第 4 章

扩展索引结构

上一章介绍了不少有关查询Elasticsearch的知识，展示了如何在Elasticsearch中选择返回字段，并说明了查询的工作方式。除此之外，我们还了解到可用的基本查询以及过滤数据的方法，如何高亮文档中的匹配文字，如何验证查询、复合查询，以及如何对数据排序。这一章的主要内容如下：

- 索引树型结构数据；
- 索引非扁平数据；
- 在可能情况下修改索引结构；
- 使用嵌套文档索引关系型数据；
- 使用主从功能索引关系型数据。

4.1 索引树形结构

树型结构随处可见。如果开发一个商店应用程序，可能会需要类别。看看文件系统，文件和目录以树状结构排列。本书也呈树型：章节包含各种主题，而主题又划分为副主题。想象得出，Elasticsearch也能够索引树状结构。让我们看看如何通过`path_analyzer`浏览这种数据类型。

4.1.1 数据结构

首先，通过以下代码创建一个简单的索引结构：

```
curl -XPUT 'localhost:9200/path' -d '{
  "settings" : {
    "index" : {
      "analysis" : {
        "analyzer" : {
          "path_analyzer" : { "tokenizer" : "path_hierarchy" }
        }
      }
    }
  },
```

```
  "mappings" : {
    "category" : {
      "properties" : {
        "category" : {
          "type" : "string",
          "fields" : {
            "name" : { "type" : "string",
                       "index" : "not_analyzed" },
            "path" : { "type" : "string",
                       "analyzer" : "path_analyzer",
                       "store" : true }
          }
        }
      }
    }
  }
}'
```

可以看到，我们创建了一个类型：category。我们将使用它在树型结构中存储文档位置的信息。想法很简单，可以用与在硬盘里显示文件和目录完全相同的方式，以路径的形式显示文档位置。例如，在一家汽车店中可以有如下路径：/cars/passenger/sport、/cars/passenger/camper，或者/cars/delivery_truck/。需要用三种方式对这个路径建立索引。我们将使用未经额外处理的name字段，以及一个使用定义的path_analyzer处理的path字段，也保留原始值以方便搜索。

4.1.2 分析

现在，让我们看看Elasticsearch在分析的过程中如何处理类别路径。为此，运用5.7节中描述的分析API，执行以下命令：

```
curl -XGET 'localhost:9200/path/_analyze?field=category.path&pretty' -d
'/cars/passenger/sport'
```

Elasticsearch返回的结果如下所示：

```
{
  "tokens" : [ {
    "token" : "/cars",
    "start_offset" : 0,
    "end_offset" : 5,
    "type" : "word",
    "position" : 1
  }, {
    "token" : "/cars/passenger",
    "start_offset" : 0,
    "end_offset" : 15,
    "type" : "word",
    "position" : 1
  }, {
    "token" : "/cars/passenger/sport",
```

```
      "start_offset" : 0,
      "end_offset" : 21,
      "type" : "word",
      "position" : 1
    } ]
}
```

可以看到，Elasticsearch把类别路径/cars/passenger/sport处理并分解成三个标记。归功于此，我们很容易通过词条过滤器找到每个属于指定类别或子类别的文档。举例如下：

```
{
  "filter" : {
   "term" : { "category.path" : "/cars" }
  }
}
```

注意，我们还在索引中建立了`category.name`字段的原始值，便于直接找到特定路径的文档，略过层次结构更深的文档。

4.2　索引非扁平数据

并非所有数据都是扁平数据，就像到目前为止本书使用的所有数据。如果我们构建使用Elasticsearch的系统，应创建有利于Elasticsearch的结构。结构不总是扁平的，因为并非所有用例都允许这样操作。让我们看看如何使用完全结构化的JSON对象创建映射。

4.2.1　数据

假定有如下数据（存储于structured_data.json文件中）：

```
{
  "book" : {
    "author" : {
      "name" : {
        "firstName" : "Fyodor",
        "lastName" : "Dostoevsky"
      }
    },
    "isbn" : "123456789",
    "englishTitle" : "Crime and Punishment",
    "year" : 1886,
    "characters" : [
      {
        "name" : "Raskolnikov"
      },
    {
        "name" : "Sofia"
      }
    ],
```

```
      "copies" : 0
    }
  }
```

