# 简介

Elasticsearch是一个开源的、实时的分布式搜索分析引擎， 它能让你以一个之前从未有过的速度和规模，去探索你的数据。 它被用作全文检索、结构化搜索、分析以及这三个功能的组合。

Elasticsearch是一个建立在全文搜索引擎 Apache Lucene™ 基础上的搜索引擎，可以说Lucene是当今最先进，最高效的全功能开源搜索引擎框架。

但是Lucene只是一个框架，要充分利用它的功能，需要使用JAVA，并且在程序中集成Lucene。需要很多的学习了解，才能明白它是如何运行的，Lucene确实非常复杂。

Elasticsearch使用Lucene作为内部引擎，但是在使用它做全文搜索时，只需要使用统一开发好的API即可，而不需要了解其背后复杂的Lucene的运行原理。

当然Elasticsearch并不仅仅是Lucene这么简单，它不但包括了全文搜索功能，还可以进行以下工作:

* 分布式实时文件存储，并将每一个字段都编入索引，使其可以被搜索。
* 实时分析的分布式搜索引擎。
* 可以扩展到上百台服务器，处理PB级别的结构化或非结构化数据。

Elasticsearch 将所有的功能打包成一个单独的服务，这样你可以通过程序与它提供的简单的 RESTful API 进行通信， 可以使用自己喜欢的编程语言充当 web 客户端，甚至可以使用命令行（去充当这个客户端）。

就 Elasticsearch 而言，起步很简单。对于初学者来说，它预设了一些适当的默认值，并隐藏了复杂的搜索理论知识。 它 *开箱即用* 。只需最少的理解，你很快就能具有生产力。

Elasticsearch提供RESTful的API，数据以JSON格式为基础。

Elasticsearch 中没有一个单独的组件是全新的或者是革命性的。全文搜索很久之前就已经可以做到了， 就像早就出现了的分析系统和分布式数据库。 革命性的成果在于将这些单独的，有用的组件融合到一个单一的、一致的、实时的应用中。它对于初学者而言有一个较低的门槛， 而当你的技能提升或需求增加时，它也始终能满足你的需求。

目前，有以下组织在使用Elasticsearch：

* Wikipedia 使用 Elasticsearch 提供带有高亮片段的全文搜索，还有 search-as-you-type 和 did-you-mean 的建议。
* 卫报 使用 Elasticsearch 将网络社交数据结合到访客日志中，实时的给它的编辑们提供公众对于新文章的反馈。
* Stack Overflow 将地理位置查询融入全文检索中去，并且使用 more-like-this 接口去查找相关的问题与答案。
* GitHub 使用 Elasticsearch 对1300亿行代码进行查询。

然而 Elasticsearch 不仅仅为巨头公司服务。它也帮助了很多初创公司，像 Datadog 和 Klout， 帮助他们将想法用原型实现，并转化为可扩展的解决方案。Elasticsearch 能运行在你的笔记本电脑上，或者扩展到上百台服务器上去处理PB级数据。

你可以免费下载，使用，修改 Elasticsearch。它在 [Apache 2 license](http://www.apache.org/licenses/LICENSE-2.0.html) 协议下发布的， 这是众多灵活的开源协议之一。Elasticsearch 的源码被托管在 Github 上 [github.com/elastic/elasticsearch](https://github.com/elastic/elasticsearch)。 如果你想加入我们这个令人惊奇的 contributors 社区，看这里 [Contributing to Elasticsearch](https://github.com/elastic/elasticsearch/blob/master/CONTRIBUTING.md)。

如果你对 Elasticsearch 有任何相关的问题，包括特定的特性(specific features)、语言客户端(language clients)、插件(plugins)，可以在这里 [discuss.elastic.co](https://discuss.elastic.co/) 加入讨论。

官网：https://www.elastic.co/products/elasticsearch

# 安装

## Elasticsearch

安装 Elasticsearch 之前，你需要先安装一个较新的版本的 Java。之后，你可以从 elastic 的官网 [elastic.co/downloads/elasticsearch](https://www.elastic.co/downloads/elasticsearch) 获取最新版本的 Elasticsearch。

要想安装 Elasticsearch，先下载并解压适合你操作系统的 Elasticsearch 版本。如果你想了解更多的信息， 可以查看 Elasticsearch 参考手册里边的安装部分，这边给出的链接指向安装说明 [Installation](https://www.elastic.co/guide/en/elasticsearch/reference/master/_installation.html)。

当你准备在生产环境安装 Elasticsearch 时，你可以在 [官网下载地址](http://www.elastic.co/downloads/elasticsearch) 找 到 Debian 或者 RPM 包，除此之外，你也可以使用官方支持的 [Puppet module](https://github.com/elasticsearch/puppet-elasticsearch) 或者 [Chef cookbook](https://github.com/elasticsearch/cookbook-elasticsearch)。

当你解压好了归档文件之后，Elasticsearch 已经准备好运行了。按照下面的操作，在前台(foregroud)启动 Elasticsearch：

cd elasticsearch-<version>

./bin/elasticsearch

如果你想把 Elasticsearch 作为一个守护进程在后台运行，那么可以在后面添加参数 -d。

如果你是在 Windows 上面运行 Elasticseach，你应该运行 bin\elasticsearch.bat 而不是 bin\elasticsearch 。

测试 Elasticsearch 是否启动成功，可以打开另一个终端，执行以下操作：

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

TIP：如果你是在 Windows 上面运行 Elasticsearch，你可以从 <http://curl.haxx.se/download.html> 中下载 cURL。 cURL 给你提供了一种将请求提交到 Elasticsearch 的便捷方式，并且安装 cURL 之后，你可以通过复制与粘贴去尝试书中的许多例子。

1、下载elasticsearch，解压，运行bin目录下的elasticsearch.bat

2、浏览器访问http://localhost:9200/

## sense

Sense 是一个 [Kibana](https://www.elastic.co/guide/en/kibana/current/index.html" \t "_top) 应用 它提供交互式的控制台，通过你的浏览器直接向 Elasticsearch 提交请求。 这本书的在线版本包含有一个 View in Sense 的链接，里面有许多代码示例。当点击的时候，它会打开一个代码示例的Sense控制台。 你不必安装 Sense，但是它允许你在本地的 Elasticsearch 集群上测试示例代码，从而使本书更具有交互性。

1. 下载Kibana：<https://www.elastic.co/downloads/kibana>

由于官方下载速度很慢，这里提供kibana 5.4版本的百度网盘下载地址：

http://pan.baidu.com/s/1dERuDWX 密码：ttty

1. 在kibana目录执行：./bin/kibana plugin --install elastic/sense，windows上bin\kibana.bat plugin --install elastic/sense
2. 启动kibana：./bin/kibana，windows上为：.bin\kibana.bat

4、浏览器访问： http://localhost:5601/app/sense

5、By default, Kibana 4.2 requires an active Elasticsearch instance of version 2.0 or higher. If you want to use Sense without this requirement see instructions here.

