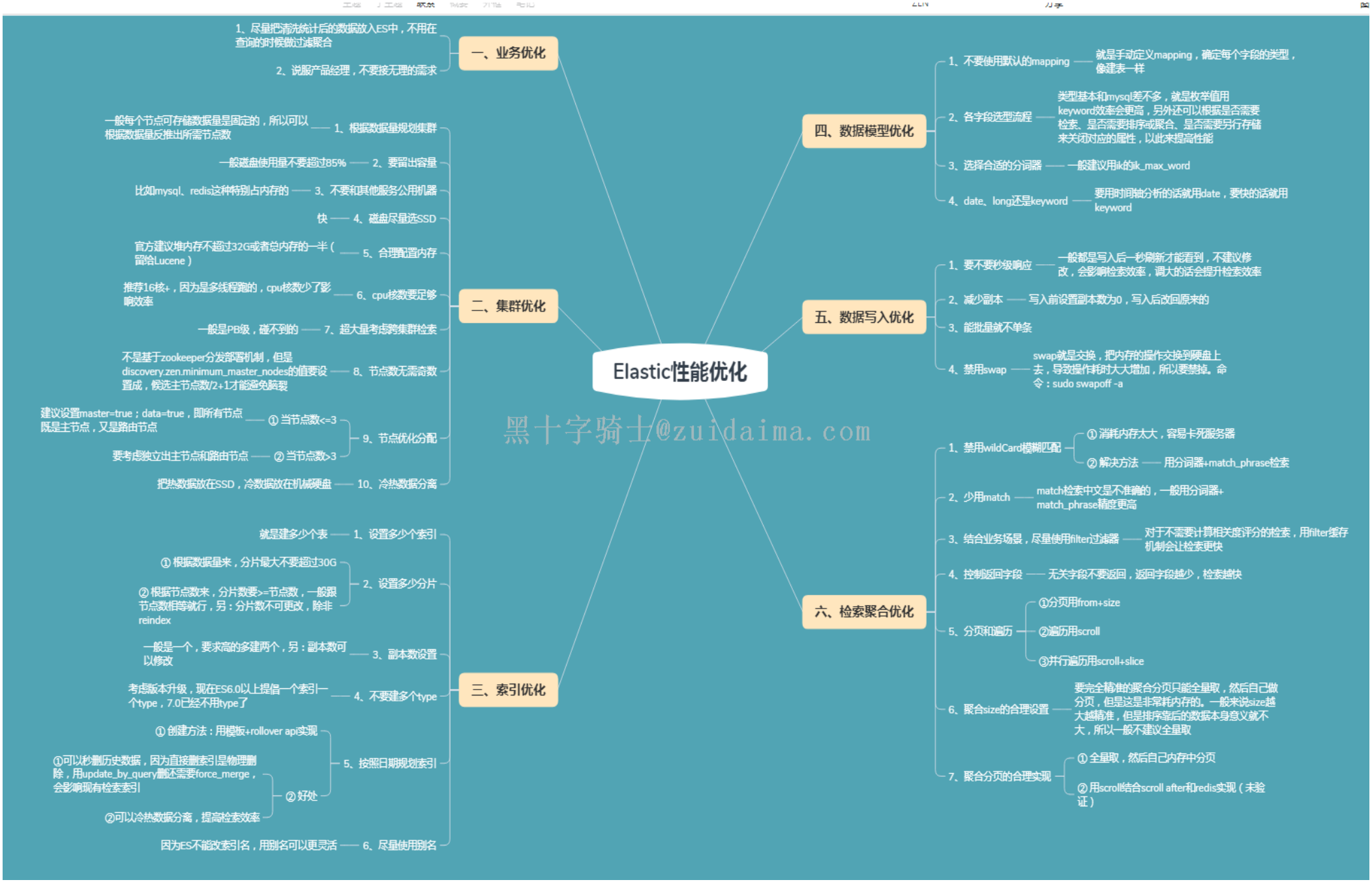
分布式搜索引擎 **Elasticsearch**



Elasticsearch 是一个分布式的 RESTful 风格的搜索和数据分析引擎，能够解决越来越多的用例。作为 Elastic Stack 的核心，它集中存储您的数据，帮助您发现意料之中以及意料之外的情况。

Elasticsearch 是一个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：

- 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。

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。

全文搜索引擎 Elasticsearch 入门教程

全文搜索属于最常见的需求，开源的 Elasticsearch （以下简称 Elastic）是目前全文搜索引擎的首选。

它可以快速地储存、搜索和分析海量数据。维基百科、Stack Overflow、Github 都采用它。

Elastic 的底层是开源库 Lucene。但是，你没法直接用 Lucene，必须自己写代码去调用它的接口。Elastic 是 Lucene 的封装，提供了 REST API 的操作接口，开箱即用。

本文从零开始，讲解如何使用 Elastic 搭建自己的全文搜索引擎。每一步都有详细的说明，大家跟着做就能学会。

一、安装

Elastic 需要 Java 8 环境。如果你的机器还没安装 Java，可以参考这篇文章，注意要保证环境变量 `JAVA_HOME` 正确设置。

安装完 Java，就可以跟着官方文档安装 Elastic。直接下载压缩包比较简单。

```
$ wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-5.5.1.zip
$ unzip elasticsearch-5.5.1.zip
$ cd elasticsearch-5.5.1/
```

接着，进入解压后的目录，运行下面的命令，启动 Elastic。

```
$ ./bin/elasticsearch
```

如果这时报错"max virtual memory areas vm.maxmapcount [65530] is too low"，要运行下面的命令。

```
$ sudo sysctl -w vm.max_map_count=262144
```

如果一切正常，Elastic 就会在默认的 9200 端口运行。这时，打开另一个命令行窗口，请求该端口，会得到说明信息。

```
$ curl localhost:9200
{
  "name" : "atntrTf",
  "cluster_name" : "elasticsearch",
```

```
  "cluster_uuid" : "tf9250XhQ6ee4h7YI11anA",
  "version" : {
    "number" : "5.5.1",
    "build_hash" : "19c13d0",
    "build_date" : "2017-07-18T20:44:24.823Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.0"
  },
  "tagline" : "You Know, for Search"
}
```

上面代码中，请求 9200 端口，Elastic 返回一个 JSON 对象，包含当前节点、集群、版本等信息。

按下 Ctrl + C，Elastic 就会停止运行。

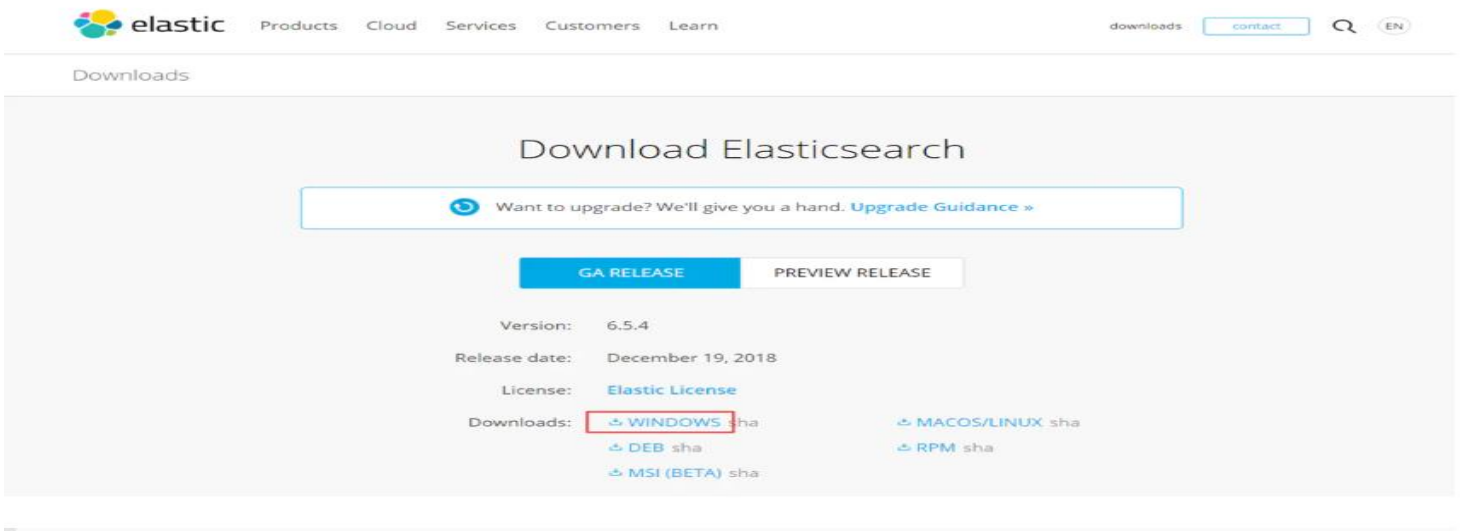
默认情况下，Elastic 只允许本机访问，如果需要远程访问，可以修改 Elastic 安装目录的 `config/elasticsearch.yml` 文件，去掉 `network.host` 的注释，将它的值改成 `0.0.0.0`，然后重新启动 Elastic。

```
network.host: 0.0.0.0
```

上面代码中，设成 `0.0.0.0` 让任何人都可以访问。线上服务不要这样设置，要设成具体的 IP。

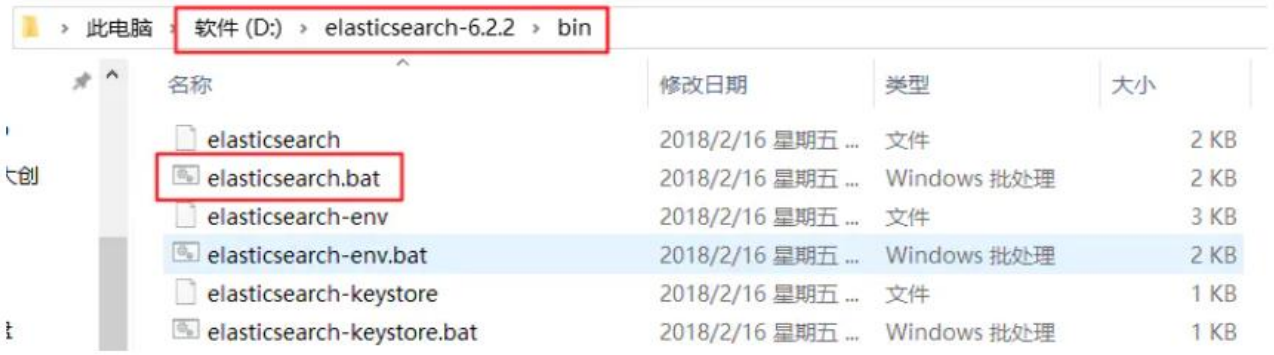
安装 Elasticsearch

官网最新版本 Elasticsearch （6.5.4），但是由于自己的环境使用最新版本的有问题（配合下面的工具 Kibana 有问题..Kibana 启动不了），所以不得不换成更低版本的 6.2.2，下载外链：戳这里，当然你也可以试一下最新的版本：

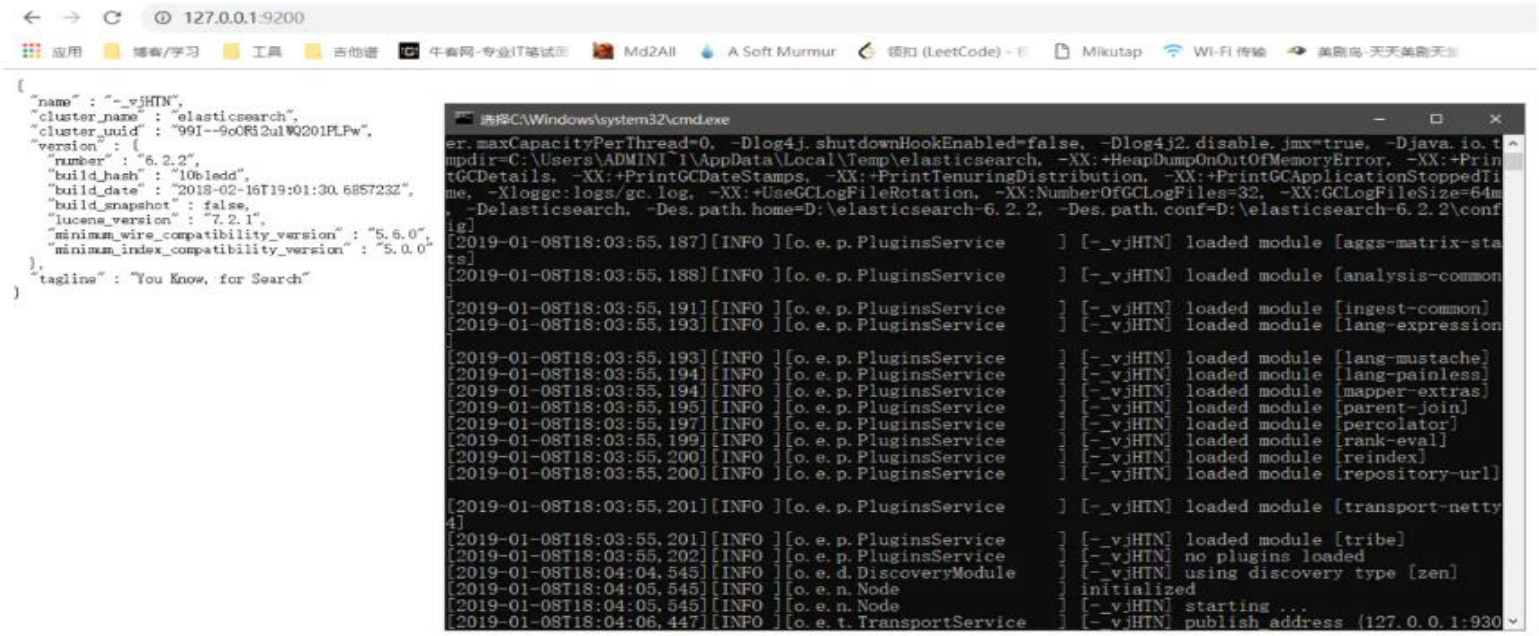


顺带一提：在下载之前你应该确保你的 Java 版本保持在 1.8 及以上（就 1.8 吧..），这是 Elasticsearch 的硬性要求，可以自行打开命令行输入 java -version 来查看 Java 的版本

下载完成后，可以看到是一个压缩包，我们直接解压在 D 盘上，然后打开 bin 目录下的 elasticsearch.bat 文件



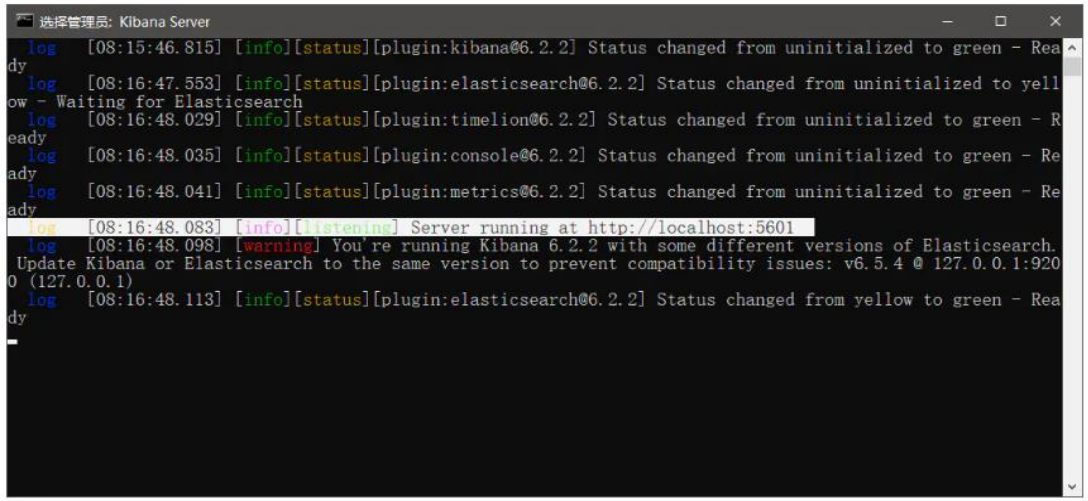
等待一段时间后，可以看到小黑框输出一行 start ，就说明我们的 Elasticsearch 已经跑起来了，我们访问地址：http://127.0.0.1:9200/，看到返回一串 JSON 格式的代码就说明已经成功了：



