1.elasticsearch的索引原理

2020年11月20日 15:02

es作为一个全文检索服务器，那么它在搜索方面肯定很在行啦！那它是怎么做到的呢？

Es官方有这么一句话：一切设计都是为了提高搜索的性能！

Es能够快速的搜索出我们需要的内容，靠的就是倒排索引的思想，或者说是一种设计！

在没有使用倒排索引的情况下，正常思路是根据搜索关键字去查找相应的内容，但是使用了倒排索引之后，ES会先将文档的所有内容拆分成多个词条，创建一个包含所有不重复词条的排序列表，然后列出每个词条出现在哪个文档。

例如，假设我们有两个文档，每个文档的 `content` 域包含如下内容：

Doc_1:**The quick brown fox jumped over the lazy dog**

Doc_2:**Quick brown foxes leap over lazy dogs in summer**

ES首先会将这两个文档拆分成多个单独的词，或者叫做词条，然后为所有的词条创建一个排序列表，并记录每个词条出现的文档的信息。就像下面这样：

```
Term      Doc_1  Doc_2
-------------------------
Quick   |       |  X                          /*
The     |   X   |                                                    Term就是词条，比如第一个Term就是Quick关键字，在Doc_1中不存
brown   |   X   |  X                                                 在，在Doc_2中存在，其他的以此类推。
dog     |   X   |                                                    */
dogs    |       |  X
fox     |   X   |
foxes   |       |  X
in      |       |  X
jumped  |   X   |
lazy    |   X   |  X
leap    |       |  X
over    |   X   |  X
quick   |   X   |
summer  |       |  X
the     |   X   |
-------------------------
```

现在，如果我们想搜索 **quick**和**brown**这两个关键字，我们只需要查找包含每个词条的文档，就相当于我们查询的时候，是通过这个索引表找到文档，在通过文档去找文档内容中的搜索关键字，与传统的通过关键字去找内容是不同的。

倒排索引到底是个怎么实现的，怎么个思想，我在这里就不一一说明了，大家可以看下官方的详细介绍：倒排索引的原理

2.ElasticSearch和MYSQL对比

2021年1月18日 10:22

关系型数据库	ElasticSearch
数据库	索引
表	类型
行	文档
列	字段
表结构	映射（Mapping）
SQL	DSL(Domain Specific Language)
Select * from xxx	GET http://
update xxx set xx=xxx	PUT http://
Delete xxx	DELETE http://
索引	全文索引

3.ES倒排索引

2021年1月19日 9:44

倒排索引就是以内容的关键字建立索引，通过索引找到文档 id，再进而找到整个文档。

分为两部分
1.单词字典（记录所有文档的词项，以及词项到倒排列表的关联关系）
2.倒排列表（记录单词和对印关系）

4. ES常用语句

2021年1月20日 14:27

curl -XPOST "http://localhost:9200/geo/_bulk?pretty" -H "content-type:application/json" --data-binary @geo.json

- 调整字段的映射，给字段指定分词器

```
PUT blog
{
  "mappings": {
    "properties": {
      "title":{
        "type":"text",
        "analyzer": "ik_smart"
      }
    }
  }
}
```

```
PUT ik_index/_mapping
{
    "properties": {
      "title":{
        "type":"text",
        "analyzer": "ik_smart"
      }
    }

}
```

```
PUT blog/_doc/1
{
  "title":"定义文本字段的分词器。默认对索引和查询都是有效的。"
}
```

```
GET blog/_termvectors/1
{
  "fields": ["title"]
}
```

