1、什么是 elasticsearch？

是一款强大的开源**搜索引擎**，可以**从海量数据中快速找到需要的内容**，负责存储、搜索、分析数据，广泛用于**日志数据分析**、**实时监控**等领域。

2、正向索引和倒排索引

【补充】**每条数据**就是一个**文档**；对文档中的内容**分词后词语**是**词条**。

1）正向索引：基于**文档 id** 创建**索引**，查询词条时必须**先逐条获取每个文档**，然后**判断是否包含词条**。（是根据**文档查找词条**的过程）

【优】：可以给**多个字段创建索引**，根据**索引字段**搜索、排序都**很快**；

【缺】：查找**非索引字段**或**索引字段中部分词条**时，只能**全表扫描**。

2）倒排索引：对**文档**内容**分词**，对**词条**创建**索引**，并**记录词条所在文档的信息**；查询时先根据**词条查询文档 id**，然后根据 **id 获取文档**。（是根据**词条查找文档**的过程）

【优】：根据**词条搜索**、**模糊搜索**时，速度**非常快**；

【缺】：只能给**词条创建索引**，而**不是字段**；根据**字段无法**做**排序**。

3、ES 相关概念（面向**文档存储**，文档数据被**序列化为 json** 后存到 ES）

1）索引：类似于数据库的表，相同类型的文档的集合；

2）文档：类似于数据库的每一行数据；

3）字段：类似于数据库中的列；

4）映射：类似表的结构约束，索引中文档的字段约束信息；

5）DSL：ES 提供的 JSON 风格的请求语句，用来操作 elasticsearch。

4、mysql 与 elasticsearch

Mysql：擅长**事务类型操作**，可以确保**数据的安全和一致性**；

Elasticsearch：擅长**海量数据的搜索**、分析、计算。

因此在企业中，往往是两者结合使用：

1）对**安全性要求较高的写**操作，使用 mysql 实现；

2）对**查询性能要求较高的搜索**需求，使用 elasticsearch 实现；

3）两者再基于 MQ 等方式，实现**数据的同步，保证一致性**。

5、mapping 映射属性

mapping 是对**索引库中文档的约束**，常见的 mapping 属性包括：

1）**type**：**字段数据类型**：

- **字符串**：**text**（**可分词**的文本）、**keyword**（**精确值**，例如：品牌、国家、ip 地址）
- 数值：long、integer、short、byte、double、float、
- 布尔：boolean
- 日期：date
- 对象：object

2）**index**：是否**创建索引**，默认为 true；（即可搜索）

3）**analyzer**：使用**哪种分词器**：ik_max_word(细粒度)，ik_smart

4）**properties**：该字段的**子字段**；

【注意】某几个字段里设置“**copy_to**”：“新字段”：将当前字段拷贝制定字段，即将多字段的值**合并搜索**。

6、索引库操作

1）创建索引库：**PUT** /索引库名{" mapping " {…}}

2）查询索引库：**GET** /索引库名

3）删除索引库：**DELETE** /索引库名

4）添加字段：**PUT** /索引库名/**_mapping**{ " properties " ： {}}

【注意】**倒排索引**中，数据结构一旦改变，就需**要重新创建倒排索引**，故**索引库一旦创建**，**无法修改 mapping 中已有字段**结构，但**可以添加新字段**。

7、文档操作

1）创建文档：**POST** /索引库名/**_doc**/文档 id　{ "字段 1": "值 1",… }

2）查询文档：**GET** /索引库名/**_doc**/文档 id

3）删除文档：**DELETE** /索引库名/**_doc**/文档 id

4）修改文档：

1>**全量**修改：**PUT** /索引库名/**_doc**/文档 id { json 文档 }

2>**增量**修改：**POST** /索引库名/**_update**/文档 id { **"doc"**: { "字段": "新值", }}

8、RestClient（与 ES 的交互都封装在 RestHighLevelClient 的类）

1）引入 es 的 **RestHighLevelClient 依赖**：

```xml
<dependency>
    <groupId>org.elasticsearch.client</groupId>
    <artifactId>elasticsearch-rest-high-level-client</artifactId>
</dependency>
```

2)因为 SpringBoot 默认的 ES 版本是 7.6.2，需要**覆盖默认的 ES 版本**：

```xml
<properties>
    <java.version>1.8</java.version>
    <elasticsearch.version>7.12.1</elasticsearch.version>
</properties>
```

