!", content": "Version 1.0 released today!", "tags": ["announce",
"elasticsearch", "release"] }'
```

注意cURL命令的一个新选项：`-d`参数。此选项的值是将作为请求负载的文本，也即请求主体（request body）。这样，我们可以发送附加信息，如文档定义。同时，注意唯一标识符（1）是放在URL，而不是请求主体中。使用HTTP `PUT`请求时，如果省略此标识符，该请求将返回以下错误：

```
No handler found for uri [/blog/article/] and method [PUT]
```

如果一切正确，Elasticsearch会返回一个JSON响应，与如下输出类似：

```
{
  "_index":"blog",
  "_type":"article",
  "_id":"1",
  "_version":1
}
```

前面的响应包含了此次操作状态的信息，并显示一个新的文档放在哪里，还包含了有关文档的唯一标识符（_id）和当前版本（_version）的信息。版本将由Elasticsearch每次更新时自动递增。

标识符的自动创建

在上面的示例中，我们自己指定了文档标识符。然而，Elasticsearch可以自动生成它。这似乎很方便，但只有当该索引是唯一的数据来源时，才能这么做。如果使用一个数据库来存储数据，用Elasticsearch全文搜索，那数据同步将会被阻碍，除非在数据库中也存储生成的标识符。使用HTTP POST请求类型并且不在URL中指定标识符，就可以生成一个唯一标识符。例如，看下面的命令：

```
curl -XPOST http://localhost:9200/blog/article/ -d '{"title": "New version of
Elasticsearch released!", "content": "Version 1.0  released today!", "tags":
["announce", "elasticsearch", "release"] }'
```

注意，要使用HTTP POST方法，而不是前面示例中的PUT方法。参考前面关于REST动词的描述，我们想更改列表中的文档索引，而不是创建一个新的实体，所以使用POST而不是PUT。服务器应该返回类似下面的响应：

```
{
  "_index" : "blog",
  "_type" : "article",
  "_id" : "XQmdeSe_RVamFgRHMqcZQg",
  "_version" : 1
}
```

注意加粗的那一行，这是一个Elasticsearch自动生成的标识符。

1.4.4　检索文档

我们已经将实例存储在了文档中，现在尝试通过标识符检索。首先执行以下命令：

```
curl -XGET http://localhost:9200/blog/article/1
```

Elasticsearch将返回类似下面的响应：

```
{
  "_index" : "blog",
  "_type" : "article",
  "_id" : "1",
```

```
  "_version" : 1,
  "exists" : true,
  "_source" : {
    "title": "New version of Elasticsearch released!",
    "content": "Version 1.0 released today!",
    "tags": ["announce", "elasticsearch", "release"]
  }
```

在前面的响应中，除了索引、类型、标识符和版本，还可以看到说明“发现文件存在”（exists属性）以及此文档来源（_source属性）的信息。如果没有找到文档，得到的响应如下所示：

```
{
  "_index" : "blog",
  "_type" : "article",
  "_id" : "9999",
  "exists" : false
}
```

因为没有找到文档，当然也就没有版本或来源的信息。

1.4.5 更新文档

更新索引中的文档是一项更复杂的任务。在内部，Elasticsearch必须首先获取文档，从_source属性获得数据，删除旧的文件，更改_source属性，然后把它作为新的文档来索引。它如此复杂，因为信息一旦在Lucene的倒排索引中存储，就不能再被更改。Elasticsearch通过一个带_update参数的脚本来实现它。这样就可以做比简单修改字段更加复杂的文档转换。下面用简单的例子看看的工作原理。

请记住之前建立的博客文章索引。为了更改其content字段，运行以下命令：

```
curl -XPOST http://localhost:9200/blog/article/1/_update -d '{
  "script": "ctx._source.content = \"new content\""
}'
```

Elasticsearch将返回如下响应：

```
{"_index":"blog","_type":"article","_id":"1","_version":2}
```

看上去更新操作执行成功了。为了确定，我们用它的标识符检索一下，执行如下命令：

```
curl -XGET http://localhost:9200/blog/article/1
```

Elasticsearch的响应应该包含修改过的content字段，事实的确如此，它包含如下信息：