离线安装sense

你可以直接从这里 <https://download.elastic.co/elastic/sense/sense-latest.tar.gz> 下载 Sense 离线安装可以查看这里 [install it on an offline machine](https://www.elastic.co/guide/en/sense/current/installing.html#manual_download) 。

注意：启动kibana之前，先确认是否已经启动elasticsearch，否则无法成功启动。

ERROR unknown command plugin

上边的安装步骤适用于kibana5之前的版本，kibana5之后sense改为了Console，而且默认已经安装，如下：

解压之后，启动：

E:\04-opensource\kibana-5.4.1-windows-x86>bin\kibana.bat

log [12:11:28.138] [info][optimize] Optimizing and caching bundles for kibana, timelion and status\_page. This may take a few minutes

log [12:12:49.611] [info][optimize] Optimization of bundles for kibana, timelion and status\_page complete in 81.47 seconds

log [12:12:49.656] [info][status][plugin:kibana@5.4.1] Status changed from uninitialized to green - Ready

log [12:12:49.744] [info][status][plugin:elasticsearch@5.4.1] Status changed from uninitialized to yellow - Waiting for Elasticsearch

log [12:12:49.765] [info][status][plugin:console@5.4.1] Status changed from uninitialized to green - Ready

log [12:12:49.773] [info][status][plugin:metrics@5.4.1] Status changed from uninitialized to green - Ready

log [12:12:49.944] [info][status][plugin:timelion@5.4.1] Status changed from uninitialized to green - Ready

log [12:12:49.948] [info][listening] Server running at http://localhost:5601

log [12:12:49.950] [info][status][ui settings] Status changed from uninitialized to yellow - Elasticsearch plugin is yellow

log [12:12:50.102] [warning] You're running Kibana 5.4.1 with some different versions of Elasticsearch. Update Kibana or Elasticsearch to the same version to prevent compatibility issues: v5.4.0 @ 127.0.0.1:9200 (127.0.0.1)

log [12:12:55.289] [info][status][plugin:elasticsearch@5.4.1] Status changed from yellow to yellow - No existing Kibana index found

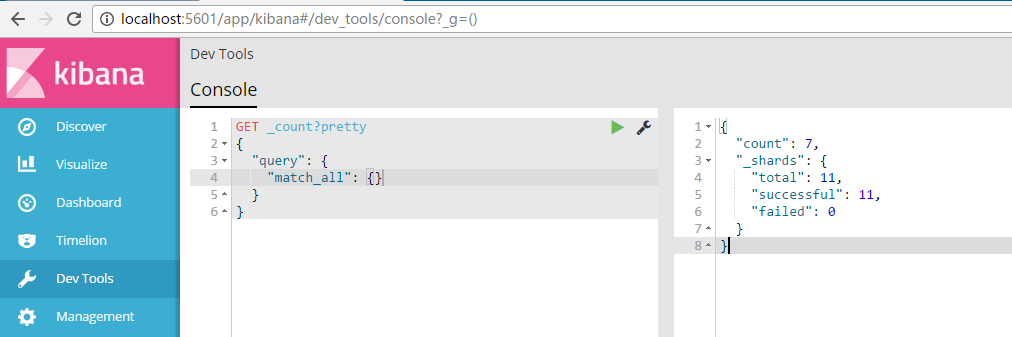
log [12:12:56.184] [info][status][plugin:elasticsearch@5.4.1] Status changed from yellow to green - Kibana index ready

log [12:12:56.186] [info][status][ui settings] Status changed from yellow to green – Ready

然后浏览器访问如下地址：

<http://localhost:5601>

点击左侧菜单的Dev Tools，既可以看到Console界面，点击Get Work按钮，进入Conosle工作界面，即可开始发送请求与ElasticSearch进行交互：



# 交互：

和 Elasticsearch 的交互方式取决于 你是否使用 Java。

Elasticsearch为以下语言提供了官方客户端 --Groovy、JavaScript、.NET、 PHP、 Perl、 Python 和 Ruby--还有很多社区提供的客户端和插件，所有这些都可以在 [Elasticsearch Clients](https://www.elastic.co/guide/en/elasticsearch/client/index.html) 中找到。

## JAVA API

如果你正在使用 Java，在代码中你可以使用 Elasticsearch内置的两个客户端：

### 节点客户端（Node client）

节点客户端作为一个非数据节点加入到本地集群中。换句话说，它本身不保存任何数据，但是它知道数据在集群中的哪个节点中，并且可以把请求转发到正确的节点。

### 传输客户端（Transport client）

轻量级的传输客户端可以将请求发送到远程集群。它本身不加入集群，但是它可以将请求转发到集群中的一个节点上。

两个 Java 客户端都是通过 9300 端口并使用本地 Elasticsearch 传输 协议和集群交互。集群中的节点通过端口 9300 彼此通信。如果这个端口没有打开，节点将无法形成一个集群。

Java 客户端作为节点必须和 Elasticsearch 有相同的 主要 版本；否则，它们之前将无法互相理解。

更多的 Java 客户端信息可以在 [Elasticsearch Clients](https://www.elastic.co/guide/en/elasticsearch/client/index.html) 中找到。

该部分内容我们将在后边深入学习，现在，我们看看如何使用CURL来与Elasticsearch的RESTful API进行交互。

## RESTful API with JSON over HTTP

所有其他语言可以使用 RESTful API 通过端口 9200 和 Elasticsearch 进行通信，你可以用你最喜爱的 web 客户端访问 Elasticsearch 。事实上，正如你所看到的，你甚至可以使用 curl 命令来和 Elasticsearch 交互。

一个 Elasticsearch 请求和任何 HTTP 请求一样由若干相同的部件组成：

curl -X<VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY\_STRING>' -d '<BODY>'

如上命令，其中使用尖括号括起来的部分详细解释如下：

|  |  |
| --- | --- |
| VERB | 适当的 HTTP *方法* 或 *谓词* : GET、 POST、 PUT、 HEAD 或者 DELETE。 |
| PROTOCOL | http 或者 https（如果你在 Elasticsearch 前面有一个 https 代理） |
| HOST | Elasticsearch 集群中任意节点的主机名，或者用 localhost 代表本地机器上的节点。 |
| PORT | 运行 Elasticsearch HTTP 服务的端口号，默认是 9200 。 |
| PATH | API 的终端路径（例如 \_count 将返回集群中文档数量）。Path 可能包含多个组件，例如：\_cluster/stats 和 \_nodes/stats/jvm 。 |
| QUERY\_STRING | 任意可选的查询字符串参数 (例如 ?pretty 将格式化地输出 JSON 返回值，使其更容易阅读) |
| BODY | 一个 JSON 格式的请求体 (如果请求需要的话) |

例如，计算集群中文档的数量，我们可以用这个:

curl -XGET 'http://localhost:9200/\_count?pretty' -d '

{

"query": {

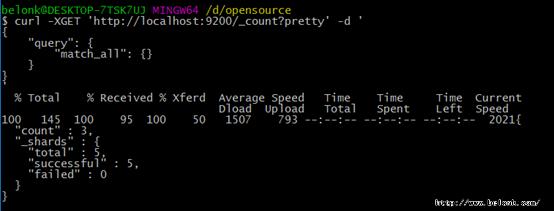
"match\_all": {}

}

}

'

执行情况如下：



如果在返回结果中我们需要看到HTTP头信息，需要使用**-i**参数，如下：

curl -i -XGET 'localhost:9200/'

Administrator@PC-201610162213 MINGW64 ~

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

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 327 100 327 0 0 2616 0 --:--:-- --:--:-- --:--:-- 2616HT TP/1.1 200 OK

content-type: application/json; charset=UTF-8

content-length: 327

{

"name" : "ZQuJdW6",

"cluster\_name" : "elasticsearch",

"cluster\_uuid" : "3kzN71MjSXG69l7O3CNaRQ",

"version" : {

"number" : "5.4.0",

"build\_hash" : "780f8c4",

"build\_date" : "2017-04-28T17:43:27.229Z",

"build\_snapshot" : false,

"lucene\_version" : "6.5.0"

},

"tagline" : "You Know, for Search"

}

## 缩写格式显示请求

Elasticsearch官方提供了一种缩写的请求标准格式，格式如下：

GET /\_count

{

"query": {

"match\_all": {}

}

}

官方推荐的sense工具直接支持以这种格式进行交互请求。sense介绍和安装见上一篇文章，这里不再赘述。

## 总结

本文介绍了如何使用CURL与Elasticsearch交互，这在初期学习Elasticsearch时很实用，更深入的JAVA客户端代码的交互方式我们放在后边来学习。

# 面向文档

Elasticsearch 是 面向文档 的，意味着它存储整个对象或 文档\_。Elasticsearch 不仅存储文档，而且 \_索引 每个文档的内容使之可以被检索。在 Elasticsearch中，你对文档进行索引、检索、排序和过滤--而不是对行列数据。这是一种完全不同的思考数据的方式，也是 Elasticsearch 能支持复杂全文检索的原因。

