第 2 章 索　　引

上一章介绍了关于全文搜索和Elasticsearch的基础知识，也知道了什么是Apache Lucene，还了解到Elasticsearch的安装、标准目录布局及注意事项。我们创建了索引，检索并更新了数据，最后使用简单的URI查询从Elasticsearch获得数据。在本章结束之时，你将学到以下内容：

- Elasticsearch索引；
- 配置索引结构映射，知道可使用的字段类型；
- 使用批量索引加快索引过程；
- 使用附加的内部信息扩展索引结构；
- 理解、设置及控制段合并；
- 理解路由的工作原理，并根据需求设置。

2.1　Elasticsearch 索引

我们已经启动并运行Elasticsearch集群，知道了如何使用Elasticsearch REST API索引、删除和检索数据，如何通过搜索来获取文档。如果你用过SQL数据库，或许会知道，在存入数据前，需要创建用来描述数据的一个结构。尽管Elasticsearch是一个无模式的搜索引擎，可以即时算出数据结构，但我们仍认为由自己控制并定义结构是更好的方法。在接下来的内容中，你将看到如何创建和删除索引。在深入了解可用的API方法之前，先了解一下建立索引的过程。

2.1.1　分片和副本

回忆之前的章节，Elasticsearch索引是由一个或多个分片组成的，每个分片包含了文档集的一部分。而且这些分片也可以有副本，它们是分片的完整副本。在创建索引的过程中，可以规定应创建的分片及副本的数量。也可以忽略这些信息，并使用全局配置文件（elasticsearch.yml）定义的默认值，或Elasticsearch内部实现的默认值。如果我们依赖Elasticsearch的默认值，索引结束时将得到5个分片及1个副本。这意味着什么？简单来说，操作结束时，将有10个Lucene索引分布在集群中。

想知道如何通过5个分片和1个副本计算得出有10个Lucene索引？“副本”（replica）这个术语有些误导。它意味着每一个分片都有自己的分片副本（copy），所以实际上有5个分片和5个相应分片副本。

一般而言，同时具有分片和与其相应的副本，意味着建立索引文档时，两者都得修改。这是因为要使分片得到精确的副本，Elasticsearch需将分片的变动通知所有副本。若要读取文件，可以使用分片或者其副本。在具有许多物理节点的系统中，可以把分片和副本放置于不同节点上，从而发挥更多处理能力（如磁盘I/O或CPU）。综上所述，得出结论如下。

- 更多分片使索引能传送到更多服务器，意味着可以处理更多文件，而不会降低性能。
- 更多分片意味着获取特定文档所需的资源量会减少，因为相较于部署更少分片时，存储在单个分片中的文件数量更少。
- 更多分片意味着搜索索引时会面临更多问题，因为必须从更多分片中合并结果，使得查询的聚合阶段需要更多资源。
- 更多副本会增强集群系统的容错性，因为当原始分片不可用时，其副本将替代原始分片发挥作用。只拥有单个副本，集群可能在不丢失数据的情况下遗失分片。当有两个副本时，即使丢失了原始分片及其中一个副本，一切工作仍可以很好地持续下去。
- 更多副本意味着查询吞吐量将会增加，因为执行查询可以使用分片或分片的任一副本。

当然，Elasticsearch中分片和副本的数量之间还有其他关系，稍后将讨论。

那么，应该以什么标准来确定分片和副本的数量？这要视情况而定。我们相信，默认值确实不错，但一个好的测试无法取代。需要注意的是，副本的数量相对没那么重要，因为可以在生成索引后，在生产环境的集群中调整。只要你想，并且有足够的资源，就可以删除和添加副本。但对于分片来说，这样的操作就不可能了。一旦创建好索引，更改分片数量的唯一途径就是创建另一个索引并重新索引数据。

2.1.2 创建索引

在Elasticsearch中创建第一个文档时，没有关心索引的建立，只是使用了如下命令：

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

这是可以的。如果这样的索引不存在，Elasticsearch会为我们自动创建索引。还可以通过运行以下命令来创建索引：

```
curl -XPUT http://localhost:9200/blog/
```

我们只是告诉Elasticsearch需要创建名为`blog`的索引。顺利的话，你会从Elasticsearch看到如

下响应：

```
{"acknowledged":true}
```

什么时候需要手动创建索引？有许多情况，比如额外设置时，设置索引结构或分片数目。

1. 修改索引的自动创建

有时你会觉得，自动创建索引并非是一件好事。当你有一个大系统，需要很多流程来将数据输送到Elasticsearch时，索引名称的一个简单拼写错误可能会破坏掉几小时的脚本工作。你可以通过在elasticsearch.yml配置文件中添加以下指令来关闭自动创建索引：

```
action.auto_create_index: false
```

需要注意的是，action.auto_create_index比看起来要复杂。我们不但可以把它的值设置成false或ture， 也可以使用索引的名字模式来指定是否在具有给定名字的索引不存在时自动创建。例如在下面的例子中，允许自动创建以a开头的索引，但以an开头的索引则不允许。其他索引也必须手动创建（因为指令中的-*）。

```
action.auto_create_index: -an*,+a*,-*
```

注意，模式定义的顺序很重要。Elasticsearch检查这些模式直到第一种匹配的模式，所以，如果你将-an*移动到最后，它将不会被使用，因为指令中含有+a*，所以优先使用+a*。

2. 新创建索引的设定

想设置一些配置选项时，也需要手动创建索引，例如设置分片和副本的数量。来看看下面的例子：

```
curl -XPUT http://localhost:9200/blog/ -d '{
    "settings" : {
        "number_of_shards" : 1,
        "number_of_replicas" : 2
    }
}'
```

上面的命令将创建名为blog的索引，它将有1个分片和2个副本，即共得到3个物理Lucene索引。此外，也可以通过这种方式设置其他值，稍后讨论。

所以，我们已经创建了新的索引。但是有一个问题，我们忘了提供映射来描述索引结构。该怎么做？既然还没有任何数据，那就选择最简单的方法：删除索引。为此，将运行类似于上面的指令，然而不是用HTTP PUT方法，而是使用DELETE。实际指令如下所示：

```
curl -XDELETE http://localhost:9200/posts
```

响应将与我们之前看到的一样，如下所示：

```
{"acknowledged":true}
```

现在我们知道什么是索引，如何创建和删除，我们已经准备好用定义好的映射来创建索引。这部分非常重要，因为数据索引化将会影响搜索过程和文档匹配方式。

2.2 映射配置

如果你习惯用SQL数据库，或许会知道，在存入数据前需要创建模式以描述数据。尽管Elasticsearch是一个无模式的搜索引擎，可以即时算出数据结构，我们仍认为由自己控制并定义结构是更好的方法。在接下来的内容中，你将看到如何创建及删除新的索引，也将了解如何创建映射以满足需求和匹配你的数据结构。

注意，我们并不会在本章中阐述现有类型的全部信息。Elasticsearch的嵌套类型、主从关系处理、存储地理点以及搜索等特性，将在接下来的章节说明。

2.2.1 类型确定机制

在开始描述如何手动创建映射之前，我们想说明一件事。Elasticsearch可以通过定义文档的JSON来猜测文档结构。在JSON中，字符串用引号括起来，布尔值使用特定的词语定义，数值则是一些数字。这是一个简单的技巧，但通常有效。举例说明，请看以下文档：

```
{
  "field1": 10,
  "field2": "10"
}
```

上面的文档有两个字段。`field1`将被确定为数字（number，准确地说是`long`类型），但`field2`被确定为字符串，因为它用引号括起来。当然，这是我们所需要的行为，但有时数据源会省略掉数据类型的相关信息，一切都以字符串的形式呈现。解决方案是在映射定义文件中把`numeric_detection`属性设置为`true`，以开启更积极的文本检测。比如，可以在索引的创建过程中执行以下命令：

```
curl -XPUT http://localhost:9200/blog/?pretty -d '{
  "mappings" : {
    "article": {
      "numeric_detection" : true
    }
  }
}'
```

可惜，如果我们想要猜中布尔类型，问题仍然存在。我们不能从文本中强制推测出布尔类型。

在这种情况下，当无法改变源格式时，只能在映射定义中直接定义字段。

造成麻烦的另一个类型是基于日期类型的字段。Elasticsearch设法猜测被提供的时间戳或与日期格式匹配的字符串。可以使用dynamic_date_formats属性定义可被识别的日期格式列表，该属性允许指定一个格式的数组。看一下创建索引和类型的命令：

```
curl -XPUT 'http://localhost:9200/blog/' -d '{
  "mappings" : {
    "article" : {
      "dynamic_date_formats" : ["yyyy-MM-dd hh:mm"]
    }
  }
}'
```

