

# 分布式检索引擎Elasticsearch实践

## 0. 能力目标

- 掌握ES的安装与基本操作
- 掌握ES的高可用集群
- 掌握ELK的部署及工作机制
- 掌握ES的高阶操作
- 掌握ES的实战技巧

## 1. ElasticSearch快速入门

### 1.1. 基本介绍

- **ElasticSearch特色**

Elasticsearch是实时的分布式搜索分析引擎，内部使用Lucene做索引与搜索

- 实时性：新增到 ES 中的数据在1秒后就可以被检索到，这种新增数据对搜索的可见性称为“准实时搜索”
- 分布式：意味着可以动态调整集群规模，弹性扩容
- 集群规模：可以扩展到上百台服务器，处理PB级结构化或非结构化数据
- 各节点组成对等的网络结构，某些节点出现故障时会自动分配其他节点代替其进行工作

Lucene是Java语言编写的全文搜索框架，用于处理纯文本的数据，但它只是一个库，提供建立索引、执行搜索等接口，但不包含分布式服务，这些正是 ES 做的

- **ElasticSearch使用场景**

ElasticSearch广泛应用于各行业领域，比如维基百科，GitHub的代码搜索，电商网站的大数据日志统计分析，BI系统报表统计分析等。

- 提供分布式的搜索引擎和数据分析引擎

比如百度，网站的站内搜索，IT系统的检索，数据分析比如热点词统计，电商网站商品TOP排名等。

- 全文检索，结构化检索，数据分析

支持全文检索，比如查找包含指定名称的商品信息；支持结构检索，比如查找某个分类下的所有商品信息；

还可以支持高级数据分析，比如统计某个商品的点击次数，某个商品有多少用户购买等等。

- 支持海量数据准实时的处理

采用分布式节点，将数据分散到多台服务器上去存储和检索，实现海量数据的处理，比如统计用户的行为日志，能够在秒级别对数据进行检索和分析。

- **ElasticSearch基本概念介绍**

ElasticSearch	Relational Database
Index	Database
Type	Table
Document	Row
Field	Column
Mapping	Schema
Everything is indexed	Index
Query DSL	SQL
GET http://...	SELECT * FROM table...
PUT http://...	UPDATE table SET...

- 索引 ( Index)

相比传统的关系型数据库，索引相当于SQL中的一个【数据库】，或者一个数据存储方案 (schema)。

- 类型 ( Type )

一个索引内部可以定义一个或多个类型，在传统关系数据库来说，类型相当于【表】的概念。

- 文档 ( Document )

文档是Lucene索引和搜索的原子单位，它是包含了一个或多个域的容器，采用JSON格式表示。相当于传统数据库【行】概念

- 集群 ( Cluster )

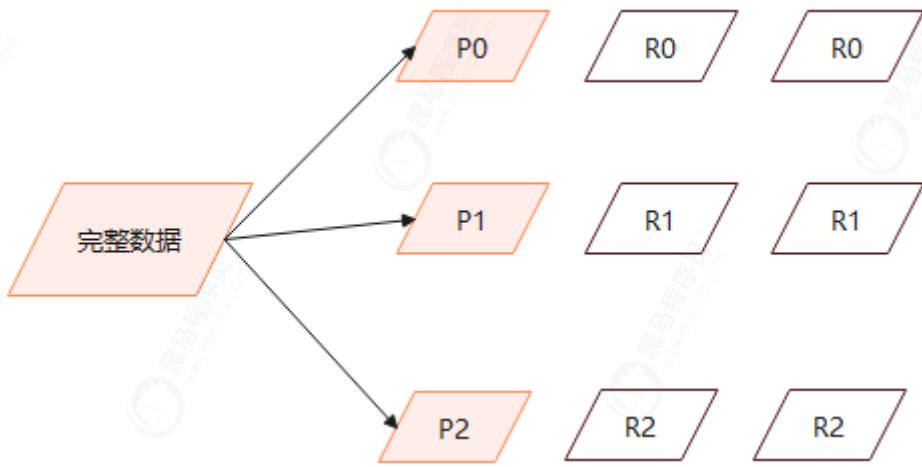
集群是由一台及以上主机节点组成并提供存储及搜索服务，多节点组成的集群拥有冗余能力，它可以在一个或几个节点出现故障时保证服务的整体可用性。

- 节点 ( Node )

Node为集群中的单台节点，其可以为master节点亦可为slave节点（节点属性由集群内部选举得出）并提供存储相关数据的功能

- 切片 ( shards)

切片是把一个大文件分割成多个小文件然后分散存储在集群中的多个节点上，可以将其看作mysql的分库分表概念。Shard有两种类型：primary主片和replica副本，primary用于文档存储，每个新的索引会自动创建5个Primary shard；Replica shard是Primary Shard的副本，用于冗余数据及提高搜索性能。



注意：ES7之后Type被舍弃，只有Index(等同于数据库+表定义)和Document(文档，行记录)。

## 1.2 ElasticSearch安装

### 1. 下载ElasticSearch服务

下载最新版ElasticSearch7.10.2：<https://www.elastic.co/cn/start>

### 2. 解压安装包

```
tar -xvf elasticsearch-7.10.2-linux-x86_64.tar.gz
```

### 3. ElasticSearch不能以Root身份运行，需要单独创建一个用户

```
1. groupadd elasticsearch
2. useradd elasticsearch -g elasticsearch -p elasticsearch
3. chown -R elasticsearch:elasticsearch /usr/local/elasticsearch-7.10.2
```

执行以上命令，创建一个名为elasticsearch用户，并赋予目录权限。

### 4. 修改配置文件

vi config/elasticsearch.yml, 默认情况下会绑定本机地址，外网不能访问，这里要修改下：

```
# 外网访问地址
network.host: 0.0.0.0
```

### 5. 关闭防火墙

```
systemctl stop firewalld.service
systemctl disable firewalld.service
```

### 6. 指定JDK版本

- 最新版的ElasticSearch需要JDK11版本，下载[JDK11压缩包](#)，并进行解压。
- 修改环境配置文件

vi bin/elasticsearch-env

参照以下位置，追加一行，设置JAVA\_HOME，指定JDK11路径。

```
JAVA_HOME=/usr/local/jdk-11.0.11

# now set the path to java
if [ ! -z "$JAVA_HOME" ]; then
    JAVA="$JAVA_HOME/bin/java"
else
    if [ "$(uname -s)" = "Darwin" ]; then
        # OSX has a different structure
        JAVA="$ES_HOME/jdk/Contents/Home/bin/java"
    else
        JAVA="$ES_HOME/jdk/bin/java"
    fi
fi
```

## 7. 启动ElasticSearch

- 切换用户

```
su elasticsearch
```

- 以后台常驻方式启动

```
bin/elasticsearch -d
```

## 8. 问题记录

出现max virtual memory areas vm.max\_map\_count [65530] is too low, increase to at least 错误信息

修改系统配置：

- vi /etc/sysctl.conf

添加

```
vm.max_map_count=655360
```

执行生效

```
sysctl -p
```

- vi /etc/security/limits.conf

在文件末尾添加

```
* soft nofile 65536
* hard nofile 131072
* soft nproc 2048
* hard nproc 4096
elasticsearch soft nproc 125535
elasticsearch hard nproc 125535
```

重新切换用户即可：

```
su - elasticsearch
```

## 9. 访问验证

访问地址：[http://192.168.116.140:9200/\\_cat/health](http://192.168.116.140:9200/_cat/health)

启动状态有green、yellow和red。 green是代表启动正常。

## 1.3 Kibana服务安装

Kibana是一个针对Elasticsearch的开源分析及可视化平台，用来搜索、查看交互存储在Elasticsearch索引中的数据。

1. 到官网下载，[Kibana安装包](#)，与之对应7.10.2版本，选择Linux 64位版本下载，并进行解压。

```
tar -xvf kibana-7.10.2-linux-x86_64.tar.gz
```

2. Kibana启动不能使用root用户，使用上面创建的elasticsearch用户，进行赋权：

```
chown -R elasticsearch:elasticsearch kibana-7.10.2-linux-x86_64
```

3. 修改配置文件

vi config/kibana.yml，修改以下配置：

```
# 服务端口
server.port: 5601
# 服务地址
server.host: "0.0.0.0"
# elasticsearch服务地址
elasticsearch.hosts: ["http://192.168.116.140:9200"]
```

4. 启动kibana

```
./kibana -q
```

看到以下日志，代表启动正常

```
log [01:40:00.143] [info][listening] server running at http://0.0.0.0:5601
```

如果出现启动失败的情况，要检查集群各节点的日志，确保服务正常运行状态。

5. 访问服务

<http://192.168.116.140:5601/app/home#/>

## 1.4 ES的基础操作

1. 进入Kibana管理后台

地址：<http://192.168.116.140:5601>

进入"Dev Tools"栏：

在Console中输入命令进行操作。

## 2. 分片设置：

这里增加名为orders的索引，因为是单节点，如果副本数，是会出现错误。

```
PUT orders
{
  "settings": {
    "index": {
      "number_of_shards": 2,
      "number_of_replicas": 2
    }
  }
}
```

查看索引信息，会出现yellow提示：

Name	Health	Status	Primaries	Replicas	Docs count	Storage
orders	yellow	open	2	2	0	416b

因为单节点模式，只有主节点信息：

删除重新创建：

```
PUT orders
{
  "settings": {
    "index": {
      "number_of_shards": 2,
      "number_of_replicas": 0
    }
  }
}
```

将分片数设为0，再次查看，则显示正常：

The screenshot shows the Elasticsearch Index Management interface. On the left, there's a sidebar with sections for Ingest, Data (with Index Management selected), and Alerts and Insights. The main area is titled 'Index Management' and has tabs for Indices, Data Streams, Index Templates, and Component Templates. Below the tabs, there's a search bar and a checkbox for 'Include rollup indices'. A table lists indices with columns for Name, Health, Status, Primaries, Replicas, and Docs count. The 'orders' index is listed with a green health status, 2 primaries, 0 replicas, and 0 documents. A red box highlights the 'Health' column for the 'orders' row.

