

课程介绍

- 了解倒排索引
- 全文搜索API的使用
- Elasticsearch集群搭建
- 集群中分片和副本
- 集群的故障转移
- 分布式文档
- Java客户端的使用
- Spring Data Elasticsearch

1、全文搜索

1.1、倒排索引

倒排索引源于实际应用中需要根据属性的值来查找记录。这种索引表中的每一项都包括一个属性值和具有该属性值的各记录的地址。由于不是由记录来确定属性值，而是由属性值来确定记录的位置，因而称为倒排索引(inverted index)。带有倒排索引的文件我们称为倒排索引文件，简称倒排文件(inverted file)。

正排索引：



文档编号	文档内容
1	谷歌地图之父跳槽Facebook
2	谷歌地图之父加盟Facebook
3	谷歌地图创始人拉斯离开谷歌加盟Facebook
4	谷歌地图之父跳槽Facebook 与Wave项目取消有关
5	谷歌地图之父拉斯加盟社交网站Facebook

转化成倒排索引：

单词ID	单词	倒排列表 (DocID)
1	谷歌	1,2,3,4,5
2	地图	1,2,3,4,5
3	之父	1,2,4,5
4	跳槽	1,4
5	Facebook	1,2,3,4,5
6	加盟	2,3,5
7	创始人	3
8	拉斯	3,5
9	离开	3
10	与	4
11	Wave	4
12	项目	4
13	取消	4
14	有关	4
15	社交	5
16	网站	5

说明：

- “单词ID”一栏记录了每个单词的单词编号；
- 第二栏是对应的单词；
- 第三栏即每个单词对应的倒排列表；
- 比如单词“谷歌”，其单词编号为1，倒排列表为{1,2,3,4,5}，说明文档集合中每个文档都包含了这个单词。

而事实上，索引系统还可以记录除此之外的更多信息，在单词对应的倒排列表中不仅记录了文档编号，还记载了单词频率信息（TF），即这个单词在某个文档中的出现次数，之所以要记录这个信息，是因为词频信息在搜索结果排序时，计算查询和文档相似度是很重要的一个计算因子，所以将其记录在倒排列表中，以方便后续排序时进行分值计算。

单词ID	单词	倒排列表 (DocID;TF)
1	谷歌	(1;1),(2;1),(3;2),(4;1),(5;1)
2	地图	(1;1),(2;1),(3;1),(4;1),(5;1)
3	之父	(1;1),(2;1),(4;1),(5;1)
4	跳槽	(1;1),(4;1)
5	Facebook	(1;1),(2;1),(3;1),(4;1),(5;1)
6	加盟	(2;1),(3;1),(5;1)
7	创始人	(3;1)
8	拉斯	(3;1),(5;1)
9	离开	(3;1)
10	与	(4;1)
11	Wave	(4;1)
12	项目	(4;1)
13	取消	(4;1)
14	有关	(4;1)
15	社交	(5;1)
16	网站	(5;1)

倒排索引还可以记载更多的信息，除了记录文档编号和单词频率信息外，额外记载了两类信息，即每个单词对应的“文档频率信息”，以及在倒排列表中记录单词在某个文档出现的位置信息。



单词ID	单词	文档频率	倒排列表 (DocID;TF;<POS>)
1	谷歌	5	(1;1;<1>),(2;1;<1>),(3;2;<1;6>),(4;1;<1>),(5;1;<1>)
2	地图	5	(1;1;<2>),(2;1;<2>),(3;1;<2>),(4;1;<2>),(5;1;<2>)
3	之父	4	(1;1;<3>),(2;1;<3>),(4;1;<3>),(5;1;<3>)
4	跳楼	2	(1;1;<4>),(4;1;<4>)
5	Facebook	5	(1;1;<5>),(2;1;<5>),(3;1;<8>),(4;1;<5>),(5;1;<8>)
6	加盟	3	(2;1;<4>),(3;1;<7>),(5;1;<5>)
7	创始人	1	(3;1;<3>)
8	拉斯	2	(3;1;<4>),(5;1;<4>)
9	离开	1	(3;1;<5>)
10	与	1	(4;1;<6>)
11	Wave	1	(4;1;<7>)
12	项目	1	(4;1;<8>)
13	取消	1	(4;1;<9>)
14	有关	1	(4;1;<10>)
15	社交	1	(5;1;<6>)
16	网站	1	(5;1;<7>)

1.2、全文搜索

全文搜索两个最重要的方面是：

- 相关性 (Relevance) 它是评价查询与其结果间的相关程度，并根据这种相关程度对结果排名的一种能力，这种计算方式可以是 TF/IDF 方法、地理位置邻近、模糊相似，或其他的某些算法。
- 分析 (Analysis) 它是将文本块转换为有区别的、规范化的 token 的一个过程，目的是为了创建倒排索引以及查询倒排索引。

1.2.1、构造数据

```

1  PUT http://172.16.55.185:9200/itcast
2  {
3      "settings": {
4          "index": {
5              "number_of_shards": "1",
6              "number_of_replicas": "0"
7          }
8      },
9      "mappings": {
10         "person": {
11             "properties": {
12                 "name": {
13                     "type": "text"
14                 },
15                 "age": {
16                     "type": "integer"
17                 },
18                 "mail": {
19                     "type": "keyword"

```



```
20         },
21         "hobby": {
22             "type": "text",
23             "analyzer": "ik_max_word"
24         }
25     }
26 }
27 }
28 }
```

批量插入数据：

```
1 POST http://172.16.55.185:9200/itcast/_bulk
2
3 {"index":{"_index":"itcast","_type":"person"}}
4 {"name":"张三","age": 20,"mail": "111@qq.com","hobby":"羽毛球、乒乓球、足球"}
5 {"index":{"_index":"itcast","_type":"person"}}
6 {"name":"李四","age": 21,"mail": "222@qq.com","hobby":"羽毛球、乒乓球、足球、篮球"}
7 {"index":{"_index":"itcast","_type":"person"}}
8 {"name":"王五","age": 22,"mail": "333@qq.com","hobby":"羽毛球、篮球、游泳、听音乐"}
9 {"index":{"_index":"itcast","_type":"person"}}
10 {"name":"赵六","age": 23,"mail": "444@qq.com","hobby":"跑步、游泳、篮球"}
11 {"index":{"_index":"itcast","_type":"person"}}
12 {"name":"孙七","age": 24,"mail": "555@qq.com","hobby":"听音乐、看电影、羽毛球"}
13
```

结果：

查询 1 个分片中用的 1 个. 4 命中. 耗时 0.001 秒

_index	_type	_id	_score ▲	name	age	mail	hobby
itcast	person	UP0cDWgBR-bSw8-LpdKZ	1	张三	20	111@qq.com	羽毛球、乒乓球、足球
itcast	person	Uf0cDWgBR-bSw8-LpdKZ	1	李四	21	222@qq.com	羽毛球、乒乓球、足球、篮球
itcast	person	Uv0cDWgBR-bSw8-LpdKZ	1	王五	22	333@qq.com	羽毛球、篮球、游泳、听音乐
itcast	person	VP0cDWgBR-bSw8-LpdKZ	1	孙七	24	555@qq.com	听音乐、看电影、羽毛球

1.2.2、单词搜索

```
1 POST http://172.16.55.185:9200/itcast/person/_search
2
3 {
4     "query":{
5         "match":{
6             "hobby":"音乐"
7         }
8     },
9     "highlight": {
10         "fields": {
11             "hobby": {}
12         }
13     }
14 }
```



结果：

```
1  {
2      "took": 9,
3      "timed_out": false,
4      "_shards": {
5          "total": 1,
6          "successful": 1,
7          "skipped": 0,
8          "failed": 0
9      },
10     "hits": {
11         "total": 2,
12         "max_score": 0.6841192,
13         "hits": [
14             {
15                 "_index": "itcast",
16                 "_type": "person",
17                 "_id": "Uv0cDWgBR-bSw8-LpdkZ",
18                 "_score": 0.6841192,
19                 "_source": {
20                     "name": "王五",
21                     "age": 22,
22                     "mail": "333@qq.com",
23                     "hobby": "羽毛球、篮球、游泳、听音乐"
24                 },
25                 "highlight": {
26                     "hobby": [
27                         "羽毛球、篮球、游泳、听<em>音乐</em>"
28                     ]
29                 }
30             },
31             {
32                 "_index": "itcast",
33                 "_type": "person",
34                 "_id": "VP0cDWgBR-bSw8-LpdkZ",
35                 "_score": 0.6841192,
36                 "_source": {
37                     "name": "孙七",
38                     "age": 24,
39                     "mail": "555@qq.com",
40                     "hobby": "听音乐、看电影、羽毛球"
41                 },
42                 "highlight": {
43                     "hobby": [
44                         "听<em>音乐</em>、看电影、羽毛球"
45                     ]
46                 }
47             }
48         ]
49     }
50 }
```

过程说明：

1. 检查字段类型

爱好 hobby 字段是一个 text 类型（指定了IK分词器），这意味着查询字符串本身也应该被分词。

2. 分析查询字符串。

将查询的字符串“音乐”传入IK分词器中，输出的结果是单个项 音乐。因为只有一个单词项，所以 match 查询执行的是单个底层 term 查询。

3. 查找匹配文档。

用 term 查询在倒排索引中查找“音乐”然后获取一组包含该项的文档，本例的结果是文档：3、5。

4. 为每个文档评分。

用 term 查询计算每个文档相关度评分 _score，这是种将 词频（term frequency，即词“音乐”在相关文档的 hobby 字段中出现的频率）和 反向文档频率（inverse document frequency，即词“音乐”在所有文档的 hobby 字段中出现的频率），以及字段的长度（即字段越短相关度越高）相结合的计算方式。

1.2.3、多词搜索

```
1 POST http://172.16.55.185:9200/itcast/person/_search
2 {
3     "query":{
4         "match":{
5             "hobby":"音乐 篮球"
6         }
7     },
8     "highlight": {
9         "fields": {
10             "hobby": {}
11         }
12     }
13 }
```