上述命令可以创建名为blog的索引，其中包含一个名为article的类型。我们还使用单一日期格式的dynamic_date_formats属性，这样，对于与此格式匹配的字段，Elasticsearch将使用date核心类型（请参阅本章的“核心类型”部分了解更多关于字段类型的信息）。Elasticsearch使用joda-time库定义日期格式，所以如果有兴趣了解更多详情，请访问http://joda-time.sourceforge.net/api-release/org/joda/time/format/DateTimeFormat.html。

记住，dynamic_date_format属性可接受一个数组。这意味着，我们可以同时处理多种日期格式。

禁用字段类型猜测

想象以下情况。首先，索引一个数字，一个整数。Elasticsearch会猜测其类型，并设置类型为整数型（integer）或长整型（long）（请参阅本章的“核心类型”部分了解更多关于字段类型的信息）。如果索引另一个文档，它在同一个字段中存储的是浮点数，会发生什么？Elasticsearch将会删除小数部分并存储剩余整数。关闭它的另一个原因在于我们不希望在现有索引中添加新字段：那些在应用程序开发过程中未知的字段。

要关闭自动添加字段，可以把dynamic属性设置为false。把dynamic属性添加为类型的属性。例如，要在blog索引中为article类型关闭自动字段类型猜测，命令如下所示：

```
curl -XPUT 'http://localhost:9200/blog/' -d '{
  "mappings" : {
    "article" : {
      "dynamic" : "false",
      "properties" : {
        "id" : { "type" : "string" },
        "content" : { "type" : "string" },
        "author" : { "type" : "string" }
      }
    }
  }
}'
```

使用上面的命令创建blog索引后，在properties部分（下一节讨论）未提到的字段会被Elasticsearch忽略。如此，除id、content和author以外的任何字段都将被忽略。当然，这只发生在blog索引的article类型中。

2.2.2 索引结构映射

模式映射（schema mapping，或简称映射）用于定义索引结构。你可能还记得，每个索引可以有多种类型，但现在会为了简单起见我们只专注一个类型。假设想创建一个保存博客帖子数据的posts索引。它可以具有以下结构：

- 唯一标识符；
- 名称；
- 发布日期；
- 内容。

在Elasticsearch中，映射在文件中以JSON对象传送。所以，创建一个映射文件来匹配上述需求，称之为posts.json。其内容如下：

```
{
  "mappings": {
    "post": {
      "properties": {
        "id": {"type":"long", "store":"yes",
        "precision_step":"0" },
        "name": {"type":"string", "store":"yes",
        "index":"analyzed" },
        "published": {"type":"date", "store":"yes",
        "precision_step":"0" },
        "contents": {"type":"string", "store":"no",
        "index":"analyzed" }
      }
    }
  }
}
```

为使用上述文件创建posts索引，运行以下命令（假设我们的映射存储在posts.json文件中）：

```
curl -XPOST 'http://localhost:9200/posts' -d @posts.json
```

注意，你可以把映射存储成你想要的任何文件名。

同样，顺利的话，可以看到如下响应：

```
{"acknowledged":true}
```

现在，有了索引结构，可以索引我们的数据。休息一下，先来讨论posts.json文件的内容。

1. 类型定义

可以看到，posts.json文件内容是个JSON对象，因此它被大括号括起来（了解更多关于JSON的信息，请参看http://www.json.org/）。上述文件中的所有类型定义都嵌套在mappings对象中。你可以在mappings的JSON对象内定义多种类型。在在我们的例子中，只有单一的post类型。但如果还要包括user类型，文件将如下所示：

```
{
  "mappings": {
    "post": {
      "properties": {
        "id": { "type":"long", "store":"yes",
        "precision_step":"0" },
        "name": { "type":"string", "store":"yes",
        "index":"analyzed" },
        "published": { "type":"date", "store":"yes",
        "precision_step":"0" },
        "contents": { "type":"string", "store":"no",
        "index":"analyzed" }
      }
    },
    "user": {
      "properties": {
        "id": { "type":"long", "store":"yes",
        "precision_step":"0" },
        "name": { "type":"string", "store":"yes",
        "index":"analyzed" }
      }
    }
  }
}
```

2. 字段

每种类型由一组属性定义，也就是定义在properties对象中的字段。先集中在单个字段上，如contents字段，其完整定义如下所示：

```
"contents": { "type":"string", "store":"yes", "index":"analyzed" }
```

它由字段的名称开始，上述例子中是contents。名称后面，使用一个对象指定该字段的行为。我们所用字段的类型有特定的属性，下一节将讨论它们。当然，如果单一类型拥有多个字段（这是常见情况），记得用逗号将它们隔开。

3. 核心类型

每个字段类型可以指定为Elasticsearch提供的一个特定核心类型。Elasticsearch有以下核心类型。

- string：字符串；

- number：数字；
- date：日期；
- boolean：布尔型；
- binary：二进制。

现在来讨论Elasticsearch中可用的每个核心类型，以及它们用来定义行为的属性。

(1) 公共属性

在继续描述所有核心类型之前，先讨论一些可用来描述所有类型（二进制除外）的公共属性。

- index_name：该属性定义将存储在索引中的字段名称。若未定义，字段将以对象的名字来命名。
- index：可设置值为analyzed和no。另外，对基于字符串的字段，也可以设置为not_analyzed。如果设置为analyzed，该字段将被编入索引以供搜索。如果设置为no，将无法搜索该字段。默认值为analyzed。在基于字符串的字段中，还有一个额外的选项not_analyzed。此设置意味着字段将不经分析而编入索引，使用原始值被编入索引，在搜索的过程中必须全部匹配。索引属性设置为no将使include_in_all属性失效。
- store：这个属性的值可以是yes或no，指定了该字段的原始值是否被写入索引中。默认值设置为no，这意味着在结果中不能返回该字段（然而，如果你使用_source字段，即使没有存储也可返回这个值），但是如果该值编入索引，仍可以基于它来搜索数据。
- boost：该属性的默认值是1。基本上，它定义了在文档中该字段的重要性。boost的值越高，字段中值的重要性也越高。
- null_value：如果该字段并非索引文档的一部分，此属性指定应写入索引的值。默认的行为是忽略该字段。
- copy_to：此属性指定一个字段，字段的所有值都将复制到该指定字段。
- include_in_all：此属性指定该字段是否应包括在_all字段中。默认情况下，如果使用_all字段，所有字段都会包括在其中。2.4节将更详细地介绍_all字段。

(2) 字符串

字符串是最基本的文本类型，我们能够用它存储一个或多个字符。字符串字段的示例定义如下所示：

```
"contents" : { "type" : "string", "store" : "no", "index" :
  "analyzed" }
```

除了公共属性，基于字符串的字段还可以使用以下属性。

- term_vector：此属性的值可以设置为no（默认值）、yes、with_offsets、with_positions和with_positions_offsets。它定义是否要计算该字段的Lucene词向量（term vector）。如果你使用高亮，那就需要计算这个词向量。

- omit_norms：该属性可以设置为true或false。对于经过分析的字符串字段，默认值为false，而对于未经分析但已编入索引的字符串字段，默认值设置为true。当属性为true时，它会禁用Lucene对该字段的加权基准计算（norms calculation），这样就无法使用索引期间的加权，从而可以为只用于过滤器中的字段节省内存（在计算所述文件的得分时不会被考虑在内）。
- analyzer：该属性定义用于索引和搜索的分析器名称。它默认为全局定义的分析器名称。
- index_analyzer：该属性定义了用于建立索引的分析器名称。
- search_analyzer：该属性定义了的分析器，用于处理发送到特定字段的那部分查询字符串。
- norms.enabled：此属性指定是否为字段加载加权基准（norms）。默认情况下，为已分析字段设置为true（这意味着字段可加载加权基准），而未经分析字段则设置为false。
- norms.loading：该属性可设置eager和lazy。第一个属性值表示此字段总是载入加权基准。第二个属性值是指只在需要时才载入。
- position_offset_gap：此属性的默认值为0，它指定索引中在不同实例中具有相同名称的字段的差距。若想让基于位置的查询（如短语查询）只与一个字段实例相匹配，可将该属性值设为较高值。
- index_options：该属性定义了信息列表（postings list）的索引选项（2.2.4节将详细讨论）。可能的值是docs（仅对文档编号建立索引），freqs（对文档编号和词频建立索引），positions（对文档编号、词频和它们的位置建立索引），offsets（对文档编号、词频、它们的位置和偏移量建立索引）。对于经分析的字段，此属性的默认值是positions，对于未经分析的字段，默认值为docs。
- ignore_above：该属性定义字段中字符的最大值。当字段的长度高于指定值时，分析器会将其忽略。