# JSON格式

Elasticsearch 使用 JavaScript Object Notation 或者 JSON 作为文档的序列化格式。JSON 序列化被大多数编程语言所支持，并且已经成为 NoSQL 领域的标准格式。 它简单、简洁、易于阅读。在 Elasticsearch 中将对象转化为 JSON 并做索引要比在一个扁平的表结构中做相同的事情简单的多。

# 实战

## 需求

我们受雇于 Megacorp 公司，作为 HR 部门新的 “热爱无人机” （\_"We love our drones!"\_）激励项目的一部分，我们的任务是为此创建一个雇员目录。该目录应当能培养雇员认同感及支持实时、高效、动态协作，因此有一些业务需求：

* 支持包含多值标签、数值、以及全文本的数据
* 检索任一雇员的完整信息
* 允许结构化搜索，比如查询 30 岁以上的员工
* 允许简单的全文搜索以及较复杂的短语搜索
* 支持在匹配文档内容中高亮显示搜索片段
* 支持基于数据创建和管理分析仪表盘

## 索引雇员文档

第一个业务需求就是存储雇员数据。 这将会以 雇员文档 的形式存储：一个文档代表一个雇员。存储数据到 Elasticsearch 的行为叫做 索引 ，但在索引一个文档之前，需要确定将文档存储在哪里。

一个 Elasticsearch 集群可以 包含多个 索引 ，相应的每个索引可以包含多个 类型 。 这些不同的类型存储着多个 文档 ，每个文档又有 多个 属性 。

### 索引的含义

索引（名词）：

如前所述，一个 索引 类似于传统关系数据库中的一个 数据库 ，是一个存储关系型文档的地方。 索引 (index) 的复数词为 indices 或 indexes 。

索引（动词）：

索引一个文档 就是存储一个文档到一个 索引 （名词）中以便它可以被检索和查询到。这非常类似于 SQL 语句中的 INSERT 关键词，除了文档已存在时新文档会替换旧文档情况之外。

倒排索引：

关系型数据库通过增加一个 索引 比如一个 B树（B-tree）索引 到指定的列上，以便提升数据检索速度。Elasticsearch 和 Lucene 使用了一个叫做 倒排索引 的结构来达到相同的目的。

+ 默认的，一个文档中的每一个属性都是 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。我们将在 倒排索引 讨论倒排索引的更多细节。

### 索引雇员信息

对于雇员目录，我们将做如下操作：

* 每个雇员索引一个文档，包含该雇员的所有信息。
* 每个文档都将是 employee 类型 。
* 该类型位于 索引 megacorp 内。
* 该索引保存在我们的 Elasticsearch 集群中。

实践中这非常简单（尽管看起来有很多步骤），我们可以通过一条命令完成所有这些动作：

PUT /megacorp/employee/1

{

"first\_name" : "John",

"last\_name" : "Smith",

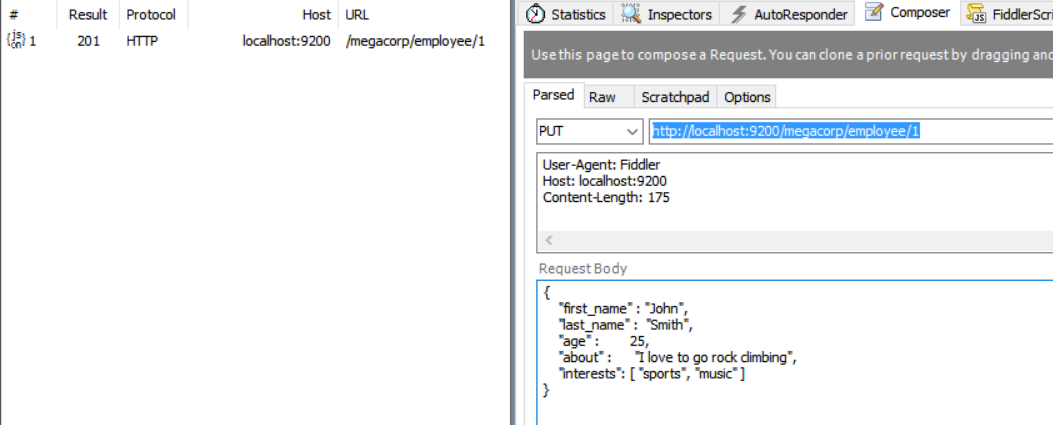
"age" : 25,

"about" : "I love to go rock climbing",

"interests": [ "sports", "music" ]

}

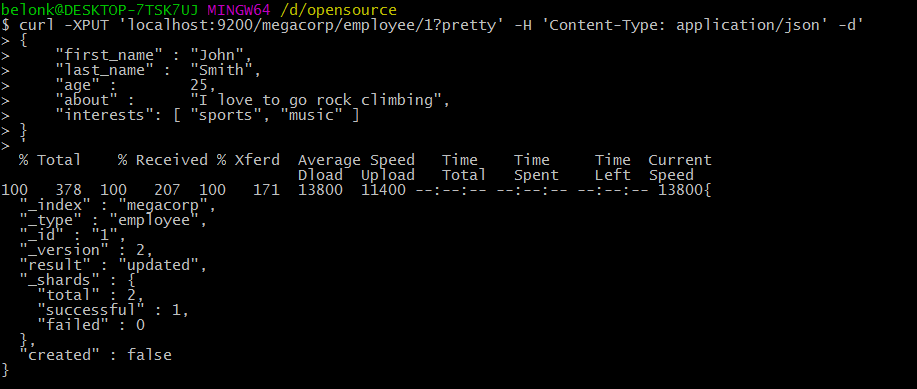
使用fiddler请求：



返回：

{"\_index":"megacorp","\_type":"employee","\_id":"1","\_version":1,"result":"created","\_shards":{"total":2,"successful":1,"failed":0},"created":true}

使用curl请求：



注意，路径 /megacorp/employee/1 包含了三部分的信息：

megacorp

索引名称

employee

类型名称

1

特定雇员的ID

请求体 —— JSON 文档 —— 包含了这位员工的所有详细信息，他的名字叫 John Smith ，今年 25 岁，喜欢攀岩。

很简单！无需进行执行管理任务，如创建一个索引或指定每个属性的数据类型之类的，可以直接只索引一个文档。Elasticsearch 默认地完成其他一切，因此所有必需的管理任务都在后台使用默认设置完成。

进行下一步前，让我们增加更多的员工信息到目录中：

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" ]

}

### 检索文档

目前我们已经在 Elasticsearch 中存储了一些数据， 接下来就能专注于实现应用的业务需求了。第一个需求是可以检索到单个雇员的数据。

这在 Elasticsearch 中很简单。简单地执行 一个 HTTP GET 请求并指定文档的地址——索引库、类型和ID。 使用这三个信息可以返回原始的 JSON 文档：

GET /megacorp/employee/1

$ curl -XGET 'localhost:9200/megacorp/employee/1?pretty'

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 294 100 294 0 0 18375 0 --:--:-- --:--:-- --:--:-- 287k{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "1",

"\_version" : 2,

"found" : true,

"\_source" : {

"first\_name" : "John",

"last\_name" : "Smith",

"age" : 25,

"about" : "I love to go rock climbing",

"interests" : [

"sports",

"music"

]

}

}

将 HTTP 命令由 PUT 改为 GET 可以用来检索文档，同样的，可以使用 DELETE 命令来删除文档，以及使用 HEAD 指令来检查文档是否存在。如果想更新已存在的文档，只需再次 PUT（最好按照RESTful标准，使用POST新增，PUT修改）。

belonk@DESKTOP-7TSK7UJ MINGW64 /d/opensource

$ curl -XPUT -i 'http://localhost:9200/megacorp/employee/3' -d '

{

"first\_name" : "Douglas",

"last\_name" : "Fir",

"age" : 35,

"about": "I like to build cabinets",

"interests": [ "forestry1" ]