Name	Health	Status	Primaries	Replicas	Docs count
orders	green	open	2	0	0

### 3. 索引

#### 3.1 新建索引orders

```
## 创建索引
PUT orders
```

#### 3.2 查询索引orders

```
## 查询索引
GET orders
```

通过查询命令，能查看到对应信息，默认分片数和副本数都为1：

```
"number_of_shards" : "1", ## 主分片数
"number_of_replicas" : "1", ## 副分片数
```

# Index Management

[Index Management docs](#)

Indices Data Streams Index Templates Component Templates

Update your Elasticsearch indices individually or in bulk. [Learn more.](#)

Include rollup indices    Include hidden indices

Name	Health	Status	Primaries	Replicas	Docs count	Storage size	Data stream
orders	yellow	open	1	1	0	208b	

Rows per page: 10 < 1 >

## 3.3 删除索引

```
## 删除索引  
DELETE orders
```

## 3.4 索引的设置

```
## 设置索引  
PUT orders  
{  
  "settings": {  
    "index": {  
      "number_of_shards": 1,  
      "number_of_replicas": 0  
    }  
  }  
}
```

# 4. 文档

## 4.1 创建文档

```
## 创建文档，生成默认的文档id  
POST orders/_doc  
{  
  "name": "袜子1双",  
  "price": "200",  
  "count": 1,  
  "address": "北京市"  
}  
## 创建文档，生成自定义文档id  
POST orders/_doc/1  
{  
  "name": "袜子1双",  
  "price": "2",  
  "count": 1,  
  "address": "北京市"  
}
```

## 4.2 查询文档

```
## 根据指定的id查询
GET orders/_doc/1
## 根据指定条件查询文档
GET orders/_search
{
  "query": {
    "match": {
      "address": "北京市"
    }
  }
}
## 查询全部文档
GET orders/_search
```

#### 4.3 更新文档

```
## 更新文档
POST orders/_doc/1
{
  "price": "200"
}
## 更新文档
POST orders/_update/1
{
  "doc": {
    "price": "200"
  }
}
```

#### 4.4 删除文档

```
## 删除文档
DELETE orders/_doc/1
```

### 5. 域

对于映射，只能进行字段添加，不能对字段进行修改或删除，如有需要，则重新创建映射。

```
## 设置mapping信息
PUT orders/_mappings
{
  "properties": {
    "price": {
      "type": "long"
    }
  }
}
## 设置分片和映射
PUT orders
{
  "settings": {
    "index": {
      "number_of_shards": 1,
      "number_of_replicas": 0
    }
  },
}
```

```

"mappings": {
  "properties": {
    "name": {
      "type": "text"
    },
    "price": {
      "type": "long"
    },
    "count": {
      "type": "long"
    },
    "address": {
      "type": "text"
    }
  }
}

```

## 1.5 ES数据类型

整体数据类型结构：

一级分类	二级分类	具体类型
核心类型	字符串类型	string, text, keyword
	整数类型	integer, long, short, byte
	浮点类型	double, float, half_float, scaled_float
	逻辑类型	boolean
	日期类型	date
	范围类型	range
	二进制类型	binary
复合类型	数组类型	array
	对象类型	object
	嵌套类型	nested
地理类型	地理坐标类型	geo_point
	地理地图	geo_shape
特殊类型	IP类型	ip
	范围类型	completion
	令牌计数类型	token_count
	附件类型	attachment
	抽取类型	percolator

### 1. String 类型

主要分为text与keyword两种类型。两者区别主要在于能否分词。

- text类型

会进行分词处理，分词器默认采用的是standard。

- keyword类型

不会进行分词处理。在ES的倒排索引中存储的是完整的字符串。

## 2. Date时间类型

数据库里的日期类型需要规范具体的传入格式，ES是可以控制，自适应处理。

传递不同的时间类型：

```
PUT my_date_index/_doc/1
{ "date": "2021-01-01" }

PUT my_date_index/_doc/2
{ "date": "2021-01-01T12:10:30Z" }

PUT my_date_index/_doc/3
{ "date": 1520071600001 }
```

查看日期数据：

```
GET my_date_index/_mapping
```

```
1 { 
2   "my_date_index" : {
3     "mappings" : {
4       "properties" : {
5         "date" : {
6           "type" : "date"
7         }
8       }
9     }
10   }
11 }
```

ES的Date类型允许可以使用的格式有：

```
yyyy-MM-dd HH:mm:ss
yyyy-MM-dd
epoch_millis (毫秒值)
```

## 3. 复合类型

复杂类型主要有三种：Array、object、nested。

- Array类型：在Elasticsearch中，数组不需要声明专用的字段数据类型。但是，在数组中的所有值都必须具有相同的数据类型。举例：

```
POST orders/_doc/1
{
  "goodsName": ["足球", "篮球", "兵乓球", 3]
}

POST orders/_doc/1
{
  "goodsName": ["足球", "篮球", "兵乓球"]
}
```

- object类型：用于存储单个JSON对象，类似于JAVA中的对象类型，可以有多个值，比如LIST，可以包含多个对象。

但是LIST只能作为整体，不能独立的索引查询。举例：

```
# 新增第一组数据， 组别为美国，两个人。  
POST my_index/_doc/1  
{  
    "group" : "america",  
    "users" : [  
        {  
            "name" : "John",  
            "age" : "22"  
        },  
        {  
            "name" : "Alice",  
            "age" : "21"  
        }  
    ]  
}  
  
# 新增第二组数据， 组别为英国， 两个人。  
POST my_index/_doc/2  
{  
    "group" : "england",  
    "users" : [  
        {  
            "name" : "Lucy",  
            "age" : "21"  
        },  
        {  
            "name" : "John",  
            "age" : "32"  
        }  
    ]  
}
```

这两组数据都包含了name为John， age为21的数据，

采用这个搜索条件， 实际结果：

```
GET my_index/_search  
{  
    "query": {  
        "bool": {  
            "must": [  
                {  
                    "match": {  
                        "users.name": "John"  
                    }  
                },  
                {  
                    "match": {  
                        "users.age": 21  
                    }  
                }  
            ]  
        }  
    }  
}
```

```
        }
    }
```

结果可以看到，这两组数据都能找出，因为每一组数据都是作为一个整体进行搜索匹配，而非具体某一条数据。

- Nested类型

用于存储多个JSON对象组成的数组，`nested` 类型是 `object` 类型中的一个特例，可以让对象数组独立索引和查询。

举例：

创建nested类型的索引：

```
PUT my_index
{
  "mappings": {
    "properties": {
      "users": {
        "type": "nested"
      }
    }
  }
}
```

发出查询请求：

```
GET my_index/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "nested": {
            "path": "users",
            "query": {
              "bool": {
                "must": [
                  {
                    "match": {
                      "users.name": "John"
                    }
                  },
                  {
                    "match": {
                      "users.age": "21"
                    }
                  }
                ]
              }
            }
          }
        }
      ]
    }
  }
}
```

采用以前的条件，这个时候查不到任何结果，将年龄改成22，就可以找出对应的数据：

```
"hits" : [
  {
    "_index" : "my_index",
    "_type" : "_doc",
    "_id" : "1",
    "_score" : 1.89712,
    "_source" : {
      "group" : "america",
      "users" : [
        {
          "name" : "John",
          "age" : "22"
        },
        {
          "name" : "Alice",
          "age" : "21"
        }
      ]
    }
]
```

#### 4. GEO地理位置类型

现在大部分APP都有基于位置搜索的功能，比如交友、购物应用等。这些功能是基于GEO搜索实现的。

对于GEO地理位置类型，分为地图：Geo-point，和形状：Geo-shape 两种类型。

经纬度	英文	简写	正数	负数
维度	latitude	lat	北纬	南纬
经度	longitude	lon或lng	东经	西经

创建地理位置索引：

```
PUT my_locations
{
  "mappings": {
    "properties": {
      "location": {
        "type": "geo_point"
      }
    }
  }
}
```

添加地理位置数据：

```
# 采用object对象类型
PUT my_locations/_doc/1
{
```

```

    "user": "张三",
    "text": "Geo-point as an object",
    "location": {
        "lat": 41.12,
        "lon": -71.34
    }
}
# 采用string类型
PUT my_locations/_doc/2
{
    "user": "李四",
    "text": "Geo-point as a string",
    "location": "45.12,-75.34"
}
# 采用geohash类型（geohash算法可以将多维数据映射为一串字符）
PUT my_locations/_doc/3
{
    "user": "王二麻子",
    "text": "Geo-point as a geohash",
    "location": "drm3btev3e86"
}
# 采用array数组类型
PUT my_locations/_doc/4
{
    "user": "木头老七",
    "text": "Geo-point as an array",
    "location": [
        -80.34,
        51.12
    ]
}

```

需求：搜索出距离我{"lat": 40,"lon": -70} 200km范围内的人：

```

GET my_locations/_search
{
    "query": {
        "bool": {
            "must": {
                "match_all": {}
            },
            "filter": {
                "geo_distance": {
                    "distance": "200km",
                    "location": {
                        "lat": 40,
                        "lon": -70
                    }
                }
            }
        }
    }
}

```

## 2. ES高可用集群配置

### 2.1 ElasticSearch集群介绍

- **主节点(或候选主节点)**

主节点负责创建索引、删除索引、分配分片、追踪集群中的节点状态等工作，主节点负荷相对较轻，客户端请求可以直接发往任何节点，由对应节点负责分发和返回处理结果。

一个节点启动之后，采用Zen Discovery机制去寻找集群中的其他节点，并与之建立连接，集群会从候选主节点中选举出一个主节点，并且一个集群只能选举一个主节点，在某些情况下，由于网络通信丢包等问题，一个集群可能会出现多个主节点，称为“脑裂现象”，脑裂会存在丢失数据的可能，因为主节点拥有最高权限，它决定了什么时候可以创建索引，分片如何移动等，如果存在多个主节点，就会产生冲突，容易产生数据丢失。要尽量避免这个问题，可以通过discovery.zen.minimum\_master\_nodes来设置最少可工作的候选主节点个数。建议设置为(候选主节点/2)+1比如三个候选主节点，该配置项为(3/2)+1，来保证集群中有半数以上的候选主节点，没有足够的master候选节点，就不会进行master节点选举，减少脑裂的可能。

主节点的参数设置：

```
node.master = true  
node.data = false
```

- **数据节点**

数据节点负责数据的存储和CRUD等具体操作，数据节点对机器配置要求比较高，首先需要有足够的磁盘空间来存储数据，其次数据操作对系统CPU、Memory和IO的性能消耗都很大。通常随着集群的扩大，需要增加更多的数据节点来提高可用性。

数据节点的参数设置：

```
node.master = false  
node.data = true
```

- **客户端节点**

客户端节点不做候选主节点，也不做数据节点的节点，只负责请求的分发、汇总等等，增加客户端节点类型更多是为了负载均衡的处理。

```
node.master = false  
node.data = false
```

- **提取节点(预处理节点)**

能执行预处理管道，有自己独立的任务要执行，在索引数据之前可以先对数据做预处理操作，不负责数据存储也不负责集群相关的事务。

参数设置：

```
node.ingest = true
```

- **协调节点**

协调节点，是一种角色，而不是真实的Elasticsearch的节点，不能通过配置项来指定哪个节点为协调节点。集群中的任何节点，都可以充当协调节点的角色。当一个节点A收到用户的查询请求后，会把查询子句分发到其它的节点，然后合并各个节点返回的查询结果，最后返回一个完整的数据集给用户。在这个过程中，节点A扮演的就是协调节点的角色。

ES的一次请求非常类似于Map-Reduce操作。在ES中对应的也是两个阶段，称之为scatter-gather。客户端发出一个请求到集群的任意一个节点，这个节点就是所谓的协调节点，它会把请求转发给含有相关数据的节点(scatter阶段)，这些数据节点会在本地执行请求然后把结果返回给协调节点。协调节点将这些结果汇总(reduce)成一个单一的全局结果集(gather阶段)。

