ElasticSearch

1、MySQL、MongoDB、ElasticSearch 类比

类型	MySQL	MongoDB	ElasticSearch
数据库	数据库 database	数据库 database	索引库 indices
表	表 table	集合 collection	类型 type (将被废弃)
行	行 row	文档 document	文档 document
列	列 column	字段 Field	字段 Field
结构	scheme	mode	mapping

2、ElasticSearch 安装

1. docker 安装方式

1. 安装 docker (Centos7)

- 卸载原来安装的 docker 环境

```
yum remove docker \
                  docker-client \
                  docker-client-latest \
                  docker-common \
                  docker-latest \
                  docker-latest-logrotate \
                  docker-logrotate \
                  docker-engine
```

- 安装工具包

```
yum install -y yum-utils
```

- 配置 docker

```
yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
```

- 安装 docker 环境

```
yum install docker-ce docker-ce-cli containerd.io
```

- 启动 docker

```
systemctl start docker
```

- 测试 docker

```
docker -v # 检查版本
docker ps # 检查运行的容器
docker images # 检查已有镜像
```

- 设置 docker 开机启动

```
systemctl enable docker
```

- 设置镜像存库（阿里云镜像存库）

```
# 创建文件夹
mkdir -p /etc/docker
# 修改配置，设置镜像
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://vw9qapdy.mirror.aliyuncs.com"]
}
EOF
# 重启后台线程
systemctl daemon-reload
# 重启 docker
systemctl restart docker
```

2. 安装 ElasticSearch 服务

- 拉取 ElasticSearch docker 镜像

```
docker pull elasticsearch：7.4.2
```

- 配置挂载数据文件夹

```
# 创建配置文件目录
mkdir -p /mydata/elasticsearch/config

# 创建数据目录
mkdir -p /mydata/elasticsearch/data

# 将/mydata/elasticsearch/文件夹中文件都可读可写
chmod -R 777 /mydata/elasticsearch/

# 配置任意机器可以访问 elasticsearch
echo "http.host: 0.0.0.0" >/mydata/elasticsearch/config/elasticsearch.yml
```

- 启动 elasticsearch

```
docker run --name elasticsearch -p 9200:9200 -p 9300:9300 \
-e  "discovery.type=single-node" \
-e ES_JAVA_OPTS="-Xms64m -Xmx512m" \
-v
/mydata/elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/conf
ig/elasticsearch.yml \
-v /mydata/elasticsearch/data:/usr/share/elasticsearch/data \
-v  /mydata/elasticsearch/plugins:/usr/share/elasticsearch/plugins \
-d elasticsearch:7.4.2
```

> `-p 9200:9200 -p 9300:9300`：向外暴露两个端口，9200用于HTTP REST API请求，9300 ES 在分布式集群状态下 ES 之间的通信端口；
>
> `-e  "discovery.type=single-node"`：es 以单节点运行
>
> `-e ES_JAVA_OPTS="-Xms64m -Xmx512m"`：设置启动占用内存，不设置可能会占用当前系统所有内存
>
> -v：挂载容器中的配置文件、数据文件、插件数据到本机的文件夹；
>
> `-d elasticsearch:7.6.2`：指定要启动的镜像

- 查看 docker 启动的容器

```
docker ps # 查看运行的
docker ps -a # 查看所有
```

- 访问 http://10.25.175.108:9200
- 设置容器随 docker 容器启动

```
docker update elasticsearch --restart=always
```

2. 普通安装方式

- 下载 elasticsearch 压缩包
- 解压缩并修改目录名称

```
tar -zxvf elasticsearch-7.4.2-linux-x86_64.tar.gz
mv elasticsearch-7.4.2-linux-x86_64 elasticsearch
```

- 修改 JVM 参数

```
cd elasticsearch
cd config
vim jvm.options
```

修改内存配置

```
-Xms512m
-Xmx512m
```

- 修改 elasticsearch.yml

```
# 设置ip地址，任意网络均可访问
network.host = 0.0.0.0
# 跨域配置
http.cors.enabled: true
http.cors.allow-origin: "*"
```

- 修改宿主机配置

```
# 到宿主机上打开文件
vim /etc/sysctl.conf
# 增加这样一条配置，一个进程在VMAs(虚拟内存区域)创建内存映射最大数量
vm.max_map_count=655360
# 让配置生效
sysctl -p
```

- 启动 elasticsearch 服务

```
# 这里说明一点，es 服务启动需要另外的用户
# adduser es
# chmod -R 777 elasticsearch
# su es
cd ../bin
./elasticsearch -d
```

- 访问 http://10.25.175.108:9200

> 可能出现的异常解决方法：
>
> - X-Pack is not supported and Machine Learning is not available for [windows-x86]; you can use the other X-Pack features (unsupported)
>
> ```
> # vim elasticsearch.yml
> xpack.ml.enabled: false
> ```
>
> - java.lang.RuntimeException: can not run elasticsearch as root
>
> ```
> # 切换用户运行 es 服务
> su es
> ```
>
> - max file descriptors [4096] for elasticsearch process is too low
>
> ```
> # 最大线程数设置的太低了，需要改成4096
> vi /etc/security/limits.conf
> # ElasticSearch添加如下内容:
>   * soft nofile 65536
>   * hard nofile 131072
>   * soft nproc 2048
>   * hard nproc 4096
> ```
>
> - max number of threads [1024] for user [elsearch] is too low

```
# Centos6不支持SecComp，而ES5.2.0默认bootstrap.system_call_filter为true
# 解决：切换到root用户，进入limits.d目录下修改配置文件。
vi /etc/security/limits.d/90-nproc.conf
# 修改如下内容：
  * soft nproc 1024
# 修改为
  * soft nproc 4096
```

- system call filters failed to install; check the logs and fix your configuration

```
vim config/elasticsearch.yml
# 添加
bootstrap.system_call_filter: false
bootstrap.memory_lock: false
```

3、Kibana 安装

1. docker 安装方式

- 拉取 Kibana docker 镜像

```
docker pull kibana:7.4.2
```

- 启动 Kibana 容器

```
docker run --name kibana \
-e ELASTICSEARCH_HOSTS=http://10.25.175.108:9200 \
-p 5601:5601 \
-d kibana:7.4.2
```

- 访问 http://10.25.175.108:5601
- 设置容器随 docker 容器启动

```
docker update kibana --restart=always
```

2. 普通安装方式

- 下载 Kibana 安装包
- 解压缩并修改目录名称

```
tar -zxvf kibana-7.4.2-linux-x86_64.tar.gz
mv kibana-7.4.2-linux-x86_64.tar.gz kibana
```

- 修改 kibana 配置文件

```
cd kibana
cd config
vim kibana.yml
```

配置 elasticsearch 服务信息

```
# es 服务
elasticsearch.hosts: ["http://10.25.175.108:9200"]
# 国际化，如果以前有启动过，请先删除 es 中关于 kibana 的索引
i18n.locale: "zh-Cn"
```

- 启动 kibana 服务

```
cd ../bin
./kibana
```

- 访问 http://10.25.175.108:5601

4、ElasticSearch CRUD - Postman

1. 服务相关

- /_cat/health 查看ES健康状况

```
GET    http://10.25.175.108:9200/_cat/health
```

- /_cat/nodes 查看所有节点

```
GET    http://10.25.175.108:9200/_cat/nodes
```

- /_cat/master 查看主节点信息

```
GET    http://10.25.175.108:9200/_cat/master
```

- /_cat/indices 查看所有索引

```
GET    http://10.25.175.108:9200/_cat/indices
```

2. 索引相关

- 创建索引 shopping

```
PUT    http://10.25.175.108:9200/shopping
```

- 查看全部索引

```
GET    http://10.25.175.108:9200/_cat/indices
```

- 查看指定索引 shopping

```
GET    http://10.25.175.108:9200/shopping
```

- 删除索引 shopping

```
DELETE   http://10.25.175.108:9200/shopping
```

3. 文档相关

- 新建文档（随机生成 ID）

```
POST   http://10.25.175.108:9200/shopping/_doc
{
 "name":"小米手机",
 "price":3999,
 "images":["1.jpg","2.jpg"]
}
```