}

'

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 315 100 146 100 169 9125 10562 --:--:-- --:--:-- --:--:-- 10562HTTP/1.1 200 OK

Warning: 299 Elasticsearch-5.4.0-780f8c4 "Content type detection for rest requests is deprecated. Specify the content type using the [Content-Type] header." "Fri, 02 Jun 2017 10:07:01 GMT"

content-type: application/json; charset=UTF-8

content-length: 146

{"\_index":"megacorp","\_type":"employee","\_id":"3","\_version":3,"result":"updated","\_shards":{"total":2,"successful":1,"failed":0},"created":false}

belonk@DESKTOP-7TSK7UJ MINGW64 /d/opensource

$ curl -XGET -i 'http://localhost:9200/megacorp/employee/3?pretty'

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 281 100 281 0 0 281 0 0:00:01 --:--:-- 0:00:01 274kHTTP/1.1 200 OK

content-type: application/json; charset=UTF-8

content-length: 281

{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "3",

"\_version" : 3,

"found" : true,

"\_source" : {

"first\_name" : "Douglas",

"last\_name" : "Fir",

"age" : 35,

"about" : "I like to build cabinets",

"interests" : [

"forestry1"

]

}

}

### 轻量搜索

一个 GET 是相当简单的，可以直接得到指定的文档。 现在尝试点儿稍微高级的功能，比如一个简单的搜索！

第一个尝试的几乎是最简单的搜索了。我们使用下列请求来搜索所有雇员：

GET /megacorp/employee/\_search

**拷贝为 CURL**[在 SENSE 中查看](http://localhost:5601/app/sense/?load_from=https://www.elastic.co/guide/cn/elasticsearch/guide/current/snippets/010_Intro/30_Simple_search.json)

可以看到，我们仍然使用索引库 megacorp 以及类型 employee`，但与指定一个文档 ID 不同，这次使用 `\_search 。返回结果包括了所有三个文档，放在数组 hits 中。一个搜索默认返回十条结果。

belonk@DESKTOP-7TSK7UJ MINGW64 /d/opensource

$ curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty'

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 1277 100 1277 0 0 8185 0 --:--:-- --:--:-- --:--:-- 8185{

"took" : 67,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 3,

"max\_score" : 1.0,

"hits" : [

{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "2",

"\_score" : 1.0,

"\_source" : {

"first\_name" : "Jane",

"last\_name" : "Smith",

"age" : 32,

"about" : "I like to collect rock albums",

"interests" : [

"music"

]

}

},

{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "1",

"\_score" : 1.0,

"\_source" : {

"first\_name" : "John",

"last\_name" : "Smith",

"age" : 25,

"about" : "I love to go rock climbing",

"interests" : [

"sports",

"music"

]

}

},

{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "3",

"\_score" : 1.0,

"\_source" : {

"first\_name" : "Douglas",

"last\_name" : "Fir",

"age" : 35,

"about" : "I like to build cabinets",

"interests" : [

"forestry1"

]

}

}

]

}

}

注意：返回结果不仅告知匹配了哪些文档，还包含了整个文档本身：显示搜索结果给最终用户所需的全部信息。

接下来，尝试下搜索姓氏为 Smith 的雇员。为此，我们将使用一个 *高亮* 搜索，很容易通过命令行完成。这个方法一般涉及到一个 *查询字符串* （\_query-string\_） 搜索，因为我们通过一个URL参数来传递查询信息给搜索接口：

GET /megacorp/employee/\_search?q=last\_name:Smith

我们仍然在请求路径中使用 \_search 端点，并将查询本身赋值给参数 q= 。返回结果给出了所有的 Smith：

belonk@DESKTOP-7TSK7UJ MINGW64 /d/opensource

$ curl -XGET 'localhost:9200/megacorp/employee/\_search?q=last\_name:Smith&pretty'

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 941 100 941 0 0 20021 0 --:--:-- --:--:-- --:--:-- 29406{

"took" : 29,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 2,

"max\_score" : 0.2876821,

"hits" : [

{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "2",

"\_score" : 0.2876821,

"\_source" : {

"first\_name" : "Jane",

"last\_name" : "Smith",

"age" : 32,

"about" : "I like to collect rock albums",

"interests" : [

"music"

]

}

},

{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "1",

"\_score" : 0.2876821,

"\_source" : {

"first\_name" : "John",

"last\_name" : "Smith",

"age" : 25,

"about" : "I love to go rock climbing",

"interests" : [

"sports",

"music"

]

}

}

]

}

}

### 使用查询表达式搜索

Elasticsearch 提供一个丰富灵活的查询语言叫做 *查询表达式* ， 它支持构建更加复杂和健壮的查询。

*领域特定语言* （DSL）， 指定了使用一个 JSON 请求。我们可以像这样重写之前的查询所有 Smith 的搜索 ：

GET /megacorp/employee/\_search

{

"query" : {

"match" : {

"last\_name" : "Smith"

}

}

}

返回结果与之前的查询一样，但还是可以看到有一些变化。其中之一是，不再使用 query-string 参数，而是一个请求体替代。这个请求使用 JSON 构造，并使用了一个 match 查询（属于查询类型之一，后续将会了解）。

$ curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type: application/json' -d'

> {

> "query" : {

> "match" : {

}

> "last\_name" : "Smith"

> }

> }

> }

> '

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 1031 100 940 100 91 15161 1467 --:--:-- --:--:-- --:--:-- 15161{

"took" : 4,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 2,

"max\_score" : 0.2876821,

"hits" : [

{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "2",

"\_score" : 0.2876821,

"\_source" : {

"first\_name" : "Jane",

"last\_name" : "Smith",

"age" : 32,

"about" : "I like to collect rock albums",

"interests" : [

"music"

]

}

},

{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "1",

"\_score" : 0.2876821,

"\_source" : {

"first\_name" : "John",

"last\_name" : "Smith",

"age" : 25,

"about" : "I love to go rock climbing",

"interests" : [

"sports",

"music"

]

}

}

]

}

}

### 更复杂的搜索

现在尝试下更复杂的搜索。 同样搜索姓氏为 Smith 的雇员，但这次我们只需要年龄大于 30 的。查询需要稍作调整，使用过滤器 filter ，它支持高效地执行一个结构化查询。

GET /megacorp/employee/\_search

{

"query" : {

"bool": {

"must": {

"match" : {

"last\_name" : "smith" // 这部分与我们之前使用的 match 查询 一样。

}

},

"filter": {

"range" : {

"age" : { "gt" : 30 } // 这部分是一个 range 过滤器 ， 它能找到年龄大于 30 的文档，其中 gt 表示\_大于(\_great than)。

}

}

}

}

}

$ curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type: application/json' -d'

> {

> "query" : {

> "bool": {

> "must": {

> "match" : {

> "last\_name" : "smith"

> }

> },

> "filter": {

> "range" : {

> "age" : { "gt" : 30 }

> }

> }

> }

> }

> }

> '

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 871 100 563 100 308 3608 1974 --:--:-- --:--:-- --:--:-- 3608{

"took" : 117,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 1,

"max\_score" : 0.2876821,

"hits" : [

{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "2",

"\_score" : 0.2876821,

"\_source" : {

"first\_name" : "Jane",

"last\_name" : "Smith",

"age" : 32,

"about" : "I like to collect rock albums",

"interests" : [

"music"

]

}

}

]

}

}

### 全文搜索

截止目前的搜索相对都很简单：单个姓名，通过年龄过滤。现在尝试下稍微高级点儿的全文搜索——一项传统数据库确实很难搞定的任务。

搜索下所有喜欢攀岩（rock climbing）的雇员：

GET /megacorp/employee/\_search

{

"query" : {

"match" : {

"about" : "rock climbing"

}

}

}

显然我们依旧使用之前的 match 查询在about 属性上搜索 “rock climbing” 。得到两个匹配的文档：

Administrator@PC-201610162213 MINGW64 ~

$ curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type: application/json' -d'