可以看到，在上面的代码中，数据并非是扁平的；它包含了数组和嵌套对象。如果要运用目前为止学到的知识创建映射，我们不得不将数据变为扁平。然而，Elasticsearch允许文档中存在一定程度的结构，可以创建能够处理上述示例的映射。

4.2.2 对象

上述示例显示了结构化的JSON文档。可以看到，示例文档的根对象是book，具备一些额外、简单的属性，比如englishTitle。它们将以正常字段的形式索引。此外，还有characters数组类型，这一点将在接下来的段落中讨论。现在，让我们关注author对象，可以看到，它还嵌套了另一个有两个属性firstName和lastName的对象name。

4.2.3 数组

我们已经使用过数组类型的数据，但未详细讨论。默认情况下，在Lucene中的所有字段都是多值的，因此在Elasticsearch中也是一样，这意味着它们可以存储多个值。为了索引这些字段，我们使用JSON数组类型，嵌套在中括号[]中。上述示例里，我们对book中的characters使用了数组类型。

4.2.4 映射

为索引数组，只需要在数组名称中指定字段的属性。因此，在我们的例子中，可添加以下映射来索引characters的数据：

```
"characters" : {
  "properties" : {
    "name" : {"type" : "string", "store" : "yes"}
  }
}
```

没什么特殊的，仅仅在数组名称（在例子中为characters）中嵌套了properties节点，并定义字段。由于前面的映射，我们在索引中获得多值字段characters.name。

同样，对于对象author，使用数据中同样的名称，除了properties部分，我们还添加了type属性并设置值为object，告知Elasticsearch应期待一个对象类型。在对象author中嵌套了对象name，因此author字段的映射如下所示：

```
"author" : {
  "type" : "object",
```

```
    "properties" : {
      "name" : {
        "type" : "object",
        "properties" : {
          "firstName" : {"type" : "string", "index" : "analyzed"},
          "lastName" : {"type" : "string", "index" : "analyzed"}
        }
      }
    }
  }
```

firstName和lastName字段在索引中体现为author.name.firstName和author.name.lastName。

其余字段为简单的核心类型，2.2节已经讨论过，这里不再赘述。

最终映射

所以，我们的映射文件structured_mapping.json的最终映射如下所示：

```
{
  "book" : {
    "properties" : {
      "author" : {
        "type" : "object",
        "properties" : {
          "name" : {
            "type" : "object",
            "properties" : {、
              "firstName" : {"type" : "string", "store": "yes"},
              "lastName" : {"type" : "string", "store": "yes"}
            }
          }
        }
      },
      "isbn" : {"type" : "string", "store": "yes"},
      "englishTitle" : {"type" : "string", "store": "yes"},
      "year" : {"type" : "integer", "store": "yes"},
      "characters" : {
        "properties" : {
          "name" : {"type" : "string", "store": "yes"}
        }
      },
      "copies" : {"type" : "integer", "store": "yes"}
    }
  }
}
```

可以看到，所有字段中store的属性值均设置为yes。这只是为了向你展示所有字段都可以正常索引。

4.2.5 向Elasticsearch发送映射

现在，映射已经完成，我们可测试确认其是否有效。这次将使用一种稍微不同的技术来创建索引和映射。首先，使用以下命令行创建library索引：

```
curl -XPUT 'localhost:9200/library'
```

接着，使用以下命令行，将映射发送至book类型：

```
curl -XPUT 'localhost:9200/library/book/_mapping' -d @structured_mapping.
json
```

现在，可使用以下命令行索引我们的示例数据：

```
curl -XPOST 'localhost:9200/library/book/1' -d @structured_data.json
```

4.2.6 动态还是非动态

我们知道，Elasticsearch是无模式的，这意味着不必创建前面的映射就可索引数据。Elasticsearch的动态行为默认是打开的，但可能想在索引的某些部分把它关掉。为此，可为指定字段增加属性dynamic，将值设置为false，该属性应该设置在与非动态对象的type属性相同的级别上。举例来说，如果我们希望对象author和name为非动态，应该将映射文件的相关部分修改成类似下面这样：

```
"author" : {
  "type" : "object",
  "dynamic" : false,
  "properties" : {
    "name" : {
      "type" : "object",
      "dynamic" : false,
      "properties" : {
        "firstName" : {"type" : "string", "index" : "analyzed"},
        "lastName" : {"type" : "string", "index" : "analyzed"}
      }
    }
  }
}
```