- 新建文档（指定 ID）

```
POST   http://10.25.175.108:9200/shopping/_doc/1
{
 "name":"小米手机",
 "price":3999,
 "images":["1.jpg","2.jpg"]
}
```

- 批量新建 _bulk

```
POST   http://10.25.175.108:9200/shopping/_bulk
{"index":{"_id":"1"}} # 指定文档 id
{"name":"小米手机"} # 文档内容
{"index":{"_id":"2"}} # 指定文档 id
{"name":"华为手机"} # 文档内容
```

- 批量增删改

```
POST   http://10.25.175.108:9200/_bulk
{"delete":{"_index":"shopping","_id":"1"}} # 删除 shopping 索引 id 为 1 的文档
{"create":{"_index":"shopping","_id":"1"}} # 新建 shopping 索引 id 为 1 的文档
{"name":"小米手机"}
{"index":{"_index":"shopping"}} # 新建 shopping 不指定 id 的文档
{"name":"华为手机"}
{"update":{"_index":"shopping","_id":"1"}} # 更新 shopping 索引 id 为 1 的文档
{"doc":{"name":"锤子手机"}}
```

- 更新文档 - 全量

```
POST  http://10.25.175.108:9200/shopping/_doc/1
{
 "name":"华为手机",
 "price":3888
}
```

- 更新文档 - 局部

```
POST  http://10.25.175.108:9200/shopping/_update/1
{
 "doc":{
  "name":"苹果手机",
  "price":4000
 }
}
```

- 删除文档

```
DELETE  http://10.25.175.108:9200/shopping/_doc/1
```

4. 文档查询

- URL 带参数查询

```
GET   http://10.25.175.108:9200/shopping/_search?q=title"小米
```

- match 查询 - 全文检索

```
POST   http://10.25.175.108:9200/shopping/_search
{
    "query":{
        "match":{
            "category":"小米"
        }
    }
}
```

- match_all 查询全部

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "query":{
  "match_all":{}
 }
}
```

- term 词条查询

```
POST   http://10.25.175.108:9200/shopping/_search
{
  "query": {
    "term": {
      "name": {
        "value": "米"
      }
    }
  }
}
```

- terms 多词条查询

```
POST   http://10.25.175.108:9200/shopping/_search
{
  "query": {
    "terms": {
      "name": [
        "小",
        "米"
      ]
    }
  }
}
```

- _source 查询指定字段

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "query":{
  "match_all":{}
 },
 "_source":["name"]
}
```

- from size 分页查询

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "query":{
  "match_all":{}
 },
 "_source":["name","price"],
 "from":0,
 "size":3
}
```

- sort 排序

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "query":{
  "match_all":{}
 },
 "_source":["name","price"],
 "sort":{
  "price":{
   "order": "asc"
  }
 }
}
```

- bool 组合查询 - must/must_not

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "query":{
```

```
  "bool":{
   "must":[
     {
      "match":{
       "name":"小米"
      }
     },{
      "match":{
       "price":3999.0
      }
     }
    ]
  }
 }
}
```

- bool 组合查询 - should

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "query":{
  "bool":{
   "should":[
     {
      "match":{
       "name":"小米"
      }
     },{
      "match":{
       "price":3999.0
      }
     }
    ]
  }
 }
}
```

- range 范围查询

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "query":{
  "bool":{
   "should":[
     {
      "match":{
       "name":"小米"
      }
     },{
      "match":{
       "price":3999.0
      }
     }
    ],
    "filter":{
     "range":{
```

```
      "price":{
       "gt":2000
      }
     }
    }
  }
 }
}
```

- match_phrase 查询 - 完全匹配

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "query":{
  "match_phrase":{
   "name":"为"
  }
 }
}
```

- fuzzy 模糊查询

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "query":{
  "fuzzy":{
   "name":"米"
  }
 }
}
```

- highlight 高亮查询

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "query":{
  "match_phrase":{
   "name":"为"
  }
 },
 "highlight":{
  "fields":{
   "name":{
    "pre_tags":"<font color='red'>",
    "post_tags":""
   }
  }
 }
}
```

- aggs 聚合查询 - terms（类似 group by）

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "aggs":{ # 聚合查询 query 为普通查询
  "price_group":{ # 聚合名称
   "terms":{ # 聚合类型
    "field":"price" # 聚合字段
   }
  }
 },
 "size":0
}
```

- aggs 聚合查询 - avg 平均值

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "aggs":{
  "price_avg":{
   "avg":{
    "field":"price"
   }
  }
 },
 "size":0
}
```

- aggs 聚合查询 - max 最大值

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "aggs":{
  "price_max":{
   "max":{
    "field":"price"
   }
  }
 },
 "size":0
}
```

- aggs 聚合查询 - min 最小值

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "aggs":{
  "price_min":{
   "min":{
    "field":"price"
   }
  }
 },
 "size":0
}
```

- aggs 聚合查询 - sum 求和

```
POST   http://10.25.175.108:9200/shopping/_search
{
 "aggs":{
  "price_sum":{
   "sum":{
    "field":"price"
   }
  }
 },
 "size":0
}
```

5. 文档映射

- 创建映射

```
# 此方式需要先创建索引 user1
PUT   http://10.25.175.108:9200/user1
# 创建映射
PUT   http://10.25.175.108:9200/user1/_mapping
{
 "properties":{
  "name":{
   "type":"text",
   "index":true
  },
  "sex":{
   "type":"keyword",
   "index":true
  },
  "tel":{
   "type":"keyword",
   "index": false
  }
 }
}
```

```
# 此方式不必先创建索引 user2
PUT    http://10.25.175.108:9200/user2
{
 "mappings":{
  "properties":{
                "name":{
                    "type":"keyword",
                    "index":true
                },
                "sex":{
                    "type":"keyword",
                    "index":true
                },
                "tel":{
                    "type":"keyword",
                    "index": false
                }
```

```
            }
 }
}
```

- 查看映射

```
GET    http://10.25.175.108:9200/user1,user2/_mapping
```

- 修改映射

> 映射不能修改，可以重建，把原数据给复制到新索引中

```
POST   http://10.25.175.108:9200/_reindex
{
 "source":{
  "index":"user1", # 原索引名称
  "type":"type1" # 如果是 elasticsearch6 以前的有 type 的情况下需要指定 type
 },
 "dest":{
  "index":"user2" # 新索引名称
 }
}
```

- 添加新字段

```
PUT http://10.25.175.108:9200/user1/_mapping
{
  "properties":{
    "addr":{
      "type":"keyword",
      "index":true
    }
  }
}
```

5、ElasticSearch CRUD - Kibana dev-tools

1. 服务相关

- /_cat/health 查看ES健康状况

```
GET _cat/health
```

- /_cat/nodes 查看所有节点

```
GET _cat/nodes
```

- /_cat/master 查看主节点信息

```
GET _cat/master
```

- /_cat/indices 查看所有索引

```
GET _cat/indices
```

2. 索引相关

- 创建索引 shopping

```
PUT   shopping
```

- 查看全部索引

```
GET   _cat/indices
```

- 查看指定索引 shopping

```
GET   shopping
```

- 删除索引 shopping

```
DELETE   shopping
```

3. 文档相关

- 新建文档（随机生成 ID）

```
POST shopping/_doc
{
 "name":"小米手机",
 "price":3999,
 "images":["1.jpg","2.jpg"]
}
```