- **部落节点**

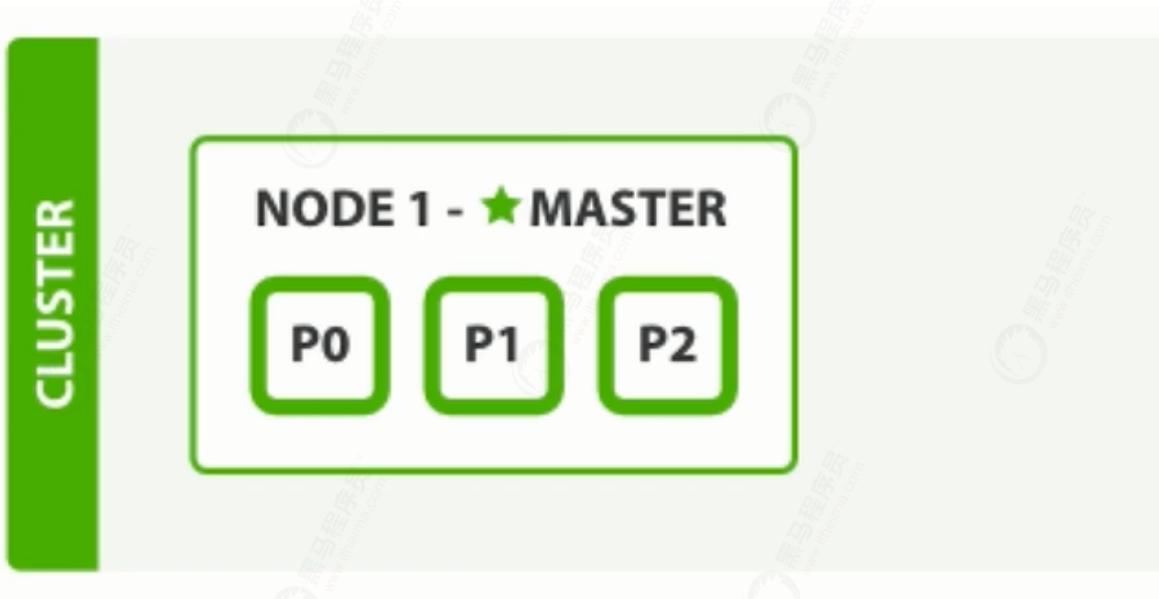
在多个集群之间充当联合客户端，它是一个特殊的客户端，可以连接多个集群，在所有连接的集群上执行搜索和其他操作。部落节点从所有连接的集群中检索集群状态并将其合并成全局集群状态。掌握这一信息，就可以对所有集群中的节点执行读写操作，就好像它们是本地的。请注意，部落节点需要能够连接到每个配置的集群中的每个单个节点。

## 2.2 ElasticSearch集群原理

### 2.2.1 集群分布式原理

ES集群可以根据节点数，动态调整分片与副本数，做到整个集群有效均衡负载。

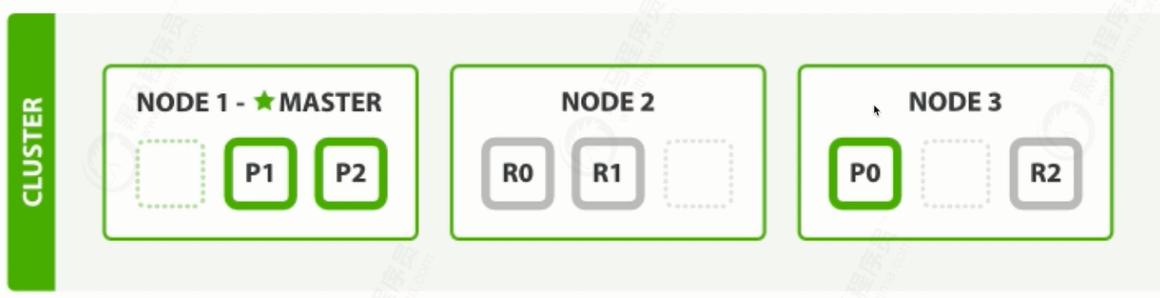
单节点状态下：



两个节点状态下，副本数为1：



三个节点状态下，副本数为1：



三个节点状态下，副本数为2：



## 2.2.2 分片处理机制

设置分片大小的时候，需预先做好容量规划，如果节点数过多，分片数过小，那么新的节点将无法分片，不能做到水平扩展，并且单个分片数据量太大，导致数据重新分配耗时过大。

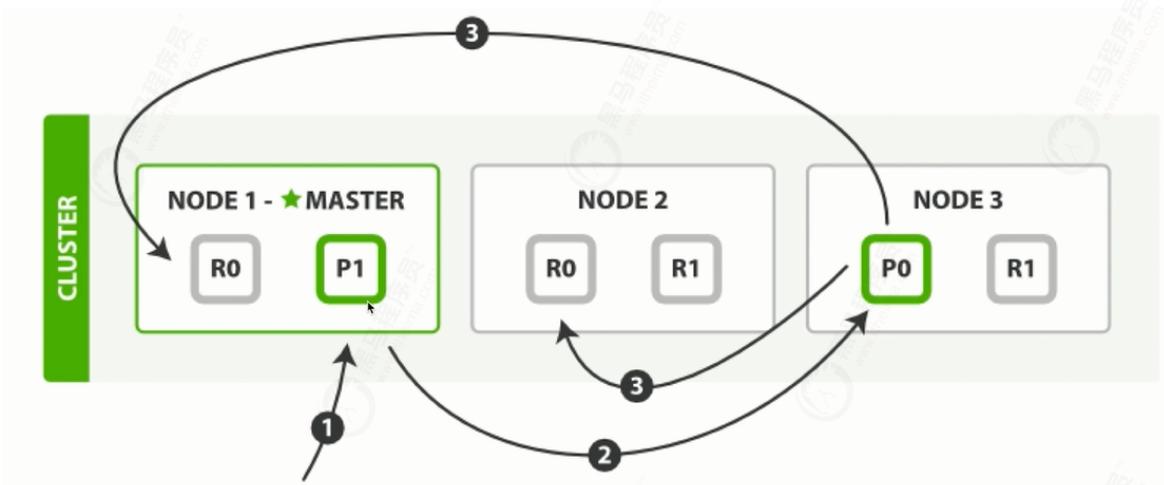
假设一个集群中有一个主节点、两个数据节点。orders索引的分片分布情况如下所示：

```
PUT orders
{
  "settings": {
    "number_of_shards": 2, ## 主分片
    "number_of_replicas": 2 ## 副分片
  }
}
```



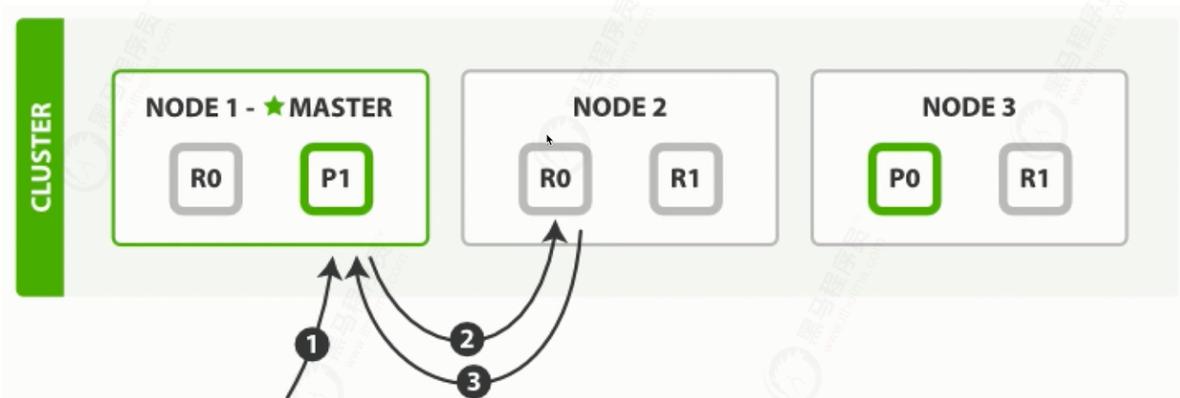
整个集群中存在P0和P1两个主分片，P0对应的两个R0副本分片，P1对应的是两个R1副本分片。

## 2.2.3 新建索引处理流程



1. 写入的请求会进入主节点，如果是NODE2副本接收到写请求，会将它转发至主节点。
2. 主节点接收到请求后，根据documentId做取模运算（外部没有传递documentId，则会采用内部自增ID），  
如果取模结果为P0，则会将写请求转发至NODE3处理。
3. NODE3节点写请求处理完成之后，采用异步方式，将数据同步至NODE1和NODE2节点。

#### 2.2.4 读取索引处理流程



1. 读取的请求进入MASTER节点，会根据取模结果，将请求转发至不同的节点。
2. 如果取模结果为R0，内部还会有负载均衡处理机制，如果上一次的读取请求是在NODE1的R0，那么当前请求会转发至NODE2的R0，保障每个节点都能够均衡的处理请求数量。
3. 读取的请求如果是直接落至副本节点，副本节点会做判断，若有数据则返回，没有的话会转发至其他节点处理。

### 2.3 ElasticSearch集群部署规划

准备一台虚拟机：

192.168.116.140 : Node-1 (节点一)，端口：9200，9300

192.168.116.140 : Node-2 (节点二)，端口：9201，9301

192.168.116.140 : Node-3 (节点三)，端口：9202，9302

## 2.4 ElasticSearch集群配置

1. 解压安装包：

```
cd /usr/local/cluster  
tar -xvf elasticsearch-7.10.2-linux-x86_64.tar.gz
```

将安装包解压至/usr/local/cluster目录。

2. 修改集群配置文件：

```
vi /usr/local/cluster/elasticsearch-7.10.2-node1/config/elasticsearch.yml
```

192.168.116.140, 第一台节点配置内容：

```
# 集群名称  
cluster.name: my-application  
#节点名称  
node.name: node-1  
# 绑定IP地址  
network.host: 192.168.116.140  
# 指定服务访问端口  
http.port: 9200  
# 指定API端户端调用端口  
transport.tcp.port: 9300  
#集群通讯地址  
discovery.seed_hosts: ["192.168.116.140:9300",  
"192.168.116.140:9301", "192.168.116.140:9302"]  
#集群初始化能够参选的节点信息  
cluster.initial_master_nodes: ["192.168.116.140:9300",  
"192.168.116.140:9301", "192.168.116.140:9302"]  
#开启跨域访问支持， 默认为false  
http.cors.enabled: true  
##跨域访问允许的域名， 允许所有域名  
http.cors.allow-origin: "*"
```

修改目录权限：

```
chown -R elasticsearch:elasticsearch /usr/local/cluster/elasticsearch-7.10.2-node1
```

3. 复制ElasticSearch安装目录：

复制其余两个节点：

```
cd /usr/local/cluster  
cp -r elasticsearch-7.10.2-node1 elasticsearch-7.10.2-node2  
cp -r elasticsearch-7.10.2-node1 elasticsearch-7.10.2-node3
```

4. 修改其余节点的配置：

192.168.116.140 第二台节点配置内容：

```
# 集群名称  
cluster.name: my-application
```

```
#节点名称
node.name: node-2
# 绑定IP地址
network.host: 192.168.116.140
# 指定服务访问端口
http.port: 9201
# 指定API端户端调用端口
transport.tcp.port: 9301
#集群通讯地址
discovery.seed_hosts: ["192.168.116.140:9300",
"192.168.116.140:9301","192.168.116.140:9302"]
#集群初始化能够参选的节点信息
cluster.initial_master_nodes: ["192.168.116.140:9300",
"192.168.116.140:9301","192.168.116.140:9302"]
#开启跨域访问支持， 默认为false
http.cors.enabled: true
##跨域访问允许的域名， 允许所有域名
http.cors.allow-origin: "*"
```

192.168.116.140 第三台节点配置内容：

```
# 集群名称
cluster.name: my-application
#节点名称
node.name: node-3
# 绑定IP地址
network.host: 192.168.116.140
# 指定服务访问端口
http.port: 9202
# 指定API端户端调用端口
transport.tcp.port: 9302
#集群通讯地址
discovery.seed_hosts: ["192.168.116.140:9300",
"192.168.116.140:9301","192.168.116.140:9302"]
#集群初始化能够参选的节点信息
cluster.initial_master_nodes: ["192.168.116.140:9300",
"192.168.116.140:9301","192.168.116.140:9302"]
#开启跨域访问支持， 默认为false
http.cors.enabled: true
##跨域访问允许的域名， 允许所有域名
http.cors.allow-origin: "*"
```

## 5. 启动集群节点

先切换elasticsearch用户，在三台节点依次启动服务：

```
su elasticsearch
/usr/local/cluster/elasticsearch-7.10.2-node1/bin/elasticsearch -d
/usr/local/cluster/elasticsearch-7.10.2-node2/bin/elasticsearch -d
/usr/local/cluster/elasticsearch-7.10.2-node3/bin/elasticsearch -d
```

