一、Elasticsearch的安装

1. 下载
下载地址：https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-5.6.4.zip

2. 解压

```
$ unzip elasticsearch-5.6.4.zip -d /home/hadoop/app/
```

3. 修改配置文件
vim ./config/elasticsearch.yml

```
cluster.name: myes-cluster
node.name: spark1234
# Path to directory where to store the data (separate multiple locations by comma):
path.data: /home/hadoop/data/es/data
# Path to log files:
path.logs: /home/hadoop/data/es/logs

# 配置主机的ip地址
network.host: 0.0.0.0

node.master: true
node.data: true
discovery.zen.ping.unicast.hosts: ["192.168.9.13:9300"]
discovery.zen.minimum_master_nodes: 1
```

4. 启动服务
启动Elasticsearch：./bin/elasticsearch
后台进程启动：./bin/elasticsearch -d

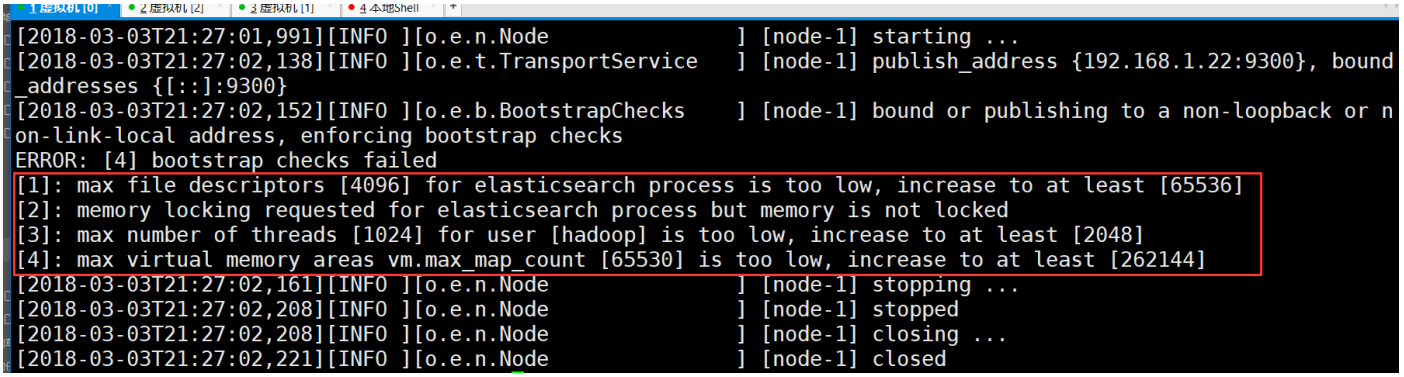
5. 错误解决

错误1：

```
[2018-02-28T11:48:01,915][WARN ][o.e.b.JNANatives         ] unable to install syscall filter:
java.lang.UnsupportedOperationException: seccomp unavailable: requires kernel 3.5+ with CONFIG_SECCOMP and CONFIG_SECCOMP_FILTER
compiled in
    at org.elasticsearch.bootstrap.SystemCallFilter.linuxImpl(SystemCallFilter.java:329) ~[elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.bootstrap.SystemCallFilter.init(SystemCallFilter.java:617) ~[elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.bootstrap.JNANatives.tryInstallSystemCallFilter(JNANatives.java:258) [elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.bootstrap.Natives.tryInstallSystemCallFilter(Natives.java:113) [elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.bootstrap.Bootstrap.initializeNatives(Bootstrap.java:111) [elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.bootstrap.Bootstrap.setup(Bootstrap.java:195) [elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.bootstrap.Bootstrap.init(Bootstrap.java:342) [elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.bootstrap.Elasticsearch.init(Elasticsearch.java:132) [elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.bootstrap.Elasticsearch.execute(Elasticsearch.java:123) [elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.cli.EnvironmentAwareCommand.execute(EnvironmentAwareCommand.java:70) [elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.cli.Command.mainWithoutErrorHandling(Command.java:134) [elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.cli.Command.main(Command.java:90) [elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.bootstrap.Elasticsearch.main(Elasticsearch.java:91) [elasticsearch-5.6.4.jar:5.6.4]
    at org.elasticsearch.bootstrap.Elasticsearch.main(Elasticsearch.java:84) [elasticsearch-5.6.4.jar:5.6.4]

```

原因： CentOS 6.6 不支持SECCOMP， 增加配置： bootstrap.system_call_filter: false



错:2：
[1]: max file descriptors [4096] for elasticsearch process is too low, increase to at least [65536]
解决：
切换到root用户：
su -root
vi /etc/security/limits.conf
增加配置：
* soft nofile 65536
* hard nofile 131072
* soft nproc 2048
* hard nproc 4096

错误3：
[2]: memory locking requested for elasticsearch process but memory is not locked
切换到root用户：
vim /etc/security/limits.conf
增加如下配置：
* soft memlock unlimited
* hard memlock unlimited

错误4：
[3]: max number of threads [1024] for user [hadoop] is too low, increase to at least [2048]
切换到root用户：
vi /etc/security/limits.d/90-nproc.conf
修改如下内容：
* soft nproc 1024
#修改为
* soft nproc 2048

错误5：
[4]: max virtual memory areas vm.max_map_count [65530] is too low, increase to at least [262144]
切换到root用户修改配置sysctl.conf
vi /etc/sysctl.conf
添加下面配置：
vm.max_map_count=655360
并执行命令：
sysctl -p

错误6：
ERROR: [1] bootstrap checks failed
[1]: system call filters failed to install; check the logs and fix your configuration or disable system call filters at your own risk

