elasticsearch 第一篇(入门篇)

介绍

elasticsearch是一个高效的、可扩展的全文搜索引擎

基本概念

- Near Realtime(NRT): es是一个接近实时查询平台，意味从存储一条数据到可以索引到数据时差很小，通常在1s内
- Cluster: es是一个分布式、可扩展的平台, 可由一个或多个服务器通过定义的cluster.name（默认为elasticsearch）标识共建同一个集群
- Node: 通常一台服务器上部署一台es node，作为集群的一部分，用于数据的存储和提供搜索功能，在一个集群中节点通过node.name区分，默认在node启动时随机生成一个的字符串做为节点名称，可配置
- Index: 类似于关系型数据库中的database，用于组织一类功能相似的数据，在一个集群中可以定义任意个索引，索引的名称只能由小写字母组成，在数据索引，更新，搜索，删除时作为数据标识的一部分
- Type: 类似于关系型数据库中的table，在Index中可以定义多个Type，原则上一个Type是由相同属性组成的数据集合
- Document: 类似于关系型数据库中的record，是数据的最基本存储单元，使用json形式表示，Document在物理上存储在Index下，但是在逻辑上会分配到具体的Type下
- Shards & Replica:
  一个Index可能存储大量的数据(超过单个节点的硬件限制)，不管是数据存储还是数据索引，为解决数据单节点存储并提高并发，es将每一个Index物理分为多个片，从而水平扩展存储容量，提高并发（可以同时对个shard进行索引和搜索）
  为防止某个存储单元出现故障后数据不能索引的情况，es提供将shard进行复制功能，将主shard出现故障后，复制shard替代主shard进行数据索引操作，已此方式实现其高可用性，因为在搜索时可以使用复制shard，从而提高的数据搜索的并发性
  在Index创建时可以进行分片数量和复制数量的设置，默认创建每个

Index设置5个shard和1个Replica，表示该Index由5个逻辑存储单元进行存储，每个逻辑存储单元具有一个复制节点进行备灾，注意，shard只能在创建Index时进行设置，shard数量与document分配到哪个shard上存储有关(通常使用hash(document _id) % shard num计算document存储在哪个shard上)
在es将主shard和replic分片在不同的Node上

安装

- elasticsearch使用java语言实现，在使用时必须安装java虚拟机（目前es1.6和1.7版本均可选择1.8版本java）
- 下载地址
- 解压到安装目录 `C:\Program Files\elasticsearch`
- 运行 `cd "C:\Program Files\elasticsearch\bin" && elasticsearch.bat`
- 安装到服务 `service install elasticsearch`
- 启动服务 `net start elasticsearch`
- 停止服务 `net stop elasticsearch`
- 测试
  访问地址: http://localhost:9200
  访问结果:

```
{
  status: 200,
  name: "Smart Alec",
  cluster_name: "elasticsearch",
  version: {
    number: "1.6.0",
    build_hash: "cdd3ac4dde4f69524ec0a14de3828cb95bbb86d0",
    build_timestamp: "2015-06-09T13:36:34Z",
    build_snapshot: false,
    lucene_version: "4.10.4"
  },
  tagline: "You Know, for Search"
}
```

接口

es对外提供标准RESTAPI接口，使用他进行集群的所有操作：

- 集群、节点、索引的状态和统计信息查看
- 管理集群、节点、索引和类型
- 执行CURD操作（创建，更新，读取，删除）和索引
- 执行高级搜索功能，比如排序，分页，筛选，聚合，js脚本执行等

格式：`curl -X<REST verb> <Node>:<Port>/<Index>/<Type>/<ID>`

使用marvel插件

- 运行 `cd "C:\Program Files\elasticsearch\bin" && plugin -i elasticsearch/marvel/latest`
- 访问地址
- marvel提供sense工具调用es的RESTAPI借口, 访问地址, 以下操作使用sense或使用linux curl命令行练习

状态查询

- 集群状态查询

  输入: `GET _cat/health?v`

  输出:

```
epoch      timestamp cluster       status node.total node.data shards
1 pri relo init unassign pending_tasks
2 1442227489 18:44:49  elasticsearch yellow          1         1     50
  50    0    0       50              0
```