> {

> "query" : {

> "match" : {

> "about" : "rock climbing"

> }

> }

> }

> '

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 1039 100 944 100 95 29500 2968 --:--:-- --:--:-- --:--:-- 29500{

"took" : 17,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 2,

"max\_score" : 0.53484553,

"hits" : [

{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "1",

"\_score" : 0.53484553, // 相关性得分

"\_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" : 0.26742277, // 相关性得分

"\_source" : {

"first\_name" : "Jane",

"last\_name" : "Smith",

"age" : 32,

"about" : "I like to collect rock albums",

"interests" : [

"music"

]

}

}

]

}

}

Elasticsearch 默认按照相关性得分排序，即每个文档跟查询的匹配程度。第一个最高得分的结果很明显：John Smith 的 about 属性清楚地写着 “rock climbing” 。

但为什么 Jane Smith 也作为结果返回了呢？原因是她的 about 属性里提到了 “rock” 。因为只有 “rock” 而没有 “climbing” ，所以她的相关性得分低于 John 的。

这是一个很好的案例，阐明了 Elasticsearch 如何 *在* 全文属性上搜索并返回相关性最强的结果。Elasticsearch中的 *相关性* 概念非常重要，也是完全区别于传统关系型数据库的一个概念（数据库中的一条记录要么匹配要么不匹配）。

### 短语搜索

找出一个属性中的独立单词是没有问题的，但有时候想要精确匹配一系列单词或者短语 。 比如， 我们想执行这样一个查询，仅匹配同时包含 “rock” 和 “climbing” ，并且 二者以短语 “rock climbing” 的形式紧挨着的雇员记录。

为此对 match 查询稍作调整，使用一个叫做 match\_phrase 的查询：

GET /megacorp/employee/\_search

{

"query" : {

"match\_phrase" : {

"about" : "rock climbing"

}

}

}

毫无悬念，返回结果仅有 John Smith 的文档。

belonk@DESKTOP-7TSK7UJ MINGW64 ~

$ curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type: application/json' -d'

> {

> "query" : {

> "match\_phrase" : {

> "about" : "rock climbing"

> }

> }

> }

> '

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 684 100 582 100 102 18774 3290 --:--:-- --:--:-- --:--:-- 36375{

"took" : 2,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 1,

"max\_score" : 0.53484553,

"hits" : [

{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "1",

"\_score" : 0.53484553,

"\_source" : {

"first\_name" : "John",

"last\_name" : "Smith",

"age" : 25,

"about" : "I love to go rock climbing",

"interests" : [

"sports",

"music"

]

}

}

]

}

}

### 高亮搜索

高亮搜索即是在检索结果中将匹配的内容高亮显示。在 Elasticsearch 中检索出高亮片段也很容易：

GET /megacorp/employee/\_search

{

"query" : {

"match\_phrase" : {

"about" : "rock climbing"

}

},

"highlight": {

"fields" : {

"about" : {}

}

}

}

当执行该查询时，中还多了一个叫做 highlight 的部分。这个部分包含了 about 属性匹配的文本片段，并以 HTML 标签em封装：

belonk@DESKTOP-7TSK7UJ MINGW64 ~

$ curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type: application/json' -d'

> {

> "query" : {

> "match\_phrase" : {

> "about" : "rock climbing"

},

> }

> },

> "highlight": {

> "fields" : {

> "about" : {}

> }

> }

> }

> '

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 894 100 710 100 184 44375 11500 --:--:-- --:--:-- --:--:-- 693k{

"took" : 3,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 1,

"max\_score" : 0.53484553,

"hits" : [

{

"\_index" : "megacorp",

"\_type" : "employee",

"\_id" : "1",

"\_score" : 0.53484553,

"\_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>"

]

}

}

]

}

}

关于高亮搜索片段，可以在 [highlighting reference documentation](https://www.elastic.co/guide/en/elasticsearch/reference/master/search-request-highlighting.html) 了解更多信息。

## 分析

终于到了最后一个业务需求：支持管理者对雇员目录做分析。 Elasticsearch 有一个功能叫聚合（aggregations），允许我们基于数据生成一些精细的分析结果。聚合与 SQL 中的 GROUP BY 类似但更强大。

举个例子，挖掘出雇员中最受欢迎的兴趣爱好：

GET /megacorp/employee/\_search

{

"aggs": {

"all\_interests": {

"terms": { "field": "interests" }

}

}

}

暂时忽略掉语法，直接看看结果：

belonk@DESKTOP-7TSK7UJ MINGW64 ~

$ curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type: application/json' -d'

> {

> "aggs": {

> "all\_interests": {

> "terms": { "field": "interests" }

> }

> }

> }

> '

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 1751 100 1661 100 90 21025 1139 --:--:-- --:--:-- --:--:-- 21025{

"took" : 68,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 3,

"max\_score" : 1.0,

"hits" : [

…… // 此处省略命中结果

]

},

"aggregations" : {

"all\_interests" : {

"doc\_count\_error\_upper\_bound" : 0,

"sum\_other\_doc\_count" : 0,

"buckets" : [

{

"key" : "music",

"doc\_count" : 2

},

{

"key" : "forestry1",

"doc\_count" : 1

},

{

"key" : "sports",

"doc\_count" : 1

}

]

}

}

}

可以看到，两位员工对音乐感兴趣，一位对林地感兴趣，一位对运动感兴趣。这些聚合并非预先统计，而是从匹配当前查询的文档中即时生成。如果想知道叫 Smith 的雇员中最受欢迎的兴趣爱好，可以直接添加适当的查询来组合查询：

GET /megacorp/employee/\_search

{

"query": {

"match": {

"last\_name": "smith"

}

},

"aggs": {

"all\_interests": {

"terms": {

"field": "interests"

}

}

}

}

all\_interests 聚合已经变为只包含匹配查询的文档：

belonk@DESKTOP-7TSK7UJ MINGW64 ~

$ curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type: application/json' -d'

> {

> "query": {

> "match": {

> "last\_name": "smith"

> }

> },

> "aggs": {

> "all\_interests": {

> "terms": {

> "field": "interests"

> }

> }

> }

> }

> '

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 1417 100 1247 100 170 15987 2179 --:--:-- --:--:-- --:--:-- 15987{

"took" : 56,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

……

]

},

"aggregations" : {

"all\_interests" : {

"doc\_count\_error\_upper\_bound" : 0,

"sum\_other\_doc\_count" : 0,

"buckets" : [

{

"key" : "music",

"doc\_count" : 2

},

{

"key" : "sports",

"doc\_count" : 1

}

]

}

}

}

聚合还支持分级汇总 。比如，查询特定兴趣爱好员工的平均年龄：

GET /megacorp/employee/\_search

{

"aggs" : {

"all\_interests" : {

"terms" : { "field" : "interests" },

"aggs" : {

"avg\_age" : {

"avg" : { "field" : "age" }

}

}

}

}

}

得到的聚合结果有点儿复杂，但理解起来还是很简单的：

belonk@DESKTOP-7TSK7UJ MINGW64 ~

$ curl -XGET 'localhost:9200/megacorp/employee/\_search?pretty' -H 'Content-Type: application/json' -d'

> {

> "aggs" : {

> "all\_interests" : {

> "terms" : { "field" : "interests" },

> "aggs" : {

> "avg\_age" : {

> "avg" : { "field" : "age" }

> }

> }

> }

> }

> }

> '

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 2099 100 1853 100 246 59774 7935 --:--:-- --:--:-- --:--:-- 59774{

"took" : 23,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 3,

"max\_score" : 1.0,

"hits" : [

……

]

},

"aggregations" : {

"all\_interests" : {

"doc\_count\_error\_upper\_bound" : 0,

"sum\_other\_doc\_count" : 0,

"buckets" : [

{

"key" : "music",

"doc\_count" : 2,

"avg\_age" : {

"value" : 28.5

}

},

{

"key" : "forestry1",

"doc\_count" : 1,

"avg\_age" : {

"value" : 35.0

}

},

{

"key" : "sports",

"doc\_count" : 1,

"avg\_age" : {

"value" : 25.0

}

}

]

}

}

}

输出基本是第一次聚合的加强版。依然有一个兴趣及数量的列表，只不过每个兴趣都有了一个附加的 avg\_age 属性，代表有这个兴趣爱好的所有员工的平均年龄。

即使现在不太理解这些语法也没有关系，依然很容易了解到复杂聚合及分组通过 Elasticsearch 特性实现得很完美。可提取的数据类型毫无限制。

### Fielddata is disabled on text fields by default

在测试上边的分析代码时，出现了一个400的错误信息：

Fielddata is disabled on text fields by default. Set fielddata=true on [your\_field\_name] in order to load fielddata in memory by uninverting the inverted index. Note that this can however use significant memory.