结果：

```
1 {
2     "took": 3,
3     "timed_out": false,
4     "_shards": {
5         "total": 1,
6         "successful": 1,
7         "skipped": 0,
8         "failed": 0
9     },
10    "hits": {
11        "total": 4,
12        "max_score": 1.3192271,
13        "hits": [
14            {
15                "_index": "itcast",
16                "_type": "person",
17                "_id": "Uv0cDWgBR-bSw8-LpdKz",
```



```
18         "_score": 1.3192271,
19         "_source": {
20             "name": "王五",
21             "age": 22,
22             "mail": "333@qq.com",
23             "hobby": "羽毛球、篮球、游泳、听音乐"
24         },
25         "highlight": {
26             "hobby": [
27                 "羽毛球、<em>篮球</em>、游泳、听<em>音乐</em>"
28             ]
29         }
30     },
31     {
32         "_index": "itcast",
33         "_type": "person",
34         "_id": "VP0cDWgBR-bSw8-LpdkZ",
35         "_score": 0.81652206,
36         "_source": {
37             "name": "孙七",
38             "age": 24,
39             "mail": "555@qq.com",
40             "hobby": "听音乐、看电影、羽毛球"
41         },
42         "highlight": {
43             "hobby": [
44                 "听<em>音乐</em>、看电影、羽毛球"
45             ]
46         }
47     },
48     {
49         "_index": "itcast",
50         "_type": "person",
51         "_id": "Vf0gDWgBR-bSw8-LOdm_",
52         "_score": 0.6987338,
53         "_source": {
54             "name": "赵六",
55             "age": 23,
56             "mail": "444@qq.com",
57             "hobby": "跑步、游泳、篮球"
58         },
59         "highlight": {
60             "hobby": [
61                 "跑步、游泳、<em>篮球</em>"
62             ]
63         }
64     },
65     {
66         "_index": "itcast",
67         "_type": "person",
68         "_id": "Uf0cDWgBR-bSw8-LpdkZ",
69         "_score": 0.50270504,
70         "_source": {
```



```
71         "name": "李四",
72         "age": 21,
73         "mail": "222@qq.com",
74         "hobby": "羽毛球、乒乓球、足球、篮球"
75     },
76     "highlight": {
77         "hobby": [
78             "羽毛球、乒乓球、足球、<em>篮球</em>"
79         ]
80     }
81 }
82 ]
83 }
84 }
```

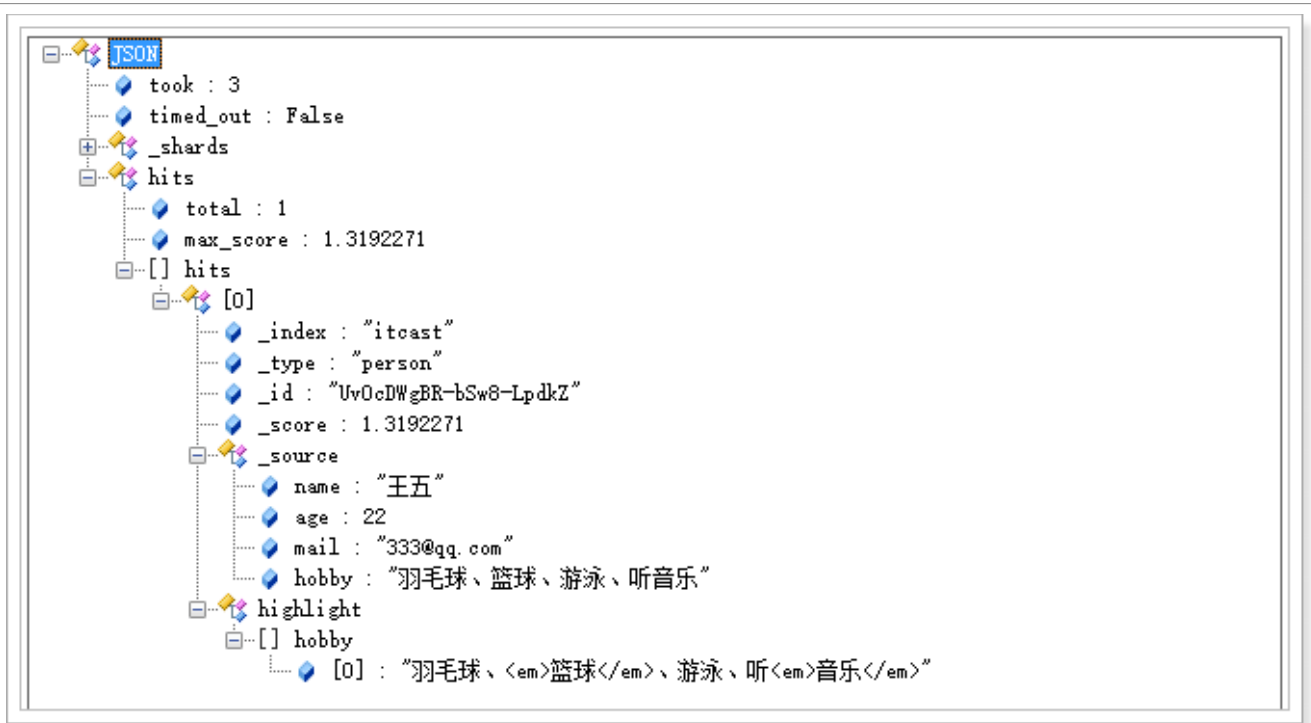
可以看到，包含了“音乐”、“篮球”的数据都已经被搜索到了。

可是，搜索的结果并不符合我们的预期，因为我们想搜索的是既包含“音乐”又包含“篮球”的用户，显然结果返回的“或”的关系。

在Elasticsearch中，可以指定词之间的逻辑关系，如下：

```
1 POST http://172.16.55.185:9200/itcast/person/_search
2 {
3     "query": {
4         "match": {
5             "hobby": {
6                 "query": "音乐 篮球",
7                 "operator": "and"
8             }
9         }
10    },
11    "highlight": {
12        "fields": {
13            "hobby": {}
14        }
15    }
16 }
```

结果：



可以看到结果符合预期。

前面我们测试了“OR”和“AND”搜索，这是两个极端，其实在实际场景中，并不会选取这2个极端，更有可能是选取这种，或者说，只需要符合一定的相似度就可以查询到数据，在Elasticsearch中也支持这样的查询，通过 minimum_should_match 来指定匹配度，如：70%；

示例：

```
1  {
2    "query": {
3      "match": {
4        "hobby": {
5          "query": "游泳 羽毛球",
6          "minimum_should_match": "80%"
7        }
8      }
9    },
10   "highlight": {
11     "fields": {
12       "hobby": {}
13     }
14   }
15 }
16
17 #结果：省略显示
18 "hits": {
19   "total": 4, #相似度为80%的情况下，查询到4条数据
20   "max_score": 1.621458,
21   "hits": [
22     .....
23   ]
24 }
25 #设置40%进行测试：
```



```
26 {
27     "query":{
28         "match":{
29             "hobby":{
30                 "query":"游泳 羽毛球",
31                 "minimum_should_match":"40%"
32             }
33         },
34     },
35     "highlight": {
36         "fields": {
37             "hobby": {}
38         }
39     }
40 }
41 #结果：
42 "hits": {
43     "total": 5, #相似度为40%的情况下，查询到5条数据
44     "max_score": 1.621458,
45     "hits": [
46         .....
47     ]
48 }
```

相似度应该多少合适，需要在实际的需求中进行反复测试，才可得到合理的值。

1.2.4、组合搜索

在搜索时，也可以使用过滤器中讲过的bool组合查询，示例：

```
1 POST http://172.16.55.185:9200/itcast/person/_search
2
3 {
4     "query":{
5         "bool":{
6             "must":{
7                 "match":{
8                     "hobby":"篮球"
9                 }
10            },
11            "must_not":{
12                "match":{
13                    "hobby":"音乐"
14                }
15            },
16            "should":[
17                {
18                    "match": {
19                        "hobby":"游泳"
20                    }
21                }
22            ]
23        }
24    },
25 }
```



```
25     "highlight": {  
26         "fields": {  
27             "hobby": {}  
28         }  
29     }  
30 }
```

上面搜索的意思是：

搜索结果中必须包含篮球，不能包含音乐，如果包含了游泳，那么它的相似度更高。

结果：

```
{  
  "took": 4,  
  "timed_out": false,  
  "_shards": {},  
  "hits": {  
    "total": 2,  
    "max_score": 1.8336569,  
    "hits": [  
      {  
        "_index": "itcast",  
        "_type": "person",  
        "_id": "Vf0gDWgBR-bSw8-L0dm_",  
        "_score": 1.8336569,  
        "_source": {  
          "name": "赵六",  
          "age": 23,  
          "mail": "444@qq.com",  
          "hobby": "跑步、游泳、篮球"  
        },  
        "highlight": {  
          "hobby": [  
            "跑步、<em>游泳</em>、<em>篮球</em>"  
          ]  
        }  
      },  
      {  
        "_index": "itcast",  
        "_type": "person",  
        "_id": "Uf0cDWgBR-bSw8-LpdkZ",  
        "_score": 0.50270504,  
        "_source": {  
          "name": "李四",  
          "age": 21,  
          "mail": "222@qq.com",  
          "hobby": "羽毛球、乒乓球、足球、篮球"  
        },  
        "highlight": {  
          "hobby": [  
            "羽毛球、乒乓球、足球、<em>篮球</em>"  
          ]  
        }  
      }  
    ]  
  }  
}
```

评分的计算规则

bool 查询会为每个文档计算相关度评分 `_score`，再将所有匹配的 `must` 和 `should` 语句的分数 `_score` 求和，最后除以 `must` 和 `should` 语句的总数。

`must_not` 语句不会影响评分；它的作用只是将不相关的文档排除。



默认情况下，should中的内容不是必须匹配的，如果查询语句中没有must，那么就会至少匹配其中一个。当然了，也可以通过minimum_should_match参数进行控制，该值可以是数字也可以的百分比。

示例：

```
1 POST http://172.16.55.185:9200/itcast/person/_search
2
3 {
4   "query":{
5     "bool":{
6       "should":[
7         {
8           "match": {
9             "hobby": "游泳"
10          }
11        },
12        {
13          "match": {
14            "hobby": "篮球"
15          }
16        },
17        {
18          "match": {
19            "hobby": "音乐"
20          }
21        }
22      ],
23      "minimum_should_match": 2
24    },
25  },
26  "highlight": {
27    "fields": {
28      "hobby": {}
29    }
30  }
31 }
```

minimum_should_match为2，意思是should中的三个词，至少要满足2个。

结果：

```
JSON
{
  took : 5
  timed_out : False
  _shards
  hits
    total : 2
    max_score : 2.135749
    hits
      [0]
        _index : "itcast"
        _type : "person"
        _id : "Uv0cDWgBR-bSw8-LpdkZ"
        _score : 2.135749
        _source
          name : "王五"
          age : 22
          mail : "333@qq.com"
          hobby : "羽毛球、篮球、游泳、听音乐"
        highlight
          hobby
            [0] : "羽毛球、<em>篮球</em>、<em>游泳</em>、听<em>音乐</em>"
      [1]
        _index : "itcast"
        _type : "person"
        _id : "Vf0gDWgBR-bSw8-L0dm_"
        _score : 1.8336569
        _source
          name : "赵六"
          age : 23
          mail : "444@qq.com"
          hobby : "跑步、游泳、篮球"
        highlight
          hobby
            [0] : "跑步、<em>游泳</em>、<em>篮球</em>"

