# **[Elasticsearch 快速开始](https://www.cnblogs.com/cjsblog/p/9439331.html)**

Elasticsearch 是一个分布式的 RESTful 风格的搜索和数据分析引擎。

* 查询 ： Elasticsearch 允许执行和合并多种类型的搜索 — 结构化、非结构化、地理位置、度量指标 — 搜索方式随心而变。
* 分析 ： 找到与查询最匹配的十个文档是一回事。但是如果面对的是十亿行日志，又该如何解读呢？Elasticsearch 聚合让您能够从大处着眼，探索数据的趋势和模式。
* 速度 ： Elasticsearch 很快。真的，真的很快。
* 可扩展性 ： 可以在笔记本电脑上运行。 也可以在承载了 PB 级数据的成百上千台服务器上运行。
* 弹性 ： Elasticsearch 运行在一个分布式的环境中，从设计之初就考虑到了这一点。
* 灵活性 ： 具备多个案例场景。数字、文本、地理位置、结构化、非结构化。所有的数据类型都欢迎。
* HADOOP & SPARK ： Elasticsearch + Hadoop

# **准备开始**

Elasticsearch是一个高度可伸缩的开源全文搜索和分析引擎。它允许您快速和接近实时地存储、搜索和分析大量数据。

这里有一些使用Elasticsearch的用例：

1. 你经营一个网上商店，你允许你的顾客搜索你卖的产品。在这种情况下，您可以使用Elasticsearch来存储整个产品目录和库存，并为它们提供搜索和自动完成建议。
2. 你希望收集日志或事务数据，并希望分析和挖掘这些数据，以查找趋势、统计、汇总或异常。在这种情况下，你可以使用loghide (Elasticsearch/ loghide /Kibana堆栈的一部分)来收集、聚合和解析数据，然后让loghide将这些数据输入到Elasticsearch中。一旦数据在Elasticsearch中，你就可以运行搜索和聚合来挖掘你感兴趣的任何信息。
3. 你运行一个价格警报平台，允许精通价格的客户指定如下规则:“我有兴趣购买特定的电子设备，如果下个月任何供应商的产品价格低于X美元，我希望得到通知”。在这种情况下，你可以抓取供应商的价格，将它们推入到Elasticsearch中，并使用其反向搜索(Percolator)功能来匹配价格走势与客户查询，并最终在找到匹配后将警报推送给客户。
4. 你有分析/业务智能需求，并希望快速调查、分析、可视化，并对大量数据提出特别问题(想想数百万或数十亿的记录)。在这种情况下，你可以使用Elasticsearch来存储数据，然后使用Kibana (Elasticsearch/ loghide /Kibana堆栈的一部分)来构建自定义仪表板，以可视化对您来说很重要的数据的各个方面。此外，还可以使用Elasticsearch聚合功能对数据执行复杂的业务智能查询。

# **基本概念**

**Near Realtime (NRT)**

Elasticsearch是一个近乎实时的搜索平台。这意味着从索引文档到可以搜索的时间只有轻微的延迟（通常是1秒）。

**Cluster**

集群是一个或多个节点(服务器)的集合，它们共同保存你的整个数据，并提供跨所有节点的联合索引和搜索功能。一个集群由一个唯一的名称标识，默认这个唯一标识的名称是"elasticsearch"。这个名称很重要，因为如果节点被设置为按其名称加入集群，那么节点只能是集群的一部分。

确保不要在不同的环境中用相同的集群名称，否则可能导致节点加入到错误的集群中。例如，你可以使用"logging-dev", "logging-test", "logging-prod"分别用于开发、测试和正式集群的名字。

**Node**

节点是一个单独的服务器，它是集群的一部分，存储数据，并参与集群的索引和搜索功能。就像集群一样，节点由一个名称来标识，默认情况下，该名称是在启动时分配给节点的随机通用唯一标识符(UUID)。如果不想用默认的节点名，可以定义任何想要的节点名。这个名称对于管理来说很重要，因为你希望识别网络中的哪些服务器对应于你的Elasticsearch集群中的哪些节点。

一个节点可以通过配置集群名称来加入到一个特定的集群中。默认情况下，每个节点都被设置加入到一个名字叫"elasticsearch"的集群中，这就意味着如果你启动了很多个节点，并且假设它们彼此可以互相发现，那么它们将自动形成并加入到一个名为"elasticsearch"的集群中。

