qz_baozi
码龄6年 暂无认证
6 原创 | 35万+ 周排名 | 52万+ 总排名 | 4928 访问 | 等级
130 积分 | 1 粉丝 | 4 获赞 | 0 评论 | 6 收藏
私信 关注

搜博主文章

热门文章
将redis设置为系统服务 719
Java 文件、文件夹权限修改 453
Elasticsearch Query DSL 整理总结（一）—— Query DSL 概要，MatchAllQuery，全文查询简述 339
MongoDB 3.2版本常用代码全整理(1) - 增删改 278
MongoDB 3.2版本常用代码全整理(3) - 地理空间索引 255

分类专栏
架构 1篇
Java7并发编程学习笔记 4篇
mongodb 2篇
redis 1篇
杂项 1篇
网络 1篇

最新文章
mysql的默认隔离级别
Flutter自学旅程
新浪微博技术架构分析 2010

2020年 2篇 2019年 10篇
2018年 1篇 2017年 8篇

目录
Elasticsearch Java Rest Client API 整理...
引言
Search APIs
Search API
Search Request
可选参数
使用 SearchSourceBuilder
构建查询条件
指定排序
高亮请求
聚合请求
建议请求 Requesting Suggestions
对请求和聚合分析

Elasticsearch Java Rest Client API 整理总结 (二) —— SearchAPI

转载 qz_baozi 2019-07-04 16:20:06 87 收藏 1
分类专栏： Elasticsearch 文章标签： Elasticsearch

作者：ReyCG

出处：ReyCG 的博客 — https://www.cnblogs.com/reycg-blog/

Elasticsearch Java Rest Client API 整理总结 (二) —— SearchAPI

目录

引言

在上一篇 中主要介绍了 Document API，本节中讲解 `search API`

Search APIs

Java High Level REST Client 支持下面的 Search API：

- Search API
- Search Scroll API
- Clear Scroll API
- Multi-Search API
- Ranking Evaluation API

Search API

Search Request

`searchRequest` 用来完成和搜索文档，聚合，建议等相关的任何操作同时也提供了各种方式来完成对查询结果的高亮操作。

最基本的查询操作如下

```
SearchRequest searchRequest = new SearchRequest();
SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
searchSourceBuilder.query(QueryBuilders.matchAllQuery()); // 添加 match_all 查询
searchRequest.source(searchSourceBuilder); // 将 SearchSourceBuilder  添加到 SeachRe
```

可选参数

```
SearchRequest searchRequest = new SearchRequest("posts");  // 设置搜索的 index
searchRequest.types("doc");  // 设置搜索的 type
```

除了配置 `index` 和 `type` 外，还有一些其他的可选参数

```
searchRequest.routing("routing"); // 设置 routing 参数
searchRequest.preference("_local");  // 配置搜索时偏爱使用本地分片，默认是使用随机分片
```

什么是 routing 参数?

当索引一个文档的时候，文档会被存储在一个主分片上。在存储时一般都会有多个主分片。Elasticsearch 如何知道一个文档应该放置在哪个分片呢？这个过程是根据下面的这个公式来决定的：

```
shard = hash(routing) % number_of_primary_shards
```

- `routing` 是一个可变值，默认是文档的 `_id` ,也可以设置成一个自定义的值
- `number_of_primary_shards` 是主分片数量

所有的文档 API 都接受一个叫做 `routing` 的路由参数，通过这个参数我们可以自定义文档到分片的映射。一个自定义的路由参数可以用来确保所有相关的文档——例如所有属于同一个用户的文档——都被存储到同一个分片中。

使用 SearchSourceBuilder

对搜索行为的配置可以使用 `SearchSourceBuilder` 来完成，来看一个实例

```
SearchSourceBuilder sourceBuilder = new SearchSourceBuilder();  // 默认配置
sourceBuilder.query(QueryBuilders.termQuery("user", "kimchy")); // 设置搜索，可以是任何
sourceBuilder.from(0); // 起始 index
sourceBuilder.size(5); // 大小 size
sourceBuilder.timeout(new TimeValue(60, TimeUnit.SECONDS)); // 设置搜索的超时时间
```

设置完成后，就可以添加到 `SearchRequest` 中。

```
SearchRequest searchRequest = new SearchRequest();
searchRequest.source(sourceBuilder);
```

