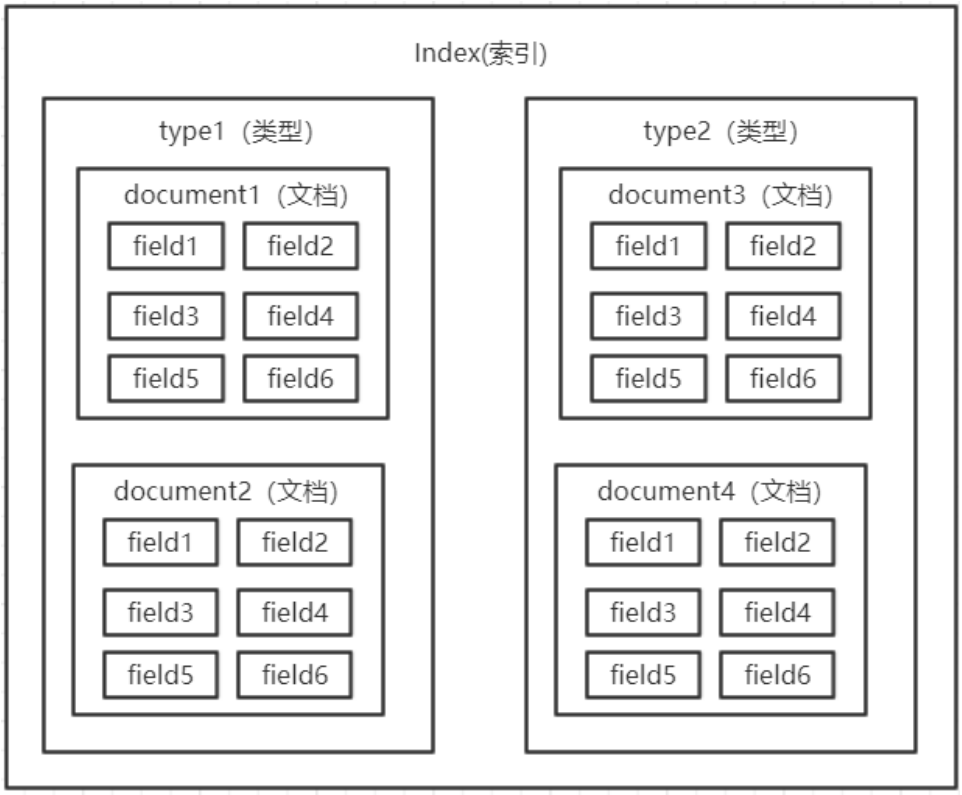
ELK

1.1、ElasticSearch

1.1.1、名词解释

1. 基本术语：
   之前版本的结构
   Es6.0 之后：一个索引中只有一个 type



- 文档 document
  用户存储在 es 中的数据文档
  - 元数据
    _index:文档所在索引名称
    _type:文档所在类型名称
    _id:文档唯一 id
    _uid:组合 id，由_type 和_id 组成（6.x 后，_type 不再起作用，同_id）
    _source:文档的原始 Json 数据，包括每个字段的内容
    _all:将所有字段内容整合起来，默认禁用（用于对所有字段内容检索）
- 索引 Index
  由具有相同字段的文档列表组成，用于定义字段名和字段值，一个集群或

elasticsearch 由多个索引组成，例如可以按照日期生成多个索引，方便数据搜索

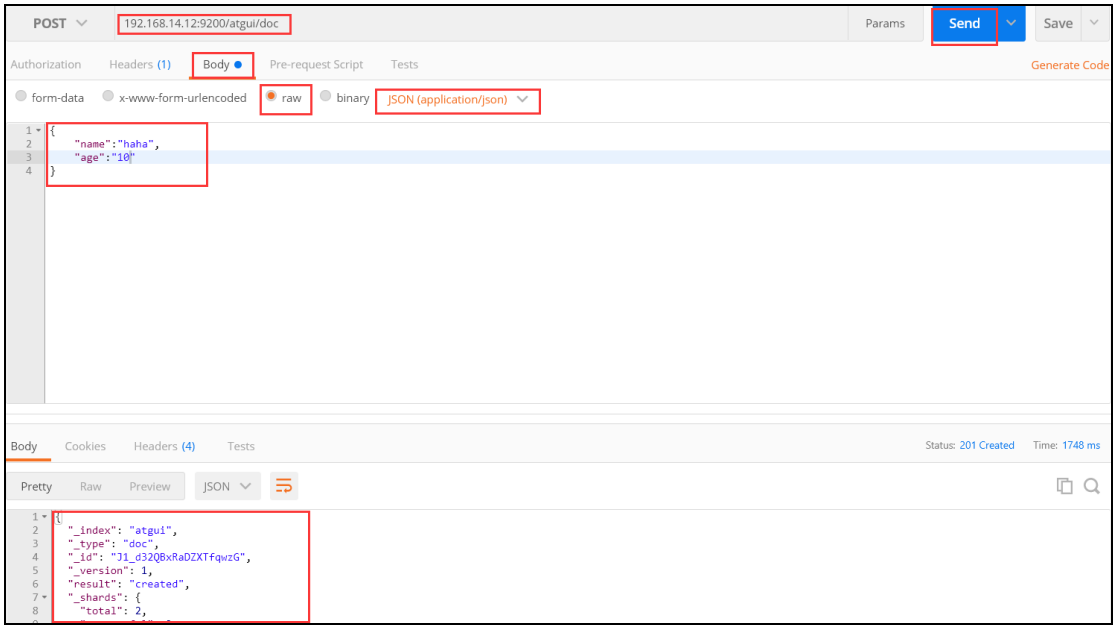
- 类型 Type
  具有相同特征文档的集合（ES6 之后一个索引中只能定义一个 type）
- 字段 Field
  具有相同特性数据名称

