增

创建索引Index

1. 使用**PUT请求**： `curl -XPUT 'localhost:9200/customer?pretty'` 创建了一个customer的索引（**加pretty参数只是为了让响应返回的是json串**）
2. 可用的参数：
   - version
   - op_type:使用这个参数时，一般是 `op_type=create` ，并且一起创建一个指定ID号的文档，表示强制执行创建索引操作并创建该ID号文档，如果已存在该ID号，则会失败，如：

```
$ curl -XPUT 'http://localhost:9200/twitter/tweet/1?op_type=create' -d '{
"user" : "kimchy",
"post_date" : "2009-11-15T14:12:12",
"message" : "trying out Elasticsearch"
}'
```

   - routing,指定分片根据什么规则进行；
   - timeout:超时时间

简单创建文档document

1. 使用**PUT请求**

```
curl -XPUT 'localhost:9200/customer/external/1?pretty' -d '
{
  "name": "John Doe"
}'
```

2. 在customer索引的external下创建了一个内容为{"name": "John Doe"}的文档
3. es不强制创建文档前一定有一个实际存在的索引，即**ES可以自动根据请求为文档创建索引**
4. 使用**POST请求**，**不需要指定ID**，es会自动生成分配一个ID；

删

删除索引INDEX

使用**DELETE请求**， `curl -XDELETE 'localhost:9200/customer'`

删除文档

1. 使用**DELETE请求**， `curl -XDELETE 'localhost:9200/customer/external/2?pretty'`

2. 使用 `delete-by-query` 插件可以使用查询语句删除特定文档；

改

更新文档

1. ES底层不会进行数据更新，更新操作实际上就是ES先将旧数据删除，然后重新创建一个新的数据；
2. **一次只能在单个文档上更新**

查

一、用URL定查询条件：

1. 用**GET请求**，用参数作为URL后缀，具体所有参数在这 https://www.elastic.co/guide/en/elasticsearch/reference/2.1/search-uri-request.html
2. 常用：
    1. `q` ，制定查询条件，具体看https://www.elastic.co/guide/en/elasticsearch/reference/2.1/query-dsl-query-string-query.html
    2. `timeout` :超时查询时间，默认没有超时时间
    3. `from` :从第几条数据开始查，默认为0；
    4. `size` :查询多少条数据，默认为10；

二、用请求体制定查询条件

1. 同样用**GET请求**，请求体用json形式；
2. json中可以加的参数：
    1. from、size两个参数：可以用在url参数上，也可以用在请求体json中，默认0和10

```
{
    "from": 0, "size": 10,
    "query":{....}
}
```

    2. 排序
        1. 使用 `sort` 字段设置，可以设置多种排序（多种排序是怎么显示？），大致模板：

```
{
    "sort":[
        {
            //第一种写法
            "date(字段名)":{"order":"desc","其他属性":""}
        },
        {
            //第二种写法
            "age(字段名)":"desc"
        }
    ]
    "query":{
        .....
    }
}
```

2. 上面展示了两种排序设置的写法，第一种可以设置更多属性，具体看https://www.elastic.co/guide/en/elasticsearch/reference/2.1/search-request-sort.html；第二种写法就直接按值排序；
3. 注意 `sort` 这个字段是数组，无论有几种排序方式，都要用[]；

3. 控制返回哪些字段： `fields` ,取值是一个数组
4. 以上的都是json第一层说明的属性，与 `query` 属性同级，Elasticsearch Reference中Search APIS章节的Request Body Search章中，除了Post filter外，其他都是类似，与 `query` 属性同级的；

请求体中的query属性

1. `term` :精确查询，相当于=; `{"term":{"user":"admin"}}` ： 查找user字段等于admin的记录
2. `terms` :相当于 `in` ； `{"terms":{"user":["admin", "admin2"]}}` ： 查找user等于admin或admin2的记录
3. `range` :相当于between，范围查询

```
{
    "range" : {
        "age" : {
            "gte" : 10,
            "lte" : 20,
            "boost" : 2.0
        }
    }
}
```

- `gte` :大于等于(Greater-than or equal to); `gt` :大于(Greater-than);
- `lte` :小于等于(Less-than or equal to); `lt` :小于(Less-than);

