# 知道的，　了搜索...

Elasticsearch 是一个　源的搜索引　，建立在一个全文搜索引　　Apache Lucene™ 基　之上。 Lucene 可以　是当下最先　、高性能、全功能的搜索引　　—无　是　源　是私有。

但是　Lucene　　只是一个　。　了充分　　其功能，　需要使用　Java　并将　Lucene　直接集成到　用程序中。更糟　的是，　可能需要　得信息　索学位才能了解其工作原理。Lucene 非常　　。

Elasticsearch 也是使用 Java　写的，它的内部使用 Lucene 做索引与搜索，但是它的目的是使全文　索　得　　，通　　藏 Lucene 的　　性，取而代之的提供一套　　一致的 RESTful API。

然而，Elasticsearch　不　　是　Lucene，并且也不　　只是一个全文搜索引　。　它可以被下面　　准　的形容：

- 一个分布式的　　文　存　，　个字段 可以被索引与搜索
- 一个分布式　　分析搜索引
- 能　任上百个服　　点的　展，并支持 PB　　的　　化或者非　　化数据

Elasticsearch 将所有的功能打包成一个　独的服　，　　可以通　程序与它提供的　　的 RESTful API　行通信，可以使用自己喜　的　程　言充当 web 客　端，甚至可以使用命令行（去充当　个客　端）。

就　　Elasticsearch　　而言，起　很　　。　于初学者来　，它　　了一些　当的　　，并　藏了　的搜索理　知　。它　箱即用。只需最少的理解，　很快就能具有生　力。

随着　知　的　累，　可以利用 Elasticsearch　更多的高　特性，它的整个引　是可配置并且　活的。　从　多高　特性中，挑　恰当去修　的 Elasticsearch，使它能解决　本地遇到的　　。

　可以免　下　，使用，修改 Elasticsearch。它在 Apache 2 license　　下　布的，　　是　多　活的源　　之一。Elasticsearch　的源　被托管在　Github　上　github.com/elastic/elasticsearch。　如果　想加入我　　个令人　奇的 contributors 社区，看　里 Contributing to Elasticsearch。

如果　　Elasticsearch　有任何相　的　　，包括特定的特性(specific　features)、　言客　端(language clients)、　件(plugins)，可以在　里 discuss.elastic.co 加入　　。

回　光

多年前，一个　婚的名叫　Shay　Banon　的失　者，跟着他的妻子去了　敦，他的妻子在那里学　厨　。　在　一个　的工作的　候，　了　他的妻子做一个食搜索引　，他　始使用 Lucene 的一个早期版本。

直接使用 Lucene 是很　的，因此 Shay　始做一个抽象　，Java　者使用它可以很　的　他　的程序添加搜索功能。他　布了他的第一个　源　目 Compass。

后来　Shay　得了一　工作，主要是高性能，分布式　境下的内存数据　格。　个　于高性能，　，分布式搜索引　的需求尤　突出，　他决定重写　Compass，把它　一个独立的服　并取名 Elasticsearch。

第一个公　版本在2010年2月　布，从此以后，Elasticsearch　已　成　了　Github　上最活　的　目之一，他　有超　300名　contributors(目前736名　contributors　)。　一家公司已　始 Elasticsearch 提供商　服　，并　新的特性，但是，Elasticsearch 将永　源并　所有人可用。

据　，Shay 的妻子　在等着　的食　搜索引　...

# 安装并　行 Elasticsearch

想用最　的方式去理解 Elasticsearch 能　做什　，那就是使用它了，　我　始　！

安装 Elasticsearch 之前，　需要先安装一个　新的版本的 Java，最好的　是，　可以从 *www.java.com* 　得官方提供的最新版本的 Java。

之后，　可以从 elastic 的官　*elastic.co/downloads/elasticsearch*　取最新版本的 Elasticsearch。

要想安装 Elasticsearch，先下　并解　合　操作系　的 Elasticsearch 版本。如果　想了解更多的信息，可以　看　Elasticsearch　参考手　里　的安装部分，　出的　接指向安装　明 {ref}/_installation.html[Installation]。

| | |
|---|---|
| **TIP** | 当　准　在生　境安装 Elasticsearch　，　可以在 官　下　地址　到 Debian 或者 RPM 包，除此之外，　也可以使用官方支持的 Puppet module 或者 Chef cookbook。 |