注意：如果启动出现错误，将各节点的数据目录清空，再重启服务。

## 6. 集群状态查看

集群安装与启动成功之后，执行请求：[http://192.168.116.140:9200/\\_cat/nodes?pretty](http://192.168.116.140:9200/_cat/nodes?pretty)

可以看到三个节点信息，三个节点会自行选举出主节点：

← → C ⌂ ▲ 不安全 | 192.168.116.140:9200/\_cat/nodes?pretty

```
192.168.116.140 46 96 6 3.20 4.01 1.94 cdhilmrstw * node-1
192.168.116.140 35 96 6 3.20 4.01 1.94 cdhilmrstw - node-2
192.168.116.140 56 96 6 3.20 4.01 1.94 cdhilmrstw - node-3
```

## 2.5 ElasticSearch集群分片测试

修改kibana的配置文件，指向创建的集群节点：

```
elasticsearch.hosts:  
["http://192.168.116.140:9200","http://192.168.116.140:9201","http://192.168.116.140:9202"]
```

重启kibana服务，进入控制台：

<http://192.168.116.140:5601/app/home#/>

再次创建索引（副本数量范围内）：

```
PUT orders
{
  "settings": {
    "index": {
      "number_of_shards": 2,
      "number_of_replicas": 2
    }
  }
}
```

可以看到，这次结果是正常：

## Index Management

 Index Manage

Indices Data Streams Index Templates Component Templates

Update your Elasticsearch indices individually or in bulk. [Learn more.](#)  

<input type="checkbox"/> Name	Health	Status	Primaries	Replicas	Docs count	Storage size	Data
<input type="checkbox"/> orders	<span style="color: green;">green</span>	open	2	2	0	1.2kb	

Rows per page: 10 

集群并非可以随意增加副本数量，创建索引（超出副本数量范围）：

```

PUT orders
{
  "settings": {
    "index": {
      "number_of_shards": 2,
      "number_of_replicas": 5
    }
  }
}

```

可以看到出现了yellow警告错误：

## Index Management

[Index Man...](#)

[Indices](#) [Data Streams](#) [Index Templates](#) [Component Templates](#)

Update your Elasticsearch indices individually or in bulk. [Learn more.](#)

[Include rollup indices](#)  [Includ...](#)

<input type="checkbox"/> Name	Health	Status	Primaries	Replicas	Docs count	Storage size
<input type="checkbox"/> orders	● yellow	open	2	5	0	1kb

Rows per page: 10

## 3. ELK部署应用与工作机制

### 3.1 ELK日志分析平台介绍

ELK是三个开源软件的缩写，分别表示：Elasticsearch , Logstash和Kibana。Elasticsearch和Kibana我们在上面做过讲解。Logstash主要是用来日志的搜集、分析、过滤日志的工具，适用大数据量场景，一般采用c/s模式，client端安装在需要收集日志的主机上，server端负责将收到的各节点日志进行过滤、修改等操作，再一并发往Elasticsearch上做数据分析。

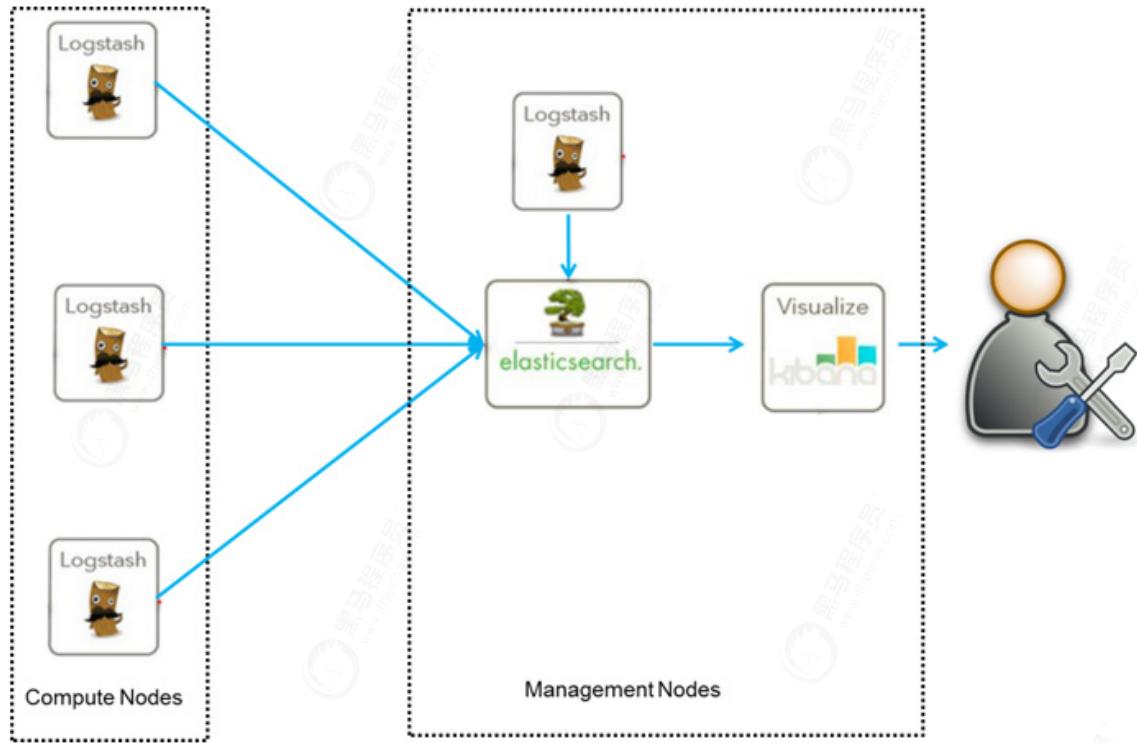
一个完整的集中式日志系统，需要包含以下几个主要特点：

- 收集 - 能够采集多种来源的日志数据
- 传输 - 能够稳定的把日志数据传输到中央系统
- 存储 - 如何存储日志数据
- 分析 - 可以支持UI分析
- 警告 - 能够提供错误报告，监控机制

ELK提供了一整套解决方案，并且都是开源软件，之间互相配合使用，完美衔接，高效的满足了很多场合的应用，是目前主流的一种日志分析平台。

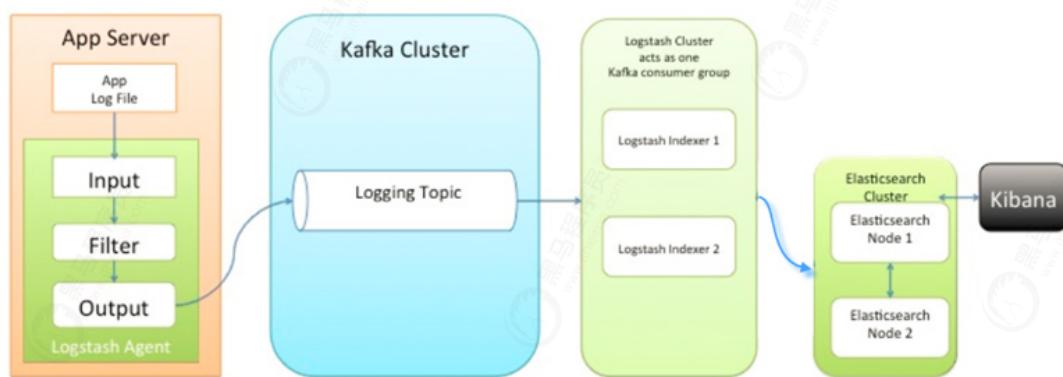
## 3.2 ELK部署架构模式

### 3.2.1 简单架构



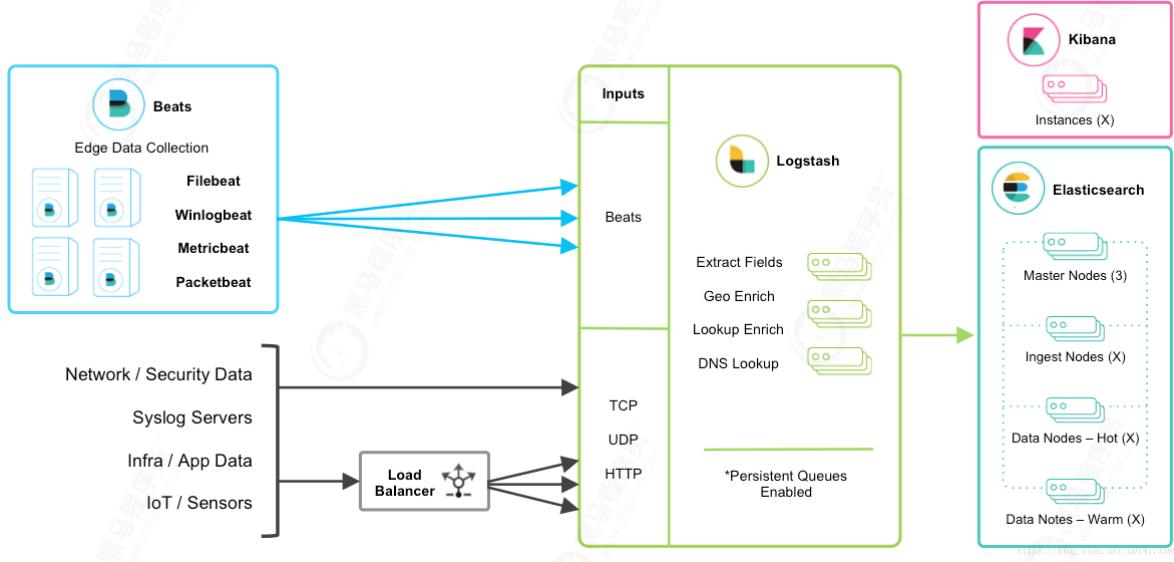
这是最简单的一种ELK部署架构方式，由Logstash分布于各个节点上搜集相关日志、数据，并经过分析、过滤后发送给远端服务器上的Elasticsearch进行存储。优点是搭建简单，易于上手，缺点是Logstash耗资源较大，依赖性强，没有消息队列缓存，存在数据丢失隐患。

### 3.2.2 消息队列架构



该队列架构引入了KAFKA消息队列，解决了各采集节点上Logstash资源耗费过大，数据丢失的问题，各终端节点上的Logstash Agent 先将数据/日志传递给Kafka，消息队列再将数据传递给Logstash，Logstash过滤、分析后将数据传递给Elasticsearch存储，由Kibana将日志和数据呈现给用户。

### 3.2.3 BEATS架构



该架构的终端节点采用Beats工具收集发送数据，更灵活，消耗资源更少，扩展性更强。同时可配置Logstash和Elasticsearch集群用于支持大集群系统的运维日志数据监控和查询，官方也推荐采用此工具，本章我们采用此架构模式进行配置讲解（如果在生产环境中，可以再增加kafka消息队列，实现了beats+消息队列的部署架构）。