3)初始化 RestHighLevelClient:参数 **RestClient.builder**(HttpHoet.create())

```
RestHighLevelClient client = new RestHighLevelClient(RestClient.builder(
        HttpHost.create("http://192.168.150.101:9200")
));
```

4）结果后需要**关闭连接**：client.close();

9、创建索引库（**CreateIndex**Request）

1）**创建 Request 对象**；

2）**准备参数 request.source**(“json 数据”,XContentType.JSON)；

3）**发送请求 client.indices()**.create(request, RequestOptions.DEFAULT)。

10、删除索引库（DeleteIndexRequest）

1）创建 Request 对象；2）准备参数，无参；3）发送请求.delete()。

```
@Test
void testDeleteHotelIndex() throws IOException {
    // 1.创建Request对象
    DeleteIndexRequest request = new DeleteIndexRequest("hotel");
    // 2.发送请求
    client.indices().delete(request, RequestOptions.DEFAULT);
}
```

11、查询索引库（GetIndexRequest）

1）创建 Request 对象；2）准备参数，无参；3）发送请求.get()。

【注】若查询索引库是否存在，则用：.exists()发送请求。

12、JavaRestClient 操作 elasticsearch 的流程：

【核心】是 client.indices()方法来获取索引库的操作对象；

1）**初始化 RestHighLevelClient**：连接到主机 ；

2）**创建 XxxIndexRequest**：得到 request 对象；

3）**准备 DSL**（ Create 时需要，其它是无参）；

4）**发送请求**：调用 client.indices().xxx()方法，

13、RestClient 新增文档

数据库表结构和索引库结构存在差异时，重新定义一个索引库实体类。

将数据库中查询出来的对象→索引库实体类对象，然后序列化为json，**JSON. toJSONString(索引实体类对象)**。（然后三步走）

```
// 1.根据id查询酒店数据
Hotel hotel = hotelService.getById(61083L);
// 2.转换为文档类型
HotelDoc hotelDoc = new HotelDoc(hotel);
// 3.将HotelDoc转json
String json = JSON.toJSONString(hotelDoc);

// 1.准备Request对象
IndexRequest request = new IndexRequest("hotel").id(hotelDoc.getId().toString());
// 2.准备Json文档
request.source(json, XContentType.JSON);
// 3.发送请求
client.index(request, RequestOptions.DEFAULT);
```

1）**创建 Request 对象**：**Index**Request ("指定索引库").id(指定 id)

2）准备请求参数；（索引实体类对象的 json 格式）

3）发送请求，**client.index**(request, RequestOptions.DEFAULT);;

14、查询文档（无需请求参数，但是需要解析响应结果）

1）创建 Request 对象。**Get**Request("指定索引库", "指定文档 id")；

2）发送请求，得到结果 response。调用 **client.get**()方法；

3）解析结果，JSON.**parseObject**(json, 索引库实体类.class);**反序列化**。

```
// 1.准备Request
GetRequest request = new GetRequest("hotel", "61082");
// 2.发送请求，得到响应
GetResponse response = client.get(request, RequestOptions.DEFAULT);
// 3.解析响应结果
String json = response.getSourceAsString();

HotelDoc hotelDoc = JSON.parseObject(json, HotelDoc.class);
System.out.println(hotelDoc);
```

15、删除文档（DeleteRequest("指定索引库", "指定文档 id")；）

```
// 1.准备Request
DeleteRequest request = new DeleteRequest("hotel", "61083");
// 2.发送请求
client.delete(request, RequestOptions.DEFAULT);
```

16、修改文档 **request.doc**("price", "952", "starName", "四钻");

```
// 1.创建Request对象
UpdateRequest request = new UpdateRequest("hotel", "61083");
// 2.准备请求参数
request.doc(
    "price", "952",
    "starName", "四钻"
);
// 3.发送请求
client.update(request, RequestOptions.DEFAULT);
```