应记住，为此类对象增加新字段时，应更新映射。

你也可以在elasticsearch.yml配置文件中添加index.mapper.dynamic属性，将值设置为false，关掉动态映射功能。

4.3　使用嵌套对象

某些情况下嵌套对象可以很方便。基本上，通过使用嵌套对象，Elasticsearch允许我们连接一个主文档和多个附属文档。主文档及嵌套文档一同被索引，放置于索引的同一段上（实际在同一块上），确保为该数据结构获取最佳性能。更改文档也是一样的，除非使用更新API，你需要同时索引父文档和其他所有嵌套文档。

如果想阅读更多Lucene中嵌套对象的工作方式，可参考迈克·麦坎德利斯（Mike McCandless）的博客，链接如下：http://blog.mikemccandless.com/2012/01/searching-relational-content-with.html。

现在，我们来看示例。假设有一个服装店，需要存储每件T恤的尺寸和颜色，那么，标准的、非嵌套映射将类似于以下的代码（存储于cloth.json中）：

```
{
  "cloth" : {
    "properties" : {
      "name" : {"type" : "string"},
      "size" : {"type" : "string", "index" : "not_analyzed"},
      "color" : {"type" : "string", "index" : "not_analyzed"}
    }
  }
}
```

假设仅有一件XXL的红色T恤和一件XL的黑色T恤，示例文档将类似于以下代码：

```
{
  "name" : "Test shirt",
  "size" : [ "XXL", "XL" ],
  "color" : [ "red", "black" ]
}
```

然而，这个数据结构有个问题：假使客户要在商店搜索XXL黑色T恤，会产生什么结果？运行以下查询来检查（假设我们已使用映射创建索引并在其中建立了示例文档）：

```
curl -XGET 'localhost:9200/shop/cloth/_search?pretty=true' -d '{
  "query" : {
   "bool" : {
    "must" : [
     {
      "term" : { "size" : "XXL" }
     },
     {
      "term" : { "color" : "black" }
     }
    ]
   }
  }
}'
```

我们不应该得到结果，对吧？但事实上，Elasticsearch返回了如下文档：

```
{
  (...)
  "hits" : {
    "total" : 1,
    "max_score" : 0.4339554,
    "hits" : [ {
      "_index" : "shop",
      "_type" : "cloth",
      "_id" : "1",
      "_score" : 0.4339554,
      "_source" : { "name" : "Test shirt",
                    "size" : [ "XXL", "XL" ],
                    "color" : [ "red", "black" ]}
    } ]
  }
}
```

这是因为，经过比对文档，在size字段和color字段上有我们需要的值。当然，这不是我们想要的。因此，修改映射，使用嵌套对象来分离color和size。最终的映射看起来如下所示（我们把这些映射存到cloth_nested.json文件中）：

```
{
  "cloth" : {
    "properties" : {
      "name" : {"type" : "string", "index" : "analyzed"},
      "variation" : {
        "type" : "nested",
        "properties" : {
          "size" : {"type" : "string", "index" : "not_analyzed"},
          "color" : {"type" : "string", "index" : "not_analyzed"}
        }
      }
    }
  }
}
```

可以看到，我们在cloth类型中引入了新对象variation，它是嵌套的（type属性设置为nested），表示想为嵌套文档建立索引。现在修改文档，添加variation对象，其中有两个属性：size和color。示例产品将如下所示：

```
{
  "name" : "Test shirt",
  "variation" : [
    { "size" : "XXL", "color" : "red" },
    { "size" : "XL", "color" : "black" }
  ]
}
```

组织文档结构，以便每个尺寸及其匹配颜色成为一个独立文档。然而，如果执行之前的查询，

将无任何文档返回。这是因为，对于嵌套文件，需要使用专门的查询。因此，查询如下（当然，我们已经再次创建了索引和类型）：

```
curl -XGET 'localhost:9200/shop/cloth/_search?pretty=true' -d '{
  "query" : {
    "nested" : {
      "path" : "variation",
      "query" : {
        "bool" : {
          "must" : [
            { "term" : { "variation.size" : "XXL" } },
            { "term" : { "variation.color" : "black" } }
          ]
        }
      }
    }
  }
}'
```

