Lucene（构建搜索引擎的核心类库）关键概念

1. Document： 用来索引和搜索的主要数据源，包含一个或者多个Field，而这些Field则包含我们跟Luence交互的数据。
2. Field：Document 的一个组成部分，有两个部分组成，name和value。
3. Term：不可分割的单词，搜索最小单元。
4. Token：一个Term呈现方式，包含这个Term的内容，在文档中的起始位置和类型。

ElasticSearch 核心概念(数据层面) 和关系型数据库mysql的对比

1. 关系型数据库中的数据库（database），等价于ES中的索引（**Index**）。索引的命名必须是全部小写，不能以下划线开头. ~~一个数据库下面有N张表（table）,等价于1个索引Index下面有N多类型（Type）~~
2. 一个数据库表（Table）下的数据由多行（Row）多列（Column，属性）组成，等价于1个Type由多个文档（Document）和多个（Field）组成。
3. 在一个关系型数据库里面，schema定义了表、每个表的字段，还有表和字段之间的关系。与之对应的，在ES中：Mapping定义索引下的Type的字段处理规则，即索引如何建立、索引类型、是否保存原始索引JSON文档、是否压缩原始JSON文档、是否需要分词处理、如何进行分词处理等。
4. 在数据库中的增insert、删delete、改update、查search操作对应着ES中的增Put/Post、删Delete、改Update、查Get。

ElasticSearch 核心概念(服务器层面)

1. **Cluster：集群**

- ES可以作为一个独立的单个搜索服务器。不过，为了处理大型数据集，实现容错和高可用性，ES可以运行在许多互相合作的服务器上。这些服务器的集合称为集群。

2. **Node：节点**

- 形成集群的每个服务器称为节点。

3. **Shard： 分片**

- 当有大量的文档时，由于内存的限制、磁盘处理能力不足、无法足够快的相应客户端的请求等，一个节点可能不够。这种情况下，数据可以分为较小的分片，每个分片放到不同的服务器上。
- 当你查询的索引分布在多个分片上时，ES会把查询发送给每个相关的分片，并将结果组合在一起，而应用程序并不知道分片的存在。即：这个过程对于用户来说是透明的。

4. **Replia： 副本**

- 为提高查询吞吐量或者实现高可用性，可以使用分片副本。
- 副本是一个分片的精确复制，每个分片可以有零个或者多个副本。ES中有许多相同的分片，其中之一被选择更改索引操作，这种特殊的分片成为主分片。 当主分片丢失时，如：该分片所在的数据不可用时，集群将副本提升为新的主分片。

5. **全文检索**

- 全文检索及时对一篇文章进行索引，可以根据关键字搜索，类似于mysql里的like语句。
- 全文索引就是把内容根据词的意义进行分词，然后分别创建索引，例如“你们的激情是因为什么事情来的” 可能会被分词成：“你们”，“激情”，“什么事情”，“来”等token ，这样当你搜索“你们”或者“激情”都会把这句搜出来。

ElasticSearch 数据类型

- **字符串类型** ~~string~~,text,keyword，从ElasticSearch ~~5.x开始不再支持string~~，由text和keyword类型替代。
- **整数类型** integer,long,short,byte
- **浮点类型** double,float,half_float,scaled_float，字段的长度越短，索引和搜索的效率越高。对于float、half_float和scaled_float,-0.0和+0.0是不同的值，使用term查询查找-0.0不会匹配+0.0，同样range查询中上边界是-0.0不会匹配+0.0，下边界是+0.0不会匹配-0.0。其中scaled_float，比如价格只需要精确到分，price为57.34的字段缩放因子为100，存起来就是5734 优先考虑使用带缩放因子的scaled_float浮点类型。
- **逻辑类型** boolean，Elasticsearch 6.0 之前的版本中，布尔类型的取值可以是true, false, on, off, yes, no, 0, 1等，6.0之后只接受true和false，否则会抛出异常。