```
{
"_index" : "blog",
"_type" : "article",
"_id" : "1",
```

```
"_version" : 2,
"exists" : true,
"_source" : {
  "title":"New version of Elasticsearch released!",
  "content":"new content",
  "tags":["announce","elasticsearch","release"]
}
```

Elasticsearch修改了文章的content和该文档的版本号。注意，不必发送整个文档，只需发送改变的部分。但是请记住，为了使用更新功能，需要使用_source字段，2.4节将描述如何使用_source字段。

关于文档更新，还有一点，如果你的脚本需要更新文档的一个字段，你可以设置一个值用来处理文档中没有该字段的情况。例如，想增加文档中的counter字段，而该字段不存在，你可以在请求中使用upsert节来提供字段的默认值。看下面的例子：

```
curl -XPOST http://localhost:9200/blog/article/1/_update -d '{
  "script": "ctx._source.counter += 1",
  "upsert": {
    "counter" : 0
  }
}'
```

执行这个示例，Elasticsearch会在示例文档中添加一个值为0的counter字段。这是因为我们的文档没有counter字段，而我们在更新请求中指定了upsert节。

1.4.6　删除文档

我们已经看到如何创建（PUT）、检索（GET）和更新文档，不难猜到，删除文档的过程是类似的：需要使用DELETE请求类型发送一个适当的HTTP请求。例如，要删除示例文档，运行以下命令：

```
curl -XDELETE http://localhost:9200/blog/article/1
```

Elasticsearch的响应如下所示：

```
{"found":true,"_index":"blog","_type":"article","_id":"1","_version":3}
```

这意味着我们找到并删除了该文档。

现在可以利用CRUD操作。我们已经可以使用Elasticsearch作为一个简单的键值存储来创建应用程序。但这仅仅是开始！

1.4.7　版本控制

在提供的例子中，你可能注意到了文档的版本信息，它看起来如下所示：

```
"_version" : 1
```

仔细观察，你会发现在更新相同标识符的文档后，这个版本是递增的。默认情况下，Elasticsearch在添加、更改或删除文档时都会递增版本号。除了告诉我们对文档所做更改的次数，还能够实现乐观锁(optimistic locking，http://en.wikipedia.org/wiki/Optimistic_concurrency_control)。这允许我们在并发处理同一文档时避免问题。例如，在两个不同的应用程序中读取相同的文档，分别修改它，然后尝试更新到Elasticsearch。没有版本控制，我们将看到最后更新的版本。使用乐观锁，Elasticsearch保证数据的准确性，尝试写入一个已更改的文档将会失败。

1. 版本控制的一个例子

我们来看一个使用版本控制的示例。假设要删除`library`索引中类型为`book`、`id`为1的文档。我们也要确保如果文档没有更新，则删除操作成功。需要做的是添加一个值为1的`version`参数，如下所示：

```
curl -XDELETE 'localhost:9200/library/book/1?version=1'
```

如果索引中文档的版本不等于1，Elasticsearch将返回如下错误：

```
{
  "error": "VersionConflictEngineException[[library][4] [book][1]:
    version conflict, current [2], provided [1]]",
   "status": 409
}
```

在我们的示例中，Elasticsearch比较我们声明的版本号和在Elasticsearch中文档的版本号，发现不一样，所以操作失败。

2. 使用外部系统提供的版本

Elasticsearch也可基于我们提供给它的版本号。在版本存储在外部系统时，这是必要的。这种情况下，当你新索引一个文档时，应该如上面的示例一样提供一个`version`参数。这时，Elasticsearch将只检查提供的版本是否比当前保存在索引中的版本大(大多少并不重要)，如果是，操作成功，否则将失败。为了告诉Elasticsearch我们要使用外部版本跟踪，除了`version`参数外，还需要添加`version_type=external`参数。

例如，在系统添加文档版本123456，将运行如下命令：

```
curl -XPUT 'localhost:9200/library/book/1?version=123456' -d {...}
```

即使文档被移除后，Elasticsearch仍然可以检查版本号。这是因为Elasticsearch保留了删除文档的版本信息。默认情况下，此信息在删除的60秒内可用。可以通过修改`index.gc_deletes`配置参数来更改这个值。

1.5　使用 URI 请求查询来搜索

进入Elasticsearch查询的详细信息之前，先使用其中简单的URI请求来搜索。当然，第3章将扩展使用Elasticsearch搜索的知识，但是现在，先使用最简单的方法。

1.5.1　示例数据

本节将创建一个简单的索引，它有两个文档类型。为此，运行以下命令：

