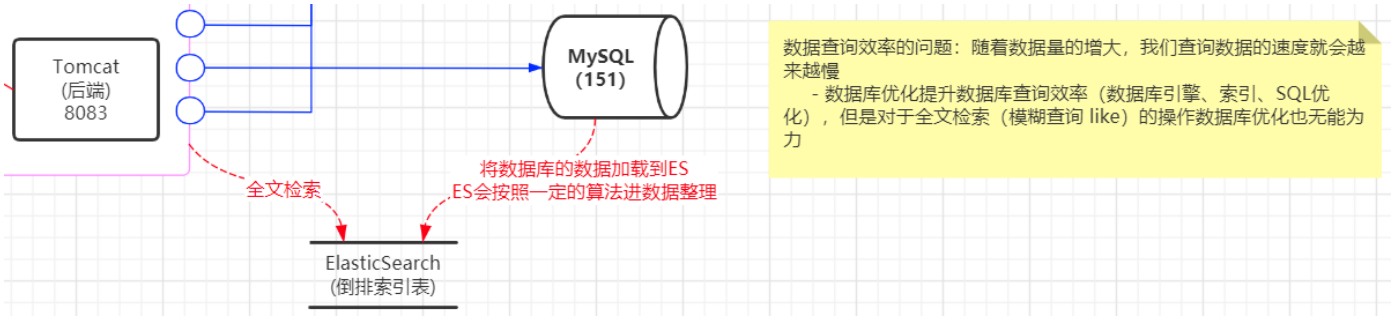
一、搜索引擎介绍

> 在互联网项目中，涉及到检索的业务需求很多，我们可以通过对数据库的模糊查询实现检索功能，但是针对大数据量的操作，基于数据库的检索就显得力不从心了（查询效率很低）。所需我们要寻求一种高效的数据检索解决方案。



所谓搜索引擎，就是根据用户需求与一定算法，运用特定策略从互联网检索出指定的信息反馈给用户的一门检索技术。搜索引擎依托于多种技术，如网络爬虫技术、检索排序技术、网页处理技术、大数据处理技术、自然语言处理技术等，为信息检索用户提供快速、高相关性的信息服务。搜索引擎技术的核心模块一般包括爬虫、索引、检索和排序等，同时可添加其他一系列辅助模块，以为用户创造更好的网络使用环境

搜索方式

搜索方式是搜索引擎的一个关键环节，大致可分为四种：全文搜索引擎、元搜索引擎、垂直搜索引擎和目录搜索引擎，它们各有特点并适用于不同的搜索环境

二、Lucene简介

2.1 Doug Cutting

1997年底，Cutting开始以每周两天的时间投入，在家里试着用Java把这个想法变成现实，不久之后，Lucene诞生了。作为第一个提供全文文本搜索的开源函数库，Lucene的伟大自不必多言。

2.2 Lucene介绍

> Lucene是Apache Jakarta家族中的一个开源项目，是一个开放源代码的全文检索引擎工具包，但它不是一个完整的 全文检索引擎，而是一个全文检索引擎的架构，提供了完整的查询引擎、索引引擎和部分文本分析引擎。
> Lucene提供了一个简单却强大的应用程式接口，能够做全文索引和搜寻。在Java开发环境里Lucene是一个成熟的免费开源工具，是目前最为流行的基于 Java 开源全文检索工

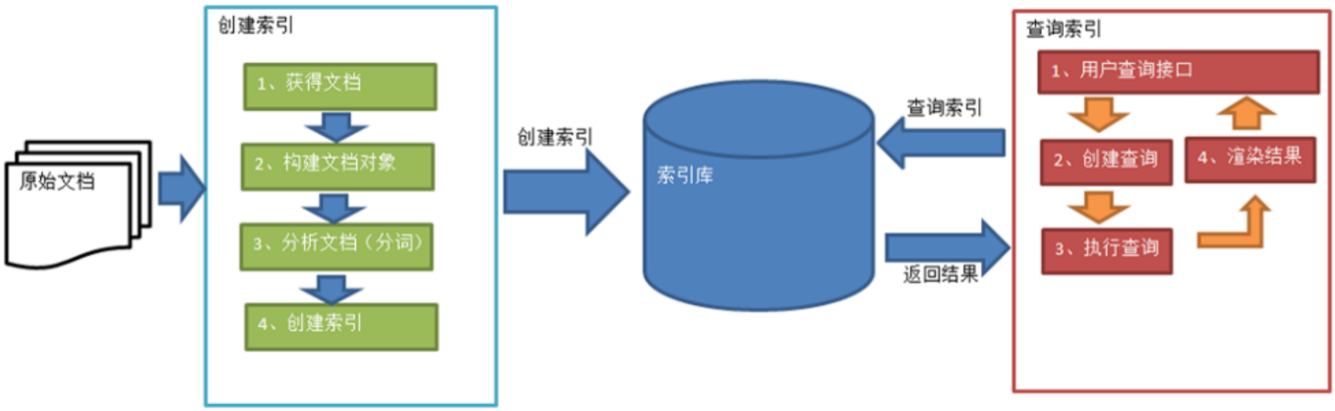
> 具包。

- 数据总体分为两种：
  - 结构化数据：指具有固定格式或有限长度的数据，如数据库、元数据等
  - 非结构化数据：指不定长或无固定格式的数据，如邮件、word文档等磁盘上的文件
- 对于结构化数据的全文搜索很简单，因为数据都是有固定格式的，例如搜索数据库中数据使用SQL语句即可
- 对于非结构化数据，有以下两种方法：
  - 顺序扫描法(Serial Scanning)
  - 全文检索(Full-text Search)

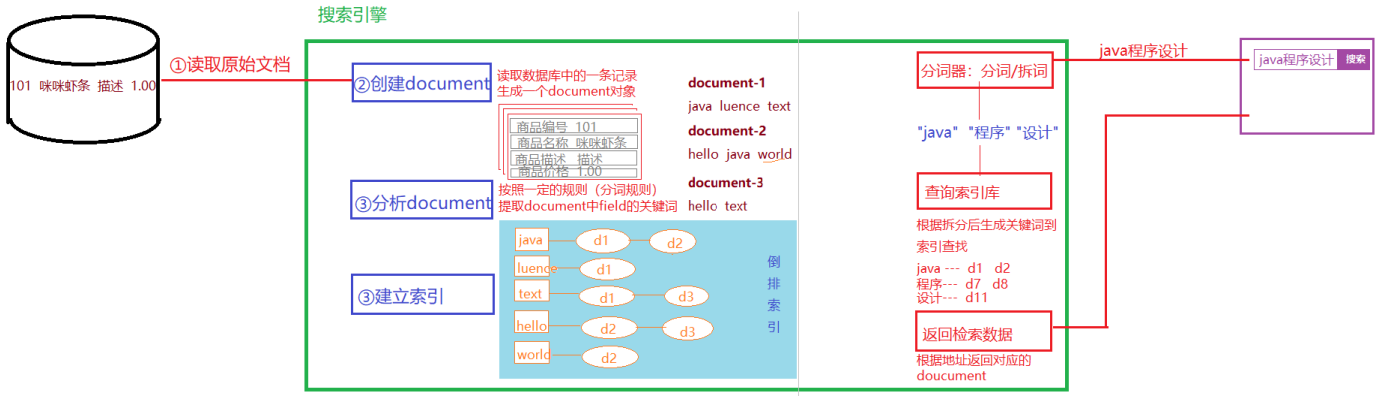
顺序扫描法：如果要找包含某一特定内容的文件，对于每一个文档，从头到尾扫描内容，如果此文档包含此字符串， 则此文档为我们要找的文件，接着看下一个文件，直到扫描完所有的文件，因此速度很慢。

全文检索：将非结构化数据中的一部分信息提取出来，重新组织，使其变得具有一定结构，然后对此有一定结构的数 据进行搜索，从而达到搜索相对较快的目的。这部分从非结构化数据中提取出的然后重新组织的信息，我们称之索引。

2.3 Lucene全文检索流程



- 创建索引过程，对要搜索的原始内容进行索引构建一个索引库。索引过程包括：确定原始内容即要搜索的内容→采集文档→创建文档→分析文档→索引文档。
- 搜索索引过程，从索引库中搜索内容。搜索过程包括：用户通过搜索界面→创建查询→执行搜索，从索引库搜索→ 渲染搜索结果。