类型名称	组成
字符串	text、keyword（不分词）
数值型	long、integer、short、byte、double、float、half_float、scaled_float(长度短)
布尔	boolean
日期	Date
二进制	binary
范围类型	Integer_range、float_range、long_range、double_range、date_range(做数据范围查询)
坐标	附近的人

2. 集群术语：
   - 节点 Node
     一个 Elasticsearch 的运行实例，时集群构成的基本单元
   - 集群 cluster
     由一个或多个节点组成，对外提供服务

1.1.2、Elasticsearch 的 Rest API

- REST 访问 ES 方式（需要 Http Method、URI）
  1. 浏览器（postman）



2. Linux 命令行

请求：

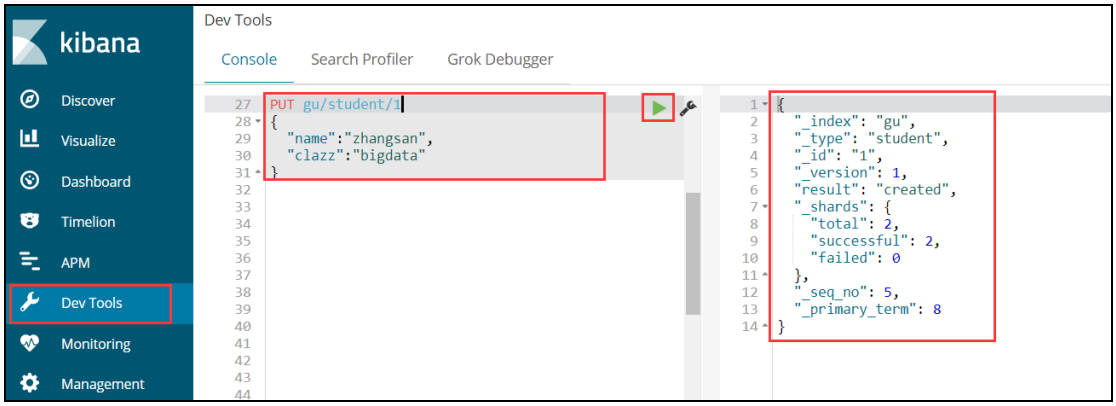
[root@localcurl -XPOST 'http://192.168.14.12:9200/atguig/doc'
-i -H
"Content-Type:application/json"
-d
'{"name":"haha","age":"10"}'

相应：

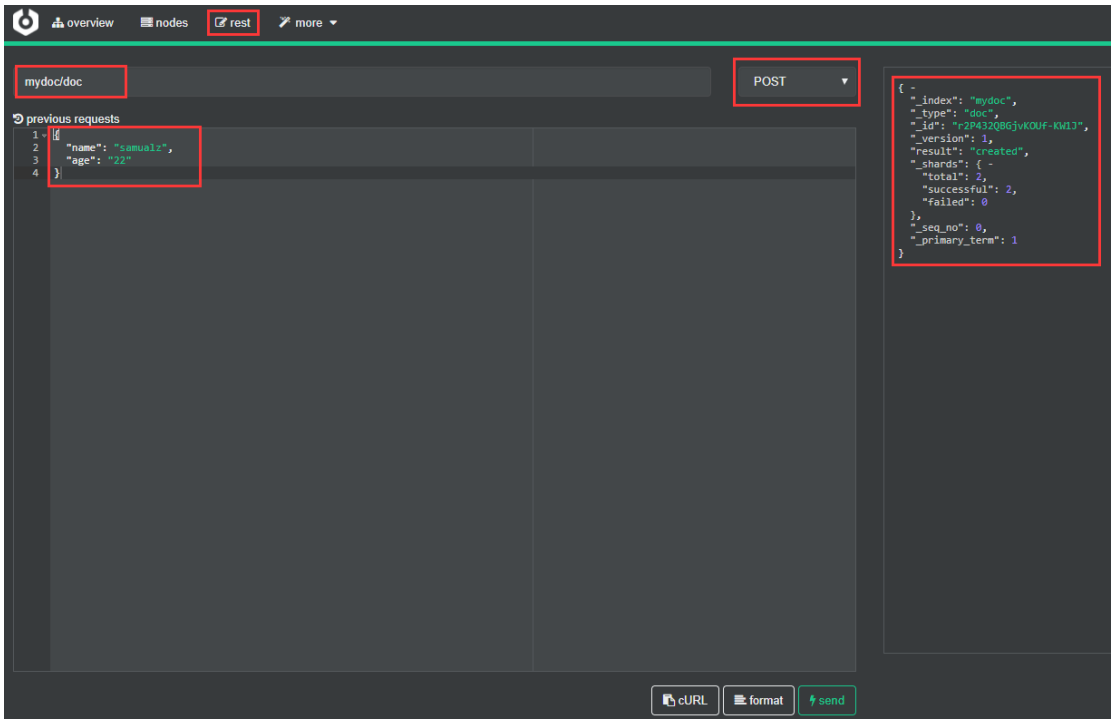
HTTP/1.1 201 Created
Location: /atguig/doc/KF_t32QBxRaDZXTftAxg
content-type: application/json; charset=UTF-8
content-length: 172