17、批量导入文档（BulkRequest）

1）利用 mybatis-plus 查询酒店数据；
2）将查询到的酒店数据（Hotel）转换为文档类型数据（HotelDoc）；
3）利用 JavaRestClient 中的 BulkRequest 批处理，实现批量新增文档。

```
// 批量查询酒店数据
List<Hotel> hotels = hotelService.list();

// 1.创建Request
BulkRequest request = new BulkRequest();
// 2.准备参数，添加多个新增的Request
for (Hotel hotel : hotels) {
    // 2.1.转换为文档类型HotelDoc
    HotelDoc hotelDoc = new HotelDoc(hotel);
    // 2.2.创建新增文档的Request对象
    request.add(new IndexRequest("hotel")
                .id(hotelDoc.getId().toString())
                .source(JSON.toJSONString(hotelDoc), XContentType.JSON));
}
// 3.发送请求
client.bulk(request, RequestOptions.DEFAULT);
```

18、Restclient 查询文档

步骤：
1>创建 SearchRequest 对象，指定索引库名；
2>利用 request.source()构建 DSL，
3>.query()表示查询文档操作，
4>内部利用 QueryBuilders 工具类构建查询条件；
5>发送请求，client.search(request, RequestOptions.DEFAULT);
6>解析响应，response.getHits()所有数据，逐条反序列化。
QueryBuilders 工具类包含各种查询：

1）match 查询
- matchAllQuery()：查询所有数据，不需要条件；
- matchQuery(“字段”,“指定内容”)：全文检索查询；
- multiMatchQuery(“指定内容”,“字段 1”,“字段 2”)：

2）精确查询
- termQuery(“字段”,“指定内容”)
- rangeQuery(“字段”).gte(>数值).lte(<数值)

3）布尔查询
- boolQuery()：用 must\must_not\filter\should 等组合其他查询方式
> must：必须匹配每个子查询，类似“与”；
> should：选择性匹配子查询，类似“或”；
> must_not：必须不匹配，不参与算分，类似“非”；
> filter：必须匹配，不参与算分；

【注意】打分的字段越多，查询的性能也越差，所以建议搜索框的字段参与算分，其他字段采用 filter 查询，不参与算分。

```
// 2.1.准备BooleanQuery
BoolQueryBuilder boolQuery = QueryBuilders.boolQuery();
// 2.2.添加term
boolQuery.must(QueryBuilders.termQuery("city", "杭州"));
// 2.3.添加range
boolQuery.filter(QueryBuilders.rangeQuery("price").lte(250));
request.source().query(boolQuery);
```

19、排序 / 分页

.from(数).size(数)和.sort(" 字段 " ， SortOrder.ASC)与.query()同级）

```
// 页码，每页大小
int page = 1, size = 5;

// 1.准备Request
SearchRequest request = new SearchRequest("hotel");
// 2.准备DSL
// 2.1.query
request.source().query(QueryBuilders.matchAllQuery());
// 2.2.排序 sort
request.source().sort("price", SortOrder.ASC);
// 2.3.分页 from、size
request.source().from((page - 1) * size).size(5);
// 3.发送请求
SearchResponse response = client.search(request, RequestOptions.DEFAULT);
// 4.解析响应
handleResponse(response);
```

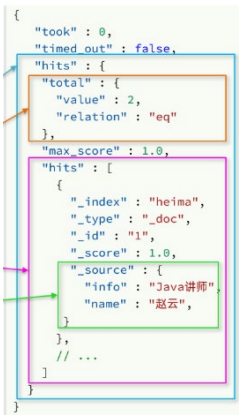
20、高亮（.highlighter()与.query()同级））

内部参数 new HighlightBuilder().field("name").requireFieldMatch(false)
【注意】1)默认情况下，高亮的字段，必须与搜索指定的字段一致，否则无法高亮。对非搜索字段高亮，则加属性：required_field_match=false。
2）高亮查询必须使用全文检索查询，并且要有搜索关键字。

```
// 1.准备Request
SearchRequest request = new SearchRequest("hotel");
// 2.准备DSL
// 2.1.query
request.source().query(QueryBuilders.matchQuery("all", "如家"));
// 2.2.高亮
request.source().highlighter(new HighlightBuilder()
                             .field("name")
                             .requireFieldMatch(false));
// 3.发送请求
SearchResponse response = client.search(request, RequestOptions.DEFAULT);
// 4.解析响应
handleResponse(response);
```