当　解　好了　文件之后，Elasticsearch　已　准　好　行了。按照下面的操作，在前台(foregroud) Elasticsearch：

```
cd elasticsearch-<version>
./bin/elasticsearch ① ②
```

① 如果　想把 Elasticsearch 作　一个守　程在后台　行，那　可以在后面添加参数 `-d` 。

② 如果　是在　Windows　上面　行　Elasticseach，　行　`bin\elasticsearch.bat`　而不是 `bin\elasticsearch` 。

Elasticsearch 是否　成功，可以打　一个　端，　行以下操作：

```
curl 'http://localhost:9200/?pretty'
```

TIP：如果 是在 Windows 上面 行 Elasticsearch， 可以从 http://curl.haxx.se/download.html 中下 cURL。 cURL 提供了一 将 求提交到 Elasticsearch 的便捷方式，并且安装 cURL 之后， 可以通 制与粘 去 中的 多例子。

得到和下面 似的 (response)：

```
{
  "name" : "Tom Foster",
  "cluster_name" : "elasticsearch",
  "version" : {
    "number" : "2.1.0",
    "build_hash" : "72cd1f1a3eee09505e036106146dc1949dc5dc87",
    "build_timestamp" : "2015-11-18T22:40:03Z",
    "build_snapshot" : false,
    "lucene_version" : "5.3.1"
  },
  "tagline" : "You Know, for Search"
}
```

就意味着 在已 并 行一个 Elasticsearch 点了， 可以用它做 了。 个 点 可以作 一个 行中的 Elasticsearch 的 例。 而一个 集群 是一 有相同 cluster.name 的 点， 他 能一起工作并共享数据， 提供容 与可伸 性。(当然，一个 独的 点也可以 成一个集群) 可以在 elasticsearch.yml 配置文件中 修改 cluster.name ， 文件会在 点 加 ( 者注： 个重 服 后才会生效)。 于上面的 cluster.name 以及其它 Important Configuration Changes 信息， 可以在 本 后面提供的生 部署章 到更多。

TIP：看到下方的 View in Sense 的例子了 ？Install the Sense console 使用 自己的 Elasticsearch 集群去 行 本 中的例子， 看会有 的 果。

当 Elastcisearch 在前台 行 ， 可以通 按 Ctrl+C 去停止。

## 安装 Sense

Sense 是一个 Kibana 用 它提供交互式的控制台，通 的 器直接向 Elasticsearch 提交 求。 本 的在 版本包含有一个 View in Sense 的 接，里面有 多代 示例。当点 的 候，它会打 一个代 示例的Sense控制台。 不必安装 Sense，但是它允 在本地的 Elasticsearch 集群上 示例代 ，从而使本 更具有交互性。

安装与 行 Sense：

1. 在 Kibana 目 下 行下面的命令，下 并安装 Sense app：

   ```
   ./bin/kibana plugin --install elastic/sense ①
   ```

   ① Windows上面 行: bin\kibana.bat plugin --install elastic/sense 。

NOTE： 可以直接从 里 https://download.elastic.co/elastic/sense/sense-latest.tar.gz 下 Sense 安装可以 看 里 install it on an offline machine 。

2. Kibana.

```
./bin/kibana ①
```

① Windows 上 kibana: `bin\kibana.bat` 。

3. 在 的 器中打 Sense: `http://localhost:5601/app/sense` 。

# 和 Elasticsearch 交互

和 Elasticsearch 的交互方式取决于 是否使用 Java

## Java API

如果 正在使用 Java，在代 中 可以使用 Elasticsearch 内置的 个客 端：

点客 端（*Node client*）

点客 端作 一个非数据 点加入到本地集群中。 句 ，它本身不保存任何数据，但是它知道数据在集群中的 个 点中，并且可以把 求 到正 的 点。

客 端（*Transport client*）

量 的 客 端可以将 求 送到 程集群。它本身不加入集群，但是它可以将 求 到集群中的一个 点上。

个 Java 客 端都是通 *9300* 端口并使用本地 Elasticsearch 和集群交互。集群中的 点通端口 9300 彼此通信。如果 个端口没有打 ， 点将无法形成一个集群。