{"_index":"atguig","_type":"doc","_id":"KF_t32QBxRaDZXTftAxg","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":0,"_primary_term":1}

3. Kibana 的 Dev Tools



4. Cerebro 插件



- ES 状态查看命令

  语法：ip:post/_cat/[args](?v|?format=json&pretty)

  （?v 表示显示字段说明,?format=json&pretty 表示显示成 json 格式）

  1、查看所有索引

  GET _cat/indices?v

  2、查看 es 集群状态

  GET _cat/health?v

  3、集群节点健康查看

  GET _cat/nodes?v

  4、列出倒叙索引

  GET _cat/segment?v

- 查看集群的状态

语法：GET _cluster/(args)[?v | ?format=json&pretty)

（?v 表示显示字段说明,?format=json&pretty 表示显示成 json 格式）

- 索引操作

1、添加

语法：PUT　index 名称

2、查看索引信息

语法：GET　index 名称

3、删除索引

语法：DELETE　　index 名称

4、查看索引状态

语法：HEAD　index 名称

语法：GET　index 名称/_status

- 文档操作

1、添加和修改

语法：(PUT|POST)　　index 名称/type 名称/[id]?

不添加 id 会自动生成 id

2、删除

语法：DELETE　　index 名称/type 名称/[id]

4、查看

语法：GET　index 名称/type 名称/[id]

1.1.3、正排索引和倒排索引

- 正排索引

记录文档 Id 到文档内容、单词的关联关系

docid	content
1	尚硅谷是最好的培训机构
2	php 是世界上最好的语言
3	尚硅谷是如何诞生的

- 倒排索引

记录单词到文档 id 的关联关系，包含：

单词词典（Term DicTionary）：记录所有文档的单词，一般比较大

倒排索引（Posting List）：记录单词倒排列表的关联信息

例如：尚硅谷

1、Term Dictionary

尚硅谷

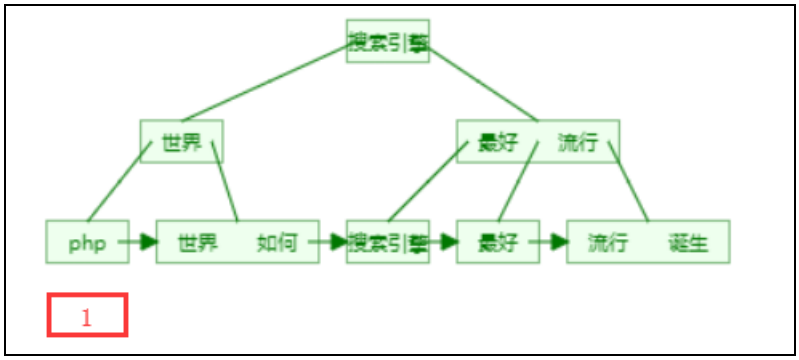
2、Posting List

DocId	TF	Position	Offset
1	1	0	<0,2>
3	1	0	<0,2>

DocId：文档 id，文档的原始信息
TF：单词频率，记录该词再文档中出现的次数，用于后续相关性算分
Position：位置，记录 Field 分词后，单词所在的位置，从 0 开始
Offset：偏移量，记录单词在文档中开始和结束位置，用于高亮显示等

