Elasticsearch教程(二)

一. Request Body深入搜索

1.1 term查询

> term是表达语义的最小单位，在搜索的时候基本都要使用到term。
>
> term查询的种类有：Term Query、Range Query等。
>
> 在ES中，Term查询不会对输入进行分词处理，将输入作为一个整体，在倒排索引中查找准确的词项。我们也可以使用 Constant Score 将查询转换为一个filter,避免算分，利用缓存，提高查询的效率。

1.1.1 term与terms

> 查询电影名字中包含有 beautiful 这个单词的所有的电影，用于查询的单词不会进行分词的处理

```
GET movies/_search
{
  "query": {
    "term": {
      "title": {
        "value": "beautiful"
      }
    }
  }
}
```

> 查询电影名字中包含有 beautiful 或者 mind 这两个单词的所有的电影，用于查询的单词不会进行分词的处理

```
GET movies/_search
{
  "query": {
    "terms": {
      "title": [
        "beautiful",
        "mind"
      ]
    }
  }
}
```

1.1.2 range

> 查询上映在2016到2018年的所有的电影，再根据上映时间的倒序进行排序

```
GET movies/_search
{
```

```
  "query": {
    "range": {
      "year": {
        "gte": 2016,
        "lte": 2018
      }
    }
  },
  "sort": [
    {
      "year": {
        "order": "desc"
      }
    }
  ]
}
```

1.1.3 Constant Score

> 查询title中包含有beautiful的所有的电影，不进行相关性算分，查询的数据进行缓存，提高效率

```
GET movies/_search
{
  "query": {
    "constant_score": {
      "filter": {
        "term": {
          "title": "beautiful"
        }
      }
    }
  }
}
```

1.2 全文查询

> 全文查询的种类有: Match Query、 Match Phrase Query、 Query String Query等
>
> 索引和搜索的时候都会进行分词，在查询的时候，会对输入进行分词，然后每个词项会逐个到底层进行查询，将最终的结果进行合并

1.2.1 match

> 查询电影名字中包含有beautiful的所有电影，每页十条，取第二页的数据

```
GET movies/_search
{
  "query": {
    "match": {
      "title": "beautiful"
    }
  },
  "from": 10,
  "size": 10
}
```

> 查询电影名字中包含有 beautiful 或者 mind 的所有的数据，但是只查询title和id两个属性

```
GET movies/_search
{
  "_source": ["title", "id"],
  "query": {
    "match": {
      "title": "beautiful mind"
    }
  }
}
```

1.2.2 match_phrase

> 查询电影名字中包含有 "beautiful mind" 这个短语的所有的数据

```
GET movies/_search
{
  "query": {
    "match_phrase": {
      "title": "beautiful mind"
    }
  }
}
```

1.2.3 multi_match

> 查询 title 或 genre 中包含有 beautiful 或者 Adventure 的所有的数据

```
GET movies/_search
{
  "query": {
    "multi_match": {
      "query": "beautiful Adventure",
      "fields": ["title", "genre"]
    }
  }
}
```

1.2.4 match_all

> 查询所有的数据

```
GET movies/_search
{
  "query": {
    "match_all": {}
  }
}
```

1.2.5 query_string

> 查询 title 中包含有 beautiful 和 mind 的所有的电影

```
GET movies/_search
{
  "query": {
    "query_string": {
      "default_field": "title",
      "query": "mind AND beautiful"
    }
  }
}
```

```
GET movies/_search
{
  "query": {
    "query_string": {
      "default_field": "title",
      "query": "mind beautiful",
      "default_operator": "AND"
    }
  }
}
```

1.2.6 simple_query_string

> simple_query_string 覆盖了很多其他查询的用法。

> 查询 title 中包含有 beautiful 和 mind 的所有的电影

```
GET movies/_search
{
  "query": {
    "simple_query_string": {
      "query": "beautiful + mind",
      "fields": ["title"]
    }
  }
}
```

```
GET movies/_search
{
  "query": {
    "simple_query_string": {
      "query": "beautiful mind",
      "fields": ["title"],
      "default_operator": "AND"
    }
  }
}
```

> 查询title中包含 "beautiful mind" 这个短语的所有的电影 (用法和match_phrase类似)

```
GET movies/_search
{
  "query": {
    "simple_query_string": {
      "query": "\"beautiful mind\"",
      "fields": ["title"]
    }
  }
}
```