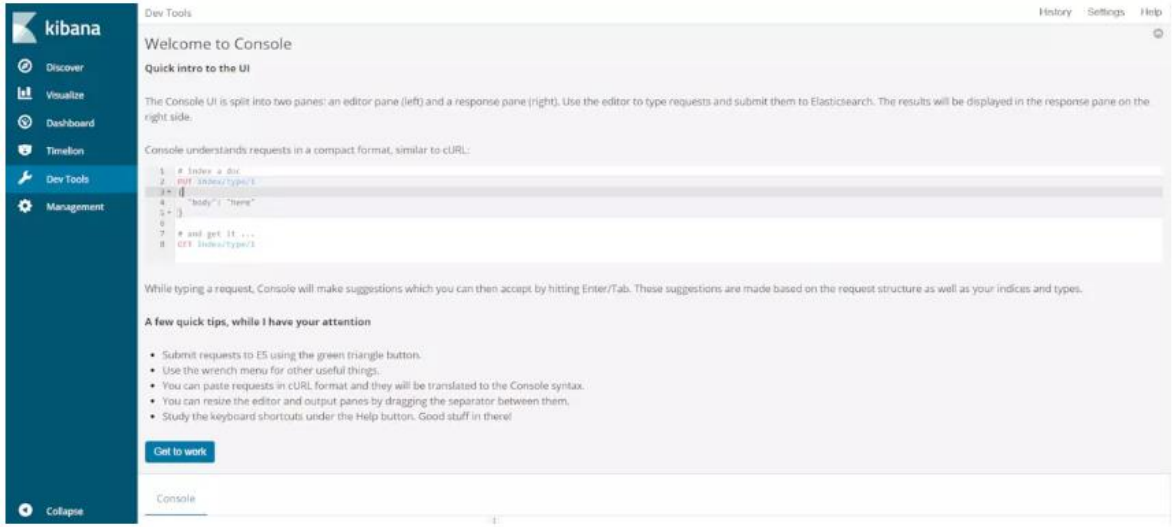
安装 Kibana

这是一个官方推出的把 Elasticsearch 数据可视化的工具，官网在这里：【传送门】，不过我们现在暂时还用不到那些数据分析的东西，不过里面有一个 Dev Tools 的工具可以方便的和 Elasticsearch 服务进行交互，去官网下载了最新版本的 Kibana（6.5.4） 结果不知道为什么总是启动不起来，所以换一了一个低版本的（6.2.2）正常，给个下载外链：下载点这里，你们也可以去官网试试能不能把最新的跑起来：

解压到 D 盘（意外的有点慢..），同样打开目录下的 bin\kibana.bat：



等待一段时间后就可以看到提示信息，运行在 5601 端口，我们访问地址 http://localhost:5601/app/kibana#/dev_tools/console?_g=() 可以成功进入到 Dev-tools 界面：



点击 【Get to work】，然后在控制台输入 GET /_cat/health?v 查看服务器状态，可以在右侧返回的结果中看到 green 即表示服务器状态目前是健康的：

Elasticsearch 基本概念

全文搜索(Full-text Search)

全文检索是指计算机索引程序通过扫描文章中的每一个词，对每一个词建立一个索引，指明该词在文章中出现的次数和位置，当用户查询时，检索程序就根据事先建立的索引进行查找，并将查找的结果反馈给用户的检索方式。

在全文搜索的世界中，存在着几个庞大的帝国，也就是主流工具，主要有：

- Apache Lucene
- Elasticsearch
- Solr
- Ferret

倒排索引（Inverted Index）

该索引表中的每一项都包括一个属性值和具有该属性值的各记录的地址。由于不是由记录来确定属性值，而是由属性值来确定记录的位置，因而称为倒排索引(inverted index)。Elasticsearch 能够实现快速、高效的搜索功能，正是基于倒排索引原理。

节点 & 集群（Node & Cluster）

Elasticsearch 本质上是一个分布式数据库，允许多台服务器协同工作，每台服务器可以运行多个 Elasticsearch 实例。单个 Elasticsearch 实例称为一个节点（Node），一组节点构成一个集群（Cluster）。

索引（Index）

Elasticsearch 数据管理的顶层单位就叫做 Index（索引），相当于关系型数据库里的数据库的概念。另外，每个 Index 的名字必须是小写。

文档（Document）

Index 里面单条的记录称为 Document（文档）。许多条 Document 构成了一个 Index。Document 使用 JSON 格式表示。同一个 Index 里面的 Document，不要求有相同的结构（scheme），但是最好保持相同，这样有利于提高搜索效率。

类型（Type）

Document 可以分组，比如 employee 这个 Index 里面，可以按部门分组，也可以按职级分组。这种分组就叫做 Type，它是虚拟的逻辑分组，用来过滤 Document，类似关系型数据库中的数据表。

不同的 Type 应该有相似的结构（Schema），性质完全不同的数据（比如 products 和 logs）应该存成两个 Index，而不是一个 Index 里面的两个 Type（虽然可以做到）。

文档元数据（Document metadata）

文档元数据为_index, _type, _id, 这三者可以唯一表示一个文档，_index 表示文档在哪存放，_type 表示文档的对象类别，_id 为文档的唯一标识。

字段（Fields）

每个 Document 都类似一个 JSON 结构，它包含了许多字段，每个字段都有其对应的值，多个字段组成了一个 Document，可以类比关系型数据库数据表中的字段。

在 Elasticsearch 中，文档（Document）归属于一种类型（Type），而这些类型存在于索引（Index）中，下图展示了 Elasticsearch 与传统关系型数据库的类比：

```
Relational DB  ⇒ Databases ⇒ Tables ⇒ Rows      ⇒ Columns
Elasticsearch  ⇒ Indices   ⇒ Types  ⇒ Documents ⇒ Fields
```

Elasticsearch 入门

Elasticsearch 提供了多种交互使用方式，包括 Java API 和 RESTful API ，本文主要介绍 RESTful API 。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 。甚至，你还可以使用 curl 命令来和 Elasticsearch 交互。

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

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

返回的数据格式为 JSON，因为 Elasticsearch 中的文档以 JSON 格式储存。其中，被 < > 标记的部件：

部件	说明
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)

对于 HTTP 方法，它们的具体作用为：

HTTP 方法	说明
GET	获取请求对象的当前状态
POST	改变对象的当前状态

HTTP 方法	说明
PUT	创建一个对象
DELETE	销毁对象
HEAD	请求获取对象的基础信息

我们以下面的数据为例，来展示 Elasticsearch 的用法。

host	title	description	attendees	date	reviews
Dave	Elasticsearch at Rangespan and Exonar	Representatives from Rangespan and Exonar will come and discuss how they use Elasticsearch.	["Dave", "Andrew", "David", "Clint"]	2013-06-24T18:30	3
Dave Nolan	real-time Elasticsearch	We will discuss using Elasticsearch to index data in real time.	["Dave", "Shay", "John", "Harry"]	2013-02-18T18:30	3
Andy	Moving Hadoop to the mainstream	Come hear about how Hadoop is moving to the main stream.	["Andy", "Matt", "Bill"]	2013-07-21T18:00	4
Andy	Big Data and the cloud at Microsoft	Discussion about the Microsoft Azure cloud and HDInsight.	["Andy", "Michael", "Ben", "David"]	2013-07-31T18:00	1
Mik	Logging and Elasticsearch	Get a deep dive for what Elasticsearch is and how it can be used for logging with Logstash as well as Kibana!	["Shay", "Rashid", "Erik", "Grant", "Mik"]	2013-04-08T18:00	3

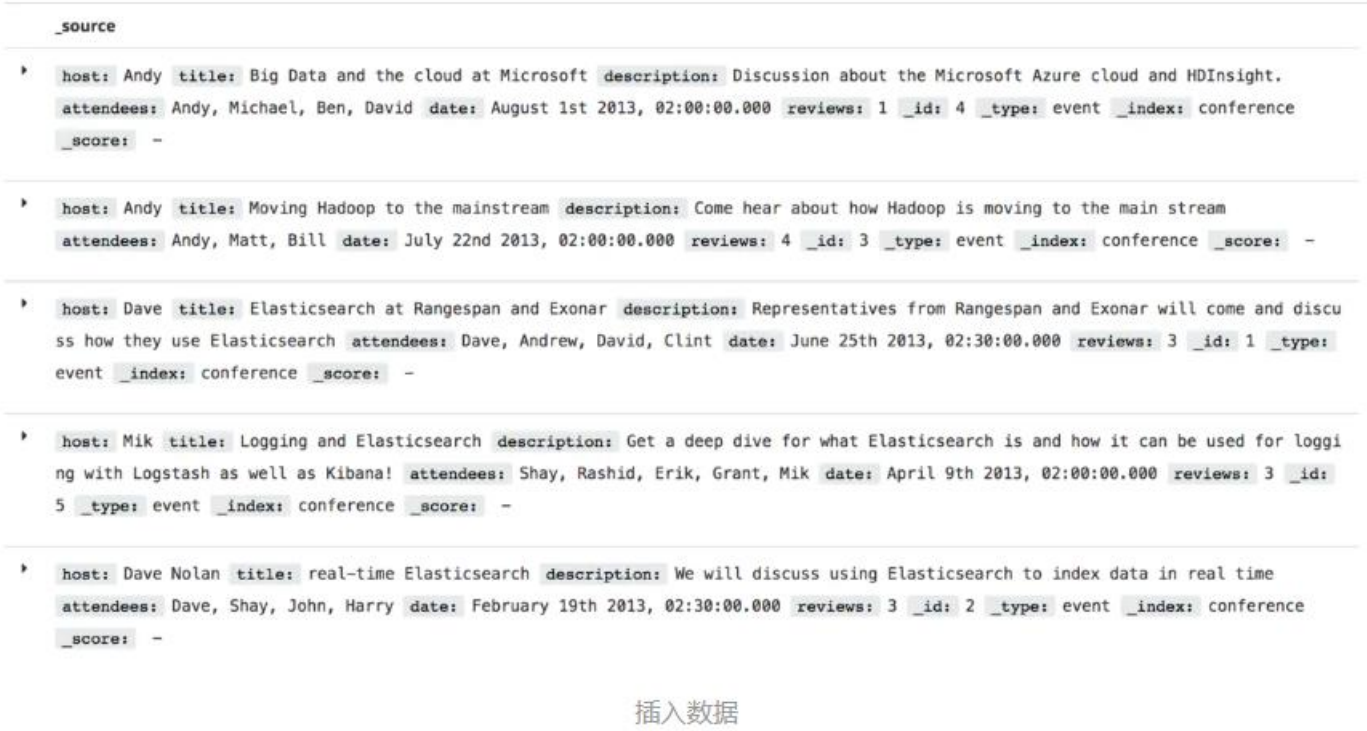
以下全部的操作都在 Kibana 中完成，创建的 index 为 conference, type 为 event .

插入数据

首先创建 index 为 conference, 创建 type 为 event, 插入 id 为 1 的第一条数据，只需运行下面命令就行：

```
PUT /conference/event/1
{
  "host": "Dave",
  "title": "Elasticsearch at Rangespan and Exonar",
  "description": "Representatives from Rangespan and Exonar will come and discuss how they use Elasticsearch",
  "attendees": ["Dave", "Andrew", "David", "Clint"],
  "date": "2013-06-24T18:30",
  "reviews": 3
}
```

在上面的命令中，路径/conference/event/1 表示文档的 index 为 conference, type 为 event, id 为 1. 类似于上面的操作，依次插入剩余的 4 条数据，完成插入后，查看数据如下：



删除数据

比如我们想要删除 conference 中 event 里面 id 为 5 的数据，只需运行下面命令即可：

```
DELETE /conference/event/5
```

返回结果如下：

```
{
  "_index" : "conference",
  "_type" : "event",
  "_id" : "5",
  "_version" : 2,
  "result" : "deleted",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 1,
```

```
  "_primary_term" : 1
}
```

表示该文档已成功删除。如果想删除整个 event 类型，可输入命令:

```
DELETE /conference/event
```

如果想删除整个 conference 索引，可输入命令:

```
DELETE /conference
```

修改数据

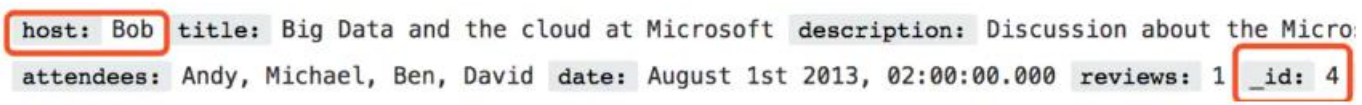
修改数据的命令为 POST，比如我们想要将 conference 中 event 里面 id 为 4 的文档的作者改为 Bob，那么需要运行命令如下：

```
POST /conference/event/4/_update
{
  "doc": {"host": "Bob"}
}
```

返回的信息如下：（表示修改数据成功）

```
{
  "_index" : "conference",
  "_type" : "event",
  "_id" : "4",
  "_version" : 7,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 7,
  "_primary_term" : 1
}
```

查看修改后的数据如下：



修改数据

查询数据

查询数据的命令为 GET，查询命令也是 Elasticsearch 最为重要的功能之一。比如我们想查询 conference 中 event 里面 id 为 1 的数据，运行命令如下：

```
GET /conference/event/1
```

返回的结果如下：

```
{
  "_index" : "conference",
  "_type" : "event",
  "_id" : "1",
  "_version" : 2,
  "found" : true,
  "_source" : {
    "host" : "Dave",
    "title" : "Elasticsearch at Rangespan and Exonar",
    "description" : "Representatives from Rangespan and Exonar will come and discuss how they use Elasticsearch",
    "attendees" : [
      "Dave",
      "Andrew",
      "David",
      "Clint"
    ],
    "date" : "2013-06-24T18:30",
    "reviews" : 3
  }
}
```

在_source 属性中，内容是原始的 JSON 文档，还包含有其它属性，比如_index, _type, _id, _found 等。

如果想要搜索 conference 中 event 里面所有的文档，运行命令如下：

```
GET /conference/event/_search
```

返回结果包括了所有四个文档，放在数组 hits 中。

当然，Elasticsearch 提供更加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。利用*查询表达式*，我们可以检索出 conference 中 event 里面所有 host 为 Bob 的文档，命令如下：

```
GET /conference/event/_search
{
    "query" : {
        "match" : {
            "host" : "Bob"
```

```
        }
    }
}
```

返回的结果只包括了一个文档，放在数组 hits 中。

接着，让我们尝试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 description 中含有"use Elasticsearch"的 event：

```
GET /conference/event/_search
{
    "query" : {
        "match" : {
            "description" : "use Elasticsearch"
        }
    }
}
```

返回的结果（部分）如下：

```
{
 ...
  "hits" : {
    "total" : 2,
    "max_score" : 0.65109104,
    "hits" : [
      {
        ...
        "_score" : 0.65109104,
        "_source" : {
          "host" : "Dave Nolan",
          "title" : "real-time Elasticsearch",
          "description" : "We will discuss using Elasticsearch to index data in real time",
          ...
        }
      },
      {
        ...
        "_score" : 0.5753642,
        "_source" : {
```

```
            "host" : "Dave",
            "title" : "Elasticsearch at Rangespan and Exonar",
            "description" : "Representatives from Rangespan and Exonar will come and discuss how they use Elasticsearch",
            ...
          }
        }
      ]
    }
}
```

返回的结果包含了两个文档，放在数组 hits 中。让我们对这个结果做一些分析，第一个文档的 description 里面含有“using Elasticsearch”，这个能匹配“use Elasticsearch”是因为 Elasticsearch 含有内置的词干提取算法，之后两个文档按_score 进行排序，_score 字段表示文档的相似度（默认的相似度算法为 BM25）。

如果想搜索下所有 description 中严格含有"use Elasticsearch"这个短语的 event，可以使用下面的命令：

```
GET /conference/event/_search
{
    "query" : {
        "match_phrase": {
            "description" : "use Elasticsearch"
        }
    }
}
```