21、高亮结果解析

高亮的结果（highlight）与查询的文档结果（_source）默认是分离的，返回多条数据解析时的步骤：
1）response.getHits()解析响应，响应数据 searchHits 中有 hits 字段，里面有 total（搜索到的总条数）和多个 hits(搜索到的具体文档内容)
2）searchHits.getTotalHits().value 获取总条数；
3）SearchHit[ ] hits = searchHits.getHits();获取所有文档到数组中；
4）遍历数组所有文档进行单个处理；
5）hit.getSourceAsString();获取 source 内容；
6）JSON.parseObject()反序列化；
7）hit.getHighlightFields()获取高亮结果，键值对：“字段”：“文本”；
8）从 map 中.get(“字段”) 根据字段名获取高亮结果；
9）.getFragments()[0].string();获取高亮 text 结果（多个的话循环）
10）索引库实体类对象.setXxx()方法覆盖原非高亮结果的字段值。




```
// 4.解析响应
SearchHits searchHits = response.getHits();
// 4.1.获取总条数
long total = searchHits.getTotalHits().value;
System.out.println("共搜索到" + total + "条数据");
// 4.2.文档数组
SearchHit[] hits = searchHits.getHits();
// 4.3.遍历
for (SearchHit hit : hits) {
    // 获取文档source
    String json = hit.getSourceAsString();
    // 反序列化
    HotelDoc hotelDoc = JSON.parseObject(json, HotelDoc.class);
    // 获取高亮结果
    Map<String, HighlightField> highlightFields = hit.getHighlightFields();
    if (!CollectionUtils.isEmpty(highlightFields)) {
        // 根据字段名获取高亮结果
        HighlightField highlightField = highlightFields.get("name");
        if (highlightField != null) {
            // 获取高亮值
            String name = highlightField.getFragments()[0].string();
            // 覆盖非高亮结果
            hotelDoc.setName(name);
        }
    }
    System.out.println("hotelDoc = " + hotelDoc);
}
```

22、地理距离排序（SortBuilders 提供 3 个参数，坐标、顺序、单位）

```
// 2.3.排序
String location = params.getLocation();
if (location != null && !location.equals("")) {
    request.source().sort(SortBuilders
            .geoDistanceSort( fieldName: "location", new GeoPoint(location))
            .order(SortOrder.ASC)
            .unit(DistanceUnit.KILOMETERS)
    );
}
```

23、数据聚合

聚合是实现对数据的统计分析运算：常见三类聚合：

1）桶（Bucket）聚合：用来对文档做分组；

1>TermAggregation：按照文档字段值分组；

2> Date Histogram：按照日期阶梯分组；

2）度量（Metric）聚合：用来计算一些值，

如 Avg；Max；Min；**Stats**：同时求 max、min、avg、sum 等；

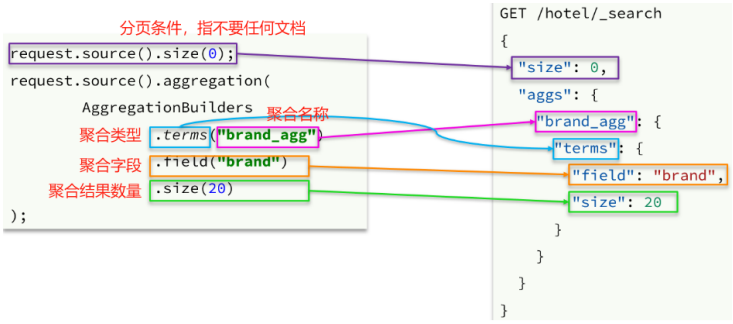
3）管道（pipeline）聚合：在其它聚合的结果上再做聚合；

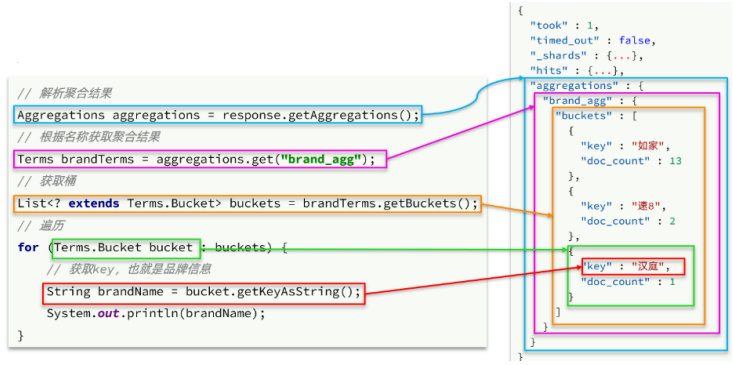
【注意】1>参加聚合的字段必须是 **keyword**、**日期**、**数值**、**布尔类型**；