查询title或genre中包含有 beautiful mind romance 这个三个单词的所有的电影 (与multi_match类似)

```
GET movies/_search
{
  "query": {
    "simple_query_string": {
      "query": "beautiful mind Romance",
      "fields": ["title", "genre"]
    }
  }
}
```

查询title中包含 “beautiful mind” 或者 "Modern Romance" 这两个短语的所有的电影

```
GET movies/_search
{
  "query": {
    "simple_query_string": {
      "query": "\"beautiful mind\" | \"Modern Romance\"",
      "fields": ["title"]
    }
  }
}
```

查询title或者genre中包含有 beautiful + mind 这个两个词，或者Comedy + Romance + Musical + Drama + Children 这个五个词的所有的数据

```
GET movies/_search
{
  "query": {
    "simple_query_string": {
      "query": "(beautiful + mind) | (Comedy + Romance + Musical + Drama + Children)",
      "fields": ["title","genre"]
    }
  }
}
```

查询 title 中包含 beautiful 和 people 但是不包含 Animals 的所有的数据

```
GET movies/_search
{
  "query": {
    "simple_query_string": {
      "query": "beautiful + people + -Animals",
      "fields": ["title"]
    }
  }
}
```

1.3 模糊搜索

> 查询title中从第6个字母开始只要最多纠正一次，就与 neverendign 匹配的所有的数据

```
GET movies/_search
{
  "query": {
    "fuzzy": {
      "title": {
        "value": "neverendign",
        "fuzziness": 1,
        "prefix_length": 5
      }
    }
  }
}
```

1.4 多条件查询

> 查询title中包含有beautiful或者mind单词，并且上映时间在2016~1018年的所有的电影

```
GET movies/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "simple_query_string": {
            "query": "beautiful mind",
            "fields": ["title"]
          }
        },
        {
          "range": {
            "year": {
              "gte": 2016,
              "lte": 2018
            }
          }
        }
      ]
    }
  }
}
```

> 查询 `title` 中包含有 `beautiful` 这个单词，并且上映年份在2016~2018年间的所有电影，但是不进行相关性的算分

```
# filter不会进行相关性的算分，并且会将查出来的结果进行缓存，效率上比 query 高
GET movies/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "title": "beautiful"
          }
        },
        {
          "range": {
            "year": {
              "gte": 2016,
              "lte": 2018
            }
          }
        }
      ]
    }
  }
}
```

二. Mapping

> mapping类似于数据库中的schema，作用如下:
>
> 1. 定义索引中的字段类型;
> 2. 定义字段的数据类型，例如：布尔、字符串、数字、日期.....
> 3. 字段倒排索引的设置

2.1 数据类型

类型名	描述
Text/Keyword	字符串，`Keyword`的意思是字符串的内容不会被分词处理，输入是什么内容，存储在ES中就是什么内容。`Text`类型ES会自动的添加一个`Keyword`类型的子字段
Date	日期类型
Integer/Float/Long	数字类型
Boolean	布尔类型

> ES中还有 "对象类型/嵌套类型"、"特殊类型（geo_point/geo_shape）"。

2.2 Mapping的定义

> 语法格式如下：

```
PUT users
{
    "mappings": {
        // define your mappings here
    }
}
```

> 定义**mapping**的建议方式：写入一个样本文档到临时索引中，**ES**会自动生成**mapping**信息，通过访问**mapping**信息的**api**查询**mapping**的定义，修改自动生成的**mapping**成为我们需要方式，创建索引，删除临时索引，简而言之就是“卸磨杀驴”。

