



好未来技术通道
TAL TECHNICAL CHANNEL

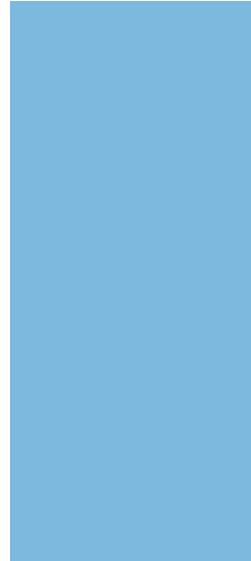
走近ElasticSearch

基础服务中台 — 基础架构部-大数据组

- ▶ 主讲人：杨松柏

2021.06.23

讲师介绍



杨松柏

数据平台高级开发工程师

长期从事大数据架构研发工作，对分布式存储有浓厚的兴趣，喜欢探索新鲜事物，喜欢探索技术本质。目前主要负责大数据架构组Elasticsearch与TiDB的维护和研发工作；对ES有较深入的理解。



Elasticsearch引言

ElasticSearch, 是一个基于**Lucene**构建的**开源**、**分布式**、具有**Restful**风格的**全文检索分析**引擎。

广泛用于：



搜索：网站、电商、企业



日志处理和分析



监测：基础设施指标、应用程序性能监测



分析：业务分析、安全、地理空间数据分析



1 初窥检索的本质

2 原理与基本概念

3 文档操作

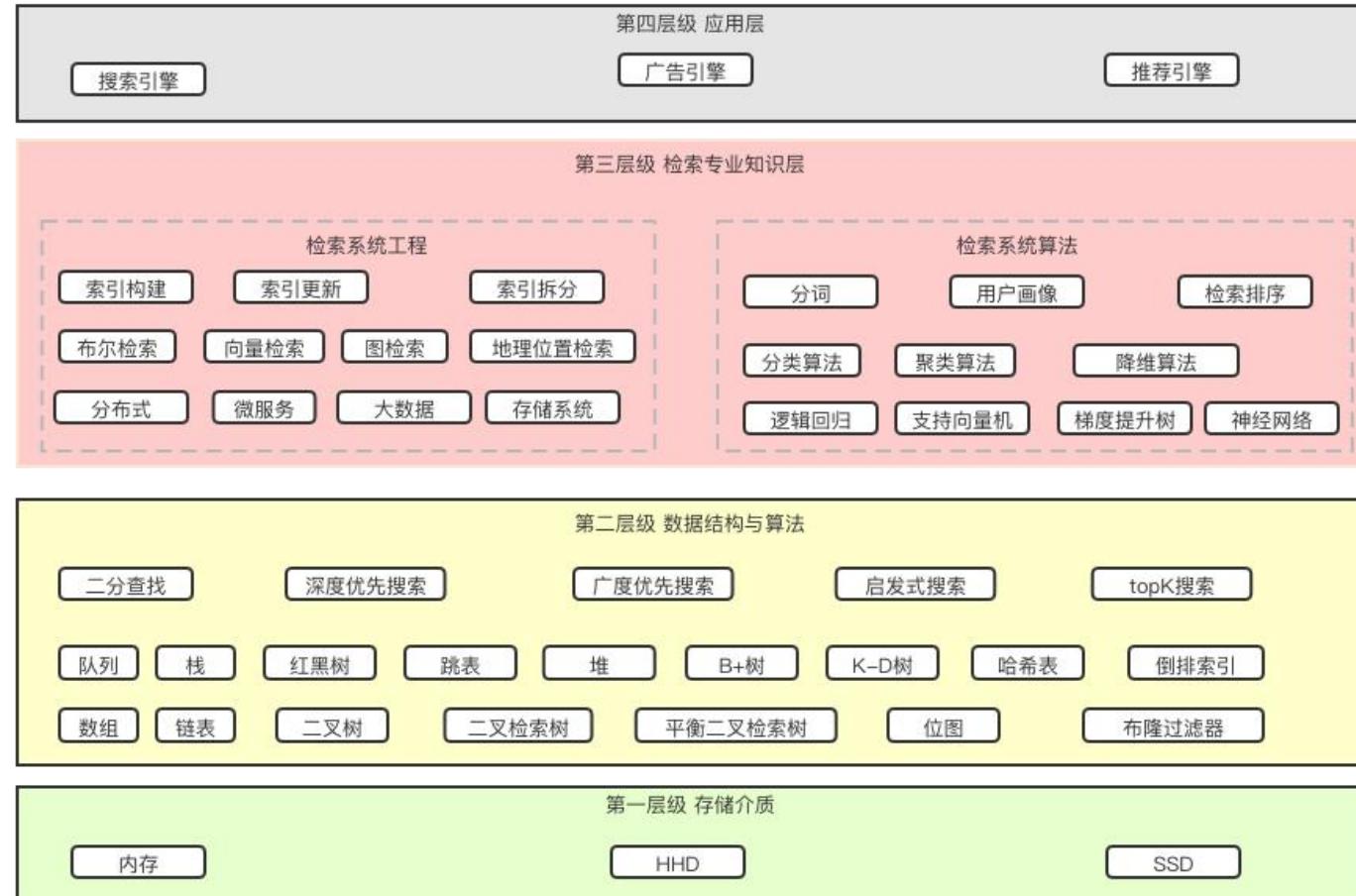