**TIP** Java 客 端作 点必 和 Elasticsearch 有相同的 主要 版本；否 ，它 之 将无法互相理解。

更多的 Java 客 端信息可以在 Elasticsearch Clients 中 到。

## RESTful API with JSON over HTTP

所有其他 言可以使用 RESTful API 通 端口 *9200* 和 Elasticsearch 行通信， 可以用 最喜 的 web 客 端 Elasticsearch 。事 上，正如 所看到的， 甚至可以使用 `curl` 命令来和 Elasticsearch 交互。

**NOTE** Elasticsearch 以下 言提供了官方客 端--Groovy、JavaScript、.NET、 PHP、 Perl、Python 和 Ruby— 有很多社区提供的客 端和 件，所有 些都可以在 Elasticsearch Clients 中 到。

一个 Elasticsearch 求和任何 HTTP 求一 由若干相同的部件 成：

```
curl -X<VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY_STRING>' -d '<BODY>'
```

被 < > 的部件：

VERB
: 当的 HTTP 方法 或 : GET、 POST、 PUT、 HEAD 或者 DELETE。

PROTOCOL
: http 或者 https（如果 在 Elasticsearch 前面有一个 https 代理）

HOST
: Elasticsearch 集群中任意 点的主机名，或者用 localhost 代表本地机器上的 点。

PORT
: 行 Elasticsearch HTTP 服 的端口号， 是 9200 。

PATH
: API 的 端路径（例如 _count 将返回集群中文 数量）。Path 可能包含多个件，例如：_cluster/stats 和 _nodes/stats/jvm 。

QUERY_STRING
: 任意可 的 字符串参数 (例如 ?pretty 将格式化地 出 JSON 返回 ，使其更容易 )

BODY
: 一个 JSON 格式的 求体 (如果 求需要的 )

例如， 算集群中文 的数量，我 可以用 个:

```
curl -XGET 'http://localhost:9200/_count?pretty' -d '
{
    "query": {
        "match_all": {}
    }
}
'
```

Elasticsearch 返回一个 HTTP 状 （例如：200 OK）和（除`HEAD` 求）一个 JSON 格式的返回 。前面的 curl 求将返回一个像下面一 的 JSON 体：

```
{
    "count" : 0,
    "_shards" : {
        "total" : 5,
        "successful" : 5,
        "failed" : 0
    }
}
```

在返回 果中没有看到 HTTP 信息是因 我 没有要求`curl` 示它 。想要看到 信息，需要 合 `-i` 参数来使用 `curl` 命令：

```
curl -i -XGET 'localhost:9200/'
```

在 中剩余的部分，我 将用 写格式来展示 些 `curl` 示例，所 的 写格式就是省略 求中所有相同的部分，例如主机名、端口号以及 `curl` 命令本身。而不是像下面 示的那 用一个完整的 求：

```
curl -XGET 'localhost:9200/_count?pretty' -d '
{
    "query": {
        "match_all": {}
    }
}'
```

我 将用 写格式 示：

```
GET /_count
{
    "query": {
        "match_all": {}
    }
}
```

事 上， Sense 控制台 也使用 相同的格式。如果 正在 本 的在 版本,可以通 点 Sense 接 在 Sense 上打 和 行示例代 。

# 面向文

在 用程序中 象很少只是一个 的 和 的列表。通常，它 有更 的数据 ，可能包括日期、地理信息、其他 象或者数 等。

也 有一天 想把 些 象存 在数据 中。使用 系型数据 的行和列存 ， 相当于是把一个表 力富的 象 到一个非常大的 子表格中： 必 将 个 象扁平化来 表 —通常一个字段 > 一列—而且又不得不在 次 重新 造 象。

Elasticsearch 是 面向文 的，意味着它存 整个 象或 文 。Elasticsearch 不 存 文 ，而且 索引 个文 的内容使之可以被 索。在 Elasticsearch 中， 文 行索引、 索、排序和 —而不是行列数据。 是一 完全不同的思考数据的方式，也是 Elasticsearch 能支持 全文 索的原因。