2.3 常见参数

2.3.1 index

> 可以给属性添加一个 布尔类型的index属性，标识该属性是否能被倒排索引，也就是说是否能通过该字段进行搜索。

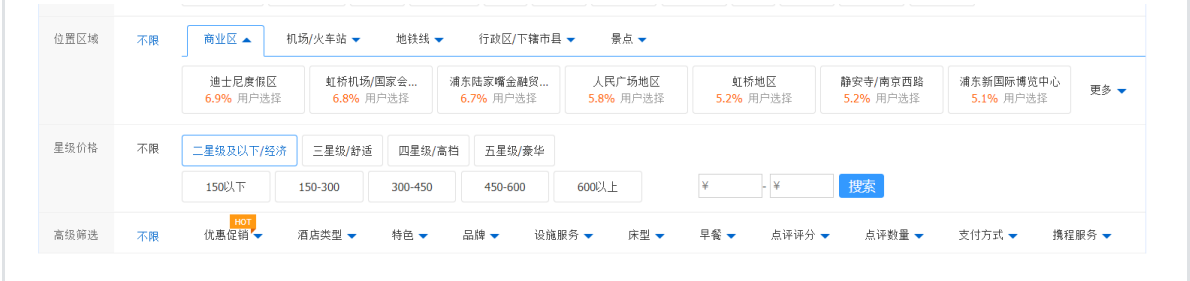
2.3.2 null_value

> 在数据索引进ES的时候，当某些数据为 `null` 的时候，该数据是不能被搜索的，可以使用 `null_value` 属性指定一个值，当属性的值为 `null` 的时候，转换为一个通过 `null_value` 指定的值。 `null_value属性只能用于Keyword类型的属性`

三. 再谈搜索

3.1 聚合查询

聚合查询的应用案例



> 聚合搜索的语法格式如下：

```
GET indexName/_search
{
    "aggs": {
        "aggs_name": {              #聚合分析的名字是由用户自定义的
            "aggs_type": {
                // aggregation body
            }
        }
    }
}
```

给users索引创建mapping信息

```
PUT employee
{
  "mappings": {
    "properties": {
      "id": {
        "type": "integer"
      },
      "name": {
        "type": "keyword"
      },
      "job": {
        "type": "keyword"
      },
      "age": {
        "type": "integer"
      },
      "gender": {
        "type": "keyword"
      }
    }
  }
}
```

往 users 索引中写入数据

```
PUT employee/_bulk
{"index": {"_id": 1}}
{"id": 1, "name": "Bob", "job": "java", "age": 21, "sal": 8000, "gender": "female"}
{"index": {"_id": 2}}
{"id": 2, "name": "Rod", "job": "html", "age": 31, "sal": 18000, "gender": "female"}
{"index": {"_id": 3}}
{"id": 3, "name": "Gaving", "job": "java", "age": 24, "sal": 12000, "gender": "male"}
{"index": {"_id": 4}}
{"id": 4, "name": "King", "job": "dba", "age": 26, "sal": 15000, "gender": "female"}
{"index": {"_id": 5}}
{"id": 5, "name": "Jonhson", "job": "dba", "age": 29, "sal": 16000, "gender": "male"}
{"index": {"_id": 6}}
{"id": 6, "name": "Douge", "job": "java", "age": 41, "sal": 20000, "gender": "female"}
{"index": {"_id": 7}}
{"id": 7, "name": "cutting", "job": "dba", "age": 27, "sal": 7000, "gender": "male"}
{"index": {"_id": 8}}
{"id": 8, "name": "Bona", "job": "html", "age": 22, "sal": 14000, "gender": "female"}
```

```
{"index": {"_id": 9}}
{"id": 9, "name": "Shyon", "job": "dba", "age": 20, "sal": 19000, "gender": "female"}
{"index": {"_id": 10}}
{"id": 10, "name": "James", "job": "html", "age": 18, "sal": 22000, "gender": "male"}
{"index": {"_id": 11}}
{"id": 11, "name": "Golsling", "job": "java", "age": 32, "sal": 23000, "gender":
"female"}
{"index": {"_id": 12}}
{"id": 12, "name": "Lily", "job": "java", "age": 24, "sal": 2000, "gender": "male"}
{"index": {"_id": 13}}
{"id": 13, "name": "Jack", "job": "html", "age": 23, "sal": 3000, "gender": "female"}
{"index": {"_id": 14}}
{"id": 14, "name": "Rose", "job": "java", "age": 36, "sal": 6000, "gender": "female"}
{"index": {"_id": 15}}
{"id": 15, "name": "Will", "job": "dba", "age": 38, "sal": 4500, "gender": "male"}
{"index": {"_id": 16}}
{"id": 16, "name": "smith", "job": "java", "age": 32, "sal": 23000, "gender": "male"}
```

3.1.1 单值的输出

> ES中大多数的数学计算只输出一个值，如：min、max、sum、avg、cardinality

> 查询工资的总和

```
GET employee/_search
{
  "size": 0,
  "aggs": {
    "other_info": {
      "sum": {
        "field": "sal"
      }
    }
  }
}
```