一个集群可以有任意数量的节点。此外，如果在你的网络上当前没有运行任何节点，那么此时启动一个节点将默认形成一个单节点的名字叫"elasticsearch"的集群。

**Index**

索引是具有某种相似特征的文档的集合。例如，你可以有一个顾客数据索引，产品目录索引和订单数据索引。索引有一个名称（必须是小写的）标识，该名称用于在对其中的文档执行索引、搜索、更新和删除操作时引用索引。

**Document**

文档是可以被索引的基本信息单元。文档用JSON表示。

**Shards & Replicas**

一个索引可能存储大量数据，这些数据可以超过单个节点的硬件限制。例如，一个包含10亿条文档占用1TB磁盘空间的索引可能不适合在单个节点上，或者可能太慢而不能单独处理来自单个节点的搜索请求。

为了解决这个问题，Elasticsearch提供了将你的索引细分为多个碎片（或者叫分片）的能力。在创建索引时，可以简单地定义所需的分片数量。每个分片本身就是一个功能完全独立的“索引”，可以驻留在集群中的任何节点上。

分片之所以重要，主要有两个原因：

* 它允许你水平地分割/扩展内容卷
* 它允许你跨分片（可能在多个节点上）分布和并行操作，从而提高性能和吞吐量

在一个网络/云环境中随时都有可能出现故障，强烈推荐你有一个容灾机制。Elasticsearch允许你将一个或者多个索引分片复制到其它地方，这被称之为副本。

复制之所以重要，有两个主要原因：

* 它提供了在一个shard/node失败是的高可用性。出于这个原因，很重要的一个点是一个副本从来不会被分配到与它复制的原始分片相同节点上。也就是说，副本是放到另外的节点上的。
* 它允许扩展搜索量/吞吐量，因为搜索可以在所有副本上并行执行。

总而言之，每个索引都可以分割成多个分片。索引也可以被复制零(意味着没有副本)或更多次。一旦被复制，每个索引都将具有主分片(被复制的原始分片)和副本分片(主分片的副本)。在创建索引时，可以为每个索引定义分片和副本的数量。创建索引后，您可以随时动态地更改副本的数量，但不能更改事后分片的数量。

在默认情况下，Elasticsearch中的每个索引都分配了5个主分片和1个副本，这意味着如果集群中至少有两个节点，那么索引将有5个主分片和另外5个副本分片（PS：这5个副本分片组成1个完整副本），每个索引总共有10个分片。

（画外音：副本是针对索引而言的，同时需要注意索引和节点数量没有关系，我们说2个副本指的是索引被复制了2次，而1个索引可能由5个分片组成，那么在这种情况下，集群中的分片数应该是 5 × (1 + 2) = 15 ）

