> 主讲老师：Fox
> 有道云笔记地址：https://note.youdao.com/s/WH8kgRmM

1. 分词器概述

分词是构建倒排索引的重要一环。分词根据语言环境的不同可以分为英文分词、中文分词等；根据分词实现的不同又分为标准分词器、空格分词器、停用词分词器等。在传统的分词器不能解决特定业务场景的问题时，往往需要自定义分词器。

1.1 认识分词

对于分词操作来说，英语单词相对而言是比较容易辨认和区分的，因为单词之间都会以空格或者标点隔开，举例如下:

```
you cannot use from and size to page through more than 10,000 hits
you / cannot / use / from / and / size / to / page / through / more / than / 10,000 / hits
```

而中文在单词、句子甚至段落之间没有空格。有些词可以用几个字来表达，但是同样的字在另外的句子中可以拆解成不同的组合。例如

```

杭州市长春药店
杭州 / 市长 / 春药 / 店 (错误)
杭州市 / 长春 / 药店 (正确)
```

1.2 为什么需要分词

中文分词是自然语言处理的基础。搜索引擎之所以需要进行中文分词，主要有如下3个维度的原因：

1. **语义维度**：单字很多时候表达不了语义，而词往往能表达。分词相当于预处理，能使后面和语义有关的分析更准确：
2. **存储维度**：如果所有文章按照单字来索引，那么所需要的存储空间和搜索计算时间就要多得多：
3. **时间维度**：通过倒排索引，我们能以O(1)的时间复杂度，通过词组找到对应的文章。

以“深入浅出Elasticsearch”这一字符串的检索为例。“深”“入”“浅”“出”这些字在全体内容中可能会无数次出现，如果以这些单独的字为索引，那么就需要添加无数条记录。而以“深入”为索引，所需记录就少了一些；以“深入浅出”为索引，则少得更多；最后以“深入浅出Elasticsearch”为索引，可能就剩余寥寥几条数据。但只有剩余的这些全字符匹配的文档才是我们期望召回的结果。

注意：设计索引的Mapping阶段，要根据业务用途确定是否需要分词。如果不需要分词，则建议设置keyword类型；如果需要分词，则建议设置为text类型并指定分词器。

1.3 分词发生的阶段

写入数据阶段

分词发生在数据写入阶段，也就是数据索引化阶段，其分词逻辑取决于映射参数analyzer。例如，当使用ik_smart分词器对“昨天，小明和他的朋友们去了市中心的图书馆”进行分词后，会将这句话分成不同的词汇或词组。

```
POST _analyze
{
  "analyzer":"ik_max_word",
  "text":"昨天，小明和他的朋友们去了市中心的图书馆"
}
```

返回结果：

```
{
  "tokens": [
    {
      "token": "昨天",
      "start_offset": 0,
      "end_offset": 2,
      "type": "CN_WORD",
      "position": 0
    },
    {
      "token": "小明",
      "start_offset": 3,
      "end_offset": 5,
      "type": "CN_WORD",
      "position": 1
    },
    {
      "token": "和他",
      "start_offset": 5,
      "end_offset": 7,
      "type": "CN_WORD",
      "position": 2
    },
    {
      "token": "的",
      "start_offset": 7,
      "end_offset": 8,
      "type": "CN_CHAR",
      "position": 3
    },
    {
      "token": "朋友们",
```

```
      "start_offset": 8,
      "end_offset": 11,
      "type": "CN_WORD",
      "position": 4
    },
    {
      "token": "去了",
      "start_offset": 11,
      "end_offset": 13,
      "type": "CN_WORD",
      "position": 5
    },
    {
      "token": "市中心",
      "start_offset": 13,
      "end_offset": 16,
      "type": "CN_WORD",
      "position": 6
    },
    {
      "token": "的",
      "start_offset": 16,
      "end_offset": 17,
      "type": "CN_CHAR",
      "position": 7
    },
    {
      "token": "图书馆",
      "start_offset": 17,
      "end_offset": 20,
      "type": "CN_WORD",
      "position": 8
    }
  ]
}

