> 主讲老师：Fox
> 有道云笔记地址：https://note.youdao.com/s/DMuehv8e

1. 什么是深度分页

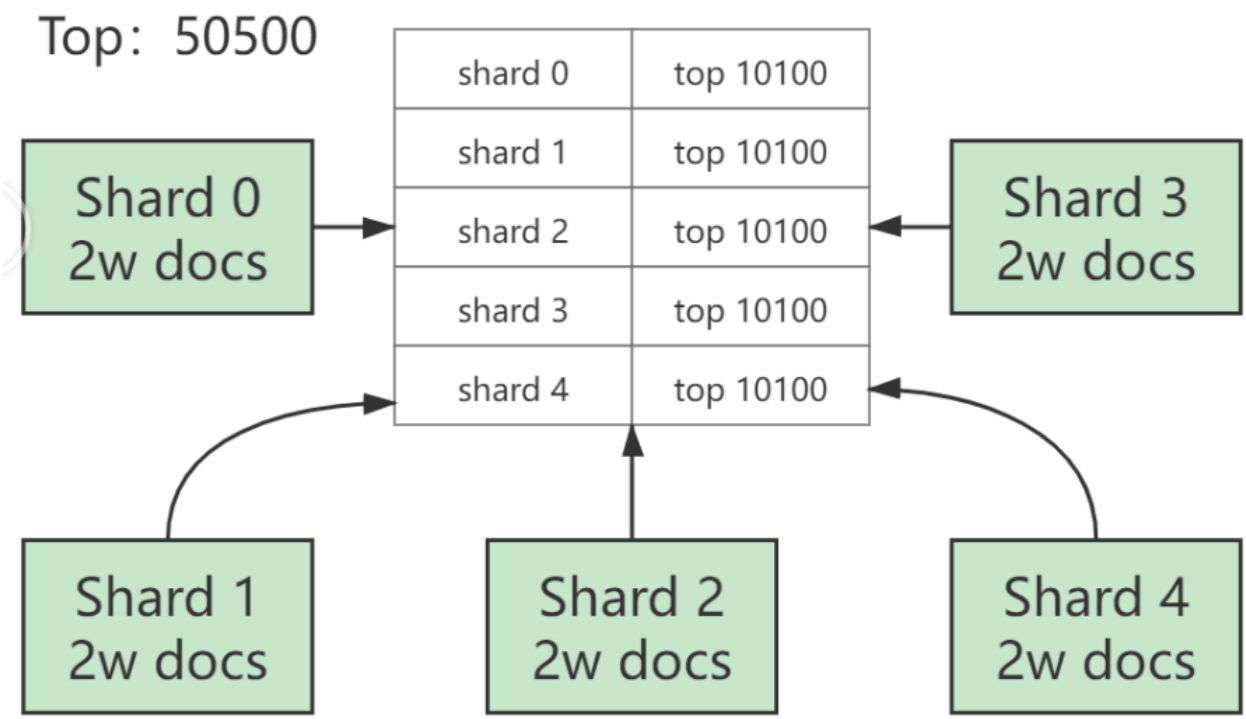
深度分页是指在处理大数据集查询时，用户尝试访问多页数据中较后面的页面时遇到的问题。当尝试访问排序后的数据列表的第1000页或更后面的页面时，数据库需要先跳过前面数十万条记录，这一过程通常涉及大量的数据扫描和排序，极大地增加了数据库的查询负载，从而成为性能瓶颈。

ES分页查询流程大致如下：

1. 数据存储在各个分片中，协调节点将查询请求转发给各个节点，当各个节点执行搜索后，将排序后的前N条数据返回给协调节点。
2. 协调节点汇总各个分片返回的数据，再次排序，最终返回前N条数据给客户端。
3. 这个流程会导致一个深度分页的问题，也就是翻页越多，性能越差，甚至导致ES出现OOM。

在分布式系统中，对结果排序的成本随分页的深度成指数上升。

从10万名高考生中查询成绩为的10001-10100位的100名考生的信息。



从上面案例中不难看出，每次有序的查询都会在每个分片中执行单独的查询，然后进行数据的二次排序，而这个二次排序的过程是发生在heap中的，也就是说当你单次查询的数量越大，那么堆内存中汇总的数据也就越多，对内存的压力也就越大。这里的单次查询的数据量取决于你查询的是第几条数据而不是查询了几条数据，比如你希望查询的是第10001-10100这一百条数据，但是ES必须将前10100全部取出进行二次查询。因此，如果查询的数据排序越靠后，就越容易导致OOM（Out Of Memory）情况的发生，频繁的深分页查询会导致频繁的FullGC。