在es的配置文件elasticsearch.yml增加配置：
bootstrap.system_call_filter: false

二、HEAD插件安装

1. 下载github源码
https://github.com/mobz/elasticsearch-head.git

并解压
unzip elasticsearch-head-master.zip -d /home/hadoop/app/

2. 下载node并安装

解压：

```
wget https://nodejs.org/dist/v6.9.1/node-v6.9.1-linux-x64.tar.xz

yum -y install xz
xz -d node-v6.9.1-linux-x64.tar.xz
tar xf node-v6.9.1-linux-x64.tar
mv node-v6.9.1-linux-x64  /home/hadoop/app/node
```

加入环境变量：

```
export NODE_HOME=/home/hadoop/app/node
export PATH=$PATH:$NODE_HOME/bin
```

[hadoop@spark1234 app]$ node -v
v6.9.1
[hadoop@spark1234 app]$
[hadoop@spark1234 app]$ npm -v
3.10.8

设置npm的镜像：

```
npm config set registry https://registry.npm.taobao.org
```

切换到目录：cd elasticsearch-head-master

```
npm install
npm install -g grunt
```

编辑Gruntfile.js, 增加属性： hostname: '0.0.0.0',

```
connect: {
        server: {
                options: {
                        port: 9100,
                        hostname: '0.0.0.0',
                        base: '.',
                        keepalive: true
                }
        }
```

3. 修改Elasticsearch的属性：
vim elasticsearch.yml

```
http.cors.enabled: true
http.cors.allow-origin: "*"
```

4. 启动head服务：

```
grunt server &
```

5. 启动elasticsearch：

打开连接：
http://192.168.0.22:9100/

三、Elasticsearch操作

1. 基本操作

主要操作：创建文档、 插入数据、获取文档、更新文档、创建索引

基本操作：
创建文档：

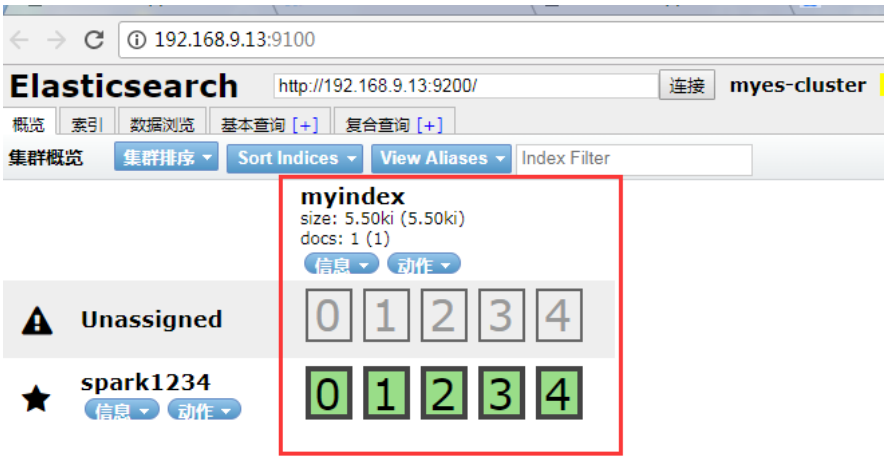
```
curl -XPUT  http://spark1234:9200/myindex/mytype/1?pretty -d '{
    "name":"xiao1",
    "age":22,
    "sex":"male",
    "hobby":["Football", "Basketball", "Sing"]
}'
```

```
[hadoop@spark1234 ~]$ curl -XPUT  http://spark1234:9200/myindex/mytype/1?pretty -d '{
>     "name":"xiao1",
>     "age":22,
>     "sex":"male",
>     "hobby":["Football", "Basketball", "Sing"]
> }'
{
  "_index" : "myindex",
  "_type" : "mytype",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "created" : true
}
```

创建文档返回信息

这里的created信息表示是否为新创建的document。

通过head插件查看：



插入部分数据：

```
curl -XPUT  http://spark1234:9200/myindex/mytype/0?pretty -d '{
    "name":"xiao1",
    "age":22,
    "sex":"male",
    "city": "beijing",
    "hobby":["Football", "Basketball", "Sing"]
}'

curl -XPUT  http://spark1234:9200/myindex/mytype/1?pretty -d '{
    "name":"xiao2",
    "age":22,
    "sex":"male",
     "city": "beijing",
    "hobby":["Football", "Basketball", "Sing"]
}'


curl -XPUT  http://spark1234:9200/myindex/mytype/2?pretty -d '{
    "name":"fish",
    "age":23,
    "city": "beijing",
    "sex":"male"
}'


curl -XPUT  http://spark1234:9200/myindex/mytype/3?pretty -d '{
```

```
    "name":"zhou",
    "age":25,
    "sex":"male",
     "city": "nanjing",
    "hobby":"fish"
}'

curl -XPUT  http://spark1234:9200/myindex/mytype/4?pretty -d '{
    "name":"ring",
    "age":26,
    "sex":"female",
    "hobby":"dance"
}'


curl -XPUT  http://spark1234:9200/myindex/mytype/5?pretty -d '{
    "name":"zhou ring",
    "age":26,
     "city": "nanjing",
    "sex":"female",
    "hobby":"dance"
}'

curl -XPUT  http://spark1234:9200/myindex/mytype/6?pretty -d '{
    "name":"zhou kai",
    "age":26,
    "city": "shanghai",
    "sex":"male",
    "hobby":"dance"
}'

curl -XPUT  http://spark1234:9200/myindex/mytype/7?pretty -d '{
    "name":"zhou kai",
    "age":27,
    "sex":"male",
    "city": "shanghai",
    "hobby":["dance"]
}'

curl -XPUT  http://spark1234:9200/myindex/mytype/8?pretty -d '{
    "name":"zhou ring kai",
    "age":27,
    "sex":"male",
    "city": "shanghai",
    "hobby":"dance"
}'

curl -XPUT  http://spark1234:9200/myindex/mytype/9?pretty -d '{
    "name":"ring zhou kai",
    "age":27,
    "city": "beijing",
    "sex":"male",
    "hobby":"dance"
}'
```