2.3.1 创建索引

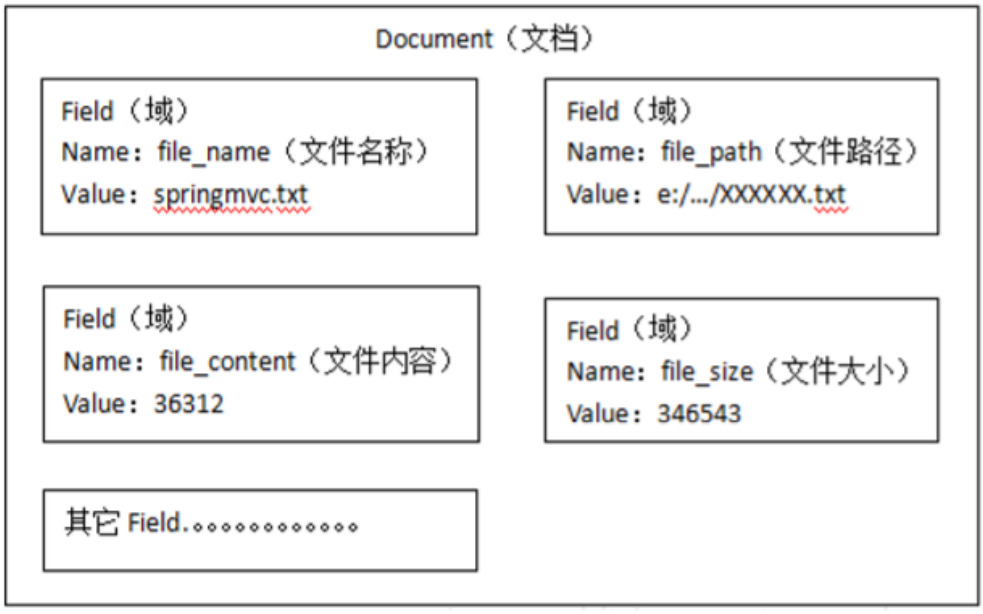
对文档索引的过程，将用户要搜索的文档内容进行索引，索引存储在索引库（index）中。这里我们要搜索的文档是 磁盘上的文本文件，根据案例描述：凡是文件名或文件内容包括关键字的文件都要找出来，这里要对文件名和文件内 容创建索引。

- **获得原始文档**

  原始文档是指要索引和搜索的内容。原始内容包括互联网上的网页、数据库中的数据、磁盘上的文件等。

- **创建文档对象**

  获取原始内容的目的是为了索引，在索引前需要将原始内容创建成文档（Document），文档中包括一个一个的域 （Field），域中存储内容。 这里我们可以将磁盘上的一个文件当成一个document，Document中包括一些 Field（file_name文件名称、file_path文件路径、file_size文件大小、file_content文件内容），如下图：



注意：

- 每个Document可以有多个Field，不同的Document可以有不同的Field

- 每个文档都有一个唯一的编号，就是文档id。

- **分析文档**

  将原始内容创建为包含域（Field）的文档（document），需要再对域中的内容进行分析，分析的过程是经过对原始文档提取单词、将字母转为小写、去除标点符号、去除停用词等过程生成最终的语汇单元，可以将语汇单元理解为一 个一个的单词。 比如下边的文档经过分析如下：

  原文档内容：

  ```
  Lucene is a Java full-text search engine. Lucene is not a complete application, but rather a code library and API that can easily be used to add search capabilities to applications.
  ```

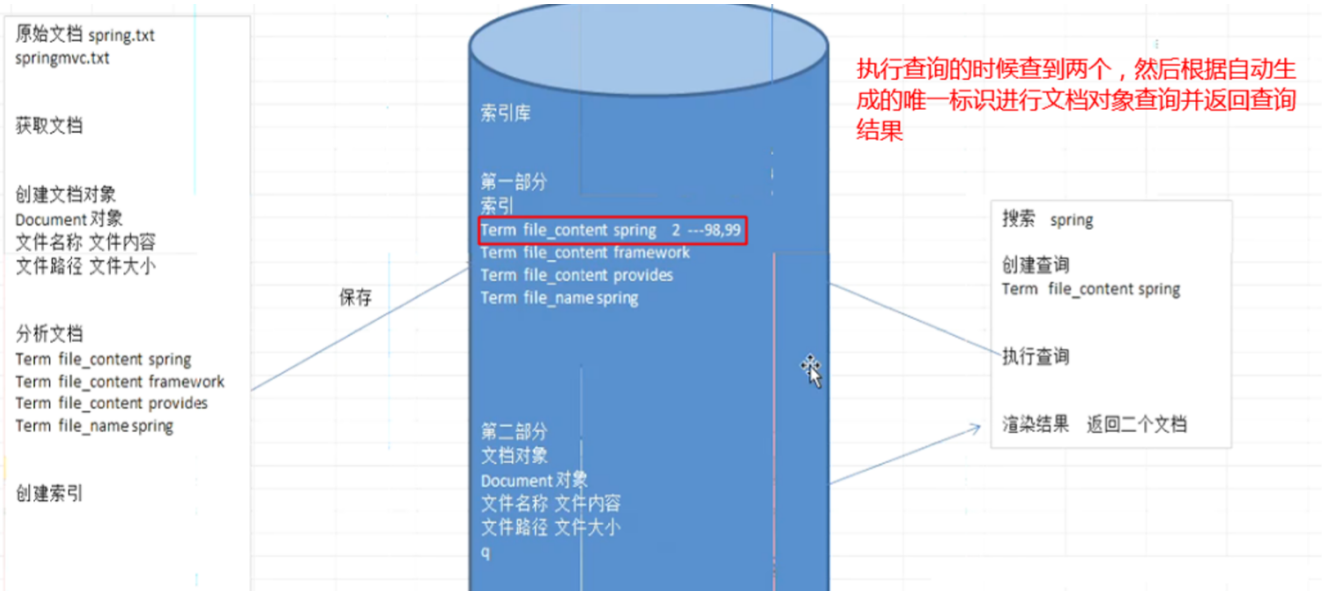
  分析后得到的语汇单元：

  ```
  lucene、java、full、search、engine...
  ```

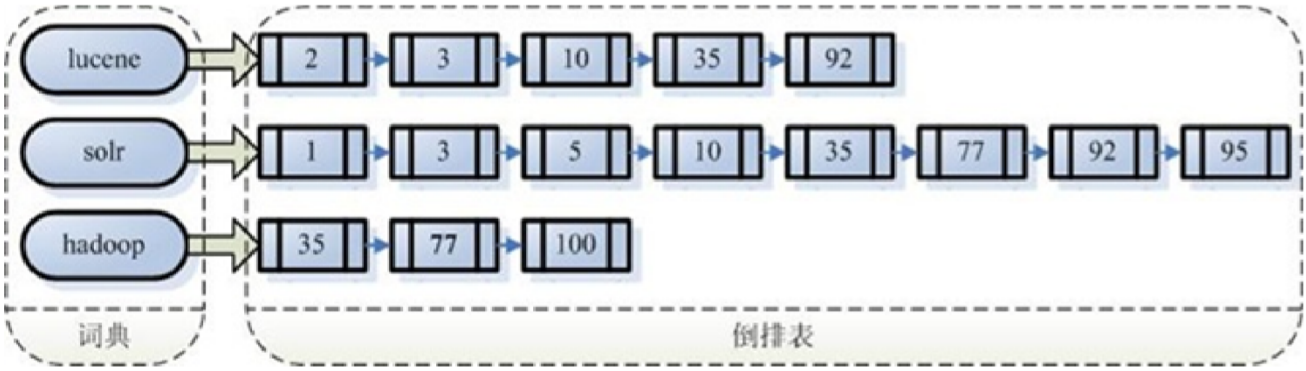
  每个单词叫做一个Term，不同的域中拆分出来的相同的单词是不同的term。term中包含两部分一部分是文档的域名，另一部分是单词的内容。 例如：文件名中包含apache和文件内容中包含的apache是不同的term。

- **创建索引—倒排索引**

  对所有文档分析得出的语汇单元进行索引，索引的目的是为了搜索，最终要实现只搜索被索引的语汇单元从而找到 Document（文档）



注意：创建索引是对语汇单元索引，通过词语找文档，这种索引的结构叫倒排索引结构。 传统方法是根据文件找到 该文件的内容，在文件内容中匹配搜索关键字，这种方法是顺序扫描方法，数据量大、搜索慢。 倒排索引结构是根 据内容（词语）找文档，如下图：



倒排索引结构也叫反向索引结构，包括索引和文档两部分，索引即词汇表，它的规模较小，而文档集合较大。

2.3.2 查询索引

查询索引也是搜索的过程。搜索就是用户输入关键字，从索引（index）中进行搜索的过程。根据关键字搜索索引， 根据索引找到对应的文档，从而找到要搜索的内容（这里指磁盘上的文件）。

- **用户查询接口**

  全文检索系统提供用户搜索的界面供用户提交搜索的关键字，搜索完成展示搜索结果。Lucene不提供制作用户搜索界面的功能，需要根据自己的需求开发搜索界面。

- **创建查询**

  用户输入查询关键字执行搜索之前需要先构建一个查询对象，查询对象中可以指定查询要搜索的Field文档域、查询关键字等，查询对象会生成具体的查询语法，例如： 语法“fileName:lucene”表示要搜索Field域的内容为“lucene”的文档

- **执行查询**

  搜索索引过程： 根据查询语法在倒排索引词典表中分别找出对应搜索词的索引，从而找到索引所链接的文档链表。 比如搜索语法为“fileName:lucene”表示搜索出fileName域中包含Lucene的文档。 搜索过程就是在索引上查找域为 fileName，并且关键字为Lucene的term，并根据term找到文档id列表。

- **渲染查询结果**

以一个友好的界面将查询结果展示给用户，用户根据搜索结果找自己想要的信息，为了帮助用户很快找到自己的结 果，提供了很多展示的效果，比如搜索结果中将关键字高亮显示，百度提供的快照等。

