> https://www.cnblogs.com/wangzhen3798/p/10070977.html
> https://www.jianshu.com/p/91d03b16af77
> http://ju.outofmemory.cn/entry/340337

浅分页（数据量低于10000条）

通过from+size的方式来进行实现。from定义了目标数据的偏移值，size定义当前返回的事件数目。

```
GET /fs/_search?pretty
{
 "from" : 0 , "size" : 10
}
```

优点：from+size在数据量不大的情况下，效率比较高
缺点：在数据量非常大的情况下，from+size分页会把全部记录加载到内存中，这样做不但运行速递特别慢，而且容易让es出现内存不足而挂掉。比如要取第5001页的数据，在分页的时候，elasticsearch需要首先在每一个节点上取出50020的数据，然后和每一个节点的所有数据进行排序，取出排序后在50010到50020的数据，然后返回。这样随着数据量的增大，每次分页时排序的开销会越来越大。

使用场景：这种分页方式只适合少量数据，因为随from增大，查询的时间就会越大，而且数据量越大，查询的效率指数下降

ES为了保证分页不占用大量的堆内存，避免OOM，参数 `index.max_result_window` 设置了 from+size的最大值为10000。即每页10条的话，最多可以翻到1000页。

深分页

scroll调用本质上是实时创建了一个快照(snapshot)，然后保持这个快照一个指定的时间，这样，下次请求的时候就不需要重新排序了。从这个方面上来说，scroll就是一个服务端的缓存。既然是缓存，就会有下面两个问题：

1. 一致性问题。ES的快照就是产生时刻的样子了，在过期之前的所有修改它都视而不见。
2. 服务端开销。ES这里会为每一个scroll操作保留一个查询上下文(Search context)。ES默认会合并多个小的索引段(segment)成大的索引段来提供索引速度，在这个时候小的索引段就会被删除。但是在scroll的时候，如果ES发现有索引段正处

于使用中，那么就不会对它们进行合并。这意味着需要更多的文件描述符以及比较慢的索引速度。

3. 其实这里还有第三个问题，但是它不是缓存的问题，而是因为ES采用的游标机制导致的。就是你只能顺序的扫描，不能随意的跳页。而且还要求客户每次请求都要带上”游标”。

Scroll API相对于from+size方式当然是性能好很多，但是也有如下问题：

1. Search context开销不小。
2. 是一个临时快照，并不是实时的分页结果。

针对这些问题，ES 5.0 开始推出了 **Search After** 机制可以提供了更实时的游标。它的思想是利用上一页的分页结果来提高下一页的分页请求。

```
GET twitter/tweet/_search
{
 "size": 10,
 "query": {
 "match" : {
 "title" : "elasticsearch"
 }
 },
 "sort": [
  {"date": "asc"},
  {"_id": "desc"}
  ]
}

GET twitter/tweet/_search
{
 "size": 10,
 "query": {
 "match" : {
 "title" : "elasticsearch"
 }
 },
 "search_after": [1463538857, "654323"],
 "sort": [
 {"date": "asc"},
 {"_id": "desc"}
```

```
27   ]
28 }
```

缺点：

1. 只能顺序的翻页，不能随意跳页。

2. 查询结果全量导出，要在短时间内不断重复同一查询成百甚至上千次，效率就显得非常低了，而scroll会把上一次的查询缓存。