说明:
status:表示集群的健康状态，值可能为green,yellow,red, green表示主shard和replica(至少一个)正常，yellow表示主shard正常但replica都不正常，red表示有的主shard和replica都有问题
node.total:表示集群中节点的数量

- 节点状态查询

  输入: `GET /_cat/nodes?v`

  输出:

```
host      ip             heap.percent ram.percent load node.role
1 master name
```

```
2  silence   192.168.1.111            30          51      d          *
   Thunderbird
```

查询所有索引

输入: `GET /_cat/indices?v`

输出:

```
1  health status index               pri rep docs.count docs.deleted
   store.size pri.store.size
2  yellow open   .marvel-2015.09.02   1   1      93564            0
   78.4mb         78.4mb
3  yellow open   .marvel-2015.09.01   1   1      39581            0
   45.9mb         45.9mb
```

创建索引

输入: `PUT /test1?pretty`

输出:

```
1  {
2    "acknowledged" : true
3  }
```

查询所有索引:

```
   health status index               pri rep docs.count docs.deleted
1  store.size pri.store.size
2  yellow open   test1                 5   1          0            0
   575b           575b
```

说明:
health:由于只运行一个节点，replica不能与主shard在同一node中，因此replica不正常，该index的状态为yellow
index:为索引名称
pri:表示主shard个数
rep:表示每个shard的复制个数

docs.count:表示index中document的个数

索引、读取、删除文档

索引文档

- 方法1:
  输入:

```
PUT /test1/user/1?pretty
{"name": "silence1"}
```

输出:

```
{
  "_index" : "test1
  "_type" : "user",
  "_id" : "1",
  "_version" : 1,
  "created" : true
}
```

- 方法2:
  输入:

```
POST /test1/user/2?pretty
{"name": "silence2"}
```

输出:

```
{
  "_index" : "test1",
  "_type" : "user",
  "_id" : "2",
  "_version" : 1,
  "created" : true
}
```

- 方法3:
  输入:

```
POST /test1/user?pretty
{"name": "silence3"}
```

输出:

```
{
  "_index" : "test1",
  "_type" : "user",
  "_id" : "AU_MdQoXRYiHSIs7UGBQ",
  "_version" : 1,
  "created" : true
}
```

说明: 在索引文档时若需要指定文档ID值则需要使用PUT或者POST提交数据并显示指定ID值，若需要由es自动生成ID，则需要使用POST提交数据

读取文档:
输入: `GET /test1/user/1?pretty`
输出:

```
{
  "_index" : "test1",
  "_type" : "user",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source":{"name": "silence1"}
}
```

说明:
_index,_type:表示文档存储的Index和Type信息
_id:表示文档的编号
_version:表示文档的版本号，主要用于并发处理时使用乐观锁防止脏数据
found:表示请求的文档是否存在
_souce:格式为json，为文档的内容

注意:在之前我们并未创建user的Type，在进行文档索引时自动创建了user，在es中可以不显示的创建Index和Type而使用默认参数或者根据提交数据自定义，但不建议这么使用，在不清楚可能导致什么情况时显示创

建Index和Type并设置参数

删除文档:

输入: `DELETE /test1/user/1?pretty`

输出:

```
{
  "found" : true,
  "_index" : "test1",
  "_type" : "user",
  "_id" : "1",
  "_version" : 2
}
```

再次读取文档输出:

```
{
  "_index" : "test1",
  "_type" : "user",
  "_id" : "1",
  "found" : false
}
```

删除索引

输入: `DELETE /test1?pretty`

输出:

```
{
  "acknowledged" : true
}
```

修改文档

初始化文档输入:

```
PUT /test1/user/1?pretty
{"name" : "silence2", "age":28}
```

修改文档输入:

```
PUT /test1/user/1?pretty
{"name" : "silence1"}
```

读取文档输出:

```
{
  "_index" : "test1",
  "_type" : "user",
  "_id" : "1",
  "_version" : 2,
  "found" : true,
  "_source":{"name" : "silence1"}
}
```

更新文档

更新数据输入:

```
POST /test1/user/1/_update?pretty
{"doc" : {"name" : "silence3", "age":28}}
```

读取数据输出:

```
{
  "_index" : "test1",
  "_type" : "user",
  "_id" : "1",
  "_version" : 3,
  "found" : true,
  "_source":{"name":"silence3","age":28}
}
```