获取文档：

```
curl -XGET  http://spark1234:9200/myindex/mytype/1?pretty
```

```
[hadoop@spark1234 ~]$ curl -XGET  http://spark1234:9200/myindex/mytype/1?pretty
{
  "_index" : "myindex",
  "_type" : "mytype",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source" : {
    "name" : "xiao1",
    "age" : 22,
    "sex" : "male",
    "hobby" : [
      "Football",
      "Basketball",
      "Sing"
    ]
  }
}
```

更新文档：

```
curl -XPOST  http://spark1234:9200/myindex/mytype/1/_update?pretty -d '{
   "doc":{
      "name":"xiao3",
      "college":"ts"
   }
}'
```

也可以使用创建文档的命令，但是此时会覆盖原来的文档，如果使用创建文档的命令，那么需要指定所有的字段。
注意：Elasticsearch的更新不是真的修改了数据，而是在底层创建了一个新的文档，_version会递增1，这个与Hbase的更新过程类似。
ElasticSearch每次只获取最新_version的数据。

```
[hadoop@spark1234 ~]$ curl -XPOST  http://spark1234:9200/myindex/mytype/1/_update?pretty -d '{
>     "doc":{
>         "name1":"xiao2",
>         "college":"ts"
>     }
> }'
{
  "_index" : "myindex",
  "_type" : "mytype",
  "_id" : "1",
  "_version" : 2,
  "result" : "updated",
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  }
}
```

删除文档：

```
curl -XDELETE  http://spark1234:9200/myindex/mytype/1?pretty
```

```
[hadoop@spark1234 ~]$ curl -XDELETE  http://spark1234:9200/myindex/mytype/1?pretty
{
  "found" : true,
  "_index" : "myindex",
  "_type" : "mytype",
  "_id" : "1",
  "_version" : 7,
  "result" : "deleted",
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  }
}
```

创建索引：

```
curl -XPUT  http://spark1234:9200/mynewindex?pretty
```

```
[hadoop@spark1234 ~]$ curl -XPUT  http://spark1234:9200/mynewindex?pretty
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "mynewindex"
}
```

2. Mapping：

查看和指定字段的数据类型
查看：

```
curl -XGET  http://spark1234:9200/myindex/mytype/_mapping?pretty
```

```
$ curl -XGET  http://spark1234:9200/myindex/mytype/_mapping?pretty
{
  "myindex" : {
    "mappings" : {
      "mytype" : {
        "properties" : {
          "age" : {
            "type" : "long"
          },
          "hobby" : {
            "type" : "text",
            "fields" : {
              "keyword" : {
                "type" : "keyword",
                "ignore_above" : 256
              }
            }
          },
          "name" : {
            "type" : "text",
```

```
            "fields" : {
              "keyword" : {
                "type" : "keyword",
                "ignore_above" : 256
              }
            }
          },
          "sex" : {
            "type" : "text",
            "fields" : {
              "keyword" : {
                "type" : "keyword",
                "ignore_above" : 256
              }
            }
          }
        }
      }
    }
  }
}

```

动态Mapping：
在前面通过XPUT创建文档， 没有显示指定字段的数据类型， Elasticsearch自动生成了每个字段的数据类型。

显式Mapping：
首先查看下Elasticseach支持的数据类型：
数据类型（5.x版本)：
简单数据类型： text, keyword, date,integer, long, double, boolean, ip
支持JSON分层特性的类型： object, nested.
特定的数据类型： geo_point, geo_shape, completion.

在2.x还有string数据类型。

Elasticsearch能根据字段的值自动推测数据类型。

创建Mapping：

```
curl -XPUT http://spark1234:9200/mytest/orders/_mapping?pretty -d '{
    "properties":{
        "orderid":{
            "type":"text"
        },
        "goodname":{
            "type":"keyword"
        },
        "producer":{
            "type":"text",
            "index":"not_analyzed"
        },
        "goodnums":{
            "type":"integer"
        },
        "price":{
            "type":"float"
        },
        "orderdate":{
            "type":"date",
            "format":"dd/MM/YYYY"
        }
    }
}'
```

如果需要字段不被分词器拆分， 而是将字段作为一个整体存储在索引中， 在2.x和5.x分别做如下设置：
在2.x版本中，指定"index":"not-analyzed"
在5.x版本中， 指定数据类型为keyword即可。

注意： 在做Mapping， index必须是已经存在的。

```
curl -XPUT http://spark1234:9200/mytest/orders/_mapping?pretty -d '{
    "properties":{
        "orderid":{
            "type":"text"
        },
        "goodname":{
            "type":"keyword"
        },
        "producer":{
            "type":"string",
            "index":"not_analyzed"
        },
        "goodnums":{
            "type":"integer"
        },
        "price":{
            "type":"float"
        },
        "orderdate":{
            "type":"date",
            "format":"dd/MM/YYYY"
        }
    }
}'
```

```
[hadoop@spark1234 ~]$  curl -XGET  http://spark1234:9200/mytest/orders/_mapping?pretty
{
  "mytest" : {
    "mappings" : {
      "orders" : {
        "properties" : {
          "goodname" : {
            "type" : "keyword"
          },
          "goodnums" : {
            "type" : "integer"
          },
          "orderdate" : {
            "type" : "date",
            "format" : "dd/MM/YYYY"
          },
          "orderid" : {
            "type" : "text"
          },
          "price" : {
            "type" : "float"
          },
          "producer" : {
            "type" : "keyword"
          }
        }
      }
    }
  }
}