这时候返回的结果只有一个文档，就是上面输出的第二个文档。

当然，Elasticsearch 还支持更多的搜索功能，比如过滤器，高亮搜索，结构化搜索等，希望接下来能有更多的时间和经历来介绍~

总结

后续有机会再介绍如何利用 Python 来操作 Elasticsearch~

本次分享到此结束，感谢大家阅读~

二、基本概念

2.1 Node 与 Cluster

Elastic 本质上是一个分布式数据库，允许多台服务器协同工作，每台服务器可以运行多个 Elastic 实例。

单个 Elastic 实例称为一个节点（node）。一组节点构成一个集群（cluster）。

2.2 Index

Elastic 会索引所有字段，经过处理后写入一个反向索引（Inverted Index）。查找数据的时候，直接查找该索引。

所以，Elastic 数据管理的顶层单位就叫做 Index（索引）。它是单个数据库的同义词。每个 Index （即数据库）的名字必须是小写。

下面的命令可以查看当前节点的所有 Index。

```
$ curl -X GET 'http://localhost:9200/_cat/indices?v'
```

2.3 Document

Index 里面单条的记录称为 Document（文档）。许多条 Document 构成了一个 Index。
Document 使用 JSON 格式表示，下面是一个例子。

```
{
  "user": "张三",
  "title": "工程师",
  "desc": "数据库管理"
}
```

同一个 Index 里面的 Document，不要求有相同的结构（scheme），但是最好保持相同，这样有利于提高搜索效率。

2.4 Type

Document 可以分组，比如 `weather` 这个 Index 里面，可以按城市分组（北京和上海），也可以按气候分组（晴天和雨天）。这种分组就叫做 Type，它是虚拟的逻辑分组，用来过滤 Document。
不同的 Type 应该有相似的结构（schema），举例来说，`id` 字段不能在这个组是字符串，在另一个组是数值。这是与关系型数据库的表的一个区别。性质完全不同的数据（比如 `products` 和 `logs`）应该存成两个 Index，而不是一个 Index 里面的两个 Type（虽然可以做到）。
下面的命令可以列出每个 Index 所包含的 Type。

```
$ curl 'localhost:9200/_mapping?pretty=true'
```

根据规划，Elastic 6.x 版只允许每个 Index 包含一个 Type，7.x 版将会彻底移除 Type。

三、新建和删除 Index

新建 Index，可以直接向 Elastic 服务器发出 PUT 请求。下面的例子是新建一个名叫 `weather` 的 Index。

```
$ curl -X PUT 'localhost:9200/weather'
```

服务器返回一个 JSON 对象，里面的 `acknowledged` 字段表示操作成功。

```
{
  "acknowledged":true,
  "shards_acknowledged":true
}
```

然后，我们发出 DELETE 请求，删除这个 Index。

```
$ curl -X DELETE 'localhost:9200/weather'
```

四、中文分词设置

首先，安装中文分词插件。这里使用的是 ik，也可以考虑其他插件（比如 smartcn）。

```
$ ./bin/elasticsearch-plugin install https://github.com/medcl/elasticsearch-analysis-ik/releases/download/v5.5.1/elasticsearch-analysis-ik-5.5.1.zip
```

上面代码安装的是 5.5.1 版的插件，与 Elastic 5.5.1 配合使用。

接着，重新启动 Elastic，就会自动加载这个新安装的插件。

然后，新建一个 Index，指定需要分词的字段。这一步根据数据结构而异，下面的命令只针对本文。基本上，凡是需要搜索的中文字段，都要单独设置一下。

```
$ curl -X PUT 'localhost:9200/accounts' -d '
{
  "mappings": {
    "person": {
      "properties": {
        "user": {
          "type": "text",
          "analyzer": "ik max word",
          "search analyzer": "ik max word"
        },
        "title": {
          "type": "text",
          "analyzer": "ik max word",
          "search analyzer": "ik max word"
        },
        "desc": {
          "type": "text",
          "analyzer": "ik max word",
          "search analyzer": "ik max word"
        }
      }
    }
  }
}'
```

上面代码中，首先新建一个名称为 `accounts` 的 Index，里面有一个名称为 `person` 的 Type。`person` 有三个字段。

- user
- title
- desc

这三个字段都是中文，而且类型都是文本（text），所以需要指定中文分词器，不能使用默认的英文分词器。

Elastic 的分词器称为 analyzer。我们对每个字段指定分词器。

```
"user": {
```

```
  "type": "text",
  "analyzer": "ik max word",
  "search analyzer": "ik max word"
}
```

上面代码中，`analyzer`是字段文本的分词器，`search_analyzer`是搜索词的分词器。`ik_max_word`分词器是插件`ik`提供的，可以对文本进行最大数量的分词。

五、数据操作

5.1 新增记录

向指定的 /Index/Type 发送 PUT 请求，就可以在 Index 里面新增一条记录。比如，向`/accounts/person`发送请求，就可以新增一条人员记录。

```
$ curl -X PUT 'localhost:9200/accounts/person/1' -d '
{
  "user": "张三",
  "title": "工程师",
  "desc": "数据库管理"
}'
```

服务器返回的 JSON 对象，会给出 Index、Type、Id、Version 等信息。

```
{
  " index":"accounts",
  " type":"person",
  " id":"1",
  " version":1,
  "result":"created",
  " shards":{"total":2,"successful":1,"failed":0},
  "created":true
}
```

如果你仔细看，会发现请求路径是`/accounts/person/1`，最后的`1`是该条记录的 Id。它不一定是数字，任意字符串（比如`abc`）都可以。

新增记录的时候，也可以不指定 Id，这时要改成 POST 请求。

```
$ curl -X POST 'localhost:9200/accounts/person' -d '
{
  "user": "李四",
  "title": "工程师",
  "desc": "系统管理"
```

```
}'
```

上面代码中，向`/accounts/person`发出一个 POST 请求，添加一个记录。这时，服务器返回的 JSON 对象里面，`_id`字段就是一个随机字符串。

```
{
  " index":"accounts",
  " type":"person",
  " id":"AV3qGfrC6jMbsbXb6k1p",
  " version":1,
  "result":"created",
  " shards":{"total":2,"successful":1,"failed":0},
  "created":true
}
```

注意，如果没有先创建 Index（这个例子是`accounts`），直接执行上面的命令，Elastic 也不会报错，而是直接生成指定的 Index。所以，打字的时候要小心，不要写错 Index 的名称。

5.2 查看记录

向`/Index/Type/Id`发出 GET 请求，就可以查看这条记录。

```
$ curl 'localhost:9200/accounts/person/1?pretty=true'
```

上面代码请求查看`/accounts/person/1`这条记录，URL 的参数`pretty=true`表示以易读的格式返回。

返回的数据中，`found`字段表示查询成功，`_source`字段返回原始记录。

```
{
  " index" : "accounts",
  " type" : "person",
  " id" : "1",
  " version" : 1,
  "found" : true,
  " source" : {
    "user" : "张三",
    "title" : "工程师",
    "desc" : "数据库管理"
  }
}
```

如果 Id 不正确，就查不到数据，`found`字段就是`false`。

```
$ curl 'localhost:9200/weather/beijing/abc?pretty=true'
```

```
{
  "_index" : "accounts",
  "_type" : "person",
  "_id" : "abc",
  "found" : false
}
```

5.3 删除记录

删除记录就是发出 DELETE 请求。

```
$ curl -X DELETE 'localhost:9200/accounts/person/1'
```

这里先不要删除这条记录，后面还要用到。

5.4 更新记录

更新记录就是使用 PUT 请求，重新发送一次数据。

```
$ curl -X PUT 'localhost:9200/accounts/person/1' -d '
{
    "user" : "张三",
    "title" : "工程师",
    "desc" : "数据库管理，软件开发"
}'
```

```
{
  "_index":"accounts",
  "_type":"person",
  "_id":"1",
  "_version":2,
  "result":"updated",
  "_shards":{"total":2,"successful":1,"failed":0},
  "created":false
}
```

上面代码中，我们将原始数据从"数据库管理"改成"数据库管理，软件开发"。 返回结果里面，有几个字段发生了变化。

```
"_version" : 2,
"result" : "updated",
"created" : false
```

可以看到，记录的 Id 没变，但是版本（version）从 `1` 变成 `2`，操作类型（result）从 `created` 变成 `updated`，`created` 字段变成 `false`，因为这次不是新建记录。

六、数据查询

6.1 返回所有记录

使用 GET 方法，直接请求`/Index/Type/_search`，就会返回所有记录。

```
$ curl 'localhost:9200/accounts/person/_search'

{
  "took":2,
  "timed_out":false,
  "_shards":{"total":5,"successful":5,"failed":0},
  "hits":{
    "total":2,
    "max_score":1.0,
    "hits":[
      {
        "_index":"accounts",
        "_type":"person",
        "_id":"AV3qGfrC6jMbsbXb6k1p",
        "_score":1.0,
        "_source": {
          "user": "李四",
          "title": "工程师",
          "desc": "系统管理"
        }
      },
      {
        "_index":"accounts",
        "_type":"person",
        "_id":"1",
        "_score":1.0,
        "_source": {
          "user" : "张三",
          "title" : "工程师",
          "desc" : "数据库管理，软件开发"
```

```
        }
      }
    ]
  }
}
```

上面代码中，返回结果的`took`字段表示该操作的耗时（单位为毫秒），`timed_out`字段表示是否超时，`hits`字段表示命中的记录，里面子字段的含义如下。

- total：返回记录数，本例是 2 条。
- max_score：最高的匹配程度，本例是 1.0。
- hits：返回的记录组成的数组。

返回的记录中，每条记录都有一个`_score`字段，表示匹配的程序，默认是按照这个字段降序排列。

6.2 全文搜索

Elastic 的查询非常特别，使用自己的查询语法，要求 GET 请求带有数据体。

```
$ curl 'localhost:9200/accounts/person/_search'  -d '
{
  "query" : { "match" : { "desc" : "软件" }}
}'
```

上面代码使用 Match 查询，指定的匹配条件是`desc`字段里面包含"软件"这个词。返回结果如下。

```
{
  "took":3,
  "timed_out":false,
  "_shards":{"total":5,"successful":5,"failed":0},
  "hits":{
    "total":1,
    "max_score":0.28582606,
    "hits":[
      {
        "_index":"accounts",
        "_type":"person",
        "_id":"1",
        "_score":0.28582606,
        "_source": {
          "user" : "张三",
```

```
        "title" : "工程师",
        "desc" : "数据库管理，软件开发"
      }
    }
  ]
 }
}
```

Elastic 默认一次返回 10 条结果，可以通过 `size` 字段改变这个设置。

```
$ curl 'localhost:9200/accounts/person/_search'  -d '
{
 "query" : { "match" : { "desc" : "管理" }},
 "size": 1
}'
```

上面代码指定，每次只返回一条结果。

还可以通过 `from` 字段，指定位移。

```
$ curl 'localhost:9200/accounts/person/_search'  -d '
{
 "query" : { "match" : { "desc" : "管理" }},
 "from": 1,
 "size": 1
}'
```

上面代码指定，从位置 1 开始（默认是从位置 0 开始），只返回一条结果。

6.3 逻辑运算

如果有多个搜索关键字， Elastic 认为它们是 `or` 关系。

```
$ curl 'localhost:9200/accounts/person/_search'  -d '
{
 "query" : { "match" : { "desc" : "软件 系统" }}
}'
```

上面代码搜索的是软件 or 系统。

如果要执行多个关键词的 and 搜索，必须使用布尔查询。

```
$ curl 'localhost:9200/accounts/person/_search'  -d '
{
  "query": {
    "bool": {
      "must": [
        { "match": { "desc": "软件" } },
        { "match": { "desc": "系统" } }
      ]
    }
  }
}'
```

七、参考链接

- ElasticSearch 官方手册
- A Practical Introduction to Elasticsearch

ElasticSearch 简介

Elasticsearch 是一个分布式可扩展的实时搜索和分析引擎,一个建立在全文搜索引擎 Apache Lucene(TM) 基础上的搜索引擎.当然 Elasticsearch 并不仅仅是 Lucene 那么简单，它不仅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。

基本概念

先说 Elasticsearch 的文件存储，Elasticsearch 是面向文档型数据库，一条数据在这里就是一个文档，用 JSON 作为文档序列化的格式，比如下面这条用户数据:

```
{
    "name" :     "John",
    "sex" :      "Male",
    "age" :      25,
    "birthDate": "1990/05/01",
    "about" :    "I love to go rock climbing",
```

```
7.    "interests": [ "sports", "music" ]
8. }
```

用 Mysql 这样的数据库存储就会容易想到建立一张 User 表，有 balabala 的字段等，在 Elasticsearch 里这就是一个*文档*，当然这个文档会属于一个 User 的*类型*，各种各样的类型存在于一个*索引*当中。这里有一份简易的将 Elasticsearch 和关系型数据术语对照表:

```
1. 关系数据库       ⇒ 数据库        ⇒  表         ⇒ 行              ⇒ 列(Columns)

2. Elasticsearch  ⇒ 索引(Index)   ⇒ 类型(type)  ⇒ 文档(Docments)  ⇒ 字段(Fields)
```

一个 Elasticsearch 集群可以包含多个索引(数据库)，也就是说其中包含了很多类型(表)。这些类型中包含了很多的文档(行)，然后每个文档中又包含了很多的字段(列)。Elasticsearch 的交互，可以使用 Java API，也可以直接使用 HTTP 的 Restful API 方式，比如我们打算插入一条记录，可以简单发送一个 HTTP 的请求:

```
1. PUT /megacorp/employee/1
2. {
3.    "name" :     "John",
4.    "sex" :      "Male",
5.    "age" :      25,
6.    "about" :    "I love to go rock climbing",
7.    "interests": [ "sports", "music" ]
8. }
```

更新，查询也是类似这样的操作。

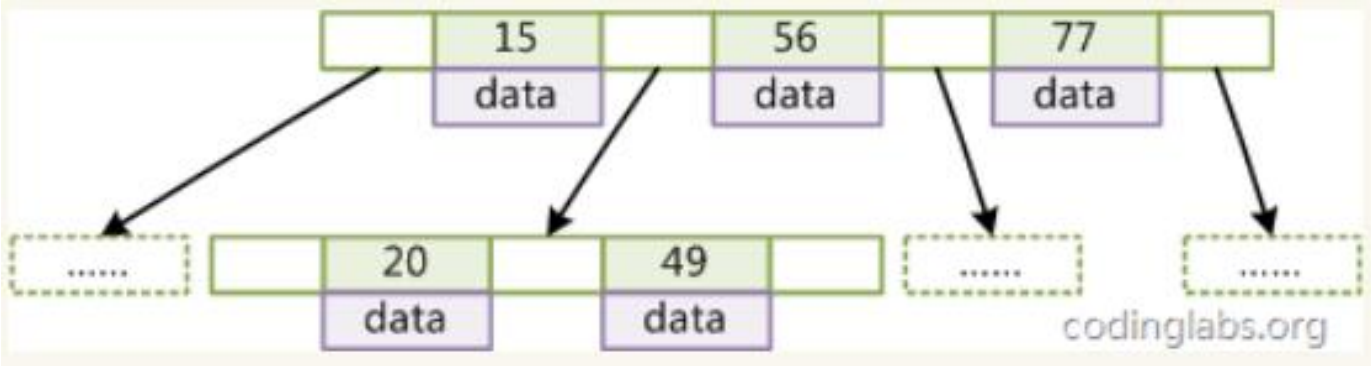
索引

Elasticsearch 最关键的就是提供强大的索引能力。
Elasticsearch 索引的精髓:
一切设计都是为了提高搜索的性能
另一层意思：为了提高搜索的性能，难免会牺牲某些其他方面，比如插入/更新，否则其他数据库不用混了。前面看到往 Elasticsearch 里插入一条记录，其实就是直接 PUT 一个 json 的对象，这个对象有多个 fields，比如上面例子中的 name, sex, age, about, interests，那么在插入这些数据到 Elasticsearch 的同时，Elasticsearch 还默默 1 的为这些字段建立索引--倒排索引，因为 Elasticsearch 最核心功能是搜索。

Elasticsearch 是如何做到快速索引的

Elasticsearch 使用的倒排索引比关系型数据库的 B-Tree 索引快，为什么呢?
什么是 B-Tree 索引?

上大学读书时老师教过我们，二叉树查找效率是 logN，同时插入新的节点不必移动全部节点，所以用树型结构存储索引，能同时兼顾插入和查询的性能。因此在这个基础上，再结合磁盘的读取特性(顺序读/随机读)，传统关系型数据库采用了 B-Tree/B+Tree 这样的数据结构：



为了提高查询的效率，减少磁盘寻道次数，将多个值作为一个数组通过连续区间存放，一次寻道读取多个数据，同时也降低树的高度。
什么是倒排索引?