```

1.2.5、权重

有些时候，我们可能需要对某些词增加权重来影响该条数据的得分。如下：

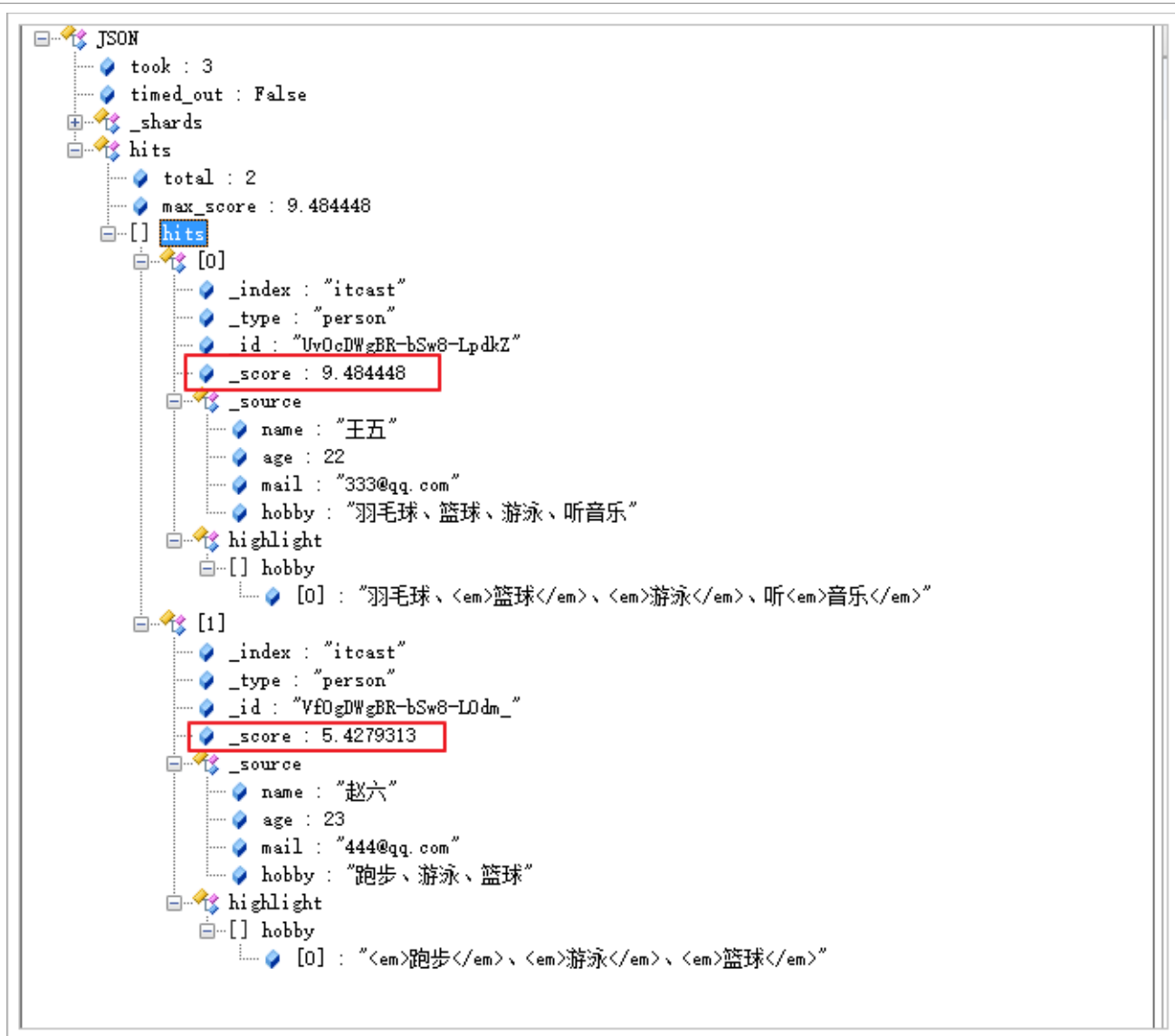
搜索关键字为“游泳篮球”，如果结果中包含了“音乐”权重为10，包含了“跑步”权重为2。

```
1 POST http://172.16.55.185:9200/itcast/person/_search
2 {
3   "query": {
4     "bool": {
5       "must": {
6         "match": {
7           "hobby": {
8             "query": "游泳篮球",
9             "operator": "and"
10          }
11        }
12      },
13      "should": [
```



```
14         {
15             "match": {
16                 "hobby": {
17                     "query": "音乐",
18                     "boost": 10
19                 }
20             }
21         },
22         {
23             "match": {
24                 "hobby": {
25                     "query": "跑步",
26                     "boost": 2
27                 }
28             }
29         }
30     ]
31 }
32 },
33 "highlight": {
34     "fields": {
35         "hobby": {}
36     }
37 }
38 }
```

结果：



如果不设置权重的查询结果是这样：



```
{
  "took": 2,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 2,
    "max_score": 3.630794,
    "hits": [
      {
        "_index": "itcast",
        "_type": "person",
        "_id": "Vf0gDWgBR-bSw8-L0dm_",
        "_score": 3.630794,
        "_source": {
          "name": "赵六",
          "age": 23,
          "mail": "444@qq.com",
          "hobby": "跑步、游泳、篮球"
        },
        "highlight": {
          "hobby": [
            "<em>跑步</em>、<em>游泳</em>、<em>篮球</em>"
          ]
        }
      },
      {
        "_index": "itcast",
        "_type": "person",
        "_id": "Uv0cDWgBR-bSw8-LpdkZ",
        "_score": 2.135749,
        "_source": {
          "name": "王五",
          "age": 22,
          "mail": "333@qq.com",
          "hobby": "羽毛球、篮球、游泳、听音乐"
        },
        "highlight": {
          "hobby": [
            "羽毛球、<em>篮球</em>、<em>游泳</em>、听<em>音乐</em>"
          ]
        }
      }
    ]
  }
}
```

1.3、短语匹配

在Elasticsearch中，短语匹配意味着不仅仅是词要匹配，并且词的顺序也要一致，如下：

```
1 POST http://172.16.55.185:9200/itcast/person/_search
2 {
3   "query": {
4     "match_phrase": {
5       "hobby": {
6         "query": "羽毛球篮球"
7       }
8     }
9   },
10  "highlight": {
11    "fields": {
12      "hobby": {}
13    }
14  }
15 }
```

这种是，要求两个
词条都是必须挨着
一起的，
2. 要想可以不是挨
着也行，如下加
sl op



结果：

```
{
  "took": 11,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "max_score": 1.307641
  },
  "hits": {
    "total": 1,
    "max_score": 1.307641,
    "hits": [
      {
        "_index": "itcast",
        "_type": "person",
        "_id": "Uv0cDWgBR-bSw8-LpdkZ",
        "_score": 1.307641,
        "_source": {
          "name": "王五",
          "age": 22,
          "mail": "333@qq.com",
          "hobby": "羽毛球、篮球、游泳、听音乐"
        },
        "highlight": {
          "hobby": [
            "<em>羽毛</em><em>球</em>、<em>篮球</em>、游泳、听音乐"
          ]
        }
      }
    ]
  }
}
```

结果符合预期。

如果觉得这样太过于苛刻，可以增加slop参数，允许跳过N个词进行匹配。

示例：

```
1 POST http://172.16.55.185:9200/itcast/person/_search
2 {
3   "query": {
4     "match_phrase": {
5       "hobby": {
6         "query": "羽毛球足球"
7       }
8     }
9   },
10  "highlight": {
11    "fields": {
12      "hobby": {}
13    }
14  }
15 }
```

结果：没有匹配到数据

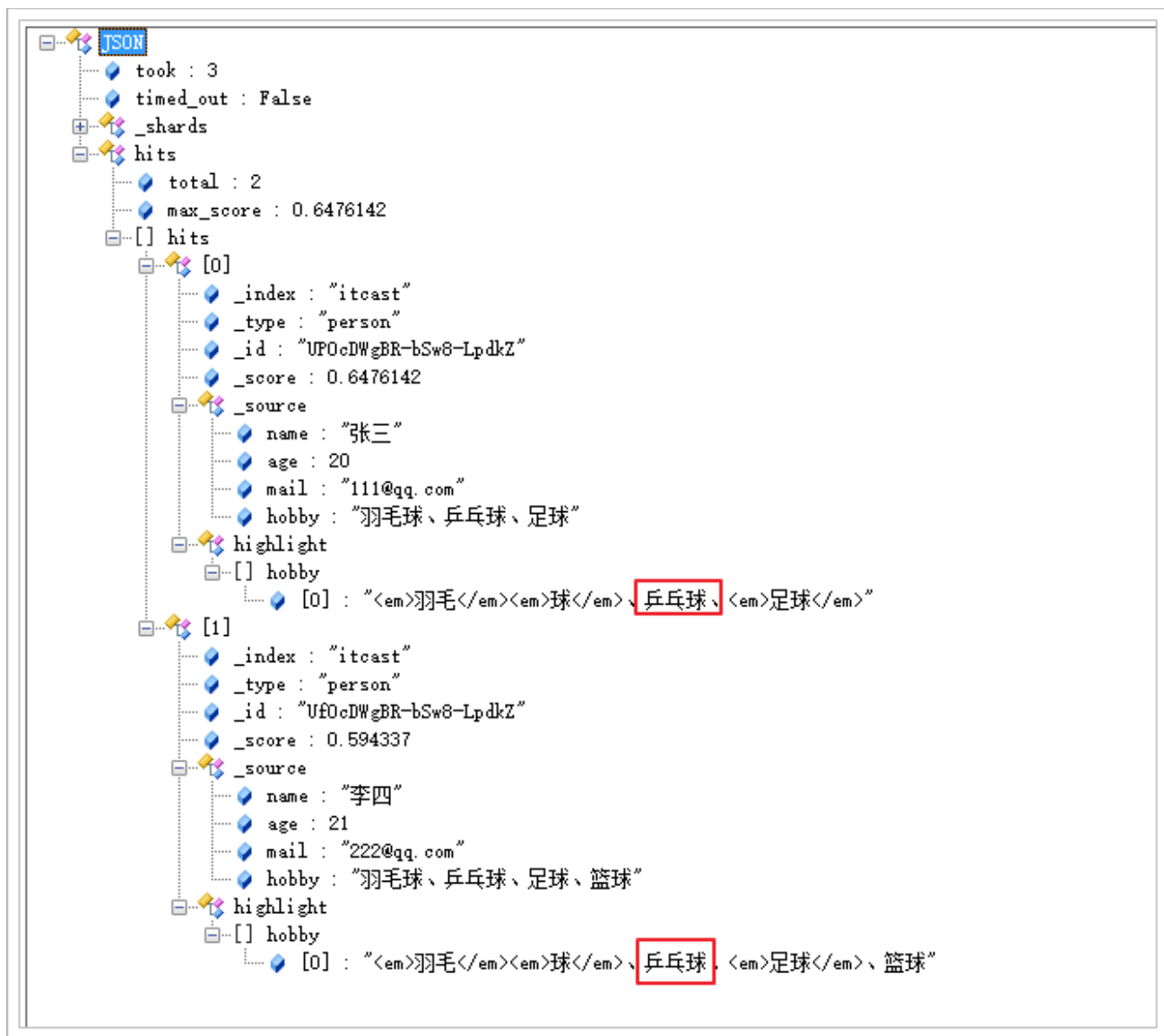


```
1 {  
2   "took": 2,  
3   "timed_out": false,  
4   "_shards": {  
5     "total": 1,  
6     "successful": 1,  
7     "skipped": 0,  
8     "failed": 0  
9   },  
10  "hits": {  
11    "total": 0,  
12    "max_score": null,  
13    "hits": []  
14  }  
15 }
```

