ES 教程

Copyrights：zlJin ,SoochowUniversity. NLPlab

官网链接：https://www.elastic.co/cn/products

目录：

1.简介：

ElasticSearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。Elasticsearch是用Java开发的，并作为Apache许可条款下的开放源码发布，是当前流行的企业级搜索引擎。

2.安装方法

1. Python环境中，第一步需要安装相对应的elasticsearch模块

```
pip install elasticsearch
```

2. 然后在文件中引用

```
from elasticsearch import Elasticsearch
```

3.使用方法

3-1 建立连接

```
  obj = ElasticSearchClass("localhost", "9200", "", "")
```

3-2 基本操作 (增删改查)

- 添加数据

```
  body = {"question":question, "answer":answer, "qtype":qtype, "timestamp":
str(now_timestamp), "origin": "baike"}
  es.create(index=index, body=body, id='77777777', doc_type='qapair')
    # 其中index是固定传入
    # id可以自己传入也可以系统生成
    # 其中body数据为自己组合的数据
    # doc_type相当于表名，index相当于数据库名
```

- 删除数据

```
  es.delete(index=index, id='77777777', doc_type='qapair')
  #delete命令只能根据id来删除
  #delete(self, indexname, doc_type, id)
```

- 更改数据

暂时没有看到更改数据的api，可以删除后再增加

- 查找数据

```
  body = {
      "query" : {
          "match" : {
              "_id" : "77777777"  #这里可以换成其他属性进行粗匹
          }
      }
  }
  res = es.search(index=index, body=body)['hits']['hits']
  #数据的返回格式
  {
     ...
     "hits": {
        "total":      1,
        "max_score":  0.23013961,
        "hits": [
           {
              ...
              "_score":         0.23013961,
              "_source": {
                 "字段名1":  "XXX",
                 "字段名2":   "XXX",
                 "字段名3":   "XXX",
              }
           }
        ]
     }
  }
```

4.进阶查询

4.1 布尔查询----多字段（或）匹配

```
  body = {
    "query": {
      "bool": {    #表示布尔查询，需要以下内容返回值为True
        "should": [   #should表示或，以下查询条件满足其一即可
          { "match": { "query":  "武大靖" }},
          { "match": { "qtype": "Which"  }},
          { "match": { "origin": 'baike' }}
        ],
      }
    }
  }

  #eg.满足了条件qtype和origin，没有满足query
  {'_index': 'oly', '_type': 'qapair', '_id':
'efd0728e660339b6ab062dcd642c7559', '_score': 1.7835083, '_source': {'question':
'在罗马第几届奥运会结束两周后，举行了第1届“残疾人奥林匹克运动会”？', 'answer': '15名',
'qtype': 'Which', 'timestamp': '1604031429.630478', 'origin': 'baike'}}
```

4.2 布尔查询----多字段（且）匹配

```
  body = {
    "query": {
      "bool": {    #表示布尔查询，需要以下内容返回值为True
        "must": [   #must表示且，以下查询条件需要全部满足 must_not表示都不满足
          { "match": { "query":  "武大靖" }},
          { "match": { "qtype": "What"  }},
          { "match": { "origin": 'baike' }}
        ],
      }
    }
  }

  #eg.
  {'_index': 'oly', '_type': 'qapair', '_id':
'a745533d8d3418805762edef5ae1667b', '_score': 18.494246, '_source': {'question':
'什么原因不会导致武大靖能频破世界纪录？', 'answer': '想赢怕输的心理', 'qtype': 'What',
'timestamp': '1604031429.630478', 'origin': 'baike'}}
```

4.3 短语匹配查询

4.3.1 全文搜索

```
    body = {
        "query" : {
            "match" : {
                "question" : "武大靖哒哒哒哒"  #进行全文搜索，即每一条数据都会作为返回结果，尽管该字符串没有完整的出现，会有一个相关性得分，
                                                #这也是区别于其他数据库检索，其他数据库要么匹配，要么不匹配，而全文搜索给出了相关性得分
            }
        }
    }
```

4.3.2 短语精确匹配

```
  精确匹配一系列单词或者短语
  body = {
      "query" : {
          "match_phrase" : {            # match字段不要求字符串出现，只会根据字符串去匹配分数最高的
              "question" : "武大靖大大大大"      # match_phrase字段要求字符串出现，若不出现则不会匹配结果
          }
      }
  }
  #res = [] 为空，没有出现过“武大靖大大大”
```