继续上面的例子，假设有这么几条数据(为了简单，去掉 about, interests 这两个 field):

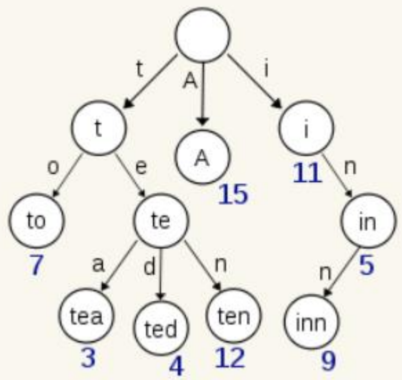
ID	Name	Age	Sex
1	Kate	24	Female
2	John	24	Male
3	Bill	29	Male

ID 是 Elasticsearch 自建的文档 id，那么 Elasticsearch 建立的索引如下:

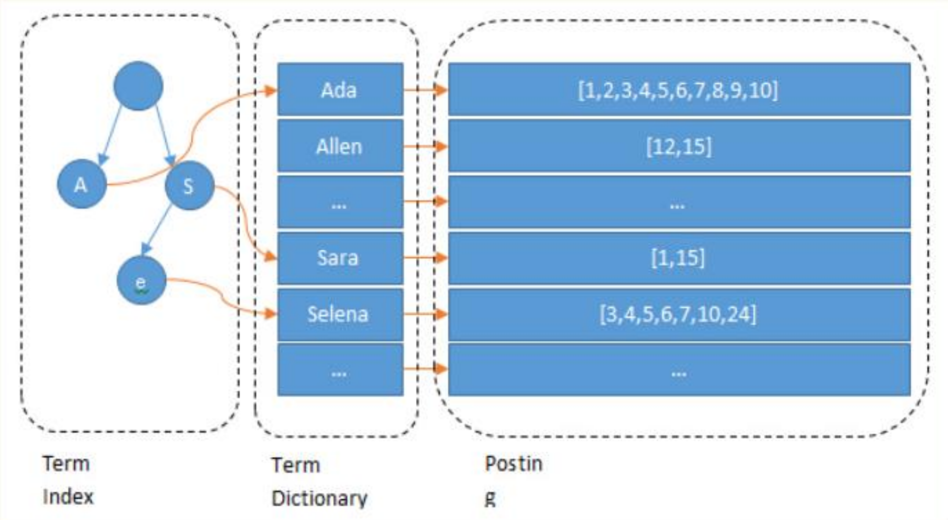
Name:

Term	Posting List
Kate	1

| John | 2 |
| Bill | 3 |
Age:
Term	Posting List
24	[1,2]
29	3
Sex:
Term	Posting List
Female	1
Male	[2,3]
Posting List
Elasticsearch 分别为每个 field 都建立了一个倒排索引，Kate, John, 24, Female 这些叫 term，而[1,2]就是 Posting List。Posting list 就是一个 int 的数组，存储了所有符合某个 term 的文档 id。
看到这里，不要认为就结束了，精彩的部分才刚开始...
通过 posting list 这种索引方式似乎可以很快进行查找，比如要找 age=24 的同学，爱回答问题的小明马上就举手回答：我知道，id 是 1，2 的同学。但是，如果这里有上千万的记录呢？如果是想通过 name 来查找呢？
Term Dictionary
Elasticsearch 为了能快速找到某个 term，将所有的 term 排个序，二分法查找 term，logN 的查找效率，就像通过字典查找一样，这就是 Term Dictionary。现在再看起来，似乎和传统数据库通过 B-Tree 的方式类似啊，为什么说比 B-Tree 的查询快呢？
Term Index
B-Tree 通过减少磁盘寻道次数来提高查询性能，Elasticsearch 也是采用同样的思路，直接通过内存查找 term，不读磁盘，但是如果 term 太多，term dictionary 也会很大，放内存不现实，于是有了 Term Index，就像字典里的索引页一样，A 开头的有哪些 term，分别在哪页，可以理解 term index 是一颗树：



这棵树不会包含所有的term，它包含的是term的一些前缀。通过term index可以快速地定位到term dictionary的某个offset，然后从这个位置再往后顺序查找。



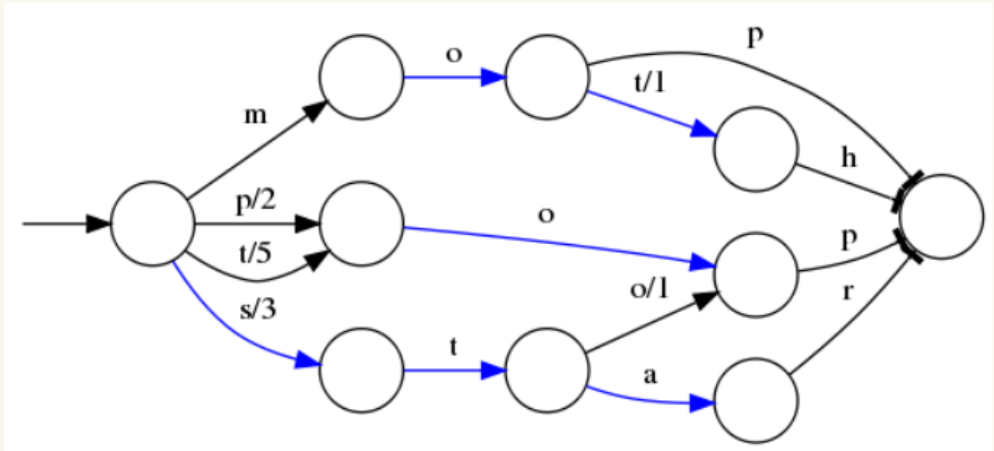
所以 term index 不需要存下所有的 term，而仅仅是他们的一些前缀与 Term Dictionary 的 block 之间的映射关系，再结合 FST(Finite State Transducers)的压缩技术，可以使 term index 缓存到内存中。从 term index 查到对应的 term dictionary 的 block 位置之后，再去磁盘上找 term，大大减少了磁盘随机读的次数。

这时候爱提问的小明又举手了:"那个 FST 是神马东东啊?"

一看就知道小明是一个上大学读书的时候跟我一样不认真听课的孩子，数据结构老师一定讲过什么是 FST。但没办法，我也忘了，这里再补下课：

FSTs are finite-state machines that map a term (byte sequence) to an arbitrary output.

假设我们现在要将 mop, moth, pop, star, stop and top(term index 里的 term 前缀)映射到序号：0，1，2，3，4，5(term dictionary 的 block 位置)。最简单的做法就是定义个 Map<string, integer="">，大家找到自己的位置对应入座就好了，但从内存占用少的角度想想，有没有更优的办法呢？答案就是：FST。



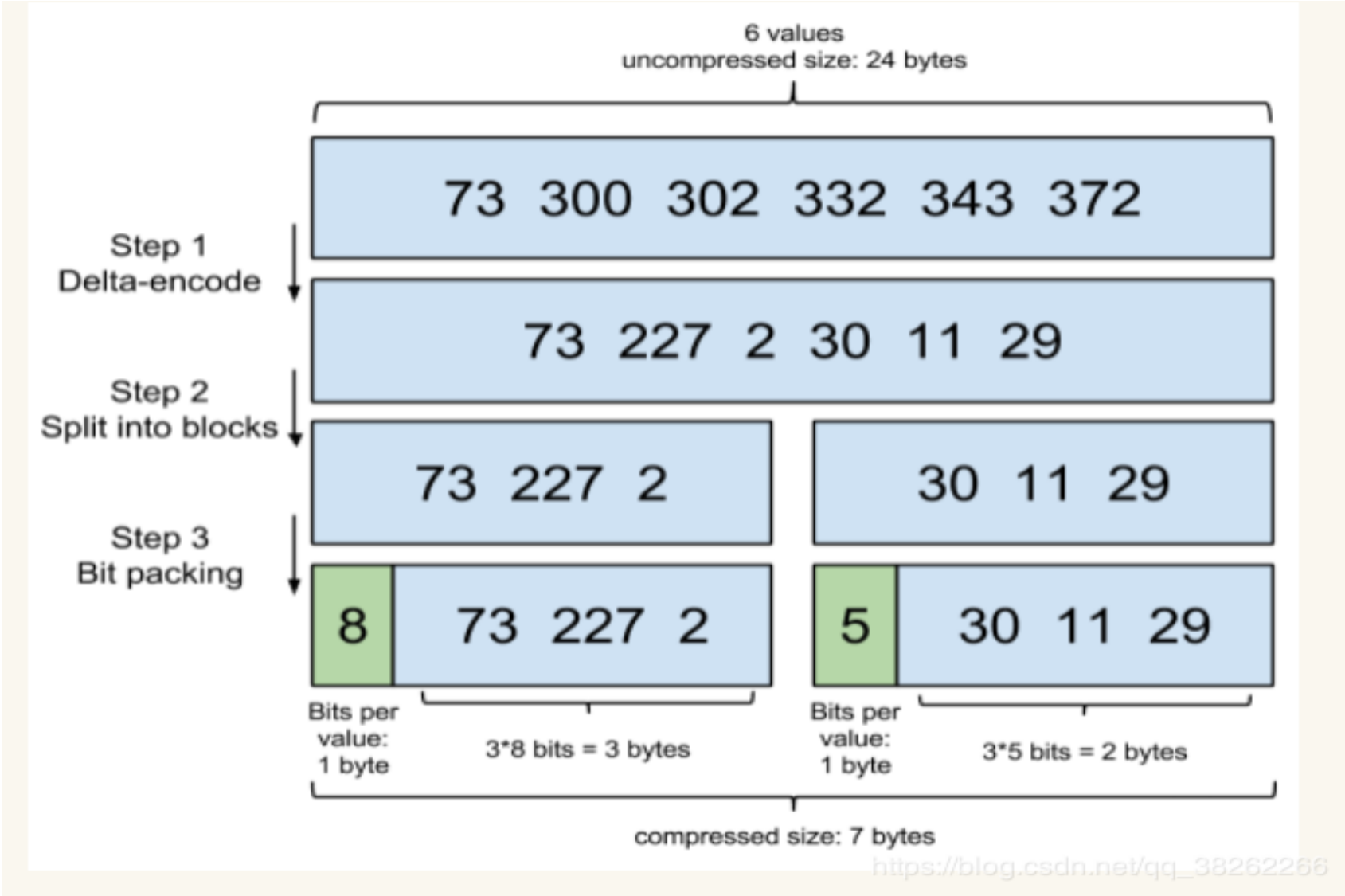
⭕表示一种状态

-->表示状态的变化过程，上面的字母/数字表示状态变化和权重

将单词分成单个字母通过⭕和-->表示出来，0权重不显示。如果⭕后面出现分支，就标记权重，最后整条路径上的权重加起来就是这个单词对应的序号。

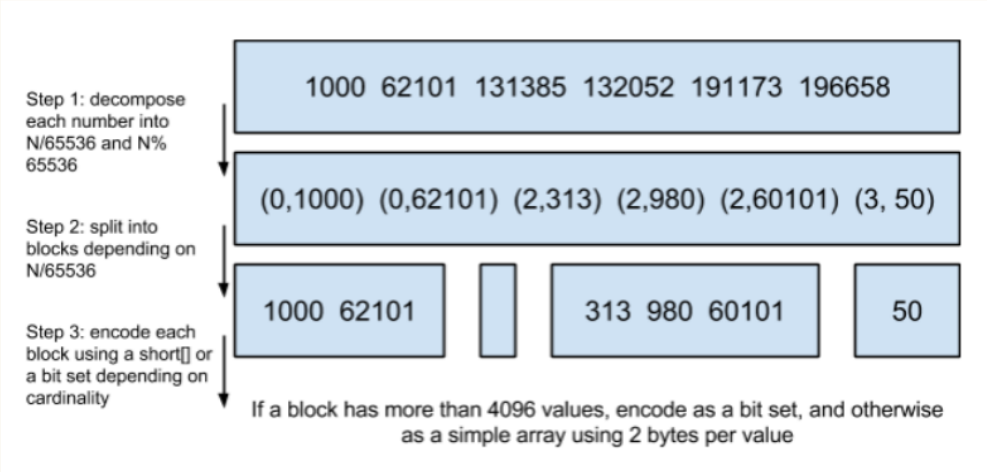
FSTs are finite-state machines that map a term (byte sequence) to an arbitrary output.

FSTs are finite-state machines that map a term (byte sequence) to an arbitrary output.
FST 以字节的方式存储所有的 term，这种压缩方式可以有效的缩减存储空间，使得 term index 足以放进内存，但这种方式也会导致查找时需要更多的 CPU 资源。
后面的更精彩，看累了的同学可以喝杯咖啡……
压缩技巧
Elasticsearch 里除了上面说到用 FST 压缩 term index 外，对 posting list 也有压缩技巧。
小明喝完咖啡又举手了:"posting list 不是已经只存储文档 id 了吗？还需要压缩？"
嗯，我们再看回最开始的例子，如果 Elasticsearch 需要对同学的性别进行索引(这时传统关系型数据库已经哭晕在厕所……)，会怎样？如果有上千万个同学，而世界上只有男/女这样两个性别，每个 posting list 都会有至少百万个文档 id。 Elasticsearch 是如何有效的对这些文档 id 压缩的呢？
Frame Of Reference
　　增量编码压缩，将大数变小数，按字节存储
首先，Elasticsearch 要求 posting list 是有序的(为了提高搜索的性能，再任性的要求也得满足)，这样做的一个好处是方便压缩，看下面这个图例：



如果数学不是体育老师教的话，还是比较容易看出来这种压缩技巧的。
原理就是通过增量，将原来的大数变成小数仅存储增量值，再精打细算按 bit 排好队，最后通过字节存储，而不是大大咧咧的尽管是 2 也是用 int(4 个字节)来存储。
Roaring bitmaps
说到 Roaring bitmaps，就必须先从 bitmap 说起。Bitmap 是一种数据结构，假设有某个 posting list：

[1,3,4,7,10]
对应的 bitmap 就是：
[1,0,1,1,0,0,1,0,0,1]
非常直观，用 0/1 表示某个值是否存在，比如 10 这个值就对应第 10 位，对应的 bit 值是 1，这样用一个字节就可以代表 8 个文档 id，旧版本(5.0 之前)的 Lucene 就是用这样的方式来压缩的，但这样的压缩方式仍然不够高效，如果有 1 亿个文档，那么需要 12.5MB 的存储空间，这仅仅是对应一个索引字段(我们往往会有很多个索引字段)。于是有人想出了 Roaring bitmaps 这样更高效的数据结构。
Bitmap 的缺点是存储空间随着文档个数线性增长，Roaring bitmaps 需要打破这个魔咒就一定要用到某些指数特性：
将 posting list 按照 65535 为界限分块，比如第一块所包含的文档 id 范围在 0~65535 之间，第二块的 id 范围是 65536~131071，以此类推。再用<商，余数>的组合表示每一组 id，这样每组里的 id 范围都在 0~65535 内了，剩下的就好办了，既然每组 id 不会变得无限大，那么我们就可以通过最有效的方式对这里的 id 存储。



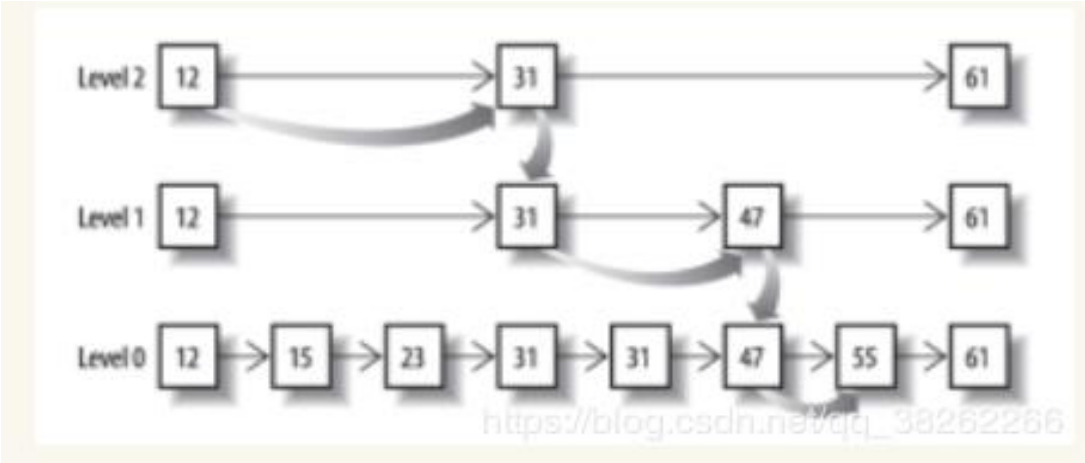
细心的小明这时候又举手了："为什么是以65535为界限?"