> 查询员工的平均工资

```
GET employee/_search
{
  "size": 0,
  "aggs": {
    "other_aggs_info": {
      "avg": {
        "field": "sal"
      }
    }
  }
}
```

> 查询总共有多少个岗位，cardinality的值类似于sql中的 count distinct,即去重统计总数

```
GET employee/_search
{
  "size": 0,
  "aggs": {
    "job_count": {
      "cardinality": {
        "field": "job"
      }
    }
  }
}
```

> 查询航班票价的最高值、平均值、最低值

```
GET kibana_sample_data_flights/_search
{
  "size": 0,
  "aggs": {
    "max_price": {
      "max": {
        "field": "AvgTicketPrice"
      }
    },
    "min_price": {
      "min": {
        "field": "AvgTicketPrice"
      }
    },
    "avg_price": {
      "avg": {
        "field": "AvgTicketPrice"
      }
    }
  }
}
```

3.1.2 多值的输出

> **ES还有些函数，可以一次性输出很多个统计的数据：terms、stats**

> 查询工资的信息

```
GET employee/_search
{
  "size": 0,
  "aggs": {
    "sal_info": {
      "stats": {
        "field": "sal"
      }
    }
  }
}
```

查询到达不同城市的航班数量

```
GET kibana_sample_data_flights/_search
{
  "size": 0,
  "aggs": {
    "flight_dest": {
      "terms": {
        "field": "DestCountry"
      }
    }
  }
}
```

查询每个岗位有多少人

```
GET employee/_search
{
  "size": 0,
  "aggs": {
    "job_count": {
      "terms": {
        "field": "job"
      }
    }
  }
}
```

查询目标地的航班次数以及天气信息

```
GET kibana_sample_data_flights/_search
{
  "size": 0,
  "aggs": {
    "dest_city": {
      "terms": {
        "field": "DestCityName"
      },
      "aggs": {
        "whether_info": {
          "terms": {
            "field": "DestWeather"
          }
        }
      }
    }
  }
}
```

查询每个岗位下工资的信息(平均工资、最高工资、最少工资等)

```
GET employee/_search
{
  "size": 0,
```

```
  "aggs": {
    "job_inf": {
      "terms": {
        "field": "job"
      },
      "aggs": {
        "sal_info": {
          "stats": {
            "field": "sal"
          }
        }
      }
    }
  }
}
```

查询不同工种的男女员工数量、然后统计不同工种下男女员工的工资信息

```
GET employee/_search
{
  "size": 0,
  "aggs": {
    "job_info": {
      "terms": {
        "field": "job"
      },
      "aggs": {
        "gender_info": {
          "terms": {
            "field": "gender"
          },
          "aggs": {
            "sal_info": {
              "stats": {
                "field": "sal"
              }
            }
          }
        }
      }
    }
  }
}
```

查询年龄最大的两位员工的信息

```
GET employee/_search
{
  "size": 0,
  "aggs": {
    "top_age_2": {
      "top_hits": {
        "size": 2,
        "sort": [
          {
            "age": {
```

```
              "order": "desc"
            }
          }
        ]
      }
    }
  }
}
```

查询不同区间员工工资的统计信息

```
GET employee/_search
{
  "size": 0,
  "aggs": {
    "sal_info": {
      "range": {
        "field": "sal",
        "ranges": [
          {
            "key": "0 <= sal <= 5000",
            "from": 0,
            "to": 5000
          },
          {
            "key": "5001 <= sal <= 10000",
            "from": 5001,
            "to": 10000
          },
          {
            "key": "10001 <= sal <= 15000",
            "from": 10001,
            "to": 15000
          }
        ]
      }
    }
  }
}
```

以直方图的方式以每5000元为一个区间查看工资信息

```
GET employee/_search
{
  "size": 0,
  "aggs": {
    "sal_info": {
      "histogram": {
        "field": "sal",
        "interval": 5000,
        "extended_bounds": {
          "min": 0,
          "max": 30000
        }
      }
    }
```

```
  }
}
```

interval: 以指定的值为一个区间。

extended_bounds: 可以指定区间的范围，如果超出了区间范围以实际为准，如果没有超出其他区间的数据依然显示。

查询平均工资大最低的工种

```
GET employee/_search
{
  "size": 0,
  "aggs": {
    "job_info": {
      "terms": {
        "field": "job"
      },
      "aggs": {
        "job_avg_sal": {
          "avg": {
            "field": "sal"
          }
        }
      }
    },
    "min_sal_job": {
      "min_bucket": {
        "buckets_path": "job_info>job_avg_sal"
      }
    }
  }
}
```