3、内存结构

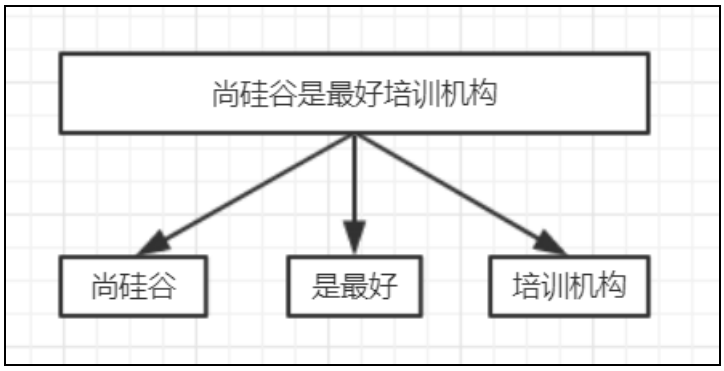
B+Tree



每个文档字段都有自己的倒排索引

1.1.4、分词

分词是指将文本转换成一系列单词（term or token）的过程，也可以叫做文本分析，在 es 里面称为 Analysis



- 分词机制

Character Filter	对原始文本进行处理	例：去除 html 标签、特殊字符等
Tokenizer	将原始文本进行分词	例：培训机构-->培训，机构
Token Filters	分词后的关键字进行加工	例：转小写、删除语气词、近义词和同义词等

- 分词 API

1、直接指定测试（指定分词器）

Request:

```
POST _analyze
{
  "analyzer": "standard",
  "text":"hello 1111"
}
```

Response:

```
{
  "tokens": [
    {
      "token": "hello",                #分词
      "start_offset": 0,               #开始偏移
      "end_offset": 5,                 #结束偏移
      "type": "<ALPHANUM>",            #单词类型
      "position": 0                    #位置
    },
    {
      "token": "world",
      "start_offset": 6,
      "end_offset": 11,
      "type": "<NUM>",
      "position": 1
    }
  ]
}
```

2、针对索引的字段进行分词测试（利用该字段的分词器）

Request：

```
POST atguigu/_analyze
{
  "field": "name",
  "text":"hello world"
}
```

Response:

```
{
  "tokens": [
    {
      "token": "hello",
      "start_offset": 0,
      "end_offset": 5,
      "type": "<ALPHANUM>",
      "position": 0
    },
    {
```

```
        "token": "world",
        "start_offset": 6,
        "end_offset": 11,
        "type": "<ALPHANUM>",
        "position": 1
      }
   ]
}
```

3、自定义分词器

Request:

```
POST _analyze
{
   "tokenizer": "standard",
   "filter": ["lowercase"],
   "text":"Hello WORLD"
}
```

Response:

```
{
   "tokens": [
      {
         "token": "hello",
         "start_offset": 0,
         "end_offset": 5,
         "type": "<ALPHANUM>",
         "position": 0
      },
      {
         "token": "world",
         "start_offset": 6,
         "end_offset": 11,
         "type": "<ALPHANUM>",
         "position": 1
      }
   ]
}
```

- Elasticsearch 自带的分词器

分词器（Analyzer）	特点
Standard（es 默认）	支持多语言，按词切分并做小写处理
Simple	按照非字母切分，小写处理
Whitespace	按照空格来切分
Stop	去除语气助词，如 the、an、的、这等

Keyword	不分词
Pattern	正则分词，默认\w+,即非字词符号做分割符
Language	常见语言的分词器（30+）

- 中文分词

分词器名称	介绍	特点	地址
IK	实现中英文单词切分	自定义词库	https://github.com/medcl/elasticsearch-analysis-ik
Jieba	python 流行分词系统，支持分词和词性标注	支持繁体、自定义、并行分词	http://github.com/sing1ee/elasticsearch-jieba-plugin
Hanlp	由一系列模型于算法组成的 java 工具包	普及自然语言处理在生产环境中的应用	https://github.com/hankcs/HanLP
THULAC	清华大学中文词法分析工具包	具有中文分词和词性标注功能	https://github.com/microbun/elasticsearch-thulac-plugin

- Character Filters

  在进行 Tokenizer 之前对原始文本进行处理，如增加、删除或替换字符等

HTML Strip	去除 html 标签和转换 html 实体
Mapping	字符串替换操作
Pattern Replace	正则匹配替换

注意：进行处理后，会影响后续 tokenizer 解析的 position 和 offset

Request：

```
POST _analyze
{
  "tokenizer": "keyword",
  "char_filter": ["html_strip"],
  "text":"<div><h1>B<sup>+</sup>Trees</h1></div>"
}
```

Response:

```
{
  "tokens": [
    {
      "token": """


B+Trees


""",
```

```
      "start_offset": 0,
      "end_offset": 38,
      "type": "word",
      "position": 0
    }
  ]
}
```

- Token Filter

  对输出的单词（term）进行增加、删除、修改等操作

Lowercase	将所有 term 转换为小写
stop	删除 stop words
NGram	和 Edge NGram 连词分割
Synonym	添加近义词的 term