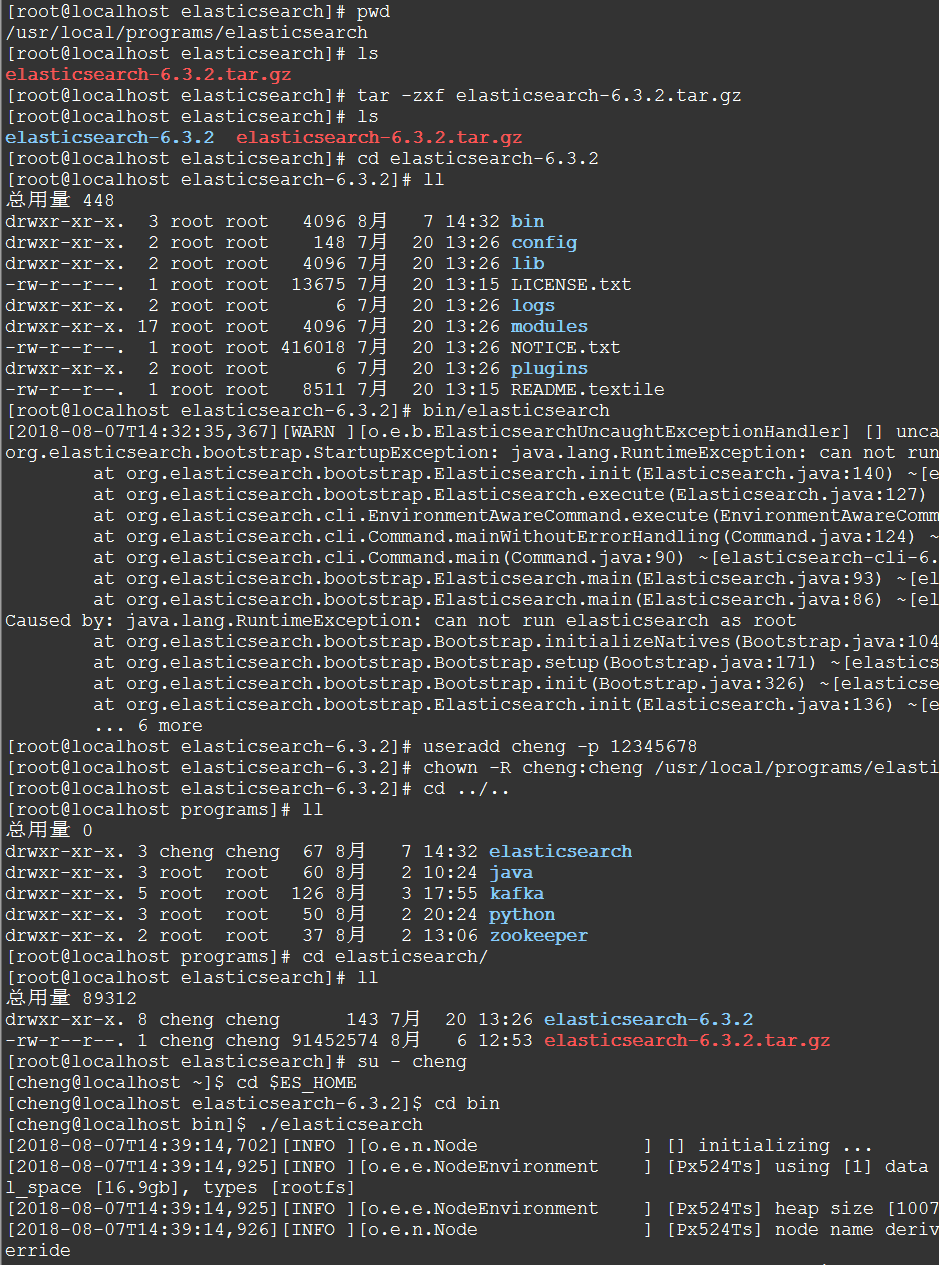
# **安装**

tar -zxf elasticsearch-6.3.2.tar.gz

cd elasticsearch-6.3.2/bin

./elasticsearch

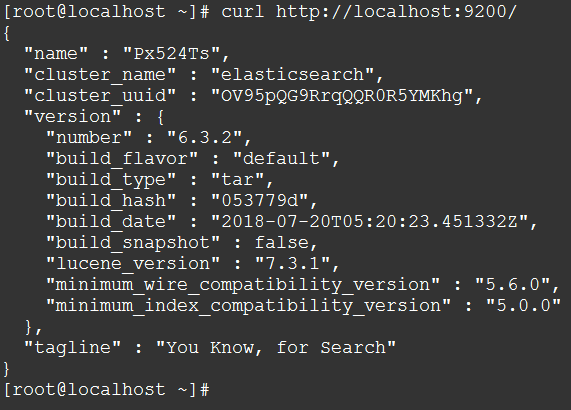
注意：不能以root用户运行elasticsearch

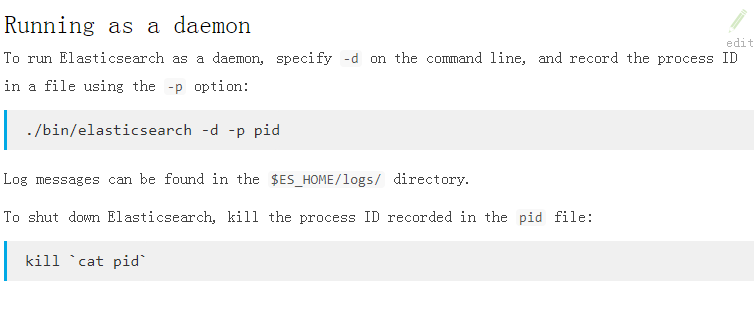


By default, Elasticsearch uses port 9200 to provide access to its REST API. This port is configurable if necessary.

检查Elasticsearch是否正在运行：

curl http://localhost:9200/





# **The REST API**

## **集群健康**

请求：

curl -X GET "localhost:9200/\_cat/health?v"

响应：

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

1533625274 15:01:14 elasticsearch green 1 1 0 0 0 0 0 0 - 100.0%

我们可以看到，我们命名为“elasticsearch”的集群现在是green状态。

无论何时我们请求集群健康时，我们会得到green, yellow, 或者 red 这三种状态。

* Green ： everything is good（一切都很好）（所有功能正常）
* Yellow ： 所有数据都是可用的，但有些副本还没有分配（所有功能正常）
* Red ： 有些数据不可用（部分功能正常）