设置跳过次数：

```
1 {  
2   "query": {  
3     "match_phrase": {  
4       "hobby": {  
5         "query": "羽毛球足球",  
6         "slop": 3  
7       }  
8     }  
9   },  
10  "highlight": {  
11    "fields": {  
12      "hobby": {}  
13    }  
14  }  
15 }
```

结果：



从结果中，可以看出已经跳过了“乒乓球”这个词。

2、Elasticsearch集群

2.1、集群节点

Elasticsearch的集群是由多个节点组成的，通过cluster.name设置集群名称，并且用于区分其它的集群，每个节点通过node.name指定节点的名称。

在Elasticsearch中，节点的类型主要有4种：

- master节点
 - 配置文件中node.master属性为true(默认为true)，就有资格被选为master节点。
 - master节点用于控制整个集群的操作。比如创建或删除索引，管理其它非master节点等。
- data节点
 - 配置文件中node.data属性为true(默认为true)，就有资格被设置成data节点。
 - data节点主要用于执行数据相关的操作。比如文档的CRUD。
- 客户端节点



- 配置文件中node.master属性和node.data属性均为false。
- 该节点不能作为master节点，也不能作为data节点。
- 可以作为客户端节点，用于响应用户的请求，把请求转发到其他节点
- 部落节点
 - 当一个节点配置tribe.*的时候，它是一个特殊的客户端，它可以连接多个集群，在所有连接的集群上执行搜索和其他操作。

2.2、使用docker搭建集群

```
1  mkdir /haoke/es-cluster
2  cd /haoke/es-cluster
3  mkdir node01
4  mkdir node02
5
6  #复制安装目录下的elasticsearch.yml、jvm.options文件，做如下修改
7  #node01的配置：
8  cluster.name: es-itcast-cluster
9  node.name: node01
10 node.master: true
11 node.data: true
12 network.host: 172.16.55.185
13 http.port: 9200
14 discovery.zen.ping.unicast.hosts: ["172.16.55.185"]
15 discovery.zen.minimum_master_nodes: 1
16 http.cors.enabled: true
17 http.cors.allow-origin: "*"
18
19 #node02的配置：
20 cluster.name: es-itcast-cluster
21 node.name: node02
22 node.master: false
23 node.data: true
24 network.host: 172.16.55.185
25 http.port: 9201
26 discovery.zen.ping.unicast.hosts: ["172.16.55.185"]
27 discovery.zen.minimum_master_nodes: 1
28 http.cors.enabled: true
29 http.cors.allow-origin: "*"
30
31
32 #创建容器
33 docker create --name es-node01 --net host -v /haoke/es-
cluster/node01/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml
-v /haoke/es-cluster/node01/jvm.options:/usr/share/elasticsearch/config/jvm.options
-v /haoke/es-cluster/node01/data:/usr/share/elasticsearch/data elasticsearch:6.5.4
34
35 docker create --name es-node02 --net host -v /haoke/es-
cluster/node02/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml
-v /haoke/es-cluster/node02/jvm.options:/usr/share/elasticsearch/config/jvm.options
-v /haoke/es-cluster/node02/data:/usr/share/elasticsearch/data elasticsearch:6.5.4
36
37 #启动容器
```

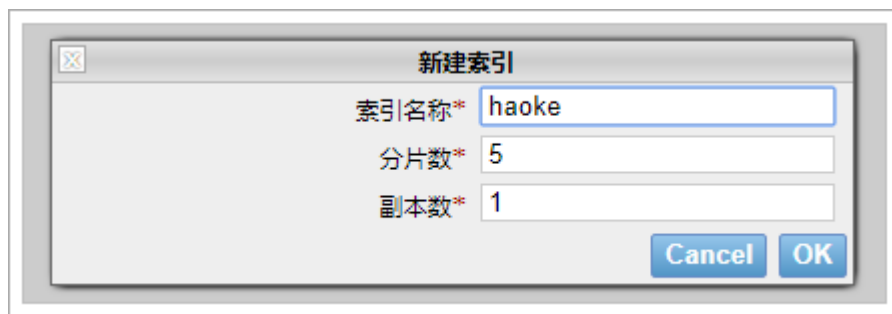


```
38 docker start es-node01 && docker logs -f es-node01
39 docker start es-node02 && docker logs -f es-node02
40
41 #提示：启动时会报文件无权限操作的错误，需要对node01和node02进行chmod 777 的操作
```

查看集群：



创建索引：



查询集群状态：http://172.16.55.185:9200/_cluster/health

响应：

```
1 {
2   "cluster_name": "es-itcast-cluster",
```

```
3    "status": "green",
4    "timed_out": false,
5    "number_of_nodes": 2,
6    "number_of_data_nodes": 2,
7    "active_primary_shards": 5,
8    "active_shards": 10,
9    "relocating_shards": 0,
10   "initializing_shards": 0,
11   "unassigned_shards": 0,
12   "delayed_unassigned_shards": 0,
13   "number_of_pending_tasks": 0,
14   "number_of_in_flight_fetch": 0,
15   "task_max_waiting_in_queue_millis": 0,
16   "active_shards_percent_as_number": 100
17 }
```

集群状态的三种颜色：

颜色	意义
green	所有主要分片和复制分片都可用
yellow	所有主要分片可用，但不是所有复制分片都可用
red	不是所有的主要分片都可用

2.3、分片和副本

为了将数据添加到Elasticsearch，我们需要索引(index)——一个存储关联数据的地方。实际上，索引只是一个用来指向一个或多个分片(shards)的“逻辑命名空间(logical namespace)”。

- 一个分片(shard)是一个最小级别“工作单元(worker unit)”，它只是保存了索引中所有数据的一部分。
- 我们需要知道分片就是一个Lucene实例，并且它本身就是一个完整的搜索引擎。应用程序不会和它直接通信。
- 分片可以是主分片(primary shard)或者是复制分片(replica shard)。
- 索引中的每个文档属于一个单独的主分片，所以主分片的数量决定了索引最多能存储多少数据。
- 复制分片只是主分片的一个副本，它可以防止硬件故障导致的数据丢失，同时可以提供读请求，比如搜索或者从别的shard取回文档。
- 当索引创建完成的时候，主分片的数量就固定了，但是复制分片的数量可以随时调整。

2.4、故障转移

为了测试故障转移，需要再向集群中添加一个节点，并且将所有节点的node.master设置为true。



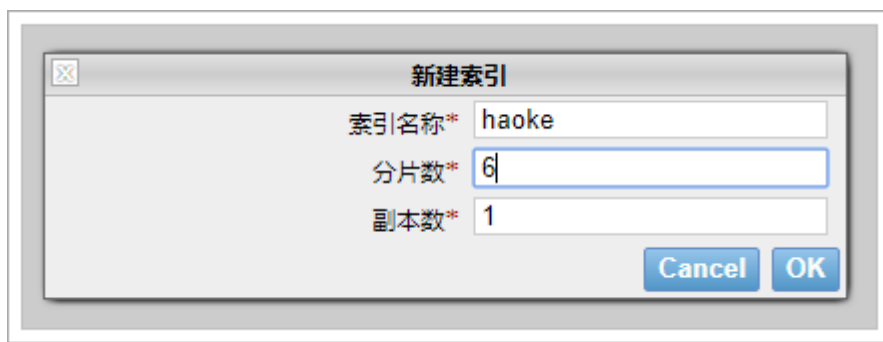
```
1 docker create --name es-node03 --net host -v /haoke/es-  
cluster/node03/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml -v  
/haoke/es-cluster/node03/jvm.options:/usr/share/elasticsearch/config/jvm.options -v  
/haoke/es-cluster/node03/data:/usr/share/elasticsearch/data elasticsearch:6.5.4  
2  
3 docker stop es-node01 es-node02  
4 docker start es-node01 es-node02 es-node03
```

查看集群状态：（需要将之前创建haoke索引删除）



此时，node02为主节点。

创建haoke索引：





2.4.1、将node01停止

```
1 | docker stop es-node01
```



说明：

- 需要连接到node2的端口9201进行查看状态
- 当前集群状态为黄色，表示主节点可用，副本节点不完全可用

过一段时间观察，发现节点列表中看不到node01，副本节点分配到了node02和node03，集群状态恢复到绿色。



将node02恢复：

```
1 | docker start es-node01
```



可以看到，node01恢复后，重新加入了集群，并且重新分配了节点信息。

2.4.2、将node02停止

接下来，测试将node02停止，也就是将主节点停止。

```
1 | docker stop es-node02
```



Elasticsearch cluster health status: yellow (8 of 12). The cluster is named 'haoke' and has a size of 1.41ki (1.86ki) and 0 docs. The cluster is in a 'yellow' state, indicating some nodes are not assigned. The nodes listed are 'node01' and 'node03'. 'node01' has 4 shards (0, 1, 2, 3) and 'node03' has 4 shards (0, 2, 4, 5). There is an 'Unassigned' section with 4 shards (1, 3, 4, 5).

从结果中可以看出，集群对master进行了重新选举，选择node01为master。并且集群状态变成黄色。

Elasticsearch cluster health status: green (12 of 12). The cluster is named 'haoke' and has a size of 1.41ki (2.82ki) and 0 docs. The cluster is in a 'green' state, indicating all nodes are assigned. The nodes listed are 'node01' and 'node03'. 'node01' has 6 shards (0, 1, 2, 3, 4, 5) and 'node03' has 6 shards (0, 1, 2, 3, 4, 5).