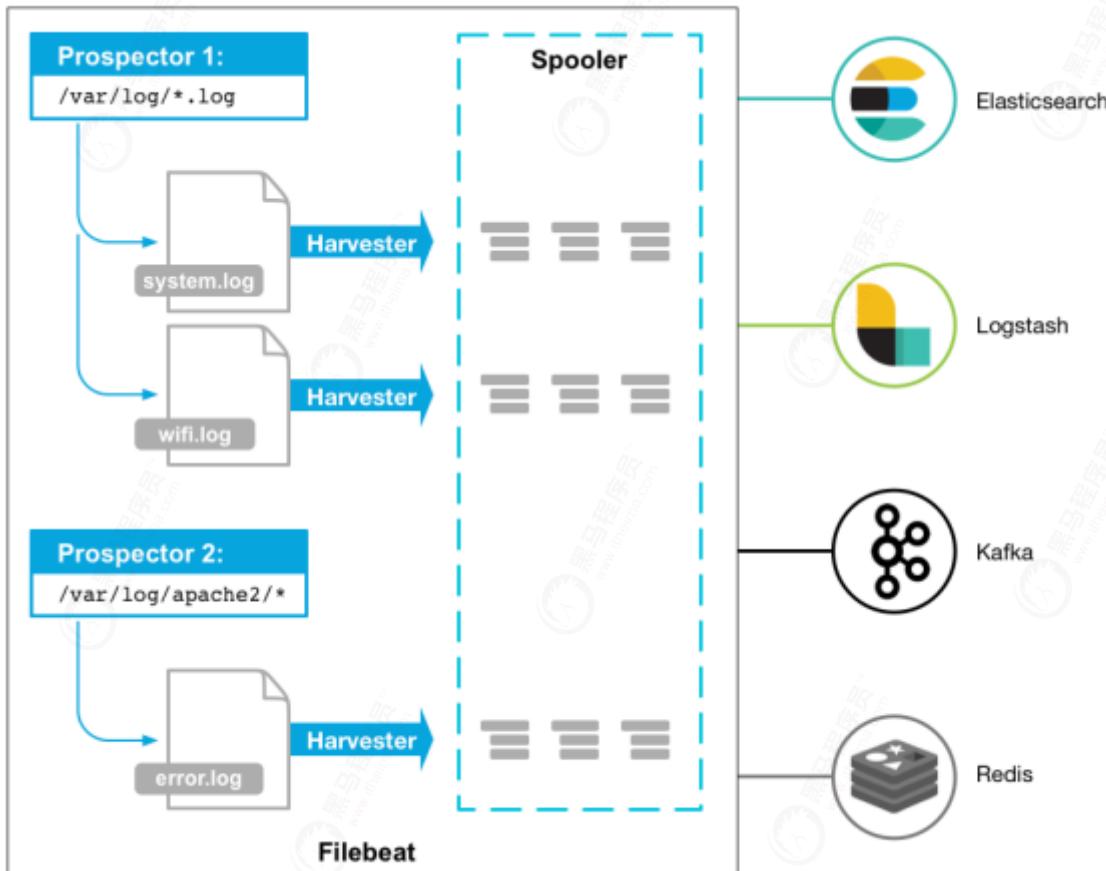
Beats工具包含四种：

- 1、Packetbeat (搜集网络流量数据)
- 2、Topbeat (搜集系统、进程和文件系统级别的CPU和内存使用情况等数据)
- 3、Filebeat (搜集文件数据)
- 4、Winlogbeat (搜集Windows事件日志数据)

## 3.3 ELK工作机制

### 3.3.1 Filebeat工作机制

Filebeat由两个主要组件组成：prospectors 和 harvesters。这两个组件协同工作将文件变动发送到指定的输出中。



**Harvester (收割机)**：负责读取单个文件内容。每个文件会启动一个Harvester，每个Harvester会逐行读取各个文件，并将文件内容发送到制定输出中。Harvester负责打开和关闭文件，意味在Harvester运行的时候，文件描述符处于打开状态，如果文件在收集中被重命名或者被删除，Filebeat会继续读取此文件。所以在Harvester关闭之前，磁盘不会被释放。默认情况filebeat会保持文件打开的状态，直到达到close\_inactive

filebeat会在指定时间内将不再更新的文件句柄关闭，时间从harvester读取最后一行的时间开始计时。若文件句柄被关闭后，文件发生变化，则会启动一个新的harvester。关闭文件句柄的时间不取决于文件的修改时间，若此参数配置不当，则可能发生日志不实时的情况，由scan\_frequency参数决定，默认10s。Harvester使用内部时间戳来记录文件最后被收集的时间。例如：设置5m，则在Harvester读取文件的最后一行之后，开始倒计时5分钟，若5分钟内文件无变化，则关闭文件句柄。默认5m】。

**Prospector (勘测者)**：负责管理Harvester并找到所有读取源。

```
filebeat.prospectors:
- input_type: log
  paths:
    - /apps/logs/*/*info.log
```

Prospector会找到/apps/logs/\*目录下的所有info.log文件，并为每个文件启动一个Harvester。Prospector会检查每个文件，看Harvester是否已经启动，是否需要启动，或者文件是否可以忽略。若Harvester关闭，只有在文件大小发生变化的时候Prospector才会执行检查。只能检测本地的文件。

**Filebeat如何记录发送状态：**

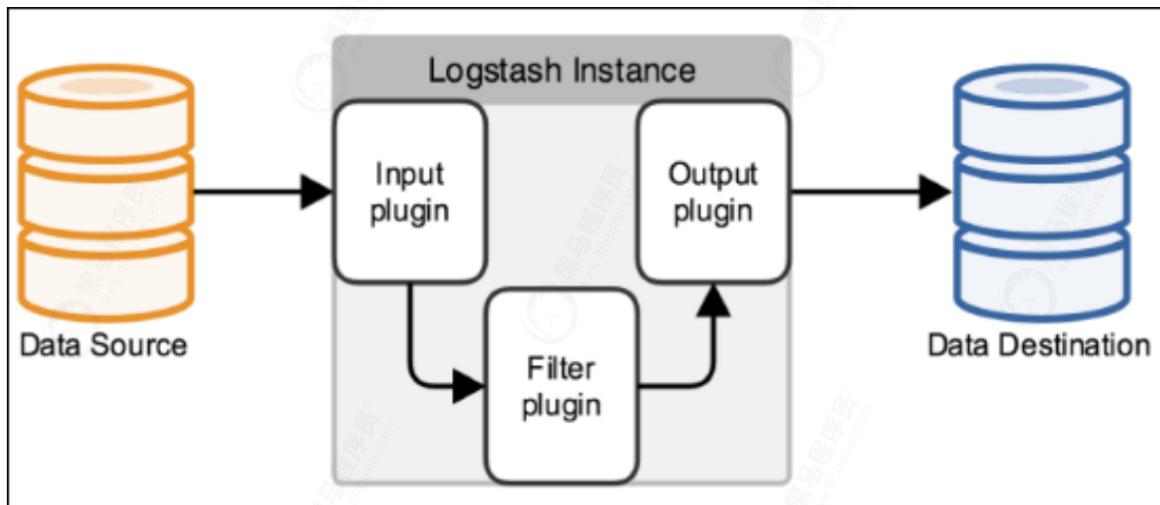
将文件状态记录在文件中（默认在/var/lib/filebeat/registry）。此状态可以记住Harvester收集文件的偏移量。若连接不上输出设备，如ES等，filebeat会记录发送前的最后一行，并再可以连接的时候继续发送。Filebeat在运行的时候，Prospector状态会被记录在内存中。Filebeat重启的时候，利用registry记录的状态来进行重建，用来还原到重启之前的状态。每个Prospector会为每个找到的文件记录一个状态，对于每个文件，Filebeat存储唯一标识符以检测文件是否先前被收集。

#### Filebeat如何保证数据发送成功：

Filebeat之所以能保证事件至少被传递到配置的输出一次，没有数据丢失，是因为filebeat将每个事件的传递状态保存在文件中。在未得到输出方确认时，filebeat会尝试一直发送，直到得到回应。若filebeat在传输过程中被关闭，则不会再关闭之前确认所有事件。任何在filebeat关闭之前未确认的事件，都会在filebeat重启之后重新发送。这可确保至少发送一次，但有可能会重复。可通过设置 shutdown\_timeout 参数来设置关闭之前的等待事件回应的时间（默认禁用）。

### 3.3.2 Logstash工作机制

Logstash事件处理有三个阶段：inputs → filters → outputs。是一个接收，处理，转发日志的工具。支持系统日志，webserver日志，错误日志，应用日志等。



#### Input：输入数据到logstash。

支持的输入类型：

file：从文件系统的文件中读取，类似于tail -f命令

syslog：在514端口上监听系统日志消息，并根据RFC3164标准进行解析

redis：从redis service中读取

beats：从filebeat中读取

#### Filters：数据中间处理，对数据进行操作。

一些常用的过滤器为：

grok：解析任意文本数据，Grok是Logstash最重要的插件。它的主要作用就是将文本格式的字符串，转换成为具体的结构化的数据，配合正则表达式使用。内置120多个解析语法。

[官方提供的grok表达式](#)

mutate：对字段进行转换。例如对字段进行删除、替换、修改、重命名等。

drop：丢弃一部分events不进行处理。

clone：拷贝 event，这个过程中也可以添加或移除字段。

geoip：添加地理信息(为前台kibana图形化展示使用)

### Outputs : outputs是logstash处理管道的最末端组件。

一个event可以在处理过程中经过多重输出，但是一旦所有的outputs都执行结束，这个event也就完成生命周期。

常见的outputs为：

elasticsearch：可以高效的保存数据，并且能够方便和简单的进行查询。

file：将event数据保存到文件中。

graphite：将event数据发送到图形化组件中，一个很流行的开源存储图形化展示的组件。

### Codecs : codecs 是基于数据流的过滤器，它可以作为input，output的一部分配置。

Codecs可以帮助你轻松的分割发送过来已经被序列化的数据。

常见的codecs：

json：使用json格式对数据进行编码/解码。

multiline：将多个事件中数据汇总为一个单一的行。比如：java异常信息和堆栈信息。

## 3.4 Logstash安装配置

在192.168.116.141机器节点上进行安装：

### 1. 下载解压

下载：

```
cd /usr/local  
wget https://artifacts.elastic.co/downloads/logstash/logstash-7.10.2-linux-x86_64.tar.gz
```

解压：

```
tar -xvf logstash-7.10.2-linux-x86_64.tar.gz
```

### 2. 创建数据存储与日志记录目录

```
[root@localhost logstash-7.10.2]# mkdir -p /usr/local/logstash-7.10.2/data  
[root@localhost logstash-7.10.2]# mkdir -p /usr/local/logstash-7.10.2/logs
```

### 3. 修改配置文件：

```
vi /usr/local/logstash-7.10.2/config/logstash.yml
```

配置内容：

```
# 数据存储路径
path.data: /usr/local/logstash-7.10.2/data
# 监听主机地址
http.host: "192.168.116.141"
# 日志存储路径
path.logs: /usr/local/logstash-7.10.2/logs
#启动监控插件
xpack.monitoring.enabled: true
#Elastic集群地址
xpack.monitoring.elasticsearch.hosts:
["http://192.168.116.140:9200","http://192.168.116.140:9201","http://192.168.116.140:9202"]
```

4. 创建监听配置文件：

```
vi /usr/local/logstash-7.10.2/config/logstash.conf
```

配置：

```
input {
  beats {
    # 监听端口
    port => 5044
  }
}

output {
  stdout {
    # 输出编码插件
    codec => rubydebug
  }

  elasticsearch {
    # 集群地址
    hosts =>
    ["http://192.168.116.140:9200","http://192.168.116.140:9201","http://192.168.116.140:9202"]
  }
}
```

5. 启动服务：

以root用户身份执行：

```
## 后台启动方式
nohup /usr/local/logstash-7.10.2/bin/logstash -f /usr/local/logstash-7.10.2/config/logstash.conf &
##
./logstash -f ../config/logstash.conf
```

成功启动后会显示以下日志：

```
[2020-10-15T06:57:40,640][INFO ][logstash.agent] successfully started Logstash API endpoint {:port=>9600}
```