```
curl -XPOST 'localhost:9200/books/es/1' -d '{"title":"Elasticsearch Server",
  "published": 2013}'
curl -XPOST 'localhost:9200/books/es/2' -d '{"title":"Mastering Elasticsearch",
  "published": 2013}'
curl -XPOST 'localhost:9200/books/solr/1' -d '{"title":"Apache Solr 4 Cookbook",
  "published": 2012}'
```

运行上述命令将创建books索引，该索引包含两种类型：es和solr。Title和published字段将被索引。如果你想检查，可以通过运行以下命令映射API，2.2节将讨论映射：

```
curl -XGET 'localhost:9200/books/_mapping?pretty'
```

Elasticsearch将返回整个索引的所有映射。

1.5.2　URI请求

Elasticsearch的所有查询都发送到_search端点。你可以搜索单个或多个索引，也可以将搜索范围缩小到给定的一个或多个文档类型。例如，为了寻找books索引，运行以下命令：

```
curl -XGET 'localhost:9200/books/_search?pretty'
```

如果还有另一个索引叫clients，也可对这两个索引执行一个查询：

```
curl -XGET 'localhost:9200/books,clients/_search?pretty'
```

以同样的方式，还可以选择搜索时要使用的类型。如果只想在books索引的es类型中搜索，将运行如下命令：

```
curl -XGET 'localhost:9200/books/es/_search?pretty'
```

请记住，为了搜索一个给定的类型，需要指定一个或多个索引。如果要寻找任意索引，只需要设置星号（*）为索引名称，或忽略索引名称。Elasticsearch在选择索引名称时支持相当丰富的语义。如果你有兴趣，请参考http://www.lasticsearch.org/guide/en/elasticsearch/reference/current/multi-index.html。

还可以省略索引和类型来搜索所有索引。例如，以下命令将搜索集群中的所有数据：

```
curl -XGET 'localhost:9200/_search?pretty'
```

1. Elasticsearch查询响应

假设想找到books索引中title字段包含elasticsearch一词的所有文档，可以运行以下查询：

```
curl -XGET
  'localhost:9200/books/_search?pretty&q=title:elasticsearch'
```

Elasticsearch返回的响应如下所示：

```
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 2,
    "max_score" : 0.625,
    "hits" : [ {
      "_index" : "books",
      "_type" : "es",
      "_id" : "1",
      "_score" : 0.625, "_source" : {"title":"Elasticsearch Server",
        "published": 2013}
    }, {
      "_index" : "books",
      "_type" : "es",
      "_id" : "2",
      "_score" : 0.19178301, "_source" : {"title":"Mastering
        Elasticsearch", "published": 2013}
    } ]
  }
}
```

响应的第一部分告诉我们该请求花了多少时间（took属性，单位是毫秒），有没有超时（timed_out属性），执行请求时查询的分片信息，包括查询的分片数量（_shards对象的total属性）、成功返回结果的分片数量（_shards对象的successful属性）、失败的分片数量（_shards对象的failed属性）。如果查询执行时间比预想的更长，它可能会超时（可以使用timeout参数指定查询的最大执行时间）。可以使用超时参数，指定最大查询执行时间。失败的分片意味着分片出了问题或在执行搜索时不可用。

当然，上述信息很有用，但是通常我们对hits对象中返回的结果感兴趣。我们有查询返回的文档总数（total属性）和计算所得的最高分（max_score属性），还有包含返回文档的hits

数组。在本例中，每个返回的文档包含索引（_index属性）、类型（_type属性）、标识符（_id属性）、得分（_score属性）和_source字段（通常，这是发送到索引的JSON对象。这一内容将在2.4节讨论）。

2. 查询分析

你可能觉得奇怪为什么上一节运行的查询可以返回结果。用Elasticsearch建立索引，然后用elasticsearch来执行查询，虽然大小写不同，还是可以找到相关文档，原因就是查询分析。在建立索引时，底层的Lucene库根据Elasticsearch配置文件分析文档并建立索引数据。默认情况下，Elasticsearch会告诉Lucene对基于字符串的数据和数字都做索引和分析。查询阶段也一样，因为URI请求查询会映射到query_string查询（将在第3章讨论），Elasticsearch会分析它。

使用索引分析API（indices analyze API，http://www.elasticsearch.org/guide/en/ elasticsearch/eference/current/indices-analyze.html），可以看到分析过程是怎样的，在建立索引时发生了什么，在查询阶段又发生了什么。

为了看到title字段上的短语“Elasticsearch Server”建立的索引具体是什么，可以执行以下命令：

```
curl -XGET 'localhost:9200/books/_analyze?field=title' -d
  'Elasticsearch Server'
```

响应如下：