(3) 数值

这一核心类型汇集了所有适用的数值字段类型。Elasticsearch中可使用以下类型（使用type属性指定）。

- byte：定义字节值，例如1。
- short：定义短整型值，例如12。
- integer：定义整型值，例如134。
- long：定义长整型值，例如123456789。
- float：定义浮点值，例如12.23。
- double：定义双精度值，例如123.45。

你可以在如下链接中了解更多所提到的Java类型：http://docs.oracle.com/avase/tutorial/java/nutsandbolts/datatypes.html。

数值类型字段的定义如下所示：

```
"price" : { "type" : "float", "store" : "yes", "precision_step" : "4" }
```

除了公共属性，以下属性也适用于数值字段。

- precision_step：此属性指定为某个字段中每个值生成的词条数。值越低，产生的词条数越高。对于每个值的词条数更高的字段，范围查询（range query）会更快，但索引会稍微大点，默认值为4。
- ignore_malformed：此属性值可以设为true或false。默认值是false。若要忽略格式错误的值，则应设置属性值为true。

(4) 布尔值

布尔值核心类型是专为索引布尔值（true或false）设计的。基于布尔值类型的字段定义如下所示：

```
"allowed" : { "type" : "boolean", "store": "yes" }
```

(5) 二进制

二进制字段是存储在索引中的二进制数据的Base64表示，可用来存储以二进制形式正常写入的数据，例如图像。基于此类型的字段在默认情况下只被存储，而不索引，因此只能提取，但无法对其执行搜索操作。二进制类型只支持index_name属性。基于binary字段的字段定义如下所示：

```
"image" : { "type" : "binary" }
```

(6) 日期

日期核心类型被设计用于日期的索引。它遵循一个特定的、可改变的格式，并默认使用UTC保存。

能被Elasticsearch理解的默认日期格式是相当普遍的，它允许指定日期，也可指定时间，例如，2012-12-24T12:10:22。基于日期类型的字段的示例定义如下所示：

```
"published" : { "type" : "date", "store" : "yes", "format" :
  "YYYY-mm-dd" }
```

使用上述字段的示例文档如下所示：

```
{
  "name" : "Sample document",
  "published" : "2012-12-22"
}
```

除了公共属性，日期类型的字段还可以设置以下属性。

- format：此属性指定日期的格式。默认值为dateOptionalTime。对于格式的完整列表，请访问http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/mapping-date-ormat.html。
- precision_step：此属性指定在该字段中的每个值生成的词条数。该值越低，产生的词条数越高，从而范围查询的速度越快（但索引大小增加）。默认值是4。
- ignore_malformed：此属性值可以设为true或false，默认值是false。若要忽略格式错误的值，则应设置属性值为true。

4. 多字段

有时候你希望两个字段中有相同的字段值，例如，一个字段用于搜索，一个字段用于排序；或一个经语言分析器分析，一个只基于空白字符来分析。Elasticsearch允许加入多字段对象来拓展字段定义，从而解决这个需求。它允许把几个核心类型映射到单个字段，并逐个分析。例如，想计算切面并在name字段中搜索，可以定义以下字段：

```
"name": {
  "type": "string",
  "fields": {
    "facet": { "type" : "string", "index": "not_analyzed" }
  }
}
```

上述定义将创建两个字段：我们将第一个字段称为name，第二个称为name.facet。当然，你不必在索引的过程中指定两个独立字段，指定一个name字段就足够了。Elasticsearch会处理余下的工作，将该字段的数值复制到多字段定义的所有字段。

5. IP地址类型

Elasticsearch添加了IP字段类型，以数字形式简化IPv4地址的使用。此字段类型可以帮搜索作为IP地址索引的数据、对这些数据排序，并使用IP值做范围查询。

基于IP地址类型的字段示例定义如下所示：

```
"address" : { "type" : "ip", "store" : "yes" }
```

除公共属性外，IP地址类型的字段还可以设置precision_step属性。该属性指定了字段中的每个值生成的词条数。值越低，词条数越高。对于每个值的词条数更高的字段，范围查询会更快，但索引会稍微大点，默认值为4。

使用上述字段的示例文档如下所示：

```
{
  "name" : "Tom PC",
  "address" : "192.168.2.123"
}
```

6. token_count类型

token_count字段类型允许存储有关索引的字数信息，而不是存储及检索该字段的文本。它接受与number类型相同的配置选项，此外，还可以通过analyzer属性来指定分析器。

基于token_count类型的字段示例定义如下：

```
"address_count" : { "type" : "token_count", "store" : "yes" }
```

7. 使用分析器

正如我们提到的那样，对于字符串类型的字段，可以指定Elasticsearch应该使用哪个分析器。回想第1章的内容，分析器是一个用于分析数据或以我们想要的方式查询数据的工具。例如，用空格和小写字符把单词隔开时，不必担心用户发送的单词是小写还是大写。Elasticsearch使我们能够在索引和查询时使用不同的分析器，并且可以在搜索过程的每个阶段选择处理数据的方式。使用分析器时，只需在指定字段的正确属性上设置它的名字，就这么简单。

(1) 开箱即用的分析器

Elasticsearch允许我们使用众多默认定义的分析器中的一种。如下分析器可以开箱即用。

- standard：方便大多数欧洲语言的标准分析器（关于参数的完整列表，请参阅http://www.lasticsearch.org/guide/en/elasticsearch/reference/current/analysis-standard-analyzer.html）。
- simple：这个分析器基于非字母字符来分离所提供的值，并将其转换为小写形式。
- whitespace：这个分析器基于空格字符来分离所提供的值。
- stop：这个分析器类似于simple分析器，但除了simple分析器的功能，它还能基于所提供的停用词（stop word）过滤数据（参数的完整列表，请参阅http://www.elasticsearch.rg/guide/en/elasticsearch/reference/current/analysis-stop-analyzer.html）。
- keyword：这是一个非常简单的分析器，只传入提供的值。你可以通过指定字段为not_analyzed来达到相同的目的。
- pattern：这个分析器通过使用正则表达式灵活地分离文本（参数的完整列表，请参阅http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/analysis-pattern-analyzer.html）。
- language：这个分析器旨在特定的语言环境下工作。该分析器所支持语言的完整列表可参考http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/analysis-lang-analyzer. tml。

- snowball：这个分析器类似于standard分析器，但提供了词干提取算法（stemming algorithm，参数的完整列表请参阅http://www.elasticsearch.org/guide/en/elasticsearch/eference/current/analysis-snowball-analyzer.html）。

词干提取（stemming）是还原屈折词和派生词至其基本形式的过程。这种方法缩减了字数，例如，对cars和car这两个单词，词干提取器（stemmer，词干提取算法的一种实现）产生一个词干car。索引完成后，在查询任一单词时，包含这些词的文档都会被匹配。如果不做词干提取，只有用cars查询时，包含“cars”的文档才会被匹配。

(2) 定义自己的分析器

除了前面提到的分析器，Elasticsearch还允许我们定义新的分析器，而无需编写Java代码。为此，需要在映射文件中加入settings节，它包含Elasticsearch创建索引时所需要的有用信息。下面演示了如何自定义settings节：

```
"settings" : {
  "index" : {
    "analysis": {
      "analyzer": {
        "en": {
          "tokenizer": "standard",
          "filter": [
            "asciifolding",
            "lowercase",
            "ourEnglishFilter"
          ]
        }
      },
      "filter": {
        "ourEnglishFilter": {
          "type": "kstem"
        }
      }
    }
  }
}
```

我们指定一个新的名为en的分析器。每个分析器由一个分词器和多个过滤器构成。默认过滤器和分词器的完整列表可以参阅http://www.elasticsearch.org/guide/en/elasticsearch/reference/urrent/analysis.html。我们的en分析器包括standard分词器和三个过滤器：默认情况下可用的asciifolding和lowercase，以及一个自定义的ourEnglishFilter。

要想定义过滤器，需要提供它的名称、类型以及该过滤器类型需要的任意数量的附加参数。Elasticsearch中可用过滤器类型的完整列表可以参考http://www.elasticsearch.org/guide/en/

elasticsearch/reference/current/analysis.html。此列表不断更新，所以这里不予讨论。

所以，定义了分析器的映射文件最终将如下所示：

```
{
  "settings" : {
    "index" : {
      "analysis": {
        "analyzer": {
          "en": {
            "tokenizer": "standard",
            "filter": [
              "asciifolding",
              "lowercase",
              "ourEnglishFilter"
            ]
          }
        },
        "filter": {
          "ourEnglishFilter": {
            "type": "kstem"
          }
        }
      }
    }
  },
  "mappings" : {
    "post" : {
      "properties" : {
        "id": { "type" : "long", "store" : "yes",
        "precision_step" : "0" },
        "name": { "type" : "string", "store" : "yes", "index" :
        "analyzed", "analyzer": "en" }
      }
    }
  }
}
```