- **日期类型** date，（1）日期格式的字符串，比如“2018-01-13”或“2018-01-13 12:10:30”（2）long类型的毫秒数(milliseconds-since-the-epoch，epoch就是指UNIX诞生的UTC时间1970年1月1日0时0分0秒）（3）integer的秒数(seconds-since-the-epoch)。ElasticSearch 内部会将日期数据转换为UTC，并存储为milliseconds-since-the-epoch的long型整数。
- **范围类型** range
- **二进制类型** binary，二进制字段是指用base64来表示索引中存储的二进制数据，可用来存储二进制形式的数据，例如图像。默认情况下，该类型的字段只存储不索引。二进制类型只支持index_name属性。
- **数组类型** array，在ElasticSearch中，没有专门的数组（Array）数据类型，但是，在默认情况下，任意一个字段都可以包含0或多个值，这意味着每个字段默认都是数组类型，只不过，数组类型的各个元素值的数据类型必须相同。在ElasticSearch中，数组是开箱即用的（out of box），不需要进行任何配置，就可以直接使用。（1）字符数组: [“one”, “two”] （2）整数数组: productid: [1, 2] （3）对象（文档）数组: “user”:[{ “name”: “Mary”, “age”: 12 }, { “name”: “John”, “age”: 10 }]，ElasticSearch内部把对象数组展开为{“user.name”: [“Mary”, “John”], “user.age”: [12,10]}
- **对象类型** object，JSON天生具有层级关系，文档会包含嵌套的对象。
- **嵌套类型** nested， 嵌套对象, 用于数组中的元素是对象的[{}, {}]
- **地理坐标类型** geo_point，支持经纬度存储和距离范围检索
- **地理地图** geo_shape，支持任意图形范围的检索，例如矩形和平面多边形
- **IP类型** ip，ip类型的字段用于存储IPv4或者IPv6的地址。
- **范围类型** completion
- **令牌计数类型** token_count
- **附件类型** attachment
- **抽取类型** percolator
- 空字段而不被索引: "empty_string" : "", "null_value" : null, "empty_array" : [], "array_with_null_value" : [null]

> 不同的数据类型索引的方式有稍许不同，如date何text类型不一样，date采用精确匹配，text采用全文检索方式。对于数组，es并没有专门的数组类型。任何域都可以包含0、1或者多个值，就像全文域分析得到多个词条。这暗示 数组中所有的值必须是相同数据类型的。

> 5.x后对排序，聚合这些操作用单独的数据结构(fielddata)缓存到内存里了，需要单独开启，官方解释在此fielddata。 es 5.x 以后 index这个只能用true或者false了，如果想要不被分词就把数据类型设置为keyword。在Elasticsearch6.0.0或者更新版本中创建的索引只会包含一个映射类型（mapping type）。在5.x中创建的具有多个映射类型的索引在Elasticsearch6.x中依然会正常工作。在Elasticsearch7.0.0中，映射类型将会被完全移除。

1. **text** 当一个字段是要被全文搜索的，比如Email内容、产品描述，应该使用text类型。设置text类型以后，字段内容会被分析，在生成倒排索引以前，字符串会被分析器分成一个一个词项。text类型的字段不用于排序，很少用于聚合。
2. **keyword**类型适用于索引结构化的字段，比如email地址、主机名、状态码和标签。如果字段需要进行过滤(比如查找已发布博客中status属性为published的文章)、排序、聚合。keyword类型的字段只能通过精确值搜索到

```
PUT megacorp/_mapping/employee/
{
  "properties": {
    "interests": {
      "type":     "text",
      "fielddata": true
    }
  }
}
```

> 在 ES2.x 版本字符串数据是没有 keyword 和 text 类型的，只有string类型，ES更新到5版本后，~~取消了 string 数据类型~~，代替它的是 keyword 和 text 数据类型。**Text** 数据类型被用来索引长文本，比如说电子邮件的主体部分或者一款产品的介绍。这些文本会被分析，在建立索引前会将这些文本进行分词，转化为词的组合，建立索引。允许 ES来检索这些词语。text 数据类型不能用来排序和聚合。Keyword 数据类型用来建立电子邮箱地址、姓名、邮政编码和标签等数据，不需要进行分词。可以被用来检索过滤、排序和聚合。**keyword** 类型字段只能用本身来进行检索。不通过mapping 配置索引时，遇到字符串类型时候的字端，系统会默认为“text”类型。检索的时候对字符串进行分析。所以要想只通过字段本身来进行检索，还是需要把该字段改为“keyword”类型。

```
PUT /employees
{
    "mappings":{
        "employee":{
             "properties": {
                 "intro":"text"
             }
        }
    }
}
```

```
PUT /employees
{
    "mappings":{
        "employee":{
             "properties": {
                 "name":"keyword"
             }
        }
    }
}
```