- 新建文档（指定 ID）

```
POST shopping/_doc/1
{
 "name":"小米手机",
 "price":3999,
 "images":["1.jpg","2.jpg"]
}
```

- 批量新建 _bulk

```
POST shoping/_bulk
{"index":{"_id":"1"}}
{"name":"小米手机"}
{"index":{"_id":"2"}}
{"name":"华为手机"}
```

- 批量增删改

```
POST /_bulk
{"delete":{"_index":"shopping","_id":"1"}}
{"create":{"_index":"shopping","_id":"1"}}
{"name":"小米手机"}
{"index":{"_index":"shopping"}}
{"name":"华为手机"}
{"update":{"_index":"shopping","_id":"1"}}
{"doc":{"name":"锤子手机"}}
```

- 更新文档 - 全量

```
POST shopping/_doc/1
{
 "name":"华为手机",
 "price":3888
}
```

- 更新文档 - 局部

```
POST shopping/_update/1
{
 "doc":{
  "name":"苹果手机",
  "price":4000
 }
}
```

- 删除文档

```
DELETE shopping/_doc/1
```

4. 文档查询

- URL 带参数查询

```
GET shopping/_search?q=name:小米
```

- match 查询 - 全文检索

```
GET shopping/_search
{
    "query":{
        "match":{
            "name":"小米"
        }
    }
}
```

- match_all 查询全部

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

- term 词条查询

```
GET shopping/_search
{
  "query": {
    "term": {
      "name": {
        "value": "米"
      }
    }
  }
}
```

- terms 多词条查询

```
GET shopping/_search
{
  "query": {
    "terms": {
      "name": [
        "小",
        "米"
      ]
    }
  }
}
```

- _source 查询指定字段

```
GET shopping/_search
{
 "query":{
  "match_all":{}
 },
 "_source":["name"]
}
```

- from size 分页查询

```
GET shopping/_search
{
 "query":{
  "match_all":{}
 },
 "_source":["name","price"],
 "from":0,
 "size":3
}
```

- sort 排序

```
GET shopping/_search
{
 "query":{
  "match_all":{}
 },
 "_source":["name","price"],
 "sort":{
  "price":{
   "order": "asc"
  }
 }
}
```

- bool 组合查询 - must

```
GET shopping/_search
{
 "query":{
  "bool":{
   "must":[
     {
      "match":{
       "name":"小米"
      }
     },{
      "match":{
       "price":3999.0
      }
     }
    ]
  }
 }
}
```

- bool 组合查询 - should

```
GET shopping/_search
{
 "query":{
  "bool":{
   "should":[
     {
      "match":{
```

```
        "name":"小米"
       }
      },{
       "match":{
        "price":3999.0
       }
      }
     ]
   }
  }
}
```

- range 范围查询

```
GET shopping/_search
{
 "query":{
  "bool":{
   "should":[
     {
      "match":{
       "name":"小米"
      }
     },{
      "match":{
       "price":3999.0
      }
     }
    ],
    "filter":{
     "range":{
      "price":{
       "gt":2000
      }
     }
    }
   }
  }
}
```

- match_phrase 查询 - 完全匹配

```
GET shopping/_search
{
 "query":{
  "match_phrase":{
   "name":"为"
  }
 }
}
```

- fuzzy 模糊查询

```
GET shopping/_search
{
 "query":{
  "fuzzy":{
   "name":"米"
  }
 }
}
```

- highlight 高亮查询

```
GET shopping/_search
{
 "query":{
  "match_phrase":{
   "name":"为"
  }
 },
 "highlight":{
  "fields":{
   "name":{
    "pre_tags":"<font color='red'>",
    "post_tags":""
   }
  }
 }
}
```

- aggs 聚合查询 - terms（类似 group by）

```
GET shopping/_search
{
 "aggs":{
  "price_group":{
   "terms":{
    "field":"price"
   }
  }
 },
 "size":0
}
```

- aggs 聚合查询 - avg 平均值

```
GET shopping/_search
{
 "aggs":{
  "price_avg":{
   "avg":{
    "field":"price"
   }
  }
 },
 "size":0
}
```

- aggs 聚合查询 - max 最大值

```
GET shopping/_search
{
 "aggs":{
  "price_max":{
   "max":{
    "field":"price"
   }
  }
 },
 "size":0
}
```

- aggs 聚合查询 - min 最小值

```
GET shopping/_search
{
 "aggs":{
  "price_min":{
   "min":{
    "field":"price"
   }
  }
 },
 "size":0
}
```

- aggs 聚合查询 - sum 求和

```
GET shopping/_search
{
 "aggs":{
  "price_sum":{
   "sum":{
    "field":"price"
   }
  }
 },
 "size":0
}
```

5. 文档映射

- 创建映射

```
# 此方式需要先创建索引 user1
PUT user1
# 创建映射
PUT user1/_mapping
{
 "properties":{
  "name":{
   "type":"text",
```

```
    "index":true
   },
   "sex":{
    "type":"keyword",
    "index":true
   },
   "tel":{
    "type":"keyword",
    "index": false
   }
  }
}
```

```
# 此方式不必先创建索引 user2
PUT user2
{
 "mappings":{
  "properties":{
                "name":{
                    "type":"keyword",
                    "index":true
                },
                "sex":{
                    "type":"keyword",
                    "index":true
                },
                "tel":{
                    "type":"keyword",
                    "index": false
                }
            }
 }
}
```

- 查看映射

```
GET user1,user2/_mapping
```

- 修改映射

> 映射不能修改，可以添加字段，修改需要创建新索引把原数据给复制到新索引中

```
POST _reindex
{
 "source":{
  "index":"user1"
 },
 "dest":{
  "index":"user2"
 }
}
```

- 添加新字段

```
PUT user1/_mapping
{
  "properties":{
    "addr":{
      "type":"keyword",
      "index":true
    }
  }
}
```

6、分词器

1. 分词器定义

> 一个tokenizer（分词器）接收一个字符流，将之分割为独立的tokens（词元，通常是独立的单词），然后输出tokens流.
>
> 该tokenizer（分词器）还负责记录各个terms(词条)的顺序或position位置（用于phrase短语和word proximity词近邻查询），以及term（词条）所代表的原始word（单词）的start（起始）和end（结束）的character offsets（字符串偏移量）（用于高亮显示搜索的内容）.

2. 常用分词器

- standard （内置分词器）
- ik

3. 安装 ik 分词器

1. docker 安装方式

- 下载 ik 分词器

```
# 在 es 的插件目录下载 ik 分词器
cd /mydata/elasticsearch/plugins
yum install -y wget
wget https://github.com/medcl/elasticsearch-analysis-
ik/releases/download/v7.4.2/elasticsearch-analysis-ik-7.4.2.zip
```

- 解压缩

```
unzip elasticsearch-analysis-ik-7.4.2.zip -d ik
rm -rf elasticsearch-analysis-ik-7.4.2.zip
chmod -R 777 ik/
```

- 查看 es 容器

```
# 进入容器内部
docker -exec -it elasticsearch /bin/bash
# 到 es 的bin目录下
cd /usr/share/elasticsearch/bin
# 查看插件
elasticsearch-plugin list
# 退出容器
exit
```

- 重启 es 服务

```
docker restart elasticsearch
# 出现问题重启 docker 服务
# systemctl restart docker
```

2. 普通安装方式

- 下载 ik 分词器
- 解压分词器到 es 的 plugins 目录

```
cd ./plugins
unzip elasticsearch-analysis-ik-7.4.2.zip -d ik
rm -rf elasticsearch-analysis-ik-7.4.2.zip
chmod -R 777 ik/
```

- 重启 es 服务

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

4. 自定义分词规则

- 在 ik 分词器中的 config 目录下

```
cd ik/config
```

- IKAnalyzer.cfg.xml 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">hello.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>
```

- 当前目录下编写 hello.dic 文件