可以通过Analyze API了解分析器的工作情况（http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/indices-analyze.html）。例如下面的命令：

```
curl -XGET 'localhost:9200/posts/_analyze?pretty&field=post.name' -d
'robots cars'
```

上面的命令要求Elasticsearch展示为post类型和它的name字段定义的分析器对指定短语（robots cars）的分析内容，得到的响应如下：

```
{
  "tokens" : [ {
    "token" : "robot",
    "start_offset" : 0,
    "end_offset" : 6,
```

```
      "type" : "<ALPHANUM>",
      "position" : 1
    }, {
      "token" : "car",
      "start_offset" : 7,
      "end_offset" : 11,
      "type" : "<ALPHANUM>",
      "position" : 2
    } ]
}
```

可以看到，短语robots cars被分离成两个词条。另外，单词robots更改成robot，cars更改成car。

(3) 分析器字段

可以通过分析器字段（_analyzer）指定一个字段，该字段的值将作为字段所属文档的分析器名称。试想一下，你有个软件检测写入文档的语言，并在文档的language字段存储相关信息。此外，你想使用这些信息来选择合适的分析器。为此，只需添加以下几行代码到你的映射文件：

```
"_analyzer" : {
  "path" : "language"
}
```

包含上述信息的映射文件如下所示：

```
{
  "mappings" : {
    "post" : {
      "_analyzer" : {
        "path" : "language"
      },
      "properties" : {
        "id": { "type" : "long", "store" : "yes",
        "precision_step" : "0" },
        "name": { "type" : "string", "store" : "yes",
        "index" : "analyzed" },
        "language": { "type" : "string", "store" : "yes",
        "index" : "not_analyzed"}
      }
    }
  }
}
```

需要注意的是，应该定义一个与language字段中提供的值一样的分析器，否则索引将会失败。

(4) 默认分析器

关于分析器，还有一点需要说明，即在没有定义分析器的情况下，应指定在默认情况下使用的分析器。这与在映射文件中的setting部分配置自定义分析器的方式相同，但应使用default关键字来命名，而不是为分析器指定一个自定义名称。因此，为了把前面定义的分析器作为默认分析器，可以将en分析器修改为下面这样：

```
{
  "settings" : {
    "index" : {
      "analysis": {
        "analyzer": {
          "default": {
            "tokenizer": "standard",
            "filter": [
              "asciifolding",
              "lowercase",
              "ourEnglishFilter"
            ]
          }
        },
        "filter": {
          "ourEnglishFilter": {
            "type": "kstem"
          }
        }
      }
    }
  }
}
```


2.2.3 不同的相似度模型

2012年发布Apache Lucene 4.0后，该全文检索库的所有用户便可以修改默认的基于TF/IDF的算法（第1章中提到过）。但这不是唯一的变化，Lucene 4.0还附加了相似度模型（similarity model），允许在文档中使用不同的评分公式。

1. 设定每个字段的相似度模型

自Elasticsearch 0.90版本开始，就可以为映射文件中的每个字段设置不同的相似度模型。例如，使用如下简单的映射来索引`blog posts`：

```
{
  "mappings" : {
    "post" : {
      "properties" : {
        "id" : { "type" : "long", "store" : "yes",
          "precision_step" : "0" },
        "name" : { "type" : "string", "store" : "yes", "index" :
          "analyzed" },
        "contents" : { "type" : "string", "store" : "no",
          "index" : "analyzed" }
      }
    }
  }
}
```

为此，可在name字段和contents字段中使用BM25相似度模型。这需要扩展我们的字段定义，并添加similarity属性以及所选择的similarity名称的值。修改映射如下：

```
{
  "mappings" : {
    "post" : {
      "properties" : {
        "id" : { "type" : "long", "store" : "yes",
          "precision_step" : "0" },
        "name" : { "type" : "string", "store" : "yes",
          "index" : "analyzed", "similarity" : "BM25" },
        "contents" : { "type" : "string", "store" : "no",
          "index" : "analyzed", "similarity" : "BM25" }
      }
    }
  }
}
```

仅此而已，无需再添加别的。做出上述修改后，Lucene Apache将为字段name和contents使用BM25相似度模型计算得分因子。

2. 可用的相似度模型

现有如下三种相似度模型。

- Okapi BM25模型：这种相似度模型基于概率模型，概率模型估算根据指定查询找到指定文档的概率。为了在Elasticsearch中使用这种相似度模型，需要将BM25作为名称。据说Okapi BM25相似度模型在处理简短的文本文档时表现最佳，在这种文档中词条的重复严重有损整体文档的得分。要使用此相似度模型，需要设置字段的similarity属性为BM25。这种相似度模型在定义后立即可用，不需要设置附加属性。
- 随机性偏差（divergence from randomness）模型：这种相似度模型基于具有相同名称的概率模型。为了在Elasticsearch中使用这种相似度模型，需要使用DFR作为名称。随机性偏差模型在处理类自然语言文本时表现良好。
- 信息基础（information-based）模型：这是新推出的最后一个相似度模型，与随机性偏差模型非常相似。为了在Elasticsearch中使用这种相似度模型，需要使用IB作为名称。类似于DFR相似度模型，信息基础模型在处理类自然语言文本时表现良好。

(1) 配置DFR相似度模型

在DFR相似度模型中，可以配置basic_model属性（可设置为be、d、g、if、in或者ine），after_effect属性（可设置为no、b或l），normalization属性（可设置为no、h1、h2、h3或z）。如果选择no以外的值作为normalization属性值，需要设置范式化因子（normalization factor）。所选择的normalization值如果是h1，则设置normalization.h1.c（浮点值）；如果是h2，则设置normalization.h2.c（浮点值）；如果是h3，则设置normalization.h3.c

（浮点值）；如果是z，则设置normalization.z.z（浮点值）。例如，下面是相似度配置的一个示例（把它放到映射文件的settings部分中）：

```
"similarity" : {
  "esserverbook_dfr_similarity" : {
    "type" : "DFR",
    "basic_model" : "g",
    "after_effect" : "l",
    "normalization" : "h2",
    "normalization.h2.c" : "2.0"
  }
}
```

(2) 配置IB相似度模型

在IB相似度模型中，有以下参数可供配置：distribution属性（可设置为ll或spl）和lambda属性（可设置为df或tff）。此外，跟DFR相似度模型一样，可以选择范式化因子，这里不再赘述。下面是一个配置IB相似度模型的例子（把它放到配置文件的settings部分）：

```
"similarity" : {
  "esserverbook_ib_similarity" : {
    "type" : "IB",
    "distribution" : "ll",
    "lambda" : "df",
    "normalization" : "z",
    "normalization.z.z" : "0.25"
  }
}
```

相似度模型是一个相当复杂的话题，需要一整章来正确地描述它。如果你有兴趣，请参阅*Mastering ElasticSearch*，或到http://elasticsearchserverbook.com/elasticsearch-0-90-similarities/阅读更多细节。

2.2.4 信息格式

Apache Lucene 4.0的显著变化之一是，可以改变索引文件写入的方式。Elasticsearch利用了此功能，可以为每个字段指定信息格式。有时你需要改变字段被索引的方式以提高性能，比如为了使主键查找更快。

Elasticsearch中的信息格式如下所示。

- default：没有明确定义格式时，此默认信息格式将被使用。它提供了实时的对存储字段和词向量的压缩。如果你想知道压缩的内容，请参阅http://solr.pl/en/2012/11/19/solr-4-1-stored-fields-compression/。