2.4 分词器

2.4.1 分词器的作用

a. 在创建索引的时候需要用到分词器，在使用字符串搜索的时候也会用到分词器，并且这两个地方要使用同一个 分词器，否则可能会搜索不出来结果。

b. 分词器(Analyzer)的作用是把一段文本中的词按规则取出所包含的所有词，对应的是Analyzer类，这是一个抽 象类(public abstract class org.apache.lucene.analysis.Analyzer)，切分词的具体规则是由子类实现的，所以对于不 同的语言规则，要有不同的分词器。

2.4.2 英文分词器的原理

a.英文的处理流程为：输入文本，词汇切分，词汇过滤(去除停用词)，词干提取(形态还原)、大写转小写，结果输 出。

b. 何为形态还原，意思是：去除单词词尾的形态变化，将其还原为词的原形，这样做可以搜索出更多有意义的结 果，比如在搜索student的时候，同事也可以搜索出students的结果。

c. 任何一个分词法对英文的支持都是还可以的。

2.4.3 中文分词器的原理

中文分词比较复杂，并没有英文分词那么简单，这主要是因为中文的词与词之间并不是像英文那样用空格来隔 开，因为不是一个字就是一个词，而且一个词在另外一个地方就可能不是一个词，如："我们是中国人"，"是中"就不是一 个词，对于中文分词，通常有三种方式：单字分词、二分法分词、词典分词。

- 单字分词：就是按照中文一个字一个字的进行分词，比如:"我们是中国人"，分词的效果就是"我"，"们"，"是"，"中"，"国"，"人"，StandardAnalyzer分词法就是单字分词。
- 二分法分词：按照两个字进行切分，比如："我们是中国人"，分词的效果就是："我们"，"们是"，"是 中"，"中国"，"国人"，CJKAnalyzer分词法就是二分法分词
- 词库分词：按照某种算法构造词，然后去匹配已建好的词库集合，如果匹配到就切分出来成为词语，通常词 库分词被认为是最好的中文分词算法，如："我们是中国人"，分词的效果就是:"我们"，"中国人"，极易分词 MMAnalyzer、庖丁分词、IkAnalyzer等分词法就是属于词库分词。

2.4.4 停用词的规

有些词在文本中出现的频率非常高，但是对文本所携带的信息基本不产生影响，例如英文的"a、an、the、of"或中文 的"的、了、着、是"，以及各种标点符号等，这样的词称为停用词，文本经过分词处理后，停用词通常会被过滤掉， 不会被进行索引，在检索的时候，用户的查询中如果含有停用词，检索系统也会将其过滤掉，这是因为用户输入查询字符串也要进行分词处理，排除停用词可以提升建立索引的速度，减小索引库文件的大小。

2.4.5 常用分词器

- WhitespaceAnalyzer

  仅仅是去掉了空格，没有其他任何操作，不支持中文。

- SimpleAnalyzer

  将除了字母以外的符号全部去除，并且将所有字符变为小写，需要注意的是这个分词器同样把数据也去除了，同样不支持中文。

- StopAnalyzer

  这个和SimpleAnalyzer类似，不过比他增加了一个的是，在其基础上还去除了所谓的stop words，比如the, a, this这些。这个也是不支持中文的。

- StandardAnalyzer

  英文方面的处理和StopAnalyzer一样的，对中文支持，使用的是单字切割。

- CJKAnalyzer

  这个支持中日韩，前三个字母也就是这三个国家的缩写。这个对于中文基本上不怎么用吧，对中文的支持很烂，它是 用每两个字作为分割，分割方式个人感觉比较奇葩，我会在下面比较举例。

- SmartChineseAnalyzer

  中文的分词,比较标准的中文分词，对一些搜索处理的并不是很好。

- IKAnalyzer
  中国人自己开发，对于中文分词比较精准

2.4.6 IK 分词器

Elasticsearch中文分词我们采用Ik分词，ik有两种分词模式：ik_max_word和ik_smart模式;

ik_max_word 和 ik_smart 什么区别?

- ik_max_word: 会将文本做最细粒度的拆分，比如会将“中华人民共和国国歌”拆分为“中华

人民共和国,中华人民, 中华,华人,人民共和国,人民,人,民,共和国,共和,和国,国歌”，会穷尽各种可能的组合；

- ik_smart: 会做最粗粒度的拆分，比如会将“中华人民共和国国歌”拆分为“中华人民,共和国,国歌”。 索引时，为了提供索引的覆盖范围，通常会采用ik_max_word分析器，会以最细粒度分词索引，搜索时为了提高搜索准确度，会采用ik_smart分析器，会以粗粒度分词

我们可以使用网上的一些工具查看分词的效果，比如https://www.sojson.com/analyzer

2.5 lucene全文检索与数据库查询的比较

2.5.1 性能上

数据库：比如我要查找某个商品，根据商品名，比如select * from product where doctname like %keywords%,这 样查询的话对于数据量少是可以的，可是一旦你的数据量巨大几万几十万的时候，你的性能将会极大的减弱。

lucene: 全文检索，建立一个索引库，一次建立多次使用。在索引库里面会把所有的商品名根据分词器建立索引，就 好比新华字典，索引对应document，比如输入衬衫，那么就会根据索引迅速的翻到衬衫对应的商品名，时间迅速， 性能很好。

2.4.2 相关度排序

数据库：数据库要实现该功能也是可以的，可是需要改变表的结构，添加一个字段，然后该字段用于排名，最后查询 的时候order by 一下

lucene: 查询出来的document都有一个算法（得分），根据算法可以计算得分，得分越高的排名越靠前，比如百度 搜索一个东西，一般排名靠前的得分越高，得分通过算法，可以人工控制，比如百度推广，企业给的钱多得分自然 高，因此排名靠前

2.4.3 准确性

数据库：select * from product where doctname like %ant%,搜索出来的可以是plant,aplant,planting等等，准确性不高

lucene：通过索引查询的，就好像你查字典一样，准确性比数据库的模糊查询高许多

三、ElasticSearch简介

3.1 ElasticSearch vs Lucene的关系

ElasticSearch vs Lucene的关系，简单一句话就是，成品与半成品的关系。

- （1）Lucene专注于搜索底层的建设，而ElasticSearch专注于企业应用。
- （2）Luncene是单节点的API，ElasticSearch是分布式的—为集群而生。
- （3）Luncene需要二次开发，才能使用。不能像百度或谷歌一样，它只是提供一个接口需要被实现才能使用，ElasticSearch直接拿来用。

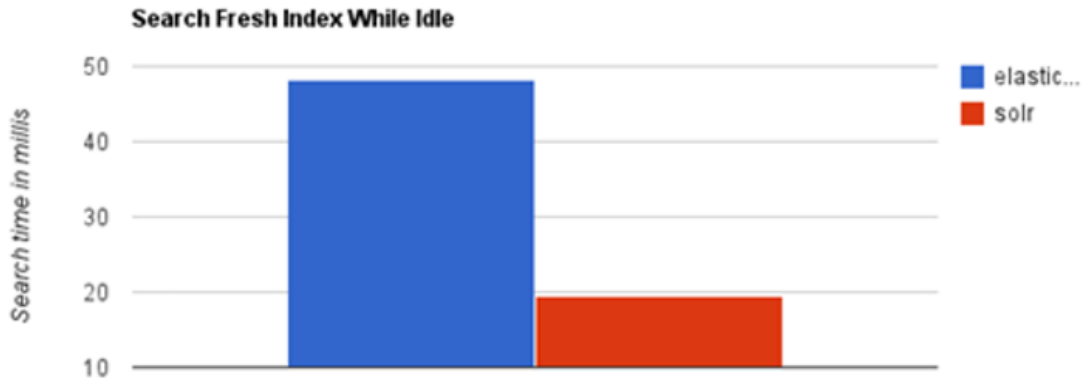
3.2 ElasticSearch与Solr对比

Solr与elasticsearch是当前两大最流行的搜索应用服务器，他们的底层都是基于lucene。

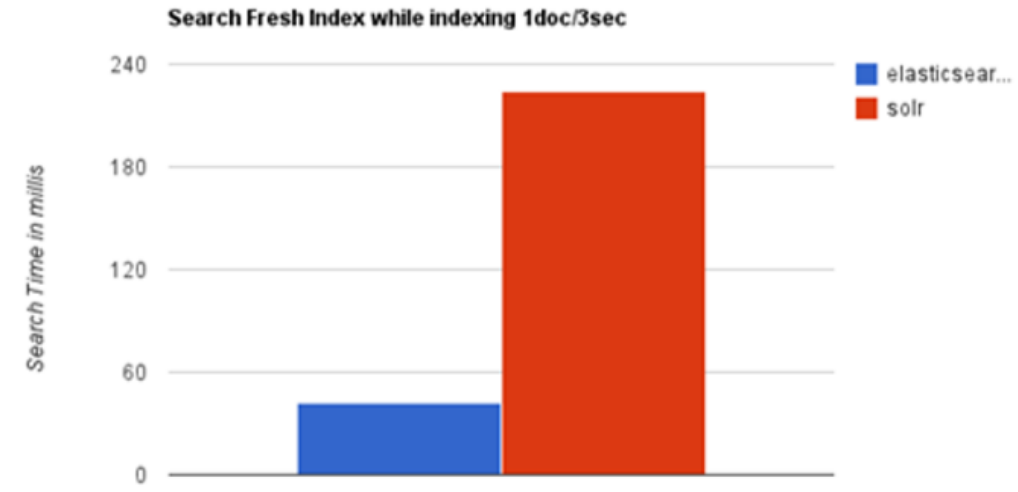
- Elasticsearch是分布式的，不需要其他组件，Solr 利用 Zookeeper 进行分布式管理，而Elasticsearch 自身带 有分布式协调管理功能
- Elasticsearch设计用于云计算中，处理多租户不需要特殊配置，而Solr则需要更多的高级设置。
- 当单纯的对已有数据进行搜索时，Solr更快，实时建立索引时, Solr会产生io阻塞，查询性能较差, Elasticsearch 具有明显的优势，随着数据量的增加，Solr的搜索效率会变得更低，而Elasticsearch却没有明显的变化