```
vim hello.dic
```

输入你需要的词汇

```
乔碧罗殿下
社死
电商
```

- 重启 es 服务

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

5. 测试分词器

- Postman

```
POST  10.25.175.108:9200/shopping/_analyze
{
 "analyzer":"ik_smart",
 "text":"乔碧罗殿下社死"
}
```

- Kibana

```
GET shopping/_analyze
{
 "analyzer":"ik_smart",
 "text":"乔碧罗殿下社死"
}
```

6. 分词器配置远程字典 - nginx

1. docker 安装 Nginx

- 创建目录

```
mkdir -p /mydata/nginx/conf
```

- 启动临时 nginx 容器

```
docker run -p 80:80 --name nginx -d nginx:1.10
```

- 拷贝出容器中 nginx 的配置

```
# 将 nginx 容器中的 nginx 目录复制到物理机的 /mydata/nginx/conf 目录
docker container cp -nginx:/etc/nginx /mydata/nginx/conf
mv /mydata/nginx/conf/nginx/*  /mydata/nginx/conf
rm -rf /mydata/nginx/conf/nginx
```

- 删除临时 nginx 容器

```
# 停止容器
docker stop nginx
# 删除容器
docker rm nginx
```

- 启动 nginx 容器

```
docker run -p 80:80 --name nginx \
-v /mydata/nginx/html:/usr/share/nginx/html \
-v /mydata/nginx/logs:/var/log/nginx \
-v /mydata/nginx/conf/:/etc/nginx \
-d nginx:1.10
```

- 设置容器随 docker 容器启动

```
docker update nginx --restart=always
```

- 测试 nginx

```
echo '<h1><a target="_blank" href="http://nginx.org">nginx home</a></h1>' \
>/mydata/nginx/html/index.html
```

打开地址：http://10.25.175.108

- 配置分词器的远程地址

```
<?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">hello.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">http://10.25.175.108/hello.txt</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>
```

- nginx 目录下创建 hello.txt 文件

```
echo "乔碧罗" > /mydata/nginx/html/hello.txt
```

- 测试远程分词

```
GET shopping/_analyze
{
 "analyzer":"ik_smart",
 "text":"乔碧罗社死"
}
```

7、Java 项目整合

1. 基础 API 操作

1.pom.xml

- pom.xml

```
<properties>
    <spring.version>2.3.1.RELEASE</spring.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <!-- lombok -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.16</version>
    </dependency>
    <!-- fastjson -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>fastjson</artifactId>
        <version>1.2.72</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.9</version>
    </dependency>
    <!-- elasticsearch 的客户端 -->
    <dependency>
        <groupId>org.elasticsearch.client</groupId>
        <artifactId>elasticsearch-rest-high-level-client</artifactId>
        <version>7.12.0</version>
    </dependency>
</dependencies>
```

2. API 演示

- 获取 es 连接 RestHighLevelClient

```
private static RestHighLevelClient getClient() {
    return new RestHighLevelClient(
        RestClient.builder(
            new HttpHost(
                "10.25.175.108",
                9200,
                "http")));
}
```

- 创建索引 CreateIndexRequest

```
/**
 * 索引 - 创建索引 CreateIndexRequest
 * @see org.elasticsearch.client.indices.CreateIndexRequest
 * @see IndicesClient#create
 */
private static void createIndex() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 创建索引请求
    CreateIndexRequest createIndexRequest = new CreateIndexRequest("user2");
    // 执行请求
    CreateIndexResponse indexResponse = client.indices()
        .create(createIndexRequest, RequestOptions.DEFAULT);
    // 请求结果
    boolean acknowledged = indexResponse.isAcknowledged();
    System.out.println("创建结果: " + acknowledged);
    // 关闭连接
    client.close();
}
```

- 查看索引 GetIndexRequest

```
/**
  * 索引 - 查看索引 GetIndexRequest
  * @see org.elasticsearch.client.indices.GetIndexRequest
  * @see IndicesClient#get
  */
private static void getIndex() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 获取索引请求
    GetIndexRequest getIndexRequest = new GetIndexRequest("user2");
    // 执行请求
    GetIndexResponse getIndexResponse = client.indices()
        .get(getIndexRequest, RequestOptions.DEFAULT);
    // 请求结果
    System.out.println("查询结果 aliases: " + getIndexResponse.getAliases());
    System.out.println("查询结果 mappings: " +
getIndexResponse.getMappings());
    System.out.println("查询结果 settings: " +
getIndexResponse.getSettings());
```

```
    // 关闭连接
    client.close();
}
```

- 删除索引 DeleteIndexRequest

```
/**
  * 索引 - 删除索引 DeleteIndexRequest
  * @see org.elasticsearch.action.admin.indices.delete.DeleteIndexRequest
  * @see IndicesClient#delete
  */
private static void deleteIndex() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 删除索引请求
    DeleteIndexRequest deleteIndexRequest = new DeleteIndexRequest("user2");
    // 执行请求
    AcknowledgedResponse acknowledgedResponse = client.indices()
        .delete(deleteIndexRequest, RequestOptions.DEFAULT);
    // 请求结果
    System.out.println("删除结果 " + acknowledgedResponse.isAcknowledged());
    // 关闭连接
    client.close();
}
```

- 添加文档 IndexRequest

```
/**
  * 文档 - 创建文档 IndexRequest
  * @see org.elasticsearch.action.index.IndexRequest
  * @see org.elasticsearch.action.index.IndexRequest#source
  * @see org.elasticsearch.client.RestHighLevelClient#index
  */
private static void insertDoc() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 索引请求
    IndexRequest indexRequest = new IndexRequest();
    indexRequest.index("user").id("1001"); // 指定索引及 id
    // 准备数据
    User user = new User();
    user.setName("zhangsan").setAge(20).setSex("男");
    String userJson = JSON.toJSONString(user);
    // 组装数据
    indexRequest.source(userJson, XContentType.JSON);
    // 发起添加请求
    IndexResponse response = client.index(indexRequest,
RequestOptions.DEFAULT);
    // 请求结果
    System.out.println("_index:" + response.getIndex());
    System.out.println("_id:" + response.getId());
    System.out.println("_result:" + response.getResult());
    // 关闭连接
    client.close();
}
```

- 更新文档 UpdateRequest

```
/**
  * 文档 - 更新文档 UpdateRequest
  * @see org.elasticsearch.action.update.UpdateRequest
  * @see org.elasticsearch.action.update.UpdateRequest#doc
  * @see org.elasticsearch.client.RestHighLevelClient#update
  */
private static void updateDoc() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 更新请求
    UpdateRequest updateRequest = new UpdateRequest();
    updateRequest.index("user").id("1001"); // 指定索引及 id
    // 更新内容
    updateRequest.doc(XContentType.JSON, "sex", "女");
    // 发起更新请求
    UpdateResponse response = client
        .update(updateRequest, RequestOptions.DEFAULT);
    // 请求结果
    System.out.println("_index:" + response.getIndex());
    System.out.println("_id:" + response.getId());
    System.out.println("_result:" + response.getResult());
    // 关闭连接
    client.close();
}
```

- 查看文档 GetRequest

```
/**
  * 文档 - 查看文档 GetRequest
  * @see org.elasticsearch.action.get.GetRequest
  * @see org.elasticsearch.client.RestHighLevelClient#get
  */
private static void queryDoc() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 查看请求，指定索引及 id
    GetRequest getRequest = new GetRequest().index("user").id("1001");
    // 发起查询请求
    GetResponse response = client.get(getRequest, RequestOptions.DEFAULT);
    // 请求结果
    System.out.println("_index:" + response.getIndex());
    System.out.println("_type:" + response.getType());
    System.out.println("_id:" + response.getId());
    System.out.println("_source:" + response.getSource());
    // 关闭连接
    client.close();
}
```

- 删除文档 DeleteRequest