从上面的响应中我们可以看到，集群"elasticsearch"总共有1个节点，0个分片因为还没有数据。

下面看一下集群的节点列表：

请求：

curl -X GET "localhost:9200/\_cat/nodes?v"

响应：

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

127.0.0.1 15 53 0 0.03 0.03 0.05 mdi \* Px524Ts

可以看到集群中只有一个节点，它的名字是“Px524Ts”

## **查看全部索引**

请求：

curl -X GET "localhost:9200/\_cat/indices?v"

响应：

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

上面的输出意味着：我们在集群中没有索引

## **创建一个索引**

现在，我们创建一个名字叫“customer”的索引，然后查看索引：

请求：

curl -X PUT "localhost:9200/customer?pretty"

（画外音：pretty的意思是响应（如果有的话）以JSON格式返回）

响应：

{

"acknowledged" : true,

"shards\_acknowledged" : true,

"index" : "customer"

}

请求：

curl -X GET "localhost:9200/\_cat/indices?v"

响应：

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

yellow open customer rG5fxdruTNmD-bdYIF5zOg 5 1 0 0 1.1kb 1.1kb

结果的第二行告诉我们，我们现在有叫"customer"的索引，并且他有5个主分片和1个副本（默认是1个副本），有0个文档。

可能你已经注意到这个"customer"索引的健康状态是yellow。回想一下我们之前的讨论，yellow意味着一些副本(尚未)被分配。

之所以会出现这种情况，是因为Elasticsearch默认情况下为这个索引创建了一个副本。由于目前我们只有一个节点在运行，所以直到稍后另一个节点加入集群时，才会分配一个副本(对于高可用性)。一旦该副本分配到第二个节点上，该索引的健康状态将变为green。

## **索引并查询一个文档**

现在，让我们put一些数据到我们的"customer"索引：

请求：

curl -X PUT "localhost:9200/customer/\_doc/1?pretty" -H 'Content-Type: application/json' -d'{"name": "John Doe"}'

响应：

[IMG_259](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

{

"\_index" : "customer",

"\_type" : "\_doc",

"\_id" : "1",

"\_version" : 1,

"result" : "created",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 0,

"\_primary\_term" : 1

}

[IMG_260](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

从上面的响应可以看到，我们在"customer"索引下成功创建了一个文档。这个文档还有一个内部id为1，这是我们在创建的时候指定的。

需要注意的是，Elasticsearch并不要求你在索引文档之前就先创建索引，然后才能将文档编入索引。在前面的示例中，如果事先不存在"customer"索引，Elasticsearch将自动创建"customer"索引。

（画外音：也就是说，在新建文档的时候如果指定的索引不存在则会自动创建相应的索引）

现在，让我重新检索这个文档：

请求：

curl -X GET "localhost:9200/customer/\_doc/1?pretty"

响应：

[IMG_261](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

{

"\_index" : "customer",

"\_type" : "\_doc",

"\_id" : "1",

"\_version" : 1,

"found" : true,

"\_source" : {

"name" : "John Doe"

}

}

[IMG_262](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

可以看到除了"found"字段外没什么不同，"\_source"字段返回了一个完整的JSON文档。

## **删除一个索引**

现在，让我们删除前面创建的索引，然后查看全部索引

请求：

curl -X DELETE "localhost:9200/customer?pretty"

响应：

{

"acknowledged" : true

}

接下来，查看一下

curl -X GET "localhost:9200/\_cat/indices?v"

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

到现在为止，我们已经学习了创建/删除索引、索引/查询文档这四个命令

curl -X PUT "localhost:9200/customer"

curl -X PUT "localhost:9200/customer/\_doc/1" -H 'Content-Type: application/json' -d'{"name": "John Doe"}'

curl -X GET "localhost:9200/customer/\_doc/1"

curl -X DELETE "localhost:9200/customer"

如果我们仔细研究上面的命令，我们实际上可以看到如何在Elasticsearch中访问数据的模式。这种模式可以概括如下：

<REST Verb> /<Index>/<Type>/<ID>

# **修改数据**

## **更新文档**

事实上，每当我们执行更新时，Elasticsearch就会删除旧文档，然后索引一个新的文档。

下面这个例子展示了如何更新一个文档（ID为1），改变name字段为"Jane Doe"，同时添加一个age字段：

请求：

curl -X POST "localhost:9200/customer/\_doc/1/\_update?pretty" -H 'Content-Type: application/json' -d'{

"doc": { "name": "Jane Doe", "age": 20 }

}'