```

3. 索引模板：

在实际生产应用中， 有些索引可能具有相同或者相似的属性， 这个时候可以希望这些索引的类型和字段相同， 这个时候就可以使用索引模板。
比如电商企业，每个月都会有订单数据， 我可以按月建index， 索引名称：myorders_{yyyymm}, 每个月的index的type和字段数据类型均相同， 这个时候就可以使用模板了。
索引模板对应的模板：myorders_*， 那么当新建的索引的名称匹配myorders_*， 就会自动引用模板中定义的类型映射。

创建模板：
下面的模板表示index名称以myorders_开始的， 都会有模板里面设置的属性、type和数据类型。

```
curl -XPUT http://spark1234:9200/_template/myorders_template -d '{
  "template": "myorders_*",
  "settings": {
    "number_of_shards": 5
  },
  "mappings": {
    "t_order": {
      "_source": {
        "enabled": false
      },
      "properties": {
        "orderid": {
          "type": "text"
        },
        "price": {
          "type": "float"
        },
        "supplier": {
          "type": "keyword"
        },
        "created_at": {
          "type": "date",
          "format": "YYYY/MM/dd HH:mm:ss"
        }
      }
    }
  }
}'
```

查看所有模板列表：

```
curl -XGET http://spark1234:9200/_template?pretty
```

根据模板名称查看模板：

```
curl -XGET http://spark1234:9200/_template/myorders_template?pretty
```

```
$ curl -XGET http://spark1234:9200/_template/myorders_template?pretty
{
  "myorders_template" : {
    "order" : 0,
    "template" : "myorders_*",
    "settings" : {
      "index" : {
        "number_of_shards" : "5"
      }
    },
    "mappings" : {
      "t_order" : {
        "_source" : {
          "enabled" : false
        },
```

```
        "properties" : {
          "orderid" : {
            "type" : "text"
          },
          "price" : {
            "type" : "float"
          },
          "supplier" : {
            "type" : "keyword"
          },
          "created_at" : {
            "type" : "date",
            "format" : "YYYY/MM/dd HH:mm:ss"
          }
        }
      }
    },
    "aliases" : { }
  }
}

```

删除模板：

```
curl -XDELETE http://spark1234:9200/_template/myorders_template?pretty
```

创建一个匹配模板的index：

```
curl -XPUT http://spark1234:9200/myorders_20180228
```

查看这个index的mapping：

```
curl -XGET http://spark1234:9200/myorders_20180228/_mapping?pretty
```

```
$ curl -XGET http://spark1234:9200/myorders_20180228/_mapping?pretty
{
  "myorders_20180228" : {
    "mappings" : {
      "t_order" : {
        "_source" : {
          "enabled" : false
        },
        "properties" : {
          "created_at" : {
            "type" : "date",
            "format" : "YYYY/MM/dd HH:mm:ss"
          },
          "orderid" : {
            "type" : "text"
          },
          "price" : {
            "type" : "float"
          },
          "supplier" : {
            "type" : "keyword"
          }
        }
      }
    }
  }
}
```

```
28
```

4. ElasticSearch查询语句：

（1）. URL查询
查询name为xiao1的记录：
select * from myindex.mytyep where name='xiao1'

```
curl -XGET http://spark1234:9200/myindex/mytype/_search?q=name:xiao1
```

（2）. match_all查询
使用curl -XGET或者curl -XPOST

```
curl -XGET http://spark1234:9200/myindex/mytype/_search?pretty -d '{
    "query":{
        "match_all": {}
    }
}'
```

（3）. terms查询
查询倒排索引中与查询条件完全匹配的文档。

```
curl -XPOST http://spark1234:9200/myindex/mytype/_search?pretty -d '{
    "query":{
        "term": {
            "name":{
                "value":"ring"
            }
        }
    },
    "size": 10
}'
```

查看结果：
将name中包含ring关键字的都查询出来了：

```
{
  "took" : 20,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 2,
    "max_score" : 0.6931472,
    "hits" : [
      {
        "_index" : "myindex",
        "_type" : "mytype",
        "_id" : "4",
        "_score" : 0.6931472,
        "_source" : {
```

```
          "name" : "ring",
          "age" : 26,
          "sex" : "female",
          "hobby" : [
            "dance"
          ]
        }
      },
      {
        "_index" : "myindex",
        "_type" : "mytype",
        "_id" : "5",
        "_score" : 0.25811607,
        "_source" : {
          "name" : "zhou ring",
          "age" : 26,
          "sex" : "female",
          "hobby" : [
            "dance"
          ]
        }
      }
    ]
  }
}
```

（4）. Boolean查看

类比关系型数据库的and或or操作：

must: 满足条件的数据

must_not: 不满足条件的数据

should： 至少满足其中一个条件的数据

select * from myindex.mytype where age=26 and name in ("zhou", "ring");

```
curl -XPOST http://spark1234:9200/myindex/mytype/_search?pretty -d '{
    "query":{
        "bool": {
            "must":[
                {
                    "term": {
                      "age": 26
                    }
                }
            ],
            "should":[
                {
                    "terms": {
                         "name":["zhou", "ring"]
                    }
                }
            ]
        }

    },
    "size": 10
}'
```

可以设置mininum_number_should_match来控制使用should时至少要满足的条件的个数。

（5）. match查询

字段name包含ring或者kai的数据：

```
curl -XGET http://spark1234:9200/myindex/mytype/_search?pretty -d '{
    "query":{
```

```
        "match": {
            "name" : {
                "query": "ring kai"
            }
        }
    }
}'
```