程序员的世界里除了1024外，65535也是一个经典值，因为它=2^16-1，正好是用2个字节能表示的最大数，一个short的存储单位，注意到上图里的最后一行"If a block has more than 4096 values, encode as a bit set, and otherwise as a simple array using 2 bytes per value"，如果是大块，用节省点用bitset存，小块就豪爽点，2个字节我也不计较了，用一个short[]存着方便。

那为什么用4096来区分大块还是小块呢?

个人理解：都说程序员的世界是二进制的，4096*2bytes = 8192bytes < 1KB，磁盘一次寻道可以顺序把一个小块的内容都读出来，再大一位就超过1KB了，需要两次读。

联合索引
上面说了半天都是单 field 索引，如果多个 field 索引的联合查询，倒排索引如何满足快速查询的要求呢?
　　利用跳表(Skip list)的数据结构快速做“与”运算，或者
　　利用上面提到的 bitset 按位“与”
先看看跳表的数据结构：



将一个有序链表 level0，挑出其中几个元素到 level1 及 level2，每个 level 越往上，选出来的指针元素越少，查找时依次从高 level 往低查找，比如 55，先找到 level2 的 31，再找到 level1 的 47，最后找到 55，一共 3 次查找，查找效率和 2 叉树的效率相当，但也是用了一定的空间冗余来换取的。
假设有下面三个 posting list 需要联合索引：
如果使用跳表，对最短的 posting list 中的每个 id，逐个在另外两个 posting list 中查找看是否存在，最后得到交集的结果。
如果使用 bitset，就很直观了，直接按位与，得到的结果就是最后的交集。

总结和思考

Elasticsearch 的索引思路:

将磁盘里的东西尽量搬进内存，减少磁盘随机读取次数(同时也利用磁盘顺序读特性)，结合各种奇技淫巧的压缩算法，用及其苛刻的态度使用内存。

所以，对于使用 Elasticsearch 进行索引时需要注意:

不需要索引的字段，一定要明确定义出来，因为默认是自动建索引的

同样的道理，对于 String 类型的字段，不需要 analysis 的也需要明确定义出来，因为默认也是会 analysis 的

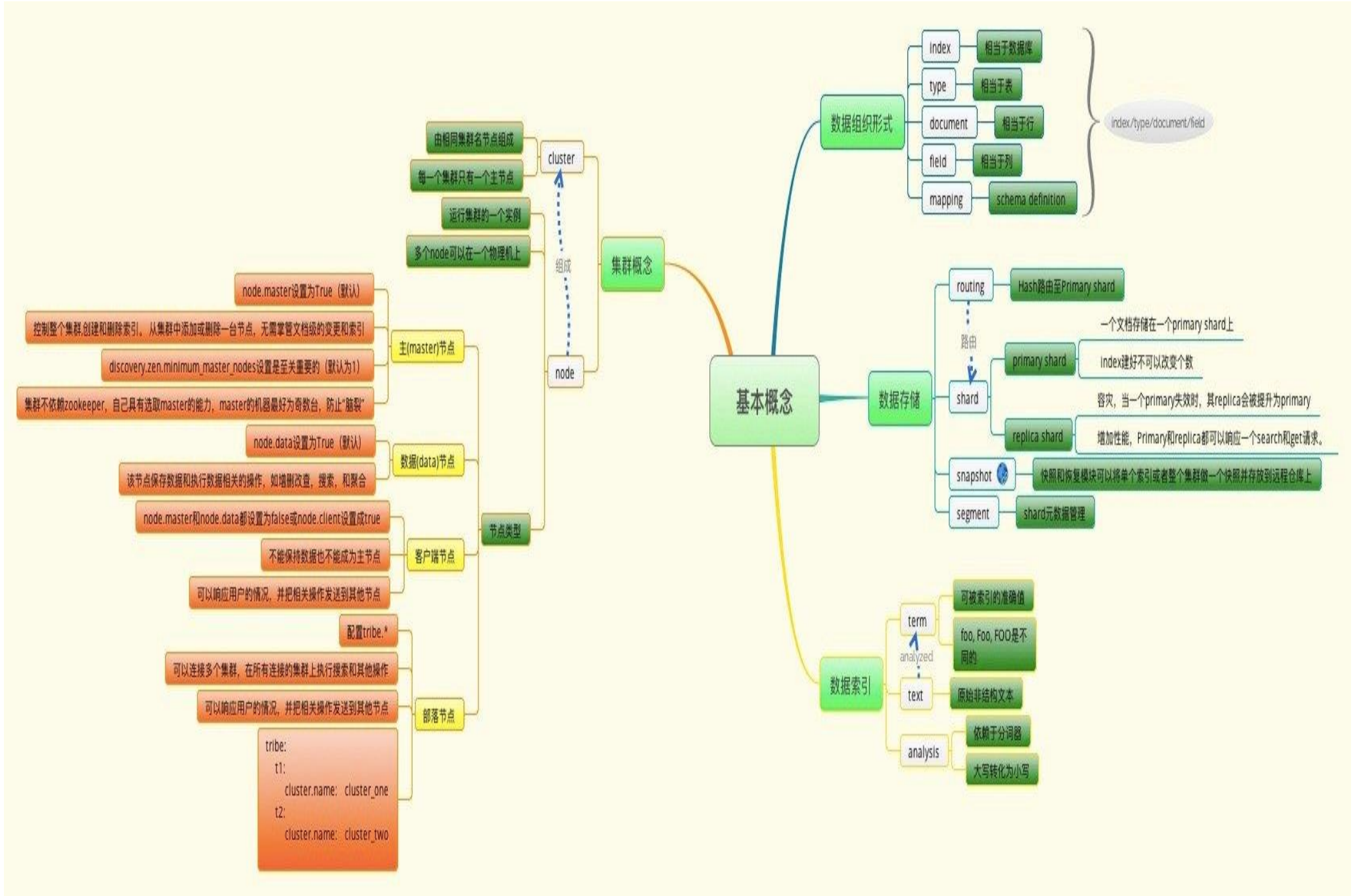
选择有规律的 ID 很重要，随机性太大的 ID(比如 java 的 UUID)不利于查询

关于最后一点，个人认为有多个因素:

其中一个(也许不是最重要的)因素：上面看到的压缩算法，都是对 Posting list 里的大量 ID 进行压缩的，那如果 ID 是顺序的，或者是有公共前缀等具有一定规律性的 ID，压缩比会比较高；

另外一个因素：可能是最影响查询性能的，应该是最后通过 Posting list 里的 ID 到磁盘中查找 Document 信息的那步，因为 Elasticsearch 是分 Segment 存储的，根据 ID 这个大范围的 Term 定位到 Segment 的效率直接影响了最后查询的性能，如果 ID 是有规律的，可以快速跳过不包含该 ID 的 Segment，从而减少不必要的磁盘读次数。

BAT 面试 Elasticsearch 必会知识点总结


基本概念
集群概念
cluster
由相同集群名节点组成
每一个集群只有一个主节点
组成
node
运行集群的一个实例
多个node可以在一个物理机上
节点类型
主(master)节点
node.master设置为True（默认）
控制整个集群,创建和删除索引。从集群中添加或删除一台节点，无需掌管文档级的变更和索引
discovery.zen.minimum_master_nodes设置是至关重要的（默认为1）
集群不依赖zookeeper，自己具有选取master的能力，master的机器最好为奇数台，防止"脑裂"
数据(data)节点
node.data设置为True（默认）
该节点保存数据和执行数据相关的操作，如增删改查，搜索，和聚合
客户端节点
node.master和node.data都设置为false或node.client设置成true
不能保持数据也不能成为主节点
可以响应用户的情况，并把相关操作发送到其他节点
部落节点
配置tribe.*
可以连接多个集群，在所有连接的集群上执行搜索和其他操作
可以响应用户的情况，并把相关操作发送到其他节点
tribe:
t1:
cluster.name: cluster_one
t2:
cluster.name: cluster_two
数据组织形式
index
相当于数据库
type
相当于表
document
相当于行
field
相当于列
mapping
schema definition
index/type/document/field
数据存储
routing
Hash路由至Primary shard
路由
shard
primary shard
一个文档存储在一个primary shard上
index建好不可以改变个数
replica shard
容灾，当一个primary失效时，其replica会被提升为primary
增加性能，Primary和replica都可以响应一个search和get请求。
snapshot
快照和恢复模块可以将单个索引或者整个集群做一个快照并存放到远程仓库上
segment
shard元数据管理
数据索引
term
可被索引的准确值
foo, Foo, FOO是不同的
analyzed
text
原始非结构文本
analysis
依赖于分词器
大写转化为小写

ElasticSearch 常见经典面试题

1.为什么要使用 Elasticsearch

因为在我们商城中的数据，将来会非常多，所以采用以往的模糊查询，模糊查询前置配置，会放弃索引，导致商品查询是全表扫面，在百万级别的数据库中，效率非常低下，而我们使用 ES 做一个全文索引，我们将经常查询的商品的某些字段，比如说商品名，描述、价格还有 id 这些字段我们放入我们索引库里，可以提高查询速度。

2.Elasticsearch 是如何实现 Master 选举的？

Elasticsearch 的选主是 ZenDiscovery 模块负责的，主要包含 Ping（节点之间通过这个 RPC 来发现彼此）和 Unicast（单播模块包含一个主机列表以控制哪些节点需要 ping 通）这两部分；

对所有可以成为 master 的节点（node.master: true）根据 nodeId 字典排序，每次选举每个节点都把自己所知道节点排一次序，然后选出第一个（第 0 位）节点，暂且认为它是 master 节点。

如果对某个节点的投票数达到一定的值（可以成为 master 节点数 n/2+1）并且该节点自己也选举自己，那这个节点就是 master。否则重新选举一直到满足上述条件。

补充：master 节点的职责主要包括集群、节点和索引的管理，不负责文档级别的管理；data 节点可以关闭 http 功能。

3.Elasticsearch 中的节点（比如共 20 个），其中的 10 个选了一个 master，另外 10 个选了另一个 master，怎么办？

当集群 master 候选数量不小于 3 个时，可以通过设置最少投票通过数量（discovery.zen.minimum_master_nodes）超过所有候选节点一半以上来解决脑裂问题；

当候选数量为两个时，只能修改为唯一的一个 master 候选，其他作为 data 节点，避免脑裂问题。

4.详细描述一下 Elasticsearch 索引文档的过程。

协调节点默认使用文档 ID 参与计算（也支持通过 routing），以便为路由提供合适的分片。

shard = hash(document_id) % (num_of_primary_shards)

当分片所在的节点接收到来自协调节点的请求后，会将请求写入到 Memory Buffer，然后定时（默认是每隔 1 秒）写入到 Filesystem Cache，这个从 Momery Buffer 到 Filesystem Cache 的过程就叫做 refresh；

当然在某些情况下，存在 Momery Buffer 和 Filesystem Cache 的数据可能会丢失，ES 是通过 translog 的机制来保证数据的可靠性的。其实现机制是接收到请求后，同时也会写入到 translog 中，当 Filesystem cache 中的数据写入到磁盘中时，才会清除掉，这个过程叫做 flush；

在 flush 过程中，内存中的缓冲将被清除，内容被写入一个新段，段的 fsync 将创建一个新的提交点，并将内容刷新到磁盘，旧的 translog 将被删除并开始一个新的 translog。

flush 触发的时机是定时触发（默认 30 分钟）或者 translog 变得太大（默认为 512M）时；

5.详细描述一下 Elasticsearch 更新和删除文档的过程

删除和更新也都是写操作，但是 Elasticsearch 中的文档是不可变的，因此不能被删除或者改动以展示其变更；

磁盘上的每个段都有一个相应的.del 文件。当删除请求发送后，文档并没有真的被删除，而是在.del 文件中被标记为删除。该文档依然能匹配查询，但是会在结果中被过滤掉。当段合并时，在.del 文件中被标记为删除的文档将不会被写入新段。

在新的文档被创建时，Elasticsearch 会为该文档指定一个版本号，当执行更新时，旧版本的文档在.del 文件中被标记为删除，新版本的文档被索引到一个新段。旧版本的文档依然能匹配查询，但是会在结果中被过滤掉。

6.详细描述一下 Elasticsearch 搜索的过程

搜索被执行成一个两阶段过程，我们称之为 Query Then Fetch；

在初始查询阶段时，查询会广播到索引中每一个分片拷贝（主分片或者副本分片）。 每个分片在本地执行搜索并构建一个匹配文档的大小为 from + size 的优先队列。

PS：在搜索的时候是会查询 Filesystem Cache 的，但是有部分数据还在 Memory Buffer，所以搜索是近实时的。

每个分片返回各自优先队列中 所有文档的 ID 和排序值 给协调节点，它合并这些值到自己的优先队列中来产生一个全局排序后的结果列表。

接下来就是 取回阶段，协调节点辨别出哪些文档需要被取回并向相关的分片提交多个 GET 请求。每个分片加载并 丰富 文档，如果有需要的话，接着返回文档给协调节点。一旦所有的文档都被取回了，协调节点返回结果给客户端。

补充：Query Then Fetch 的搜索类型在文档相关性打分的时候参考的是本分片的数据，这样在文档数量较少的时候可能不够准确，DFS Query Then Fetch 增加了一个预查询的处理，询问 Term 和 Document frequency，这个评分更准确，但是性能会变差。

9.Elasticsearch 对于大数据量（上亿量级）的聚合如何实现？

Elasticsearch 提供的首个近似聚合是 cardinality 度量。它提供一个字段的基数，即该字段的 distinct 或者 unique 值的数目。它是基于 HLL 算法的。HLL 会先对我们的输入作哈希运算，然后根据哈希运算的结果中的 bits 做概率估算从而得到基数。其特点是：可配置的精度，用来控制内存的使用（更精确 ＝ 更多内存）；小的数据集精度是非常高的；我们可以通过配置参数，来设置去重需要的固定内存使用量。无论数千还是数十亿的唯一值，内存使用量只与你配置的精确度相关 .

10.在并发情况下，Elasticsearch 如果保证读写一致？

可以通过版本号使用乐观并发控制，以确保新版本不会被旧版本覆盖，由应用层来处理具体的冲突；

另外对于写操作，一致性级别支持 quorum/one/all，默认为 quorum，即只有当大多数分片可用时才允许写操作。但即使大多数可用，也可能存在因为网络等原因导致写入副本失败，这样该副本被认为故障，分片将会在一个不同的节点上重建。

对于读操作，可以设置 replication 为 sync(默认)，这使得操作在主分片和副本分片都完成后才会返回；如果设置 replication 为 async 时，也可以通过设置搜索请求参数 _preference 为 primary 来查询主分片，确保文档是最新版本。

14.ElasticSearch 中的集群、节点、索引、文档、类型是什么？

群集是一个或多个节点（服务器）的集合，它们共同保存您的整个数据，并提供跨所有节点的联合索引和搜索功能。群集由唯一名称标识，默认情况下为“elasticsearch”。此名称很重要，因为如果节点设置为按名称加入群集，则该节点只能是群集的一部分。

节点是属于集群一部分的单个服务器。它存储数据并参与群集索引和搜索功能。

索引就像关系数据库中的“数据库”。它有一个定义多种类型的映射。索引是逻辑名称空间，映射到一个或多个主分片，并且可以有零个或多个副本分片。 MySQL =>数据库 ElasticSearch =>索引

文档类似于关系数据库中的一行。不同之处在于索引中的每个文档可以具有不同的结构（字段），但是对于通用字段应该具有相同的数据类型。 MySQL => Databases => Tables => Columns / Rows ElasticSearch => Indices => Types =>具有属性的文档

类型是索引的逻辑类别/分区，其语义完全取决于用户。

15.ElasticSearch 中的分片是什么?