更新文档输入:

```
POST /test1/user/1/_update?pretty
{"script" : "ctx._source.age += 1"}
```

读取文档输出:

```
{
  "_index" : "test1",
  "_type" : "user",
  "_id" : "1",
  "_version" : 4,
  "found" : true,
  "_source":{"name":"silence3","age":29}
}
```

说明：需要POST使用script则必须在elasticsearch/config/elasticsearch.yml配置`script.groovy.sandbox.enabled: true`

修改(PUT)和更新(POST+_update)的区别在于修改使用提交的文档覆盖es中的文档，更新使用提交的参数值覆盖es中文档对应的参数值

根据查询删除文档

输入:

```
DELETE /test1/user/_query?pretty
{"query" : {"match" : {"name" : "silence3"}}}
```

输出:

```
{
  "_indices" : {
    "test1" : {
      "_shards" : {
        "total" : 5,
        "successful" : 5,
        "failed" : 0
      }
    }
  }
}
```

获取文档数量

输入: `GET /test1/user/_count?pretty`

输出:

```
{
  "count" : 0,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  }
}
```

批量操作

输入:

```
POST /test1/user/_bulk?pretty
{"index" : {"_id" : 1}}
{"name" : "silence1"}
{"index" : {"_id" : 2}}
{"name" : "silence2"}
{"index" : {}}
{"name" : "silence3"}
{"index" : {}}
{"name" : "silence4"}
```

输入:

```
POST /test1/user/_bulk?pretty
{"update" : {"_id" : 1}}
{"doc" : {"age" : 28}}
{"delete" : {"_id" : 2}}
```

通过文件导入数据: `curl -XPOST "localhost:9200/test1/account/_bulk?pretty" --data-binary @accounts.json`

Query查询

查询可以通过两种方式进行，一种为使用查询字符串进行提交参数查询，一种为使用RESTAPI提交requesbody提交参数查询

获取所有文档输入：GET /test1/user/_search?q=*&pretty

```
POST /test1/user/_search?pretty
{
  "query" : {"match_all" : {}}
}
```

输出：

```
{
   "took": 2,
   "timed_out": false,
   "_shards": {
      "total": 5,
      "successful": 5,
      "failed": 0
   },
   "hits": {
      "total": 3,
      "max_score": 1,
      "hits": [
         {
            "_index": "test1",
            "_type": "user",
            "_id": "1",
            "_score": 1,
            "_source": {
               "name": "silence1",
               "age": 28
            }
         },
         {
            "_index": "test1",
            "_type": "user",
            "_id": "AU_M2zgwLNdQvgqQS3MP",
            "_score": 1,
            "_source": {
               "name": "silence3"
            }
         },
         {
            "_index": "test1",
            "_type": "user",
            "_id": "AU_M2zgwLNdQvgqQS3MQ",
            "_score": 1,
            "_source": {
               "name": "silence4"
            }
         }
```

```
        ]
    }
}
```

说明:
took: 执行查询的时间(单位为毫秒)
timed_out: 执行不能超时
_shards: 提示有多少shard参与查询以及查询成功和失败shard数量
hits: 查询结果
hits.total: 文档总数
_score, max_score: 为文档与查询条件匹配度和最大匹配度

Query SDL

输入:

```
POST /test1/account/_search?pretty
{
  "query" : {"match_all":{}},
  "size": 2,
  "from" : 6,
  "sort" : {
    "age" : {"order" : "asc"}
  }
}
```

说明:
query: 用于定义查询条件过滤
match_all: 表示查询所有文档
size: 表示查询返回文档数量，若未设置默认为10
from: 表示开始位置, es使用0作为开始索引，常与size组合进行分页查询，若未设置默认为0
sort: 用于设置排序属性和规则

- 使用_source设置查询结果返回的文档属性
  输入:

```
POST /test1/account/_search?pretty
{
  "query": {
```

```
    "match_all": {}
  },
  "_source":["firstname", "lastname", "age"]
}
```