```
/**
  * 文档 - 删除文档 DeleteRequest
  * @see org.elasticsearch.action.delete.DeleteRequest
  * @see org.elasticsearch.client.RestHighLevelClient#delete
```

```
  */
private static void deleteDoc() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 删除请求，指定索引及 id
    DeleteRequest deleteRequest = new
DeleteRequest().index("user").id("1001");
    // 发起删除请求
    DeleteResponse response = client
        .delete(deleteRequest, RequestOptions.DEFAULT);
    // 请求结果
    System.out.println(response.toString());
    // 遍历连接
    client.close();
}
```

- 批量添加文档 BulkRequest + IndexRequest

```
/**
  * 文档 - 批量创建文档 BulkRequest + IndexRequest
  * @see org.elasticsearch.action.bulk.BulkRequest
  * @see org.elasticsearch.action.index.IndexRequest
  * @see org.elasticsearch.client.RestHighLevelClient#bulk
  */
private static void batchInsertDoc() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 批量请求
    BulkRequest bulkRequest = new BulkRequest();
    // 添加记录
    bulkRequest.add(new IndexRequest()
                    .index("user")
                    .id("1001")
                    .source(XContentType.JSON,"name","张三","age",20))
        .add(new IndexRequest()
             .index("user")
             .id("1002")
             .source(XContentType.JSON,"name","李四","age",22,"sex","女"))
        .add(new IndexRequest()
             .index("user")
             .id("1003")
             .source(XContentType.JSON,"name","wq","age",18,"sex","男"))
        .add(new IndexRequest()
             .index("user")
             .id("1004")
             .source(XContentType.JSON,"name","shadow","age",22,"sex","男"))
        .add(new IndexRequest()
             .index("user")
             .id("1005")

.source(XContentType.JSON,"name","wangchirl","age",22,"sex","男"));
    // 发起批量请求
    BulkResponse response = client.bulk(bulkRequest,
RequestOptions.DEFAULT);
    // 请求结果
    System.out.println("took:" + response.getTook());
    System.out.println("items:" + response.getItems());
```

```
    // 关闭连接
    client.close();
}
```

- 批量删除文档 BulkRequest + DeleteRequest

```
    /**
     * 文档 - 批量删除文档 BulkRequest + DeleteRequest
     * @see org.elasticsearch.action.bulk.BulkRequest
     * @see org.elasticsearch.action.delete.DeleteRequest
     * @see org.elasticsearch.client.RestHighLevelClient#bulk
     */
    private static void batchDeleteDoc() throws IOException {
        // 获取客户端连接
        RestHighLevelClient client = getClient();
        // 批量请求
        BulkRequest bulkRequest = new BulkRequest();
        // 删除记录条件
        bulkRequest.add(new DeleteRequest().index("user").id("1001"));
        bulkRequest.add(new DeleteRequest().index("user").id("1002"));
        bulkRequest.add(new DeleteRequest().index("user").id("1003"));
        // 发起批量请求
        BulkResponse response = client.bulk(bulkRequest,
RequestOptions.DEFAULT);
        System.out.println("took:" + response.getTook());
        System.out.println("items:" + response.getItems());
        // 关闭连接
        client.close();
  }
```

- match_all 匹配查询 QueryBuilders#matchAllQuery

```
/**
  * 查询 - 匹配查询 SearchRequest + SearchSourceBuilder +
QueryBuilders.matchAllQuery
  * @see org.elasticsearch.action.search.SearchRequest
  * @see org.elasticsearch.search.builder.SearchSourceBuilder#query
  * @see org.elasticsearch.index.query.QueryBuilders#matchAllQuery
  * @see org.elasticsearch.action.search.SearchRequest#source
  * @see org.elasticsearch.client.RestHighLevelClient#search
  */
private static void queryMatchAll() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 搜索请求
    SearchRequest searchRequest = new SearchRequest();
    searchRequest.indices("user"); // 指定索引
    // 搜索资源构建器
    SearchSourceBuilder sourceBuilder = new SearchSourceBuilder();
    // 查询
    sourceBuilder.query(QueryBuilders.matchAllQuery());
    searchRequest.source(sourceBuilder);
    // 发起查询请求
    SearchResponse response = client
        .search(searchRequest, RequestOptions.DEFAULT);
    // 请求结果
```

```
    SearchHits hits = response.getHits();
    System.out.println("took:" + response.getTook());
    System.out.println("timeout:" + response.isTimedOut());
    System.out.println("total:" + hits.getTotalHits());
    System.out.println("maxScore:" + hits.getMaxScore());
    System.out.println("hits:============>");
    for (SearchHit hit : hits) {
        System.out.println(hit.getSourceAsString());
    }
    // 关闭连接
  client.close();
}
```

- term 词条精确查询 QueryBuilders#termQuery

```
/**
  * 查询 - 精确匹配查询 SearchRequest + SearchSourceBuilder +
QueryBuilders.termQuery
  * @see org.elasticsearch.action.search.SearchRequest
  * @see org.elasticsearch.search.builder.SearchSourceBuilder#query
  * @see org.elasticsearch.index.query.QueryBuilders#termQuery
  * @see org.elasticsearch.action.search.SearchRequest#source
  * @see org.elasticsearch.client.RestHighLevelClient#search
  */
private static void termQuery() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 搜索请求
    SearchRequest searchRequest = new SearchRequest();
    searchRequest.indices("user"); // 指定索引
    // 搜索资源构建器
    SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
    // term 查询
    searchSourceBuilder.query(QueryBuilders.termQuery("age",22));
    searchRequest.source(searchSourceBuilder);
    // 发起查询请求
    SearchResponse response = client
        .search(searchRequest, RequestOptions.DEFAULT);
    // 请求结果
    SearchHits hits = response.getHits();
    for (SearchHit hit : hits) {
        System.out.println(hit.getSourceAsString());
    }
    // 关闭连接
    client.close();
}
```

- terms 多词条精确查询 QueryBuilders#termsQuery

```
/**
  * 查询 - 精确匹配查询 SearchRequest + SearchSourceBuilder +
QueryBuilders.termsQuery
  * @see org.elasticsearch.action.search.SearchRequest
  * @see org.elasticsearch.search.builder.SearchSourceBuilder#query
  * @see org.elasticsearch.index.query.QueryBuilders#termsQuery
  * @see org.elasticsearch.action.search.SearchRequest#source
```

```
     * @see org.elasticsearch.client.RestHighLevelClient#search
     */
    private static void termsQuery() throws IOException {
        // 获取客户端连接
        RestHighLevelClient client = getClient();
        // 搜索请求
        SearchRequest searchRequest = new SearchRequest();
        searchRequest.indices("user"); // 指定索引
        // 搜索资源构建器
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        // terms 查询
        searchSourceBuilder.query(QueryBuilders.termsQuery("age",22,"age",20));
        searchRequest.source(searchSourceBuilder);
        // 发起查询请求
        SearchResponse response = client.search(searchRequest,
    RequestOptions.DEFAULT);
        // 请求结果
        SearchHits hits = response.getHits();
        for (SearchHit hit : hits) {
            System.out.println(hit.getSourceAsString());
        }
        // 关闭连接
        client.close();
    }
```

- from size 分页查询 SearchSourceBuilder#from/size