4 分布式搜索

1

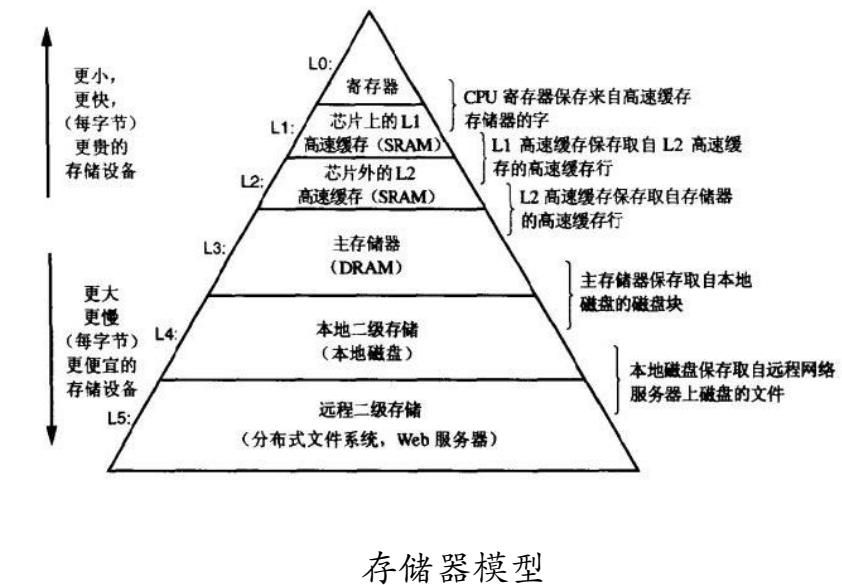
初探检索的本质



检索，其实就是将我们所需要的信息从数据存储的地方高效取出的一种技术。



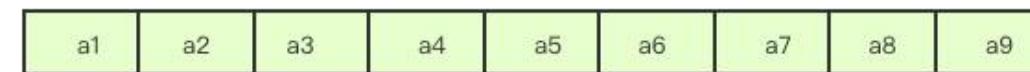
检索技术相关知识(取材自网络)



线性结构的典型代表：数组（连续空间）、链表（不连续空间）；

其他所有的数据结构，比如栈、队列、二叉树、B+ 树等，都不外乎是这两者的结合和变化；

数组



第一步

查找 $k=a_6$

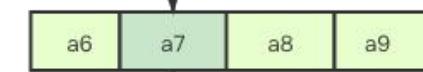
从数组中间开始比较
如果 $k > a_5$, 则继续从右边开始找



查找 $k=a_6$

第二步

从数组中间开始比较
如果 $k < a_7$, 则从左边继续查找



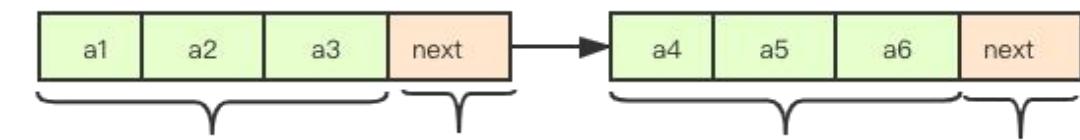
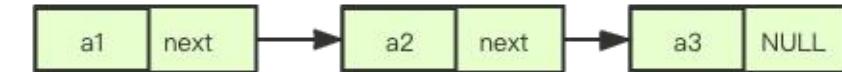
查找 $k=a_6$