```
{
  "tokens" : [ {
    "token" : "elasticsearch",
    "start_offset" : 0,
    "end_offset" : 13,
    "type" : "<ALPHANUM>",
    "position" : 1
  }, {
    "token" : "server",
    "start_offset" : 14,
    "end_offset" : 20,
    "type" : "<ALPHANUM>",
    "position" : 2
  } ]
}
```

可以看到，Elasticsearch把文本划分为两个词，第一个标记值（token value）为elasti search，第二个标记值为server。

现在看看查询文本是如何被分析的，运行以下命令：

```
curl -XGET 'localhost:9200/books/_analyze?pretty&field=title' -d
  'elasticsearch'
```

响应如下：

```
{
  "tokens" : [ {
    "token" : "elasticsearch",
    "start_offset" : 0,
    "end_offset" : 13,
    "type" : "<ALPHANUM>",
    "position" : 1
  } ]
}
```

可以看到，这个词和传到查询的原始值是一样的。我们不会详细介绍Lucene查询以及查询解析器如何构建查询，但总地来说，分析之后的索引词和分析之后查询词是一样的，因此，该文档与查询匹配并作为结果返回。

3. URI查询中的字符参数

有几个参数，可以用来控制URI查询行为，现在来讨论一下。查询中的每个参数应加上&字符，如以下示例所示：

```
curl -XGET
  'localhost:9200/books/_search?pretty&q=published:
  2013&df=title&explain= rue&default_operator=AND'
```

请记得'字符，因为在类Linux系统上，&字符会被Linux shell解析。

(1) 查询

参数q用来指定我们希望文件匹配的查询条件。可以使用Lucene查询语法来指定查询，1.5.3节会描述。例如，一个简单的查询可能类似q=title:elasticsearch。

(2) 默认查询字段

使用df参数，可以指定在q参数中没有字段时应该默认使用的字段。默认情况下，将使用_all字段。Elasticsearch把其他所有字段的内容复制到_all字段。2.4节将更深入地讨论。一个df参数的例子是df=title。

(3) 分析器

可以将analyzer属性定义用于分析查询的分析器名称。默认情况下，索引阶段对字段内容做分析的分析器将用来分析我们的查询。

(4) 默认操作符

Default_operator属性可以设置成OR或AND，用来指定用于查询的默认布尔运算符。默认情况下，它设置为OR，意味着只要有一个查询条件匹配，就将返回文档。此参数设置为AND时，所有查询条件都匹配时才会返回文档。

(5) 查询解释

如果将explain参数设置为true，Elasticsearch将在结果的每个文档里包括额外的解释信息，如文档是从哪个分片上获取的、计算得分的详细信息（5.7节将深入讨论）。记住，不要在正常的搜索查询中设置explain为true，因为它需要额外的资源并使查询的性能下降。下面的代码是一个例子：