## JSON

Elasticsearch 使用 JavaScript Object Notation 或者 *JSON* 作 文 的序列化格式。JSON 序列化被大多数 程 言所支持，并且已 成 NoSQL 域的 准格式。 它 、 、易于 。

考 一下 个 JSON 文 ，它代表了一个 user 象：

```
{
    "email":      "john@smith.com",
    "first_name": "John",
    "last_name":  "Smith",
    "info": {
        "bio":         "Eco-warrior and defender of the weak",
        "age":         25,
        "interests": [ "dolphins", "whales" ]
    },
    "join_date": "2014/05/01"
}
```

然原始的 `user` 象很 ，但 个 象的 和含 在 JSON 版本中都得到了体 和保留。在 Elasticsearch 中将 象 化 JSON 并做索引要比在一个扁平的表 中做相同的事情 的多。

**NOTE**

几乎所有的 言都有相 的模 可以将任意的数据 或 象 化成 JSON 格式，只是 各不相同。具体 看 *serialization* 或者 *marshalling* 个 理 JSON 的模 。官方 Elasticsearch 客 端 自 提供 JSON 化。

## 新 境

了 Elasticsearch 能 什 及其上手容易程度有一个基本印象， 我 从一个 的教程 始并介 索引、搜索及聚合等基 概念。

我 将一并介 一些新的技 和基 概念，因此即使无法立即全 理解也无妨。在本 后 内容中，我 将深入介 里提到的所有概念。

接下来尽情享受 Elasticsearch 探索之旅。

### 建一个雇 目

我 受雇于 *Megacorp* 公司，作 HR 部 新的 “ 无人机” （*"We love our drones!"*）激励 目的一部分，我 的任 是 此 建一个雇 目 。 目 当能培 雇 同感及支持 、高效、 作，因此有一些 需求：

- 支持包含多 、数 、以及全文本的数据
- 索任一雇 的完整信息
- 允 化搜索，比如 30 以上的 工
- 允 的全文搜索以及 的短 搜索
- 支持在匹配文 内容中高亮 示搜索片段
- 支持基于数据 建和管理分析 表

# 索引雇　文

第一个　　需求就是存　雇　数据。　　将会以 雇　文　　的形式存　：一个文　代表一个雇　。存　数据到 Elasticsearch 的行　叫做 索引，但在索引一个文　之前，需要　定将文　存　在　里。

一个 Elasticsearch 集群可以 包含多个 索引，相　的　个索引可以包含多个　　型 。　　些不同的　型存着多个 文　，　个文　又有 多个 属性 。

> **Index Versus Index Versus Index**
>
> 也　已　注意到 索引　个　在 Elasticsearch　境中包含多重意思，所以有必要做一点儿　明：
>
> 索引（名　）：
>
> 如前所述，一个 索引　似于　　系数据　中的一个 数据　，是一个存　　系型文　的地方。索引 (*index*) 的　数　　*indices* 或 *indexes* 。
>
> 索引（　　）：
>
> 索引一个文　就是存　一个文　到一个 索引 （名　）中以便它可以被　索和　　到。　非常　似于 SQL　句中的 `INSERT`　　，除了文　已存在　新文　会替　旧文　情况之外。
>
> 倒排索引：
>
> 系型数据　通　　加一个　索引　比如一个　B　（B-tree）索引　到指定的列上，以便提升数据索速度。Elasticsearch 和 Lucene 使用了一个叫做 倒排索引 的　　来　到相同的目的。

+　　的，一个文　中的　一个属性都是　　被索引的（有一个倒排索引）和可搜索的。一个没有倒排索引的属性是不能被搜索到的。我　将在 [inverted-index]　　倒排索引的更多　　。

于雇　目　，我　将做如下操作：

- 个雇　索引一个文　，包含　雇　的所有信息。
- 个文　都将是 `employee`　型 。
- 　型位于 索引 `megacorp` 内。
- 索引保存在我　的 Elasticsearch 集群中。

践中　非常　　（尽管看起来有很多　），我　可以通　一条命令完成所有　些　作：

```
PUT /megacorp/employee/1
{
    "first_name" : "John",
    "last_name" :  "Smith",
    "age" :        25,
    "about" :      "I love to go rock climbing",
    "interests": [ "sports", "music" ]
}
```