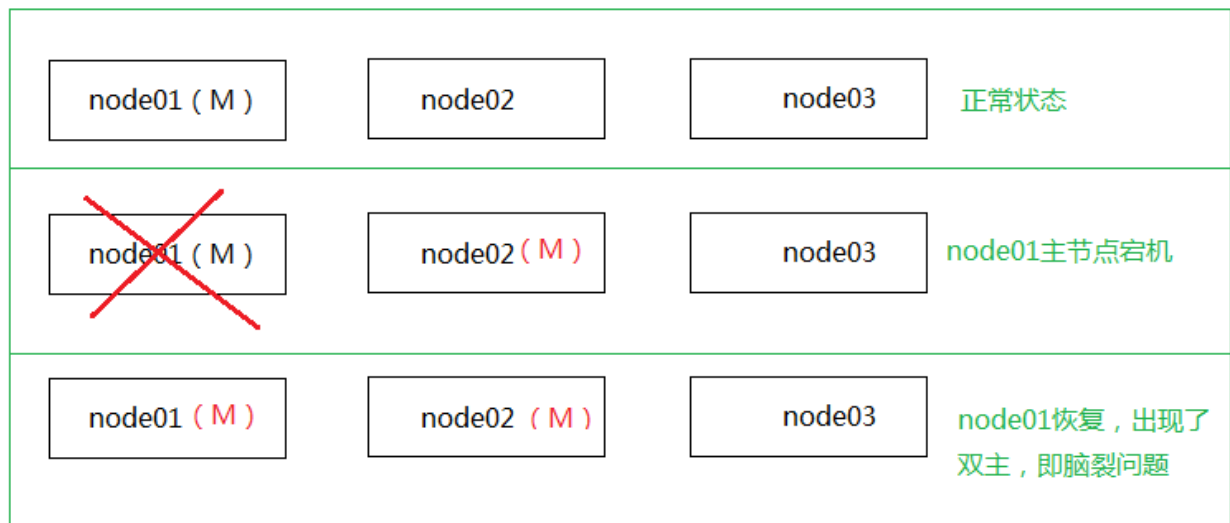
集群状态从黄色变为了绿色。

恢复node02节点：

```
1 | docker start es-node02
```

重启之后，发现node02加不到集群中了。这其实是集群中脑裂问题。

2.4.3、脑裂问题

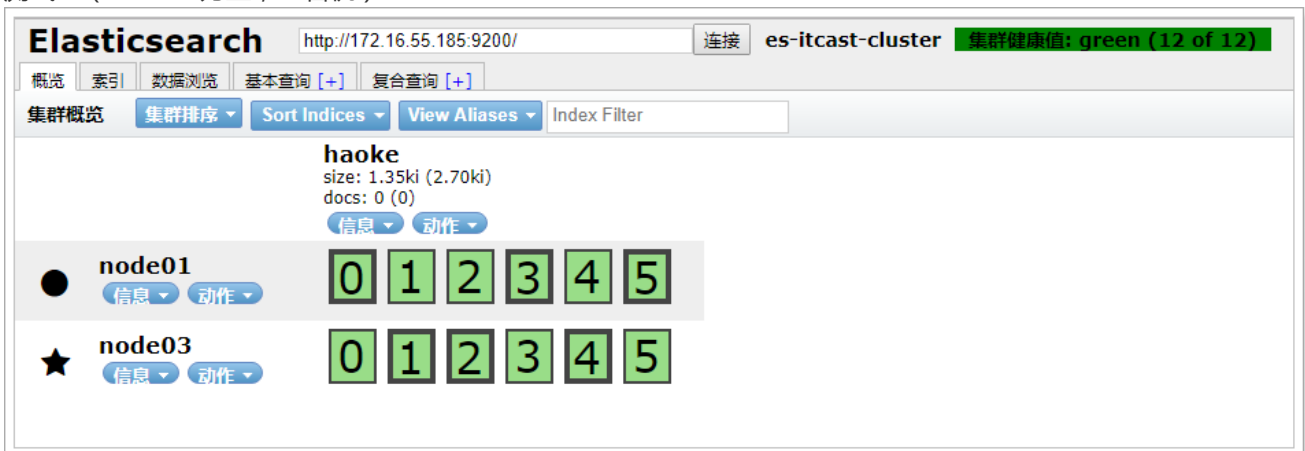


解决方案：

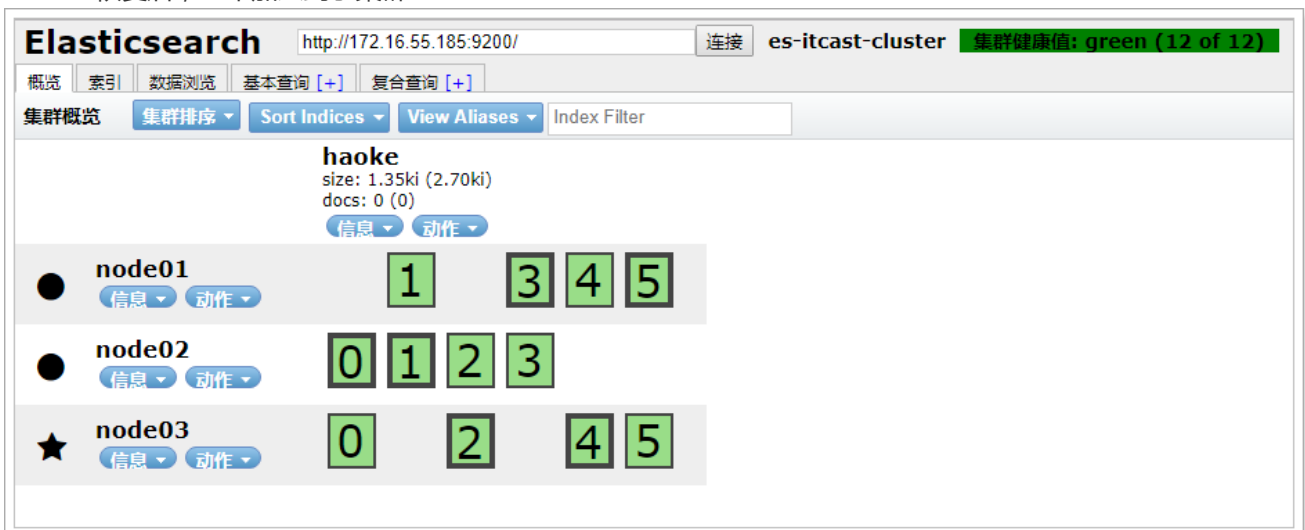
```
discovery.zen.minimum_master_nodes: 1
```

- 思路：不能让节点很容易的变成master，必须有多个节点认可后才可以。
- 设置minimum_master_nodes的大小为2
 - 官方推荐： $(N/2)+1$ ，N为集群中节点数

测试：（node02为主，一宕机）



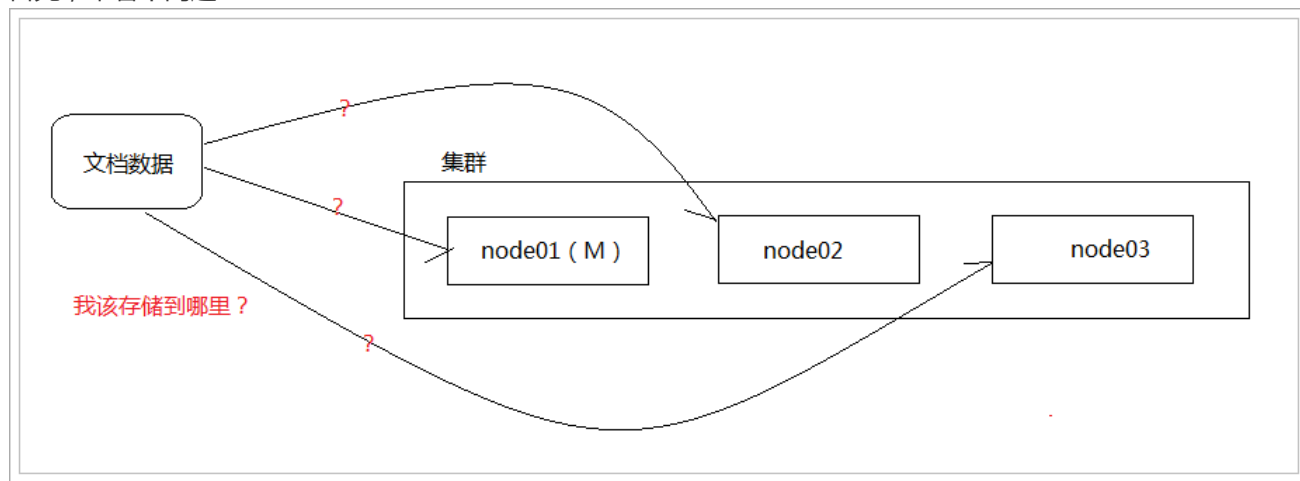
node02恢复后，正常加入到了集群：



2.5、分布式文档

2.5.1、路由

首先，来看个问题：



如图所示：当我们想一个集群保存文档时，文档该存储到哪个节点呢？是随机吗？是轮询吗？

实际上，在Elasticsearch中，会采用计算的方式来确定存储到哪个节点，计算公式如下：

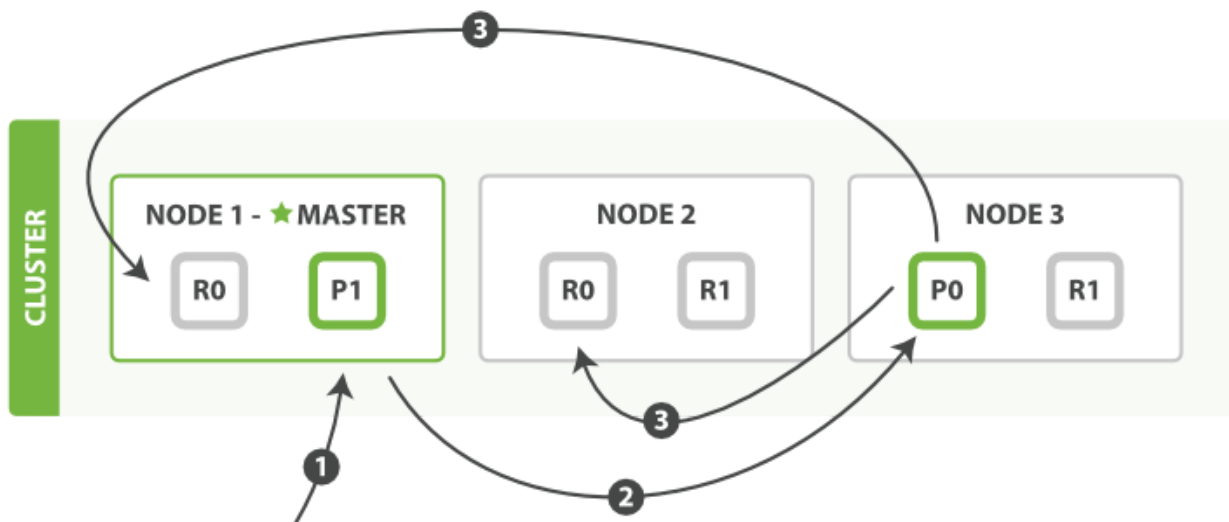
```
1 | shard = hash(routing) % number_of_primary_shards
```

- routing值是一个任意字符串，它默认是_id但也可以自定义。
- 这个routing字符串通过哈希函数生成一个数字，然后除以主切片的数量得到一个余数(remainder)，余数的范围永远是0到number_of_primary_shards - 1，这个数字就是特定文档所在的分片。