Request:

```
POST _analyze
{
  "tokenizer": "standard",
  "text":"a Hello World",
  "filter": [
      "stop",
      "lowercase",
      {
        "type":"ngram",
        "min_gram":3,
        "max_gram":4

      }
    ]
}
```

Response:

```
{
  "tokens": [
    {
      "token": "hel",
      "start_offset": 2,
      "end_offset": 7,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "hell",
```

```
      "start_offset": 2,
      "end_offset": 7,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "ell",
      "start_offset": 2,
      "end_offset": 7,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "ello",
      "start_offset": 2,
      "end_offset": 7,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "llo",
      "start_offset": 2,
      "end_offset": 7,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "wor",
      "start_offset": 8,
      "end_offset": 13,
      "type": "<ALPHANUM>",
      "position": 2
    },
    {
      "token": "worl",
      "start_offset": 8,
      "end_offset": 13,
      "type": "<ALPHANUM>",
      "position": 2
    },
    {
      "token": "orl",
      "start_offset": 8,
      "end_offset": 13,
```

```
            "type": "<ALPHANUM>",
            "position": 2
        },
        {
            "token": "orld",
            "start_offset": 8,
            "end_offset": 13,
            "type": "<ALPHANUM>",
            "position": 2
        },
        {
            "token": "rld",
            "start_offset": 8,
            "end_offset": 13,
            "type": "<ALPHANUM>",
            "position": 2
        }
    ]
}
```

- 自定义分词 api

Request:

```
PUT my_analyzer
{
   "settings": {
      "analysis": {
         "analyzer": {
            "my":{
               "tokenizer":"punctuation",
               "type":"custom",
               "char_filter":["emoticons"],
               "filter":["lowercase","english_stop"]
            }
         },
         "tokenizer": {
            "punctuation":{
               "type":"pattern",
               "pattern":"[.,!?]"
            }
         },
         "char_filter": {
            "emoticons":{
               "type":"mapping",
```

```
                "mappings":[
                    ":)=>_happy_",
                    ":(=>_sad_"
                ]
            }
        },
        "filter": {
            "english_stop":{
                "type":"stop",
                "stopwords":"_english_"
            }
        }
    }
}
```

测试：

```
POST my_analyzer/_analyze
{
  "analyzer": "my",
  "text":"I'm a :) person,and you?"
}
```

```
{
  "tokens": [
    {
      "token": "I'm a _happy_ person",
      "start_offset": 0,
      "end_offset": 15,
      "type": "word",
      "position": 0
    },
    {
      "token": "and you",
      "start_offset": 16,
      "end_offset": 23,
      "type": "word",
      "position": 1
    }
  ]
}
```

- 分词使用场景

  1、索引时分词：创建或更新文档时，会对相应得文档进行分词(指定字段分词)

```
PUT my_test
{
“mappings”:{
“doc”:{
“properties”:{
“title”:{
“type”:”text”,
“analyzer”:”whitespace”
}
}
}
}
}
```

2、查询时分词：查询时会对查询语句进行分词

```
POST my_test/_search
{
“query”:{
“match”:{
“message”:{
“query”:”hello”,
“analyzer”:”standard”
}
}
}
}
```

```
PUT my_test
{
“mappings”:{
“doc”:{
“properties”:{
“title”:{
“type”:”text”,
“analyzer”:”whitespace”,
“search_analyzer”:”standard”                    #查询指定分词器
}
}
}
}
}
```

一般不需要特别指定查询时分词器，直接使用索引时分词器即可，否则会出现无法匹配得情况，如果不需要分词将字段 type 设置成 keyword，可以节省空间

1.1.5、Mapping

- 作用：
  定义数据库中的表的结构的定义，通过 mapping 来控制索引存储数据的设置
  a. 定义 Index 下的字段名（Field Name）
  b. 定义字段的类型，比如数值型、字符串型、布尔型等
  c. 定义倒排索引相关的配置，比如 documentId、记录 position、打分等
- 获取索引 mapping
  不进行配置时，自动创建的 mapping
  请求：
  GET /atguigu/_mapping
  响应：

```
{
  "atguigu": {                                    #索引名称
    "mappings": {                                 #mapping 设置
      "student": {                                #type 名称
        "properties": {                           #字段属性
          "clazz": {
            "type": "text",                       #字段类型，字符串默认类型
            "fields": {                           #子字段属性设置
              "keyword": {                        #分词类型（不分词）
                "type": "keyword",
                "ignore_above": 256
              }
            }
          },
          "description": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          },
          "name": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
```

```
                }
            }
        }
    }
  }
}
```