在大多数环境中，每个节点都在单独的盒子或虚拟机上运行。

索引 - 在 Elasticsearch 中，索引是文档的集合。

分片 -因为 Elasticsearch 是一个分布式搜索引擎，所以索引通常被分割成分布在多个节点上的被称为分片的元素。

问题一：

什么是 ElasticSearch？

Elasticsearch 是一个基于 Lucene 的搜索引擎。它提供了具有 HTTP Web 界面和无架构 JSON 文档的分布式，多租户能力的全文搜索引擎。Elasticsearch 是用 Java 开发的，根据 Apache 许可条款作为开源发布。

问题三：

Elasticsearch 中的倒排索引是什么？

倒排索引是搜索引擎的核心。搜索引擎的主要目标是在查找发生搜索条件的文档时提供快速搜索。倒排索引是一种像数据结构一样的散列图，可将用户从单词导向文档或网页。它是搜索引擎的核心。其主要目标是快速搜索从数百万文件中查找数据。

问题四：

ElasticSearch 中的集群、节点、索引、文档、类型是什么？

- 群集是一个或多个节点（服务器）的集合，它们共同保存您的整个数据，并提供跨所有节点的联合索引和搜索功能。群集由唯一名称标识，默认情况下为“elasticsearch”。此名称很重要，因为如果节点设置为按名称加入群集，则该节点只能是群集的一部分。
- 节点是属于集群一部分的单个服务器。它存储数据并参与群集索引和搜索功能。
- 索引就像关系数据库中的“数据库”。它有一个定义多种类型的映射。索引是逻辑名称空间，映射到一个或多个主分片，并且可以有零个或多个副本分片。 MySQL =>数据库 ElasticSearch =>索引
- 文档类似于关系数据库中的一行。不同之处在于索引中的每个文档可以具有不同的结构（字段），但是对于通用字段应该具有相同的数据类型。 MySQL => Databases => Tables => Columns / Rows ElasticSearch => Indices => Types =>具有属性的文档
- 类型是索引的逻辑类别/分区，其语义完全取决于用户。

问题五：

ElasticSearch 是否有架构？

ElasticSearch 可以有一个架构。架构是描述文档类型以及如何处理文档的不同字段的一个或多个字段的描述。Elasticsearch 中的架构是一种映射，它描述了 JSON 文档中的字段及其数据类型，以及它们应该如何在 Lucene 索引中进行索引。因此，在 Elasticsearch 术语中，我们通常将此模式称为“映射”。

Elasticsearch 具有架构灵活的能力，这意味着可以在不明确提供架构的情况下索引文档。如果未指定映射，则默认情况下，Elasticsearch 会在索引期间检测文档中的新字段时动态生成一个映射。

问题六：

ElasticSearch 中的分片是什么？

在大多数环境中，每个节点都在单独的盒子或虚拟机上运行。

- 索引 - 在 Elasticsearch 中，索引是文档的集合。
- 分片 -因为 Elasticsearch 是一个分布式搜索引擎，所以索引通常被分割成分布在多个节点上的被称为分片的元素。

问题七：

ElasticSearch 中的副本是什么？

一个索引被分解成碎片以便于分发和扩展。副本是分片的副本。一个节点是一个属于一个集群的 ElasticSearch 的运行实例。一个集群由一个或多个共享相同集群名称的节点组成。

问题八：

ElasticSearch 中的分析器是什么？

在 ElasticSearch 中索引数据时，数据由为索引定义的 Analyzer 在内部进行转换。 分析器由一个 Tokenizer 和零个或多个 TokenFilter 组成。编译器可以在一个或多个 CharFilter 之前。分析模块允许您在逻辑名称下注册分析器，然后可以在映射定义或某些 API 中引用它们。

Elasticsearch 附带了许多可以随时使用的预建分析器。或者，您可以组合内置的字符过滤器，编译器和过滤器器来创建自定义分析器。

问题九：

什么是 ElasticSearch 中的编译器？

编译器用于将字符串分解为术语或标记流。一个简单的编译器可能会将字符串拆分为任何遇到空格或标点的地方。Elasticsearch 有许多内置标记器，可用于构建自定义分析器。

问题十一：
启用属性，索引和存储的用途是什么？
enabled 属性适用于各类 ElasticSearch 特定/创建领域，如 index 和 size。用户提供的字段没有"已启用"属性。 存储意味着数据由 Lucene 存储，如果询问，将返回这些数据。
存储字段不一定是可搜索的。默认情况下，字段不存储，但源文件是完整的。因为您希望使用默认值(这是有意义的)，所以不要设置 store 属性 该指数属性用于搜索。
索引属性只能用于搜索。只有索引域可以进行搜索。差异的原因是在分析期间对索引字段进行了转换，因此如果需要的话，您不能检索原始数据。
程序员的眼里，不止有代码和 bug，还有诗与远方和妹子！！！

ElasticSearch 面试题

1. 如何检查 *Elasticsearch* 服务器是否正在运行？

通常，*ElasticSearch* 使用 9200-9300 的端口范围。因此，要检查它是否在您的服务器上运行，只需键入主页的 URL，然后输入端口号。

例如：mysitename.com:9200

2. 列出 *Elasticsearch* 的不同类型的查询？

这些查询分为两种类型，在它们下面有多个查询分类：

- **基于全文检索**：匹配查询、匹配词组查询、多匹配查询、匹配词组前缀查询、常用词查询、查询字符串查询、简单查询字符串查询。
- **基于词条检索**：词条查询、词条集查询、范围查询、前缀查询、通配符查询、regexp 查询、模糊查询、存在查询、类型查询、id 查询

3. 如何在 *Elasticsearch* 群集中添加或创建索引？

通过在索引名称之前使用命令 PUT，创建索引，如果要添加另一个索引，则在索引名称前使用命令 POST。

例如：put 网站

创建名为 computer 的索引

4. 什么是文档？ *Elasticsearch* 中的文档与关系数据库中的行类似。唯一的区别是索引中的每个文档都可以具有不同的结构或字段但对于公共字段必须具有相同的数据类型。文档中的每个数据类型不同的字段可以出现多次。这些字段还可以包含其他文档

5. *Elasticsearch* 中的群集是什么？

它是由一个或多个节点或服务器组成的集合或集合，这些节点或服务器用来保存完整的数据，并提供跨所有节点的联合索引和搜索功能。它由一个不同且唯一的名称标识，默认为"*ElasticSearch*"。

此名称被认为是重要的，因为只有将节点设置为按名称加入集群时，它才能成为集群的一部分。

6. *Elasticsearch* 数据储存在哪里？*ElasticSearch* 是一个带有多个目录的分布式文档存储。它可以实时存储和检索序列化为 JSON 文档的复杂数据结构。

7. 列出安装 *Elasticsearch* 的软件要求？由于 *Elasticsearch* 是使用 Java 构建的，因此我们需要以下任何软件才能在我们的设备上运行 *Elasticsearch*

Java 8 系列的最新版本建议使用 Java 版本 1.8.0_131

8. 在 *ElasticSearch* 中聚合是如何工作的？聚合框架提供基于搜索查询的聚合数据。它可以看作是一个工作单元，用于在一组文档上构建分析信息。聚合的类型不同，用途和输出也不同。

9. 什么是 *Elasticsearch*？*ElasticSearch* 是一种基于 Lucene 的搜索引擎，它提供了一个分布式的、多用户的、具有 HTTP（超文本传输协议）Web 界面和无架构 JSON（JavaScriptObject Notation）文档的全文搜索引擎。它是用 Java 开发的，是在 APACHE 许可下发布的一个开放源代码。

10. 什么是 *SHARDS*？由于应用程序在不同的机器上使用了多个 *ElasticSearch* 实例，因此在扩展方面存在诸如 RAM、VCPU 等资源限制。索引中的数据可以分为多个部分，由一个单独的 *ElasticSearch* 节点或实例管理。每个部分称为一个 SHARDS。默认情况下，ElasticSearch 索引有 5 个 *SHARDS*。

11. 您能列出一些使用 *Elasticsearch* 的公司吗？

一些使用 *Elasticsearch* 以及 Logstash 和 Kibana 的公司是：

- Wikipedia
- Netflix
- Accenture
- Stack Overflow
- Fujitsu
- Tripwire
- Medium
- Swat.io
- Hip chat
- IFTTT

12. 如何在 ES 中列出集群的所有索引？通过使用 *GET /_index name/ indices*，我们可以获得集群中存在的索引列表

13. 什么是 *REPLICAS*？*Elasticsearch* 中的每个分片再次具有两个称为副本的分片副本。它们服务于容错和高可用性的目的

14. 如何在 *Elasticsearch* 中完成相关性和评分？

Lucene 使用布尔模型来查找类似文档，并使用一个称为实用评分函数的公式来计算相关性。该公式从逆文档/术语 - 文档频率和向量空间模型中复制概念，并添加了细条印子字段长度归一化等现代特征。

得分（q，d）是查询“q”是文档“d”的相关得分。

15. *Elasticsearch* 可以取代数据库吗？是的，*Elasticsearch* 可以用作数据库的替代品，因为 *Elasticsearch* 非常强大。它提供了多用户，分片和备份，分发和云实时获取，刷新，提交，版本控制和重新索引等功能，使其成为数据库的适当替代品。

16. *Elasticsearch* 中如何删除索引？ 要在 *ElasticSearch* 中删除索引，请使用命令 delete/index name。例如：DELETE /website（删除/网站）

17. 如何在索引中添加映射？

基本上，*Elasticsearch* 将根据用户在请求正文中提供的数据自动创建映射。其批量功能可用于在索引中添加多个 *JSON* 对象。

例如：POST 网站/ _bulk

18. 如何在 ES 中按 ID 检索文档？要在 ElasticSearch 中检索文档，我们使用 get 动词，后跟_index、_type、_id。

例如：get/computer/blog/123？=漂亮

19. 什么是索引？*ElasticSearch* 中的索引类似于关系数据库中的表，唯一的区别在于将实际值存储在关系数据库中，而 *ElasticSearch* 中的索引是可选的。索引能够在索引中存储实际值或分析值。

20. 如何启动 *Elasticsearch* 服务器？

在终端上运行以下命令以启动 *Elasticsearch* 服务器：

```
cd elasticsearch
./bin/elasticsearch
```

curl’http：// localhost：9200 /？pretty’命令用于检查 *ElasticSearch* 服务器是否正在运行

21. *Elasticsearch* 的当前稳定版本是什么？截至 2018 年 3 月，版本 6.2.2 是 Elasticsearch 的最新稳定版本。

22. 在 *Elasticsearch* 中搜索的方式有哪些？

我们可以在 Elasticsearch 中执行以下搜索：

- 多索引，多类型搜索：所有搜索 API 都可以应用于所有多个索引，并支持多索引系统

我们可以搜索所有索引中的某些标签以及所有索引和所有类型的所有标签。

- URI 搜索：通过提供请求参数，纯粹使用 URI 执行搜索请求。
- 请求正文搜索：搜索请求可以由搜索 DSL 执行，搜索 DSL 包括正文中的查询 DSL。

23. 什么是 *Elasticsearch*？*Elasticsearch* 中的类型是索引的逻辑类别，其语义完全取决于用户。

24. 基于词条的查询和全文的查询有什么区别？

- 基于词条的查询：词条查询或模糊查询等查询是没有分析阶段的低级查询。词条查询术语 Foo 在倒排索引中搜索确切的词条并计算 IDF / TF 相关性分数对于每个有词条的文档。
- 全文查询：匹配查询或查询字符串查询等查询是了解字段映射的高级查询。只要查询汇总了完整的项目列表，它就会为每个项执行适当的低级查询，最后结合他们的结果来产生每个文档的相关性分数。

25. 什么是节点？ *Elasticsearch* 的每个实例都是一个节点，多个节点的集合可以协调工作，形成一个 *Elasticsearch* 集群

26. 请解析映射？

影射是定义文档如何映射到搜索引擎的过程，包括可搜索的特征，例如那些字段是标记以即刻搜索的。

在 *Elasticsearch* 中，创建的索引可能包含所有“映射类型”的文档。

新年手打，24 道进阶必备 Elasticsearch 面试真题（建议收藏!）

Elasticsearch 面试题

1、elasticsearch 了解多少，说说你们公司 es 的集群架构，索引数据大小，分片有多少，以及一些调优手段 。

面试官：想了解应聘者之前公司接触的 ES 使用场景、规模，有没有做过比较大规模的索引设计、规划、调优。

解答：

如实结合自己的实践场景回答即可。

比如：ES 集群架构 13 个节点，索引根据通道不同共 20+索引，根据日期，每日递增 20+，索引：10 分片，每日递增 1 亿+数据，

每个通道每天索引大小控制：150GB 之内。

仅索引层面调优手段：

1.1、设计阶段调优

1、根据业务增量需求，采取基于日期模板创建索引，通过 roll over API 滚动索引；

2、使用别名进行索引管理；

3、每天凌晨定时对索引做 force_merge 操作，以释放空间；

4、采取冷热分离机制，热数据存储到 SSD，提高检索效率；冷数据定期进行 shrink

操作，以缩减存储；
5、采取 curator 进行索引的生命周期管理；
6、仅针对需要分词的字段，合理的设置分词器；
7、Mapping 阶段充分结合各个字段的属性，是否需要检索、是否需要存储等。.....

1.2、写入调优

1、写入前副本数设置为 0；
2、写入前关闭 refresh_interval 设置为-1，禁用刷新机制；
3、写入过程中：采取 bulk 批量写入；
4、写入后恢复副本数和刷新间隔；
5、尽量使用自动生成的 id。

1.3、查询调优

1、禁用 wildcard；
2、禁用批量 terms（成百上千的场景）；
3、充分利用倒排索引机制，能 keyword 类型尽量 keyword；
4、数据量大时候，可以先基于时间敲定索引再检索；
5、设置合理的路由机制。

1.4、其他调优

部署调优，业务调优等。
上面的提及一部分，面试者就基本对你之前的实践或者运维经验有所评估了。

2、elasticsearch 的倒排索引是什么

面试官：想了解你对基础概念的认知。
解答：通俗解释一下就可以。
传统的我们的检索是通过文章，逐个遍历找到对应关键词的位置。
而倒排索引，是通过分词策略，形成了词和文章的映射关系表，这种词典+映射表即为倒排索引。
有了倒排索引，就能实现 o（1）时间复杂度的效率检索文章了，极大的提高了检索效率。

学术的解答方式：
倒排索引，相反于一篇文章包含了哪些词，它从词出发，记载了这个词在哪些文档中出现过，由两部分组成——词典和倒排表。
加分项：倒排索引的底层实现是基于：FST（Finite State Transducer）数据结构。
lucene 从 4+版本后开始大量使用的数据结构是 FST。FST 有两个优点：
1、空间占用小。通过对词典中单词前缀和后缀的重复利用，压缩了存储空间；
2、查询速度快。O(len(str))的查询时间复杂度。

3、elasticsearch 索引数据多了怎么办，如何调优，部署

面试官：想了解大数据量的运维能力。
解答：索引数据的规划，应在前期做好规划，正所谓“设计先行，编码在后”，这样才能有效的避免突如其来的数据激增导致集群处理能力不足引发的线上客户检索或者其他业务受到影响。
如何调优，正如问题 1 所说，这里细化一下：

3.1 动态索引层面

基于模板+时间+rollover api 滚动创建索引，举例：设计阶段定义：blog 索引的模板格式为：blog_index_时间戳的形式，每天递增数据。
这样做的好处：不至于数据量激增导致单个索引数据量非常大，接近于上线 2 的 32 次幂-1，索引存储达到了 TB+甚至更大。
一旦单个索引很大，存储等各种风险也随之而来，所以要提前考虑+及早避免。

3.2 存储层面

冷热数据分离存储，热数据（比如最近 3 天或者一周的数据），其余为冷数据。
对于冷数据不会再写入新数据，可以考虑定期 force_merge 加 shrink 压缩操作，节省存储空间和检索效率。

3.3 部署层面

一旦之前没有规划，这里就属于应急策略。
结合 ES 自身的支持动态扩展的特点，动态新增机器的方式可以缓解集群压力，注意：如果之前主节点等规划合理，不需要重启集群也能完成动态新增的。