#### 原因

官方的关于fielddata的文档如下：

<https://www.elastic.co/guide/en/elasticsearch/reference/current/fielddata.html>

Fielddata can consume a **lot** of heap space, especially when loading high cardinality text fields. Once fielddata has been loaded into the heap, it remains there for the lifetime of the segment. Also, loading fielddata is an expensive process which can cause users to experience latency hits. This is why fielddata is disabled by default.

If you try to sort, aggregate, or access values from a script on a text field, you will see this exception:

Fielddata is disabled on text fields by default. Set fielddata=true on [your\_field\_name] in order to load fielddata in memory by uninverting the inverted index. Note that this can however use significant memory.

大概的意思是说：

Fielddata会耗费大量的堆内存，尤其是在加载大文件时。一旦fielddata被加载到内存中，他会在segment的生命周期一直存在。而且，搜索时处理fielddata还会导致速度下降甚至严重延迟，所以默认fielddata被禁用。

如果你试图对text字段进行排序、聚合或者访问值等操作，你将会遇到如下异常：

Fielddata is disabled on text fields by default. Set fielddata=true on [your\_field\_name] in order to load fielddata in memory by uninverting the inverted index. Note that this can however use significant memory.

#### 解决办法

由于我们是在测试，需要对文本字段开启fielddata，需要使用 [PUT mapping API](https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-put-mapping.html)，如下命令:

PUT my\_index/\_mapping/my\_type

{

"properties": {

"my\_field": {

"type": "text",

"fielddata": true

}

}

}

其中my\_field字段是你需要进行排序、聚合等操作的包含现有映射的字段。

上边的实例我们开启fielddata如下：

belonk@DESKTOP-7TSK7UJ MINGW64 ~

$ curl -XPUT 'localhost:9200/megacorp/\_mapping/employee' -d '

{

"properties": {

"interests": {

"type": "text",

"fielddata": true

}

}

}'

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 123 100 21 100 102 338 1645 --:--:-- --:--:-- --:--:-- 1645{"acknowledged":true}

# 基本概念

## Near realtime(NRT)

## Cluster

## Node

## Index

## Type

## Document

## Shards & Replicas

# 基本操作

## 健康检查

## 创建索引

## 索引和查询文档

## 查询所有索引

## 删除索引

## 修改文档

## 删除文档

## 批量处理

# 数据检索

## 搜索API

## 查询语言

## 执行搜索

## 执行过滤

## 执行聚合

# 集群内的原理

ElasticSearch 的主旨是随时可用和按需扩容。 而扩容可以通过购买性能更强大（ 垂直扩容 ，或 纵向扩容） 或者数量更多的服务器（ 水平扩容 ，或 横向扩容 ）来实现。

虽然 Elasticsearch 可以获益于更强大的硬件设备，但是垂直扩容是有极限的。 真正的扩容能力是来自于水平扩容--为集群添加更多的节点，并且将负载压力和稳定性分散到这些节点中。

对于大多数的数据库而言，通常需要对应用程序进行非常大的改动，才能利用上横向扩容的新增资源。 与之相反的是，ElastiSearch天生就是 分布式的 ，它知道如何通过管理多节点来提高扩容性和可用性。 这也意味着你的应用无需关注这个问题。

本章将讲述如何按需配置集群、节点和分片，并在硬件故障时确保数据安全。

## 空集群

如果我们启动了一个单独的节点，里面不包含任何的数据和 索引，那我们的集群看起来就是一个 [图 1 “包含空内容节点的集群”](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_an-empty-cluster.html#img-cluster)。

**图 1. 包含空内容节点的集群**



一个运行中的 Elasticsearch 实例称为一个 节点，而集群是由一个或者多个拥有相同 cluster.name 配置的节点组成， 它们共同承担数据和负载的压力。当有节点加入集群中或者从集群中移除节点时，集群将会重新平均分布所有的数据。

当一个节点被选举成为 主 节点时， 它将负责管理集群范围内的所有变更，例如增加、删除索引，或者增加、删除节点等。 而主节点并不需要涉及到文档级别的变更和搜索等操作，所以当集群只拥有一个主节点的情况下，即使流量的增加它也不会成为瓶颈。 任何节点都可以成为主节点。我们的示例集群就只有一个节点，所以它同时也成为了主节点。

作为用户，我们可以将请求发送到 集群中的任何节点 ，包括主节点。 每个节点都知道任意文档所处的位置，并且能够将我们的请求直接转发到存储我们所需文档的节点。 无论我们将请求发送到哪个节点，它都能负责从各个包含我们所需文档的节点收集回数据，并将最终结果返回給客户端。 Elasticsearch 对这一切的管理都是透明的。

## 集群健康

Elasticsearch 的集群监控信息中包含了许多的统计数据，其中最为重要的一项就是 集群健康 ， 它在 status 字段中展示为 green 、 yellow 或者 red 。

GET /\_cluster/health

在一个不包含任何索引的空集群中，它将会有一个类似于如下所示的返回内容：

{

"cluster\_name": "elasticsearch",

"status": "green", https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png

"timed\_out": false,

"number\_of\_nodes": 1,

"number\_of\_data\_nodes": 1,

"active\_primary\_shards": 0,

"active\_shards": 0,

"relocating\_shards": 0,

"initializing\_shards": 0,

"unassigned\_shards": 0

}

|  |  |
| --- | --- |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/cluster-health.html#CO7-1) | status 字段是我们最关心的。 |

status 字段指示着当前集群在总体上是否工作正常。它的三种颜色含义如下：

green

所有的主分片和副本分片都正常运行。

yellow

所有的主分片都正常运行，但不是所有的副本分片都正常运行。

red

有主分片没能正常运行。

也可以使用\_cat命令来检查集群健康状况：

GET /\_cat/health?v

belonk@DESKTOP-7TSK7UJ MINGW64 ~

$ curl -XGET 'localhost:9200/\_cat/health?v&pretty'

epoch timestamp cluster status node.total node.data shards pri relo init unassign pending\_tasks max\_task\_wait\_time active\_shards\_percent

1496648088 15:34:48 my-application green 3 3 26 13 0 0 0 0 - 100.0%

列出集群中的节点：

GET /\_cat/nodes?v

belonk@DESKTOP-7TSK7UJ MINGW64 ~

$ curl -XGET 'localhost:9200/\_cat/nodes?v&pretty'

ip heap.percent ram.percent cpu load\_1m load\_5m load\_15m node.role master name

127.0.0.1 13 42 8 mdi - ZQuJdW6

127.0.0.1 15 42 9 mdi \* ZKbVKef

127.0.0.1 14 42 10 mdi - wwj6PD\_

在本章节剩余的部分，我们将解释什么是 主 分片和 副本 分片，以及上面提到的这些颜色的实际意义

## 添加索引

我们往 Elasticsearch 添加数据时需要用到 **索引 —— 保存相关数据的地方。 索引实际上是指向一个或者多个物理 分片 的 逻辑命名空间**。

一个 **分片 是一个底层的 工作单元 ，它仅保存了 全部数据中的一部分**。 在[分片内部机制](https://www.elastic.co/guide/cn/elasticsearch/guide/current/inside-a-shard.html)中，我们将详细介绍分片是如何工作的，而现在我们只需知道**一个分片是一个 Lucene 的实例，以及它本身就是一个完整的搜索引擎**。 我们的文档被存储和索引到分片内，但是应用程序是直接与索引而不是与分片进行交互。