构建查询条件

查询请求是通过使用 `QueryBuilder` 对象来完成的，并且支持 Query DSL。

> DSL (domain-specific language) 领域特定语言，是指专注于某个应用程序领域的计算机语言。
> — 百度百科

可以使用构造函数来创建 `QueryBuilder`

```
MatchQueryBuilder matchQueryBuilder = new MatchQueryBuilder("user", "kimchy");
```

```
matchQueryBuilder.fuzziness(Fuzziness.AUTO);  // 模糊查询
matchQueryBuilder.prefixLength(3); // 前缀查询的长度
matchQueryBuilder.maxExpansions(10); // max expansion 选项，用来控制模糊查询
```

也可以使用 QueryBuilders 工具类来创建 QueryBuilder 对象。这个类提供了函数式编程风格的各种方法用来快速创建 QueryBuilder 对象。

```
QueryBuilder matchQueryBuilder = QueryBuilders.matchQuery("user", "kimchy")
                                        .fuzziness(Fuzziness.AUTO)
                                                .prefixLength(3)
                                                .maxExpansions(10);
```

> fuzzy-matching 拼写错误时的匹配：
> 好的全文检索不应该是完全相同的限定逻辑，相反，可以扩大范围来包括可能的匹配，从而根据相关性得分将更好的匹配放在前面。
> 例如，搜索 quick brown fox 时会匹配一个包含 fast brown foxes 的文档

不论什么方式创建的 QueryBuilder ，最后都需要添加到 `SearchSourceBuilder 中

```
searchSourceBuilder.query(matchQueryBuilder);
```

构建查询 文档中提供了一个丰富的查询列表，里面包含各种查询对应的 QueryBuilder 对象以及 QueryBuilder helper 方法，大家可以去参考。

关于构建查询的内容会在下篇文章中讲解，敬请期待。

指定排序

SearchSourceBuilder 允许添加一个或多个 SortBuilder 实例。这里包含 4 种特殊的实现, (Field- , Score- , GeoDistance- 和 ScriptSortBuilder)

```
sourceBuilder.sort(new ScoreSortBuilder().order(SortOrder.DESC)); // 根据分数 _score
sourceBuilder.sort(new FieldSortBuilder("_uid").order(SortOrder.ASC));  // 根据 id 升
```

过滤数据源

默认情况下，查询请求会返回文档的内容 _source ,当然我们也可以配置它。例如，禁止对 _source 的获取

```
sourceBuilder.fetchSource(false);
```

也可以使用通配符模式以更细的粒度包含或排除特定的字段：

```
String[] includeFields = new String[] {"title", "user", "innerObject.*"};
String[] excludeFields = new String[] {"_type"};
sourceBuilder.fetchSource(includeFields, excludeFields);
```

高亮请求

可以通过在 SearchSourceBuilder 上设置 HighlightBuilder 完成对结果的高亮，而且可以配置不同的字段具有不同的高亮行为。

```
SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
HighlightBuilder highlightBuilder = new HighlightBuilder();
HighlightBuilder.Field highlightTitle =
        new HighlightBuilder.Field("title"); // title 字段高亮
highlightTitle.highlighterType("unified");  // 配置高亮类型
highlightBuilder.field(highlightTitle);  // 添加到 builder
HighlightBuilder.Field highlightUser = new HighlightBuilder.Field("user");
highlightBuilder.field(highlightUser);
searchSourceBuilder.highlighter(highlightBuilder);
```

聚合请求

要实现聚合请求分两步

1. 创建合适的 `AggregationBuilder
2. 作为参数配置在 `SearchSourceBuilder 上

```
SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
TermsAggregationBuilder aggregation = AggregationBuilders.terms("by_company")
        .field("company.keyword");
aggregation.subAggregation(AggregationBuilders.avg("average_age")
        .field("age"));
searchSourceBuilder.aggregation(aggregation);
```

建议请求 Requesting Suggestions

SuggestionBuilder 实现类是由 SuggestBuilders 工厂类来创建的。

```
SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
SuggestionBuilder termSuggestionBuilder =
    SuggestBuilders.termSuggestion("user").text("kmichy");
SuggestBuilder suggestBuilder = new SuggestBuilder();
suggestBuilder.addSuggestion("suggest_user", termSuggestionBuilder);
searchSourceBuilder.suggest(suggestBuilder);
```