映射mappings

映射就是创建索引时指定都包含哪些字段以及字段的数据类型、分词器等一些设置。

字段设置参数

- **“type”**: “text”, // 指定字段的数据类型
- **“analyzer”**:”ik _max_word”, //指定分词器的名称,即可用使用内置的分词器也可以使用第三方分词器
  - 默认standard,
  - 内置的分析器有whitespace 、 simple和english
  - 第三方分词器：ik分词器 包括ik_max_word和ik_smart，ik_max_word：会将文本做最细粒度的拆分；尽可能多的拆分出词语 ，ik_smart：会做最粗粒度的拆分；已被分出的词语将不会再次被其它词语占有
- **“search_analyzer”**:”ik _max_word” // 指定查询的分词器,默认和analyzer保持一致，一般分词器和查询分词器要保持一致

- “**properties**”: {}, // 当数据类型是object时，需要具体指定内部对象对应的字段
- “**format**”: “yyy-MM-dd HH:mm:ss||yyyy-MM-dd||epoch_millis” // 格式化，一般用于指定日期类型的格式
- “**dynamic**”: “strict” 动态映射配置，当文档中发现新字段时应该如何处理, 可以用在根 object 或任何 object 类型的字段上，你可以将 dynamic 的默认值设置为 strict , 而只在指定的内部对象中开启它
  - true: 默认值，动态添加新的字段
  - false: 忽略新的字段
  - strict: 如果遇到新字段抛出异常,如果Elasticsearch是作为重要的数据存储，可能就会期望遇到新字段就会抛出异常，这样能及时发现问题。
- **dynamic_templates** 动态模板：为满足条件的字段做统一映射，可以通过字段名称或者字段类型来匹配指定的映射规则，每个模板都有一个名称，你可以用来描述这个模板的用途， 一个mapping来指定映射应该怎样使用，以及至少一个参数 (如 match) 来定义这个模板适用于哪个字段。 模板按照顺序来检测；第一个匹配的模板会被启用
  - match 参数只匹配字段名称
  - match_mapping_type 允许你应用模板到特定类型的字段上，就像有标准动态映射规则检测的一样
- “**fielddata**”:boolean //针对分词字段，参与排序或聚合时能提高性能，默认是false，false是不允许聚合操作的
- “**boost**”:1.23 // 权重：字段级别的分数加权，指定字段在搜索时所占的权重，所占的百分比
- “**fields**”:{“raw”:{“type”:”keyword”}} //可以对一个字段提供多种索引模式，同一个字段的值，一个分词，一个不分词，应用场景：即可以用于对字符串进行字符排序，也可以全文索引,排序时使用字段.raw来引用排序字段
- “**index**”: ~~“analyzed”~~, // 指定文本类型的字段是否分词、是否存储该字段的值（在新版本中index值为boolean类型，语意也发生了变化），旧版有三个值：
  - ~~analyzed:首先分析字符串，然后索引(存储)它。换句话说，以全文索引这个域（也就是说即分词，又存储字段的值，即可以通过全文检索的方式对该字段进行搜索）~~
  - ~~not_analyzed:索引(存储)这个域，所以它能够被搜索，但索引的是精确值。不会对它进行分析(不对字段的值进行分词，而是完整的存储该值，所以只能通过精确值才能搜索出来，即完全匹配，相当于sql中的等号=的作用)~~
  - ~~no:不索引这个域。这个域不会被搜索到(对该字段不分词，也不存储，相当于没有这个字段一样？？？)~~