Elasticsearch与Solr的性能测试比较:

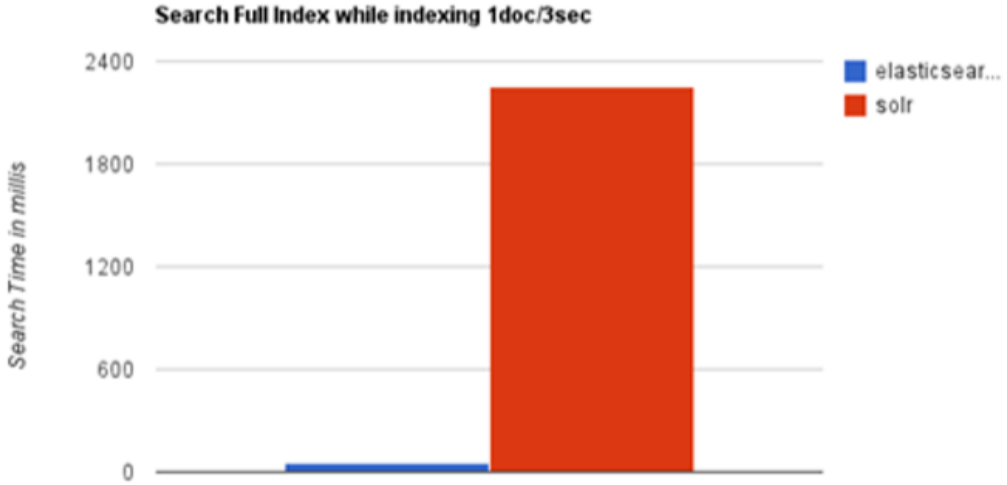
当单纯的对已有数据进行搜索时，Solr更快



当实时建立索引时, Solr会产生io阻塞，查询性能较差, Elasticsearch具有明显的优势



随着数据量的增加，Solr的搜索效率会变得更低，而Elasticsearch却没有明显的变化



3.3 Elasticsearch特性

3.3.1 安装管理方便

Elasticsearch没有其他依赖，下载后安装非常方便；只用修改几个参数就可以搭建起来一个集群。

3.3.2 大规模分布式

Elasticsearch允许你开始小规模使用，但是随着你使用数据的增长，它可以建立在横向扩展的开箱即用。当你需要更多的容量，只需添加更多的节点，并让集群重组，只需要增加额外的硬件，让集群自动利用额外的硬件。

`可以在数以百计的服务器上处理PB级别的数据`。

节点对外表现对等（每个节点都可以用来做入口）；加入节点自动均衡，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致力于隐藏分布式系统的复杂性。以下这些操作都是在底层自动完成的：

- 将你的文档分区到不同的容器或者分片(shards)中，它们可以存在于一个或多个节点中；
- 将分片均匀的分配到各个节点，对索引和搜索做负载均衡；
- 冗余每一个分片，防止硬件故障造成的数据丢失；
- 将集群中任意一个节点上的请求路由到相应数据所在的节点；
- 无论是增加节点，还是移除节点，分片都可以做到无缝的扩展和迁移

3.3.3 多租户支持

ES处理多租户不需要特殊配置，可根据不同的用途分索引；可以同时操作多个索引。

ES 的多租户简单的说就是通过多索引机制同时提供给多种业务使用，每种业务使用一个索引。我们可以把索引理解 为关系型数据库里的库，那多索引可以理解为一个数据库系统建立多个库给不同的业务使用。

在实际使用时，我们可以通过每个租户一个索引的方式将他们的数据进行隔离，并且每个索引是可以单独配置参数的 （可对特定租户进行调优），这在典型的多租户场景下非常有用：例如我们的一个多租户应用需要提供搜索支持，这 时可以通过 ES 根据租户建立索引，这样每个租户就可以在自己的索引下搜索相关内容了

3.3.4 高可用性

Elasticsearch集群是有弹性的 - 他们会自动检测到新的或失败的节点，以及重组和重新平衡数据，以确保数据安全。

3.3.5 操作持久化

Elasticsearch把数据安全第一。文档改变被记录在群集上的多个节点上的事务日志(transaction logs)中记录，以减少 任何数据丢失的机会。

3.3.6 友好的RESTful API

Elasticsearch是API驱动。几乎任何动作都可以用一个简单的RESTful API使用JSON基于HTTP请求。ElasticSearch 提 供多种语言的客户端 API。

3.4 典型使用案例

- 维基百科使用Elasticsearch来进行全文搜做并高亮显示关键词，以及提供search-as-you-type、did-you-mean 等搜索建议功能。
- 英国卫报使用Elasticsearch来处理访客日志，以便能将公众对不同文章的反应实时地反

馈给各位编辑。
- StackOverflow将全文搜索与地理位置和相关信息进行结合，以提供more-like-this相关问题的展现。
- GitHub使用Elasticsearch来检索超过1300亿行代码。
- Goldman Sachs使用它来处理5TB数据的索引，还有很多投行使用它来分析股票市场的变动。

四、Elasticsearch逻辑结构

> 集群-->index(索引)-->types(类型)-->document(文档)-->field(字段)

4.1 索引（index）

索引是ElasticSearch存放数据的地方，可以理解为关系型数据库中的一个数据库。

事实上，我们的数据被存储和索引 在分片(shards)中，索引只是一个把一个或多个分片分组在一起的逻辑空间。然而，这只是一些内部细节——我们的 程序完全不用关心分片。对于我们的程序而言，文档存储在索引(index)中。剩下的细节由Elasticsearch关心既可。

`索引的名字必须是全部小写，不能以下划线开头，不能包含逗号`

4.2 类型（type）

类型用于区分同一个索引下不同的数据类型,相当于关系型数据库中的表。在Elasticsearch中，我们使用相同类型 (type)的文档表示相同的“事物”，因为他们的数据结构也是相同的。每个类型(type)都有自己的映射(mapping)或者结 构定义，就像传统数据库表中的列一样。所有类型下的文档被存储在同一个索引下，但是类型的映射(mapping)会告 诉Elasticsearch不同的文档如何被索引。

`es 6.0 开始不推荐一个index下多个type的模式，并且会在 7.0 中完全移除。在7.0 的index下是无法创建多个type`

4.3 文档（documents）

文档是ElasticSearch中存储的实体，类比关系型数据库，每个文档相当于数据库表中的一行数据。 在Elasticsearch 中，文档(document)这个术语有着特殊含义。它特指最顶层结构或者根对象(root object)序列化成的JSON数据（以 唯一ID标识并存储于Elasticsearch中）。

4.4 字段（fields）

文档由字段组成，相当于关系数据库中列的属性，不同的是ES的不同文档可以具有不同的字段集合。

4.5 节点与集群

一个集群是由一个或多个节点组成的集合，集群上的节点将会存储数据，并提供跨节点的索引和搜索功能。

集群通过一个唯一的名称作为标识，节点通过设置集群名称就可以加入相应的集群，当然这需要节点所在的网络能够发现集群。所以要注意在同一个网络中，不同环境、服务的集群的名称不能重复。

一个节点就是一个 Elasticsearch 服务 （实例），可以实现存储数据，索引并且搜索的功能。和集群一样，每个节点都有一个唯一的名称作为身份标识，如果没有设置名称，默认使用 UUID 作为名称。如果想更好的管理集群，最好给每个节点都定义上有意义的名称，在集群中区分出各个节点。节点通过设置集群名称，在同一网络中发现具有相同集群名称的节点，组成集群。默认的集群名称为 elasticsearch 。