响应：

[IMG_263](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

{

"\_index" : "customer",

"\_type" : "\_doc",

"\_id" : "1",

"\_version" : 2,

"result" : "updated",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 1,

"\_primary\_term" : 1

}

[IMG_264](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

下面这个例子用脚本来将age增加5

请求：

curl -X POST "localhost:9200/customer/\_doc/1/\_update?pretty" -H 'Content-Type: application/json' -d'{

"script" : "ctx.\_source.age += 5"

}'

在上面例子中，ctx.\_source引用的是当前源文档

响应：

[IMG_265](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

{

"\_index" : "customer",

"\_type" : "\_doc",

"\_id" : "1",

"\_version" : 3,

"result" : "updated",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 2,

"\_primary\_term" : 1

}

[IMG_266](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

## **删除文档**

删除文档相当简单。这个例子展示了如何从"customer"索引中删除ID为2的文档：

请求：

curl -X DELETE "localhost:9200/customer/\_doc/2?pretty"

响应：

[IMG_267](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

{

"\_index" : "customer",

"\_type" : "\_doc",

"\_id" : "2",

"\_version" : 1,

"result" : "not\_found",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 0,

"\_primary\_term" : 1

}

[IMG_268](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

## **批处理**

除了能够索引、更新和删除单个文档之外，Elasticsearch还可以使用\_bulk API批量执行上述任何操作。

这个功能非常重要，因为它提供了一种非常有效的机制，可以在尽可能少的网络往返的情况下尽可能快地执行多个操作。

下面的例子，索引两个文档（ID 1 - John Doe 和 ID 2 - Jane Doe）

请求：

curl -X POST "localhost:9200/customer/\_doc/\_bulk?pretty" -H 'Content-Type: application/json' -d'{"index":{"\_id":"1"}}

{"name": "John Doe" }

{"index":{"\_id":"2"}}

{"name": "Jane Doe" }'

响应：

[IMG_269](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

{

"took" : 5,

"errors" : false,

"items" : [

{

"index" : {

"\_index" : "customer",

"\_type" : "\_doc",

"\_id" : "1",

"\_version" : 4,

"result" : "updated",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 3,

"\_primary\_term" : 1,

"status" : 200

}

},

{

"index" : {

"\_index" : "customer",

"\_type" : "\_doc",

"\_id" : "2",

"\_version" : 1,

"result" : "created",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 1,

"\_primary\_term" : 1,

"status" : 201

}

}

]

}

[IMG_270](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

接下来的例子展示了，更新第一个文档（ID为1），删除第二个文档（ID为2）：

请求：

curl -X POST "localhost:9200/customer/\_doc/\_bulk?pretty" -H 'Content-Type: application/json' -d'{"update":{"\_id":"1"}}

{"doc": { "name": "John Doe becomes Jane Doe" } }

{"delete":{"\_id":"2"}}'

响应：

[IMG_271](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

{

"took" : 8,

"errors" : false,

"items" : [

{

"update" : {

"\_index" : "customer",

"\_type" : "\_doc",

"\_id" : "1",

"\_version" : 5,

"result" : "updated",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 4,

"\_primary\_term" : 1,

"status" : 200

}

},

{

"delete" : {

"\_index" : "customer",

"\_type" : "\_doc",

"\_id" : "2",

"\_version" : 2,

"result" : "deleted",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 2,

"\_primary\_term" : 1,

"status" : 200

}

}

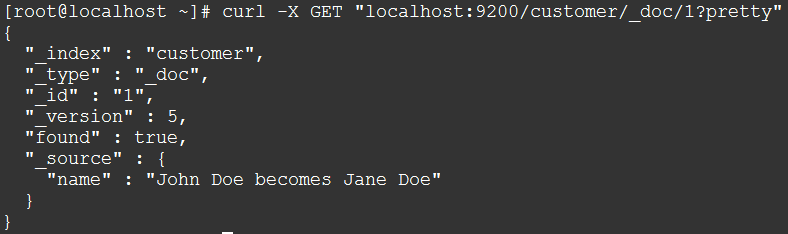
]

}

[IMG_272](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

现在，我们来重新查看一下索引文档

curl -X GET "localhost:9200/customer/\_doc/1?pretty"



# **检索数据**

## **示例数据**

现在我们已经了解了基础知识，让我们尝试处理一个更真实的数据集。我准备了一个关于客户银行账户信息的虚构JSON文档示例。每个文档都有以下格式：

[IMG_274](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

{

"account\_number": 0,

"balance": 16623,

"firstname": "Bradshaw",

"lastname": "Mckenzie",

"age": 29,

"gender": "F",

"address": "244 Columbus Place",

"employer": "Euron",

"email": "bradshawmckenzie@euron.com",

"city": "Hobucken",

"state": "CO"