求工资和工种的信息

```
GET employee/_search
{
  "size": 0,
  "aggs": {
    "job_inf": {
      "terms": {
        "field": "job"
      }
    },
    "sal_info": {
      "stats": {
        "field": "sal"
      }
    }
  }
}
```

查询年龄大于30岁的员工的平均工资

```
GET employee/_search
{
  "size": 0,
  "query": {
    "range": {
      "age": {
        "gte": 30
      }
    }
  },
  "aggs": {
    "avg_sal": {
      "avg": {
        "field": "sal"
      }
    }
  }
}
```

查询Java员工的平均工资

```
GET employee/_search
{
  "size": 0,
  "query": {
    "constant_score": {
      "filter": {
        "term": {
          "job": "java"
        }
      },
      "boost": 1.2
    }
  },
  "aggs": {
    "avg_sal": {
      "avg": {
        "field": "sal"
      }
    }
  }
}
```

求30岁以上的员工的平均工资和所有员工的平均工资

```
GET employee/_search
{
  "size": 0,
  "aggs": {
    "older_emp": {
      "filter": {
        "range": {
          "age": {
            "gte": 30
          }
```

```
        }
      },
      "aggs": {
        "avg_sal": {
          "avg": {
            "field": "sal"
          }
        }
      }
    },
    "job_info": {
      "terms": {
        "field": "job"
      }
    }
  }
}
```

3.2 推荐搜索

> 在搜索过程中，因为单词的拼写错误，没有得到任何的结果，希望ES能够给我们一个推荐搜索。

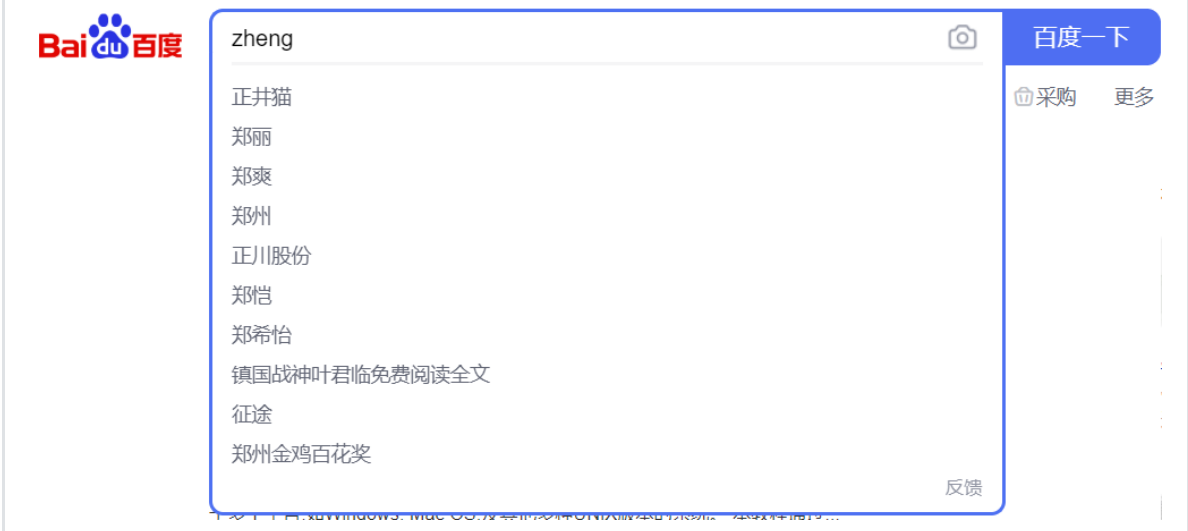
```
GET movies/_search
{
  "suggest": {
    # title_suggestion为我们自定义的名字
    "title_suggestion": {
      "text": "drema",
      "term": {
        "field": "title",
        "suggest_mode": "popular"
      }
    }
  }
}
```

> suggest_mode，有三个值：popular、missing、always
>
> 1. popular 是推荐词频更高的一些搜索。
> 2. missing 是当没有要搜索的结果的时候才推荐。
> 3. always无论什么情况下都进行推荐。