2> aggs 代表聚合，此时同级的 query 是为了限定聚合文档的范围；

3>聚合必须的三要素：聚合名称、聚合类型、聚合字段；

4>可配置属性：size()聚合结果数量；order()：聚合结果排序方式；





24、数据同步（3 种方案）

1）同步调用

某服务对数据库完成修改时，同时远程调用搜索服务中提供的接口用于更新 ES；（实现简单，但，耦合度高）

2）异步调用

某服务对数据库完成修改后，发送 MQ 消息，搜索服务进行监听 MQ，然后收到消息后修改 ES；（低耦合，但，依靠 MQ 可靠性）

3）监听 binlog

在 mysql 数据库中开启 binlog 功能；mysql 完成修改的操作都会记录在 binlog 中，用 canal 监听 binlog 变化，实时通知搜索服务更新 ES。

（完全解除微服务间耦合，但，开启 binlog 增加数据库负担）

25、ES 集群各节点的职责划分

1）备选主节点：主节点可以管理和记录集群状态，决定分片在哪个节点处理创建和删除索引库的要求；（对 **CPU 要求高**，但内存要求低）

2）data 节点：**存储数据**、搜索、聚合、CRUD 操作；（**CPU 和内存要求都高**）

3）ingest 节点：数据存储之前的**预处理**；

4）coordinating 节点：**路由请求到其他节点**，**合并**其他节点处理的**结果**，**返回给用户**。（对网络带宽、**CPU** 要求高）

【注意】**默认**情况下，集群中任何节点**都具备**四种角色，但真实的集群中**各节点职责分离**。根据不同节点的需求**分配不同的硬件**去部署，**避免业务之间的互相干扰**。

26、ES 的 master 选举流程？

1）**默认所有候选节点**都有**被选举**和**投票权利**，根据 **nodeId 字典排序**；

2）选举时，**每个节点**都把自己**所知道的节点**排一次序，然后**选第一个节点**暂定为主节点；

3）如果对某个节点的***投票数达到 n/2+1***，并且**自己选举自己**，则为**主节点**；否则重新选举一直满足上述条件。

27、ES 集群脑裂问题

由于集群中的节点失联，这些**节点与主节点失联**，**重新选主**并**继续对外服务**，导致与原集群数据不同步；当**网络恢复后**，该集群有**两个主**节点，**集群状态不一致**，故称为脑裂。

【失联原因】**1）网络问题**：集群间的网络延迟导致一些节点访问不到主节点，重新选主并重新分配主分片；

2）节点负载：主节点的角色**既为 master 又为 data**，**访问量较大**时可能会导致 **ES 停止响应**造成大面积**延迟**，认为主节点挂了，重新选主；

3）内存回收：***data 节点上***的 **ES 进程**占用的**内存较大**，引发 **JVM 的大规模内存回收**，造成 **ES 进程失去响应**。

【解决】**1）节点状态的响应时间**适当调大（默认为 3s）；

2）控制**选举行为发生**的**最小集群主节点数量**适当调大（大于该值的节点数量认为挂了，才能重新选举）；

3）职责分离，node.master 和 node.data 节点分离。

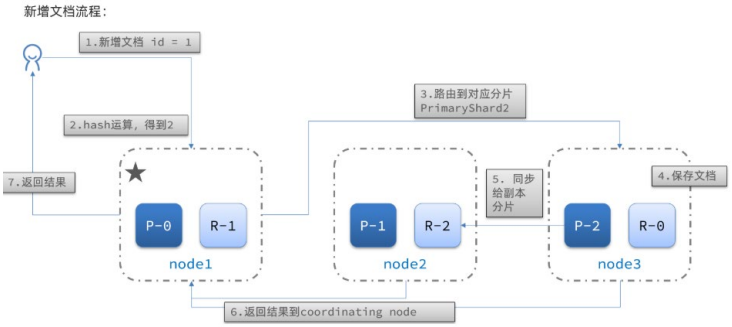
27、ES 分布式存储过程（新增单个文档的流程）

1）ES 会利用**文档 id** 通过 **hash 算法**来计算文档应该**存储到哪个分片**；（**算法与分片数量有关**，因此索引库一旦创建，**分片数量不能修改**）