如果要精确匹配， 使用"type": "phrase"， 可以指定slop参数指定间隔多少个词还能匹配， 比如设置slop为1， 那么查询条件（ "query": "ring kai" ）可以匹配"zhou ring kai" 或者 "ring zhou kai"

```
curl -XGET http://spark1234:9200/myindex/mytype/_search?pretty -d '{
    "query":{
        "match": {
            "name" : {
                "query": "zhou kai",
                "type": "phrase",
                "slop": 1
            }
        }
    }
}'
```

（6）. multi_match
一个查询关键字， 在多个字段中查询， 可以使用multi_match

查询所有字段， 匹配ring关键字的

```
curl -XGET http://spark1234:9200/myindex/mytype/_search?q=fish
```

name或者hobby字段匹配fish关键字的记录：
select * from mytype where name ="fish" or hobby="fish"

```
curl -XGET http://spark1234:9200/myindex/mytype/_search?pretty -d '{
    "query":{
        "multi_match": {
            "query" : "fish",
            "fields": ["name", "hobby"]
        }
    }
}'
```

（7）. range查询
查看age在23-25之间的记录

```
curl -XGET http://spark1234:9200/myindex/mytype/_search?pretty -d '{
    "query":{
        "range": {
            "age" : {
                "gte":23,
                "lte":25
            }
        }
    }
}'
```

（8） wildcard查询
查询name匹配xiao*的记录

```
curl -XGET http://spark1234:9200/myindex/mytype/_search?pretty -d '{
    "query":{
        "wildcard": {
            "name" : "xiao*"
        }
    }
}'
```

查询条件最好不要以?或者*开始， 否则查询的性能会非常差。

（9）. 过滤器

在2.x中可以这样使用， 在5.x中，filtered已经废弃了， 不过可以使用bool替换，

```
curl -XGET http://spark1234:9200/myindex/mytype/_search?pretty -d '{
    "query":{
        "filtered": {
            "filter" : {
                "exists": {
                    "field":"age"
                }
            }
        }
    }
}'
```

在5.x中， 这样使用：

```
curl -XGET http://spark1234:9200/myindex/mytype/_search?pretty -d '{
    "query":{
        "bool": {
            "filter" : {
                "exists": {
                    "field":"age"
                }
            }
        }
    }
}'
```

过滤器与查询的区别：
过滤器和查询很像， 唯一的区别是查询执行的是相关性操作， 过滤器不会使用计分的方法返回相似的结果， 它只会返回和查询结果完全匹配的结果。 如果有过滤器， 那么过滤器会先被执行， 可以减少需要查询的范围， 在第一次使用filter之后， 返回的结果缓存在内存中， 可以提升性能。

再看另外一个过滤器：

```
curl -XGET http://spark1234:9200/myindex/mytype/_search?pretty -d '{
    "query":{
        "bool": {
            "must":[
                {
                    "multi_match":{
                        "query": "ring",
                        "fields":["name", "sex"]
                    }
                }
            ],
            "filter" : {
                "range": {
                    "age": {
                        "gt": 26
                    }
                }
            }
        }
```

```
    }
}'
```

6. 聚合分析

（1）. terms聚合
group by
根据性别统计人数， 按照人数倒序排列。

select sex, count(*) from myindex.mytype group sex

```
curl -XPOST 'http://spark1234:9200/myindex/mytype/_search?pretty' -d '
{
    "aggs": {
        "num_by_sex": {
            "terms": {
                "field" : "sex",
                "size":2
            }
        }
    }
}'
```

报错：

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "illegal_argument_exception",
        "reason" : "Fielddata is disabled on text fields by default. Set fielddata=true on [sex] in order to load fielddata in memory by uninverting the inverted index. Note that this can however use significant memory. Alternatively use a keyword field instead."
      }
    ],
    "type" : "search_phase_execution_exception",
    "reason" : "all shards failed",
    "phase" : "query",
    "grouped" : true,
    "failed_shards" : [
      {
        "shard" : 0,
        "index" : "myindex",
        "node" : "SagfD8QySs-RCE7I-HZWAA",
        "reason" : {
          "type" : "illegal_argument_exception",
          "reason" : "Fielddata is disabled on text fields by default. Set fielddata=true on [sex] in order to load fielddata in memory by uninverting the inverted index. Note that this can however use significant memory. Alternatively use a keyword field instead."
        }
      }
    ]
  },
  "status" : 400
}

```

报错的原因：
Fielddata can consume a lot of heap space, especially when loading high cardinality text fields. Once fielddata has been loaded into the heap, it remains there for the lifetime of the segment.

Fielddata默认是关闭， 因为Fielddata会占用大量的heap内存。
有两种解决方法：
(1). 开启字段的fielddata

```
curl -XPUT 'http://spark1234:9200/myindex/_mapping/mytype?pretty' -d '
{
    "properties": {
        "sex": {
            "type": "text",
            "fielddata":true
            }
        }
    }
}'
```

然后执行聚合：

```
curl -XPOST 'http://spark1234:9200/myindex/mytype/_search?pretty' -d '
{
    "aggs": {
        "num_by_sex": {
            "terms": {
                "field" : "sex",
                "size":2
            }
        }
    }
}'
```

(2). 聚合时候使用字段名.keyword

```
curl -XPOST 'http://spark1234:9200/myindex/mytype/_search?pretty' -d '
{
    "aggs": {
        "num_by_sex": {
            "terms": {
                "field" : "sex.keyword",
                "size":2
            }
        }
    }
}'
```

如果按照性别的人数正序排序：