注意，路径 `/megacorp/employee/1` 包含了三部分的信息：

*megacorp*
: 索引名称

*employee*
: 型名称

*1*
: 特定雇 的ID

求体 —— JSON 文 —— 包含了 位 工的所有 信息，他的名字叫 John Smith ，今年 25 ，喜 岩。

很 ！无需 行 行管理任 ，如 建一个索引或指定 个属性的数据 型之 的，可以直接只索引一个文 。Elasticsearch 地完成其他一切，因此所有必需的管理任 都在后台使用 置完成。

行下一 前， 我 加更多的 工信息到目 中：

```
PUT /megacorp/employee/2
{
    "first_name" :  "Jane",
    "last_name" :   "Smith",
    "age" :         32,
    "about" :       "I like to collect rock albums",
    "interests":  [ "music" ]
}

PUT /megacorp/employee/3
{
    "first_name" :  "Douglas",
    "last_name" :   "Fir",
    "age" :         35,
    "about":        "I like to build cabinets",
    "interests":  [ "forestry" ]
}
```

## 索文

目前我 已 在 Elasticsearch 中存 了一些数据， 接下来就能 注于 用的 需求了。第一个需求是可以 索到 个雇 的数据。

在 Elasticsearch 中很 。 地 行 一个 HTTP GET 求并指定文 的地址——索引 、 型和ID。使用 三个信息可以返回原始的 JSON 文 ：

```
GET /megacorp/employee/1
```

返回 果包含了文 的一些元数据，以及 `_source` 属性，内容是 John Smith 雇 的原始 JSON 文 ：

```
{
  "_index" :   "megacorp",
  "_type" :    "employee",
  "_id" :      "1",
  "_version" : 1,
  "found" :    true,
  "_source" :  {
      "first_name" :  "John",
      "last_name" :   "Smith",
      "age" :         25,
      "about" :       "I love to go rock climbing",
      "interests":  [ "sports", "music" ]
  }
}
```

**TIP** 将 HTTP 命令由 `PUT` 改 `GET` 可以用来 索文 ，同 的，可以使用 `DELETE` 命令来 除文 ，以及使用 `HEAD` 指令来 文 是否存在。如果想更新已存在的文 ，只需再次 `PUT` 。

## 量搜索

一个 `GET` 是相当 的，可以直接得到指定的文 。 在 点儿 微高 的功能，比如一个 的搜索！

第一个 的几乎是最 的搜索了。我 使用下列 求来搜索所有雇 ：

```
GET /megacorp/employee/_search
```

可以看到，我 然使用索引 `megacorp` 以及 型 `employee`，但与指定一个文 ID 不同， 次使用 `_search` 。返回 果包括了所有三个文 ，放在数 `hits` 中。一个搜索 返回十条 果。

```
{
   "took":      6,
   "timed_out": false,
   "_shards": { ... },
   "hits": {
      "total":      3,
      "max_score":  1,
      "hits": [
         {
            "_index":         "megacorp",
            "_type":          "employee",
            "_id":            "3",
            "_score":         1,
            "_source": {
               "first_name":  "Douglas",
               "last_name":   "Fir",
               "age":         35,
               "about":       "I like to build cabinets",
               "interests": [ "forestry" ]
            }
         },
         {
            "_index":         "megacorp",
            "_type":          "employee",
            "_id":            "1",
            "_score":         1,
            "_source": {
               "first_name":  "John",
               "last_name":   "Smith",
               "age":         25,
               "about":       "I love to go rock climbing",
               "interests": [ "sports", "music" ]
            }
         },
         {
            "_index":         "megacorp",
            "_type":          "employee",
            "_id":            "2",
            "_score":         1,
            "_source": {
               "first_name":  "Jane",
               "last_name":   "Smith",
               "age":         32,
               "about":       "I like to collect rock albums",
               "interests": [ "music" ]
            }
         }
      ]
   }
}
```

注意：返回　果不　告知匹配了　些文　，　包含了整个文　本身：　示搜索　果　最　用　所需的全部信息。

接下来，　下搜索姓氏　Smith　的雇　。　此，我　将使用一个　高亮　搜索，很容易通　命令行完成。　个方法一般　及到一个　字符串　（*query-string*）　搜索，因　我　通　一个URL参数来　信息　搜索接口：

```
GET /megacorp/employee/_search?q=last_name:Smith
```