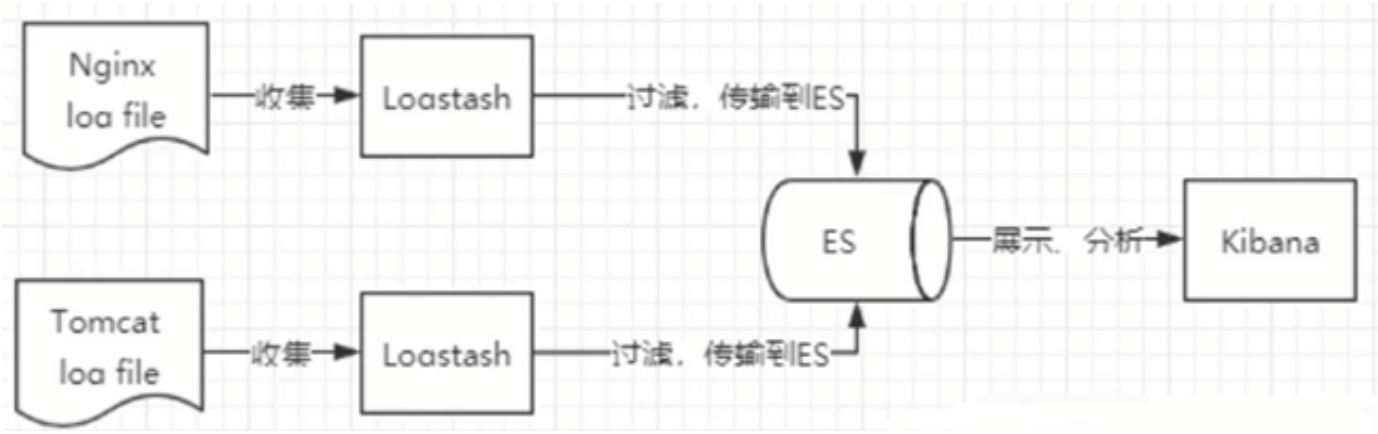
如果在同一网络中只有一个节点，则这个节点成为一个单节点集群，换句话说就是每个节点都是功能齐全的服务。

五、Elasticsearch安装

5.1 Elastic 和 Elasticsearch

Elastic官网：https://www.elastic.co/cn/

Elastic有一条完整的产品线及解决方案：Elasticsearch、Logstash、Kibana等，这三个就是大家常说的ELK技术栈。



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

5.2 Linux下安装ES

> 出于安全考虑，elasticsearch默认不允许以root账号运行

- 创建用户设置密码

```
[root@localhost ~]# useradd es
[root@localhost ~]# passwd es
Changing password for user es.
New password:     【QFedu123】
Retype new password:
[root@localhost ~]# chmod 777 /usr/local    【授予es用户/usr/local目录可读可写可执行权限】
[root@localhost ~]# su - es
[es@localhost ~]$
```

- 检查JDK版本(需要JDK1.8+)

```
[es@localhost ~]# java -version
openjdk version "1.8.0_222-ea"
OpenJDK Runtime Environment (build 1.8.0_222-ea-b03)
OpenJDK 64-Bit Server VM (build 25.222-b03, mixed mode)
```

- 将ES的压缩包上传至`/usr/local`目录并解压

```
[es@localhost local]$ tar -zxvf elasticsearch-7.6.1-linux-x86_64.tar.gz
```

- 查看配置文件

```
[es@localhost local]# cd elasticsearch-7.6.1/config/
[es@localhost config]# ls
elasticsearch.yml jvm.options log4j2.properties role_mapping.yml roles.yml users users_roles
```

- 修改 jvm.options

  Elasticsearch基于Lucene的，而Lucene底层是java实现，因此我们需要配置jvm参数

```
[es@localhost config]# vim jvm.options

# 默认配置如下
# Xms represents the initial size of total heap space
# Xmx represents the maximum size of total heap space
-Xms1g
-Xmx1g
```

- **修改 elasticsearch.yml**
  - 修改集群节点信息

```
# ---------------------------------- Cluster -----------------------
------------17
cluster.name: my-application

# ------------------------------------ Node ------------------------
------------23
node.name: node-1

# ---------------------------------- Discovery ---------------------
------------72
cluster.initial_master_nodes: ["node-1"]
```

  - 修改数据文件和日志文件存储目录路径（如果目录不存在则需创建）

```
[root@localhost config]# vim elasticsearch.yml

# ---------------------------- Paths ------------------------------
path.data: /usr/local/elasticsearch-7.6.1/data
path.logs: /usr/local/elasticsearch-7.6.1/logs
```

  - 修改绑定的ip，默认只允许本机访问，修改为0.0.0.0后则可以远程访问

```
# ---------------------------- Network -----------------------------
-
# 默认只允许本机访问，修改为0.0.0.0后则可以远程访问
network.host: 0.0.0.0
```

  - 配置信息说明

目前我们是做的单机安装，如果要做集群，只需要在这个配置文件中添加其它节点信息即可。

> elasticsearch.yml的其它可配置信息：

属性名	说明
cluster.name	配置elasticsearch的集群名称，默认是elasticsearch。建议修改成一个有意义的名称。
node.name	节点名，es会默认随机指定一个名字，建议指定一个有意义的名称，方便管理
path.conf	设置配置文件的存储路径，tar或zip包安装默认在es根目录下的config文件夹，rpm安装默认在/etc/ elasticsearch
path.data	设置索引数据的存储路径，默认是es根目录下的data文件夹，可以设置多个存储路径，用逗号隔开
path.logs	设置日志文件的存储路径，默认是es根目录下的logs文件夹
path.plugins	设置插件的存放路径，默认是es根目录下的plugins文件夹
bootstrap.memory_lock	设置为true可以锁住ES使用的内存，避免内存进行swap
network.host	设置bind_host和publish_host，设置为0.0.0.0允许外网访问
http.port	设置对外服务的http端口，默认为9200。
transport.tcp.port	集群结点之间通信端口
discovery.zen.ping.timeout	设置ES自动发现节点连接超时的时间，默认为3秒，如果网络延迟高可设置大些
discovery.zen.minimum_master_nodes	主结点数量的最少值 ,此值的公式为：(master_eligible_nodes / 2) + 1 ，比如：有3个符合要求的主结点，那么这里要设置为2

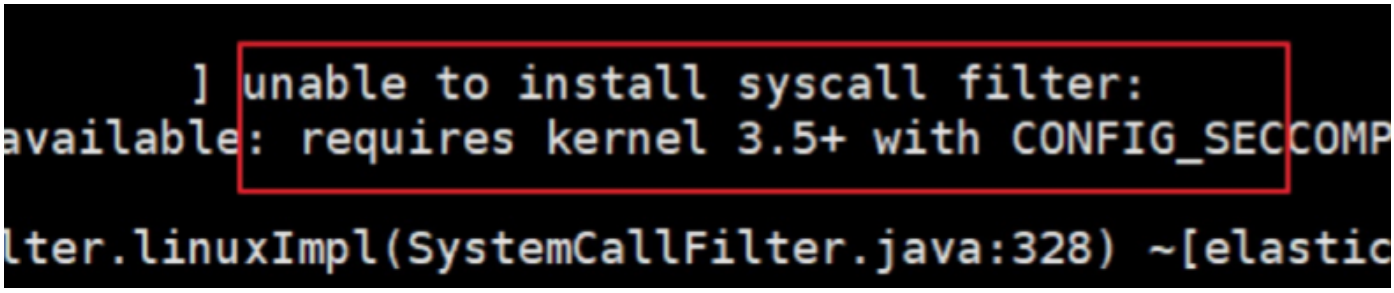
- 进入elasticsearch/bin目录运行

```
[es@localhost elasticsearch-7.6.1]# cd /usr/local/elasticsearch-7.6.1/bin
[es@localhost elasticsearch-7.6.1]# ./elasticsearch
```

```
* soft nofile 666666666
  * hard nofile 131072
  * soft nproc 4096
  * hard nproc 4096
```

5.3 启动错误问题总结

错误1：内核过低



我们使用的是centos6，其linux内核版本为2.6。而Elasticsearch的插件要求至少3.5以上版本。不过没关系，我们禁 用这个插件即可。
修改elasticsearch.yml文件，在最下面添加如下配置：

```
bootstrap.system_call_filter: false
```

然后重启

错误2：文件权限不足

我们用的是es用户，而不是root，所以文件权限不足。

首先用root用户登录,然后修改配置文件:

```
vim /etc/security/limits.conf
```

添加下面的内容：

```
* soft nofile 65536
* hard nofile 131072
* soft nproc 4096
* hard nproc 4096
```

错误3：线程数不够

```
 [1]: max number of threads [1024] for user [es] is too low, increase to at least [4096]
```