```
    /**
     * 文档分页查询 - from/size 查询 - 分页查询 SearchRequest + SearchSourceBuilder
     * @see org.elasticsearch.action.search.SearchRequest
     * @see org.elasticsearch.search.builder.SearchSourceBuilder#query
     * @see org.elasticsearch.search.builder.SearchSourceBuilder#from
     * @see org.elasticsearch.search.builder.SearchSourceBuilder#size
     * @see org.elasticsearch.action.search.SearchRequest#source
     * @see org.elasticsearch.client.RestHighLevelClient#search
     */
    private static void pageQuery() throws IOException {
        // 获取客户端连接
        RestHighLevelClient client = getClient();
        // 搜索请求
        SearchRequest searchRequest = new SearchRequest();
        searchRequest.indices("user"); // 指定索引
        // 搜索资源构建器
        SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
        searchSourceBuilder.query(QueryBuilders.matchAllQuery());
        // 分页参数
        searchSourceBuilder.from(0);
        searchSourceBuilder.size(2);
        searchRequest.source(searchSourceBuilder);
        // 发起查询请求
        SearchResponse response = client
            .search(searchRequest, RequestOptions.DEFAULT);
        // 请求结果
        SearchHits hits = response.getHits();
        for (SearchHit hit : hits) {
            System.out.println(hit.getSourceAsString());
        }
```

```
    // 关闭连接
    client.close();
}
```

- sort 查询排序 SearchSourceBuilder#sort

```
/**
  * 文档排序查询 - sort  查询 - 排序查询 SearchRequest + SearchSourceBuilder
  * @see org.elasticsearch.action.search.SearchRequest
  * @see org.elasticsearch.search.builder.SearchSourceBuilder#query
  * @see org.elasticsearch.search.builder.SearchSourceBuilder#sort
  * @see org.elasticsearch.action.search.SearchRequest#source
  * @see org.elasticsearch.client.RestHighLevelClient#search
  */
private static void sortQuery() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 搜索请求
    SearchRequest searchRequest = new SearchRequest();
    searchRequest.indices("user"); // 指定索引
    // 搜索资源构建器
    SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
    searchSourceBuilder.query(QueryBuilders.matchAllQuery());
    // 排序参数 sort
    searchSourceBuilder.sort("age", SortOrder.DESC);
    searchRequest.source(searchSourceBuilder);
    // 发起查询请求
    SearchResponse response = client
        .search(searchRequest, RequestOptions.DEFAULT);
    // 请求结果
    SearchHits hits = response.getHits();
    for (SearchHit hit : hits) {
        System.out.println(hit.getSourceAsString());
    }
    // 关闭连接
    client.close();
}
```

- bool 组合查询 QueryBuilders.boolQuery

```
/**
  * 查询 - 组合查询 SearchRequest + SearchSourceBuilder +
BoolQueryBuilder(QueryBuilders.boolQuery)
  * @see org.elasticsearch.action.search.SearchRequest
  * @see org.elasticsearch.search.builder.SearchSourceBuilder
  * @see org.elasticsearch.index.query.QueryBuilders#boolQuery
  * @see org.elasticsearch.index.query.BoolQueryBuilder#must
  * @see org.elasticsearch.index.query.BoolQueryBuilder#mustNot
  * @see org.elasticsearch.index.query.BoolQueryBuilder#should
  * @see org.elasticsearch.action.search.SearchRequest#source
  * @see org.elasticsearch.client.RestHighLevelClient#search
  */
private static void combinationQuery() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 搜索请求
```

```
    SearchRequest searchRequest = new SearchRequest();
    searchRequest.indices("user"); // 指定索引
    // 搜索资源构建器
    SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
    // 组合查询 boolQuery
    BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();
    // must
    boolQueryBuilder.must(QueryBuilders.matchQuery("age","22"));
    // must not
    boolQueryBuilder.mustNot(QueryBuilders.matchQuery("nage","shadow"));
    // should
    boolQueryBuilder.should(QueryBuilders.matchQuery("sex","男"));
    searchSourceBuilder.query(boolQueryBuilder);
    searchRequest.source(searchSourceBuilder);
    // 发起查询请求
    SearchResponse response = client
        .search(searchRequest, RequestOptions.DEFAULT);
    // 请求结果
    SearchHits hits = response.getHits();
    for (SearchHit hit : hits) {
        System.out.println(hit.getSourceAsString());
    }
    // 关闭连接
    client.close();
}
```

- range 范围查询 QueryBuilders#rangeQuery

```
/**
  * 查询 - 范围查询 SearchRequest + SearchSourceBuilder +
RangeQueryBuilder(QueryBuilders.rangeQuery)
  * @see org.elasticsearch.action.search.SearchRequest
  * @see org.elasticsearch.search.builder.SearchSourceBuilder
  * @see org.elasticsearch.index.query.QueryBuilders#rangeQuery
  * @see org.elasticsearch.index.query.RangeQueryBuilder#lt
  * @see org.elasticsearch.index.query.RangeQueryBuilder#gt
  * @see org.elasticsearch.index.query.RangeQueryBuilder#lte
  * @see org.elasticsearch.index.query.RangeQueryBuilder#gte
  * @see org.elasticsearch.action.search.SearchRequest#source
  * @see org.elasticsearch.client.RestHighLevelClient#search
  */
private static void rangeQuery() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 搜索请求
    SearchRequest searchRequest = new SearchRequest();
    searchRequest.indices("user"); // 指定索引
    // 搜索资源构建器
    SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
    // rangeQuery
    RangeQueryBuilder rangeQuery = QueryBuilders.rangeQuery("age");
    // lte
    rangeQuery.lte(21);
    // gt
    rangeQuery.gt(20);
    searchSourceBuilder.query(rangeQuery);
    searchRequest.source(searchSourceBuilder);
```

```
    // 发起查询请求
    SearchResponse response = client
        .search(searchRequest, RequestOptions.DEFAULT);
    // 请求结果
    SearchHits hits = response.getHits();
    hits.forEach(item -> {
        System.out.println(item.getSourceAsString());
    });
    // 关闭连接
    client.close();
}
```

- fuzzy 模糊查询 QueryBuilders#fuzzyQuery

```
/**
  * 查询 - 模糊查询 SearchRequest + SearchSourceBuilder +
QueryBuilders.fuzzyQuery
  * @see org.elasticsearch.action.search.SearchRequest
  * @see org.elasticsearch.search.builder.SearchSourceBuilder
  * @see org.elasticsearch.index.query.QueryBuilders#fuzzyQuery
  * @see org.elasticsearch.action.search.SearchRequest#source
  * @see org.elasticsearch.client.RestHighLevelClient#search
  */
private static void fuzzyQuery() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 搜索请求
    SearchRequest searchRequest = new SearchRequest();
    searchRequest.indices("user"); // 指定索引
    // 搜索资源构建器
    SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
    // fuzzyQuery 模糊查询
    searchSourceBuilder.query(QueryBuilders.fuzzyQuery("name","shadow")
                              .fuzziness(Fuzziness.ONE));
    searchRequest.source(searchSourceBuilder);
    // 发起查询请求
    SearchResponse response = client
        .search(searchRequest, RequestOptions.DEFAULT);
    // 请求结果
    SearchHits hits = response.getHits();
    hits.forEach(item -> System.out.println(item.getSourceAsString()));
    // 关闭连接
    client.close();
}
```

- highlight 高亮查询 HighlightBuilder

```
/**
  * 文档高亮（精确）查询 - termsQuery 与 termQuery 的区别是支持多个精确条件
  * 查询 - 高亮查询 SearchRequest + SearchSourceBuilder + HighlightBuilder
  * @see org.elasticsearch.action.search.SearchRequest
  * @see org.elasticsearch.search.builder.SearchSourceBuilder
  * @see org.elasticsearch.search.fetch.subphase.highlight.HighlightBuilder
  * @see
org.elasticsearch.search.fetch.subphase.highlight.HighlightBuilder#preTags
```

```
  * @see
org.elasticsearch.search.fetch.subphase.highlight.HighlightBuilder#postTags
  * @see
org.elasticsearch.search.fetch.subphase.highlight.HighlightBuilder#field
  * @see org.elasticsearch.action.search.SearchRequest#source
  * @see org.elasticsearch.client.RestHighLevelClient#search
  */
private static void highlightQuery() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 搜索请求
    SearchRequest searchRequest = new SearchRequest();
    searchRequest.indices("user"); // 指定索引
    // 搜索资源构建器
    SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
    searchSourceBuilder.query(QueryBuilders