- “**store**”:false//是否单独设置此字段的是否存储而从_source字段中分离，默认是false，只能搜索，不能获取值
- “**doc_values**”:false//对not_analyzed字段，默认都是开启，分词字段不能使用，对排序和聚合能提升较大性能，节约内存
- “**ignore_above**”:100 //超过100个字符的文本，将会被忽略，不被索引
- “**include_in_all**”:ture//设置是否此字段包含在_all字段中，默认是true，除非index设置成no选项
- “**index_options**”:”docs”//4个可选参数docs（索引文档号） ,freqs（文档号+词频），positions（文档号+词频+位置，通常用来距离查询），offsets（文档号+词频+位置+偏移量，通常被使用在高亮字段）分词字段默认是position，其他的默认是docs
- “**norms**”:{“enable”:true,”loading”:”lazy”}//分词字段默认配置，不分词字段：默认{“enable”:false}，存储长度因子和索引时boost，建议对需要参与评分字段使用 ，会额外增加内存消耗量
- “**null_value**”:”NULL”//设置一些缺失字段的初始化值，只有string可以使用，分词字段的null值也会被分词
- “**position_increament_gap**”:0//影响距离查询或近似查询，可以设置在多值字段的数据上火分词字段上，查询时可指定slop间隔，默认值是100
- “**similarity**”:”BM25”//默认是TF/IDF算法，指定一个字段评分策略，仅仅对字符串型和分词类型有效
- “**term_vector**”:”no”//默认不存储向量信息，支持参数yes（term存储），with_positions（term+位置）,with_offsets（term+偏移量），with_positions_offsets(term+位置+偏移量) 对快速高亮fast vector highlighter能提升性能，但开启又会加大索引体积，不适合大数据量用

```
// 默认standard，内置的分析器有whitespace 、 simple 和 english
// ik支持两种分词器：ik_max_word，ik_smart
// 分析器：用于测试分词的数据
GET /_analyze
{
  "analyzer" : "ik_max_word",
  "text": "美国留给伊拉克的是个烂摊子吗"
}
```

settings设置

settings用于设置索引的分片数量、副本数量、默认的分词器等，Elasticsearch 提供了优化好的默认配置。 除非你理解这些配置的作用并且知道为什么要去修改，否则不要随意修改。

- “number_of_shards” : 5, // 每个索引的主分片数，默认值是 5 。这个配置在索引创建后不能修改。
- “number_of_replicas” : 1, // 每个主分片的副本数，默认值是 1 。对于活动的索引库，这个配置可以随时修改。
- “analysis” : { “analyzer” : { “ik” : { “tokenizer” : “ik_ max_word” } } }

```
// 创建只有 一个主分片，没有副本的小索引：
PUT /my_temp_index
{
    "settings": {
        "number_of_shards" :   1,
        "number_of_replicas" : 0
    }
}
// 用update-index-settings API 动态修改副本数：
PUT /my_temp_index/_settings
{
    "number_of_replicas": 1
}
```

aliases别名

索引别名就像一个快捷方式或软连接,或者是一个指向，都是最终指的同一个东西，别名 带给我们极大的灵活性，允许我们做下面这些：

在运行的集群中可以无缝的从一个索引切换到另一个索引 给多个索引分组 (例如，last_three_months) 给索引的一个子集创建 视图 有两种方式管理别名： _alias用于单个操作， _aliases用于执行多个原子级操作。

Mapping一旦创建是不允许修改字段的数据类型的，为了防止以后有可能修改索引的情况，刚开始创建索引时最好就为该索引创建一个别名，然后在程序中直接使用别名，而不使用真实的索引名称。 如果后面需要要修改映射，可以再创建一个新的索引，然后把之前索引里的数据导入到新创建的索引里, 为新索引增加一个别名，将别名从老索引中移除，这样应用程序仍然使用的是别名，而这个别名已经指向了新的索引，这样就达到了不修改索引名而修改索引的目的。