我　然在　求路径中使用 `_search` 端点，并将　本身　参数 `q=` 。返回　果　出了所有的 Smith：

```
{
   ...
   "hits": {
      "total":      2,
      "max_score":  0.30685282,
      "hits": [
         {
            ...
            "_source": {
               "first_name":  "John",
               "last_name":   "Smith",
               "age":         25,
               "about":       "I love to go rock climbing",
               "interests": [ "sports", "music" ]
            }
         },
         {
            ...
            "_source": {
               "first_name":  "Jane",
               "last_name":   "Smith",
               "age":         32,
               "about":       "I like to collect rock albums",
               "interests": [ "music" ]
            }
         }
      ]
   }
}
```

# 使用　表　式搜索

Query-string 搜索通　命令非常方便地　行　性的即席搜索，但它有自身的局限性（参　[search-lite]）。Elasticsearch 提供一个　富　活的　言叫做　表　式，它支持　建更加　和健壮的　。

域特定　言（DSL），指定了使用一个 JSON　求。我　可以像　重写之前的　所有 Smith 的搜索：

```
GET /megacorp/employee/_search
{
    "query" : {
        "match" : {
            "last_name" : "Smith"
        }
    }
}
```

返回 果与之前的 一 ，但 是可以看到有一些 化。其中之一是，不再使用 *query-string* 参数，而是一个 求体替代。 个 求使用 JSON 造，并使用了一个 `match` （属于 型之一，后 将会了解）。

## 更 的搜索

在 下更 的搜索。 同 搜索姓氏 Smith 的雇 ，但 次我 只需要年 大于 30 的。 需要作 整，使用 器 *filter* ，它支持高效地 行一个 化 。

```
GET /megacorp/employee/_search
{
    "query" : {
        "bool": {
            "must": {
                "match" : {
                    "last_name" : "smith" ①
                }
            },
            "filter": {
                "range" : {
                    "age" : { "gt" : 30 } ②
                }
            }
        }
    }
}
```

① 部分与我 之前使用的 `match` 一 。

② 部分是一个 `range` 器， 它能 到年 大于 30 的文 ，其中 `gt` 表示_大于_(*great than*)。

目前无需太多担心 法 ，后 会更 地介 。只需明 我 添加了一个 器 用于行一个 ，并 用之前的 `match` 。 在 果只返回了一个雇 ，叫 Jane Smith，32 。

```
{
   ...
   "hits": {
      "total":      1,
      "max_score":  0.30685282,
      "hits": [
         {
            ...
            "_source": {
               "first_name":  "Jane",
               "last_name":   "Smith",
               "age":         32,
               "about":       "I like to collect rock albums",
               "interests": [ "music" ]
            }
         }
      ]
   }
}
```

## 全文搜索

截止目前的搜索相 都很 ： 个姓名，通 年 。 在 下 微高 点儿的全文搜索——一 数据 很 定的任 。

搜索下所有喜 岩（rock climbing）的雇 ：

```
GET /megacorp/employee/_search
{
    "query" : {
        "match" : {
            "about" : "rock climbing"
        }
    }
}
```

然我 依旧使用之前的 `match` 在`about` 属性上搜索 ``rock climbing'' 。得到 个匹配的文 ：

```
{
   ...
   "hits": {
      "total":      2,
      "max_score":  0.16273327,
      "hits": [
         {
            ...
            "_score":         0.16273327, ①
            "_source": {
               "first_name":  "John",
               "last_name":   "Smith",
               "age":         25,
               "about":       "I love to go rock climbing",
               "interests": [ "sports", "music" ]
            }
         },
         {
            ...
            "_score":         0.016878016, ①
            "_source": {
               "first_name":  "Jane",
               "last_name":   "Smith",
               "age":         32,
               "about":       "I like to collect rock albums",
               "interests": [ "music" ]
            }
         }
      ]
   }
}
```

① 相　性得分

Elasticsearch　　按照相　性得分排序，即　个文　跟　　的匹配程度。第一个最高得分的　果很明：John Smith 的 about 属性清楚地写着 ``rock climbing" 。