这就是为什么创建了主分片后，不能修改的原因。

2.5.2、文档的写操作

新建、索引和删除请求都是写(write)操作，它们必须在主分片上成功完成才能复制到相关的复制分片上。



下面我们罗列在主分片和复制分片上成功新建、索引或删除一个文档必要的顺序步骤：

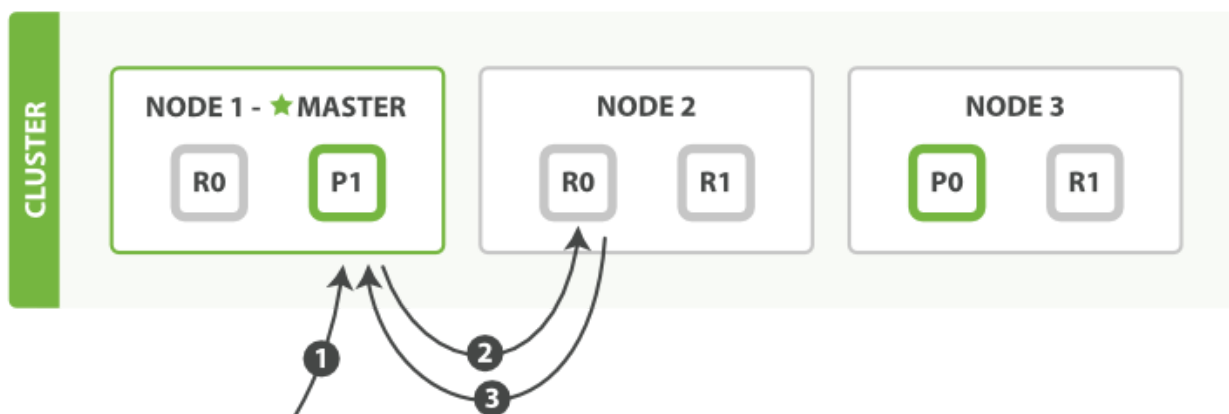
1. 客户端给 Node 1 发送新建、索引或删除请求。
2. 节点使用文档的 `_id` 确定文档属于分片 0。它转发请求到 Node 3，分片 0 位于这个节点上。
3. Node 3 在主分片上执行请求，如果成功，它转发请求到相应的位于 Node 1 和 Node 2 的复制节点上。当所有的复制节点报告成功，Node 3 报告成功到请求的节点，请求的节点再报告给客户端。

客户端接收到成功响应的时候，文档的修改已经被应用于主分片和所有的复制分片。你的修改生效了。

2.5.3、搜索文档（单个文档）

1个数据存在一个分片

文档能够从主分片或任意一个复制分片被检索。



下面我们罗列在主分片或复制分片上检索一个文档必要的顺序步骤：

1. 客户端给 Node 1 发送get请求。
2. 节点使用文档的 `_id` 确定文档属于分片 0。分片 0 对应的复制分片在三个节点上都有。此时，它转发请求到 Node 2。
3. Node 2 返回文档(document)给 Node 1 然后返回给客户端。

对于读请求，为了平衡负载，请求节点会为每个请求选择不同的分片——它会循环所有分片副本。

可能的情况是，一个被索引的文档已经存在于主分片上却还没来得及同步到复制分片上。这时复制分片会报告文档未找到，主分片会成功返回文档。一旦索引请求成功返回给用户，文档则在主分片和复制分片都是可用的。

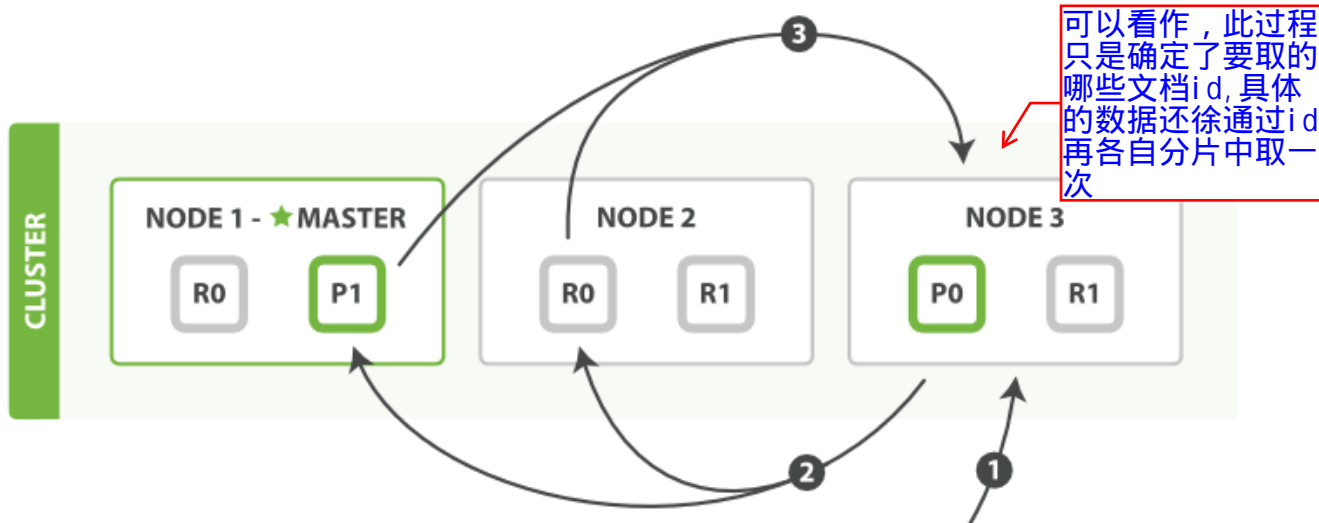
2.5.4、全文搜索

即针对多个文档一起的搜索，比单个复杂，因为一次搜索的数据可能存在于多个分片中，

对于全文搜索而言，文档可能分散在各个节点上，那么在分布式的情况下，如何搜索文档呢？

搜索，分为2个阶段，搜索（query）+取回（fetch）。

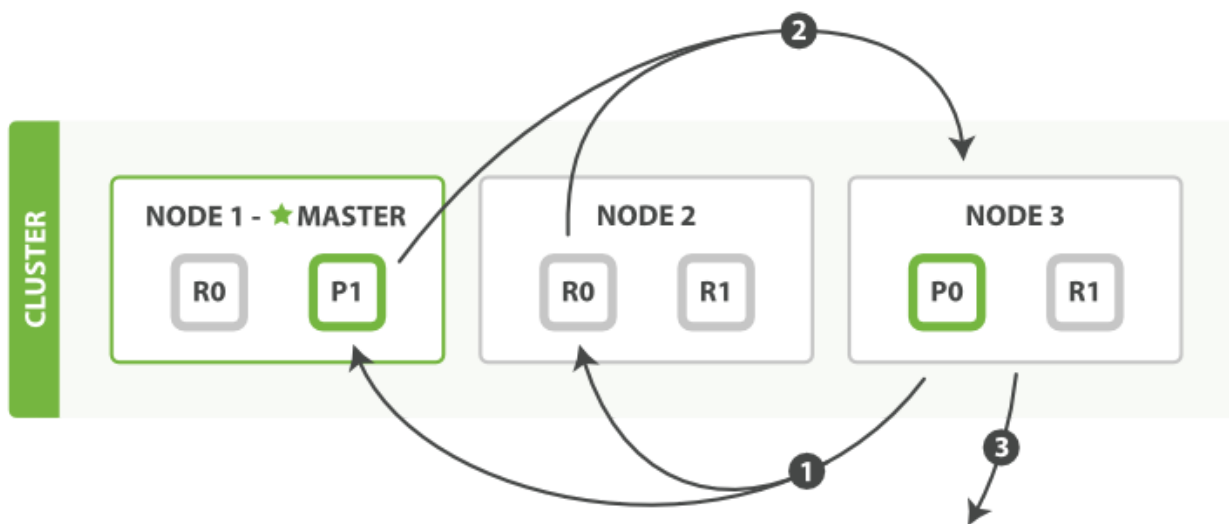
搜索（query）



查询阶段包含以下三步：

1. 客户端发送一个 `search` (搜索) 请求给 Node 3, Node 3 创建了一个长度为 `from+size` 的空优先级队
2. Node 3 转发这个搜索请求到索引中每个分片的原本或副本。每个分片在本地执行这个查询并且结果将结果到一个大小为 `from+size` 的有序本地优先队列里去。
3. 每个分片返回document的ID和它优先队列里的所有document的排序值给协调节点 Node 3。Node 3 把这些值合并到自己的优先队列里产生全局排序结果。

取回 (fetch)



分发阶段由以下步骤构成：

1. 协调节点辨别出哪个document需要取回，并且向相关分片发出 `GET` 请求。
2. 每个分片加载document并且根据需要丰富 (*enrich*) 它们，然后再将document返回协调节点。
3. 一旦所有的document都被取回，协调节点会将结果返回给客户端。

3、Java客户端

在Elasticsearch中，为java提供了2种客户端，一种是REST风格的客户端，另一种是Java API的客户端。

<https://www.elastic.co/guide/en/elasticsearch/client/index.html>

Elasticsearch Clients

- [Java REST Client \[6.5\] — other versions](#)
- [Java API \[6.5\] — other versions](#)
- [JavaScript API](#)
- [Groovy API \[2.4\] — other versions](#)
- [.NET API \[6.x\] — other versions](#)
- [PHP API \[6.0\] — other versions](#)
- [Perl API](#)
- [Python API](#)
- [Ruby API](#)
- [Community Contributed Clients](#)

3.1、REST客户端

Elasticsearch提供了2种REST客户端，一种是低级客户端，一种是高级客户端。

- Java Low Level REST Client：官方提供的低级客户端。该客户端通过http来连接Elasticsearch集群。用户在使用该客户端时需要将请求数据手动拼接成Elasticsearch所需JSON格式进行发送，收到响应时同样也需要将返回的JSON数据手动封装成对象。虽然麻烦，不过该客户端兼容所有的Elasticsearch版本。
- Java High Level REST Client：官方提供的高级客户端。该客户端基于低级客户端实现，它提供了很多便捷的API来解决低级客户端需要手动转换数据格式的问题。

3.2、构造数据

```
1 POST http://172.16.55.185:9200/haoke/house/_bulk
2
3 {"index":{"_index":"haoke","_type":"house"}}
4 {"id":"1001","title":"整租 · 南丹大楼 1居室 7500","price":"7500"}
5 {"index":{"_index":"haoke","_type":"house"}}
6 {"id":"1002","title":"陆家嘴板块，精装设计一室一厅，可拎包入住诚意租。","price":"8500"}
7 {"index":{"_index":"haoke","_type":"house"}}
8 {"id":"1003","title":"整租 · 健安坊 1居室 4050","price":"7500"}
9 {"index":{"_index":"haoke","_type":"house"}}
10 {"id":"1004","title":"整租 · 中凯城市之光+视野开阔+景色秀丽+拎包入住","price":"6500"}
11 {"index":{"_index":"haoke","_type":"house"}}
12 {"id":"1005","title":"整租 · 南京西路品质小区 21213三轨交汇 配套齐* 拎包入住","price":"6000"}
13 {"index":{"_index":"haoke","_type":"house"}}
```



```
14 {"id":"1006","title":"祥康里 简约风格 *南户型 拎包入住 看房随时","price":"7000"}
15
```

查询 6 个分片中用的 6 个, 6 命中, 耗时 0.012 秒

_index	_type	_id	_score ▲	id	title	price
haoke	house	F0pdE2gBCKv8opxuOj12	1	1003	整租 · 健安坊 1居室 4050	7500
haoke	house	FUpdE2gBCKv8opxuOj12	1	1001	整租 · 南丹大楼 1居室 7500	7500
haoke	house	GEpdE2gBCKv8opxuOj12	1	1004	整租 · 中凯城市之光+视野开阔+景色秀丽+拎包入住	6500
haoke	house	GkpdE2gBCKv8opxuOj12	1	1006	祥康里 简约风格 *南户型 拎包入住 看房随时	7000
haoke	house	GUpdE2gBCKv8opxuOj12	1	1005	整租 · 南京西路品质小区 21213三轨交汇 配套齐* 拎包入住	6000
haoke	house	FkpdE2gBCKv8opxuOj12	1	1002	陆家嘴板块, 精装设计一室一厅, 可拎包入住诚意租。	8500