> 别名就是索引的另一个名字，就像人的姓名和笔名一样，都是指向的同一个人，可以通过POST /_aliases 路径对别名进行add、remove操作

> 在你的应用中使用别名而不是索引名。然后你就可以在任何时候重建索引。别名的开销很小，应该广泛使用。

```
/ 0. 创建索引(age的类型为long)
PUT /school_v1
{
  "mappings": {
    "students": {
      "properties": {
        "name": { "type": "text" },
        "age": {"type": "long"}
      }
    }
  }
}

// 1. 为索引创建一个别名
PUT /school_v1/_alias/school

// 查看别名指向的索引
GET /*/_alias/school
// 查询索引对应的别名
GET /school_v1/_alias/*

// 2.创建一个新的索引，名字不能和之前的不一样，这次将age的数据类型改为short
PUT /school_v2
{
  "mappings": {
    "students": {
      "properties": {
        "name": { "type": "text" },
        "age": {"type": "short"}
      }
    }
  }
}

// 3.迁移数据：将之前的索引里的文档迁移到新的索引上
// 先将数据批量查询出来，然后批量插入
GET /school/students/_search?scroll=1m

POST /schools/students/_bulk
{"index": {"_id": 1}}
```

```
{"name": "张三","age": 27}
{"index": {"_id": 2}}
{"name": "小明","age": 28}

GET /schools/students/_search

// 4.为新索引增加别名，别名名称为老索引名称，这样系统可以直接使用老索引的名称来操作新索引
POST /_aliases
{
  "actions": [
    {
      "add": { "index": "school_v2", "alias": "school"},
      "remove": { "index": "school_v1", "alias": "school"}
    }
  ]
}

GET /school/students/_search
```

创建索引的例子

```
PUT /employees
{
    "settings":{
        "number_of_shards": 3,
        "number_of_replicas": 1
    },
    "mappings":{
        "man":{
            "properties":{
                "word_count":{
                    "type": "integer"
                },
                "author":{
                     "type": "keyword"
                },
                "title":{
                    "type": "text"
                },
                "publish_date":{
                    "type": "date",
                    "format": "yyyy-MM-dd HH:mm:ss || yyyy-MM-dd ||
epoch_millis"
                }
            }
        }
    }
}
```

ElasticSearch的速度已经很快了，但甚至能更快。将多个请求合并成一个，避免单独处理每个请求话费的网络延时和开销。如果你需要从ElasticSearch中检索很多文档，那么使用multi_get或者mgetAPI来将这些检索请求放在一个请求中，将比逐个文档请求更快的检索到全部文档。

mgetAPI要求有一个docs数组作为参数，每个元素包含检索文档的元数据，包括_index，_type，_id。如果你想检索一个或多个特定的字段，那么你可以通过_source参数来指定这些字段的名字

```
GET /_mget
{
    "docs":[
        {
            "_index":"csdn",
            "_type":"blog",
            "_id":"1"
        },
        {
            "_index":"grade3",
            "_type":"class2",
            "_id":"1",
            "_source":["name","age"]
        }
    ]
}
```

```
GET /csdn/blog/_mget
{
    "docs":[
        {
            "_id":"1"
        },
        {
            "_index":"grade3",
            "_type":"class2",
            "_id":"1",
            "_source":["name","age"]
        }
    ]
}
```

```
GET /csdn/blog/_mget
{
    "ids":["7","8"]
}
```

批量操作-bulk

与mgetAPI可以一次性取回多个文档的方式相同，bulk允许在一个步骤进行多次create、index、update和delete请求。如果你需要索引一个数据量，比如日志事件，他可以排队和索引数百或数千批次。bulk基本格式如下：

```
{action:{metadata}}\n
{request body}\n
{action:{metadata}}\n
{request body}\n
...
```

例子

```
POST /library/books/_bulk
{"index":{"_id":"10"}}
{"title":"ten title10","price":"1.10"}
{"index":{"_id":"11"}}
{"title":"ten title11","price":"1.11"}
{"index":{"_id":"12"}}
{"title":"ten title12","price":"1.12"}
{"index":{"_id":"13"}}
{"title":"ten title13","price":"1.13"}
{"index":{"_id":"14"}}
{"title":"ten title14","price":"1.14"}
{"index":{"_id":"15"}}
{"title":"ten title15","price":"1.15"}
```

多索引多类型

在csdn和grade3索引下进行搜索

```
GET /csdn,grade3/_search
```

在以c开头或以g开头的索引下进行搜索

```
GET /c*,g*/_search
```

在csdn和grade3索引、blog类型和class2类型下进行搜索

```
GET /csdn,grade3/blog,class2/_search
```

在所有索引下进行搜索

```
GET /_all/_search
```