这是线程数不够
继续修改配置

```
vim /etc/security/limits.d/20-nproc.conf
```

修改下面的内容：

```
soft nproc 1024
```

改为

```
soft nproc 4096
```

错误4：进程虚拟内存

```
[3]: max virtual memory areas vm.max_map_count [65530] likely too low, increase to at least [262144]
```

vm.max_map_count：限制一个进程可以拥有的VMA(虚拟内存区域)的数量

继续修改配置文件， vim /etc/sysctl.conf 添加下面内容：

```
vm.max_map_count=655360
```

修改完成之后在终端执行

```
##然后执行命令
sysctl -p
```

错误5：未设置节点

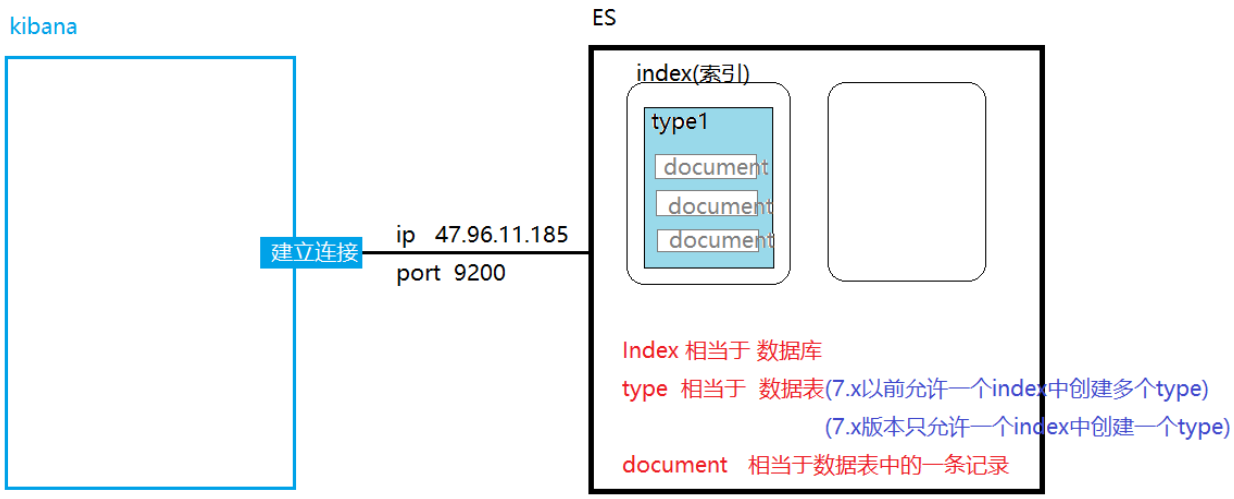
```
the default discovery settings are unsuitable for production use; at least one of [discovery.seed_ho...]
```

修改elasticsearch.yml

```
cluster.name: my-application
node.name: node-1
cluster.initial_master_nodes: ["node-1"]
```

六、安装Kibana

Kibana是一个基于Node.js的Elasticsearch索引库数据统计工具，可以利用Elasticsearch的聚合功能，生成各种图 表，如柱形图，线状图，饼图等。
而且还提供了操作Elasticsearch索引数据的控制台，并且提供了一定的API提示，非常有利于我们学习Elasticsearch 的语法。



6.1 安装

- kibana版本与elasticsearch保持一致，也是7.6.1解压到特定目录即可

```
tar -zxvf kibana-7.6.1-linux-x86_64.tar.gz
```

6.2 配置

进入安装目录下的config目录，修改kibana.yml文件：

```
server.port: 5601
server.host: "0.0.0.0"
```

6.3 运行

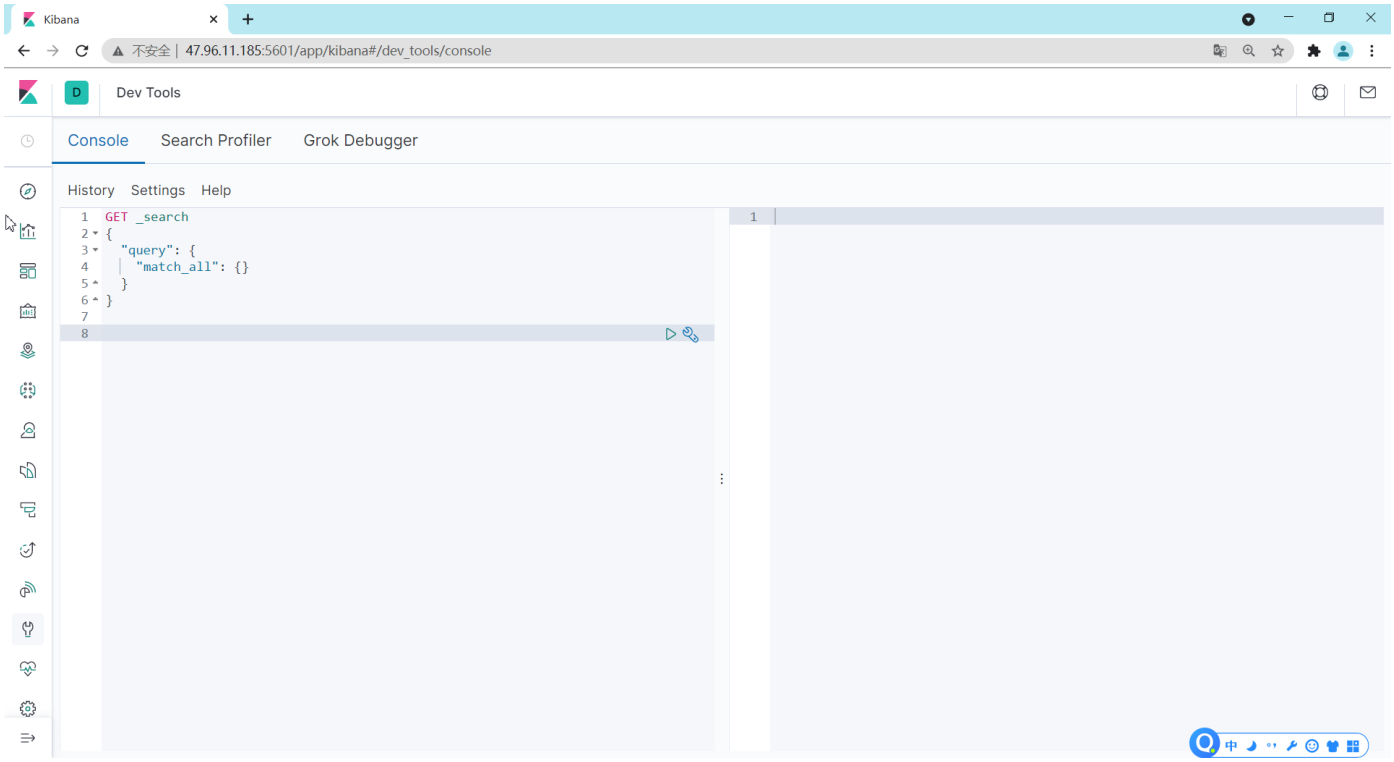
进入安装目录下的bin目录启动：

```
./kibana
```

发现kibana的监听端口是5601

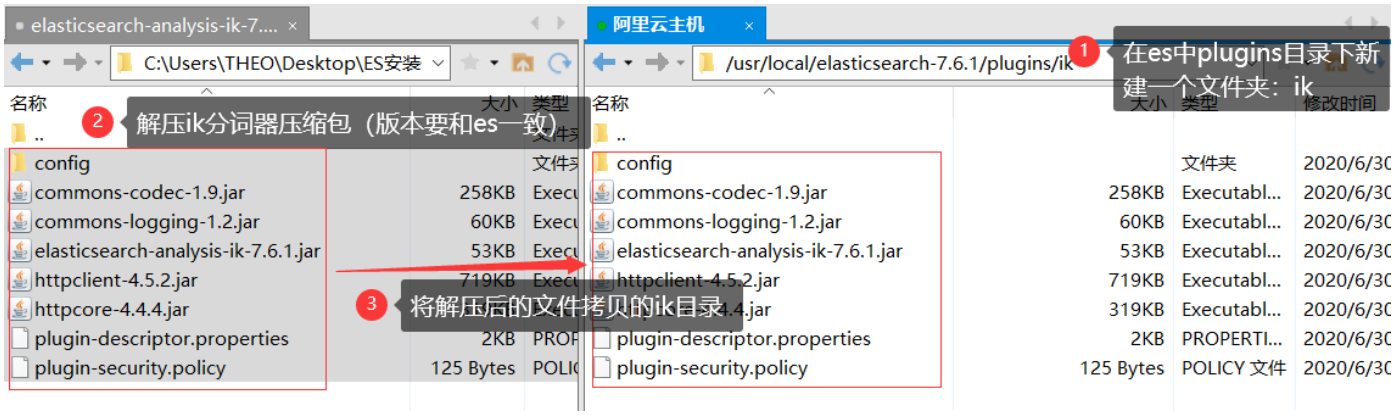
我们访问：http://47.96.11.185:5601

6.4 控制台



七、安装IK分词器

7.1 安装ik分词器



分词器配置完成以后，重启es

7.2 测试分词器

```
GET _analyze
{
  "analyzer": "ik_smart",
  "text": "中华人民共和国国旗"
}

GET _analyze
{
  "analyzer": "ik_max_word",
  "text": "中华人民共和国国旗"
}
```

```
{
  "tokens" : [
    {
      "token" : "中华人民共和国",
      "start_offset" : 0,
      "end_offset" : 7,
      "type" : "CN_WORD",
      "position" : 0
    },
    {
      "token" : "国旗",
      "start_offset" : 7,
      "end_offset" : 9,
      "type" : "CN_WORD",
      "position" : 1
    }
  ]
}
```