```
curl -XPOST 'http://spark1234:9200/myindex/mytype/_search?pretty' -d '
{
    "aggs": {
        "num_by_sex": {
            "terms": {
                "field" : "sex.keyword",
                "size":2,
                "order": {
                    "_count": "asc"
                }
            }
        }
    }
}'
```

如果要返回所有的聚合结果， 将size设置为2， sise默认是10.

（2）. histogram聚合
将结果以直方图的形式展示， 对数值的字段按照数据值区间进行聚合。
下面的查询， 以2年为一个区间进行聚合， 获取每个区间的人数。

```
curl -XPOST 'http://spark1234:9200/myindex/mytype/_search?pretty' -d '
{
    "aggs": {
        "num_by_age": {
            "histogram": {
                "field" : "age",
                "interval":2,
                "min_doc_count": 0
            }
        }
    }
}'
```

查询结果：

```
"aggregations" : {
    "num_by_age" : {
      "buckets" : [
        {
          "key" : 22.0,
          "doc_count" : 3
        },
        {
          "key" : 24.0,
          "doc_count" : 1
        },
        {
          "key" : 26.0,
          "doc_count" : 6
        }
      ]
    }
  }
```

（3）. range聚合
自定义数值区间进行聚合。

按照年纪区间进行聚合：

```
curl -XPOST 'http://spark1234:9200/myindex/mytype/_search?pretty' -d '
{
    "aggs": {
        "num_by_age": {
            "range": {
                "field" : "age",
                "ranges": [
                    {
                        "from": 0,
                        "to": 23
                    },
                    {
                        "from":23,
                        "to":26
                    },
                    {
                        "from": 26,
                        "to":100
                    }
                ]
            }
        }
    }
}'
```

返回结果：

```
  "aggregations" : {
    "num_by_age" : {
      "buckets" : [
        {
          "key" : "0.0-23.0",
          "from" : 0.0,
          "to" : 23.0,
          "doc_count" : 2
        },
        {
          "key" : "23.0-26.0",
          "from" : 23.0,
          "to" : 26.0,
          "doc_count" : 2
        },
        {
          "key" : "26.0-100.0",
          "from" : 26.0,
          "to" : 100.0,
          "doc_count" : 6
        }
      ]
    }
  }

```

（4）统计每个区间的sum或者avg：

```
curl -XPOST 'http://spark1234:9200/myindex/mytype/_search?pretty' -d '
{
    "aggs": {
        "num_by_age": {
            "range": {
                "field" : "age",
                "ranges": [
                    {
                        "from": 0,
                        "to": 23
                    },
                    {
                        "from":23,
                        "to":26
                    },
                    {
                        "from": 26,
                        "to":100
                    }
                ]
            },
        "aggs":{
            "ageavg":{
                "avg": {
                    "field": "age"
                }
            }
        }
        }
    }
}'
```

执行结果：

```
"aggregations" : {
    "num_by_age" : {
      "buckets" : [
        {
```

```
          "key" : "0.0-23.0",
          "from" : 0.0,
          "to" : 23.0,
          "doc_count" : 2,
          "ageavg" : {
            "value" : 22.0
          }
        },
        {
          "key" : "23.0-26.0",
          "from" : 23.0,
          "to" : 26.0,
          "doc_count" : 2,
          "ageavg" : {
            "value" : 24.0
          }
        },
        {
          "key" : "26.0-100.0",
          "from" : 26.0,
          "to" : 100.0,
          "doc_count" : 6,
          "ageavg" : {
            "value" : 26.5
          }
        }
      ]
    }
  }
```

（5）统计sum：

```
curl -XPOST 'http://spark1234:9200/myindex/mytype/_search?pretty' -d '
{
    "aggs": {
        "num_by_age": {
            "range": {
                "field" : "age",
                "ranges": [
                    {
                        "from": 0,
                        "to": 23
                    },
                    {
                        "from":23,
                        "to":26
                    },
                    {
                        "from": 26,
                        "to":100
                    }
                ]
            },
        "aggs":{
            "agesum":{
                "sum": {
                    "field": "age"
                }
            }
        }
        }
    }
}'
```

统计结果：

```
"aggregations" : {
```

```
    "num_by_age" : {
      "buckets" : [
        {
          "key" : "0.0-23.0",
          "from" : 0.0,
          "to" : 23.0,
          "doc_count" : 2,
          "agesum" : {
            "value" : 44.0
          }
        },
        {
          "key" : "23.0-26.0",
          "from" : 23.0,
          "to" : 26.0,
          "doc_count" : 2,
          "agesum" : {
            "value" : 48.0
          }
        },
        {
          "key" : "26.0-100.0",
          "from" : 26.0,
          "to" : 100.0,
          "doc_count" : 6,
          "agesum" : {
            "value" : 159.0
          }
        }
      ]
    }
  }

```

（6）. 嵌套聚合

统计每个城市，每个性别的总年纪和平均年纪
select city, sex, sum(age), avg(age) from mytype grouup by city, sex

```
curl -XPOST 'http://spark1234:9200/myindex/mytype/_search?pretty' -d '
{
    "aggs": {
        "bySex": {
            "terms": {
                "field" : "city.keyword",
                "size":10
            },
        "aggs":{
            "byage":{
                "terms": {
                    "field": "sex.keyword",
                    "size": 10
                },
                "aggs": {
                    "sex_age_sum": {
                        "sum": {
                            "field": "age"
                        }
                    },
                    "sex_age_avg" :{
                        "avg": {
                            "field": "age"
                        }
                    }
                }
            }
          }
        }
    }
```