- 自定义 mapping

  请求：

```
PUT my_index                                #索引名称
{
  "mappings":{
    "doc":{                                 #类型名称
      "dynamic":false,
      "properties":{
        "title":{
          "type":"text"                     #字段类型
        },
        "name":{
          "type":"keyword"
        },
        "age":{
          "type":"integer"
        }
      }
    }
  }
}
```

  响应：

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

- Dynamic Mapping

  es 依靠 json 文档字段类型来实现自动识别字段类型，支持的类型

JSON 类型	es 类型
null	忽略
boolean	boolean
浮点类型	float
整数	long

object	object
array	由第一个非 null 值的类型决定
string	匹配为日期则设为 data 类型（默认开启） 匹配为数字的话设为 float 或 long 类型（默认关闭） 设为 text 类型，并附带 keyword 的子字段

- 注意：
  mapping 中的字段类型一旦设定后，禁止修改
  原因：Lucene 实现的倒排索引生成后不允许修改(提高效率)
  如果要修改字段的类型，需要从新建立索引，然后做 reindex 操作
- dynamic 设置
  a. true：允许自动新增字段（默认的配置）
  b. False：不允许自动新增字段，但是文档可以正常写入，无法对字段进行查询操作
  c. strict：文档不能写入（如果写入会报错）

- cope_to
  作用:将字段的值赋值到目标字段，实现类似_all 的作用

  例如：
  1、创建 mapping，包含 copy_to 字段

```
PUT my_index
{
    "mappings":{
        "doc":{
            "properties":{
                "frist_name":{
                    "type":"text",
                    "copy_to":"full_name"
                },
                "last_name":{
                    "type":"text",
                    "copy_to":"full_name"
                },
                "full_name":{
                    "type":"text"
                }
            }
        }
    }
}
```

  2、创建文档

```
PUT my_index/doc/1
{
  "frist_name":"John",
  "last_name":"Smith"
}
```

3、查询文档

```
GET my_index/_search
{
  "query":{
    "match": {
      "full_name":{
        "query":"John Smith",
        "operator":"and"
      }
    }
  }
}
```

- Index 属性

Index 属性，控制当前字段是否索引，默认为 true，即记录索引，false 不记录，即不可以搜索，比如：手机号、身份证号等敏感信息，不希望被检索

例如：

1、创建 mapping

```
PUT my_index
{
  "mappings": {
    "doc":{
      "properties": {
        "cookie":{
          "type":"text",
          "index": false
        }
      }
    }
  }
}
```

2、创建文档

```
PUT my_index/doc/1
{
  "cookie":"123",
```

```
    "name":"home"
  }
```

3、查询

```
GET my_index/_search
{
  "query": {
    "match": {
      "cookie":"123"
    }
  }
}
#报错
GET my_index/_search
{
  "query": {
    "match": {
      "name":"home"
    }
  }
}
#有结果
```

- index_option 用于记录倒排索引记录的内容，有一下 4 种配置
  1、docs 只记录 doc id
  2、freqs 记录 docid 和 term frequencies
  3、positions 记录 docid、term frequencies 和 term position
  4、offsets 记录 docid、term frequencies、term position 和 character offsets
  Text 类型默认配置为 positions，其他默认为 docs
  记录的内容越多暂用的空间越大

```
PUT my_index
{
“doc”:{
“properties”:{
“cookie”:{
“type”:”text”,
“index_option”:”offsets”
}
}
}
}
```

- null_value

  当字段遇到 null 值时，默认为 null，即空值，此时而社会忽略该值。可以通过设定该值设定字段的默认值

1.1.6、数据类型

- 核心数据类型

  字符串型：text、keyword

  数值型：long、integer、short、byte、double、float、half_float、scaled_float

  日期类型：date

  布尔类型：boolean

  二进制类型：binary

  范围类型：integer_range、float_range、long_range、double_range、date_range
- 复杂数据类型

  数组类型：array

  对象类型：object

  嵌套类型：nested object
- 地理位置数据类型

  geo_point(点)、geo_shape(形状)
- 专用类型

  记录 IP 地址 ip

  实现自动补全 completion

  记录分词数：token_count

  记录字符串 hash 值母乳 murmur3
- 多字段特性 multi-fields

  允许对同一个字段采用不同的配置，比如分词，例如对人名实现拼音搜索，只需要在人名中新增一个子字段为 pinyin 即可

1、创建 mapping

```
PUT my_index1
{
  "mappings": {
    "doc":{
      "properties":{
        "username":{
          "type": "text",
          "fields": {
            "pinyin":{
              "type": "text"
            }
          }
        }
      }
    }
  }
```

```
}
```

2、创建文档

```
PUT my_index1/doc/1
{
  "username":"haha heihei"
}
```

3、查询

```
GET my_index1/_search
{
  "query": {
    "match": {
      "username.pinyin": "haha"
    }
  }
}
```