现在，上述查询将无法返回索引中的文档，因为无法找到尺寸XXL且颜色为黑色的嵌套文档。这里简单讨论一下我们的查询，可以看到，我们使用nested查询来查询嵌套文档。path属性指定了嵌套对象的名称（可以使用多个名称）。nested类型包括了一个标准查询部分。应注意的是，在嵌套对象中为字段名称指定完整的路径，在多级嵌套中很方便操作（这也是可能的）。

如果你想在嵌套对象的基础上过滤数据，可使用嵌套过滤器，它具备与嵌套查询相同的功能。更多相关信息，请参阅3.5节。

评分与嵌套查询

在查询过程中处理嵌套文档时，有一个附加属性。除path属性外，还有个score_mode属性，它允许我们定义如何从嵌套查询中计算得分。在Elasticsearch中可将此属性设置为如下值。

- avg：这是默认值。使用这个值时，Elasticsearch可在指定的嵌套查询中计算出平均值。该平均值包含在主查询的得分中。
- total：score_mode属性设置为此值时，Elasticsearch可对每个嵌套查询的得分求和。该值包含在主查询的得分中。
- max：score_mode属性设置为此值时，Elasticsearch可得出嵌套查询的最高得分。该值包含在主查询的得分中。
- none：score_mode属性设置为此值时，Elasticsearch不计算嵌套查询的得分。

4.4 使用父子关系

上一节已讨论了索引嵌套文档及其父文档的能力。然而，即使嵌套文档在索引中是作为独立文档检索的，除非使用更新API，否则还是无法更改单个嵌套文档。而在Elasticsearch中，我们可利用父子关系操作。请看以下内容。

4.4.1 索引结构和数据索引

在此，参考之前讨论嵌套文档时使用的示例：假想的服装店。然而我们希望的是：在每次变更后，无需索引整个文档即可更新尺寸和颜色。

1. 父文档映射

在父文档中，name是我们需要的唯一字段。因此，在shop索引中创建cloth类型，执行如下命令：

```
curl -XPOST 'localhost:9200/shop'
curl -XPUT 'localhost:9200/shop/cloth/_mapping' -d '{
  "cloth" : {
    "properties" : {
      "name" : {"type" : "string"}
    }
  }
}'
```

2. 子文档映射

为创建子文档映射，要在_parent属性中添加父类型的名称，在我们的示例中为cloth。因此，创建类型variation的命令行将如下所示：

```
curl -XPUT 'localhost:9200/shop/variation/_mapping' -d '{
  "variation" : {
    "_parent" : { "type" : "cloth" },
    "properties" : {
      "size" : {"type" : "string", "index" : "not_analyzed"},
      "color" : {"type" : "string", "index" : "not_analyzed"}
    }
  }
}'
```

我们无需指定连接父子文档的字段，因为默认情况下Elasticsearch会使用唯一标识符。如前面相关章节所述，唯一标识符以默认的形式存在于索引中。

3. 父文档

现在，我们来索引父文档。操作很简单，只要执行索引命令，示例如下：

```
curl -XPOST 'localhost:9200/shop/cloth/1' -d '{
  "name" : "Test shirt"
}'
```

上述命令中，我们指定的文档标识符为1。

4. 子文档

为索引子文档，需要使用parent参数提供父文档的相关信息，将该参数设置为父文档的标识符。所以，为索引父文档中的两个子文档，执行下面的命令：

```
curl -XPOST 'localhost:9200/shop/variation/1000?parent=1' -d '{
  "color" : "red",
  "size" : "XXL"
}'
```

同样，执行如下命令行索引第二个子文档：

```
curl -XPOST 'localhost:9200/shop/variation/1001?parent=1' -d '{
  "color" : "black",
  "size" : "XL"
}'
```

这样，我们索引了两个附加文档，它们是新类型，但是我们已为其指定标识符为1的父文档。

4.4.2　查询

我们已经索引了数据，现在需要恰当的查询来匹配拥有子文档数据的文档。当然，也可针对子文档来执行查询并检测其父文档是否存在。然而要注意的是，针对父文档执行查询时，子文档将无法返回，反之亦然。