```
}'
```

Elasticsearch集群相关概念
Elasticsearch是主从架构， 但是Elasticsearch集群又是具备去中心化的特性， 因为可以通过任意节点都可以同Elasticsearch集群通信， 并且是等价的。
Elasticsearch集群的节点主要有主节点（master ）和数据节点（data ）。

1. 主节点
主节点负责集群的状态变更， 包括：增删节点， index（索引）、mapping（映射）的管理， 副本的管理， 分片的重分配等。
为了防止脑裂， 通常设置多个主节点。
在生产中，主节点是独占一台服务器， 这个服务器不会存储数据。 因为如果主节点的负载过重， 有可能导致主节点不能提供服务， 甚至导致脑裂。

主节点独占一台服务器的配置：
node.master: true
node.data:false

2. 数据节点
数据节点用于存放数据， 也就是存放lucene索引的。
数据节点在生产环境也是配置成独占的， 配置如下：
node.master: false
node.data:true

3. discovery.zen（节点发现）
Elasticsearch集群是通过cluster.name配置集群的名称， 所有具有相同cluster.name的节点组成一个集群。
Elasticsearch通过discover的方式自动发现节点， 并把节点加入到集群中。

节点发现有两种方式， 组播和单播。

组播： 每个节点实例向指定的多播组和端口发送多播的ping请求， 每个节点相应请求， 当找到主节点， 就将这个节点示例接入集群。 如果多播没有发现主节点， 集群会选择一个主节点。
在生产环境中， 一般不推荐使用组播的方式， 因为可能会将不相干的节点加入到集群。
discovery.zen.ping.multicast.enabled=false

单播： 只向配置好的主机列表和端口发送请求。

```
discovery.zen.ping.unicast.hosts:
   - 192.168.1.10:9300
   - 192.168.1.11
   - seeds.mydomain.com
```

注意： 不需要把集群中所有的节点都配置上， 新的节点会根据配置的主机和端口通信， 只要发现了某个集群， 就会加入到这个集群。 但是从高可用考虑，节点也不能配置的太少。

4. 分片/副本
一个索引可以存储超出单个结点硬件限制的大量数据。比如，一个具有10亿文档的索引占据1TB的磁盘空间，而任一节点可能没有这样大的磁盘空间来存储或者单个节点处理搜索请求，响应会太慢。
为了解决这个问题，Elasticsearch提供了将索引划分成多片的能力，这些片叫做分片。当你创建一个索引的时候，你可以指定你想要的分片的数量。每个分片本身也是一个功能完善并且独立的“索引”，这个“索引” 可以被放置到集群中的任何节点上。

一个索引可以有多个分片（shard）， 每个shard就是一个lucene索引。
从集群分布式角度， 可以把分片类比Kafka中topic的分区（partition）。

索引默认会创建5个shard。

分片很重要， 主要有两个方面的原因：

- 允许你水平分割/扩展你的内容容量
- 允许你在分片（位于多个节点上）之上进行分布式的、并行的操作，进而提高性能/吞吐量

在一个网络/云的环境里，失败随时都可能发生。在某个分片/节点因为某些原因处于离线状态或者消失的情况下，故障转移机制是非常有用且强烈推荐的。为此， Elasticsearch允许你创建分片的一份或多份拷贝，这些拷贝叫做复制分片，或者直接叫复制。
复制之所以重要，有两个主要原因：

- 在分片/节点失败的情况下，复制提供了高可用性。复制分片不与原/主要分片置于同一节点上是非常重要的。
- 因为搜索可以在所有的复制上并行运行，复制可以扩展你的搜索量/吞吐量

默认情况下，Elasticsearch中的每个索引分配5个主分片和1个复制。这意味着，如果你的集群中至少有两个节点，你的索引将会有5个主分片和另外5个复制分片（1个完全拷贝），这样每个索引总共就有10个分片。

注意： 分片可以在创建index的时候指定， 制定后就不能修改。 副本可以修改。

每个shard的最大数据量：2,147,483,519 （20亿）

```
Each Elasticsearch shard is a Lucene index. There is a maximum number of documents you can have in a single Lucene index. As of
LUCENE-5843, the limit is 2,147,483,519 (= Integer.MAX_VALUE - 128) documents. You can monitor shard sizes using the _cat/shards API.
```

分片设置：

```
curl  -XPUT  http://spark1234:9200/myshard/ -d '{
    "settings": {"number_of_shards":2, "number_of_replicas":0}
}'
```

设置副本数：