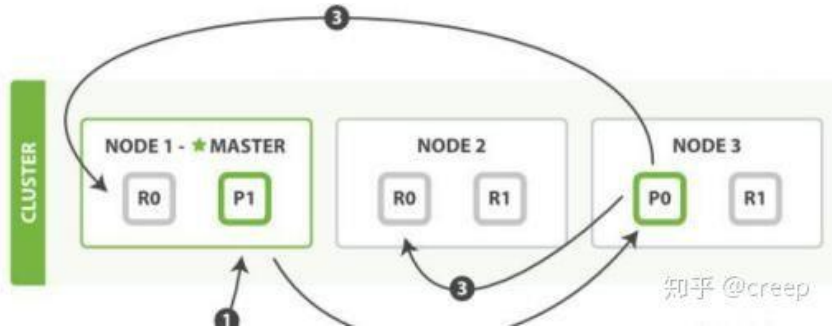
4、elasticsearch 是如何实现 master 选举的

面试官：想了解 ES 集群的底层原理，不再只关注业务层面了。
解答：
前置前提：
1、只有候选主节点（master：true）的节点才能成为主节点。
2、最小主节点数（min_master_nodes）的目的是防止脑裂。
这个我看了各种网上分析的版本和源码分析的书籍，云里雾里。
核对了一下代码，核心入口为 findMaster，选择主节点成功返回对应 Master，否则返回 null。选举流程大致描述如下：
第一步：确认候选主节点数达标，elasticsearch.yml 设置的值 discovery.zen.minimum_master_nodes；
第二步：比较：先判定是否具备 master 资格，具备候选主节点资格的优先返回；若两节点都为候选主节点，则 id 小的值会主节点。注意这里的 id 为 string 类型。
题外话：获取节点 id 的方法。

5、详细描述一下 Elasticsearch 索引文档的过程

面试官：想了解 ES 的底层原理，不再只关注业务层面了。
解答：
这里的索引文档应该理解为文档写入 ES，创建索引的过程。
文档写入包含：单文档写入和批量 bulk 写入，这里只解释一下：单文档写入流程。

记住官方文档中的这个图。



第一步：客户写集群某节点写入数据，发送请求。（如果没有指定路由/协调节点，请求的节点扮演路由节点的角色。）
第二步：节点 1 接受到请求后，使用文档_id 来确定文档属于分片 0。请求会被转到另外的节点，假定节点 3。因此分片 0 的主分片分配到节点 3 上。
第三步：节点 3 在主分片上执行写操作，如果成功，则将请求并行转发到节点 1 和节点 2 的副本分片上，等待结果返回。所有的副本分片都报告成功，节点 3 将向协调节点（节点 1）报告成功，节点 1 向请求客户端报告写入成功。
如果面试官再问：第二步中的文档获取分片的过程?
回答：借助路由算法获取，路由算法就是根据路由和文档 id 计算目标的分片 id 的过程。
1shard = hash(_routing) % (num_of_primary_shards)

6、详细描述一下 Elasticsearch 搜索的过程?

面试官：想了解 ES 搜索的底层原理，不再只关注业务层面了。
解答：
搜索拆解为“query then fetch” 两个阶段。
query 阶段的目的：定位到位置，但不取。
步骤拆解如下：
1、假设一个索引数据有 5 主+1 副本 共 10 分片，一次请求会命中（主或者副本分片中）的一个。
2、每个分片在本地进行查询，结果返回到本地有序的优先队列中。
3、第 （2）步骤的结果发送到协调节点，协调节点产生一个全局的排序列表。
fetch 阶段的目的：取数据。
路由节点获取所有文档，返回给客户端。

7、Elasticsearch 在部署时，对 Linux 的设置有哪些优化方法

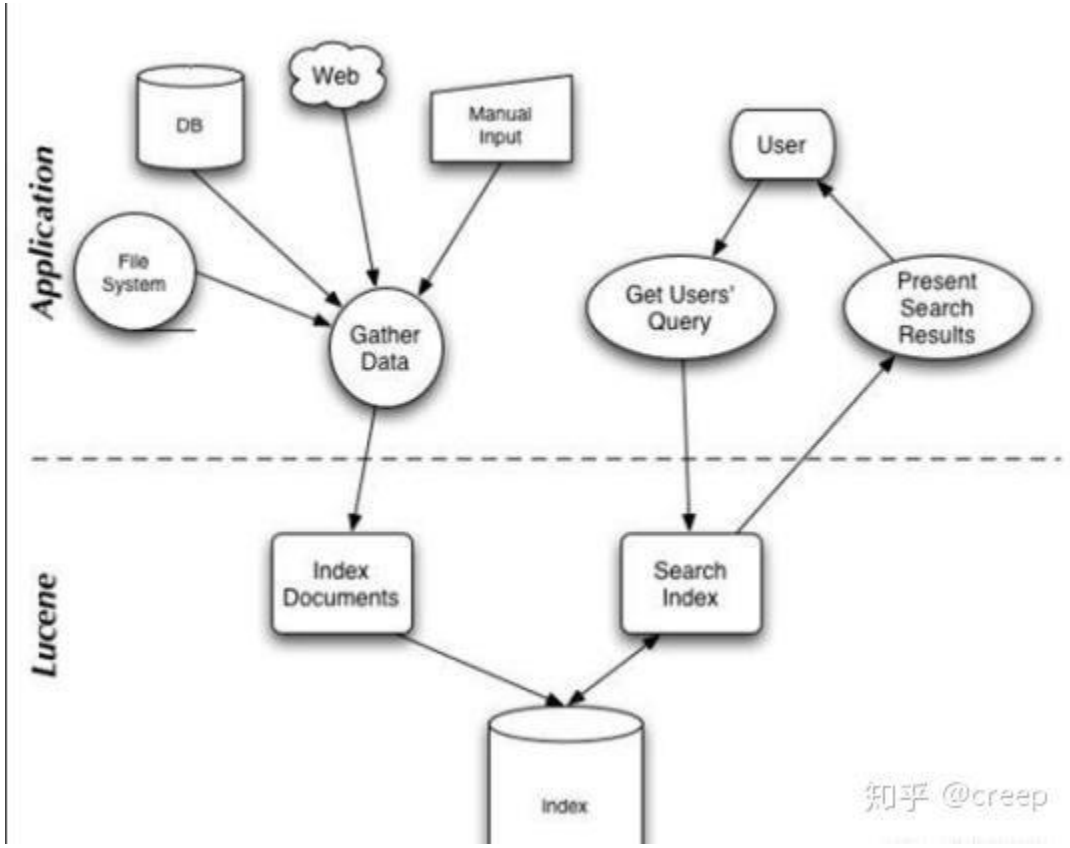
面试官：想了解对 ES 集群的运维能力。
解答：
1、关闭缓存 swap;
2、堆内存设置为：Min（节点内存/2, 32GB）;
3、设置最大文件句柄数;
4、线程池+队列大小根据业务需要做调整;
5、磁盘存储 raid 方式——存储有条件使用 RAID10，增加单节点性能以及避免单

节点存储故障。

8、lucence 内部结构是什么?

面试官：想了解你的知识面的广度和深度。

解答：



Lucene 是有索引和搜索的两个过程，包含索引创建，索引，搜索三个要点。可以基于这个脉络展开一些。

最近面试一些公司，被问到的关于 Elasticsearch 和搜索引擎相关的问题，以及自己总结的回答。

9、Elasticsearch 是如何实现 Master 选举的?

1、Elasticsearch 的选主是 ZenDiscovery 模块负责的，主要包含 Ping（节点之

间通过这个 RPC 来发现彼此）和 Unicast（单播模块包含一个主机列表以控制哪些节点需要 ping 通）这两部分；
2、对所有可以成为 master 的节点（**node.master: true**）根据 nodeId 字典排序，每次选举每个节点都把自己所知道节点排一次序，然后选出第一个（第 0 位）节点，暂且认为它是 master 节点。
3、如果对某个节点的投票数达到一定的值（可以成为 master 节点数 n/2+1）并且该节点自己也选举自己，那这个节点就是 master。否则重新选举一直到满足上述条件。
4、补充：master 节点的职责主要包括集群、节点和索引的管理，不负责文档级别的管理；data 节点可以关闭 http 功能*。

10、Elasticsearch 中的节点（比如共 20 个），其中的 10 个选了一个 master，另外 10 个选了另一个 master，怎么办?

1、当集群 master 候选数量不小于 3 个时，可以通过设置最少投票通过数量（**discovery.zen.minimum_master_nodes**）超过所有候选节点一半以上来解决脑裂问题；
2、当候选数量为两个时，只能修改为唯一的一个 master 候选，其他作为 data 节点，避免脑裂问题。

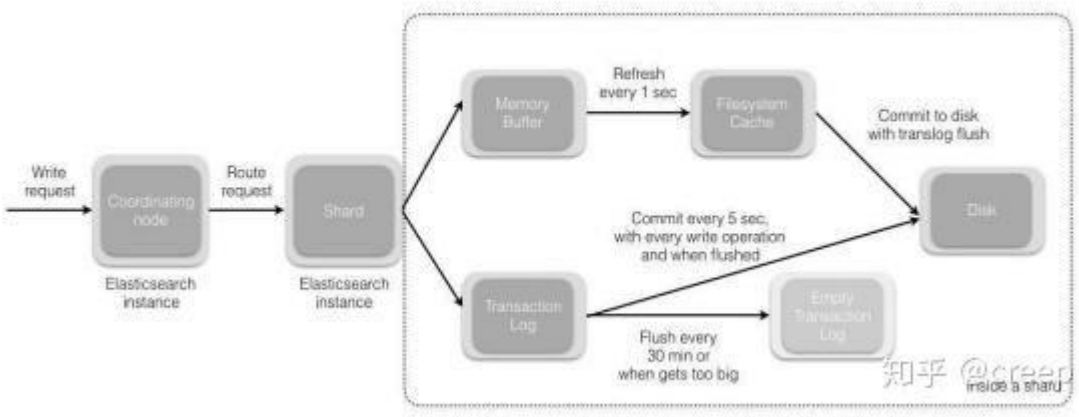
11、客户端在和集群连接时，如何选择特定的节点执行请求的?

1、TransportClient 利用 transport 模块远程连接一个 elasticsearch 集群。它并不加入到集群中，只是简单的获得一个或者多个初始化的 transport 地址，并以 **轮询** 的方式与这些地址进行通信。

12、详细描述一下 Elasticsearch 索引文档的过程。

协调节点默认使用文档 ID 参与计算（也支持通过 routing），以便为路由提供合适的分片。
shard = hash(document_id) % (num_of_primary_shards)
1、当分片所在的节点接收到来自协调节点的请求后，会将请求写入到 Memory Buffer，然后定时（默认是每隔 1 秒）写入到 Filesystem Cache，这个从 Momery Buffer 到 Filesystem Cache 的过程就叫做 refresh；
2、当然在某些情况下，存在 Momery Buffer 和 Filesystem Cache 的数据可能会丢失，ES 是通过 translog 的机制来保证数据的可靠性的。其实现机制是接收到请求后，同时也会写入到 translog 中，当 Filesystem cache 中的数据写入到磁盘中

时，才会清除掉，这个过程叫做 flush；
3、在 flush 过程中，内存中的缓冲将被清除，内容被写入一个新段，段的 fsync 将创建一个新的提交点，并将内容刷新到磁盘，旧的 translog 将被删除并开始一个新的 translog。
4、flush 触发的时机是定时触发（默认 30 分钟）或者 translog 变得太大（默认为 512M）时；



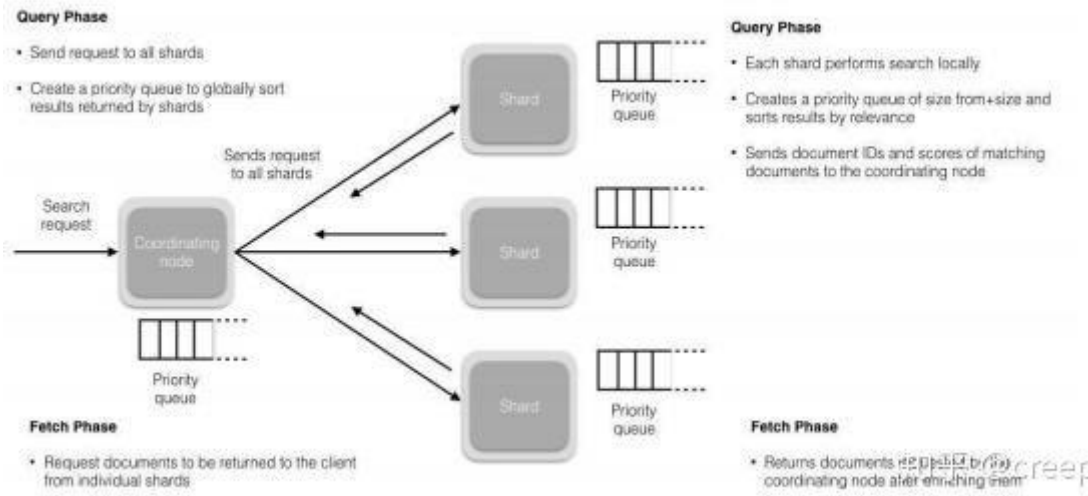
补充：关于 Lucene 的 Segement：
1、Lucene 索引是由多个段组成，段本身是一个功能齐全的倒排索引。
2、段是不可变的，允许 Lucene 将新的文档增量地添加到索引中，而不用从头重建索引。
3、对于每一个搜索请求而言，索引中的所有段都会被搜索，并且每个段会消耗 CPU 的时钟周、文件句柄和内存。这意味着段的数量越多，搜索性能会越低。
4、为了解决这个问题，Elasticsearch 会合并小段到一个较大的段，提交新的合并段到磁盘，并删除那些旧的小段。

13、详细描述一下 Elasticsearch 更新和删除文档的过程。
1、删除和更新也都是写操作，但是 Elasticsearch 中的文档是不可变的，因此不能被删除或者改动以展示其变更；
2、磁盘上的每个段都有一个相应的.del 文件。当删除请求发送后，文档并没有真的被删除，而是在.del 文件中被标记为删除。该文档依然能匹配查询，但是会在结果中被过滤掉。当段合并时，在.del 文件中被标记为删除的文档将不会被写入

新段。
3、在新的文档被创建时，Elasticsearch 会为该文档指定一个版本号，当执行更新时，旧版本的文档在.del 文件中被标记为删除，新版本的文档被索引到一个新段。旧版本的文档依然能匹配查询，但是会在结果中被过滤掉。

14、详细描述一下 Elasticsearch 搜索的过程。

1、搜索被执行成一个两阶段过程，我们称之为 Query Then Fetch；
2、在初始**查询阶段**时，查询会广播到索引中每一个分片拷贝（主分片或者副本分片）。 每个分片在本地执行搜索并构建一个匹配文档的大小为 from + size 的优先队列。
PS：在搜索的时候是会查询 Filesystem Cache 的，但是有部分数据还在 Memory Buffer，所以搜索是近实时的。
3、每个分片返回各自优先队列中 **所有文档的 ID 和排序值** 给协调节点，它合并这些值到自己的优先队列中来产生一个全局排序后的结果列表。
4、接下来就是 **取回阶段**，协调节点辨别出哪些文档需要被取回并向相关的分片提交多个 GET 请求。每个分片加载并 丰富 文档，如果有需要的话，接着返回文档给协调节点。一旦所有的文档都被取回了，协调节点返回结果给客户端。
5、补充：Query Then Fetch 的搜索类型在文档相关性打分的时候参考的是本分片的数据，这样在文档数量较少的时候可能不够准确，DFS Query Then Fetch 增加了一个预查询的处理，询问 Term 和 Document frequency，这个评分更准确，但是性能会变差。*



15、在 Elasticsearch 中，是怎么根据一个词找到对应的倒排索引的?

SEE:

- Lucene 的索引文件格式(1)
- Lucene 的索引文件格式(2)

16、Elasticsearch 在部署时，对 Linux 的设置有哪些优化方法?