输出:

```
{
   "took": 5,
   "timed_out": false,
   "_shards": {
      "total": 5,
      "successful": 5,
      "failed": 0
   },
   "hits": {
      "total": 1000,
      "max_score": 1,
      "hits": [
         {
            "_index": "test1",
            "_type": "account",
            "_id": "4",
            "_score": 1,
            "_source": {
               "firstname": "Rodriquez",
               "age": 31,
               "lastname": "Flores"
            }
         },
         {
            "_index": "test1",
            "_type": "account",
            "_id": "9",
            "_score": 1,
            "_source": {
               "firstname": "Opal",
               "age": 39,
               "lastname": "Meadows"
            }
         }
      ]
   }
}
```

- 使用match设置查询匹配值
  输入:

```
POST /test1/account/_search?pretty
```

```
{
  "query": {
    "match": {"address" : "986 Wyckoff Avenue"}
  },
  "size" : 2
}
```

输出:


```
{
   "took": 1,
   "timed_out": false,
   "_shards": {
      "total": 5,
      "successful": 5,
      "failed": 0
   },
   "hits": {
      "total": 216,
      "max_score": 4.1231737,
      "hits": [
         {
            "_index": "test1",
            "_type": "account",
            "_id": "4",
            "_score": 4.1231737,
            "_source": {
               "account_number": 4,
               "balance": 27658,
               "firstname": "Rodriquez",
               "lastname": "Flores",
               "age": 31,
               "gender": "F",
               "address": "986 Wyckoff Avenue",
               "employer": "Tourmania",
               "email": "rodriquezflores@tourmania.com",
               "city": "Eastvale",
               "state": "HI"
            }
         },
         {
            "_index": "test1",
            "_type": "account",
            "_id": "34",
            "_score": 0.59278774,
            "_source": {
               "account_number": 34,
               "balance": 35379,
               "firstname": "Ellison",
               "lastname": "Kim",
               "age": 30,
```

```
                "gender": "F",
                "address": "986 Revere Place",
                "employer": "Signity",
                "email": "ellisonkim@signity.com",
                "city": "Sehili",
                "state": "IL"
            }
        }
    ]
  }
}
```

说明：根据查询结果可见在查询结果中并非只查询address包含”986 Wyckoff Avenue”的文档，而是包含986,wychoff,Avenue三个词中任意一个，这就是es分词的强大之处
可见查询结果中_score(与查询条件匹配度)按从大到小的顺序排列
此时你可能想要值查询address包含”986 Wyckoff Avenue”的文档，怎么办呢？使用match_phrase
输入:

```
POST /test1/account/_search?pretty
{
  "query": {
    "match_phrase": {"address" : "986 Wyckoff Avenue"}
  }
}
```

可能你已经注意到, 以上query中只有一个条件，若存在多个条件，我们必须使用bool query将多个条件进行组合
输入:

```
POST /test1/account/_search?pretty
{
  "query": {
    "bool" : {
      "must":[
        {"match_phrase": {"address" : "986 Wyckoff Avenue"}},
        {"match" : {"age" : 31}}
      ]
    }
  }
}
```

说明: 查询所有条件都满足的结果

输入:

```
POST /test1/account/_search
{
  "query": {
    "bool" : {
      "should":[
        {"match_phrase": {"address" : "986 Wyckoff Avenue"}},
        {"match_phrase": {"address" : "963 Neptune Avenue"}}
      ]
    }
  }
}
```

说明: 查询有一个条件满足的结果

输入:

```
POST /test1/account/_search
{
  "query": {
    "bool" : {
      "must_not":[
        {"match": {"city" : "Eastvale"}},
        {"match": {"city" : "Olney"}}
      ]
    }
  }
}
```

说明: 查询有条件都不满足的结果

在Query SDL中可以将must, must_not和should组合使用

输入:

```
POST /test1/account/_search
{
  "query": {
    "bool" : {
      "must": [{
        "match" : {"age":20}
      }],
      "must_not":[
```

```
9           {"match": {"city" : "Steinhatchee"}}
10        ]
11      }
12    }
13  }
```

Filters 查询

在使用Query 查询时可以看到在查询结果中都有_score值, _score值需要进行计算, 在某些情况下我们并不需要_socre值，在es中提供了Filters查询，它类似于Query查询，但是效率较高，原因:

1. 不需要对查询结果进行_score值的计算
2. Filters可以被缓存在内存中，可被重复搜索从而提高查询效率

- range 过滤器， 用于设置条件在某个范围内
  输入:

```
POST /test1/account/_search?pretty
{
  "query": {
    "filtered":{
      "query": {
        "match_all" : {}
      },
      "filter": {
        "range" : {
          "age" : {
            "gte" : 20,
            "lt" : 28
          }
        }
      }
    }
  }
}
```

判断使用filter还是使用query的最简单方法就是是否关注_score值，若关注则使用query，若不关注则使用filter

聚合分析

es提供Aggregations支持分组和聚合查询，类似于关系型数据库中的

GROUP BY和聚合函数，在ES调用聚合RESTAPI时返回结果包含文档查询结果和聚合结果，也可以返回多个聚合结果，从而简化API调用和减少网络流量使用

输入:

```
POST /test1/account/_search?pretty
{
  "size" : 0,
  "aggs" : {
    "group_by_gender" : {
      "terms" : {"field":"gender"}
    }
  }
}
```

输出:

```
{
   "took": 1,
   "timed_out": false,
   "_shards": {
      "total": 5,
      "successful": 5,
      "failed": 0
   },
   "hits": {
      "total": 1000,
      "max_score": 0,
      "hits": []
   },
   "aggregations": {
      "group_by_gender": {
         "doc_count_error_upper_bound": 0,
         "sum_other_doc_count": 0,
         "buckets": [
            {
               "key": "m",
               "doc_count": 507
            },
            {
               "key": "f",
               "doc_count": 493
            }
         ]
      }
   }
}
```

说明:
size: 返回文档查询结果数量
aggs: 用于设置聚合分类
terms: 设置group by属性值

输入:

```
POST /test1/account/_search?pretty
{
  "size" : 0,
  "aggs" : {
    "group_by_gender" : {
      "terms" : {
        "field":"state",
        "order" : {"avg_age":"desc"},
        "size" : 3
      },
      "aggs" : {
        "avg_age" : {
          "avg" : {"field" : "age"}
        },
        "max_age" : {
          "max" : {"field": "age"}
        },
        "min_age" : {
          "min": {"field":"age"}
        }
      }
    }
  }
}
```

输出:

```
{
   "took": 9,
   "timed_out": false,
   "_shards": {
      "total": 5,
      "successful": 5,
      "failed": 0
   },
   "hits": {
      "total": 1000,
      "max_score": 0,
      "hits": []
   },
```

```
  "aggregations": {
     "group_by_gender": {
        "doc_count_error_upper_bound": -1,
        "sum_other_doc_count": 992,
        "buckets": [
           {
              "key": "de",
              "doc_count": 1,
              "max_age": {
                 "value": 37
              },
              "avg_age": {
                 "value": 37
              },
              "min_age": {
                 "value": 37
              }
           },
           {
              "key": "il",
              "doc_count": 3,
              "max_age": {
                 "value": 39
              },
              "avg_age": {
                 "value": 36.333333333333336
              },
              "min_age": {
                 "value": 32
              }
           },
           {
              "key": "in",
              "doc_count": 4,
              "max_age": {
                 "value": 39
              },
              "avg_age": {
                 "value": 36
              },
              "min_age": {
                 "value": 34
              }
           }
        ]
     }
  }
}
```


说明:根据state进行分类，并查询每种分类所有人员的最大，最小，平均年龄, 查询结果按平均年龄排序并返回前3个查询结果

若需要按照分类总数进行排序时可以使用_count做为sort的field值
在聚合查询时通过size设置返回的TOP数量，默认为10

在聚合查询中可任意嵌套聚合语句进行查询
输入:

```
POST /test1/account/_search?pretty
{
  "size" : 0,
  "aggs" : {
    "group_by_age" : {
      "range" : {
        "field": "age",
        "ranges" : [{
          "from" : 20,
          "to" : 30
        }, {
          "from": 30,
          "to" : 40
        },{
          "from": 40,
          "to": 50
        }]
      },
      "aggs":{
        "group_by_gender" : {
          "terms" : {"field": "gender"},
          "aggs" : {
            "group_by_balance" :{
              "range" : {
                "field":"balance",
                "ranges" : [{
                  "to" : 5000
                }, {
                  "from" : 5000
                }
                ]
              }
            }
          }
        }
      }
    }
  }
}
```