```


执行检索阶段

搜索发生时期，其分词仅对搜索词产生作用。在执行“图书馆”检索时，Elasticsearch会根据倒排索引查找所有包含“图书馆”的文档。

2. 分词器的组成

文档被写入并转换为倒排索引之前，Elasticsearch对文档的操作称为分析。而分析是基于Elasticsearch内置分词器(analyzer)或者自定义分词器实现的。分词器由如下三部分组成，如下图所示：

字符过滤器Character Filter

字符过滤器(character filter)将原始文本作为字符流接收，并通过添加、删除或更改字符来转换字符流。

作用：分词之前的预处理，过滤无用字符。

字符过滤器分类如下：

1) HTML Strip Character Filter：用于删除HTML元素，如删除 < b > 标签；解码HTML实体，如将&amp转义为&。

```
PUT test_html_strip_filter
{
  "settings": {
    "analysis": {
      "char_filter": {
        "my_char_filter": {
          "type": "html_strip",  // html_strip 代表使用 HTML 标签过滤器
          "escaped_tags": [     // 当前仅保留 a 标签，escaped_tags：需要保留的 html 标签
            "a"
          ]
        }
      }
    }
  }
}
GET test_html_strip_filter/_analyze
{
  "tokenizer": "standard",
  "char_filter": ["my_char_filter"],
  "text": ["<p>I&apos;m so <a>happy</a>!</p>"]
}
```

2) Mapping Character Filter：用于替换指定的字符。

```
PUT test_html_strip_filter
{
  "settings": {
    "analysis": {
      "char_filter": {
        "my_char_filter": {
          "type": "mapping", // mapping 代表使用字符映射过滤器
          "mappings": [ // 数组中规定的字符会被等价替换为 => 指定的字符
            "滚 => *",
            "垃 => *",
            "圾 => *"
          ]
        }
      }
    }
  }
}
GET test_html_strip_filter/_analyze
{
  //"tokenizer": "standard",
  "char_filter": ["my_char_filter"],
  "text": "你就是个垃圾！滚"
}
```

3) Pattern Replace Character Filter：可以基于正则表达式替换指定的字符。

```
PUT text_pattern_replace_filter
{
  "settings": {
    "analysis": {
      "char_filter": {
        "my_char_filter": {
          "type": "pattern_replace", // pattern_replace 代表使用正则替换过滤器
          "pattern": """(\d{3})\d{4}(\d{4})""", // 正则表达式
          "replacement": "$1****$2"
        }
      }
    }
  }
}
GET text_pattern_replace_filter/_analyze
{
  "char_filter": ["my_char_filter"],
  "text": "您的手机号是18868686688"
}
```

切词器Tokenizer

若进行了字符过滤，则系统将接收过滤后的字符流；若未进行过滤，则系统接收原始字符流。在接收字符流后，系统将对其进行分词，并记录分词后的顺序或位置(position)、起始值(start_offset)以及偏移量(end_offset-start_offset)。而tokenizer负责初步进行文本分词。官方内置了很多种切词器，默认的切词器为 standard。

词项过滤器Token Filter

词项过滤器用来处理切词完成之后的词项，例如把大小写转换，删除停用词或同义词处理等。官方同样预置了很多词项过滤器，基本可以满足日常开发的需要。当然也是支持第三方也自行开发的。

```
GET _analyze
{
  "tokenizer" : "standard",
  "filter" : ["uppercase"],
  "text" : ["www.elastic.org.cn","www elastic org cn"]
}
```

停用词

在切词完成之后，会被干掉词项，即停用词。停用词可以自定义

英文停用词（**english**）：a, an, and, are, as, at, be, but, by, for, if, in, into, is, it, no, not, of, on, or, such, that, the, their, then, there, these, they, this, to, was, will, with。

中日韩停用词（**cjk**）：a, and, are, as, at, be, but, by, for, if, in, into, is, it, no, not, of, on, or, s, such, t, that, the, their, then, there, these, they, this, to, was, will, with, www。

```
GET _analyze
{
  "tokenizer": "standard",
  "filter": ["stop"],
  "text": ["What are you doing"]
}

### 自定义 filter
DELETE test_token_filter_stop
PUT test_token_filter_stop
{
  "settings": {
    "analysis": {
      "filter": {
        "my_filter": {
          "type": "stop",
          "stopwords": [
            "www"
          ],
          "ignore_case": true
        }
      }
    }
  }
}
GET test_token_filter_stop/_analyze
{
  "tokenizer": "standard",
  "filter": ["my_filter"],
  "text": ["What www WWW are you doing"]
}
```