对请求和聚合分析

分析 API 可用来对一个特定的查询操作中的请求和聚合进行分析，此时要将 SearchSourceBuilder 的 profile 标志位设置为 true

```
SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
searchSourceBuilder.profile(true);
```

只要 SearchRequest 执行完成，对应的 SearchResponse 响应中就会包含 分析结果

同步执行

同步执行是阻塞式的，只有结果返回后才能继续执行。

```
SearchResponse searchResponse = client.search(searchRequest);
```

异步执行

异步执行使用的是 listener 对结果进行处理。

```
ActionListener<SearchResponse> listener = new ActionListener<SearchResponse>() {
    @Override
    public void onResponse(SearchResponse searchResponse) {
        // 查询成功
    }
```

```
<span class="hljs-meta">@Override</span>
<span class="hljs-function"><span class="hljs-keyword">public</span> <span
    <span class="hljs-comment">// 查询失败</span>
}
```

};

查询响应 SearchResponse

查询执行完成后，会返回 SearchResponse 对象，并在对象中包含查询执行的细节和符合条件的文档集合。

- 请求本身的信息，如 HTTP 状态码，执行时间，或者请求是否超时

```
RestStatus status = searchResponse.status(); // HTTP 状态码
TimeValue took = searchResponse.getTook(); // 查询占用的时间
Boolean terminatedEarly = searchResponse.isTerminatedEarly(); // 是否由于 SearchSourc
boolean timedOut = searchResponse.isTimedOut(); // 是否超时
```

- 查询影响的分片数量的统计信息，成功和失败的分片

```
int totalShards = searchResponse.getTotalShards();
int successfulShards = searchResponse.getSuccessfulShards();
int failedShards = searchResponse.getFailedShards();
for (ShardSearchFailure failure : searchResponse.getShardFailures()) {
    // failures should be handled here
}
```

检索 SearchHits

要访问返回的文档，首先要在响应中获取其中的 `SearchHits`

```
SearchHits hits = searchResponse.getHits();
```

`SearchHits` 中包含了所有命中的全局信息，如查询命中的数量或者最大分值：

```
long totalHits = hits.getTotalHits();
float maxScore = hits.getMaxScore();
```

查询的结果嵌套在 `SearchHits` 中，可以通过遍历循环获取

```
SearchHit[] searchHits = hits.getHits();
for (SearchHit hit : searchHits) {
    // do something with the SearchHit
}
```

`SearchHit` 提供了如 `index` ， `type` ， `docId` 和每个命中查询的分数

```
String index = hit.getIndex();
String type = hit.getType();
String id = hit.getId();
float score = hit.getScore();
```

而且，还可以获取到文档的源数据，以 JSON-String 形式或者 key-value map 对的形式。在 map 中，字段可以是普通类型，或者是列表类型，嵌套对象。

```
String sourceAsString = hit.getSourceAsString();
Map<String, Object> sourceAsMap = hit.getSourceAsMap();
String documentTitle = (String) sourceAsMap.get("title");
List<Object> users = (List<Object>) sourceAsMap.get("user");
Map<String, Object> innerObject =
        (Map<String, Object>) sourceAsMap.get("innerObject");
```

Search API 查询关系

上面的 `QueryBuilder` ， `SearchSourceBuilder` 和 `SearchRequest` 之间都是嵌套关系，为此我专门整理了一个关系图，以便更清楚的确认它们之间的关系。感兴趣的同学可用此图与前面的 API 进行对应，以加深理解。

![search API 关系图]
(https://imgconvert.csdnimg.cn/aHR0cHM6Ly9pbWFnZXMuY25ibG9ncy5jb20vY25ibG9nc19jb20vcmV5Y2ctYmxvZy8xMzQwNTczL29fcmVzdF9oaWdoX2NsaWVudF9zZWFyY2hfcmVzcG9uc2UucG5n)

结语

本篇包含了 **Java High level Rest Client** 的 SearchAPI 部分，获取高亮，聚合，分析的结果并没有在本文涉及，需要的同学可参考官方文档，下篇会包含查询构建，敬请期待~

系列文章列表

1. Elasticsearch Java Rest Client API 整理总结 (一)——Document API
2. Elasticsearch Java Rest Client API 整理总结 (二) —— SearchAPI
3. Elasticsearch Java Rest Client API 整理总结 (三)——Building Queries