访问地址：<http://192.168.116.141:9600/>，可以看到返回信息：

```
JSON
host : localhost
version : 7.10.2
http_address : 192.168.116.141:9600
id : 56473e6e-2e91-4e2d-9cc5-eecbe29b7f82
name : localhost
ephemeral_id : 26ac9f37-6b0f-4a1a-81fc-db014dbb9f98
status : green
snapshot : false
pipeline
  workers : 1
  batch_size : 125
  batch_delay : 50
monitoring
  hosts :
    http://192.168.116.140:9200
    http://192.168.116.140:9201
    http://192.168.116.140:9202
    username : logstash_system
  build_date : 2021-01-13T02:43:06Z
  build_sha : 7cebafee7a073fa9d58c97de074064a540d6c317
  build_snapshot : false
```

### 3.5 Filebeat安装配置

在192.168.116.141机器节点上操作：

#### 1. 下载解压

与ElasticSearch版本一致，下载7.10.2版本。

```
cd /usr/local
wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-7.10.2-
linux-x86_64.tar.gz
```

解压：

```
tar -xvf filebeat-7.10.2-linux-x86_64.tar.gz
```

#### 2. 修改配置文件

```
vi /usr/local/filebeat-7.10.2/filebeat.yml
```

修改内容：

```
# 需要收集发送的日志文件
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /var/log/messages
# 如果需要添加多个日志，只需要添加
- type: log
  enabled: true
  paths:
    - /var/log/test.log
# filebeat 配置模块， 可以加载多个配置
filebeat.config.modules:
  path: ${path.config}/modules.d/*.yml
  reload.enabled: false
# 索引分片数量设置
setup.template.settings:
  index.number_of_shards: 2
# kibana 信息配置
setup.kibana:
  host: "192.168.116.140:5601"
# logstash 信息配置（注意只能开启一项output设置， 如果采用logstash， 将
# output.elasticsearch关闭）
output.logstash:
  hosts: ["192.168.116.141:5044"]
# 附加metadata元数据信息
processors:
- add_host_metadata: ~
- add_cloud_metadata: ~
```

### 3. 启动服务

```
## 后台启动
nohup /usr/local/filebeat-7.10.2/filebeat -e -c /usr/local/filebeat-
7.10.2/filebeat.yml &
##
./filebeat -e -c filebeat.yml
```

启动成功后显示日志：

```

2020-12-15T07:09:33.922-0400      WARN    beater/filebeat.go:367 Filebeat is
unable to load the Ingest Node pipelines for the configured modules because
the Elasticsearch output is not configured/enabled. If you have already
loaded the Ingest Node pipelines or are using Logstash pipelines, you can
ignore this warning.
2020-12-15T07:09:33.922-0400      INFO    crawler/crawler.go:72 Loading
Inputs: 1
2020-12-15T07:09:33.923-0400      INFO    log/input.go:148 Configured
paths: [/var/log/messages]
2020-12-15T07:09:33.923-0400      INFO    input/input.go:114 Starting
input of type: log; ID: 14056778875720462600
2020-12-15T07:09:33.924-0400      INFO    crawler/crawler.go:106 Loading and
starting Inputs completed. Enabled inputs: 1
2020-12-15T07:09:33.924-0400      INFO    cfgfile/reload.go:150 Config
reloader started

```

我们监听的是/var/log/messages系统日志信息，当日志发生变化后，filebeat会通过logstash上报到Elasticsearch中。我们可以查看下集群的全部索引信息：

[http://192.168.116.140:9200/\\_cat/indices?v](http://192.168.116.140:9200/_cat/indices?v)

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
green	open	logstash-2021.07.20-000001	P8PvC3kcQxS0q8WFU18FA	1	1	0	0	416b	208b
green	open	.apm-custom-link	D11oJWyZRHKcpnYUlpekBg	1	1	0	0	416b	208b
green	open	.kibana_task_manager_1	115AvBElRcqK0FxqekZ10w	1	1	5	190	292.3kb	133.3kb
green	open	.apm-agent-configuration	OKPqgkRpTKWFWi7Cc3tgsA	1	1	0	0	416b	208b
yellow	open	orders	WtS6exYVTSeWwThHjgpcDg	2	5	0	0	1.2kb	416b
green	open	.async-search	U-m_u4LPQY0ommX60xbwriA	1	1	0	2	6.8kb	3.4kb
green	open	.kibana-event-log-7.10.2-000001	e8a402f0QsWpaR0qmA21A	1	1	4	0	43.9kb	21.9kb
green	open	.kibana_1	KcNf2dMCR9uTZUs2Cz3lRQ	1	1	80	0	4.2mb	2.1mb
green	open	.kibana-event-log-7.10.2-000002	6BzWB_ZyTR6MIVql9jeIGg	1	1	0	0	416b	208b

可以看到，已经生成了名为logstash-2021.07.20-000001索引。

## 3.6 Kibana配置与查看数据

1. 进入Kibana后台，进行配置：

<http://192.168.116.140:5601>

The screenshot shows the Kibana Management interface at the URL [http://10.10.20.28:5601/app/kibana#/management/kibana/index\\_pattern\\_g=0](http://10.10.20.28:5601/app/kibana#/management/kibana/index_pattern_g=0). On the left, there's a sidebar with various management options like Discover, Visualize, Dashboard, etc. The main area is titled 'Create index pattern' with the sub-section 'Step 1 of 2: Define index pattern'. It shows the index pattern 'logstash-\*' entered into a text field. A note below says 'You can use a \* as a wildcard in your index pattern. You can't use spaces or the characters \, /, ?, <, >, [, ].' Below the pattern, it says 'Success! Your index pattern matches 1 index.' followed by 'logstash-2019.10.15'. At the bottom right, there's a 'Next step' button.

进入【Management】-->在Index Pattern中输入"logstash-\*" -->点击【next step】，选择"@timestamp"，

点击【Create index pattern】进行创建。

## 2. 查看数据

进入【Discover】，可以查看到收集的数据：

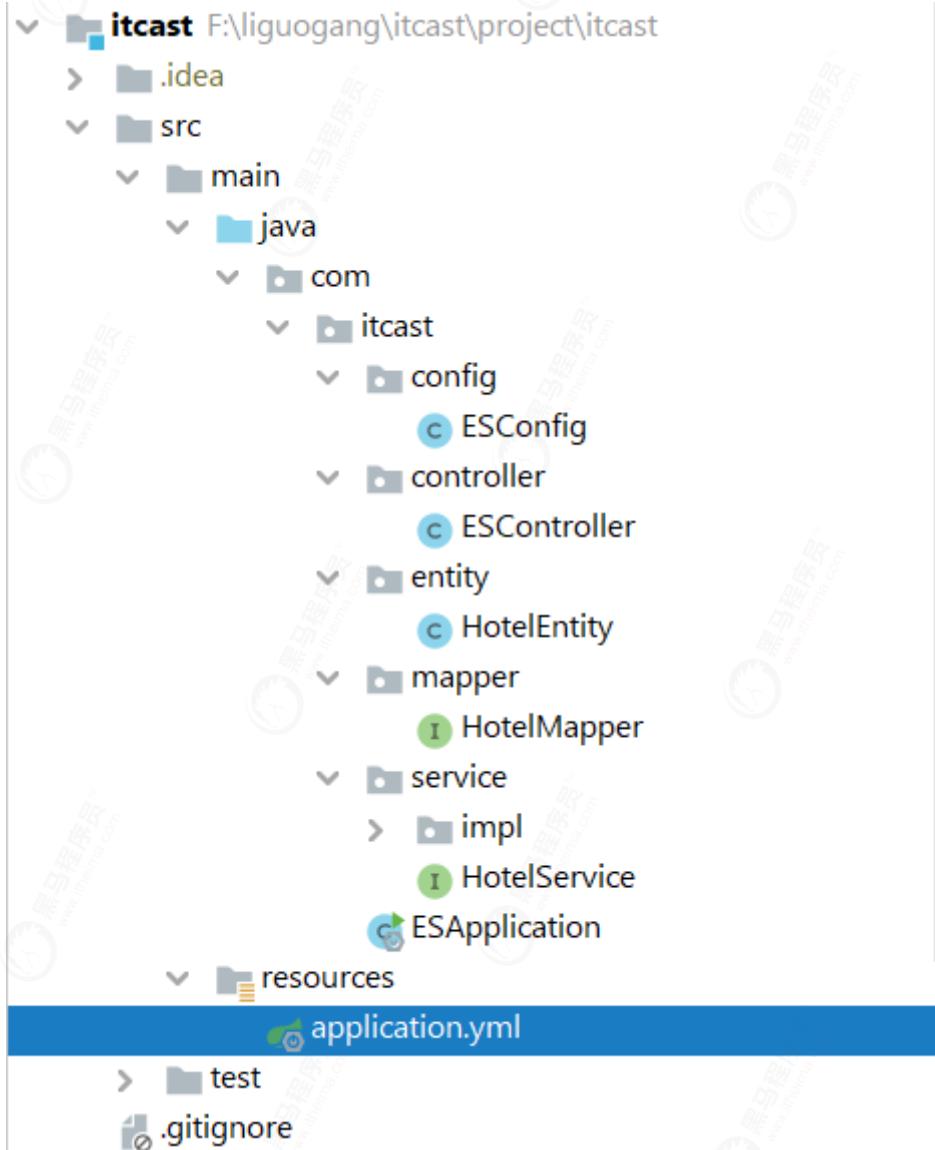
如果没有显示，可以重新调整Time Range时间范围。

## 4. ElasticSearch高阶操作

## 4.1 准备数据

1. 使用es.sql文件，完成初始化库表创建与数据准备

2. 导入工程



ESConfig连接配置类：

```
@Bean
public RestHighLevelClient restHighLevelClient(){
    RestClientBuilder builder = RestClient.builder(new HttpHost(host,
    port, "http"));
    builder.setRequestConfigCallback(requestConfigBuilder ->{
        requestConfigBuilder.setConnectionRequestTimeout(500000);
        requestConfigBuilder.setSocketTimeout(500000);
        requestConfigBuilder.setConnectTimeout(500000);
        return requestConfigBuilder;
    });
    return new RestHighLevelClient(builder);
}
```

ESController提供导入接口：

```
//批量导入
@Override
```

```
public int addDocToES() {  
  
    BulkRequest bulkRequest = new BulkRequest();  
  
    Querywrapper<HotelEntity> wrapper = new Querywrapper<>();  
    wrapper.last("LIMIT 10000");  
    List<HotelEntity> hotelEntityList = hotelMapper.selectList(wrapper);  
  
    for (HotelEntity hotelEntity : hotelEntityList) {  
  
        String data = JSON.toJSONStringwithDateFormat(hotelEntity, "yyyy-MM-dd", SerializerFeature.WriteDateUseDateFormat);  
        IndexRequest indexRequest = new IndexRequest("hotel").source(data, XContentType.JSON);  
        bulkRequest.add(indexRequest);  
    }  
  
    try {  
        BulkResponse response = restHighLevelClient.bulk(bulkRequest,  
RequestOptions.DEFAULT);  
  
        return response.status().getStatus();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
  
    return 0;  
}
```

### 3. 通过kibana创建索引结构

```
PUT hotel  
{  
  "settings": {  
    "number_of_shards": 1,  
    "number_of_replicas": 0  
  },  
  "mappings": {  
    "properties": {  
  
      "name":{  
        "type": "text"  
      },  
      "address":{  
        "type": "text"  
      },  
      "brand":{  
        "type": "keyword"  
      },  
      "type":{  
        "type": "keyword"  
      },  
      "price":{  
        "type": "integer"  
      },  
    }  
  }  
}
```

```
        "specs": {
            "type": "keyword"
        },
        "salesVolume": {
            "type": "integer"
        },
        "area": {
            "type": "text"
        },
        "imageUrl": {
            "type": "text"
        },
        "synopsis": {
            "type": "text"
        },
        "createTime": {
            "type": "date",
            "format": "yyyy-MM-dd"
        },
        "isAd": {
            "type": "integer"
        }
    }
}
```