但　什　Jane Smith 也作　　果返回了　？原因是　的 about 属性里提到了 rock'' 。因　只有 rock" 而没有 ``climbing" ，所以　的相　性得分低于 John 的。

　是一个很好的案例，　明了　Elasticsearch　如何　在　全文属性上搜索并返回相　性最　的果。Elasticsearch中的　相　性　概念非常重要，也是完全区　于　系型数据　的一个概念，数据　中的一条　要　匹配要　不匹配。

## 短　搜索

　出一个属性中的独立　是没有　的，但有　候想要精　匹配一系列　或者_短　_　。　比如，　我想　行　一个　，　匹配同　包含 rock''　和　climbing" ，并且　二者以短　``rock climbing" 的形式　挨着的雇　。

此 match 作 整，使用一个叫做 match_phrase 的 ：

```
GET /megacorp/employee/_search
{
    "query" : {
        "match_phrase" : {
            "about" : "rock climbing"
        }
    }
}
```

无 念，返回 果 有 John Smith 的文 。

```
{
   ...
   "hits": {
      "total":      1,
      "max_score":  0.23013961,
      "hits": [
         {
            ...
            "_score":         0.23013961,
            "_source": {
               "first_name":  "John",
               "last_name":   "Smith",
               "age":         25,
               "about":       "I love to go rock climbing",
               "interests": [ "sports", "music" ]
            }
         }
      ]
   }
}
```

## 高亮搜索

多 用都 向于在 个搜索 果中 高亮 部分文本片段，以便 用 知道 何 文 符合 条件。在 Elasticsearch 中 索出高亮片段也很容易。

再次 行前面的 ，并 加一个新的 highlight 参数：

```
GET /megacorp/employee/_search
{
    "query" : {
        "match_phrase" : {
            "about" : "rock climbing"
        }
    },
    "highlight": {
        "fields" : {
            "about" : {}
        }
    }
}
```

当 行 ，返回 果与之前一 ，与此同 果中 多了一个叫做 highlight 的部分。 个部分包含了 about 属性匹配的文本片段，并以 HTML  封装：

```
{
   ...
   "hits": {
      "total":      1,
      "max_score":  0.23013961,
      "hits": [
         {
            ...
            "_score":         0.23013961,
            "_source": {
               "first_name":  "John",
               "last_name":   "Smith",
               "age":         25,
               "about":       "I love to go rock climbing",
               "interests": [ "sports", "music" ]
            },
            "highlight": {
               "about": [
                  "I love to go <em>rock</em> <em>climbing</em>" ①
               ]
            }
         }
      ]
   }
}
```

① 原始文本中的高亮片段

于高亮搜索片段，可以在 {ref}/search-request-highlighting.html[highlighting reference documentation] 了解更多信息。

# 分析

于到了最后一个 需求：支持管理者 雇 目 做分析。 Elasticsearch 有一个功能叫聚合（aggregations），允 我 基于数据生成一些精 的分析 果。聚合与 SQL 中的 GROUP BY 似但更 大。

个例子， 掘出雇 中最受 迎的 趣 好：

```
GET /megacorp/employee/_search
{
  "aggs": {
    "all_interests": {
      "terms": { "field": "interests" }
    }
  }
}
```

忽略掉 法，直接看看 果：

```
{
   ...
   "hits": { ... },
   "aggregations": {
      "all_interests": {
         "buckets": [
            {
               "key":       "music",
               "doc_count": 2
            },
            {
               "key":       "forestry",
               "doc_count": 1
            },
            {
               "key":       "sports",
               "doc_count": 1
            }
         ]
      }
   }
}
```

可以看到， 位 工 音 感 趣，一位 林地感 趣，一位 感 趣。 些聚合并非 先 ，而是从匹配当前 的文 中即 生成。如果想知道叫 Smith 的雇 中最受 迎的 趣 好，可以直接添加 当的 来 合 ：

```
GET /megacorp/employee/_search
{
  "query": {
    "match": {
      "last_name": "smith"
    }
  },
  "aggs": {
    "all_interests": {
      "terms": {
        "field": "interests"
      }
    }
  }
}
```

all_interests 聚合已　　只包含匹配　　的文　：

```
  ...
  "all_interests": {
     "buckets": [
        {
           "key": "music",
           "doc_count": 2
        },
        {
           "key": "sports",
           "doc_count": 1
        }
     ]
  }
```