- pulsing：此信息格式将高基数字段（high cardinality field）[1]的信息列表编码为词条矩阵，这让Lucene检索文档时可以少执行一个搜索。对高基数字段使用此信息格式可以加快此字段的查询速度。
- direct：此信息格式可在读操作过程中将词条加载到矩阵中。这些矩阵未经压缩保存在内存中。这种信息格式可以提升常用字段的性能，但是使用时应谨慎，因为它属于内存密集型，词条和信息矩阵都存储于内存中。请记住，因为所有的词条都存储在比特组里，所以每个段需要多达2.1 GB的内存。
- memory：此信息格式正如它的名字所示，将所有的数据写入磁盘，但需要用到一个名为FST（Finite State Transducer，有限状态传感器）的结构读取词条和信息列表到内存中。你可以在http://blog.mikemccandless.com/2010/12/using-finite-state-transducers-in.html上Mike McCandless的帖子中了解更多关于这个结构的知识。由于数据存储于内存中，这个信息格式会提升常用词条的性能。
- bloom_default：默认信息格式的扩展，增加了把布隆过滤器（bloom filter）写入磁盘的功能。在读取过程中，布隆过滤器被读取并存入内存，以便非常快速地检查给定的值是否存在。该信息格式对于高基数字段相当有效，比如主键字段。如果你想知道更多关于布隆过滤器的知识，请参阅http://en.wikipedia.org/wiki/Bloom_filter。此信息格式除了默认格式的功能，还使用了布隆过滤器。
- bloom_pulsing：这是pulsing信息格式的扩展，除了pulsing格式的功能，还使用了布隆过滤器。

配置信息格式

信息格式可在每个字段上设置，就像type或name。为了把字段配置成默认格式以外的格式，需要添加一个名为postings_format的属性，将所选择信息格式的名字作为值。如果想在id字段使用pulsing信息格式，映射将如下所示：

```
{
  "mappings" : {
    "post" : {
      "properties" : {
        "id" : { "type" : "long", "store" : "yes",
          "precision_step" : "0", "postings_format" : "pulsing" },
        "name" : { "type" : "string", "store" : "yes",
          "index" : "analyzed" },
        "contents" : { "type" : "string", "store" : "no",
          "index" : "analyzed" }
      }
    }
  }
}
```

① 基数越高，字段的重复值就越少，可选性越高。——译者注

2.2.5 文档值

文档值将是本节讨论的最后一个字段属性。文档值格式是在Lucene 4.0中引入的另一个新功能。它允许定义一个给定字段的值被写入一个具有较高内存效率的列式结构，以便进行高效的排序和切面搜索。使用了文档值的字段将有专属的字段数据缓存实例，无需像标准字段一样倒排（以避免像第1章所描述的方法一样存储）。因此，它使索引刷新操作速度更快，让你可以在磁盘上存储字段数据，从而节省堆内存的使用。

1. 配置文档值

我们扩展一下posts索引示例，增加一个votes字段。假设新添加的字段包含指定文章的得票数量，并且我们希望对它排序。因为需要排序，所以它很适合使用文档值。为了在给定字段上使用文档值，需要将doc_values_format属性添加到其定义中并指定其格式。映射将如下所示：

```
{
  "mappings" : {
    "post" : {
      "properties" : {
        "id" : { "type" : "long", "store" : "yes",
          "precision_step" : "0" },
        "name" : { "type" : "string", "store" : "yes",
          "index" : "analyzed" },
        "contents" : { "type" : "string", "store" : "no",
          "index" : "analyzed" },
        "votes" : { "type" : "integer",
          "doc_values_format" : "memory" }
      }
    }
  }
}
```

如你所见，定义非常简单。来看看对于doc_values_format属性，有哪些可以设置的值。

2. 文档值格式

目前有以下三种可用的doc_values_format属性值。

- default：当未指定任何格式时，使用此默认格式。此格式使用少量内存而且性能良好。
- disk：此文档值格式将数据存入磁盘，几乎无需内存。然而，在使用这种数据结构执行切面和排序等操作时，性能略有降低。需要执行切面或排序操作，而又苦恼于内存空间时，可使用这种格式。
- memory：此文档值格式将数据存入内存。这种格式中，切面或排序的功能与标准倒排索引字段的功能不相上下。由于这种数据结构存储于内存中，索引的刷新速度更快，而这对快速更改索引及缩短索引更新频率很有帮助。

2.3　批量索引以提高索引速度

在第1章中，我们学习了如何将特定文档添加索引至Elasticsearch。现在来学习如何以更方便有效的方式索引多个文档，而不是逐个索引。

2.3.1　为批量索引准备数据

Elasticsearch可以合并多个请求至单个包中，而这些包可以单个请求的形式传送。如此，可以将如下操作结合起来：

- 在索引中增加或更换现有文档（index）；
- 从索引中移除文档（delete）；
- 当索引中不存在其他文档定义时，在索引中增加新文档（create）。

为了获得较高的处理效率，选择这样的请求格式。它假定，请求的每一行包含描述操作说明的JSON对象，第二行为JSON对象本身。可以把第一行视为信息行，第二类为数据行。唯一的例外是delete操作，它只包含信息行。来看看下面的例子：

```
{ "index": { "_index": "addr", "_type": "contact", "_id": 1 }}
{ "name": "Fyodor Dostoevsky", "country": "RU" }
{ "create": { "_index": "addr", "_type": "contact", "_id": 2 }}
{ "name": "Erich Maria Remarque", "country": "DE" }
{ "create": { "_index": "addr", "_type": "contact", "_id": 2 }}
{ "name": "Joseph Heller", "country": "US" }
{ "delete": { "_index": "addr", "_type": "contact", "_id": 4 }}
{ "delete": { "_index": "addr", "_type": "contact", "_id": 1 }}
```

重要的是，每一个文档或操作说明放置在一行中（以换行符结束）。这意味着无法美化文档格式。批量索引文件的大小存在限制，它被设定为100 MB，在Elasticsearch配置文件中可以通过http.max_content_length属性来改变。这避免了请求过大时可能存在的请求超时及内存问题。

需要注意的是，在一个批量索引文档中，可以将数据载入多个索引中，文档也可以有不同的类型。

2.3.2　索引数据

为了执行批量请求，Elasticsearch提供了_bulk端点，形式可以是/_bulk，也可以是/index_name/_bulk，甚至是/index_name/type_name/_bulk。第二种和第三种形式定义了索引名称和类型名称的默认值。可以在请求的信息行中省略这些属性，Elasticsearch将使用默认值。

假设已经在documents.json文件中存储了数据，可以运行下面的命令将这些数据发送到Elasticsearch：

```
curl -XPOST 'localhost:9200/_bulk?pretty' --data-binary
  @documents.json
```

没必要设置?pretty参数。之前使用这个参数仅仅是为了方便分析命令的响应。在这个例子中，我们在curl中使用--data-binary参数，而不是使用-d，这很重要。这是因为标准的-d参数忽略换行符，正如前面所说的那样，换行符在解析Elasticsearch的批量请求内容时很重要。现在，来看看由Elasticsearch返回的响应：


```
{
  "took" : 139,
  "errors" : true,
  "items" : [ {
    "index" : {
      "_index" : "addr",
      "_type" : "contact",
      "_id" : "1",
      "_version" : 1,
      "status" : 201
    }
  }, {
    "create" : {
      "_index" : "addr",
      "_type" : "contact",
      "_id" : "2",
      "_version" : 1,
      "status" : 201
    }
  }, {
    "create" : {
      "_index" : "addr",
      "_type" : "contact",
      "_id" : "2",
      "status" : 409,
      "error" : "DocumentAlreadyExistsException[[addr][3]
         [contact][2]: document already exists]"
    }
  }, {
    "delete" : {
      "_index" : "addr",
      "_type" : "contact",
      "_id" : "4",
      "_version" : 1,
      "status" : 404,
      "found" : false
    }
  }, {
    "delete" : {
      "_index" : "addr",
      "_type" : "contact",
      "_id" : "1",
      "_version" : 2,
      "status" : 200,
```

```
        "found" : true
      }
    }]
}
```

正如我们看到的，每一个结果是items数组的一部分。简要对比这些结果和输入的数据：前两个命令index和create执行得很顺利，而第三个操作失败了，因为想用一个已经存在于索引中的标识符来创建一条记录。接下来的两个删除操作都成功了。注意，其中第一个操作试图删除一个不存在的文档。可以看到，这对Elasticsearch来说不是问题。Elasticsearch返回有关每个操作的信息，因此对于大批量请求，响应也是巨大的。

2.3.3　更快的批量请求

批量操作的速度很快，但如果你想知道是否存在更有效、更快的索引方法，可以看看*用户数据报协议*（User Datagram Protocol，UDP）的批量操作。请注意，使用UDP并不能保证数据在与Elasticsearch服务器通信的过程中不会丢失。因此，只有在性能至关重要且比精确度还重要，并且要索引全部文档时，才考虑使用它。

2.4　用附加的内部信息扩展索引结构

除了保存数据的字段之外，还可以与文档一道存储附加的信息。本书已经讨论过各种不同的映射选项及可用的数据类型。现在我们打算更详细地讨论一些并非每天都使用到的Elasticsearch功能，这些功能可以更方便地处理数据。