## 4. 导入数据

调用接口：<http://127.0.0.1:8081/importData>

会读取数据库，自动导入数据到es。

## 5. 查看导入结果

通过kibana后台，查看导入的数据：

Index Management							<a href="#">Index Management docs</a>
Indices		Data Streams		Index Templates		Component Templates	
Update your Elasticsearch indices individually or in bulk. <a href="#">Learn more.</a>							
<input type="checkbox"/>	Search	<input type="checkbox"/> <a href="#">Include rollup indices</a>	<input type="checkbox"/> <a href="#">Include hidden indices</a>	Lifecycle status	Lifecycle phase	<a href="#">Reload indices</a>	
Name	Health	Status	Primaries	Replicas	Docs count	Storage size	Data stream
hotel	● green	open	1	0	9300	1.2mb	

4.2 基础查询

## 1. 查询所有酒店

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

## 2. 分页查询酒店列表

```
GET hotel/_search
{
  "query": {
    "match_all": {}
  },
  "from": 0,
  "size": 5
}
```

## 3. 酒店品牌模糊搜索

wildcard : 会对查询条件进行分词。还可以使用通配符 ? (任意单个字符) 和 \* (0个或多个字符)

```
GET hotel/_search
{
  "query": {
    "wildcard": {
      "brand": {
        "value": "美*"
      }
    }
  }
}
```

## 4. 品牌精确搜索 :

展示出"万豪"品牌下的所有酒店信息

term : 不会对查询条件进行分词

```
GET hotel/_search
{
  "query": {
    "term": {
      "brand": "万豪"
    }
  }
}
```

可以看到没有匹配到任何结果，因为term是拿整个词“万豪”进行匹配，而ES默认保存数据是做单字分词，将“万豪”划分为了“万”和“豪”，所以匹配不到结果。

## 4.3 bool查询

1. should查询：只要其中一个为true则成立。

```
# should 品牌是万豪 或者 specs是五星级
GET hotel/_search
{
  "query": {
    "bool": {
      "should": [
        {
          "term": {
            "brand": {
              "value": "万豪"
            }
          }
        },
        {
          "term": {
            "specs": {
              "value": "五星级"
            }
          }
        }
      ]
    },
    "from": 0,
    "size": 100
  }
}
```

2. must查询：必须所有条件都成立。

```
# must 品牌必须是万豪并且specs是五星级
GET hotel/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "term": {
            "brand": {
              "value": "万豪"
            }
          }
        },
        {
          "term": {
            "specs": {
              "value": "五星级"
            }
          }
        }
      ]
    }
  }
}
```

```
        ]
    }
}
```

### 3. must\_not查询：必须所有条件都不成立。

```
# must_not 品牌必须不是万豪并且specs也不是五星级
GET hotel/_search
{
  "query": {
    "bool": {
      "must_not": [
        {
          "term": {
            "brand": {
              "value": "万豪"
            }
          }
        },
        {
          "term": {
            "specs": {
              "value": "五星级"
            }
          }
        }
      ]
    }
  }
}
```

### 4. filter过滤查询：

查询品牌为万豪下的酒店

```
GET hotel/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "brand": {
              "value": "万豪"
            }
          }
        }
      ]
    }
  }
}
```

## 4.4 聚合查询操作

统计品牌为万豪的最贵酒店价格，max、min、sum等等。

```
#统计品牌为万豪的最贵酒店价格
GET hotel/_search
{
  "query": {
    "term": {
      "brand": {
        "value": "万豪"
      }
    }
  },
  "aggs": {
    "max_price": {
      "max": {
        "field": "price"
      }
    }
  }
}
```

可以看到，最后会有分组统计的汇总输出。

## 4.5 分词查询操作

查询出了很多万豪相关的酒店，现在以北京市东城区万豪酒店 查询name域，可以发现无法查询到结果。

```
GET hotel/_search
{
  "query": {
    "term": {
      "name": "北京市东城区万豪酒店"
    }
  }
}
```

在创建索引时，对于name域，数据类型是 text。当添加文档时，对于该域的值会进行分词，形成若干 term（词条）存储在倒排索引中。

根据倒排索引结构，当查询条件在词条中存在，则会查询到数据。如果词条中没有，则查询不到数据。

那么对于 北京市东城区万豪酒店 的分词结果是什么呢？

```
GET _analyze
{
  "text": "北京市东城区万豪酒店"
}
```

此时可以发现，每个字形成了一个词，所以并没有找到相匹配的词，导致无法查询到结果

在ElasticSearch默认内置了多种分词器：

- **Standard Analyzer** - 默认分词器，按英文空格切分
- Simple Analyzer - 按照非字母切分(符号被过滤)
- Stop Analyzer - 小写处理，停用词过滤(the,a,is)
- Whitespace Analyzer - 按照空格切分，不转小写
- Keyword Analyzer - 不分词，直接将输入当作输出
- Patter Analyzer - 正则表达式，默认\W+(非字符分割)

而我们想要的是，分词器能够智能的将中文按照词义分成若干个有效的词。此时就需要额外安装中文分词器。对于中文分词器的类型也有很多，其中首选的是：IK分词器。

## 4.6 IK分词器

### 1. 安装IK分词插件

[下载地址](#)

### 2. 执行安装

采用本地文件安装方式，进入ES安装目录，执行插件安装命令：

```
[elasticsearch@localhost plugins]$ ./bin/elasticsearch-plugin install  
file:///usr/local/elasticsearch-7.10.2/elasticsearch-analysis-ik-7.10.2.zip
```

安装成功后，会给出对应提示：

```
-> Installing file:///usr/local/elasticsearch-7.10.2/elasticsearch-analysis-  
ik-7.10.2.zip  
-> Downloading file:///usr/local/elasticsearch-7.10.2/elasticsearch-  
analysis-ik-7.10.2.zip  
[=====] 100%  
@     WARNING: plugin requires additional permissions @  
* java.net.SocketPermission * connect,resolve  
See  
http://docs.oracle.com/javase/8/docs/technotes/guides/security/permissions.html  
for descriptions of what these permissions allow and the associated risks.  
  
Continue with installation? [y/N]y  
-> Installed analysis-ik
```

### 3. 重启ElasticSearch服务

### 4. 测试IK分词器

标准分词器：

```
GET _analyze  
{  
  "analyzer": "standard",  
  "text": "北京市东城区万豪酒店"  
}
```

采用IK智能化分词器：

```
GET _analyze
{
  "analyzer": "ik_smart",
  "text": "北京市东城区万豪酒店"
}
```

IK最大化分词：

```
GET _analyze
{
  "analyzer": "ik_max_word",
  "text": "北京市东城区万豪酒店"
}
```

## 5. IK分词器最佳运用

analyzer指定的是构建索引的分词，search\_analyzer指定的是搜索关键字的分词。

实践运用的时候，构建索引的时候采用max\_word，将分词最大化；查询的时候则使用smartword智能化分词，这样能够最大程度的匹配出结果。

```
PUT hotel
{
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 0
  },
  "mappings": {
    "properties": {
      "name": {
        "type": "text",
        "analyzer": "ik_max_word"
      },
      "address": {
        "type": "text",
        "analyzer": "ik_max_word"
      },
      "brand": {
        "type": "keyword"
      },
      "type": {
        "type": "keyword"
      },
      "price": {
        "type": "integer"
      },
      "specs": {
        "type": "keyword"
      },
      "salesVolume": {
        "type": "integer"
      }
    }
  }
}
```

```
"area":{  
    "type": "text",  
    "analyzer": "ik_max_word"  
},  
"imageUrl":{  
    "type": "text"  
},  
"synopsis":{  
    "type": "text",  
    "analyzer": "ik_max_word"  
},  
"createTime":{  
    "type": "date",  
    "format": "yyyy-MM-dd"  
},  
"isAd":{  
    "type": "integer"  
}  
}  
}  
}
```

## 4.7 搜索匹配进阶

### 1. or关系

```
GET hotel/_search  
{  
    "query": {  
        "match": {  
            "name": "金龙 金辉"  
        }  
    },  
    "from": 0,  
    "size": 200  
}
```

match搜索实质上就是or关系，分为“金龙”和“金辉”两个关键词进行or关系搜索。

### 2. or关系最小词匹配

```
GET hotel/_search  
{  
    "query":{  
        "match":{  
            "name": {  
                "query": "金龙 金辉",  
                "operator": "or",  
                "minimum_should_match": 2  
            }  
        }  
    }  
}
```

这里minimum\_should\_match设定为2，只要匹配两个字符，即出现“金龙”和“金辉”，那么这样的数据都会展示出来。

### 3. and关系

```
GET hotel/_search
{
  "query": {
    "match": {
      "name": {
        "query": "金龙 金辉",
        "operator": "and"
      }
    }
  }
}
```

通过operator属性来标识对应的操作。这个时候搜索出来的name会包含“金龙”和“金辉”两个关键字。

### 4. 短语查询

如果想直接搜索某个短语，比如：金龙 金辉，可以采用match\_phrase

```
GET hotel/_search
{
  "query": {
    "match_phrase": {
      "name": "金龙 金辉"
    }
  }
}
```

会做整个短语的完整匹配，不会再进行拆分匹配。

### 5. 多字段查询

如果想对多个字段同时查询，可以采用multi\_match方式。

```
GET hotel/_search
{
  "query": {
    "multi_match": {
      "query": "如心 天津市",
      "fields": ["name", "address"]
    }
  },
  "from": 0,
  "size": 200
}
```

查询name和address两个属性，都包含“如心 天津市”的记录，相比一个属性name的查询，多出更多的记录。

## 4.8 Query String查询

可以采用更简便的方式，直接使用AND、OR和NOT操作。

```
GET hotel/_search
{
  "query": {
    "query_string": {
      "fields": ["name"],
      "query": "如心 AND 天津市"
    }
  }
}
```

查出name当中既包含"如心"又包含"天津市"的数据。

## 5. 倒排索引

### 5.1 倒排索引

要想理解倒排索引，首先先思考一个问题，获取某个文件夹下所有文件名中包含Spring的文件

- 1) 确定要搜索的文件夹
- 2) 遍历文件夹下所有文件
- 3) 判断文件名中是否包含Spring

这种思维可以理解为是一种正向思维的方式，从外往内，根据key找value。这种方式可以理解为正向索引。

而ElasticSearch为了提升查询效率，采用反向思维方式，根据value找key。



### 5.2 评分TF/IDF/BM25计算

每条搜索记录ES都会给出一个评分，ES有两个打分计算方式：

1. **TF:** Term Frequency，即词频。它表示一个词在内容中出现的次数。定义：

$$TF = \frac{\text{某个词在文档中出现的次数}}{\text{文档的总词数}}$$

某个词出现越多，表示越重要，如果某篇文章出现了elasticsearch多次，而spring出现了两三次，那很可能就是一篇关于elasticsearch的专业文章。