- Dynamic Mapping

  es 可以自动识别文档字段类型，从而降低用户使用成本

```
PUT /test_index/doc/1
{
  "username":"alfred",
  "age":1
}
```

```
{
  "test_index": {
    "mappings": {
      "doc": {
        "properties": {
          "age": {
            "type": "long"
          },
          "username": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          }
        }
```

```
        }
    }
  }
}
```

age 自动识别为 long 类型，username 识别为 text 类型

```
PUT test_index/doc/1
{
  "username":"samualz",
  "age":14,
  "birth":"1991-12-15",
  "year":18,
  "tags":["boy","fashion"],
  "money":"100.1"
}
```

```
{
  "test_index": {
    "mappings": {
      "doc": {
        "properties": {
          "age": {
            "type": "long"
          },
          "birth": {
            "type": "date"
          },
          "money": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          },
          "tags": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
```

```
            },
            "username": {
              "type": "text",
              "fields": {
                "keyword": {
                  "type": "keyword",
                  "ignore_above": 256
                }
              }
            },
            "year": {
              "type": "long"
            }
          }
        }
      }
    }
```

日期的自动识别可以自行配置日期格式，以满足各种需求

1、自定义日期识别格式

```
PUT my_index
{
  "mappings":{
    "doc":{
      "dynamic_date_formats": ["yyyy-MM-dd","yyyy/MM/dd"]
    }
  }
}
```

2、关闭日期自动识别

```
PUT my_index
{
  "mappings": {
    "doc": {
      "date_detection": false
    }
  }
}
```

字符串是数字时，默认不会自动识别为整形，因为字符串中出现数字时完全合理的

Numeric_datection 可以开启字符串中数字的自动识别

```
PUT my_index
{
```

```
    "mappings":{
      "doc":{
        "numeric_datection": true
      }
    }
  }
```

- Dynamic Templates

  允许根据 es 自动识别的数据类型、字段名等来自动设定字段类型

  -所有字符串类型都设定为 keyword 类型，即默认不分词

  -所有以 message 开头的字段都设定为 text 类型，即分词

  -所有以 long_开头的字段都设定为 long 类型

  -所有自动匹配为 double 类型的都设定为 float 类型，以节省空间

  1、匹配规则

  match_mapping_type：匹配 es 自动识别的字段类型，如 boolean，long，string 等

  match,unmatch：匹配字段名

  path_match,path_unmatch：匹配对象内部字段

  2、例子

  把所有字符串类型的匹配成 keyword 类型

```
PUT test_index
{
  "mappings": {
    "doc": {
      "dynamic_templates":[                    #数组，可指定多个模板
        {
          "strings":{                          #模板名称
            "match_mapping_type":"string",     #匹配规则
            "mapping":{                        #设置 mapping 信息
              "type":"keyword"
            }
          }
        }
      ]
    }
  }
}
```

  以 message 开头的字段都设置成 text 类型

```
PUT test_index
{
  "mappings": {
    "doc":{
      "dynamic_templates":[
```

```
            {
              "message_as_text":{
                "match_mapping_type":"string",
                "match":"message*",
                "mapping":{
                  "type":"text"
                }
              }
            }
          ]
        }
      }
    }
```

double 类型设定为 float，节省空间

```
PUT test_index
{
  "mappings": {
    "doc":{
      "dynamic_templates":[
        {
          "message_as_text":{
            "match_mapping_type":"double",
            "mapping":{
              "type":"float"
            }
          }
        }
      ]
    }
  }
}
```

- 索引模板
  用于在新建索引时自动应用预先设定的配置，简化索引创建的操作步骤
  1、可以设定索引的配置和 mapping
  2、可以有多个模板，根据 order 设置，order 大的覆盖小的配置
  3、

- 自定义 mapping 步骤
  1、写入一条文档到 es 的临时索引中，获取 es 自动生成的 mapping
  2、修改步骤 1 得到的 mapping，自定义相关配置
  3、使用步骤 2 中的 mapping 创建实际所需的索引

```
PUT _template/test_template
{
  "index_patterns":["te*","bar*"],
  "order":0,                          #order 越大，优先级越高，覆盖 order 小的模板
  "settings": {
    "number_of_shards": 1
  },
  "mappings": {
    "doc":{
      "_source":{
        "enabled":false
      },
      "properties":{
        "name":{
          "type":"keyword"
        }
      }
    }
  }
}
```

有时索引创建出问题，首先查看模板

1.1.7、Search API(URI)

```
GET /_search                        #查询所有索引文档
GET /my_index/_search               #查询指定索引文档
GET /my_index1,my_index2/_search    #多索引查询
GET /my_*/_search
```

- URI 查询方式（查询有限制，很多配置不能实现）

```
GET /my_index/_search?q=user:alfred     #指定字段查询
GET /my_index/_search?q=alfred&df=user&sort=age:asc&from=4&size=10&timeout=1s
```

q：指定查询的语句，例如 q=aa 或 q=user:aa
df:q 中不指定字段默认查询的字段，如果不指定，es 会查询所有字段
Sort：排序，asc 升序，desc 降序
timeout：指定超时时间，默认不超时
from，size：用于分页