7.3 配置自定义词库

- 在elasticsearch-analysis-ik-7.6.1/plugins/ik/config目录中定义词典文件（.dic）
- 在词典文件中定义自定义词汇
- elasticsearch-analysis-ik-7.6.1/plugins/ik/config/IKAnalyzer.cfg.xml加载自定义词典文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM
"http://java.sun.com/dtd/properties.dtd">
<properties>
  <comment>IK Analyzer 扩展配置</comment>
  <!--用户可以在这里配置自己的扩展字典 -->
  <entry key="ext_dict">mywords.dic</entry>
   <!--用户可以在这里配置自己的扩展停止词字典-->
  <entry key="ext_stopwords"></entry>
  <!--用户可以在这里配置远程扩展字典 -->
  <!-- <entry key="remote_ext_dict">words_location</entry> -->
  <!--用户可以在这里配置远程扩展停止词字典-->
  <!-- <entry key="remote_ext_stopwords">words_location</entry> -->
</properties>
```

八、ES基本操作

8.1 ES是基于RESTful实现访问

ES是支持web访问的，但必须遵从RESTful访问规范

ES逻辑结构

- 数据库：数据是存储在数据表中的，数据表是创建在数据库中的

- ES：document是存储在type中的，type是创建在index中
  - index 索引 --- 相当于数据库 （索引的命名不能包含特殊字符，必须小写）
  - type类型 --- 相当于数据表 （在es7以前，一个index中可以创建多个type ）
  - document文档 --- 相当于数据表中的一条记录

RESTful

- 不同操作需要使用不同的请求方式
- 基于REST的基本访问规范

请求方式	REST请求	功能描述
PUT	http://eshost:9200/index1	创建index(索引)
POST	http://eshost:9200/索引名/类型名/文档ID	添加document
POST	http://eshost:9200/索引名/类型名/文档ID/_update	修改document文档
DELETE	http://eshost:9200/索引名/类型名/文档ID	根据ID删除document
GET	http://eshost:9200/索引名/类型名/文档ID	根据ID查询document
POST	http://eshost:9200/索引名/类型名/_search	查询索引下所有数据

8.2 基本操作

- 创建索引

```
# 【基本操作】
# 1.创建索引 PUT
PUT index1

PUT index3
{
  "mappings": {
    "properties": {
      "book_id":{
        "type":"long"
```

```
      },
      "book_name":{
        "type": "text"
      },
      "book_author":{
        "type": "keyword"
      },
      "book_price":{
        "type": "float"
      },
      "book_desc":{
        "type": "text"
      }
    }
  }
}

# 索引是一个逻辑单元，ES中的数据实际上是存储在分片中的,我们可以在settings中设置索引的属性
PUT index2
{
  "settings": {
    "number_of_shards": 2
  }
}
```

- 查询索引

```
# 查询索引信息
GET index1
# 查询索引的mappings信息
GET index1/_mappings
# 查询索引的属性设置
GET index1/_settings
```

- 创建文档：新增一条记录到ES

```
POST index3/_doc/101
{
  "book_id":101,
  "book_name":"Java程序设计",
  "book_author":"千锋亮哥",
```

```
  "book_price":22.22,
  "book_desc":"这是一本看了就会的Java秘籍"
}

POST index3/_doc/102
{
  "book_id":102,
  "book_name":"C++程序设计",
  "book_author":"谭浩强",
  "book_price":22.22,
  "book_desc":"C++程序设计中的名著"
}

POST index3/_doc/103
{
  "book_id":103,
  "book_name":"Python王者归来",
  "book_author":"杰哥",
  "book_price":33.22,
  "book_desc":"Python从入门到放弃"
}
```

注意： 在ES 7.0版本以后，一个index中只能存在一个type(默认名称为_doc)

- 修改文档：修改记录
  - 使用新增操作的请求覆盖原记录

```
POST index3/_doc/103
{
  "book_id":103,
  "book_name":"Python王者归来",
  "book_author":"杰哥",
  "book_price":33.22,
  "book_desc":"Python从入门到放弃"
}
```

  - 使用_update修改

```
POST index3/_doc/103/_update
{
  "book_id":103,
  "book_name":"Python王者归来",
  "book_author":"杰哥",
  "book_price":33.22,
  "book_desc":"Python从入门到放弃"
}
```

- 查询文档

```
根据文档id查询数据
GET index3/_doc/101

查询索引中的所有数据（type使用自定名称）
POST index3/_doc/_search
```

- 删除文档

```
DELETE index3/_doc/103
```

- 查看es状态 _cat

```
GET _cat/indices?v
GET _cat/health?v
```

8.3 数据类型

> es中一个document表示一条记录，记录中field值的存储是有类型的

https://www.elastic.co/guide/en/elasticsearch/reference/6.5/mapping-types.html

string

- text 可分词
- keyword 不能分词

Numeric datatypes

`long`, `integer`, `short`, `byte`, `double`, `float`, `half_float`, `scaled_float`

Date datatype

- data --- 日期的存储时以 long 类型存储的毫秒数

Boolean datatype

- boolean --- true | false | "true" | "false"

Binary datatype

- binary 基于base64编码的字符串

Range datatypes

`integer_range`, `float_range`, `long_range`, `double_range`, `date_range`

创建Index并指定field类型

```
PUT index3
{
  "mappings": {
    "properties": {
      "bookId":{
        "type": "long"
      },
      "bookName":{
        "type": "text"
      },
      "author":{
        "type": "keyword"
      },
      "time":{
        "type": "date"
      }
    }
  }
}

GET index3/_doc/_search

POST index3/_doc/1
{
  "bookId":10001,
  "bookName":"Java程序设计",
  "author":"张三",
  "time":234567890
```

```
}
```

8.4 复杂查询-数据搜索

8.4.1 数据准备

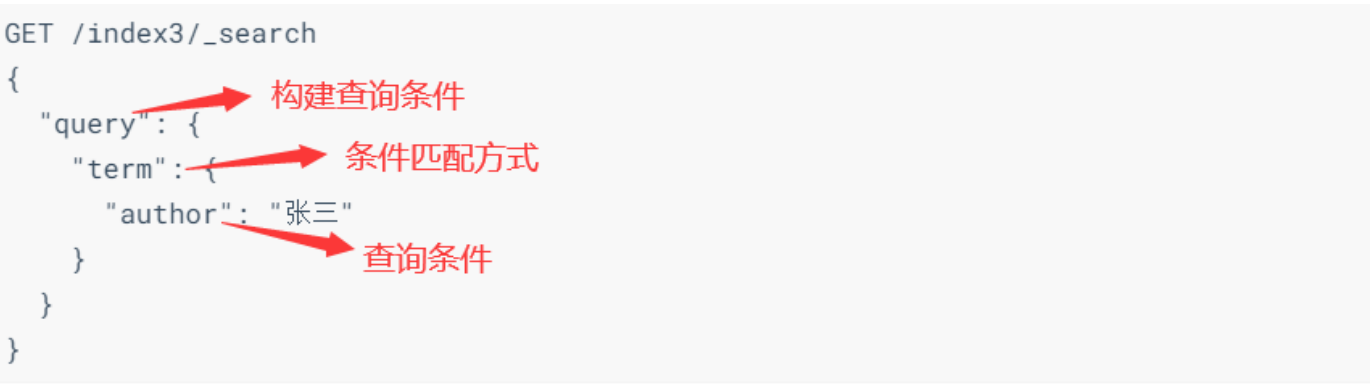
```
PUT index4
{
  "mappings": {
    "properties": {
      "bookId":{
        "type": "long"
      },
      "bookName":{
        "type": "text"
      },
      "author":{
        "type": "keyword"
      },
      "time":{
        "type": "date"
      }
    }
  }
}

POST index4/_doc/1
{
  "bookId":10001,
  "bookName":"Java程序设计",
  "author":"张三",
  "time":234567890
}
POST index4/_doc/2
{
  "bookId":10002,
  "bookName":"C语言程序设计",
  "author":"Java谭浩强",
  "time":2345678999
}
POST index4/_doc/3
{
```

```
  "bookId":10003,
  "bookName":"程序设计进阶",
  "author":"李三",
  "time":2345678222
}
POST index4/_doc/4
{
  "bookId":10004,
  "bookName":"Java编程思想",
  "author":"三毛",
  "time":23456783452
}

```

8.4.2 复杂查询语法



8.4.3 term和terms

> 用于对keyword字段进行精确匹配

- term 表示完全匹配，搜索之前不会对关键字进行分词

```
GET /index3/_search
{
  "query": {
    "term": {
      "author": "涛哥"
    }
  }
}
```

- terms 也表示完全匹配，可以为一个field指定多个匹配关键词

```
GET /index3/_search
{
  "query": {
    "terms": {
      "author": ["涛哥","李三"]
    }
  }
}
```

8.4.4 match查询（重点）

> match查询表示对text字段进行部分匹配（模糊查询）

- `match` 表示部分匹配，搜索之前会对关键词进行分词

```
GET /index4/_search
{
  "query": {
    "match": {
      "bookName": "Java程序"
    }
  }
}
```

- match_all 表示查询全部内容，不指定任何条件

```
GET /index4/_search
{
  "query": {
    "match_all": {}
  }
}
```

- `multi_match` 在多个字段中匹配同一个关键字

```
GET /index4/_search
{
  "query": {
    "multi_match": {
      "query": "Java",
      "fields": ["bookName","author"]
    }
  }
}
```

8.4.5 根据id查询

- 根据一个id查询一个document