第三步

从数组中间开始比较
如果 $k == a_6$, 返回



链表



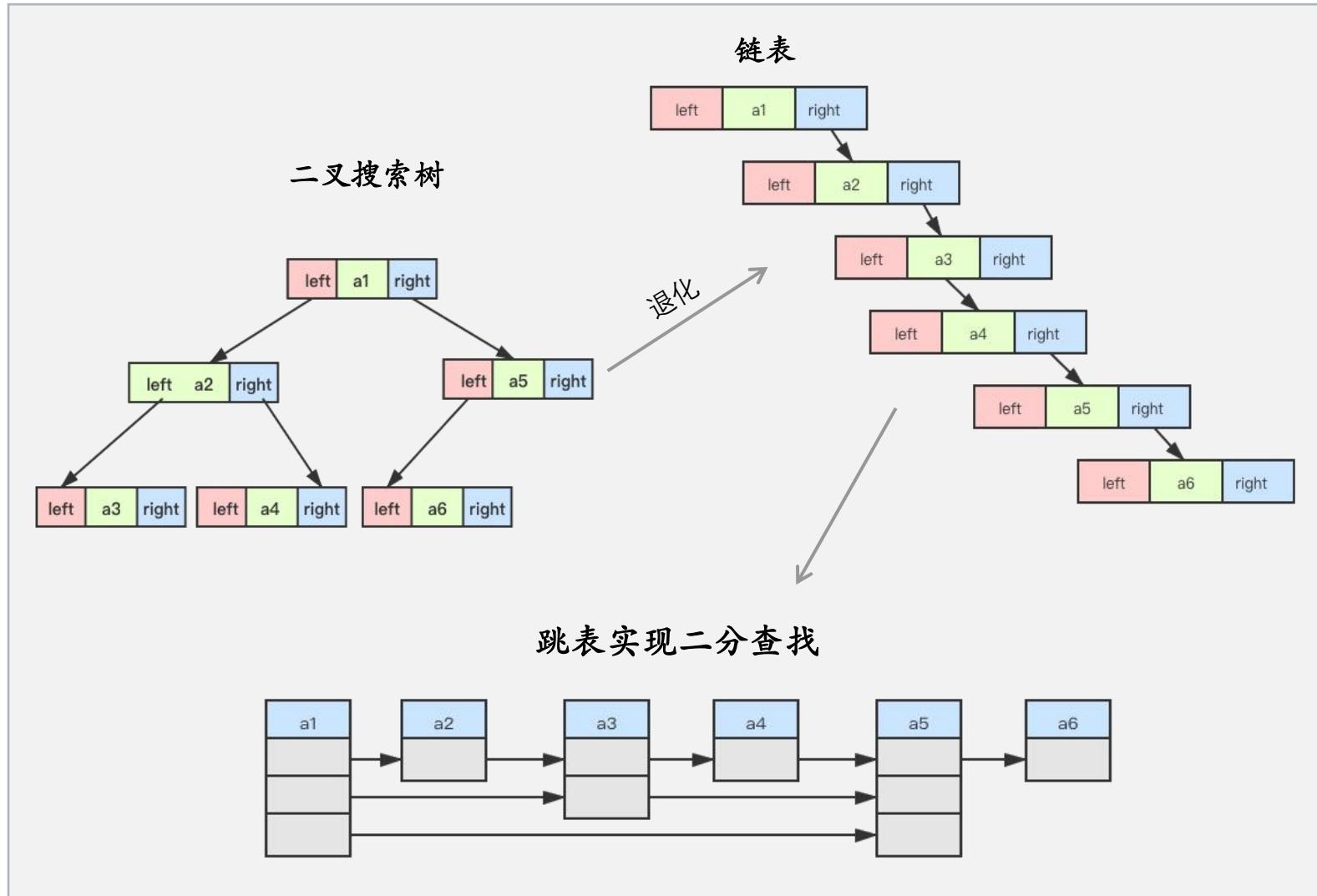
数组

指针

数组

指针

如何进一步的优化链表的查找速率？（树）

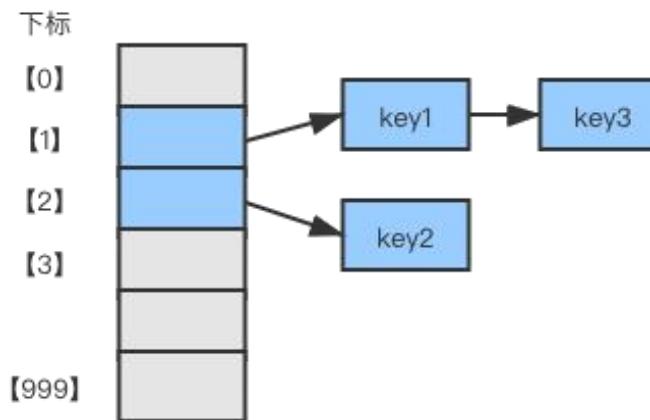


特性：

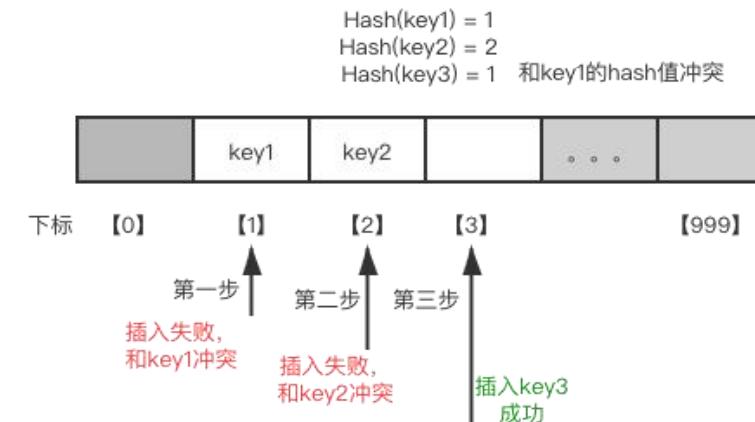
- 各种手段控制二叉搜索数据的平衡性，如：AVL 树（平衡二叉树）和红黑树；
- range查找；



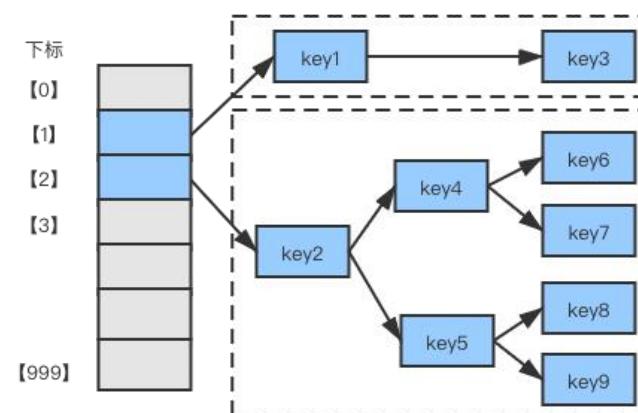
① Hash查找用户 ($O(1)$)



③链表法解决Hash冲突



②开放寻址法解决Hash冲突



④红黑树优化长链表 ($O(1)+O(\log n)$)