4.4 高亮搜索

许多应用都倾向于在每个搜索结果中 高亮 部分文本片段，以便让用户知道为何该文档符合查询条件。在 Elasticsearch 中检索出高亮片段也很容易。

再次执行前面的查询，并增加一个新的 highlight 参数

```
  目前2.x和5.x的语法好像不同，在5.x中调用没有成功，暂时没啥用，后续再研究
```

4.5 filter过滤器

filter仅仅判断一个对象是否符合条件，并不计算分数

query是计算相关性得分

```
  body = {
      "query" : {
          "bool": {
              "must": {
                "match" : {
                      "last_name" : "smith"
                  }
              },
              "filter": {   #过滤器 和must并齐
                  "range" : {   #range过滤器
                      "age" : { "gt" : 30 }    #age属性 gt代表greater than,大于
                  }
              }
          }
```

```
        }
    }
    #感觉就对数字型的属性有点用，对字符型没啥用
```

4.6 同时使用

```
        "bool": {
            "must":     { "match": { "tweet": "elasticsearch" }},
            "must_not": { "match": { "name":  "mary" }},
            "should":   { "match": { "tweet": "full text" }},
            "filter":   { "range": { "age" : { "gt" : 30 }} }
        }
```

4.7 聚合

```
body = {
    'aggs':{
        'all_which':{
            'terms':{
                'field':'qtype'
            }
        }
    }
}

#报错！！！！！！！  'Fielddata is disabled on text fields by default.'

#官网解释为ES5.0后Fielddata is disabled on text fields by default意思是默认禁用Fielddata，原因是加载fielddata是一个昂贵的过程，可能会导致用户遇到延迟命中。

#但是我这个qtype是'keyword'的属性，不是text

#官方给出的结果类似于数据统计吧
"aggregations": {
        "all_interests": {
           "buckets": [
              {
                 "key":       "music",
                 "doc_count": 2
              },
              {
                 "key":       "forestry",
                 "doc_count": 1
              },
              {
                 "key":       "sports",
                 "doc_count": 1
              }
           ]
        }
}
#可以统计不同的字段属性出现的次数（这个倒无所谓，写个程序查就完事了）

#可以与query进行组合查询统计：
{
  "query": {
```

```
    "match": {
      "last_name": "smith"
    }
  },
  "aggs": {
    "all_interests": {
      "terms": {
        "field": "interests"
      }
    }
  }
}

#先通过query查询，再通过aggs进行统计
#结果：
  "all_interests": {
     "buckets": [
        {
           "key": "music",
           "doc_count": 2
        },
        {
           "key": "sports",
           "doc_count": 1
        }
     ]
  }
#只对query检索出来的数据条目进行aggregation统计
```

Mapping

在导入数据之间，先需要确定各个属性，即建立mapping:

```
mapping = {
    "mappings": {
        "data": {
            "properties": { #属性
                "question": {
                    "type": "text",
                    "analyzer": "ik_max_word",
                    "search_analyzer": "ik_max_word"
                },
                "qtype": {
                    "type": "keyword"
                },
                "answer": {
                    "type": "text"
                },
                "timestamp": {
                    "type": "text"
                },
                "origin": {
                    "type": "text"
                },
```

```
            }
        }
    }
}
```

5.1 Type: keyword & text

在建立一个mapping时，需要对一个属性的类型type进行设置，有两种类型 keyword和text：

- keyword:

  不可分词，直接索引,支持模糊、支持精确匹配，支持聚合、排序操作。

  - 适用于：邮箱号码、url、name、title，手机号码、主机名、状态码、邮政编码、标签、年龄、性别等数据。
  - 用于筛选数据（可以进行聚合操作（统计））
- text:

  支持分词，全文检索,支持模糊、精确查询、不支持聚合、排序操作

  - 适用于：内容、地址、代码块、博客文章内容等
  - 默认结合标准分析器进行词命中、词频相关度打分。

5.2 分析器analyzer

```
"question": {
    "type": "text",
    "analyzer": "ik_max_word",
    "search_analyzer": "ik_max_word"
},
```

对于一个属性，可以对其添加分析器，默认为stardard analyzer(标准解析器)：用于对文本进行分词、倒排索引等。

- **ik_max_word:**

  会将文本做最细粒度的拆分，比如会将“中华人民共和国人民大会堂”拆分为“中华人民共和国、中华人民、中华、华人、人民共和国、人民、共和国、大会堂、大会、会堂等词语。
- **ik_smart_word:**

  会做最粗粒度的拆分，比如会将“中华人民共和国人民大会堂”拆分为中华人民共和国、人民大会堂。