.termsQuery("name","shadow","name","wangchirl"));
    // HighlightBuilder 高亮查询
    HighlightBuilder highlightBuilder = new HighlightBuilder();
    highlightBuilder.preTags("<font color='red'>");
    highlightBuilder.postTags(">");
    highlightBuilder.field("name");
    searchSourceBuilder.highlighter(highlightBuilder);
    searchRequest.source(searchSourceBuilder);
    // 发起查询请求
    SearchResponse response = client
        .search(searchRequest, RequestOptions.DEFAULT);
    // 请求结果
    SearchHits hits = response.getHits();
    for (SearchHit hit : hits) {
        String sourceAsString = hit.getSourceAsString();
        System.out.println(sourceAsString);
        Map<String, HighlightField> highlightFields =
hit.getHighlightFields();
        System.out.println(highlightFields);
    }
    // 关闭连接
    client.close();
}
```

- terms 聚合分组查询 AggregationBuilders#terms

```
/**
  * 文档聚合查询 - 分组 AggregationBuilders#terms
  * 查询 - 分组查询 SearchRequest + SearchSourceBuilder +
AggregationBuilders.terms
  * @see org.elasticsearch.action.search.SearchRequest
  * @see org.elasticsearch.search.builder.SearchSourceBuilder#aggregation
  * @see org.elasticsearch.search.aggregations.AggregationBuilders#terms
  * @see org.elasticsearch.action.search.SearchRequest#source
  * @see org.elasticsearch.client.RestHighLevelClient#search
  */
private static void groupQuery() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 搜索请求
```

```
    SearchRequest searchRequest = new SearchRequest();
    searchRequest.indices("user"); // 指定索引
    // 搜索资源构建器
    SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
    // 聚合索引 AggregationBuilders#terms 类似于 group  by
    searchSourceBuilder.aggregation(AggregationBuilders
                                    .terms("age_group").field("age"));
    searchRequest.source(searchSourceBuilder);
    // 发起查询请求
    SearchResponse response = client
        .search(searchRequest, RequestOptions.DEFAULT);
    // 请求结果
    Aggregations aggregations = response.getAggregations();
    Map<String, Aggregation> aggregationMap = aggregations.asMap();
    aggregationMap.forEach((k,v) -> {
        System.out.println(k);
        List<? extends Terms.Bucket> buckets = ((ParsedLongTerms)
v).getBuckets();
        for (Terms.Bucket bucket : buckets) {
            System.out.println(bucket.getKeyAsNumber());
        }
    });
    // 关闭连接
    client.close();
}
```

- max/min/avg/sum 聚合查询

```
/**
  * 文档聚合查询 - 最大、最小、平均、求和
* 查询 - 聚合查询 SearchRequest + SearchSourceBuilder +
AggregationBuilders.max/min/avg/sum
  * @see org.elasticsearch.action.search.SearchRequest
  * @see org.elasticsearch.search.builder.SearchSourceBuilder#aggregation
  * @see org.elasticsearch.search.aggregations.AggregationBuilders#terms
  * @see org.elasticsearch.search.aggregations.AggregationBuilders#max
  * @see org.elasticsearch.search.aggregations.AggregationBuilders#min
  * @see org.elasticsearch.search.aggregations.AggregationBuilders#avg
  * @see org.elasticsearch.search.aggregations.AggregationBuilders#sum
  * @see org.elasticsearch.action.search.SearchRequest#source
  * @see org.elasticsearch.client.RestHighLevelClient#search
  */
private static void maxQuery() throws IOException {
    // 获取客户端连接
    RestHighLevelClient client = getClient();
    // 搜索请求
    SearchRequest searchRequest = new SearchRequest();
    searchRequest.indices("user"); // 指定索引
    // 搜索资源构建器
    SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
    // 聚合查询 AggregationBuilders
    searchSourceBuilder.aggregation(AggregationBuilders
                                    .max("maxAge").field("age"));
    searchRequest.source(searchSourceBuilder);
    // 发起查询请求
    SearchResponse response = client
        .search(searchRequest, RequestOptions.DEFAULT);
```

```
    // 请求结果
    SearchHits hits = response.getHits();
    System.out.println(response);
    Aggregations aggregations = response.getAggregations();
    Map<String, Aggregation> aggregationMap = aggregations.asMap();
    aggregationMap.forEach((k,v) -> {
        System.out.println(k + " => " + ((ParsedMax)v).getValue());
    });
    // 关闭连接
    client.close();
}
```

2. 模拟JD首页搜索

1. pom.xml

- pom.xml

```
<properties>
    <spring.version>2.3.1.RELEASE</spring.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.10</version>
    </dependency>
    <!-- 爬虫工具 -->
    <dependency>
        <groupId>org.jsoup</groupId>
        <artifactId>jsoup</artifactId>
        <version>1.9.2</version>
    </dependency>
    <!-- elasticsearch 的客户端 -->
    <dependency>
        <groupId>org.elasticsearch.client</groupId>
        <artifactId>elasticsearch-rest-high-level-client</artifactId>
        <version>7.12.0</version>
```

```
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.72</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.11.0</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.16</version>
        </dependency>
    </dependencies>
```

2. 业务代码

- controller

```
@RestController
public class ContentController {

    @Autowired
    private ContentService contentService;

    @GetMapping("/parse/{keyword}")
    public Boolean parse(@PathVariable("keyword") String keyword) throws
Exception {
        return contentService.parseContent(keyword);
    }

    @GetMapping("/search/{keywords}/{pageNo}/{pageSize}")
    public List<Map<String, Object>> search(@PathVariable("keywords") String
keywords
            , @PathVariable("pageNo") int pageNo, @PathVariable("pageSize")
int pageSize) throws Exception {
        return contentService.searchPageForHighlight(keywords, pageNo,
pageSize);
    }
}
```

```
@Controller
public class IndexController {

    @GetMapping({"/", "/index"})
    public String index() {
        return "index";
    }
}
```

- service

```java
@Service
public class ContentService {

    @Autowired
    private RestHighLevelClient restHighLevelClient;

    // 1. 解析数据放入到es索引当中
    public Boolean parseContent(String keywords) throws Exception {
        List<Content> contents = new HtmlParseUtil().parseJD(keywords);
        BulkRequest bulkRequest = new BulkRequest();
        bulkRequest.timeout("2m");

        for (Content content : contents) {
            bulkRequest.add(
                    new IndexRequest(Constants.INDEX)
                            .source(JSON.toJSONString(content),
XContentType.JSON));
        }

        BulkResponse bulk = restHighLevelClient.bulk(bulkRequest,
RequestOptions.DEFAULT);
        return !bulk.hasFailures();
    }

    // 2. 查询这些数据实现搜索功能
    public List<Map<String, Object>> searchPage(String keywords, int pageNo,
int pageSize) throws IOException {
        if (pageNo <= 1) {
            pageNo = 1;
        }
        // 条件搜索
        SearchRequest searchRequest = new SearchRequest(Constants.INDEX);
        SearchSourceBuilder sourceBuilder = new SearchSourceBuilder();
        // 分页
        sourceBuilder.from(pageNo);
        sourceBuilder.size(pageSize);
        // 精准匹配
        TermQueryBuilder termQueryBuilder = QueryBuilders.termQuery("title",
keywords);
        sourceBuilder.query(termQueryBuilder);
        sourceBuilder.timeout(new TimeValue(60, TimeUnit.SECONDS));
        // 执行搜索
        searchRequest.source(sourceBuilder);
        SearchResponse searchResponse =
restHighLevelClient.search(searchRequest, RequestOptions.DEFAULT);
        // 解析结果
        List<Map<String, Object>> list = new ArrayList<>();
        for (SearchHit documentField : searchResponse.getHits().getHits()) {
            list.add(documentField.getSourceAsMap());
        }
        return list;
    }