Elasticsearch 是利用分片将数据分发到集群内各处的。**分片是数据的容器，文档保存在分片内，分片又被分配到集群内的各个节点里**。 当你的集群规模扩大或者缩小时， Elasticsearch 会自动的在各节点中迁移分片，使得数据仍然均匀分布在集群里。

一个分片可以是 **主 分片或者 副本 分片**。**索引内任意一个文档都归属于一个主分片，所以主分片的数目决定着索引能够保存的最大数据量**。

注意

技术上来说，一个主分片最大能够存储 Integer.MAX\_VALUE - 128 个文档，但是实际最大值还需要参考你的使用场景：包括你使用的硬件， 文档的大小和复杂程度，索引和查询文档的方式以及你期望的响应时长。

**一个副本分片只是一个主分片的拷贝**。 副本分片作为硬件故障时保护数据不丢失的**冗余备份**，并为搜索和返回文档等**读操作**提供服务。

在索引建立的时候就已经确定了主分片数，但是副本分片数可以随时修改。

让我们在包含一个空节点的集群内创建名为 blogs 的索引。 索引在默认情况下会被分配5个主分片， 但是为了演示目的，我们将分配3个主分片和一份副本（每个主分片拥有一个副本分片）：

PUT /blogs

{

"settings" : {

"number\_of\_shards" : 3,

"number\_of\_replicas" : 1

}

}

belonk@DESKTOP-7TSK7UJ MINGW64 ~

$ curl -XPUT 'localhost:9200/blogs?pretty' -H 'Content-Type: application/json' -d'

> {

> "settings" : {

> "number\_of\_shards" : 3,

> "number\_of\_replicas" : 1

> }

> }

> '

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 149 100 60 100 89 256 380 --:--:-- --:--:-- --:--:-- 380{

"acknowledged" : true,

"shards\_acknowledged" : true

}

我们的集群现在是[图 2 “拥有一个索引的单节点集群”](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_add-an-index.html#cluster-one-node)。所有3个主分片都被分配在 Node 1 。

**图 2. 拥有一个索引的单节点集群**



如果我们现在查看[集群健康](https://www.elastic.co/guide/cn/elasticsearch/guide/current/cluster-health.html)， 我们将看到如下内容：

{

"cluster\_name": "elasticsearch",

"status": "yellow", https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png

"timed\_out": false,

"number\_of\_nodes": 1,

"number\_of\_data\_nodes": 1,

"active\_primary\_shards": 3,

"active\_shards": 3,

"relocating\_shards": 0,

"initializing\_shards": 0,

"unassigned\_shards": 3, https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/2.png

"delayed\_unassigned\_shards": 0,

"number\_of\_pending\_tasks": 0,

"number\_of\_in\_flight\_fetch": 0,

"task\_max\_waiting\_in\_queue\_millis": 0,

"active\_shards\_percent\_as\_number": 50

}

|  |  |
| --- | --- |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_add-an-index.html#CO8-1) | 集群 status 值为 yellow 。 |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/2.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_add-an-index.html#CO8-2) | 没有被分配到任何节点的副本数。 |

belonk@DESKTOP-7TSK7UJ MINGW64 ~

$ curl -XGET 'localhost:9200/\_cluster/health?pretty'

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 469 100 469 0 0 29312 0 --:--:-- --:--:-- --:--:-- 458k{

"cluster\_name" : "elasticsearch",

"status" : "yellow",

"timed\_out" : false,

"number\_of\_nodes" : 1,

"number\_of\_data\_nodes" : 1,

"active\_primary\_shards" : 13,

"active\_shards" : 13,

"relocating\_shards" : 0,

"initializing\_shards" : 0,

"unassigned\_shards" : 13,

"delayed\_unassigned\_shards" : 0,

"number\_of\_pending\_tasks" : 0,

"number\_of\_in\_flight\_fetch" : 0,

"task\_max\_waiting\_in\_queue\_millis" : 0,

"active\_shards\_percent\_as\_number" : 50.0

}

集群的健康状况为 yellow 则表示全部 主 分片都正常运行（集群可以正常服务所有请求），但是 副本 分片没有全部处在正常状态。 实际上，所有3个副本分片都是 unassigned —— 它们都没有被分配到任何节点。 在同一个节点上既保存原始数据又保存副本是没有意义的，因为一旦失去了那个节点，我们也将丢失该节点上的所有副本数据。

当前我们的集群是正常运行的，但是在硬件故障时有丢失数据的风险。

## 添加故障转移

当集群中只有一个节点在运行时，意味着会有一个单点故障问题——没有冗余。 幸运的是，我们只需再启动一个节点即可防止数据丢失。

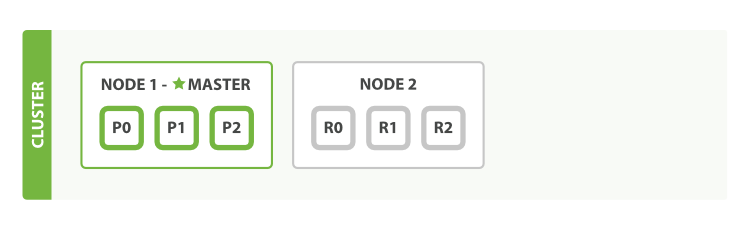
**启动第二个节点**

为了测试第二个节点启动后的情况，你可以在同一个目录内，完全依照启动第一个节点的方式来启动一个新节点（参考[安装并运行 Elasticsearch](https://www.elastic.co/guide/cn/elasticsearch/guide/current/running-elasticsearch.html)）。多个节点可以共享同一个目录。

当你在同一台机器上启动了第二个节点时，只要它和第一个节点有同样的 cluster.name 配置，它就会自动发现集群并加入到其中。 但是在不同机器上启动节点的时候，为了加入到同一集群，你需要配置一个可连接到的单播主机列表。 详细信息请查看[最好使用单播代替组播](https://www.elastic.co/guide/cn/elasticsearch/guide/current/important-configuration-changes.html#unicast)

如果启动了第二个节点，我们的集群将会如[图 3 “拥有两个节点的集群——所有主分片和副本分片都已被分配”](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_add_failover.html#cluster-two-nodes)所示。

**图 3. 拥有两个节点的集群——所有主分片和副本分片都已被分配**



当第二个节点加入到集群后，3个 副本分片 将会分配到这个节点上——每个主分片对应一个副本分片。 这意味着当集群内任何一个节点出现问题时，我们的数据都完好无损。

所有新近被索引的文档都将会保存在主分片上，然后被并行的复制到对应的副本分片上。这就保证了我们既可以从主分片又可以从副本分片上获得文档。

cluster-health 现在展示的状态为 green ，这表示所有6个分片（包括3个主分片和3个副本分片）都在正常运行。

{

"cluster\_name": "elasticsearch",

"status": "green", https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png

"timed\_out": false,

"number\_of\_nodes": 2,

"number\_of\_data\_nodes": 2,

"active\_primary\_shards": 3,

"active\_shards": 6,

"relocating\_shards": 0,

"initializing\_shards": 0,

"unassigned\_shards": 0,

"delayed\_unassigned\_shards": 0,

"number\_of\_pending\_tasks": 0,

"number\_of\_in\_flight\_fetch": 0,

"task\_max\_waiting\_in\_queue\_millis": 0,

"active\_shards\_percent\_as\_number": 100

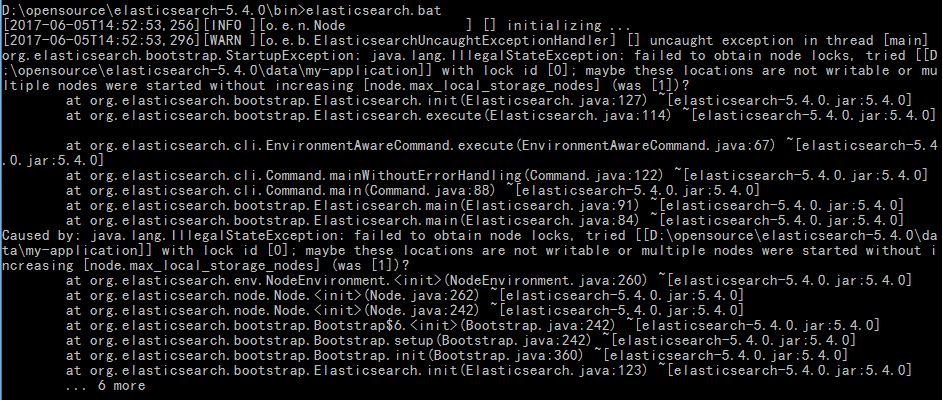
}

|  |  |
| --- | --- |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_add_failover.html#CO9-1) | 集群 status 值为 green 。 |