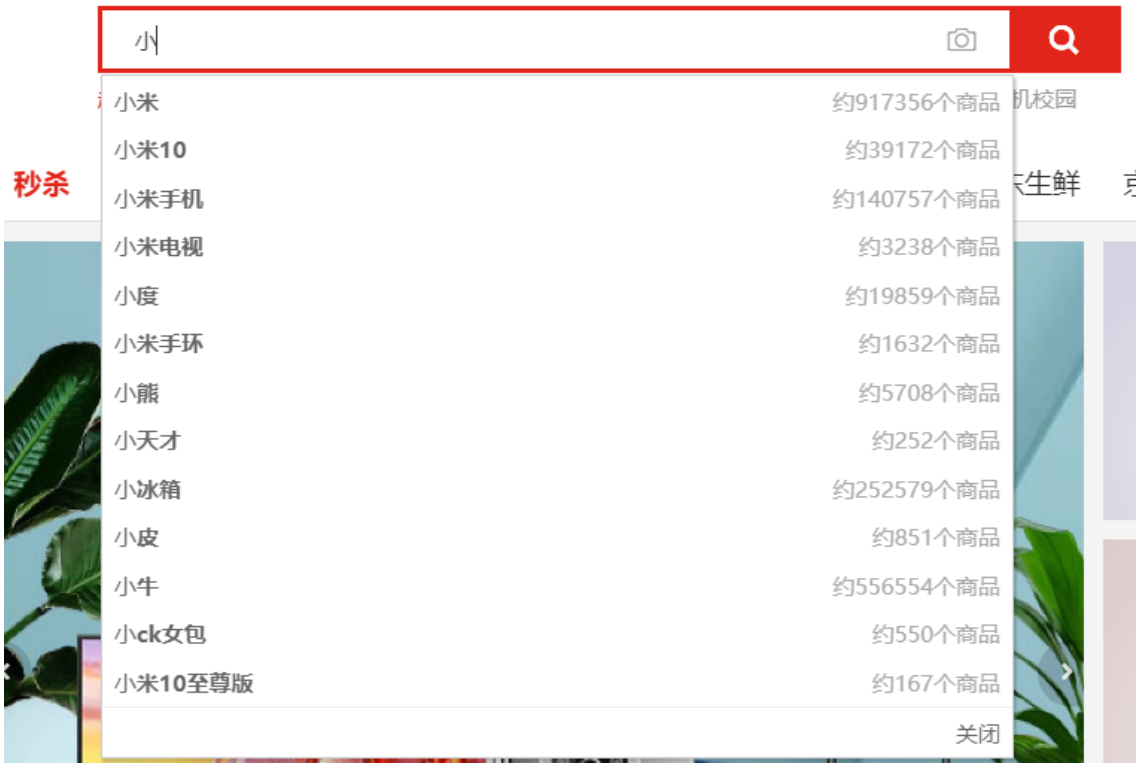
3.3 自动补全

> 自动补全应该是我们在日常的开发过程中最常见的搜索方式了，如百度搜索和京东商品搜索。

百度搜索提示



京东搜索提示



> 自动补全的功能对性能的要求极高，用户每发送输入一个字符就要发送一个请求去查找匹配项。ES采取了不同的数据结构来实现，并不是通过倒排索引来实现的；需要将对应的数据类型设置为 `completion`；所以在将数据索引进ES之前需要先定义 mapping 信息。

3.3.1 定义mapping

```
{
  "movies" : {
    "mappings" : {
      "properties" : {
        "@version" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
```

```
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "genre" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "id" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "title" : {
          # 需要自动提示的属性类型必须是 completion
          "type" : "completion"
        },
        "year" : {
          "type" : "long"
        }
      }
    }
  }
}
```

> 定义完 mapping 信息之后，导入响应的数据

3.3.2 前缀搜索

```
GET movies/_search
{
  "_source": ["title"],
  "suggest": {
    "prefix_suggestion": {
      "prefix": "Lan",
      "completion": {
        "field": "title",
        "skip_duplicates": true,
        "size": 10
      }
    }
  }
}
```

自动提示功能的实现

```
GET movies/_search
{
  "_source": ["title"],
  "suggest": {
    "prefix_suggestion": {
      "prefix": "Lan",
      "completion": {
        "field": "title",
        "skip_duplicates": true,
        "size": 10
      }
    }
  }
}
```