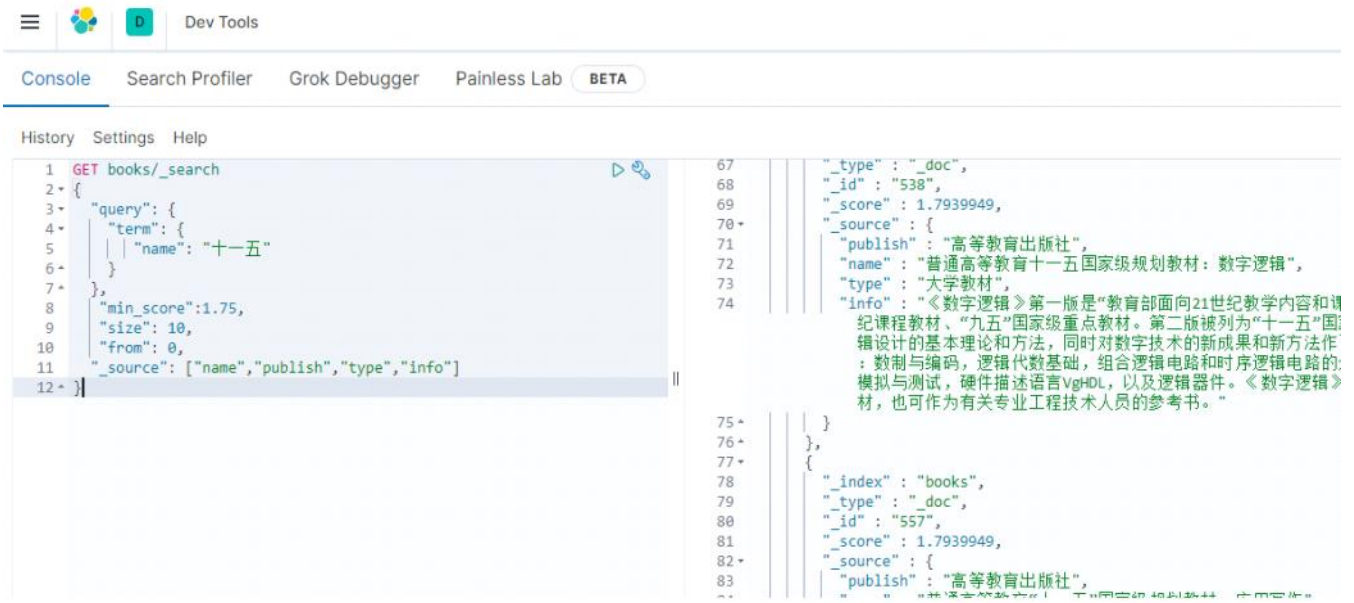
- match query 和 term query的区别
  Match query会对查询内容进行分词后，再去匹配查询；term query是直接用查询内容去匹配，如果查询字段未分词，会出现查询不到数据的结果。

```
GET books/_search
{
  "query": {
    "match": {
      "name": {
        "query": "美术计算机",
        "operator": "and"
      }
    }
  }
}
```

```
GET ik_index/_search
{
  "query": {
    "term": {
      "address": "上海市"
    }
  }
}
```

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 0,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  }
}
```

- 有的文档得分特别低，说明这个文档和我们查询的关键字相关度很低。我们可以设置一个最低分，只有得分超过最低分的文档才会被返回。



- Match_phrase query
  分词后的词项顺序必须和文档中词项的顺序一致
  所有的词都必须出现在文档中

```
GET books/_search
{
  "query": {
    "match_phrase": {
        "name": {
          "query": "十一五计算机",
          "slop": 7
        }
    }
  }
}
```

- match_phrase_prefix query
  这里多了一个通配符，match_phrase_prefix 支持最后一个词项的前缀匹配

```
GET books/_search
{
  "query": {
    "match_phrase_prefix": {
      "name": "计"
    }
  }
}
```

- multi_match query
  可以指定多个查询域，指定字段的权重

```
GET books/_search
{
  "query": {
    "multi_match": {
      "query": "阳光",
      "fields": ["name^4","info"]
    }
  }
}
```

- query_string query
  紧密结合Lucene的查询方式，在一个查询语句中用到Lucene的一些查询语句

```
GET books/_search
{
  "query": {
    "query_string": {
      "default_field": "name",
      "query": "(十一五) AND (计算机)"
    }
  }
}
```

- simple_query_string query
  使用 +、|、- 代替 AND、OR、NOT

```
GET books/_search
{
  "query": {
    "simple_query_string": {
      "fields": ["name"],
      "query": "(十一五) + (计算机)"
    }
  }
}
```

- range query
  范围查询，可以按照日期范围，数字范围查询

```
GET books/_search
{
  "query": {
    "range": {
      "price": {
        "gte": 10,
        "lt": 20
      }
    }
  },
  "sort": [
    {
      "price": {
        "order": "desc"
      }
    }
  ]
}
```

- exists query
  返回指定字段，在当前索引中至少一个非空的文档

```
GET books/_search
{
  "query": {
    "exists": {
      "field": "javaboy"
    }
  }
}
```

- wildcard query
  通配符查询，支持单字符或者多字符查询
  ？表示一个任意字符。
  * 表示零个或者多个字符。

```
GET books/_search
{
  "query": {
    "wildcard": {
      "author": {
        "value": "张?"
      }
    }
  }
}
```

- Fuzzy query
  打错别字也能搜索，返回与搜索关键字相似的文档