以下讨论的字段类型应定义在适当的类型级别上，它们不是索引范围的类型。

2.4.1　标识符字段

你可能还记得，Elasticsearch索引的每个文档都有自己的标识符和类型。在Elasticsearch中，文档存在两种内部标识符。

第一个是_uid字段，它是索引中文档的唯一标识符，由该文档的标识符和文档类型构成。这基本意味着，不同类型的文档编入到相同的索引时可以具有相同的文档标识符，而Elasticsearch仍能够区分它们。此字段不需要任何额外的设置，它总是被索引，但知道它的存在是有好处的。

持有标识符的第二个字段是_id字段。此字段存储着索引时设置的实际标识符。为了使_id字段能够被索引（或存储，如果需要的话），需要在映射文件中像设置任何其他属性一样添加_id字段的定义（但是，如前面所说，应添加到类型定义的主体中）。所以，book类型的示例定义如下：

```
{
  "book" : {
    "_id" : {
      "index": "not_analyzed",
      "store" : "no"
    },
    "properties" : {
      .
      .
      .
    }
  }
}
```

可以看到，上述例子在代码中指明希望_id字段不经分析但要编入索引，而且不希望存储。

除了在索引时间指定标识符，也可从指定我们希望从索引文档的一个字段中获取标识符（由于需要额外的解析，速度会稍慢些）。为此，需要设定path属性，并将它的值设置为一个字段名称，该字段的值将作为标识符。如果我们的索引中含有book_id字段，并打算使用它作为_id字段的值，将上述映射文件的内容更改为如下内容：

```
{
  "book" : {
    "_id" : {
      "path": "book_id"
    },
    "properties" : {
      .
      .
      .
    }
  }
}
```

最后需要记得一件事：当禁用_id字段时，所有需要文档唯一标识符的功能都能继续工作，因为它们将用_uid字段作为代替。

2.4.2 _type字段

之前说过，在Elasticsearch中每个文档至少需要由它的标识符和类型来描述。默认情况下，文档的类型会编入索引，但不会被分析并且不存储。如果想存储这个字段，可按如下方式修改映射文件内容：

```
{
  "book" : {
    "_type" : {
      "store" : "yes"
    },
    "properties" : {
```

```
        .
        .
        .
      }
    }
}
```

也可改成不索引_type字段，但是那样的话，一些查询（例如词条查询）和过滤器将失效。

2.4.3 _all字段

Elasticsearch使用_all字段存储其他字段中的数据以便于搜索。当要执行简单的搜索功能，搜索所有数据（或复制到_all字段的所有字段），但又不想去考虑字段名称之类的事情时，这个字段很有用。默认情况下，_all字段是启用的，包含了索引中所有字段的所有数据。然而，这一字段使得索引有点大，且这并不总是必要的。可以完全禁用_all字段，或排除某些字段。为了在_all字段不包括某个特定字段，我们将使用本章前面讨论的include_in_all属性。要完全关闭_all字段功能，可修改映射文件，如下所示：

```
{
  "book" : {
    "_all" : {
      "enabled" : "false"
    },
    "properties" : {
      .
      .
      .
    }
  }
}
```

除了enabled属性，_all字段还支持以下属性：

- store
- term_vector
- analyzer
- index_analyzer
- search_analyzer

有关上述属性的信息，请参阅2.2节。

2.4.4 _source字段

_source字段可以在生成索引过程中存储发送到Elasticsearch的原始JSON文档。默认情况下，_source字段会被开启，因为部分Elasticsearch功能依赖于这个字段（如局部更新功能）。除此之

外，当某字段没有存储时，_source字段可用作高亮功能的数据源。但如果不需要这样的功能，可禁用_source字段避免存储开销。为此，需设置_source对象的enabled属性值为false，如下所示：

```
{
  "book" : {
    "_source" : {
      "enabled" : false
    },
    "properties" : {
      .
      .
      .
    }
  }
}
```

排除字段和包含字段

还可以告诉Elasticsearch我们希望从_source字段中排除哪些字段，包含哪些字段。可以通过在_source字段定义中添加includes或excludes属性来实现。如果想从_source中排除author路径下的所有字段，映射将如下所示：

```
{
  "book" : {
    "_source" : {
      "excludes" : [ "author.*" ]
    },
    "properties" : {
      .
      .
      .
    }
  }
}
```

2.4.5 _index字段

Elasticsearch允许我们存储文档相关索引的信息。可以通过使用内部_index字段做到这一点。试想一下，我们每天创建索引，对索引使用别名，且想知道返回文档存储在哪个索引中。在这种情况下，_index字段可以帮助确定文档源自哪个索引。

默认情况下，_index字段是禁用的。为了启用它，需要设置_index对象的enabled属性为true，如下所示：

```
{
  "book" : {
    "_index" : {
```

```
        "enabled" : true
      },
      "properties" : {
        .
        .
        .
      }
    }
}
```

2.4.6 _size字段

默认情况下，_size字段未启用，这使我们能够自动索引_source字段的原始大小，并与文件一道存储。如果需要启用_size字段，则需添加_size属性并设置enabled属性值为true。此外，还可以通过使用通常的store属性设置_size字段使其被存储。因而，如果希望我们的映射包括_size字段并被存储，需要把映射文件修改成下面这样：

```
{
  "book" : {
    "_size" : {
      "enabled": true,
      "store" : "yes"
    },
    "properties" : {
      .
      .
      .
    }
  }
}
```

2.4.7 _timestamp字段

默认情况下，禁用的_timestamp字段允许文档在被索引时存储。启用此功能很简单，只要添加_timestamp到映射文件并将enabled属性设置为true，如下所示：

```
{
  "book" : {
    "_timestamp" : {
      "enabled" : true
    },
    "properties" : {
      .
      .
      .
    }
  }
}
```

默认情况下，_timestamp字段未经分析编入索引，但不保存。你可以改变这两个参数以满足实际需求。除此之外，_timestamp字段与普通的日期字段一样，因而可以像处理寻常的、基于日期的字段一样改变它的格式。为此，只需要用所需的格式指定format属性（请参阅本章前面关于“日期”核心类型的描述，以了解更多关于日期格式的内容）。

另外，可以添加path属性，并将其设置为某字段的名称来获取日期，而不是在文件检索过程中自动创建_timestamp字段。因此，若希望_timestamp字段基于year字段，可修改映射文件，如下所示：

```
{
  "book" : {
    "_timestamp" : {
      "enabled" : true,
      "path" : "year",
      "format" : "YYYY"
    },
    "properties" : {
      .
      .
      .
    }
  }
}
```

你可能已经注意到，我们还修改了_timestamp字段的格式以匹配存储在year字段中的值。

如果你使用_timestamp字段，并让Elasticsearch自动创建它，则该字段的值会被设置为文档索引的时间。请注意，使用局部文档更新功能时，_timestamp字段也将被更新。

2.4.8 _ttl字段

_ttl字段表示time to live（生存时间），它允许定义文档的生命周期，周期结束之后文档会被自动删除。默认情况下，_ttl字段是禁用的。要启用，需要添加_ttl JSON对象并设置它的enabled属性为true，参考下面的例子：

```
{
  "book" : {
    "_ttl" : {
      "enabled" : true
    },
    "properties" : {
      .
      .
      .
    }
  }
}
```

如果要提供文件的默认过期时间，只需在_ttl字段定义中添加default属性和期望的过期时间。例如，要在30天后删除文件，将做以下设置：

```
{
  "book" : {
    "_ttl" : {
      "enabled" : true,
      "default" : "30d"
    },
    "properties" : {
      .
      .
      .
    }
  }
}
```

默认情况下，该_ttl值未经分析即存储和索引。你可以改变这两个参数，但要记住这个字段要未经分析才能工作。

2.5　段合并介绍

1.1节提到段及其不变性，指出Lucene库以及Elasticsearch中一旦数据被写入某些结构，就不再改变。虽然这简化了一些东西，但是也引入了额外的工作，其中一个例子是删除。由于段是无法改变的，因而有关删除的相关信息必须单独存储并动态应用到搜索过程中。这样做是为了从返回结果中去除已删除的文件。另一个例子是文档无法修改（有些修改是可能的，例如修改数值型doc值）。当然，我们可以说，Elasticsearch支持文档更新（请参阅1.4节）。然而在底层，实际上是删除旧文档，再把更新内容的文档编入索引。