- term 与 phrase
  term 相当于单词查询，phrase 相当于词语查询
  term：Alfred way 等效于 alfred or way
  phrase：”Alfred way” 词语查询，要求先后顺序
- 泛查询

Alfred 等效于在所有字段去匹配该 term(不指定字段查询)

- 指定字段

  name:alfred

- Group 分组设定（），使用括号指定匹配的规则

  （quick OR brown）AND fox：通过括号指定匹配的优先级

  status:(active OR pending) title:(full text search)：把关键词当成一个整体

- 查询案例及详解

1、批量创建文档

```
POST test_search_index/doc/_bulk
{
  "index":{
    "_id":1
  }
}
{
  "username":"alfred way",
  "job":"java engineer",
  "age":18,
  "birth":"1991-12-15",
  "isMarried":false
}
{
  "index":{
    "_id":2
  }
}
{
  "username":"alfred",
  "job":"java senior engineer and java specialist",
  "age":28,
  "birth":"1980-05-07",
  "isMarried":true
}
{
  "index":{
    "_id":3
  }
}
{
  "username":"lee",
  "job":"java and ruby engineer",
  "age":22,
  "birth":"1985-08-07",
  "isMarried":false
```

```
}
{
  "index":{
    "_id":4
  }
}
{
  "username":"lee junior way",
  "job":"ruby engineer",
  "age":23,
  "birth":"1986-08-07",
  "isMarried":false
}
```

2、泛查询

```
GET test_search_index/_search?q=alfred
```

3、查询语句执行计划查看

```
GET test_search_index/_search?q=alfred
{
  "profile":true
}
```

4、term 查询

```
GET test_search_index/_search?q=username:alfred way          #alfred OR way
```

5、phrase 查询

```
GET test_search_index/_search?q=username:"alfred way"
```

6、group 查询

```
GET test_search_index/_search?q=username:(alfred way)
```

7、布尔操作符

（1）AND(&&),OR(||),NOT(!)

例如：name:(tom NOT lee)

#表示 name 字段中可以包含 tom 但一定不包含 lee

（2）+、-分别对应 must 和 must_not

例如：name:(tom +lee -alfred)

#表示 name 字段中，一定包含 lee，一定不包含 alfred，可以包含 tom

注意：+在 url 中会被解析成空格，要使用 encode 后的结果才可以，为%2B

```
GET test_search_index/_search?q=username:(alfred %2Bway)
```

- 范围查询，支持数值和日期

1、区间：闭区间：[]，开区间:{}

age:[1 TO 10] #1<=age<=10

age:[1 TO 10} #1<=age<10

age:[1 TO] #1<=age

age:[* TO 10] #age<=10

2、算术符号写法

age:>=1

age:(>=1&&<=10)或者 age:(+>=1 +<=10)

- 通配符查询

?:1 个字符

*:0 或多个字符

例如：name:t?m

name:tom*

name:t*m

注意：通配符匹配执行效率低，且占用较多内存，不建议使用，如无特殊要求，不要讲?/*放在最前面

- 正则表达式

name:/[mb]oat/

- 模糊匹配 fuzzy query

name:roam~1

匹配与 roam 差 1 个 character 的词，比如 foam、roams 等

- 近似度查询 proximity search

“fox quick”~5

以 term 为单位进行差异比较，比如”quick fox” “quick brown fox”

1.1.8、Search API(Request Body Search)

- Match Query

对字段作全文检索，最基本和常用的查询类型

```
GET test_search_index/_search
{
  "profile":true,                # 显示执行计划
  "query":{
    "match": {
      "username": "alfred way"
    }
  }
}
```

通过 operator 参数可以控制单词间的匹配关系，可选项为 or 和 and

```
GET test_search_index/_search
{
```

```
    "query":{
      "match": {
        "username": {
          "query":"alfred way",
          "operator":"and"
        }
      }
    }
}
```

通过 minimum_should_match 参数可以控制需要匹配的单词数

如下例子，匹配 alfred way 分词后，其中的一个词即可

```
GET test_search_index/_search
{
  "query": {
    "match": {
      "username": {
        "query": "alfred way",
        "minimum_should_match":1
      }
    }
  }
}
```

1.1.9、相关性算分

相关性算分是指文档与查询语句间的相关度，英文为 relevance

本质就是搜索结果返回文档的排序问题

Term Frequency(TF)	词频	单词在该文档中出现的次数。词频越高，相关度越高
Document Frequency(DF)	文档频率	单词出现的文档数
Inverse Document Frequency(IDF)	逆向文档频率	
Field-length Norm		