输出:

```
{
   "took": 1,
   "timed_out": false,
   "_shards": {
      "total": 5,
      "successful": 5,
      "failed": 0
   },
   "hits": {
      "total": 1000,
      "max_score": 0,
      "hits": []
   },
   "aggregations": {
      "group_by_age": {
         "buckets": [
            {
               "key": "20.0-30.0",
               "from": 20,
               "from_as_string": "20.0",
               "to": 30,
               "to_as_string": "30.0",
               "doc_count": 451,
               "group_by_gender": {
                  "doc_count_error_upper_bound": 0,
                  "sum_other_doc_count": 0,
                  "buckets": [
                     {
                        "key": "m",
                        "doc_count": 232,
                        "group_by_balance": {
                           "buckets": [
                              {
                                 "key": "*-5000.0",
                                 "to": 5000,
                                 "to_as_string": "5000.0",
                                 "doc_count": 9
                              },
                              {
                                 "key": "5000.0-*",
                                 "from": 5000,
                                 "from_as_string": "5000.0",
                                 "doc_count": 223
                              }
                           ]
                        }
                     },
                     {
                        "key": "f",
                        "doc_count": 219,
                        "group_by_balance": {
                           "buckets": [
                              {
```

```
                                "key": "*-5000.0",
                                "to": 5000,
                                "to_as_string": "5000.0",
                                "doc_count": 20
                            },
                            {
                                "key": "5000.0-*",
                                "from": 5000,
                                "from_as_string": "5000.0",
                                "doc_count": 199
                            }
                        ]
                    }
                }
            ]
        }
    },
    {
        "key": "30.0-40.0",
        "from": 30,
        "from_as_string": "30.0",
        "to": 40,
        "to_as_string": "40.0",
        "doc_count": 504,
        "group_by_gender": {
            "doc_count_error_upper_bound": 0,
            "sum_other_doc_count": 0,
            "buckets": [
                {
                    "key": "f",
                    "doc_count": 253,
                    "group_by_balance": {
                        "buckets": [
                            {
                                "key": "*-5000.0",
                                "to": 5000,
                                "to_as_string": "5000.0",
                                "doc_count": 26
                            },
                            {
                                "key": "5000.0-*",
                                "from": 5000,
                                "from_as_string": "5000.0",
                                "doc_count": 227
                            }
                        ]
                    }
                },
                {
                    "key": "m",
                    "doc_count": 251,
                    "group_by_balance": {
                        "buckets": [
                            {
```

```
                                    "key": "*-5000.0",
                                    "to": 5000,
                                    "to_as_string": "5000.0",
                                    "doc_count": 21
                                },
                                {
                                    "key": "5000.0-*",
                                    "from": 5000,
                                    "from_as_string": "5000.0",
                                    "doc_count": 230
                                }
                            ]
                        }
                    }
                ]
            }
        },
        {
            "key": "40.0-50.0",
            "from": 40,
            "from_as_string": "40.0",
            "to": 50,
            "to_as_string": "50.0",
            "doc_count": 45,
            "group_by_gender": {
                "doc_count_error_upper_bound": 0,
                "sum_other_doc_count": 0,
                "buckets": [
                    {
                        "key": "m",
                        "doc_count": 24,
                        "group_by_balance": {
                            "buckets": [
                                {
                                    "key": "*-5000.0",
                                    "to": 5000,
                                    "to_as_string": "5000.0",
                                    "doc_count": 3
                                },
                                {
                                    "key": "5000.0-*",
                                    "from": 5000,
                                    "from_as_string": "5000.0",
                                    "doc_count": 21
                                }
                            ]
                        }
                    },
                    {
                        "key": "f",
                        "doc_count": 21,
                        "group_by_balance": {
                            "buckets": [
                                {
```

```
                                "key": "*-5000.0",
                                "to": 5000,
                                "to_as_string": "5000.0",
                                "doc_count": 0
                            },
                            {
                                "key": "5000.0-*",
                                "from": 5000,
                                "from_as_string": "5000.0",
                                "doc_count": 21
                            }
                        ]
                    }
                }
            ]
        }
    }
}
```


使用head插件

- 运行 `cd "C:\Program Files\elasticsearch\bin" && plugin -install mobz/elasticsearch-head`
- 访问地址

from: http://imsilence.github.io/2015/09/14/elasticsearch/elasticsearch_01/