随着时间的推移和持续索引数据，越来越多的段被创建。因此，搜索性能可能会降低，而且索引可能比原先大，因为它仍含有被删除的文件。这使得段合并有了用武之地。

2.5.1　段合并

段合并的处理过程是：底层的Lucene库获取若干段，并在这些段信息的基础上创建一个新的段。由此产生的段拥有所有存储在原始段中的文档，除了被标记为删除的那些之外。合并操作之后，源段将从磁盘上删除。这是因为段合并在CPU和I/O的使用方面代价是相当高的，关键是要适当地控制这个过程被调用的时机和频率。

2.5.2　段合并的必要性

你可能会问为何要费心段合并。首先，构成索引的段越多，搜索速度越慢，需要使用的Lucene

内存也越多。其次，索引使用的磁盘空间和资源，例如文件描述符。如果从索引中删除许多文档，直到合并发生，则这些文档只是被标记为已删除，而没有在物理上删除。因而，大多数占用了CPU和内存的文档可能并不存在！好在Elasticsearch使用合理的默认值做段合并，这些默认值很可能不再需要做任何更改。

2.5.3 合并策略

合并策略描述了应执行合并过程的时机。Elasticsearch允许配置以下三种不同的策略。

- `tiered`：这是默认合并策略，合并尺寸大致相似的段，并考虑到每个层（tier）允许的最大段数量；
- `log_byte_size`：这个合并策略下，随着时间推移，将产生由索引大小的对数构成的索引，其中存在着一些较大的段以及一些合并因子较小的段等；
- `log_doc`：这个策略类似于`log_byte_size`合并策略，但根据索引中的文档数而非段的实际字节数来操作。

上述的每个策略都有自己的参数来定义行为以及可以覆盖的默认值。本书不做详细描述。如果你想了解更多，请查看*Mastering ElasticSearch*，或去http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/index-modules-merge.html了解详情。

可使用`index.merge.policy.type`属性来设置想使用的合并策略，如下所示：

```
index.merge.policy.type: tiered
```

值得一提的是，索引创建后将无法再对值进行修改。

2.5.4 合并调度器

合并调度器指示Elasticsearch合并过程的方式，有如下两种可能。

- *并发合并调度器*：这是默认的合并过程，在独立的线程中执行，定义好的线程数量可以并行合并。
- *串行合并调度器*：这一合并过程在调用线程（即执行索引的线程）中执行。合并进程会一直阻塞线程直到合并完成。

调度器可使用`index.merge.scheduler.type`参数设置。若要使用串行合并调度器，需把参数值设为`serial`；若要使用并发调度器，则需把参数值设为`concurrent`。例如下面的调度程序：

```
index.merge.scheduler.type: concurrent
```

2.5.5 合并因子

每种合并策略都有好几种设置，我们已经说过在这里不一一介绍，但是合并因子是个例外，

它指定了索引过程中段合并的频率。合并因子较小时，搜索的速度更快，占用的内存也更少，但索引的速度会减慢；合并因子较大时，则索引速度加快，这是因为发生的合并较少，但搜索的速度变慢，占用的内存也会变大。对于log_byte_size和log_doc合并策略，可以通过index.merge.policy.merge_factor参数来设置合并因子。

```
index.merge.policy.merge_factor: 10
```

上述例子将合并因子的值设置成10，10也是默认值。建议在批量索引时设置更高的merge_factor属性值，普通的索引维护则设置较低的属性值。

2.5.6　调节

之前提到过，合并可能需要很多的服务器资源。合并过程通常与其他操作并行执行，所以理论上不会产生太大的影响。在实践中，磁盘I/O操作的数量可能非常大，以致严重影响了整体性能。这时，调节（throttling）可以改善此情况。事实上，此功能既可用于限制合并的速度，也可以用于使用数据存储的所有操作。可以在Elasticsearch的配置文件中对调节进行设置（elasticsearch.yml文件），也可以动态使用设置API来设置（请参阅8.7.2节）。调整调节的设置有两个：type和value。

为了设置调节类型，设置indices.store.throttle.type属性值为下列值之一。

- none：该值定义不打开调节。
- merge：该值定义调节仅在合并过程中有效。
- all：该值定义调节在所有数据存储活动中有效。

第二个属性，indices.store.throttle.max_bytes_per_sec，描述了调节限制I/O操作的数量。顾名思义，该属性告诉我们每秒可以处理的字节数量。来看看以下配置：

```
indices.store.throttle.type: merge
indices.store.throttle.max_bytes_per_sec: 10mb
```

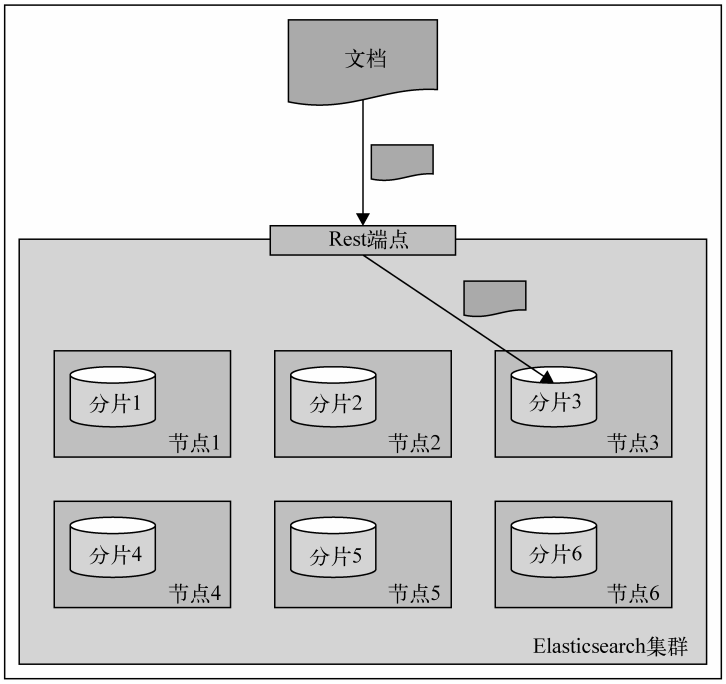
这个例子中，我们限制合并操作为每秒10 MB。默认情况下，Elasticsearch使用merge调节类型，max_bytes_per_sec属性设置值为20 mb。这意味着所有的合并操作都限于每秒20 MB。

2.6　路由介绍

默认情况下，Elasticsearch会在所有索引的分片中均匀地分配文档。然而，这并不总是理想情况。为了获得文档，Elasticsearch必须查询所有分片并合并结果。然而，如果你可以把数据按照一定的依据来划分（例如，客户端标识符），就可以使用一个强大的文档和查询分布控制机制：路由。简而言之，它允许选择用于索引和搜索数据的分片。

2.6.1 默认索引过程

在创建索引的过程中，当你发送文档时，Elasticsearch会根据文档的标识符，选择文档应编入索引的分片。默认情况下，Elasticsearch计算文档标识符的散列值，以此为基础将文档放置于一个可用的主分片上。接着，这些文档被重新分配至副本。下面的流程图简单演示了索引在默认情况下是如何工作的：



2.6.2 默认搜索过程

搜索与索引略有不同，在多数情况下，为了得到感兴趣的数据，你需要查询所有分片。试想一下，使用如下映射描述你的索引：

```
{
  "mappings" : {
    "post" : {
      "properties" : {
        "id" : { "type" : "long", "store" : "yes",
        "precision_step" : "0" },
        "name" : { "type" : "string", "store" : "yes",
        "index" : "analyzed" },
        "contents" : { "type" : "string", "store" : "no",
        "index" : "analyzed" },
        "userId" : { "type" : "long", "store" : "yes",
        "precision_step" : "0" }
```