ES为了避免用户在不了解其内部原理的情况下而做出错误的操作，设置了一个阈值，即max_result_window，其默认值为10000，其作用是为了保护堆内存不被错误操作导致溢出。

2. 深度分页不推荐使用from+size

在Elasticsearch中，分页查询的实现主要通过两个参数from和size来实现。from参数指定了从结果集中的第几条数据开始返回，而size参数指定了返回数据的数量。正常情况下分页代码如实下面这样的：

```
# 查询第一页5条数据
GET /employee/_search
{
  "query": {
    "match_all": {}
  },
  "from": 0,
  "size": 5
}

```

输出结果如下图：

但是如果我们查询的数据页数特别大，当from + size大于10000的时候，就会出现问题，如下图报错信息所示：

分析可知，查询结果的窗口大小超过了最大窗口的限制，而index. max_result_window默认值为10000。
Elasticsearch会限制最大分页数，避免因大数据量的召回导致系统性能低下。Elasticsearch的max_result_window默认值是10000，意味着每页有10条数据，会最大翻页至1000页。主流搜索引擎实际都翻不了那么多页。
对此，有两个可行的解决方案，如下所示：

- 方案一：对于大型数据集，我们可采用scroll API来召回数据。这个策略我们将在后续的内容中进行详细分析。
- 方案二：调大index.max_result_window默认值

```
PUT /employee/_settings
{
  "index.max_result_window": 20000
}
```

官方建议避免使用from+size来过度分页或一次请求太多结果。

不推荐使用from+size来深度分页的核心原因如下：

- 搜索请求通常会跨多个分片，每个分片必须将其请求的命中内容以及先前页面的命中内容加载到内存中。
- 对于分页较多的页面或大量结果，这样操作会显著增加内存和CPU使用率，导致性能下降，甚至导致节点故障。

from+size查询的优缺点及适用场景

from+size分页查询的优缺点如下：

- from+size查询优点：支持随机翻页。
- from+size查询缺点：
  - 限于max_result_window设置，不能无限制翻页；
  - 存在深度翻页问题，越往后翻页越慢。

from+size查询适用场景如下：

- 非常适合小型数据集或者从大数据集中返回Top N(N≤10000)结果集的业务场景。
- 主流PC搜索引擎中支持随机跳转分页的业务场景，如下图所示：

3. 深度分页问题的常见解决方案

尝试避免使用深度分页

解决深度分页问题最好的办法就是避免使用深度分页。谷歌、百度目前作为全球和国内做大的搜索引擎不约而同的在分页条中删除了“跳页”功能，其目的就是为了避免用户使用深度分页检索。

在百度中搜索“Elasticsearch”，在搜索结果中翻到第20页，就无法再往下翻页了，提示信息如下图：

淘宝虽然没有删除“跳页”功能，但不管我们搜索什么内容，只要商品结果足够多，返回的商品列表都是仅展示前100页的数据，其本质和ES中的max_result_window作用是一样的，都是限制你去搜索更深页数的数据。

手机端APP就更不用说了，直接是下拉加载更多，连分页条都没有，相当于你只能点击“下一页”。

Scroll Search滚动查询

scroll API可从单个搜索请求中检索大量结果（甚至所有结果），这种方式与传统数据库中的游标(cursor)类似。scroll滚动遍历查询是非实时的，数据量大的时候，响应时间可能会比较长。

> 官方文档：https://www.elastic.co/guide/en/elasticsearch/reference/8.14/paginate-search-results.html#scroll-search-results

ES7之后，官方已经不再建议使用scroll API进行深度分页。如果要分页检索超过 Top 10,000+ 结果时，推荐使用：search_after。

适合场景

单个滚动搜索请求中检索大量结果，即非“C端业务”场景

实现步骤

scroll查询的核心执行步骤如下。

1) 第一次进行scroll查询，指定检索语句的同时设置scroll上下文保留时间。

scroll请求返回的结果反映了发出初始搜索请求时索引的状态，就像在那一个时刻做了快照，随后对文档的更改（写入、更新或删除）只会影响以后的搜索请求。

```
# 使用kibana提供的航班测试数据集
#查询命令中新增scroll=5m,说明采用游标查询，保持游标查询窗口5分钟，也就是本次快照的结果缓存起来的有效时间是5分钟。
GET /kibana_sample_data_flights/_search?scroll=5m
{
  "query": {
    "term": {
      "OriginWeather": "Sunny"
    }
  },
  "size": 100
}
```