2. **IDF**: Inverse Document Frequency，即逆文档频率，它是一个表达词语重要性的指标。计算公式：

$$\text{IDF} = \log(\text{库中的文档数} / (\text{包含该词的文档数} + 1))$$

log: 对数

log为对数函数，如果所有文章内容都包涵某一个词，那这个词的IDF = log(1)=0，重要性为零。停用词的IDF约等于0。

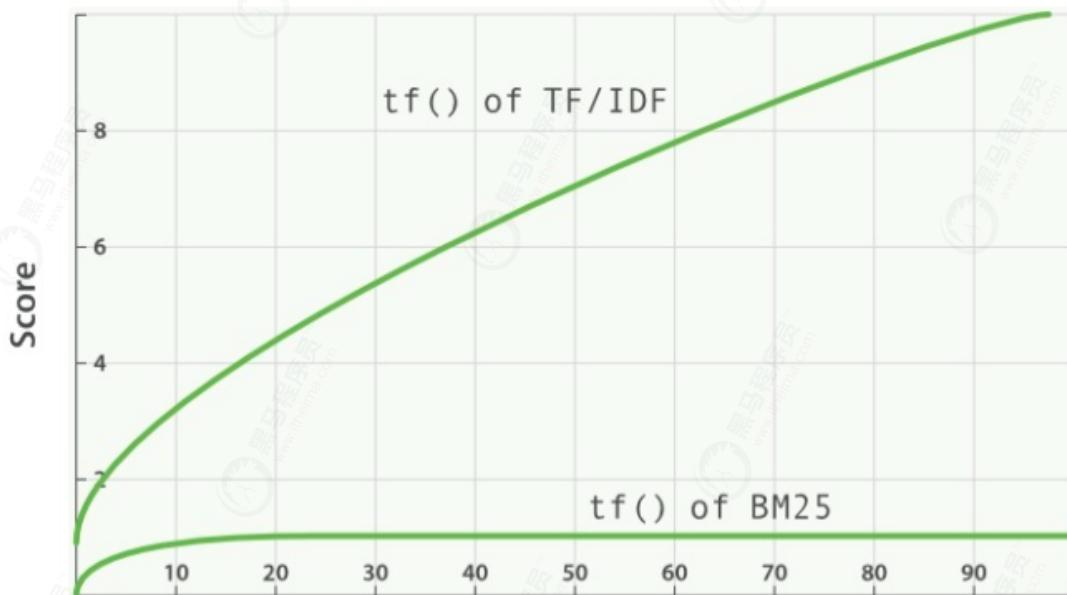
如果某个词只在很少的文章中出现，则IDF很大，其重要性也越高。为了避免分母为0，所以 + 1.

### 3. BM25

BM25 实质是对 TF-IDF 算法的改进，对于 TF-IDF 算法，TF(t) 部分的值越大，整个公式返回的值就会越大。

随着TF(t) 的逐步加大，该算法的返回值会趋于一个数值，BM25 就针对这点进行来优化。

例如，某个文章的关键词出现的频率不断增多，得分就会越来越高，有的文章关键词出现40次，和有的文章关键词出现60次或80次，但实际上出现40次的文章，可能就是所期望的结果。



### 4. 查看ES评分计算：

增加explain标识为true，会列出计算执行计划：

```
GET hotel/_search
{
  "explain": true,
  "query":{
    "match":{
      "name": "北京市东城区七天酒店"
    }
  }
}
```

里面会详细记录评分细则：

```
39 |     "explanation" : {  
40 |         "value" : 17.39165,  
41 |         "description" : "sum of:",  
42 |         "details" : [  
43 |             {  
44 |                 "value" : 1.7387722,  
45 |                 "description" : "weight(name:北 in 0) [PerFieldSimilarity],  
46 |                     result of:",  
47 |                 "details" : [  
48 |                     {  
49 |                         "value" : 1.7387722,  
50 |                         "description" : "score(freq=1.0), computed as boost * idf  
51 |                             * tf from:",  
52 |                         "details" : [  
53 |                             {  
54 |                                 "label" : "2.0
```

整个评分计算： boost \* idf \* tf ( boost为放大系数， 默认为2.2 )

BM25的计算在tf的描述中：(freq + k1 \* (1 - b + b \* dl / avgdl))

## 6. ElasticSearch实战技巧

## 6.1 优化多字段查询

## 1. 提升字段查询得分：

将name字段查询比重提升10倍：

```
GET hotel/_search
{
  "explain": true,
  "query": {
    "multi_match": {
      "query": "北京市酒店",
      "fields": ["name^10", "address"]
    }
  }
}
```

得分系数提升了10倍：

```
10    "hits" : {
11      "total" : {
12        "value" : 9300,
13        "relation" : "eq"
14      },
15      "max_score" : 70.12234,
16      "hits" : [
17        {
18          "_shard" : "[hotel][0]",
19          "_node" : "tKJGjK3kS_eD_eBNZIR2zg",
20          "_index" : "hotel",
21          "_type" : "_doc",
22          "_id" : "f5k5xHoBeWi2Spj0rVkk",
23          "_score" : 70.12234,
24          "_source" : {
25            "address" : "天津市蓟 县七天酒店",
26            "area" : "天津市",
27            "brand" : "七天"
```

## 2. 综合提升字段查询得分：

使用tie\_breaker将其他query的分数也考虑进去

```
GET hotel/_search
{
  "explain": true,
  "query": {
    "multi_match": {
      "query": "北京市酒店",
      "fields": ["name", "address"],
      "tie_breaker": 0.3
    }
  }
}
```

使用 tie\_breaker 和不使用tie\_breaker , 查询出来的某一条数据的 \_score 分数 , 会有相应的提高 , 例如 :

name中包含关键词matched query 的得分 , 假设是 0.1984226

address中包含关键词matched query的得分 , 假设是 12.07466

添加了 tie\_breaker = 0.3 , 那么就是这样的了 ,  $0.1984226 * 0.3 + 12.07466 = 12.13418678$  ;

大于最高一条的得分12.07466 , 这样搜索的关联性就提升上去了 , 更为合理。

## 3. 自定义评分 :

通过function\_score实现自定义评分 :

query中的内容为主查询条件 , functions中为判断要为哪些数据加权。 weight为加权值。

#为品牌为万豪的酒店, 权重值增加50倍

```
GET hotel/_search
{
  "query": {
```

```
"function_score": {
    "query": {
        "query_string": {
            "fields": ["name", "area", "address"],
            "query": "北京市spa三星"
        }
    },
    "functions": [
        {
            "filter": {
                "term": {
                    "brand": "万豪"
                }
            },
            "weight": 50
        }
    ]
}
```

## 6.2 多条件查询

### 1. 多条件查询：

根据用户输入的日期，统计某品牌下所有酒店销量。对于该功能的实现，需要进行多层聚合。

1. 对日期时间段范围查询
2. 根据品牌进行分组查询
3. 对分组查询结果进行sum聚合

```
POST hotel/_search
{
    "query": {
        "range": {
            "createTime": {
                "gte": "2015-01-01",
                "lte": "2021-01-01",
                "format": "yyyy-MM-dd"
            }
        }
    },
    "aggs": {
        "hotel-brand": {
            "terms": {
                "field": "brand",
                "size": 100
            },
            "aggs": {
                "countSale": {
                    "sum": {
                        "field": "salesvolume"
                    }
                }
            }
        }
    }
}
```

```
        }
    }
}
```

## 2. 增加排序处理：

```
GET hotel/_search
{
  "query": {
    "range": {
      "createTime": {
        "gte": "2015-01-01",
        "lte": "2021-01-01",
        "format": "yyyy-MM-dd"
      }
    }
  },
  "aggs": {
    "hotel-brand": {
      "terms": {
        "field": "brand",
        "size": 100
      },
      "aggs": {
        "countSale": {
          "sum": {
            "field": "salesVolume"
          }
        }
      }
    }
  },
  "sort": [
    {
      "price": {
        "order": "desc"
      }
    }
  ]
}
```

根据price进行倒序排列。

## 6.3 查全率与查准率

### 1. 查全率：

索引内符合条件的结果有N个，查询出来的符合条件的结果有X个，则查全率为： $X/N$

比如：用户的关键词为笔记本（笔记本包含写字的笔记本以及电脑笔记本，在索引中，这些记录为1000条，即N），查询出来的结果如果是100条，即X（包含写字的笔记本以及电脑笔记本），则查全率为10%。

### 2. 查准率：

查询出来的X个文档中，有M个是正确的，则查准率为： $M/X$

比如：用户的关键词为笔记本，这些记录为1000条，查询出来的结果如果是100条，而在这100条(X)当中只有20条(M)为用户期望的电脑笔记本，则查准率为20%。

## 6.4 全量索引构建

1. 下载logstash

[下载地址](#)

2. 安装logstash-input-jdbc插件

进入logstash目录，执行：

```
bin/logstash-plugin install logstash-input-jdbc
```

3. 配置mysql驱动包

```
[root@localhost bin]# mkdir mysql  
[root@localhost bin]# cp mysql-connector-java-5.1.34.jar  
/usr/local/logstash-7.10.2/bin/mysql/
```

4. 配置JDBC连接

创建索引数据是从mysql中通过select语句查询，然后再通过logstash-input-jdbc的配置文件方式导入

elasticsearch中。

在/usr/local/logstash-7.10.2/bin/mysql/目录创建jdbc.conf与jdbc.sql文件。

jdbc.conf文件：

```
input {  
    stdin {  
    }  
    jdbc {  
        # mysql 数据库链接,users为数据库名  
        jdbc_connection_string => "jdbc:mysql://127.0.0.1:3306/esdb"  
        # 用户名和密码  
        jdbc_user => "root"  
        jdbc_password => "123456"  
        # 驱动  
        jdbc_driver_library => "/usr/local/logstash-7.10.2/bin/mysql/mysql-  
        connector-java-5.1.34.jar"  
        # 驱动类名  
        jdbc_driver_class => "com.mysql.jdbc.Driver"  
        jdbc_paging_enabled => "true"  
        jdbc_page_size => "50000"  
        # 执行的sql 文件路径+名称  
        statement_filepath => "/usr/local/logstash-  
        7.10.2/bin/mysql/jdbc.sql"  
        # 设置监听间隔 各字段含义（由左至右）分、时、天、月、年，全部为*默认含义为每分钟  
        都更新  
        schedule => "* * * * *"  
    }  
}
```

```
output {
    elasticsearch {
        #ES的连接信息
        hosts => ["192.168.116.140:9200"]
        #索引名称
        index => "hotel"
        document_type => "_doc"
        #自增ID， 需要关联的数据库的ID字段， 对应索引的ID标识
        document_id => "%{id}"
    }
    stdout {
        #JSON格式输出
        codec => json_lines
    }
}
```

jdbc.sql文件：

```
SELECT
    id,
    NAME,
    address,
    brand,
    type,
    price,
    specs,
    salesVolume,
    synopsis,
    area,
    imageUrl,
    createTime,
    isAd
FROM
    t_hotel
```

## 5. 执行全量同步

执行命令：

```
./logstash -f mysql/jdbc.conf
```

检查结果：

```
GET hotel/_search
```

## 6.5 增量索引同步

### 1. 修改jdbc.conf配置文件：

增加配置：

```
input {
    jdbc{
        #设置timezone
        jdbc_default_timezone => "Asia/Shanghai"
        ...
        # 增量同步属性标识
        last_run_metadata_path => "/usr/local/logstash-7.10.2/bin/mysql/last_value"
    }
}
```

## 2. 修改jdbc.sql配置文件

这里根据最后更新时间，做增量同步：

```
SELECT
    id,
    NAME,
    address,
    brand,
    type,
    price,
    specs,
    salesVolume,
    synopsis,
    area,
    imageUrl,
    createTime,
    isAd
FROM
    t_hotel
WHERE
    createTime >= :sql_last_value
```

## 3. 创建同步最后记录时间

```
vi /usr/local/logstash-7.10.2/bin/mysql/last_value
```

给定一个初始的时间：

```
2021-12-30 00:00:00
```

## 4. 验证

启动logstash，会根据初始时间，加载对应的数据。

如果修改的数据的更新时间，会自动检测，同步增量的数据。