需要自动提示的内容

建议类型需要为completion

返回的数据

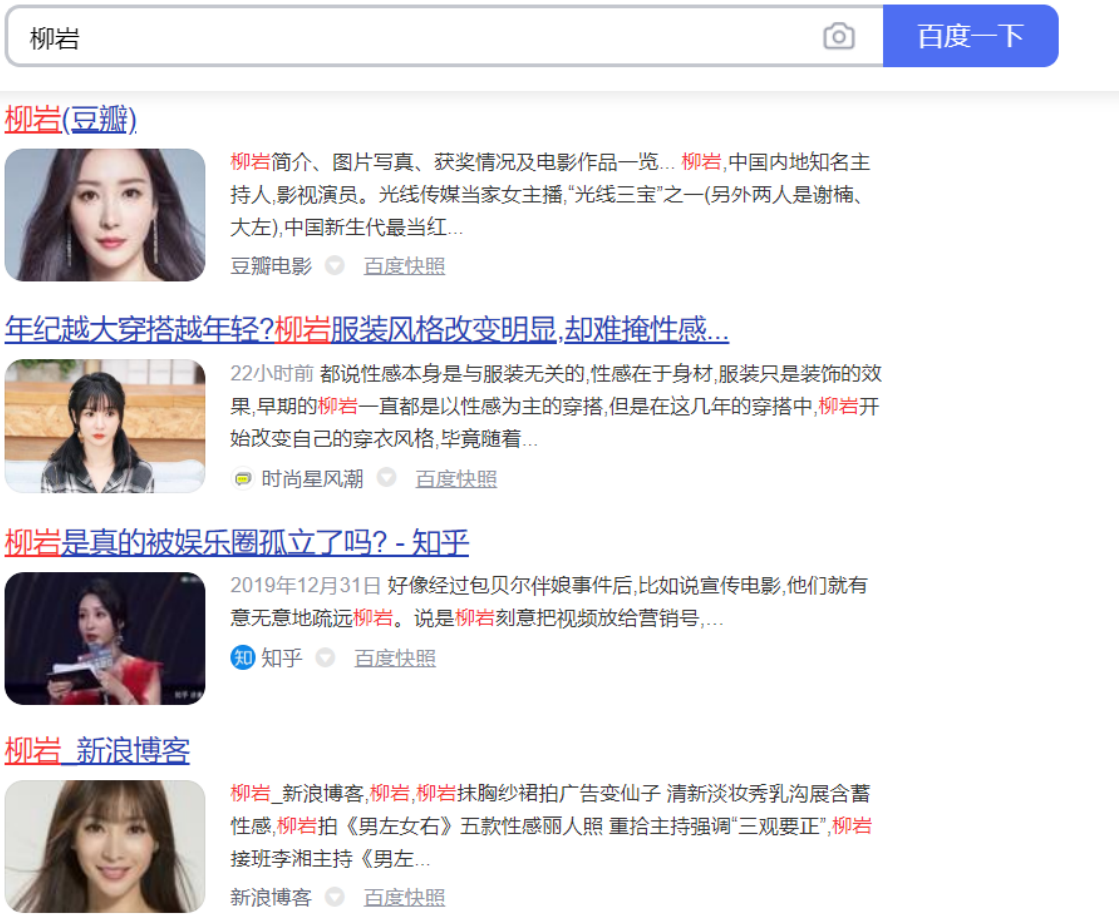
```
"suggest" : {
  "prefix_suggestion" : [
    {
      "text" : "Lan",
      "offset" : 0,
      "length" : 3,
      "options" : [
        {
          "text" : "3801 Lancaster: American Tragedy",
          "_index" : "movies",
          "_type" : "_doc",
          "_id" : "182663",
          "_score" : 1.0,
          "_source" : {
            "title" : "3801 Lancaster: American Tragedy"
          }
        },
        {
          "text" : "Lan Yu",
          "_index" : "movies",
          "_type" : "_doc",
          "_id" : "5514",
          "_score" : 1.0,
          "_source" : {
            "title" : "Lan Yu"
          }
        },
```