返回结果：

2) 向后翻页，继续获取数据，直到没有要返回的结果为止

```
# scroll_id 的值就是上一个请求中返回的 _scroll_id 的值
GET /_search/scroll
{
    "scroll": "5m",
    "scroll_id" :
"FGluY2x1ZGVfY29udGV4dF91dWlkDXF1ZXJ5QW5kRmV0Y2gBFnQ5MUF6M3dYUkhPQW81czY3RXBDckEAAAAAAB
kMUBZPOVotS1A1MlI1dU43QXFsdkRGUEhB"
}
```

多次根据scroll_id游标查询，直到没有数据返回则结束查询。采用游标查询索引全量数据，更安全高效，限制了单次对内存的消耗。

删除游标scroll

scroll超过超时后，搜索上下文会自动删除。然而，保持scroll打开是有代价的，因此一旦不再使用，就应明确清除scroll上下文

```
DELETE /_search/scroll
{
    "scroll_id" :
"FGluY2x1ZGVfY29udGV4dF91dWlkDXF1ZXJ5QW5kRmV0Y2gBFmNwcVdjblRxUzVhZXlicG9HeU02bWcAAAAAAA
BmzRY2YlV3Z0o5VVNTdWJobkE5Z3MtXzJB"
}
```

scroll查询的优缺点及适用场景

scroll查询的优缺点如下：

- **scroll查询优点：**支持全量遍历，是检索大量文档的重要方法，但单次遍历的size值不能超过max_result_window的大小。
- **scroll查询缺点：**
  - 响应是非实时的；
  - 保留上下文需要具有足够的堆内存空间；
  - 需要通过更多的网络请求才能获取所有结果。

scroll查询的适用场景如下：

- 大量文档检索：当要检索的文档数量很大，甚至需要全量召回数据时，scroll查询是一个很好的选择。
- 大量文档的数据处理：滚动API适合对大量文档进行数据处理，例如索引迁移或将数据导入其他技术栈。

注意：

1) ES7.x之后不建议使用scroll API进行深度分页。

2) 如果要分页检索并获得超过10000条结果时，则推荐使用search_after。

search_after查询

search_after查询的基本工作原理是以前一页结果的排序值作为参照点，进而检索与这个参照点相邻的下一页的匹配数据。

这种方法在处理大规模数据分页时更为高效且实用。使用该查询的前置条件是要求后续的多个请求返回与第一次查询相同的排序结果序列。也就是说，在后续翻页的过程中，即便有新数据写入等操作，也不会对原有结果集构成影响。

scroll API适用于高效的深度滚动，但滚动上下文成本高昂，不建议将其用于实时用户请求。而search_after参数通过提供一个活动光标来规避这个问题。这样可以使用上一页的结果来帮助检索下一页。

> 官方文档：https://www.elastic.co/guide/en/elasticsearch/reference/8.14/paginate-search-results.html#search-after

那么，如何实现呢?

可以创建一个时间点PIT(Point In Time)来保障在搜索过程中能保留特定事件点的索引状态。

search_after的后续查询都是基于PIT视图进行的，能有效保障数据的一致性。

> PIT是Elasticsearch 7.10版本之后才有的新特性，实际上是存储索引数据状态的轻量级视图。

实现步骤

search_after 分页查询可以简单概括为如下几个步骤：

1）获取索引的pit

使用 search_after 需要具有相同查询和排序值的多个搜索请求。 如果在这些请求之间发生刷新，结果的顺序可能会发生变化，从而导致跨页面的结果不一致。 为防止出现这种情况，可以创建一个时间点(PIT) 以保留搜索中的当前索引状态。Point In Time（PIT）是 Elasticsearch 7.10 版本之后才有的新特性。

```
# 使用kibana提供的航班测试数据集
# 创建一个时间点(PIT)来保存搜索期间的当前索引状态
POST /kibana_sample_data_flights/_pit?keep_alive=5m
#返回结果如下，会返回一个PID的值
{
  "id":
"4YyPBAEaa2liYW5hX3NhbXBsZV9kYXRhX2ZsaWdodHMWZENSdWh0NWNSai1EdUhpcnBCZXgyZwAWTzlaLUtQNTJSNXVON0FxbHZERlBIQQAAAAAAABkI4hZ0OTFBejN3WFJIT0FvNXM2N0VwQ3JBAAEWZENSdWh0NWNSai1EdUhpcnBCZXgyZwAA"
}