同义词

同义词定义规则

- a, b, c => d：这种方式，a、b、c 会被 d 代替。
- a, b, c, d：这种方式下，a、b、c、d 是等价的。

```
PUT test_token_filter_synonym
{
  "settings": {
    "analysis": {
      "filter": {
        "my_synonym": {
          "type": "synonym",
          "synonyms": [ "good, nice => excellent" ] //good, nice, excellent
        }
      }
    }
  }
}
GET test_token_filter_synonym/_analyze
{
  "tokenizer": "standard",
  "filter": ["my_synonym"],
  "text": ["good"]
}
```

实践练习：自定义分词器实现对书籍作者的精确匹配

业务需求是这样的：有一个作者字段，比如Li，LeiLei；Han，MeiMei以及LeiLei Li；……现在要对其进行精确匹配。

```
POST /booksdemo/_bulk
{"index":{"_id":1}}
{"name":"Li,LeiLei;Han,MeiMei"}
{"index":{"_id":2}}
{"name": "LeiLei,Li;MeiMei,Han"}

# 查不出数据
POST /booksdemo/_search
{
  "query": {
    "match": {
      "name": "lileilei"
    }
  }
}

```

自定义分词器

```
DELETE /booksdemo
PUT /booksdemo
{
  "settings": {
    "analysis": {
      "char_filter": {
        "my_char_filter": {
          "type": "mapping",
          "mappings": [        //将“,”过滤掉
            ", => "
          ]
        }
      },
      "tokenizer": {
        "my_tokenizer": {
          "type": "pattern",
          "pattern": """\;"""     //将“;”作为自定义分词分隔符
        }
      },
      "filter": {
        "my_synonym_filter": {
          "type": "synonym",
          "expand": true,
          "synonyms": [               //添加同义词词组
            "leileili  => lileilei",
            "meimeihan => hanmeimei"
          ]
        }
      },
      "analyzer": {
        "my_analyzer": {
          "tokenizer": "my_tokenizer",
          "char_filter": [
            "my_char_filter"
          ],
          "filter": [
            "lowercase",
            "my_synonym_filter"
          ]
```

```
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "name": {
        "type": "text",
        "analyzer": "my_analyzer"
      }
    }
  }
}


```

测试自定义分词器效果

```
#借助analyzer API验证分词结果是否正确
POST booksdemo/_analyze
{
  "analyzer": "my_analyzer",
  "text": "Li,LeiLei;Han,MeiMei"
}

POST booksdemo/_analyze
{
  "analyzer": "my_analyzer",
  "text": "LeiLei,Li;MeiMei,Han"
}

POST /booksdemo/_bulk
{"index":{"_id":1}}
{"name":"Li,LeiLei;Han,MeiMei"}
{"index":{"_id":2}}
{"name": "LeiLei,Li;MeiMei,Han"}

POST /booksdemo/_search
{
  "query": {
    "match": {
      "name": "lileilei"
    }
  }
}
```

3. Ngram自定义分词实战

需求背景

当对keyword类型的字段进行高亮查询时，若值为123asd456，查询sd4，则高亮结果是 < em > 123asd456 < em > 。那么，有没有办法只对sd4高亮呢?

用一句话来概括问题：明明只想查询ID的一部分，但高亮结果是整个ID串，此时应该怎么办?

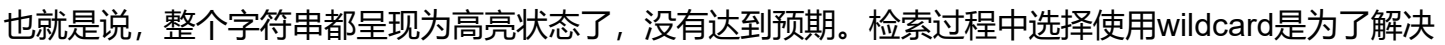
解决方案分析

实战问题拆解

```
###定义索引
PUT my_index_phone
{
  "mappings": {
    "properties": {
      "phoneNum": {
        "type": "keyword"
      }
    }
  }
}

####批量写入数据
POST my_index_phone/_bulk
{"index":{"_id":1}}
{"phoneNum":"13611112222"}
{"index":{"_id":2}}
{"phoneNum":"13944248474"}



###执行模糊检索和高亮显示
POST my_index_phone/_search
{
  "highlight": {
    "fields": {
      "phoneNum": {}
    }
  },
  "query": {
    "bool": {
      "should": [
        {
          "wildcard": {
            "phoneNum": "*1111*"
          }
        }
      ]
```

```
        }
    }
}
```