`wildcard` :通配符查询

```
{
    "wildcard" : { "user" : "ki*y" }
}
```

exists :是否存在,返回文档字段中至少有一个非null的值的记录；如

```
{ "exists": {"field" : "user"} }

//匹配的,返回true
{"user":""}             --空字符串不是null
{"user":"-"}            --该字符串会导致分词器返回0个tokens，但仍不是Null，说明判null与分词器无关
{"user":["jan"]}        --数组也可以
{"user":["jas",null]}   --数组中有null，但是有一个非null的

//不匹配，返回false
{"user":null}           --说明ES中不带分号的null就是
{"user":[]}             --空数组被当做是没有值与下面第四种{"foo":"bar"}情况一样
{"user":[null]}
{"foo":"bar"}           --没有该字段也算  null
```

关于字段中null这个值，可以通过mapping定义中的 null_value 修改：
如：

```
{
    "user":{
        "type":"string",
        "null_value":"_null_"
    }
}
```

上面user字段的null就从原本的null改为_null_,即如果有记录为 { "user":null } 这时候会被判断为非Null

missing :与上面exists相反，这里是返回对应字段值为Null或者没有值的记录

- 可以添加两个属性 existence 与 null_value
- existence 默认为true，如果设置为false，则对应字段没有值记录
  如 {"foo":"bar"} 和{"user":[]} 将不会返回；
- null_value 默认为false，如果设置为true，则所有对应字段包含Null的记录都会返回，
  如 { "user": ["jane",null] } 该记录虽然**有非null的值，但会被返回！**并且不受mapping中 null_value 的影响

通配符匹配 wildcard :可以使用的通配符有两个： * （匹配多个）， ? （匹配单个）
正则表达式查询： regexp

复合查询 使用 bool

1. BoolQuery布尔值查询，分为四种： `must`，`filter`，`should`，`must_not`
    1. 对四种类型的查询语句得出的布尔值进行判断，全部子句布尔值都为true则符合整个查询条件；里面包含分数判断；
    2. `must` :必须符合条件并且会返回分数；
    3. `filter` :必须符合条件但不返回分数；
    4. `should` :其子句中包含多个条件，任一一个符合则返回true；
    5. `must_not` :必须不符合条件；
    6. 分数不影响查询的布尔值判断
    7. `bool` 中四种语句之间，取都符合 `must` 、 `filter` 的条件记录返回，如果 `should` 和以上两个条件一起出现时，则不会管should的结果，should结果只影响返回的数据的分数，但不影响返回；如：

```
{
    "query":{
        //这里，不会取符合must的条件和should条件的交集，只取符合must条件的记录
        "bool":{
            "must":{
                "term":{"user":"admin"}
            },
            "should":[
                {
                    "term":{"work":"student"}
                },
                {
                    "range":{
                        "age":{
                            "lt":30,
                            "gt":20
                        }
                    }
                }
            ]
        }
    }
}
```

    8. 如果**bool语句**放在**filter或者bool中就一个should子句**，则可以满足返回的记录中符合should中的条件了；如：

```
{
    //下面这个就实现了查询满足filter语句与should语句交集的记录
    "query": {
         "bool": {
         "filter": { // filter上下文
             "bool": {
             "should": [ // should子句
                 {
                 "match_phrase": {
                     "name": {
                     "query": "星起",
                     "boost": 30,
                     "slop": 5
                     }
                 }
                 }
             ],
             "filter": { // filter子句
                 "bool": {
                 "must": [
                     {
                     "terms": {
                         "round": ["A轮"]
                     }
                     }
                 ]
                 }
             }
             }
         }
         }
     }
}
```

```
{
     "bool" : {
         "must" : {
             "term" : { "user" : "kimchy" }
         },
         "filter": {
             "term" : { "tag" : "tech" }
         },
         "must_not" : {
             "range" : {
                 "age" : { "from" : 10, "to" : 20 }
             }
         },
         "should" : [        --是一个数组
             {
                 "term" : { "tag" : "wow" }
             },
             {
                 "term" : { "tag" : "elasticsearch" }
             }
         ]
     }
 }
```

2. 多索引查询：Indices Query

```
{
     "indices" : {
         "indices" : ["index1", "index2"],
         "query" : {
             "term" : { "tag" : "wow" }
         },
         "no_match_query" : {
             "term" : { "tag" : "kow" }
         }
     }
 }
```

indices和query是必填的，no_match_query可选；