3.3、REST低级客户端

3.3.1、创建工程

创建工程itcast-elasticsearch：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>cn.itcast.elasticsearch</groupId>
8     <artifactId>itcast-elasticsearch</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <dependencies>
12         <dependency>
13             <groupId>org.elasticsearch.client</groupId>
14             <artifactId>elasticsearch-rest-client</artifactId>
15             <version>6.5.4</version>
16         </dependency>
17         <dependency>
18             <groupId>junit</groupId>
19             <artifactId>junit</artifactId>
20             <version>4.12</version>
21         </dependency>
22
23         <dependency>
24             <groupId>com.fasterxml.jackson.core</groupId>
25             <artifactId>jackson-databind</artifactId>
26             <version>2.9.4</version>
27         </dependency>
28     </dependencies>
29
30     <build>
31         <plugins>
32             <!-- java编译插件 -->
33             <plugin>
```




```
34         <groupId>org.apache.maven.plugins</groupId>
35         <artifactId>maven-compiler-plugin</artifactId>
36         <version>3.2</version>
37         <configuration>
38             <source>1.8</source>
39             <target>1.8</target>
40             <encoding>UTF-8</encoding>
41         </configuration>
42     </plugin>
43 </plugins>
44 </build>
45 </project>
```

3.3.2、编写测试用例

```
1  package cn.itcast.es.rest;
2
3  import com.fasterxml.jackson.databind.ObjectMapper;
4  import org.apache.http.HttpHost;
5  import org.apache.http.util.EntityUtils;
6  import org.elasticsearch.client.*;
7  import org.junit.After;
8  import org.junit.Before;
9  import org.junit.Test;
10
11 import java.io.IOException;
12 import java.util.HashMap;
13 import java.util.Map;
14
15 public class TestESREST {
16
17     private static final ObjectMapper MAPPER = new ObjectMapper();
18
19     private RestClient restClient;
20
21     @Before
22     public void init() {
23         RestClientBuilder restClientBuilder = RestClient.builder(
24             new HttpHost("172.16.55.185", 9200, "http"),
25             new HttpHost("172.16.55.185", 9201, "http"),
26             new HttpHost("172.16.55.185", 9202, "http"));
27
28         restClientBuilder.setFailureListener(new RestClient.FailureListener() {
29             @Override
30             public void onFailure(Node node) {
31                 System.out.println("出错了 -> " + node);
32             }
33         });
34
35         this.restClient = restClientBuilder.build();
36     }
37
38     @After
```



```
39     public void after() throws IOException {
40         restClient.close();
41     }
42
43     // 查询集群状态
44     @Test
45     public void testGetInfo() throws IOException {
46         Request request = new Request("GET", "/_cluster/state");
47         request.addParameter("pretty", "true");
48         Response response = this.restClient.performRequest(request);
49
50         System.out.println(response.getStatusLine());
51         System.out.println(EntityUtils.toString(response.getEntity()));
52     }
53
54
55     // 新增数据
56     @Test
57     public void testCreateData() throws IOException {
58         Request request = new Request("POST", "/haoke/house");
59
60         Map<String, Object> data = new HashMap<>();
61         data.put("id", "2001");
62         data.put("title", "张江高科");
63         data.put("price", "3500");
64
65         request.setJsonEntity(MAPPER.writeValueAsString(data));
66         Response response = this.restClient.performRequest(request);
67
68         System.out.println(response.getStatusLine());
69         System.out.println(EntityUtils.toString(response.getEntity()));
70     }
71
72     // 根据id查询数据
73     @Test
74     public void testQueryData() throws IOException {
75         Request request = new Request("GET", "/haoke/house/G0pfE2gBCKv8opxuRz1y");
76
77         Response response = this.restClient.performRequest(request);
78
79         System.out.println(response.getStatusLine());
80         System.out.println(EntityUtils.toString(response.getEntity()));
81     }
82
83     // 搜索数据
84     @Test
85     public void testSearchData() throws IOException {
86         Request request = new Request("POST", "/haoke/house/_search");
87         String searchJson = "{ \"query\": { \"match\": { \"title\": \"拎包入住\" } } }";
88         request.setJsonEntity(searchJson);
89         request.addParameter("pretty", "true");
90
91         Response response = this.restClient.performRequest(request);
```



```
92
93     System.out.println(response.getStatusLine());
94     System.out.println(EntityUtils.toString(response.getEntity()));
95 }
96
97
98 }
99
```

从使用中，可以看出，基本和我们使用RESTful api使用几乎是一致的。

3.4、REST高级客户端

3.4.1、引入依赖

```
1 <dependency>
2     <groupId>org.elasticsearch.client</groupId>
3     <artifactId>elasticsearch-rest-high-level-client</artifactId>
4     <version>6.5.4</version>
5 </dependency>
```

3.4.2、编写测试用例

```
1 package cn.itcast.es.rest;
2
3 import org.apache.http.HttpHost;
4 import org.elasticsearch.action.ActionListener;
5 import org.elasticsearch.action.delete.DeleteRequest;
6 import org.elasticsearch.action.delete.DeleteResponse;
7 import org.elasticsearch.action.get.GetRequest;
8 import org.elasticsearch.action.get.GetResponse;
9 import org.elasticsearch.action.index.IndexRequest;
10 import org.elasticsearch.action.index.IndexResponse;
11 import org.elasticsearch.action.search.SearchRequest;
12 import org.elasticsearch.action.search.SearchResponse;
13 import org.elasticsearch.action.update.UpdateRequest;
14 import org.elasticsearch.action.update.UpdateResponse;
15 import org.elasticsearch.client.RequestOptions;
16 import org.elasticsearch.client.RestClient;
17 import org.elasticsearch.client.RestClientBuilder;
18 import org.elasticsearch.client.RestHighLevelClient;
19 import org.elasticsearch.common.Strings;
20 import org.elasticsearch.common.unit.TimeValue;
21 import org.elasticsearch.index.query.QueryBuilders;
22 import org.elasticsearch.search.SearchHit;
23 import org.elasticsearch.search.SearchHits;
24 import org.elasticsearch.search.builder.SearchSourceBuilder;
25 import org.elasticsearch.search.fetch.subphase.FetchSourceContext;
26 import org.junit.After;
27 import org.junit.Before;
28 import org.junit.Test;
29
```



```
30 import java.util.HashMap;
31 import java.util.Map;
32 import java.util.concurrent.TimeUnit;
33
34 public class TestRestHighLevel {
35
36     private RestHighLevelClient client;
37
38     @Before
39     public void init() {
40         RestClientBuilder restClientBuilder = RestClient.builder(
41             new HttpHost("172.16.55.185", 9200, "http"),
42             new HttpHost("172.16.55.185", 9201, "http"),
43             new HttpHost("172.16.55.185", 9202, "http"));
44
45         this.client = new RestHighLevelClient(restClientBuilder);
46     }
47
48     @After
49     public void after() throws Exception {
50         this.client.close();
51     }
52
53     /**
54      * 新增文档，同步操作
55      *
56      * @throws Exception
57      */
58     @Test
59     public void testCreate() throws Exception {
60
61         Map<String, Object> data = new HashMap<>();
62         data.put("id", "2002");
63         data.put("title", "南京西路 拎包入住 一室一厅");
64         data.put("price", "4500");
65
66         IndexRequest indexRequest = new IndexRequest("haoke", "house")
67             .source(data);
68
69         IndexResponse indexResponse = this.client.index(indexRequest,
70             RequestOptions.DEFAULT);
71         System.out.println("id->" + indexResponse.getId());
72         System.out.println("index->" + indexResponse.getIndex());
73         System.out.println("type->" + indexResponse.getType());
74         System.out.println("version->" + indexResponse.getVersion());
75         System.out.println("result->" + indexResponse.getResult());
76         System.out.println("shardInfo->" + indexResponse.getShardInfo());
77     }
78
79     /**
80      * 新增文档，异步操作
81      */

```



```
82      * @throws Exception
83      */
84      @Test
85      public void testCreateAsync() throws Exception {
86
87          Map<String, Object> data = new HashMap<>();
88          data.put("id", "2003");
89          data.put("title", "南京东路 最新房源 二室一厅");
90          data.put("price", "5500");
91
92          IndexRequest indexRequest = new IndexRequest("haoke", "house")
93              .source(data);
94
95          this.client.indexAsync(indexRequest, RequestOptions.DEFAULT, new
ActionListener<IndexResponse>() {
96
97              @Override
98              public void onResponse(IndexResponse indexResponse) {
99                  System.out.println("id->" + indexResponse.getId());
100                 System.out.println("index->" + indexResponse.getIndex());
101                 System.out.println("type->" + indexResponse.getType());
102                 System.out.println("version->" + indexResponse.getVersion());
103                 System.out.println("result->" + indexResponse.getResult());
104                 System.out.println("shardInfo->" + indexResponse.getShardInfo());
105             }
106
107             @Override
108             public void onFailure(Exception e) {
109                 System.out.println(e);
110             }
111         });
112
113         Thread.sleep(20000);
114     }
115
116     @Test
117     public void testQuery() throws Exception {
118         GetRequest getRequest = new GetRequest("haoke", "house",
"GkpdE2gBCKv8opxuOj12");
119
120         // 指定返回的字段
121         String[] includes = new String[]{"title", "id"};
122         String[] excludes = Strings.EMPTY_ARRAY;
123         FetchSourceContext fetchSourceContext =
124             new FetchSourceContext(true, includes, excludes);
125         getRequest.fetchSourceContext(fetchSourceContext);
126
127         GetResponse response = this.client.get(getRequest,
RequestOptions.DEFAULT);
128
129         System.out.println("数据 -> " + response.getSource());
130     }
131 }
```



```
132
133     /**
134     * 判断是否存在
135     *
136     * @throws Exception
137     */
138     @Test
139     public void testExists() throws Exception {
140         GetRequest getRequest = new GetRequest("haoke", "house",
141         "GkpdE2gBCKv8opxuOj12");
142
143         // 不返回的字段
144         getRequest.fetchSourceContext(new FetchSourceContext(false));
145
146         boolean exists = this.client.exists(getRequest, RequestOptions.DEFAULT);
147
148         System.out.println("exists -> " + exists);
149     }
150
151     /**
152     * 删除数据
153     *
154     * @throws Exception
155     */
156     @Test
157     public void testDelete() throws Exception {
158         DeleteRequest deleteRequest = new DeleteRequest("haoke", "house",
159         "GkpdE2gBCKv8opxuOj12");
160         DeleteResponse response = this.client.delete(deleteRequest,
161         RequestOptions.DEFAULT);
162         System.out.println(response.status()); // OK or NOT_FOUND
163     }
164
165     /**
166     * 更新数据
167     *
168     * @throws Exception
169     */
170     @Test
171     public void testUpdate() throws Exception {
172         UpdateRequest updateRequest = new UpdateRequest("haoke", "house",
173         "G0pfE2gBCKv8opxuRz1y");
174
175         Map<String, Object> data = new HashMap<>();
176         data.put("title", "张江高科2");
177         data.put("price", "5000");
178
179         updateRequest.doc(data);
180
181         UpdateResponse response = this.client.update(updateRequest,
182         RequestOptions.DEFAULT);
183         System.out.println("version -> " + response.getVersion());
184     }
185 }
```