```
curl -XPUT  http://spark1234:9200/myindex/_settings -d '{
    "number_of_replicas": 0
}'

```

5. 倒排索引：

正排索引：

文档ID	文档内容	分词
1	spark provides high-level APIs in Java, Scala, Python and R	spark,provides,high-level,APIs,in,Java,Scala,P
2	Scala and Java users can include Spark in their projects using Java Maven coordinates	Scala,and,Java,users,can,include,Spark,in,the
3	spark currently provides several options for spark deployment	spark,currently,provides,several,options,for,de

通过文档的id， 可以确定这个文档包含了哪些单词。 但是， 如果需要查看某个单词（比如spark）在哪个文档出现过， 那就需要遍历所有文档了。

倒排索引：

单词	文档ID
spark	1,2,3
Java	1,2
Scala	1,2
provides	1,2
Python	1
high-level	1
APIs	1
in	1,
and	1,2
...	...

可以快速定位单词在哪个文档里面出现和位置信息。

生产环境集群的配置：

1. 集群主机的选型
集群中主机最好配置相近， 集群的性能符合木桶定律， 即集群的查询性能是由集群中性能最差的主机决定的。

2. 硬盘
ES对IO的性能要求比较高， 建议使用SSD硬盘替换传统的机械硬盘。

3. 内存
Elasticsearch默认的堆（heap）内存是1G。
在生产环境中， 建议将ES的内存设置为物理内存的一半， 但是也不要超过32G。

https://www.elastic.co/guide/en/elasticsearch/guide/current/heap-sizing.html#compressed_oops

https://www.elastic.co/guide/en/elasticsearch/guide/current/heap-sizing.html#compressed_oops

Docs

If you are not aggregating on analyzed string fields (e.g. you won't be needing fielddata) you can consider lowering the heap even more. The smaller you can make the heap, the better performance you can expect from both Elasticsearch (faster GCs) and Lucene (more memory for caching).

Don't Cross 32 GB!

edit

There is another reason to not allocate enormous heaps to Elasticsearch. As it turns out, the HotSpot JVM uses a trick to compress object pointers when heaps are less than around 32 GB.

In Java, all objects are allocated on the heap and referenced by a pointer. Ordinary object pointers (OOP) point at these objects, and are traditionally the size of the CPU's native *word*: either 32 bits or 64 bits, depending on the processor. The pointer references the exact byte location of the value.

For 32-bit systems, this means the maximum heap size is 4 GB. For 64-bit systems, the heap size can

4. 集群名称和节点名称必须要配置
es根据集群的名称组成一个集群， 在生产中可以指定一个具有实际意义的集群名称。
节点名称一般指定主机名， 便于管理和监控， 出现故障也易于定位。

5. 使用独占的主节点和数据节点。

6. 减少swap内存交换
bootstrap.memory_lock: true

锁定内存， 避免swap内存交换提升性能。

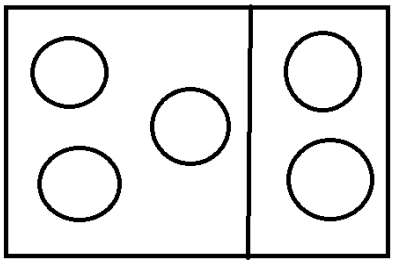
7. 副本、分片数量的规划

分片的数量需要根据实际数据量测试， 确定最合适的分片数量。
分片数过少， 并发少。 分片数过多， 消耗的系统资源多（打开文件句柄、内存）， 影响性能。

一个分片最多能存储20亿条记录， 根据实际性能规划。

副本一般推荐设置2-3个。 副本数量能提升数据检索的性能。
副本需要从主分片同步数据， 为了避免服务器压力， 副本数量不能过多。

8. 避免脑裂



一个集群里面出现了两个或以上的master节点， 就称为脑裂。
发生脑裂的原因主要由两个： 网络质量不好、master节点负载过高。

因此， 在前面的优化中， 使用独占的主节点和数据节点可以避免负载过高。

缓解脑裂还可以设置一个参数： discovery.zen.minimum_master_node。这个参数决定了至少有多少个存活的节点， 才选举产生新的master节点。

discovery.zen.minimum_master_node: (master_eligible_nodes / 2) + 1

换句话说，如果有三个主节点，那么最小主节点应该设置为（3/2）+ 1或2：

discovery.zen.minimum_master_nodes：2

如果集群网络环境不太好， 建议将网络故障的超时时间设置的长一点。
discovery.zen.ping_timeout
设置集群中自动发现其它节点时ping连接超时时间，默认为3秒，对于比较差的网络环境可以高点的值来防止自动发现时出错。

9. 集群启动数据恢复相关参数

集群维护后重启。 比如一共5个节点， 其中3个节点启动很快， 2个节点启动很慢。
当3个节点启动后， 另外两个节点还没启动， 这个时候集群找不到没启动服务的节点上的shard， 这个时候集群就会尝试做分片的重新分配 。 等另外两个节点启动后， 集群再次对集群的分片重新分配。 这样就导致了不必要的IO操作， 对性能影响非常大。
gateway.recover_after_nodes: 3 当集群中至少启动3个节点，才开始做分片的分配。

gateway.recover_after_nodes
设置集群中N个节点启动时进行数据恢复，默认为1。
gateway.recover_after_time
设置初始化数据恢复进程的超时时间，默认是5分钟。
gateway.expected_nodes
设置这个集群中节点的数量，默认为2，一旦这N个节点启动，就会立即进行数据恢复。

10. 数据导入优化
两种方式提升性能。
（1）、在数据导入前， 将副本数设置为0.
数据导入完成后， 将副本数设置为正常数值。
在数据导入， 如果存在副本， 需要从主分片同步数据， 增加ES服务器的压力。
（2）、设置index.refresh_interval参数
index.refresh_interval默认是1秒，刷新后文档就能被检索到。
可以将index.refresh_interval设置 为-1， 不刷新。
或者将index.refresh_interval增大， 比如30秒， 避免频繁刷新。

11. 关闭不必要的index
open的索引的shard会加载到内存中， close不必要的index可以减少内存占用。

```
curl  -XPOST  http://spark1234:9200/myshard/_close
```

12. 事务刷新阈值
Elasticsearch在写入时是先写日志， 即事务日志。 事务日志在满足一定的阈值大小就会进行延迟提交， 可以确保插入和删除操作的原子性。
可以将事务日志的刷新阈值提高， index.translog.flush_threshold_size默认是512M， 比如设置成1g：
index.translog.flush_threshold_size=1GB

13. 索引缓存大小
增加索引缓存， 可以在内存中缓存更多的数据。
indices.memory.index_buffer_size 默认是10%， 即为堆内存的10%。 一般不需要设置， 如果需要设置， 一定要进行性能测试。

集群状态查看：

```
curl -XGET http://spark1234:9200/_cluster/health?pretty

curl -XGET http://spark1234:9200/_cluster/health?level=indices
```

```
[hadoop@spark1234 target]$ curl -XGET http://spark1234:9200/_cluster/health?pretty
{
  "cluster_name" : "myes-cluster",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 1,
  "number_of_data_nodes" : 1,
  "active_primary_shards" : 36,
  "active_shards" : 36,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 36,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 50.0
}
```