高亮检索结果如下：



也就是说，整个字符串都呈现为高亮状态了，没有达到预期。检索过程中选择使用wildcard是为了解决子串匹配的问题，wildcard的实现逻辑类似于MySQL的like模糊匹配。传统的text标准分词器，包括中文分词器ik、英文分词器english、standard等都不能解决上述子串匹配问题。
而实际业务需求是这样的：一方面要求输入子串能召回全串；另一方面要求检索的子串实现高亮。对此，只能更换一种分词来实现，即Ngram。

Ngram分词实战

Ngram分词定义

Ngram是一种基于统计语言模型的算法。Ngram基本思想是将文本里面的内容按照字节大小进行滑动窗口操作，形成长度是N的字节片段序列。此时每一个字节片段称为gram。对所有gram的出现频度进行统计，并且按照事先设定好的阈值进行过滤，形成关键gram列表，也就是这个文本的向量特征空间。列表中的每一种gram就是一个特征向量维度。该模型基于这样一种假设，第N个词的出现只与前面N-1个词相关，而与其他任何词都不相关，整句的概率就是各个词出现概率的乘积。这些概率可以通过直接从语料中统计N个词同时出现的次数得到。常用的是二元的Bi-Gram（二元语法）和三元的Tri-Gram（三元语法）。

Ngram分词示例

以“你今天吃饭了吗“这一中文句子为例，它的Bi-Gram分词结果如下：

Ngram分词应用场景

- 场景1：文本压缩、检查拼写错误、加速字符串查找、文献语种识别。
- 场景2：自然语言处理自动化领域得到新的应用。如自动分类、自动索引、超链的自动生成、文献检索、无分隔符语言文本的切分等。
- 场景3：自然语言的自动分类功能。针对Elasticsearch检索，Ngram针对无分隔符语言文本的分词（比如手机号检索），可提高检索效率（相较于wildcard检索和正则匹配检索来说）。

Ngram分词实战

```
DELETE my_index_phone
###定义索引
PUT my_index_phone
{
    "settings":{
        "number_of_shards":1,
        "number_of_replicas":0,
        "index.max_ngram_diff" : 10,
        "analysis":{
            "analyzer":{
                "phoneNo_analyzer":{
                    "tokenizer": "phoneNo_analyzer"
                }
            },
            "tokenizer":{
                "phoneNo_analyzer":{
                    "type": "ngram",
                    "min_gram": 4,
                    "max_gram": 11,
                    "token_chars": [
                        "letter","digit"
                    ]
                }
            }
        }
    },
    "mappings":{
        "dynamic":"strict",
        "properties":{
            "phoneNo":{
                "type":"text",
                "analyzer": "phoneNo_analyzer"
            }
        }
    }
}


####批量写入数据
```

```
POST my_index_phone/_bulk
{"index":{"_id":1}}
{"phoneNo":"13611112222"}
{"index":{"_id":2}}
{"phoneNo":"13944248474"}

```

如上示例共有3个核心参数。

min_gram：最小字符长度（切分），默认为1。

max_gram：最大字符长度（切分），默认为2。

token_chars：表示生成的分词结果中包含的字符类型，默认是全部类型，而在如上的示例中代表保留数字、字母。若只指定letter分词器，则数字就会被过滤掉，分词结果只剩下串中的字符。

借助analyzer API查看分词结果:

```

POST my_index_phone/_analyze
{
  "analyzer": "phoneNo_analyzer",
  "text": "13611112222"
}
```

检索及高亮的执行语句如下:

```
POST my_index_phone/_search
{
  "highlight": {
    "fields": {
      "phoneNo": {}
    }
  },
  "query": {
    "bool": {
      "should": [
        {
          "match_phrase": {
            "phoneNo": "1111"
          }
        }
      ]
    }
  }
}
```

返回结果的片段如下:

可以看出，此时代码已经能满足检索和高亮的双重需求，也就是说自定义分词完美地解决了提出的问题。