```
GET /index4/_doc/1
```

- 根据多个id查询多个document ==> select * from ... where id in [1,2,3]

```
GET /index4/_search
{
  "query":{
    "ids":{
      "values":["1","2","3"]
    }
  }
}
```

8.4.5 其他查询

- prefix查询，根据指定字段的前缀值进行查询

```
GET /index4/_search
{
  "query": {
    "prefix": {
      "author": {
        "value": "张"
      }
    }
  }
}
```

- fuzzy查询，模糊查询，输入大概的内容es检索相关的数据

```
GET /index4/_search
{
  "query": {
    "fuzzy": {
      "bookName": {
        "value": "jav"
      }
    }
  }
}
```

- wildcard查询：正则匹配

```
GET /index4/_search
{
    "query": {
        "wildcard": {
            "author": {
                "value": "张*"
            }
        }
    }
}
```

- range查询，根据范围进行查询

```
GET /index4/_search
{
    "query": {
        "range" : {
            "bookId" : {
                "gt" : 10001,
                "lte" : 10003
            }
        }
    }
}
```

- 分页查询

```
GET /index4/_search
{
  "query": {
    "match_all": {}
  },
  "_source": ["bookId","bookName"],
  "from": 0,
  "size": 20
}
```

8.5 复合查询—bool

> 复合查询——多条件查询
>
> - should ==> or
> - must ==> and
> - must_not ==> not

```
GET /index4/_search
{
  "query": {
    "bool":{
      "must_not": [
        {
          "match": {
            "bookName": "Java"
          }
        },
        {
          "match": {
            "author": "张三"
          }
        }
      ]
    }
  }
}
```

8.6 结果过滤—filter

> filter——根据条件进行查询，不计算分数，会对经常被过滤的数据进行缓存

```
GET /index3/_search
{
  "query": {
    "bool":{
      "filter": [
        {
          "match": {
            "bookName": "Java"
          }
        },
        {
          "match": {
            "author": "张三"
          }
        }
      ]
    }
  }
}
```

8.7 高亮显示（重点）

> 对匹配的关键词进行特殊样式的标记

```
GET /index3/_search
{
  "query": {
    "match": {
      "bookName": "Java"
    }
  },
  "highlight": {
    "fields": {
      "bookName": {}
    },
    "pre_tags": "<label style='color:red'>",
    "post_tags": "</label>"
```

```
    }
}
```

九、SpringBoot整合ES

官方参考地址 https://www.elastic.co/guide/en/elasticsearch/client/index.html

- RestLowerLevelClient
- RestHighLevelClient

9.1 创建SpringBoot应用

略

9.2 添加es的依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
</dependency>
```

9.3 配置Bean

> 在springboot应用中已经提供了RestHighLevelClient实例，无需进行实例配置，但是需要进行es服务器地址配置

```
@Bean
public RestHighLevelClient getRestHighLevelClient(){
    HttpHost httpHost = new HttpHost("47.96.11.185", 9200, "http");
    RestClientBuilder restClientBuilder = RestClient.builder(httpHost);
    RestHighLevelClient restHighLevelClient = new
RestHighLevelClient(restClientBuilder);
    return restHighLevelClient;
}
```

在springboot应用配置连接：

```yaml
spring:
  elasticsearch:
    rest:
      uris: http://47.96.11.185:9200
```

9.4 使用案例

```
@SpringBootTest
class Esdemo3ApplicationTests {


    @Resource
    private RestHighLevelClient restHighLevelClient;


    /**
     * 在es中创建索引
     */
    @Test
    public void testCreateIndex() throws IOException {
        CreateIndexRequest createIndexRequest = new CreateIndexRequest("index4");
        CreateIndexResponse createIndexResponse = restHighLevelClient.indices().create(createIndexRequest, RequestOptions.DEFAULT);
        System.out.println(createIndexResponse);
    }

    /**
     * 删除索引
     */
    @Test
    public void testDeleteIndex() throws IOException {
        DeleteIndexRequest deleteIndexRequest = new DeleteIndexRequest("index4");
        AcknowledgedResponse deleteIndexRes = restHighLevelClient.indices().delete(deleteIndexRequest, RequestOptions.DEFAULT);
        System.out.println(deleteIndexRes);
    }

```

```
    /**
     * 添加文档：将数据存入es
     */
    @Test
    public void testCreateDocument() throws IOException {
        Book book = new Book(10005,"平凡的世界","路遥",new
Date().getTime());
        ObjectMapper objectMapper = new ObjectMapper();
        String jsonStr = objectMapper.writeValueAsString(book);

        IndexRequest request = new IndexRequest("index3");
        request.id("10005");
        request.source(jsonStr, XContentType.JSON);
        IndexResponse indexResponse =
restHighLevelClient.index(request, RequestOptions.DEFAULT);
        System.out.println(indexResponse);
    }

    /**
     * 搜索
     */
    @Test
    public void testSearch() throws IOException {
        SearchRequest searchRequest = new SearchRequest("index3");

        SearchSourceBuilder searchSourceBuilder = new
SearchSourceBuilder();
        searchSourceBuilder.from(0);
        searchSourceBuilder.size(10);
//        searchSourceBuilder.query(QueryBuilders.matchAllQuery());

 searchSourceBuilder.query(QueryBuilders.matchQuery("bookName","Java"))
;

        HighlightBuilder highlightBuilder = new HighlightBuilder();
        HighlightBuilder.Field highlightTitle = new
HighlightBuilder.Field("bookName");
        highlightTitle.highlighterType("unified");
        highlightBuilder.field(highlightTitle);
        highlightBuilder.preTags("<label style='color:red'>");
        highlightBuilder.postTags("</label>");

```

```
65
66          searchSourceBuilder.highlighter(highlightBuilder);
67          searchRequest.source(searchSourceBuilder);
68
69          SearchResponse searchResp =
restHighLevelClient.search(searchRequest, RequestOptions.DEFAULT);
70
71          SearchHits hits = searchResp.getHits();
72          for (SearchHit hit : hits){
73              System.out.println(hit);
74          }
75
76      }
77
78  }
```

批量添加（参考代码）

```
1   /**
2       *  批量插入ES
3       * @param indexName 索引
4       * @param type 类型
5       * @param idName id名称
6       * @param list 数据集合
7       */
8   public void bulkData(String indexName,String type ,String idName
,List<Map<String,Object>> list ){
9       try {
10          if(null == list || list.size()<=0){
11              return;
12          }
13
if(StringUtils.isBlank(indexName)||StringUtils.isBlank(idName)||StringU
tils.isBlank(type))          {
14              return;
15          }
16          BulkRequest request = new BulkRequest();
17          for(Map<String,Object> map : list){
18              if(map.get(idName)!=null){
19                  request.add(new IndexRequest(indexName, type,
String.valueOf(map.get(idName)))
20                              .source(map,XContentType.JSON));
```

```
            }
        }
        // 2、可选的设置
        /*
           request.timeout("2m");
           request.setRefreshPolicy("wait_for");
           request.waitForActiveShards(2);
           */
        //3、发送请求:同步请求
        BulkResponse bulkResponse = client.bulk(request);
        //4、处理响应
        if(bulkResponse != null) {
            for (BulkItemResponse bulkItemResponse : bulkResponse) {
                DocWriteResponse itemResponse =
bulkItemResponse.getResponse();

                if (bulkItemResponse.getOpType() ==
DocWriteRequest.OpType.INDEX
                    || bulkItemResponse.getOpType() ==
DocWriteRequest.OpType.CREATE) {
                    IndexResponse indexResponse = (IndexResponse)
itemResponse;
                    //TODO 新增成功的处理
                    System.out.println("新增成功,{}"+
indexResponse.toString());
                } else if (bulkItemResponse.getOpType() ==
DocWriteRequest.OpType.UPDATE) {
                    UpdateResponse updateResponse = (UpdateResponse)
itemResponse;
                    //TODO 修改成功的处理
                    System.out.println("修改成功,{}"+
updateResponse.toString());
                } else if (bulkItemResponse.getOpType() ==
DocWriteRequest.OpType.DELETE) {
                    DeleteResponse deleteResponse = (DeleteResponse)
itemResponse;
                    //TODO 删除成功的处理
                    System.out.println("删除成功,{}"+
deleteResponse.toString());
                }
            }
        }
```

```
52      } catch (IOException e) {
53          e.printStackTrace();
54      }
55  }
```

查询数据封装

```
1  Iterator<SearchHit> iterator = hits.iterator();
2  List<Product>  products = new ArrayList<>();
3  while(iterator.hasNext()){
4      SearchHit searchHit = iterator.next();
5      String str = searchHit.getSourceAsString();
6      Product product = objectMapper.readValue(str, Product.class);
7
8      HighlightField highlightField =
   searchHit.getHighlightFields().get("productName");
9      if(highlightField != null){
10         String s = Arrays.toString(highlightField.fragments());
11         product.setProductName(s);
12     }
13
14     products.add(product);
15 }
```

千锋教育Java教研院 关注公众号【Java架构栈】下载所有课程代码课件及工具 让技术回归本该有的纯静!