```
GET books/_search
{
  "query": {
    "fuzzy": {
      "name": "javba"
    }
  }
}
```

- 复合查询 constant_score query
  使用 constant_score 将查询语句或者过滤语句包裹起来。

```
GET books/_search
{
  "query": {
    "constant_score": {
      "filter": {
        "term": {
          "name": "java"
        }
      },
      "boost": 1.5
    }
  }
}
```

- 复合查询 bool query

  bool query 可以将任意多个简单查询组装在一起，有四个关键字可供选择，四个关键字所描述的条件可以有一个或者多个。

  must：文档必须匹配 must 选项下的查询条件。

  should：文档可以匹配 should 下的查询条件，也可以不匹配。

  must_not：文档必须不满足 must_not 选项下的查询条件。

  filter：类似于 must，但是 filter 不评分，只是过滤数据。

```
GET books/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "term": {
            "name": {
              "value": "java"
            }
          }
        }
      ],
      "must_not": [
        {
          "range": {
            "price": {
```

```
            "gte": 0,
            "lte": 35
          }
        }
      }
    ],
    "should": [
      {
        "match": {
          "info": "程序设计"
        }
      }
    ]
  }
 }
}
```

- should　query中的评分策略
  1. 执行should中的两个查询
  2. 对两个查询结果的评分求和
  3.对求和结果乘以匹配语句的总和

- 自定义评分脚本
  默认情况下，id 为 2 的记录得分较高，因为他的 title 中包含两个 java。
  如果我们在查询中，希望能够充分考虑 votes 字段，将 votes 较高的文档优先展示，就可以通过 function_score 来实现。

```
GET blog/_search
{
  "query": {
    "function_score": {
      "query": {
        "match": {
          "title": "java"
        }
      },
      "functions": [
        {
          "script_score": {
            "script": {
              "lang": "painless",
              "source": "_score + doc['votes'].value"
            }
          }
        }
      ]
    }
  }
```

- more_like_this_query
  实现基于内容的推荐，查询出和该文章相似的内容

```
GET books/_search
{
  "query": {
    "more_like_this": {
      "fields": [
        "info"
      ],
      "like": "大学战略",
      "min_term_freq": 1,
      "max_query_terms": 12
    }
  }
}
```

5. ES多表关联查询

2021年1月21日 13:59

一对多的实现方式

- 嵌套文档

  nested 类型，嵌套查询

```
GET movies/_search
{
  "query": {
    "nested": {
      "path": "actors",
      "query": {
        "bool": {
          "must": [
            {
              "match": {
                "actors.name": "张国荣"
              }
            },
            {
              "match": {
                "actors.gender": "男"
              }
            }
          ]
        }
      }
    }
  }
}
```

- 父子文档

  父子文档的优势，父子文档分离，不会相互影响。

  父子文档需要注意的地方：

  1. 每个索引只能定义一个 join filed
  2. 父子文档需要在同一个分片上（查询，修改需要routing）
  3. 可以向一个已经存在的 join filed 上新增关系

  创建父子文档

```
PUT stu_class
{
  "mappings": {
    "properties": {
      "name":{
        "type": "keyword"
      },
      "s_c":{
        "type": "join",
        "relations":{
          "class":"student"
        }
      }
    }
  }
}
```

插入父文档和子文档

```
PUT stu_class/_doc/1
{
  "name":"一班",
  "s_c":{
    "name":"class"
  }
}
PUT stu_class/_doc/2
{
  "name":"二班",
  "s_c":{
    "name":"class"
  }
}
```

```
PUT stu_class/_doc/3?routing=1
{
  "name":"zhangsan",
  "s_c":{
    "name":"student",
    "parent":1
  }
```

```
PUT stu_class/_doc/3?routing=1
{
  "name":"zhangsan",
  "s_c":{
    "name":"student",
    "parent":1
  }
}
```

父子文档之间查询

子查父

```
GET stu_class/_search
{
  "query": {
    "has_child": {
      "type": "student",
      "query": {
        "match": {
          "name": "wangwu"
        }
      }
    }
  }
}
```

父查子