```
{
  "_shard" : 3,
  "_node" : "kyuzK62NQcGJyhc2gI1P2w",
  "_index" : "books",
  "_type" : "es",
  "_id" : "2",
  "_score" : 0.19178301, "_source" : {"title":"Mastering
    Elasticsearch", "published": 2013},
  "_explanation" : {
    "value" : 0.19178301,
    "description" : "weight(title:elasticsearch in 0)
      [PerFieldSimilarity], result of:",
    "details" : [ {
      "value" : 0.19178301,
      "description" : "fieldWeight in 0, product of:",
      "details" : [ {
        "value" : 1.0,
        "description" : "tf(freq=1.0), with freq of:",
        "details" : [ {
          "value" : 1.0,
          "description" : "termFreq=1.0"
        } ]
      }, {
        "value" : 0.30685282,
        "description" : "idf(docFreq=1, maxDocs=1)"
      }, {
        "value" : 0.625,
        "description" : "fieldNorm(doc=0)"
      } ]
    } ]
  }
}
```

(6) 返回字段

默认情况下，返回的每个文档中，Elasticsearch将包括索引名称、类型名称、文档标识符、得分和_source字段。我们可以修改这个行为，通过添加fields参数并指定一个以逗号分隔的字段名称列表。这些字段将在存储字段（如果存在的话）或内部_source字段中检索。默认情况下，字段的fields参数值是_source。一个例子是fields=title。

也可以加上_source参数并把值设为false，来禁用_source字段的读取。

(7) 结果排序

通过使用sort参数，可以指定自定义排序。Elasticsearch的默认行为是把返回文档按它们的得分降序排列。如果想有不同的排序，则需要指定sort参数。例如，添加sort=published:desc，文档将按published字段降序排序；添加sort=published:asc，则告诉Elasticsearch把文档按published字段升序排序。

如果指定自定义排序，Elasticsearch将省略计算文档的_score字段。这可能不是你想要的。如果在自定义排序的同时还想保持追踪每个文档的得分，你应该把track_scores=true添加到你的查询。请注意，进行自定义排序时跟踪分数，会使查询稍微慢一点（你可能根本察觉不到），因为需要处理能力来计算得分。

(8) 搜索超时

默认情况下，Elasticsearch没有查询超时，但你可能希望查询在一段时间（比如5秒）后超时。Elasticsearch允许你设置timeout参数。查询将执行到给定的timeout值，在那一刻，收集的结果将返回。把timeout=5s添加到你的查询，就可指定一个5秒的超时。

(9) 查询结果窗口

Elasticsearch允许你指定结果窗口（应返回的结果列表中文件的范围）。有两个参数用来指定结果窗口大小：size和from。size参数默认为10，它定义了返回结果的最大数量。from参数的默认值为0，它指定结果应该从哪个记录开始返回。为了从第11个开始返回5个文档，我们将在查询中添加以下参数：size=5&from=10

(10) 搜索类型

URI查询允许使用search_type参数指定搜索类型，搜索类型默认为query_then_fetch。我们可以使用以下6个值：

- dfs_query_then_fetch
- dfs_query_and_fetch
- query_then_fetch
- query_and_fetch
- count
- scan

3.2节将介绍更多搜索类型的相关知识。

(11) 小写扩展词

一些查询使用查询扩展，比如前缀查询（prefix query），3.9节将讨论这一内容。可以使用

lowercase_expanded_terms属性来定义扩展词是否应该被转为小写。默认该属性为true，意味着扩展词将被小写。

(12) 分析通配符和前缀

默认情况下，通配符查询和前缀查询不会被分析。如果要更改此行为，可以把analyze_ildcard属性设置为true。

1.5.3　Lucene查询语法

我们认为最好大致了解在URI查询里的q参数中可以使用的语法。Elasticsearch中的一些查询，比如正在讨论的这个查询，支持使用Lucene查询解析器语法，这是一种用来构建查询的语言。来看看它并讨论一些基本功能。如果想阅读完整的Lucene查询语法，请访问如下网页：http://lucene.apache.org/core/4_6_1/queryparser/org/apache/lucene/queryparser/classic/package-summary.html。

我们传到Lucene的查询被查询解析器分为词（term）和操作符（operator）。先从词开始，你可以区分两种类型的词：单词和短语。例如，为了查询title字段中的book一词，传入如下查询：

```
title:book
```

为了查询title字段中elasticsearch book这个短语，传入如下查询：

```
title:"elasticsearch book"
```

你可能已经注意到，字段的名字在前面，单词或短语在后面。

前面说过，Lucene查询语法支持操作符。例如，操作符+告诉Lucene给定部分必须在文档中匹配。操作符-正相反，查询的这一部分不能出现在文档中。查询中既没有+又没有 – 操作符的部分将被视为可以匹配、但非强制性的查询。所以，如果想找title字段包含book一词但description字段不包含cat一词的文档，传入以下查询：

```
+title:book -description:cat
```

也可以用括号来组合多个词，如下面的查询：

```
title:(crime punishment)
```

还可以使用^操作符接上一个值来助推（boost）[①]一个词，比如以下查询：

```
title:book^4
```

① boost将加强查询对该词的相关性。——译者注

1.6 小结

本章介绍了什么是全文搜索，以及Apache Lucene是如何实现的；熟悉了Elasticsearch的基本概念和它的顶层架构；使用Elasticsearch REST API来索引、更新、检索，最终删除数据；最后，使用简单的URI查询搜索了我们的数据。下一章的重点是建立索引数据。我们将看到Elasticsearch索引的工作原理，主分片和其副本的作用。还会看到Elasticsearch如何处理它不知道的数据，或者说如何创建我们自己的映射，也就是描述索引结构的JSON结构。我们还将学习如何使用批量索引来加快索引过程，可以存储什么额外的信息来帮助实现目标。此外，我们将讨论什么是索引段，什么是段合并以及如何调整段。最后，我们将看到在Elasticsearch中路由是如何工作的，以及在谈到索引路由和查询路由时，有什么选择。