}

[IMG_275](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

### **加载示例数据**

你可以从这里[下载](https://raw.githubusercontent.com/elastic/elasticsearch/master/docs/src/test/resources/accounts.json" \t "https://www.cnblogs.com/cjsblog/p/_blank)示例数据

提取它到我们的当前目录，并且加载到我们的集群中：

新建一个文件accounts.json，然后将数据复制粘贴到该文件中，保存退出

在这个accounts.json文件所在目录下执行如下命令：

curl -H "Content-Type: application/json" -XPOST "localhost:9200/bank/\_doc/\_bulk?pretty&refresh" --data-binary "@accounts.json"

此时，accounts.json中的文档数据便被索引到"bank"索引下

让我们查看一下索引：

请求：

curl "localhost:9200/\_cat/indices?v"

响应：

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

yellow open customer DoM-O7QmRk-6f3Iuls7X6Q 5 1 1 0 4.5kb 4.5kb

yellow open bank 59jD3B4FR8iifWWjrdMzUg 5 1 1000 0 474.7kb 474.7kb

可以看到，现在我们的集群中有两个索引，分别是"customer"和"bank"

"customer"索引，1个文档，"bank"索引有1000个文档

## **The Search API**

现在让我们从一些简单的搜索开始。运行搜索有两种基本方法：一种是通过REST请求URI发送检索参数，另一种是通过REST请求体发送检索参数。

（画外音：一种是把检索参数放在URL后面，另一种是放在请求体里面。相当于HTTP的GET和POST请求）

请求体方法允许你更有表现力，也可以用更可读的JSON格式定义搜索。

用于搜索的REST API可从\_search端点访问。下面的例子返回"bank"索引中的所有文档：

curl -X GET "localhost:9200/bank/\_search?q=\*&sort=account\_number:asc&pretty"

让我们来剖析一下上面的请求。

我们在"bank"索引中检索，q=\*参数表示匹配所有文档；sort=account\_number:asc表示每个文档的account\_number字段升序排序；pretty参数表示返回漂亮打印的JSON结果。

响应结果看起来是这样的：

[IMG_276](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

{

"took" : 96,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 1000,

"max\_score" : null,

"hits" : [

{

"\_index" : "bank",

"\_type" : "\_doc",

"\_id" : "0",

"\_score" : null,

"\_source" : {

"account\_number" : 0,

"balance" : 16623,

"firstname" : "Bradshaw",

"lastname" : "Mckenzie",

"age" : 29,

"gender" : "F",

"address" : "244 Columbus Place",

"employer" : "Euron",

"email" : "bradshawmckenzie@euron.com",

"city" : "Hobucken",

"state" : "CO"

},

"sort" : [

0

]

},

{

"\_index" : "bank",

"\_type" : "\_doc",

"\_id" : "1",

"\_score" : null,

"\_source" : {

"account\_number" : 1,

"balance" : 39225,

"firstname" : "Amber",

"lastname" : "Duke",

"age" : 32,

"gender" : "M",

"address" : "880 Holmes Lane",

"employer" : "Pyrami",

"email" : "amberduke@pyrami.com",

"city" : "Brogan",

"state" : "IL"

},

"sort" : [

1

]

},

{

"\_index" : "bank",

"\_type" : "\_doc",

"\_id" : "2",

"\_score" : null,

"\_source" : {

"account\_number" : 2,

"balance" : 28838,

"firstname" : "Roberta",

"lastname" : "Bender",

"age" : 22,

"gender" : "F",

"address" : "560 Kingsway Place",

"employer" : "Chillium",

"email" : "robertabender@chillium.com",

"city" : "Bennett",

"state" : "LA"

},

"sort" : [

2

]

},

......

]

}

[IMG_277](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

可以看到，响应由下列几部分组成：

* took ： Elasticsearch执行搜索的时间（以毫秒为单位）
* timed\_out ： 告诉我们检索是否超时
* \_shards ： 告诉我们检索了多少分片，以及成功/失败的分片数各是多少
* hits ： 检索的结果
* hits.total ： 符合检索条件的文档总数
* hits.hits ： 实际的检索结果数组（默认为前10个文档）
* hits.sort ： 排序的key（如果按分值排序的话则不显示）
* hits.\_score 和 max\_score 现在我们先忽略这些字段

下面是一个和上面相同，但是用请求体的例子：

[IMG_278](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": { "match\_all": {} },