我们的集群现在不仅仅是正常运行的，并且还处于 始终可用 的状态。

### [elasticsearch start more than one nodes on the same data folder](http://blog.csdn.net/the_conquer_zzy/article/details/72818896)

在同一个目录下启动第二个节点时，出现如下异常：



解决方法是：在 config/elasticsearch.yml文件中新增一个配置变量：

node.max\_local\_storage\_nodes: 10

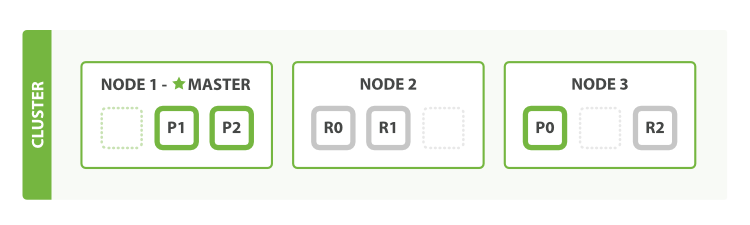
配置一个大于1的整数

[应对故障  »](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_coping_with_failure.html)

## 水平扩容

怎样为我们的正在增长中的应用程序按需扩容呢？ 当启动了第三个节点，我们的集群将会看起来如[图 4 “拥有三个节点的集群——为了分散负载而对分片进行重新分配”](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_scale_horizontally.html#cluster-three-nodes)所示。

**图 4. 拥有三个节点的集群——为了分散负载而对分片进行重新分配**



Node 1 和 Node 2 上各有一个分片被迁移到了新的 Node 3 节点，现在每个节点上都拥有2个分片，而不是之前的3个。 这表示每个节点的硬件资源（CPU, RAM, I/O）将被更少的分片所共享，每个分片的性能将会得到提升。

分片是一个功能完整的搜索引擎，它拥有使用一个节点上的所有资源的能力。 我们这个拥有6个分片（3个主分片和3个副本分片）的索引可以最大扩容到6个节点，每个节点上存在一个分片，并且每个分片拥有所在节点的全部资源。

## 更多的扩容

但是如果我们想要扩容超过6个节点怎么办呢？

主分片的数目在索引创建时 就已经确定了下来。实际上，这个数目定义了这个索引能够 存储 的最大数据量。（实际大小取决于你的数据、硬件和使用场景。） 但是，读操作——搜索和返回数据——可以同时被主分片 或 副本分片所处理，所以当你拥有越多的副本分片时，也将拥有越高的吞吐量。

在运行中的集群上是可以动态调整副本分片数目的 ，我们可以按需伸缩集群。让我们把副本数从默认的 1增加到 2 ：

PUT /blogs/\_settings

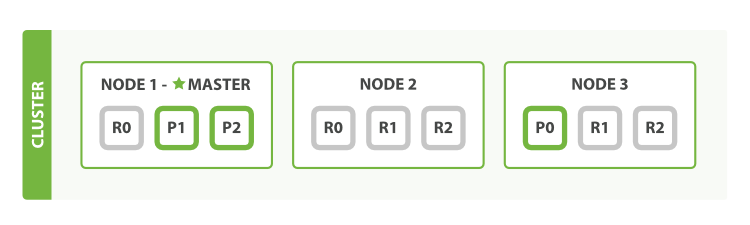
{

"number\_of\_replicas" : 2

}

如[图 5 “将参数 number\_of\_replicas 调大到 2”](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_scale_horizontally.html" \l "cluster-three-nodes-two-replicas" \o "图 5. 将参数 number_of_replicas 调大到 2)所示， blogs 索引现在拥有9个分片：3个主分片和6个副本分片。 这意味着我们可以将集群扩容到9个节点，每个节点上一个分片。相比原来3个节点时，集群搜索性能可以提升 3 倍。

**图 5. 将参数 number\_of\_replicas 调大到 2**



注意

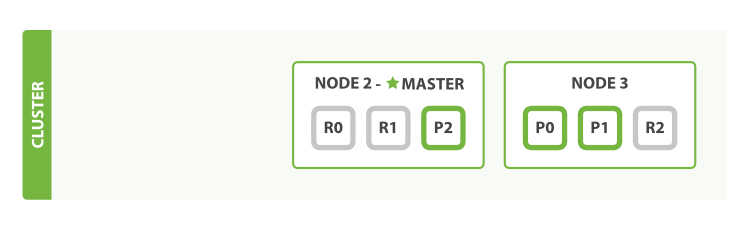
当然，如果只是在相同节点数目的集群上增加更多的副本分片并不能提高性能，因为每个分片从节点上获得的资源会变少。 你需要增加更多的硬件资源来提升吞吐量。

但是更多的副本分片数提高了数据冗余量：按照上面的节点配置，我们可以在失去2个节点的情况下不丢失任何数据。

## 应对故障

我们之前说过 Elasticsearch 可以应对节点故障，接下来让我们尝试下这个功能。 如果我们关闭第一个节点，这时集群的状态为[图 6 “关闭了一个节点后的集群”](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_coping_with_failure.html#cluster-post-kill)

**图 6. 关闭了一个节点后的集群**



我们关闭的节点是一个主节点。而集群必须拥有一个主节点来保证正常工作，所以发生的第一件事情就是选举一个新的主节点： Node 2 。

在我们关闭 Node 1 的同时也失去了主分片 1 和 2 ，并且在缺失主分片的时候索引也不能正常工作。 如果此时来检查集群的状况，我们看到的状态将会为 red ：不是所有主分片都在正常工作。

幸运的是，在其它节点上存在着这两个主分片的完整副本， 所以新的主节点立即将这些分片在 Node 2 和 Node 3 上对应的副本分片提升为主分片， 此时集群的状态将会为 yellow 。 这个提升主分片的过程是瞬间发生的，如同按下一个开关一般。

为什么我们集群状态是 yellow 而不是 green 呢？ 虽然我们拥有所有的三个主分片，但是同时设置了每个主分片需要对应2份副本分片，而此时只存在一份副本分片。 所以集群不能为 green 的状态，不过我们不必过于担心：如果我们同样关闭了 Node 2 ，我们的程序 依然 可以保持在不丢任何数据的情况下运行，因为 Node 3 为每一个分片都保留着一份副本。

如果我们重新启动 Node 1 ，集群可以将缺失的副本分片再次进行分配，那么集群的状态也将如[图 5 “将参数 number\_of\_replicas 调大到 2”](https://www.elastic.co/guide/cn/elasticsearch/guide/current/_scale_horizontally.html#cluster-three-nodes-two-replicas)所示。 如果 Node 1 依然拥有着之前的分片，它将尝试去重用它们，同时仅从主分片复制发生了修改的数据文件。

到目前为止，你应该对分片如何使得 Elasticsearch 进行水平扩容以及数据保障等知识有了一定了解。 接下来我们将讲述关于分片生命周期的更多细节。