    // 2. 查询这些数据实现搜索功能
    public List<Map<String, Object>> searchPageForHighlight(String keywords,
int pageNo, int pageSize) throws Exception {
        if (pageNo <= 1) {
```

```
            pageNo = 1;
        }
        // 条件搜索
        SearchRequest searchRequest = new SearchRequest(Constants.INDEX);
        SearchSourceBuilder sourceBuilder = new SearchSourceBuilder();
        // 分页
        sourceBuilder.from(pageNo);
        sourceBuilder.size(pageSize);
        // 精准匹配
        TermQueryBuilder termQueryBuilder = QueryBuilders.termQuery("title",
keywords);
        sourceBuilder.query(termQueryBuilder);
        sourceBuilder.timeout(new TimeValue(60, TimeUnit.SECONDS));
        // 高亮
        HighlightBuilder highlightBuilder = new HighlightBuilder();
        highlightBuilder.field("title");
        highlightBuilder.requireFieldMatch(false);   // 多个高亮显示!
        highlightBuilder.preTags("<span style='color:red'>");
        highlightBuilder.postTags("</span>");
        sourceBuilder.highlighter(highlightBuilder);

        // 执行搜索
        searchRequest.source(sourceBuilder);
        SearchResponse searchResponse =
restHighLevelClient.search(searchRequest, RequestOptions.DEFAULT);
        // 解析结果
        List<Map<String, Object>> list = new ArrayList<>();
        for (SearchHit documentField : searchResponse.getHits().getHits()) {
            Map<String, HighlightField> highlightFields =
documentField.getHighlightFields();
            HighlightField title = highlightFields.get("title");
            Map<String, Object> sourceAsMap =
documentField.getSourceAsMap();   // 原来的结果
            // 解析高亮的字段,  将原来的字段换为我们高亮的字段即可!
            if (title != null) {
                Text[] fragments = title.fragments();
                StringBuilder n_title = new StringBuilder();
                for (Text text : fragments) {
                    n_title.append(text);
                }
                sourceAsMap.put("title", n_title.toString());
            }
            list.add(sourceAsMap);
        }
        return list;
    }
}
```

- entity

```
@Data
@Accessors(chain = true)
@AllArgsConstructor
@NoArgsConstructor
public class Content implements Serializable {

    private String title;
    private String imgUrl;
    private String price;

}
```

- config

```
@Configuration
public class EsConfig {

    @Bean
    public RestHighLevelClient restHighLevelClient() {
        return new RestHighLevelClient(RestClient
                .builder(new HttpHost("localhost",9200,"http")));
    }
}
```

- utils

```
@Component
public class HtmlParseUtil {

    public List<Content> parseJD(String keyword) throws Exception{
        String url = Constants.JDURL + "?keyword=" + keyword;

        Document document = Jsoup.parse(new URL(url), 30000);
        Element divElement = document.getElementById("J_goodsList");

        Elements lis = divElement.getElementsByTag("li");

        ArrayList<Content> contents = new ArrayList<>();

        for (Element li : lis) {
            String title = li.getElementsByClass("p-name").eq(0).text();
            String img = li.getElementsByTag("img").eq(0).attr("data-lazy-
img");
            String price = li.getElementsByClass("p-price").eq(0).text();
            contents.add(new Content(title,img,price));
        }
        return contents;
    }
}
```

```
public interface Constants {

    String JDURL = "https://search.jd.com/Search";

    String INDEX = "test_jd_goods";

}
```

- index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">

<head>
    <meta charset="utf-8"/>
    <title>狂神说Java-ES仿京东实战</title>
    <link rel="stylesheet" th:href="@{/css/style.css}"/>
</head>

<body class="pg">
<div class="page" id="app">
    <div id="mallPage" class=" mallist tmall- page-not-market ">

        <!-- 头部搜索 -->
        <div id="header" class=" header-list-app">
            <div class="headerLayout">
                <div class="headerCon ">
                    <!-- Logo-->
                    <h1 id="mallLogo">
                        <img th:src="@{/images/jdlogo.png}" alt="">
                    </h1>

                    <div class="header-extra">

                        <!--搜索-->
                        <div id="mallSearch" class="mall-search">
                            <form name="searchTop" class="mallSearch-form
clearfix">
                                <fieldset>
                                    <legend>天猫搜索</legend>
                                    <div class="mallSearch-input clearfix">
                                        <div class="s-combobox" id="s-
combobox-685">
                                            <div class="s-combobox-input-
wrap">
                                                <input v-model="keyword"
type="text" autocomplete="off" value="dd" id="mq"
                                                       class="s-combobox-
input" aria-haspopup="true">


                                            <button type="submit"
@click.prevent="searchKey" id="searchbtn">搜索</button>

                                </fieldset>
                            </form>
```

```
                        <ul class="relKeyTop">
                            <li><a>狂神说Java</a></li>
                            <li><a>狂神说前端</a></li>
                            <li><a>狂神说Linux</a></li>
                            <li><a>狂神说大数据</a></li>
                            <li><a>狂神聊理财</a></li>
                        </ul>




        <!-- 商品详情页面 -->
        <div id="content">
            <div class="main">
                <!-- 品牌分类 -->
                <form class="navAttrsForm">
                    <div class="attrs j_NavAttrs" style="display:block">
                        <div class="brandAttr j_nav_brand">
                            <div class="j_Brand attr">
                                <div class="attrKey">
                                    品牌

                                <div class="attrValues">
                                    <ul class="av-collapse row-2">
                                        <li><a href="#"> 狂神说 </a></li>
                                        <li><a href="#"> Java </a></li>
                                    </ul>



                </form>

                <!-- 排序规则 -->
                <div class="filter clearfix">
                    <a class="fSort fSort-cur">综合<i class="f-ico-arrow-d">
</a>
                    <a class="fSort">人气<i class="f-ico-arrow-d"></a>
                    <a class="fSort">新品<i class="f-ico-arrow-d"></a>
                    <a class="fSort">销量<i class="f-ico-arrow-d"></a>
                    <a class="fSort">价格<i class="f-ico-triangle-mt"><i
class="f-ico-triangle-mb"></a>


                <!-- 商品详情 -->
                <div class="view grid-nosku">

                    <div class="product" v-for="result in results">
                        <div class="product-iWrap">
                            <!--商品封面-->
                            <div class="productImg-wrap">
                                <a class="productImg">
                                    <img :src="result.imgUrl">
                                </a>

                            <!--价格-->
```

```
                            <p class="productPrice">
                                <em>{{result.price}}</em>
                            </p>
                            <!--标题-->
                            <p class="productTitle">
                                <a v-html="result.title"> </a>
                            </p>
                            <!-- 店铺名 -->
                            <div class="productShop">
                                <span>店铺： 狂神说Java </span>

                            <!-- 成交信息 -->
                            <p class="productStatus">
                                <span>月成交<em>999笔</em></span>
                                <span>评价 <a>3</a></span>
                            </p>
```

```
<script th:src="@{/js/axios.min.js}"></script>
<script th:src="@{/js/vue.min.js}"></script>
<script>
    new Vue({
        el: '#app',
        data:{
            keyword: '',
            results:[]
        },
        methods:{
            searchKey(){
                let keyword=this.keyword;
                console.log(keyword);
                ///search/{keyword}/{pageNum}/{pageSize}
                axios.get('search/'+keyword+"/1/20").then(response=>{
                    console.log(response);
                    this.results=response.data;
                });
            }
        }

    })
</script>

</body>
</html>
```