```
GET stu_class/_search
{
  "query": {
    "has_parent": {
      "parent_type": "class",
      "query": {
        "match": {
          "name": "二班"
        }
      }
    }
  }
}
```

1. 普通子对象实现一对多，会损失子文档的边界，子对象之间的属性关系丢失。
2. nested 可以解决第 1 点的问题，但是 nested 有两个缺点：更新主文档的时候要全部更新，不支持子文档属于多个主文档。
3. 父子文档解决 1、2 点的问题，但是它主要适用于写多读少的场景。

6. ES中字段类型的区别

2021年1月21日 14:59

简单的类型有 text、keyword、date、long、double、boolean和ip
复杂类型有：object和nested
较特殊的类型：geo_point,geo_shape,和completion

准确数据类型： keyword，直接被存储为了二进制，检索时我们直接匹配，不匹配就返回false
全文文本类型： text，这个的检索不是直接给出是否匹配，而是检索出相似度，并按照相似度由高到低返回结果

7. 指标聚合

2021年1月21日 16:59

- cardinality
  统计总数

```
GET books/_search
{
  "aggs": {
    "publish_count": {
      "cardinality": {
        "field": "publish"
      }
    }
  }
}
```

- Stats Aggregation
  基本统计，一次性返回 count、max、min、avg、sum

```
GET books/_search
{
  "aggs": {
    "stats_query": {
      "stats": {
        "field": "price"
      }
    }
  }
}
```

- Value Count Aggregation
  查询包含指定字段的文档数量

```
GET books/_search
{
  "aggs": {
    "count": {
      "value_count": {
        "field": "price"
      }
```

```
GET books/_search
{
  "aggs": {
    "count": {
      "value_count": {
        "field": "price"
      }
    }
  }
}
```

8. 桶聚合

2021年1月21日 17:02

- Terms Aggregation
  分组聚合，统计各个出版社出版的图书总数

```
GET books/_search
{
  "aggs": {
    "NAME": {
      "terms": {
        "field": "publish",
        "size": 20
      }
    }
  }
}
```

- 在terms分桶的基础上，对每个桶进行指标聚合
  将图书按照出版社分组，再统计每组图书的价格平均值

```
GET books/_search
{
  "aggs": {
    "NAME": {
      "terms": {
        "field": "publish",
        "size": 20
      },
      "aggs": {
        "avg_price": {
          "avg": {
            "field": "price"
          }
        }
      }
    }
  }
}
```

- Filter Aggregation
  过滤器聚合，将符合条件的文档分到一个桶中，在对桶中的文档进行指标聚合操作（取平均值，求和...）

```
GET books/_search
{
  "aggs": {
    "NAME": {
      "filter": {
        "term": {
          "name": "java"
        }
      },
      "aggs": {
        "avg_price": {
          "avg": {
            "field": "price"
          }
        }
      }
    }
  }
}
```

- Range Aggregation
  按照某一范围，分组进行统计

```
GET books/_search
{
  "aggs": {
    "NAME": {
      "range": {
        "field": "price",
        "ranges": [
          {
            "to": 50
          },{
            "from": 50,
            "to": 100
          },{
            "from": 100,
            "to": 150
          },{
            "from": 150
          }
        ]
      }
    }
  }
}
```

- Children Aggregation
  根据父子关系进行分桶
  查询子类型为student的文档数量

```
GET stu_class/_search
{
  "aggs": {
    "NAME": {
      "children": {
        "type": "student"
      }
    }
  }
}
```

-

9. 管道聚合

2021年1月21日 17:45

管道聚合就是在之前聚合的基础上，再次聚合。

- Avg Bucket Aggregation
  计算聚合平均值。 统计每个出版社出版图书的平均值，再计算一次总的平均值。

```
GET books/_search
{
  "aggs": {
    "book_count": {
      "terms": {
        "field": "publish",
        "size": 3
      },
      "aggs": {
        "book_avg": {
          "avg": {
            "field": "price"
          }
        }
      }
    },
    "avg_book":{
      "avg_bucket": {
        "buckets_path": "book_count>book_avg"
      }
    }
  }
}
```

10. ES常用java客户端

2021年1月22日 9:14

- Spring Data elasticsearch
  目前支持的NoSQL存储

Spring Data 项目支持 NoSQL 存储：

- MongoDB （文档数据库）
- Neo4j（图形数据库）
- Redis（键/值存储）
- Hbase（列族数据库）
- ElasticSearch

- Java Low Level REST Client
  兼容所有的ES版本,API没有封装json操作，所有的JSON操作还是开发者自己完成

- Java High Level REST Client
  （1）将请求参数和响应参数封装成相应的API，只需要拼接参数和解析响应结果。
  （2）每个API都可以异步或者同步调用，异步的方法以Asyc结尾。异步请求需要一个监听器参数。
  （3）兼容性差，需要JDK8以上版本，依赖版本需要和ES版本一致。