聚合　支持分　　。比如，　　特定　趣　好　工的平均年　：

```
GET /megacorp/employee/_search
{
    "aggs" : {
        "all_interests" : {
            "terms" : { "field" : "interests" },
            "aggs" : {
                "avg_age" : {
                    "avg" : { "field" : "age" }
                }
            }
        }
    }
}
```

得到的聚合 果有点儿 ，但理解起来 是很 的：

```
...
"all_interests": {
   "buckets": [
      {
         "key": "music",
         "doc_count": 2,
         "avg_age": {
            "value": 28.5
         }
      },
      {
         "key": "forestry",
         "doc_count": 1,
         "avg_age": {
            "value": 35
         }
      },
      {
         "key": "sports",
         "doc_count": 1,
         "avg_age": {
            "value": 25
         }
      }
   ]
}
```

出基本是第一次聚合的加 版。依然有一个 趣及数量的列表，只不 个 趣都有了一个附加的 `avg_age` 属性，代表有 个 趣 好的所有 工的平均年 。

即使 在不太理解 些 法也没有 系，依然很容易了解到 聚合及分 通 Elasticsearch 特性 得很完美。可提取的数据 型 无限制。

# 教程

欣喜的是， 是一个 于 Elasticsearch 基 描述的教程，且 是浅 止，更多 如 suggestions、geolocation、percolation、fuzzy 与 partial matching 等特性均被省略，以便保持教程的 。但它 突 了 始 建高 搜索功能多 容易。不需要配置——只需要添加数据并 始搜索！

很可能 法会 在某些地方有所困惑，并且 各个方面如何微 也有一些 。没 系！本 后 内容将 个 解 ， 全方位地理解 Elasticsearch 的工作原理。

# 分布式特性

在本章 ，我 提到 Elasticsearch 可以横向 展至数百（甚至数千）的服 器 点，同 可以理PB 数据。我 的教程 出了一些使用 Elasticsearch 的示例，但并不 及任何内部机制。Elasticsearch 天生就是分布式的，并且在 屏蔽了分布式的 性。

Elasticsearch 在分布式方面几乎是透明的。教程中并不要求了解分布式系 、分片、集群或其他的各 分布式概念。可以使用 本上的 点 松地 行教程里的程序，但如果 想要在 100 个 点的集群上 行程序，一切依然 。

Elasticsearch 尽可能地屏蔽了分布式系 的 性。 里列 了一些在后台自 行的操作：

- 分配文 到不同的容器 或 分片 中，文 可以 存在一个或多个 点中
- 按集群 点来均衡分配 些分片，从而 索引和搜索 程 行 均衡
- 制 个分片以支持数据冗余，从而防止硬件故障 致的数据 失
- 将集群中任一 点的 求路由到存有相 数据的 点
- 集群 容 无 整合新 点，重新分配分片以便从 群 点恢

当 本 ，将会遇到有 Elasticsearch 分布式特性的 充章 。 些章 将介 有 集群容、故障 移([distributed-cluster]) 、 文 存 ([distributed-docs]) 、 行分布式搜索([distributed-search])，以及分区（shard）及其工作原理([inside-a-shard])。

> 些章 并非必 ，完全可以无需了解内部机制就使用 Elasticsearch，但是它 将从 一个角度 助了解更完整的 Elasticsearch 知 。可以根据需要跳 它 ，或者想更完整地理解 再回 也无妨。

# 后

在 于通 Elasticsearch 能 什 的功能、以及上手的 易程度 有了初概念。Elasticsearch 力 通 最少的知 和配置做到 箱即用。学 Elasticsearch 的最好方式是投入 践：尽管 始索引和搜索 ！

然而， 于 Elasticsearch 知道得越多，就越有生 效率。告 Elasticsearch 越多的 域知，就越容易 行 果 。

本 的后 内容将 助 从新手成 家， 个章 不 述必要的基 知 ，而且包含 家建 。如果 上手， 些建 可能无法立竿 影；但 Elasticsearch 有着合理的 置，在无需干 的情况下通常都能工作得很好。当追求 秒 的性能提升 ，随 可以重温 些章 。