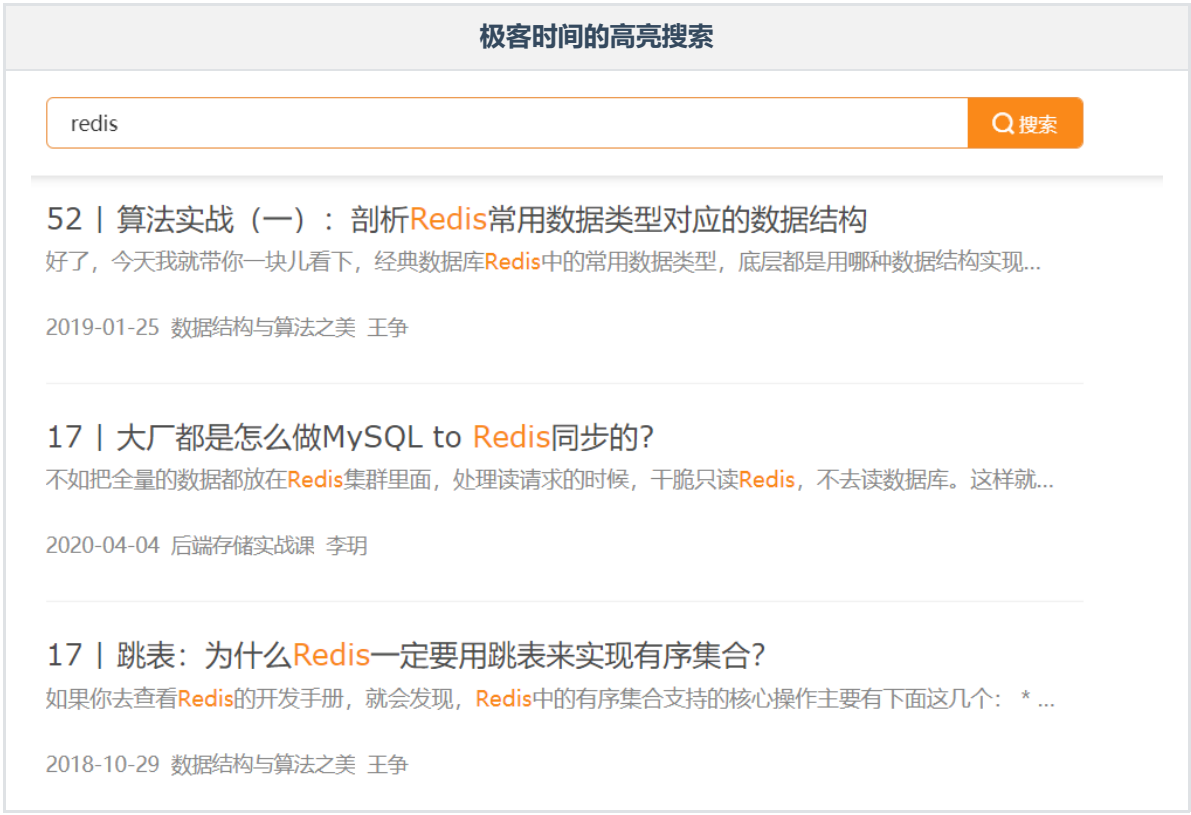
> skip_duplicates: 表示忽略掉重复。
>
> size: 表示返回多少条数据。

3.4 高亮显示

> 高亮显示在实际的应用中也会碰到很多，如下给出了百度和极客时间的两个高亮搜索的案例：

百度高亮搜索





> 将所有的包含有 beautiful 的单词高亮显示

```
GET movies/_search
{
  "query": {
    "match": {
      "title": "beautiful"
    }
  },
  "highlight": {
    "post_tags": "</span>",
    "pre_tags": "<span color='red'>",
    "fields": {
      "title": {}
    }
  }
}
```

> pre_tags: 是需要高亮文本的前置 html 内容。
>
> post_tags: 是需要高亮的后置html内容。
>
> fields: 是需要高亮的属性。

高亮结果

```
GET movies/_search
{
  "query": {
    "match": {
      "title": "beautiful"
    }
  },
  "highlight": {
    "post_tags": "</span>",
    "pre_tags": "<span color='red'>",
    "fields": {
      "title": {}
    }
  }
}
```

高亮的结果

```
"max_score" : 8.774164,
"hits" : [
  {
    "_index" : "movies",
    "_type" : "_doc",
    "_id" : "3912",
    "_score" : 8.774164,
    "_source" : {
      "title" : "Beautiful",
      "id" : "3912",
      "genre" : [
        "Comedy",
        "Drama"
      ],
      "year" : 2000,
      "@version" : "1"
    },
    "highlight" : {
      "title" : [
        "<span color='red'>Beautiful</span>"
      ]
    }
  },
```