1、64 GB 内存的机器是非常理想的， 但是 32 GB 和 16 GB 机器也是很常见的。少于 8 GB 会适得其反。

2、如果你要在更快的 CPUs 和更多的核心之间选择，选择更多的核心更好。多个内核提供的额外并发远胜过稍微快一点点的时钟频率。

3、如果你负担得起 SSD，它将远远超出任何旋转介质。 基于 SSD 的节点，查询和索引性能都有提升。如果你负担得起，SSD 是一个好的选择。

4、即使数据中心们近在咫尺，也要避免集群跨越多个数据中心。绝对要避免集群跨越大的地理距离。

5、请确保运行你应用程序的 JVM 和服务器的 JVM 是完全一样的。 在 Elasticsearch 的几个地方，使用 Java 的本地序列化。
6、通过设置 gateway.recover_after_nodes、gateway.expected_nodes、gateway.recover_after_time 可以在集群重启的时候避免过多的分片交换，这可能会让数据恢复从数个小时缩短为几秒钟。
7、Elasticsearch 默认被配置为使用单播发现，以防止节点无意中加入集群。只有在同一台机器上运行的节点才会自动组成集群。最好使用单播代替组播。
8、不要随意修改垃圾回收器（CMS）和各个线程池的大小。
9、把你的内存的（少于）一半给 Lucene（但不要超过 32 GB！），通过 ES_HEAP_SIZE 环境变量设置。
10、内存交换到磁盘对服务器性能来说是致命的。如果内存交换到磁盘上，一个 100 微秒的操作可能变成 10 毫秒。 再想想那么多 10 微秒的操作时延累加起来。 不难看出 swapping 对于性能是多么可怕。
11、Lucene 使用了大量 的文件。同时，Elasticsearch 在节点和 HTTP 客户端之间进行通信也使用了大量的套接字。 所有这一切都需要足够的文件描述符。你应该增加你的文件描述符，设置一个很大的值，如 64,000。

补充：索引阶段性能提升方法

1、使用批量请求并调整其大小：每次批量数据 5–15 MB 大是个不错的起始点。
2、存储：使用 SSD
3、段和合并：Elasticsearch 默认值是 20 MB/s，对机械磁盘应该是个不错的设置。如果你用的是 SSD，可以考虑提高到 100–200 MB/s。如果你在做批量导入，完全不在意搜索，你可以彻底关掉合并限流。另外还可以增加 index.translog.flush_threshold_size 设置，从默认的 512 MB 到更大一些的值，比如 1 GB，这可以在一次清空触发的时候在事务日志里积累出更大的段。
4、如果你的搜索结果不需要近实时的准确度，考虑把每个索引的 index.refresh_interval 改到 30s。
5、如果你在做大批量导入，考虑通过设置 index.number_of_replicas: 0 关闭副本。

17、对于 GC 方面，在使用 Elasticsearch 时要注意什么?

1、SEE：elasticsearch.cn/articl
2、倒排词典的索引需要常驻内存，无法 GC，需要监控 data node 上 segment

memory 增长趋势。
3、各类缓存，field cache, filter cache, indexing cache, bulk queue 等等，要设置合理的大小，并且要应该根据最坏的情况来看 heap 是否够用，也就是各类缓存全部占满的时候，还有 heap 空间可以分配给其他任务吗？避免采用 clear cache 等“自欺欺人”的方式来释放内存。
4、避免返回大量结果集的搜索与聚合。确实需要大量拉取数据的场景，可以采用 scan & scroll api 来实现。
5、cluster stats 驻留内存并无法水平扩展，超大规模集群可以考虑分拆成多个集群通过 tribe node 连接。
6、想知道 heap 够不够，必须结合实际应用场景，并对集群的 heap 使用情况做持续的监控。

18、Elasticsearch 对于大数据量（上亿量级）的聚合如何实现?

Elasticsearch 提供的首个近似聚合是 cardinality 度量。它提供一个字段的基数，即该字段的 distinct 或者 unique 值的数目。它是基于 HLL 算法的。HLL 会先对我们的输入作哈希运算，然后根据哈希运算的结果中的 bits 做概率估算从而得到基数。其特点是：可配置的精度，用来控制内存的使用（更精确 = 更多内存）；小的数据集精度是非常高的；我们可以通过配置参数，来设置去重需要的固定内存使用量。无论数千还是数十亿的唯一值，内存使用量只与你配置的精确度相关。

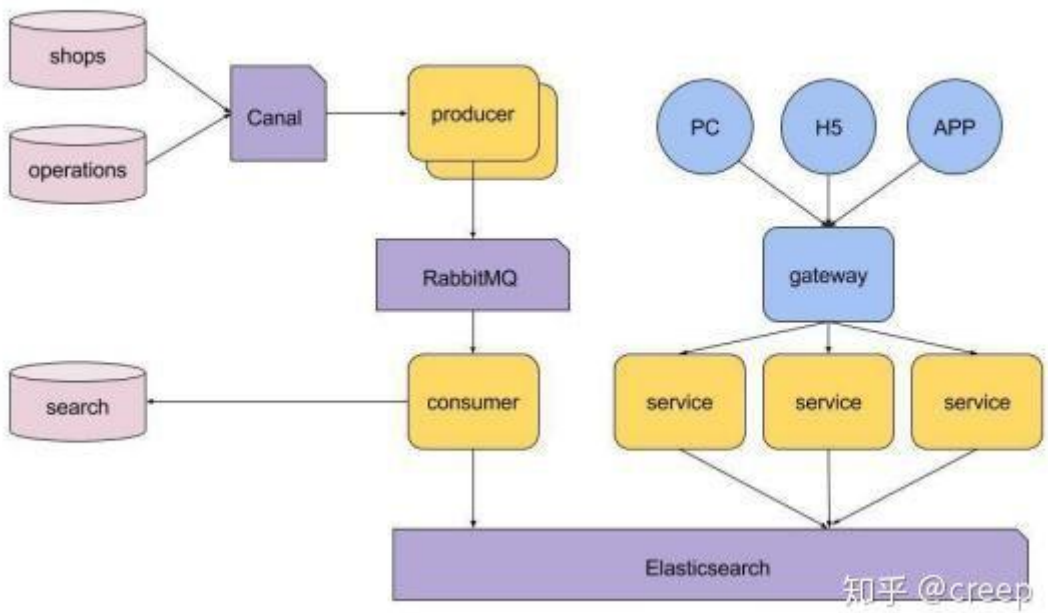
19、在并发情况下，Elasticsearch 如果保证读写一致?

1、可以通过版本号使用乐观并发控制，以确保新版本不会被旧版本覆盖，由应用层来处理具体的冲突；
2、另外对于写操作，一致性级别支持 quorum/one/all，默认为 quorum，即只有当大多数分片可用时才允许写操作。但即使大多数可用，也可能存在因为网络等原因导致写入副本失败，这样该副本被认为故障，分片将会在一个不同的节点上重建。
3、对于读操作，可以设置 replication 为 sync(默认)，这使得操作在主分片和副本分片都完成后才会返回；如果设置 replication 为 async 时，也可以通过设置搜索请求参数_preference 为 primary 来查询主分片，确保文档是最新版本。

20、如何监控 Elasticsearch 集群状态?

Marvel 让你可以很简单的通过 Kibana 监控 Elasticsearch。你可以实时查看你的集群健康状态和性能，也可以分析过去的集群、索引和节点指标。

21、介绍下你们电商搜索的整体技术架构。



22、介绍一下你们的个性化搜索方案?

SEE 基于 word2vec 和 Elasticsearch 实现个性化搜索

23、是否了解字典树?

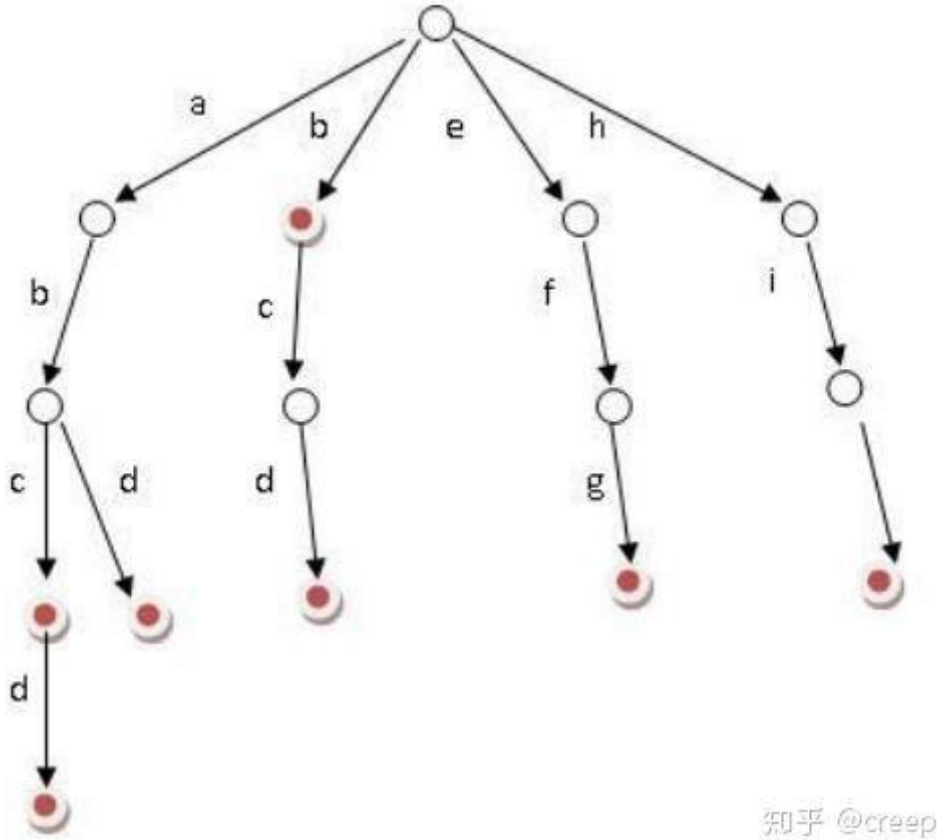
常用字典数据结构如下所示:

数据结构	优缺点
排序列表Array/List	使用二分法查找，不平衡
HashMap/TreeMap	性能高，内存消耗大，几乎是原始数据的三倍
Skip List	跳跃表，可快速查找词语，在lucene、redis、Hbase等均有实现。相对于TreeMap等结构，特别适合高并发场景（Skip List介绍）
Trie	适合英文词典，如果系统中存在大量字符串且这些字符串基本没有公共前缀，则相应的trie树将非常消耗内存（数据结构之trie树）
Double Array Trie	适合做中文词典，内存占用小，很多分词工具均采用此种算法（深入双数组Trie）
Ternary Search Tree	三叉树，每一个node有3个节点，兼具省空间和查询快的优点（Ternary Search Tree）
Finite State Transducers (FST)	一种有限状态转移机，Lucene 4有开源实现，并大量使用

Trie 的核心思想是空间换时间，利用字符串的公共前缀来降低查询时间的开销以
达到提高效率的目的。它有 3 个基本性质:

1、根节点不包含字符，除根节点外每一个节点都只包含一个字符。

2、从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串。
3、每个节点的所有子节点包含的字符都不相同。



1、可以看到，trie 树每一层的节点数是 26^i 级别的。所以为了节省空间，我们还可以用动态链表，或者用数组来模拟动态。而空间的花费，不会超过单词数×单词长度。
2、实现：对每个结点开一个字母集大小的数组，每个结点挂一个链表，使用左儿子右兄弟表示法记录这棵树；
3、对于中文的字典树，每个节点的子节点用一个哈希表存储，这样就不用浪费太大的空间，而且查询速度上可以保留哈希的复杂度 O(1)。

24、拼写纠错是如何实现的?

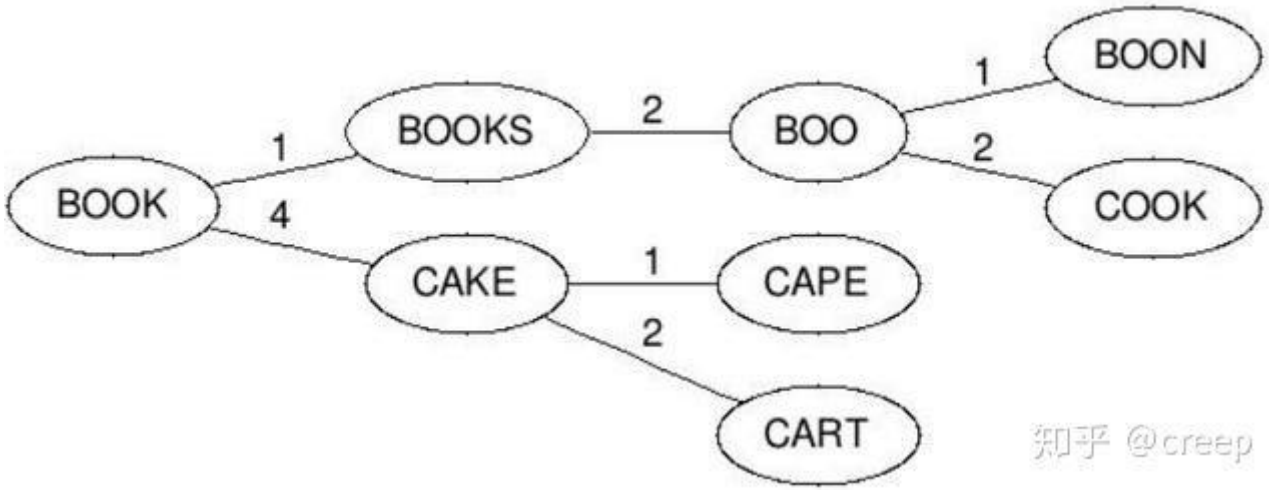
1、拼写纠错是基于编辑距离来实现；编辑距离是一种标准的方法，它用来表示经

过插入、删除和替换操作从一个字符串转换到另外一个字符串的最小操作步数；
2、编辑距离的计算过程：比如要计算 batyu 和 beauty 的编辑距离，先创建一个 7×8 的表（batyu 长度为 5，coffee 长度为 6，各加 2），接着，在如下位置填入黑色数字。其他格的计算过程是取以下三个值的最小值：
如果最上方的字符等于最左方的字符，则为左上方的数字。否则为左上方的数字 +1。（对于 3,3 来说为 0）
左方数字+1（对于 3,3 格来说为 2）
上方数字+1（对于 3,3 格来说为 2）
最终取右下角的值即为编辑距离的值 3。

		b	e	a	u	t	y
	0	1	2	3	4	5	6
b	1	0	1	2	3	4	5
a	2	1	1	1	2	3	4
t	3	2	2	2	2	2	3
y	4	3	3	3	3	3	2
u	5	4	4	4	3	4	3

对于拼写纠错，我们考虑构造一个度量空间（Metric Space），该空间内任何关系满足以下三条基本条件：
d(x,y) = 0 -- 假如 x 与 y 的距离为 0，则 x=y
d(x,y) = d(y,x) -- x 到 y 的距离等同于 y 到 x 的距离
d(x,y) + d(y,z) >= d(x,z) -- 三角不等式
1、根据三角不等式，则满足与 query 距离在 n 范围内的另一个字符转 B，其与 A 的距离最大为 d+n，最小为 d-n。
2、BK 树的构造就过程如下：每个节点有任意个子节点，每条边有个值表示编辑距离。所有子节点到父节点的边上标注 n 表示编辑距离恰好为 n。比如，我们有棵树父节点是”book”和两个子节点”cake”和”books”，”book”到”books”的边标号 1，”book”到”cake”的边上标号 4。从字典里构造好树后，无论何

时你想插入新单词时，计算该单词与根节点的编辑距离，并且查找数值为
d(neweord, root)的边。递归得与各子节点进行比较，直到没有子节点，你就可
以创建新的子节点并将新单词保存在那。比如，插入”boo”到刚才上述例子的树
中，我们先检查根节点，查找 d(“book”，“boo”) = 1 的边，然后检查标号为
1 的边的子节点，得到单词”books”。我们再计算距离 d(“books”，“boo”)=2，
则将新单词插在”books”之后，边标号为 2。
3、查询相似词如下：计算单词与根节点的编辑距离 d，然后递归查找每个子节点
标号为 d-n 到 d+n（包含）的边。假如被检查的节点与搜索单词的距离 d 小于 n，
则返回该节点并继续查询。比如输入 cape 且最大容忍距离为 1，则先计算和根的
编辑距离 d(“book”，“cape”)=4，然后接着找和根节点之间编辑距离为 3 到 5 的，这
个就找到了 cake 这个节点，计算 d(“cake”，“cape”)=1，满足条件
所以返回 **cake**，然后再找和 cake 节点编辑距离是 0 到 2 的，分别找到 cape 和
cart 节点，这样就得到 **cape** 这个满足条件的结果。



最后，由于这份文档的内容过多，为了节省读者朋友们的时间，我只整理出来了一部分供大家学习参考。
需要本【Java 进阶核心技术面试知识点】技术文档的小伙伴，可以点击下方文档即可获取 ：
书籍、Java 进阶路线、面试题，PDF