"sort": [

{ "account\_number": "asc" }

]

}'

[IMG_279](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

区别在于，我们没有在URI中传递q=\*，而是向\_search API提供json风格的查询请求体

很重要的一点是，一旦返回搜索结果，Elasticsearch就完全完成了对请求的处理，不会在结果中维护任何类型的服务器端资源或打开游标。这是许多其他平台如SQL形成鲜明对比。

## **查询语言**

Elasticsearch提供了一种JSON风格的语言，您可以使用这种语言执行查询。这被成为查询DSL。

查询语言非常全面，乍一看可能有些吓人，但实际上最好的学习方法是从几个基本示例开始。

回到我们上一个例子，我们执行这样的查询：

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": { "match\_all": {} }

}'

查询部分告诉我们查询定义是什么，match\_all部分只是我们想要运行的查询类型。这里match\_all查询只是在指定索引中搜索所有文档。

除了查询参数外，我们还可以传递其他参数来影响搜索结果。在上面部分的例子中，我们传的是sort参数，这里我们传size：

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": { "match\_all": {} },

"size": 1

}'

注意：如果size没有指定，则默认是10

下面的例子执行match\_all，并返回第10到19条文档：

[IMG_280](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": { "match\_all": {} },

"from": 10,

"size": 10

}'

[IMG_281](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

from参数（从0开始）指定从哪个文档索引开始，并且size参数指定从from开始返回多少条。这个特性在分页查询时非常有用。

注意：如果没有指定from，则默认从0开始

这个示例执行match\_all，并按照帐户余额降序对结果进行排序，并返回前10个（默认大小）文档。

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": { "match\_all": {} },

"sort": { "balance": { "order": "desc" } }

}'

## **搜索**

继续学习查询DSL。首先，让我们看一下返回的文档字段。默认情况下，会返回完整的JSON文档（PS：也就是返回所有字段）。这被成为source（hits.\_source）

如果我们不希望返回整个源文档，我们可以从源文档中只请求几个字段来返回。

下面的例子展示了只返回文档中的两个字段：account\_number 和 balance字段

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": { "match\_all": {} },

"\_source": ["account\_number", "balance"]

}'

（画外音：相当于SELECT account\_number， balance FROM bank）

现在让我们继续查询部分。以前，我们已经看到了如何使用match\_all查询匹配所有文档。现在让我们引入一个名为match query的新查询，它可以被看作是基本的字段搜索查询(即针对特定字段或字段集进行的搜索)。

下面的例子返回account\_number为20的文档

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": { "match": { "account\_number": 20 } }

}'

（画外音：相当于SELECT \* FROM bank WHERE account\_number = 20）

下面的例子返回address中包含"mill"的账户：

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": { "match": { "address": "mill" } }

}'

（画外音：相当于SELECT \* FROM bank WHERE address LIKE '%mill%'）

下面的例子返回address中包含"mill"或者"lane"的账户：

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": { "match": { "address": "mill lane" } }

}'

（画外音：相当于SELECT \* FROM bank WHERE address LIKE '%mill' OR address LIKE '%lane%'）

让我们来引入bool查询，bool查询允许我们使用布尔逻辑将较小的查询组合成较大的查询。

下面的例子将两个match查询组合在一起，返回address中包含"mill"和"lane"的账户：

[IMG_282](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": {

"bool": {

"must": [

{ "match": { "address": "mill" } },

{ "match": { "address": "lane" } }

]

}

}

}'

[IMG_283](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

（画外音：相当于SELECT \* FROM bank WHERE address LIKE '%mill%lane%'）

上面是bool must查询，下面这个是bool shoud查询：

[IMG_284](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": {

"bool": {

"should": [

{ "match": { "address": "mill" } },

{ "match": { "address": "lane" } }

]

}

}

}'

[IMG_285](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

（画外音：must相当于and，shoud相当于or，must\_not相当于！）

（画外音：逻辑运算符：与/或/非，and/or/not，在这里就是must/should/must\_not）

我们可以在bool查询中同时组合must、should和must\_not子句。此外，我们可以在任何bool子句中编写bool查询，以模拟任何复杂的多级布尔逻辑。

下面的例子是一个综合应用：

[IMG_286](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": {

"bool": {

"must": [

{ "match": { "age": "40" } }

],

"must\_not": [

{ "match": { "state": "ID" } }

]

}

}

}'