```

keep_alive=5m是一个类似于scroll的参数，表示滚动视图的保留时间是5min，超过5min Elasticsearch会清除这个滚动视图并报错，如下图所示

2） 根据pit首次查询

创建基础查询语句，主要是设置分页的条件

```
GET /_search
{
  "query": {
    "term": {
      "OriginWeather": "Sunny"
    }
  },
  "pit": {
    "id": "4YyPBAEaa2liYW5hX3NhbXBsZV9kYXRhX2ZsaWdodHMWZENSdWh0NWNSai1EdUhpcnBCZXgyZwAWTzlaLUtQNTJSNXVON0FxbHZERlBIQQAAAAAAABkI4hZ0OTFBejN3WFJIT0FvNXM2N0VwQ3JBAAEWZENSdWh0NWNSai1EdUhpcnBCZXgyZwAA",
    "keep_alive": "1m"
  },
  "size": 10,
  "sort": [
    {
      "timestamp": "asc"
    }
  ]
}

```

代码中设置了PIT，因此检索时候就不需要再指定索引。id是基于第一步返回的id值。排序sort指的是按照哪个关键字排序。
在每个返回文档的最后会有两个结果值，如下所示:

在每个返回文档的最后会有两个结果值，如下所示。
其中，1723434063000就是我们指定的排序方式，所以上述示例是基于{"timestamp": "asc"}升序排列的。130代表隐含的排序值，官方文档把这种隐含的字段叫作tiebreaker（决胜字段），tiebreaker代表了每个文档的唯一值，确保分页不会丢失或者分页结果数据出现重复（包括相同页重复和跨页重复）
。

3）根据search_after和pit实现后续翻页。

要获得下一页结果，请使用最后一次命中的排序值（包括 tiebreaker）作为 search_after 参数重新运行先前的搜索。 如果使用 PIT，请在 pit.id 参数中使用最新的 PIT ID。 搜索的查询和排序参数必须保持不变。

```
#后续翻页都需要借助search_after来指定前一页中最后一个文档的sort字段值
GET /_search
{
  "query": {
    "term": {
      "OriginWeather": "Sunny"
    }
  },
  "pit": {
    "id": "4YyPBAEaa2liYW5hX3NhbXBsZV9kYXRhX2ZsaWdodHMWZENSdWh0NWNSai1EdUhpcnBCZXgyZwAWTzlaLUtQNTJSNXVON0FxbHZERlBIQQAAAAAAABkI4hZ0OTFBejN3WFJIT0FvNXM2N0VwQ3JBAAEWZENSdWh0NWNSai1EdUhpcnBCZXgyZwAA",
    "keep_alive": "5m"
  },
  "size": 10,
  "sort": [
    {
      "timestamp": "asc"
    }
  ],
  "search_after": [
    1723434063000,
    130
  ]
}
```

显然，search_after查询仅支持向后翻页。

优缺点

search_after优点：

- 不严格受制于max_result_window，可以无限地往后翻页。此处的“不严格”是指单次请求值不能超过max_result_window，但总翻页结果集可以超过。

search_after缺点：

- 只支持向后翻页，不支持随机翻页。search_after不支持随机翻页，更适合在手机端应用的场景中使用，类似今日头条等产品的分页搜索。

4. ES三种分页方式总结

分页方式	性能	优点	缺点	适用场景
from + size	低	支持随机翻页	受制于max_result_window设置，不能无限制翻页； 存在深度翻译问题，越往后翻译越慢。	需要随机跳转不同页（PC端主流搜索引擎）； 在10000条数据之内分页显示
scroll	中	支持全量遍历，但单次遍历的size值不能超过max_result_window的大小	响应是非实时的；保留上下文需要具有足够的堆内存空间；需要通过更多的网络请求才能获取所有结果。	需要遍历全量数据
search_after	高	不严格受制于max_result_window，可以无限地往后翻页。	只支持向后翻页，不支持随机翻页。	仅需要向后翻页;超过10000条数据，需要分页