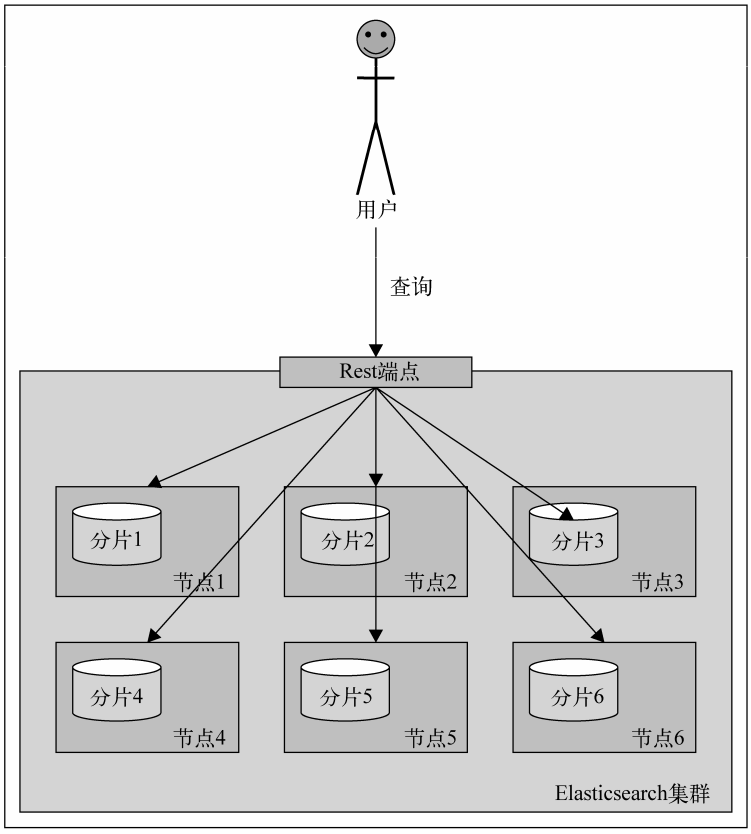
```
        }
      }
    }
  }
```

可以看到，索引包含了4个字段：标识符（id字段）、文档名称（name字段）、文档内容（contents字段）及文档所属用户的标识符（userId字段）。为了得到特定用户的所有文档（userId值为12），可以运行如下命令：

```
curl -XGET 'http://localhost:9200/posts/_search?q=userId:12'
```

一般而言，我们将查询发送到Elasticsearch的一个节点，Elasticsearch将会根据搜索类型（将在第3章讨论）来执行查询。这通常意味着它首先查询所有节点得到标识符和匹配文档的得分，接着发送一个内部查询，但仅发送到相关的分片（包含所需文档的分片），最后获取所需文档来构建响应。

以下视图简单演示了在搜索过程中默认路由的工作方式：



假使把单个用户的所有文档放置于单个分片之中，并对此分片查询，会出现什么情况？是否

对性能来说不明智？不，这种操作是相当便利的，也正是路由所允许的。

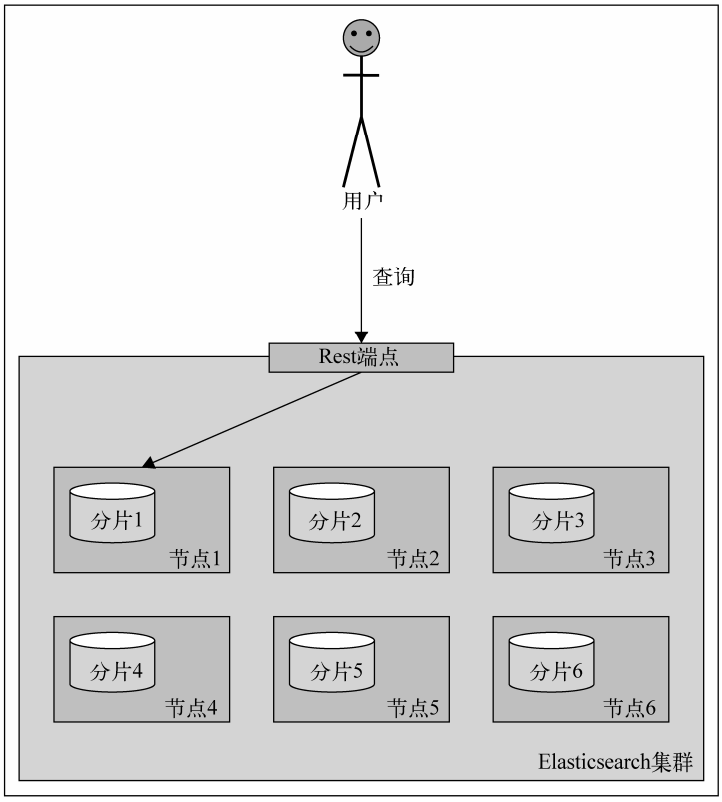
2.6.3 路由

路由可以控制文档和查询转发的目的分片。现在，你可能已经猜到了，可以在索引和查询时都指定路由值。事实上，如果你决定指定明确的路由值，可能会在索引和搜索过程中都这样做。

在我们的例子中，索引时使用`userId`值来设置路由，在搜索时也一样。你可以想象，对于相同的`userId`值，计算出的散列值是相同的，因而特定用户的所有文档将被放置在相同的分片中。在搜索中使用相同的属性值，则只需搜索单个分片而不是整个索引。

要记住，使用路由时，你仍然应该为与路由值相同的值添加一个过滤器。这是因为，路由值的数量或许会比索引分片的数量多。因此，一些不同的属性值可以指向相同的分片，如果你忽略过滤，得到的数据并非是路由的单个值，而是特定分片中驻留的所有路由值。

下图简单展示了给定路由值时，搜索是如何工作的：



如你所见，Elasticsearch将把查询发送到单个分片上。现在来看看如何指定路由值。

2.6.4　路由参数

最简单的方法（但并不总是最方便的一个）是使用路由参数来提供路由值。索引或查询时，你可以添加路由参数到HTTP，或使用你所选择的客户端库来设置。

所以，为了在前面所示的索引中建立一个示例文档，使用下列命令：

```
curl -XPUT 'http://localhost:9200/posts/post/1?routing=12' -d '{
  "id": "1",
  "name": "Test document",
  "contents": "Test document",
  "userId": "12"
}'
```

下面是使用路由参数的查询：

```
curl -XGET
  'http://localhost:9200/posts/_search?routing=12&q=userId:12'
```

可以看到，索引和查询时我们使用相同的路由值。这么做是因为我们知道在索引时设置的属性值为12，我们想要查询指向同一分片，因此需要使用完全相同的值。

请注意，你可以指定多个路由值，并由逗号分隔开来。如果还想在前面的查询中使用section参数（如果存在的话）来路由，并根据这个参数过滤，查询将会如下所示：

```
curl -XGET
  'http://localhost:9200/posts/_search?routing=12,
  6654&q=userId:12+AND+section:6654'
```

记住，路由值不是为了获取特定用户结果而唯一要指定的值。这是因为通常情况下分片很少有唯一的路由值。这意味着我们在单个分片中会有来自多个用户的数据。所以使用路由时，你还应该过滤结果。3.5节会介绍更多关于过滤的知识。

2.6.5　路由字段

为每个发送到Elasticsearch的请求指定路由值并不方便。事实上，在索引过程中，Elasticsearch允许指定一个字段，用该字段的值作为路由值。这样只需要在查询时提供路由参数。为此，在类型定义中需要添加以下代码：

```
"_routing" : {
  "required" : true,
  "path" : "userId"
}
```

上述定义意味着需要提供路由值（"required": true属性），否则，索引请求将失败。除此之外，我们还指定了path属性，说明文档的哪个字段值应被设置为路由值，在上述示例中，我们使用了userId字段值。这两个参数意味着用于索引的每个文档都需要定义userId字段。这

很便捷，因为现在使用批量索引时，无需单个分支的所有文档都使用相同的路由值（而设置路由参数的情况下，只能这样）。然而，请记住，在使用路由字段时，Elasticsearch需要一些额外的解析，因此比使用路由参数时慢一点。

添加路由部分后，整个更新的映射文件将如下所示：

```
{
  "mappings" : {
    "post" : {
      "_routing" : {
        "required" : true,
        "path" : "userId"
      },
      "properties" : {
        "id" : { "type" : "long", "store" : "yes",
        "precision_step" : "0" },
        "name" : { "type" : "string", "store" : "yes",
        "index" : "analyzed" },
        "contents" : { "type" : "string", "store" : "no",
        "index" : "analyzed" },
        "userId" : { "type" : "long", "store" : "yes",
        "precision_step" : "0" }
      }
    }
  }
}
```

如果想使用上述映射来创建posts索引，可使用下面的命令来为单个测试文档建立索引：

```
curl -XPOST 'localhost:9200/posts/post/1' -d '{
  "id":1,
  "name":"New post",
  "contents": "New test post",
  "userId":1234567
}'
```

Elasticsearch将使用1234567作为索引时的路由值。

2.7 小结

本章介绍了Elasticsearch索引的工作原理；如何创建自己的映射以定义索引结构，并使用它们创建索引；批量索引是什么以及如何使用，如何有效地索引数据；文档中可以存储哪些附加信息。除此之外，我们还了解了段合并是什么，如何配置，以及什么是调节。最后，学习了路由的使用和配置。

下一章的重点放在搜索上。我们将先介绍如何对Elasticsearch查询，有哪些基本的查询可用。之后，将使用过滤器，并了解它们为什么重要。接下来，我们将学习如何验证查询和使用高亮功能。最后，使用复合查询，探索查询的内部机制，对查询结果排序。