[IMG_287](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

（画外音：相当于SELECT \* FROM bank WHERE age LIKE '%40%' AND state NOT LIKE '%ID%'）

## **过滤**

分数是一个数值，它是文档与我们指定的搜索查询匹配程度的相对度量（PS：相似度）。分数越高，文档越相关，分数越低，文档越不相关。

但是查询并不总是需要产生分数，特别是当它们仅用于“过滤”文档集时。Elasticsearch检测到这些情况并自动优化查询执行，以便不计算无用的分数。

我们在前一节中介绍的bool查询还支持filter子句，该子句允许使用查询来限制将由其他子句匹配的文档，而不改变计算分数的方式。

作为一个例子，让我们引入range查询，它允许我们通过一系列值筛选文档。这通常用于数字或日期过滤。

下面这个例子用一个布尔查询返回所有余额在20000到30000之间（包括30000，BETWEEN...AND...是一个闭区间）的账户。换句话说，我们想要找到余额大于等于20000并且小于等等30000的账户。

[IMG_288](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"query": {

"bool": {

"must": { "match\_all": {} },

"filter": {

"range": {

"balance": {

"gte": 20000,

"lte": 30000

}

}

}

}

}

}'

[IMG_289](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

## **聚集**

（画外音：相当于SQL中的聚集函数，比如分组、求和、求平均数之类的）

首先，这个示例按state对所有帐户进行分组，然后按照count数降序（默认）返回前10条（默认）：

（画外音：相当于按state分组，然后count(\*)，每个组中按照COUNT(\*)数取 top 10）

[IMG_290](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"size": 0,

"aggs": {

"group\_by\_state": {

"terms": {

"field": "state.keyword"

}

}

}

}'

[IMG_291](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

在SQL中，上面的聚集操作类似于：

SELECT state, COUNT(\*) FROM bank GROUP BY state ORDER BY COUNT(\*) DESC LIMIT **10**;

响应：

[IMG_292](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

{

"took":50,

"timed\_out":false,

"\_shards":{

"total":5,

"successful":5,

"skipped":0,

"failed":0

},

"hits":{

"total":1000,

"max\_score":0,

"hits":[

]

},

"aggregations":{

"group\_by\_state":{

"doc\_count\_error\_upper\_bound":20,

"sum\_other\_doc\_count":770,

"buckets":[

{

"key":"ID",

"doc\_count":27

},

{

"key":"TX",

"doc\_count":27

},

{

"key":"AL",

"doc\_count":25

},

{

"key":"MD",

"doc\_count":25

},

{

"key":"TN",

"doc\_count":23

},

{

"key":"MA",

"doc\_count":21

},

{

"key":"NC",

"doc\_count":21

},

{

"key":"ND",

"doc\_count":21

},

{

"key":"ME",

"doc\_count":20

},

{

"key":"MO",

"doc\_count":20

}

]

}

}

}

[IMG_293](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

注意，我们将size=0设置为不显示搜索结果，因为我们只想看到响应中的聚合结果。

接下来的例子跟上一个类似，按照state分组，然后取balance的平均值

[IMG_294](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"size": 0,

"aggs": {

"group\_by\_state": {

"terms": {

"field": "state.keyword"

},

"aggs": {

"average\_balance": {

"avg": {

"field": "balance"

}

}

}

}

}

}'

[IMG_295](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

在SQL中，相当于：

SELECT state, COUNT(\*), AVG(balance) FROM bank GROUP BY state ORDER BY COUNT(\*) DESC LIMIT **10**;

下面这个例子展示了我们如何根据年龄段(20-29岁，30-39岁，40-49岁)来分组，然后根据性别分组，最后得到平均账户余额，每个年龄等级，每个性别：

[IMG_296](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

curl -X GET "localhost:9200/bank/\_search" -H 'Content-Type: application/json' -d'{

"size": 0,

"aggs": {

"group\_by\_age": {

"range": {

"field": "age",

"ranges": [

{

"from": 20,

"to": 30

},

{

"from": 30,

"to": 40

},

{

"from": 40,

"to": 50

}

]

},

"aggs": {

"group\_by\_gender": {

"terms": {

"field": "gender.keyword"

},

"aggs": {

"average\_balance": {

"avg": {

"field": "balance"

}

}

}

}

}

}

}

}'

[IMG_297](https://www.cnblogs.com/cjsblog/p/javascript:void(0);)

先写到这里吧！！！

## **参考**

[https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html](https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html" \t "https://www.cnblogs.com/cjsblog/p/_blank)

发现有一本书叫《[Elasticsearch: 权威指南](https://www.elastic.co/guide/cn/elasticsearch/guide/current/index.html" \t "https://www.cnblogs.com/cjsblog/p/_blank)》