```
180
181     /**
182     * 测试搜索
183     *
184     * @throws Exception
185     */
186     @Test
187     public void testSearch() throws Exception {
188         SearchRequest searchRequest = new SearchRequest("haoke");
189         searchRequest.types("house");
190
191         SearchSourceBuilder sourceBuilder = new SearchSourceBuilder();
192         sourceBuilder.query(QueryBuilders.matchQuery("title", "拎包入住"));
193         sourceBuilder.from(0);
194         sourceBuilder.size(5);
195         sourceBuilder.timeout(new TimeValue(60, TimeUnit.SECONDS));
196
197         searchRequest.source(sourceBuilder);
198
199         SearchResponse search = this.client.search(searchRequest,
200 RequestOptions.DEFAULT);
201         System.out.println("搜索到 " + search.getHits().totalHits + " 条数据.");
202         SearchHits hits = search.getHits();
203         for (SearchHit hit : hits) {
204             System.out.println(hit.getSourceAsString());
205         }
206     }
207 }
208
```

4、Spring Data Elasticsearch

Spring Data项目对Elasticsearch做了支持，其目的就是简化对Elasticsearch的操作。

地址：<https://spring.io/projects/spring-data-elasticsearch>

4.1、导入依赖

这里采用SpringBoot整合的方式进行。

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7
8     <parent>
9         <groupId>org.springframework.boot</groupId>
10        <artifactId>spring-boot-starter-parent</artifactId>
11        <version>2.1.0.RELEASE</version>
```




```
11     </parent>
12
13     <groupId>cn.itcast.elasticsearch</groupId>
14     <artifactId>itcast-elasticsearch</artifactId>
15     <version>1.0-SNAPSHOT</version>
16
17     <dependencies>
18         <dependency>
19             <groupId>org.springframework.boot</groupId>
20             <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
21         </dependency>
22         <dependency>
23             <groupId>org.springframework.boot</groupId>
24             <artifactId>spring-boot-starter-test</artifactId>
25             <scope>test</scope>
26         </dependency>
27         <dependency>
28             <groupId>org.elasticsearch.client</groupId>
29             <artifactId>elasticsearch-rest-client</artifactId>
30             <version>6.5.4</version>
31         </dependency>
32         <dependency>
33             <groupId>org.elasticsearch.client</groupId>
34             <artifactId>elasticsearch-rest-high-level-client</artifactId>
35             <version>6.5.4</version>
36         </dependency>
37         <dependency>
38             <groupId>junit</groupId>
39             <artifactId>junit</artifactId>
40             <version>4.12</version>
41         </dependency>
42
43         <dependency>
44             <groupId>com.fasterxml.jackson.core</groupId>
45             <artifactId>jackson-databind</artifactId>
46             <version>2.9.4</version>
47         </dependency>
48
49         <dependency>
50             <groupId>org.projectlombok</groupId>
51             <artifactId>lombok</artifactId>
52             <version>1.18.4</version>
53         </dependency>
54     </dependencies>
55
56     <build>
57         <plugins>
58             <!-- java编译插件 -->
59             <plugin>
60                 <groupId>org.apache.maven.plugins</groupId>
61                 <artifactId>maven-compiler-plugin</artifactId>
62                 <version>3.2</version>
63                 <configuration>
```



```
64         <source>1.8</source>
65         <target>1.8</target>
66         <encoding>UTF-8</encoding>
67     </configuration>
68 </plugin>
69 </plugins>
70 </build>
71 </project>
```

4.2、编写application.properties

```
1 spring.application.name = itcast-elasticsearch
2
3 spring.data.elasticsearch.cluster-name=es-itcast-cluster
4 spring.data.elasticsearch.cluster-
  nodes=172.16.55.185:9300,172.16.55.185:9301,172.16.55.185:9302
```



这里要注意，使用的端口是9300，而并非9200，原因是9200是RESTful端口，9300是API端口。

4.3、编写启动类

```
1 package cn.itcast.es;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class MyApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(MyApplication.class, args);
11     }
12
13 }
```

4.4、编写测试用例

4.4.1、编写User对象

```
1 package cn.itcast.es.pojo;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6 import org.springframework.data.annotation.Id;
7 import org.springframework.data.elasticsearch.annotations.Document;
8 import org.springframework.data.elasticsearch.annotations.Field;
9
10 @Data
11 @AllArgsConstructor
12 @NoArgsConstructor
```



```
13 @Document(indexName = "itcast", type = "user", shards = 6, replicas = 1)
14 public class User {
15
16     @Id
17     private Long id;
18
19     @Field(store = true)
20     private String name;
21
22     @Field
23     private Integer age;
24
25     @Field
26     private String hobby;
27
28 }
```

4.4.2、新增数据

```
1 package cn.itcast.es;
2
3 import cn.itcast.es.pojo.User;
4 import org.junit.Test;
5 import org.junit.runner.RunWith;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.boot.test.context.SpringBootTest;
8 import org.springframework.data.elasticsearch.core.ElasticsearchTemplate;
9 import org.springframework.data.elasticsearch.core.query.IndexQuery;
10 import org.springframework.data.elasticsearch.core.query.IndexQueryBuilder;
11 import org.springframework.test.context.junit4.SpringRunner;
12
13 @RunWith(SpringRunner.class)
14 @SpringBootTest
15 public class TestSpringBootES {
16
17     @Autowired
18     private ElasticsearchTemplate elasticsearchTemplate;
19
20     @Test
21     public void testSave(){
22         User user = new User();
23         user.setId(1001L);
24         user.setAge(20);
25         user.setName("张三");
26         user.setHobby("足球、篮球、听音乐");
27
28         IndexQuery indexQuery = new IndexQueryBuilder().withObject(user).build();
29         String index = this.elasticsearchTemplate.index(indexQuery);
30         System.out.println(index);
31     }
32 }
33
```

运行效果：



查询 5 个分片中用的 5 个, 1 命中, 耗时 0.010 秒

_index	_type	_id	_score ▲	id	name	age	hobby
itcast	user	1001	1	1001	张三	20	足球、篮球、听音乐

4.4.3、批量插入数据

```

1  @Test
2      public void testBulk() {
3          List list = new ArrayList();
4          for (int i = 0; i < 5000; i++) {
5              User user = new User();
6              user.setId(1001L + i);
7              user.setAge(i % 50 + 10);
8              user.setName("张三" + i);
9              user.setHobby("足球、篮球、听音乐");
10
11              IndexQuery indexQuery = new
12                  IndexQueryBuilder().withObject(user).build();
13
14              list.add(indexQuery);
15          }
16          Long start = System.currentTimeMillis();
17          this.elasticsearchTemplate.bulkIndex(list);
18          System.out.println("用时：" + (System.currentTimeMillis() - start)); //用时：
19          7836
20      }
    
```



查询 11 个分片中用的 11 个, 5010 命中, 耗时 0.053 秒

_index	_type	_id	_score ▲	id	name	age	hobby
itcast	user	1001	1	1001	张三0	10	足球、篮球、听音乐
itcast	user	1004	1	1004	张三3	13	足球、篮球、听音乐
itcast	user	1006	1	1006	张三5	15	足球、篮球、听音乐
itcast	user	1007	1	1007	张三6	16	足球、篮球、听音乐
itcast	user	1013	1	1013	张三12	22	足球、篮球、听音乐
itcast	user	1032	1	1032	张三31	41	足球、篮球、听音乐
itcast	user	1033	1	1033	张三32	42	足球、篮球、听音乐
itcast	user	1035	1	1035	张三34	44	足球、篮球、听音乐
itcast	user	1036	1	1036	张三35	45	足球、篮球、听音乐
itcast	user	1044	1	1044	张三43	53	足球、篮球、听音乐
itcast	user	1045	1	1045	张三44	54	足球、篮球、听音乐
itcast	user	1052	1	1052	张三51	11	足球、篮球、听音乐
itcast	user	1054	1	1054	张三53	13	足球、篮球、听音乐
itcast	user	1055	1	1055	张三54	14	足球、篮球、听音乐
itcast	user	1061	1	1061	张三60	20	足球、篮球、听音乐
itcast	user	1064	1	1064	张三63	23	足球、篮球、听音乐
itcast	user	1069	1	1069	张三68	28	足球、篮球、听音乐
itcast	user	1073	1	1073	张三72	32	足球、篮球、听音乐
itcast	user	1075	1	1075	张三74	34	足球、篮球、听音乐
itcast	user	1082	1	1082	张三81	41	足球、篮球、听音乐

4.4.4、更新数据

```
1  /**
2   * 局部更新，全部更新使用index覆盖即可
3   */
4   @Test
5   public void testUpdate() {
6       IndexRequest indexRequest = new IndexRequest();
7       indexRequest.source("age", "30");
8
9       UpdateQuery updateQuery = new UpdateQueryBuilder()
10          .withId("1001")
11          .withClass(User.class)
12          .withIndexRequest(indexRequest).build();
13
14       this.elasticsearchTemplate.update(updateQuery);
15   }
```

4.4.5、删除数据



```
1 @Test
2 public void testDelete(){
3     this.elasticsearchTemplate.delete(User.class, "1001");
4 }
```

4.4.6、搜索

```
1 @Test
2 public void testSearch(){
3     PageRequest pageRequest = PageRequest.of(1,10); //设置分页参数
4     SearchQuery searchQuery = new NativeSearchQueryBuilder()
5         .withQuery(QueryBuilders.matchQuery("name", "张三")) // match查询
6         .withPageable(pageRequest)
7         .build();
8     AggregatedPage<User> users =
9     this.elasticsearchTemplate.queryForPage(searchQuery, User.class);
10    System.out.println("总页数：" + users.getTotalPages()); //获取总页数
11    for (User user : users.getContent()) { // 获取搜索到的数据
12        System.out.println(user);
13    }
14 }
```

效果：

```
总页数：500
User(id=1052, name=张三51, age=11, hobby=足球、篮球、听音乐)
User(id=1054, name=张三53, age=13, hobby=足球、篮球、听音乐)
User(id=1055, name=张三54, age=14, hobby=足球、篮球、听音乐)
User(id=1061, name=张三60, age=20, hobby=足球、篮球、听音乐)
User(id=1064, name=张三63, age=23, hobby=足球、篮球、听音乐)
User(id=1069, name=张三68, age=28, hobby=足球、篮球、听音乐)
User(id=1073, name=张三72, age=32, hobby=足球、篮球、听音乐)
User(id=1075, name=张三74, age=34, hobby=足球、篮球、听音乐)
User(id=1082, name=张三81, age=41, hobby=足球、篮球、听音乐)
User(id=1084, name=张三83, age=43, hobby=足球、篮球、听音乐)
```