1. 查询子文档中的数据

如果要寻找XXL号的红色衣服，可以运行如下命令行：

```
curl -XGET 'localhost:9200/shop/_search?pretty' -d '{
  "query" : {
    "has_child" : {
      "type" : "variation",
      "query" : {
        "bool" : {
          "must" : [
            { "term" : { "size" : "XXL" } },
            { "term" : { "color" : "red" } }
          ]
        }
      }
    }
  }
}'
```

查询很简单。类型has_child告知Elasticsearch我们想在子文档中搜索。为了指定感兴趣的子类型，指定type属性为子类型的名称。然后用一个标准的bool查询（已经讨论过），查询的结果仅包含父文档，示例如下：

```
{
  (...)
  "hits" : {
    "total" : 1,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "shop",
      "_type" : "cloth",
      "_id" : "1",
      "_score" : 1.0, "_source" : { "name" : "Test shirt" }
    } ]
  }
}
```

- top_children查询

除has_child查询之外，Elasticsearch还公开了top_children查询，它查询子文档但返回父文档。此查询可针对特定数量的子文档，示例如下：

```
{
  "query" : {
    "top_children" : {
      "type" : "variation",
      "query" : {
        "term" : { "size" : "XXL" }
      },
      "score" : "max",
      "factor" : 10,
      "incremental_factor" : 2
    }
  }
}
```

上述查询首先在100个子文档中运行（factor乘以size的默认参数10）。如果找到10个父文档（因为默认size的参数值为10），这些文档将返回并结束查询。然而，如果返回的父文档数量较少，且尚有子文档未经查询，那么另外20个子文档将被查询（incremental_factor参数乘以size），直到找到规定数量的父文档或者所有子文档查询结束为止。

top_children查询通过使用score参数指定得分的计算方式，可能的参数值包括：max（所有子查询得分的最大值）、sum（所有子查询得分的总和）或avg（所有子查询得分的平均值）。

2. 查询父文档中的数据

如果想要返回与父文档中指定数据匹配的子文档，可使用类似于has_child的查询：has_parent。然而，我们用父文档类型的值指定parent_type属性，而不是type属性。这么

这个查询将返回索引的子文档，而不是父文档：

```
curl -XGET 'localhost:9200/shop/_search?pretty' -d '{
  "query" : {
    "has_parent" : {
      "parent_type" : "cloth",
      "query" : {
        "term" : { "name" : "test" }
      }
    }
  }
}'
```

Elasticsearch的响应如下所示：

```
{
  (...)
  "hits" : {
    "total" : 2,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "shop",
      "_type" : "variation",
      "_id" : "1000",
      "_score" : 1.0, "_source" : {"color" : "red","size" : "XXL"}
    }, {
      "_index" : "shop",
      "_type" : "variation",
      "_id" : "1001",
      "_score" : 1.0, "_source" : {"color" : "black","size" : "XL"}
    } ]
  }
}
```

4.4.3　父子关系和过滤

如果想要将父子查询作为过滤器使用，可以用过滤器has_child和has_parent，它们具备了与has_child和has_parent查询相同的功能。实际上，Elasticsearch将那些过滤器封装为常数得分查询，使其可作为查询使用。

4.4.4　性能考虑

使用Elasticsearch父子的功能时，必须注意它的性能影响。需要记住的第一件事是父子文档需要存储在相同的分片中，查询才能够工作。如果单一父文档有大量的子文档，可能导致分片上的文档数量不平均。因此，其中的一个节点的性能会降低，造成整个查询速度变慢。另外，请记住，比起查询无任何关联的文档，父子查询的速度较慢。

第二个非常重要的事情是，执行has_child等查询时，Elasticsearch需要预加载并缓存文档标识符。这些标识符将存储在内存中，必须确保Elasticsearch有足够的内存。否则，你将得到OutOfMemory异常，节点或整个集群将无法运作。

最后，我们提到过，首次查询将花一定时间预加载和缓存文档标识符。为了提升首次查询父子关系文档的性能，可以使用预热API。关于如何在Elasticsearch中添加预热查询，请参考8.5节。

4.5 使用更新 API 修改索引结构

前面的章节讨论过如何创建索引映射和索引数据。如果你已经创建了映射和索引数据，想要修改索引的结构，应该怎么办？在某种程度上这是可行的。例如，默认情况下索引一个带新字段的文档，Elasticsearch会将该字段增加到索引结构中。现在看看如何手动修改索引结构。