2）**路由**到**对应主分片所在节点**上，在**该节点上执行请求**，保存文档；

3）保存成功后，同步**将请求转发给其他分片节点**；

4）**所有分片**都**保存成功后**向**协调节点返回结果**，协调节点**返回给用户**。

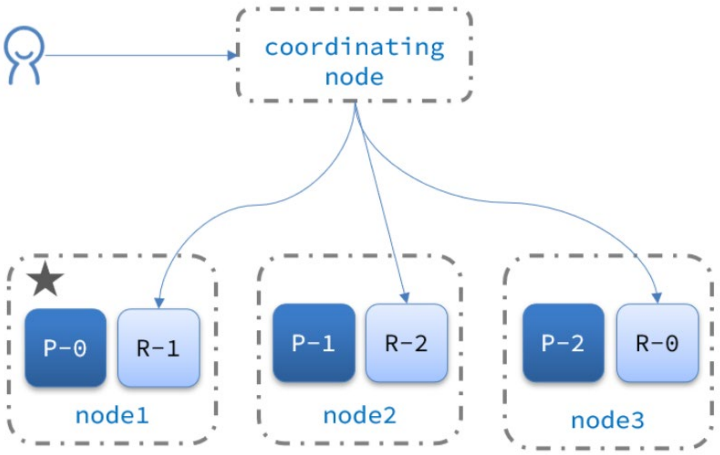


28、ES 分布式搜索查询过程（分散和聚集阶段）

1）分散：协调节点会把**请求分发到每一个分片**上，每个分片**在本地搜索**，并构建一个 from+size 的**优先队列**，将**所有文档 ID** 和**排序值**放在优先队列中；

2）聚合：**协调节点**向各分片**发送 get 请求**，**合并**他们优先队列的**信息到自己的优先队列**中，并**全局排序**，**返回给用户**最终结果。

【注意】**相关性打分**是在**本地分片**上进行的，最后结果不够准确，特别好的分片最后一名在另一个分片里是第一名，却被省略了。



29、Elasticsearch 更新和删除文档的流程？

1）更新和删除都是写操作，但 **ES 中文档不可变**。

2）在**创建新文档**时，**ES** 会为该文档**指定一个版本号**，当**执行更新**时，**旧版本的文档**在.del 文件中**被标记为删除**，**新版本的文档**被索引到一个**新段**。旧版本的文档依然**能匹配查询**，但是会在**结果中被过滤掉**。

3）**磁盘上每个段**都有一个**相应的.del 文件**。当删除请求发送后，文档**并没有真的被删除**，而是在.del 文件中**被标记为删除**。该文档依然**能匹配查询**，但是会**在结果中被过滤掉**。当段**合并时**，在**.del 文件中被标记**为删除的文档将**不会被写入新段**。

30、如何在保留不变性的前提下实现倒排索引的更新？

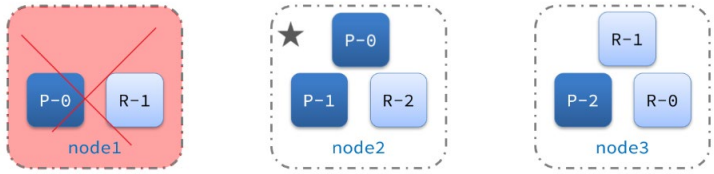
用更多的索引。通过**增加新的补充索引**来反映新近的修改，而**不是直接重写整个倒排索引**。**每一个倒排索引**都会**被轮流查询到**，从最早的开始查询完后再对结果进行合并。

31、在并发情况下，Elasticsearch 如果保证读写一致？

通过**版本号使用乐观锁**并发控制，以确保新版本不会被旧版本覆盖。

32、集群故障转移

集群的**主节点**会**监控集群中的节点状态**，如果发现**有节点宕机**，会立即**将宕机节点的分片**数据**迁移到其它节点**，确保数据安全。



33、深分页问题

每次分页查询都要**查询（页数*size）数量的文档**，当**页数越深**，每个分片每次查询这么多数据，还需**再次聚合排序**，对**内存**和 **CPU** 产生**压力**。

【解决】**1）search after**：分页时需要排序，原理是**从上一次的排序值开始**，**查询下一页数据**。官方推荐使用的方式。

2）scroll：原理将**排序后的文档 id 形成快照**，**保存在内存**。不推荐