4.5.1 映射

假设我们的users索引有以下映射，存储于user.json文件中：

```
{
  "user" : {
    "properties" : {
      "name" : {"type" : "string"}
    }
  }
}
```

可以看到，它很简单，只有一个属性保存用户名。现在，让我们创建一个名为users的索引，并使用上面的映射创建自己的类型。为此，运行以下命令：

```
curl -XPOST 'localhost:9200/users'
curl -XPUT 'localhost:9200/users/user/_mapping' -d @user.json
```

如果一切正常，我们的索引和类型便创建好了。现在，添加一个新字段到映射中去。

4.5.2 添加一个新字段

为了说明如何为映射添加新字段，我们假设要为每个存储的用户添加一个电话号码。为此，需要将HTTP PUT命令发送到带有合适主体的/index_name/type_name/_mapping REST端点，该主体中包含我们的新字段。例如，为添加phone字段，执行以下命令：

```
curl -XPUT 'http://localhost:9200/users/user/_mapping' -d '{
  "user" : {
    "properties" : {
      "phone" : {"type" : "string",
```

```
                "store" : "yes",
                "index" : "not_analyzed"}
    }
  }
}'
```

同样，如果一切正常，新字段便添加到我们的索引结构中去了。为了确保一切正常，可以运行HTTP GET请求到_mapping端点；Elasticsearch将返回适当的映射。在索引users中获得user类型映射的示例命令如下所示：

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

在现有类型中添加新字段后，需要再次对所有文档进行索引，因为Elasticsearch不会自动更新。这很关键，可以使用初始数据源或从_source字段中获得初始数据并再次索引。

4.5.3　修改字段

现在，我们的索引结构包含两个字段：name和phone。我们索引了一些数据，但之后又决定搜索phone字段，并希望更改index属性，从not_analyzed改为analyzed，为此，执行以下命令：

```
curl -XPUT 'http://localhost:9200/users/user/_mapping' -d '{
  "user" : {
    "properties" : {
      "phone" : {"type" : "string",
                 "store" : "yes",
                 "index" : "analyzed"}
    }
  }
}'
```

执行上面的命令行后，Elasticsearch返回以下输出：

```
{"error":"MergeMappingException[Merge failed with failures {[mapper
[phone] has different index values, mapper [phone] has different 'norms.
enabled' values, mapper [phone] has different tokenize values, mapper
[phone] has different index_analyzer]}]","status":400}
```

这是因为无法将not_analyzed字段更改为analyzed。不仅如此，在大部分情况下字段映射是无法更新的。这是好事，因为如果我们可以更改这样的设置，会让Elasticsearch和Lucene混乱。假设已经有很多文档带有设置为not_analyzed的phone字段，将这些设置更改为analyzed，Elasticsearch将无法更改已索引的文档，但已经分析的查询将以不同的逻辑处理，那么我们就无法正确查找数据。

我们在此将提及一些操作，举例说明哪些是禁止的，哪些是允许的。如下修改是安全的：

- 增加新的类型定义；
- 增加新的字段；
- 增加新的分析器。

而以下修改是不允许或是无法实现的：

- 更改字段类型（如将文本改为数字）；
- 更改“存储到”字段为不存储，反之亦然；
- 更改索引属性的值；
- 更改已索引文档的分析器。

注意一点，上述允许和不允许的操作没有涵盖更新API的全部可能性，你必须实际操作以验证更新是否可行。

如果你想忽略冲突并设置新映射，可设置ignore_conflicts的参数值为true，Elasticsearch将会重写映射。带有额外参数的命令行如下：

```
curl -XPUT 'http://localhost:9200/users/user/_mapping?
ignore_conflicts=true' -d '...'
```


4.6 小结

这一章讲述了如何使用Elasticsearch索引树型结构，索引非扁平数据，以及修改已创建的索引结构。最后，介绍了如何使用嵌套文档及Elasticsearch中的父子功能来处理关系型数据。

下一章的重点是更高效地搜索，我们将看到Apache Lucene得分的工作方式及其重要性；学习使用Elasticsearch的函数得分查询，用函数以及提供的脚本功能来调整不同文档的重要性，使用不同的语言来搜索，并讨论索引时加权什么时候有意义。下一章将使用同义词匹配具有相同含义的单词，介绍检测文档被查询到的原因。最后，使用加权来影响查询，并解释Elasticsearch的得分计算。