问题：

如果hash数组，存储的不是对象而是状态码，我们应该如何优化查找速率？

一个栗子

index—用户id

value—用户状态信息

index1=id/8

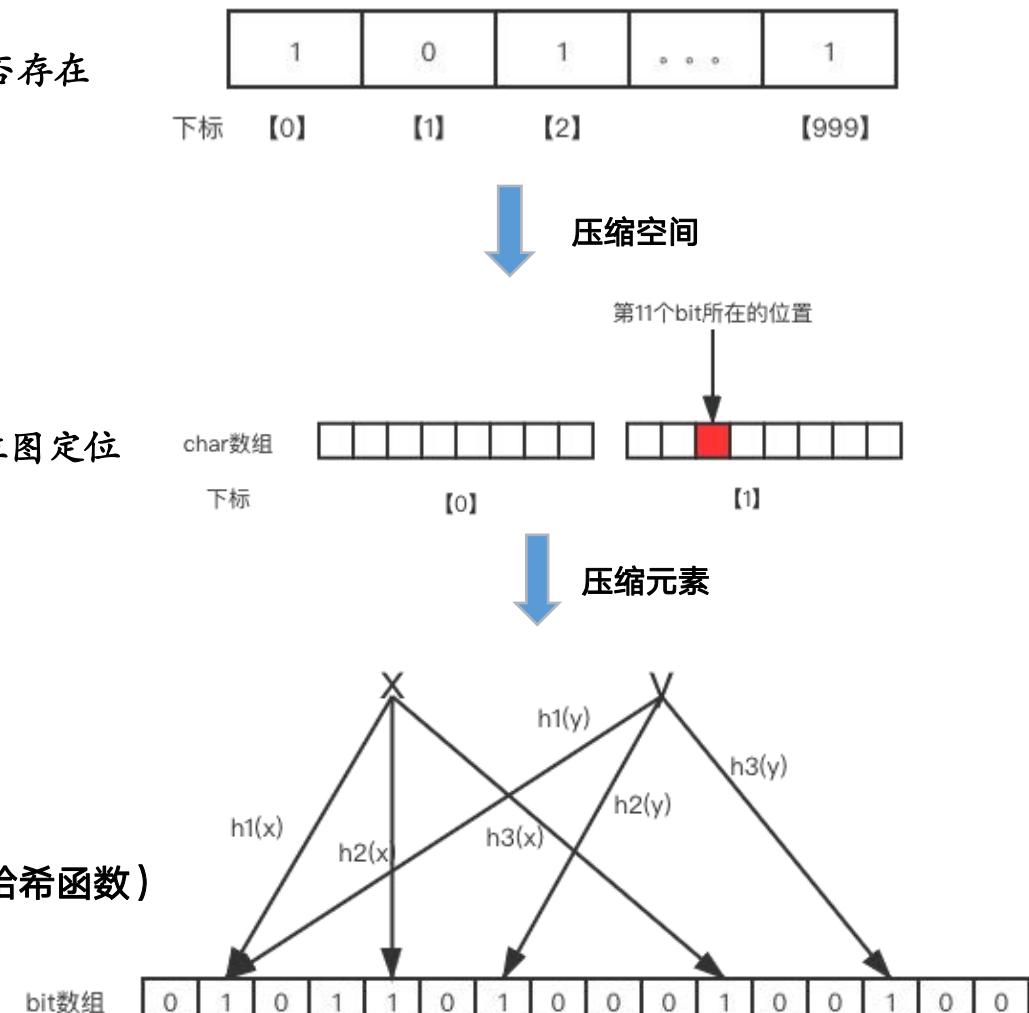
index2=id%8

result=char[index1]&00100000

判断用户是否存在

用户 ID 为 11 的位图定位

Bloom filter (使用三个哈希函数)



索引，辅助数据库表快速访问，是对数据库表中一列或多列的值进行排序的一种结构。

搜索技术中常见：正排索引、倒排索引；

试想这样一个场景：假设你是一位资深王者荣耀的玩家，那么你一定熟悉每位英雄的技能。这个时候：

- 如果给你一个英雄的名字，你可以说出这个英雄有哪几种属性吗（坦克/辅助/法师等等）？（正排索引）
- 如果再问你，有哪些英雄同时包含“坦克”和“辅助”属性？（倒排索引）

第一个问题：

- 给每个英雄一个唯一的编号作为 ID；
- 使用哈希表将英雄的 ID 作为键（Key），把英雄的属性作为键对应的值（Value）；

这样，就能在 $O(1)$ 的时间代价内，完成对指定 key 的检索；

这样一个以对象的唯一 ID 为 key 的哈希索引结构，
即为正排索引（Forward Index）。



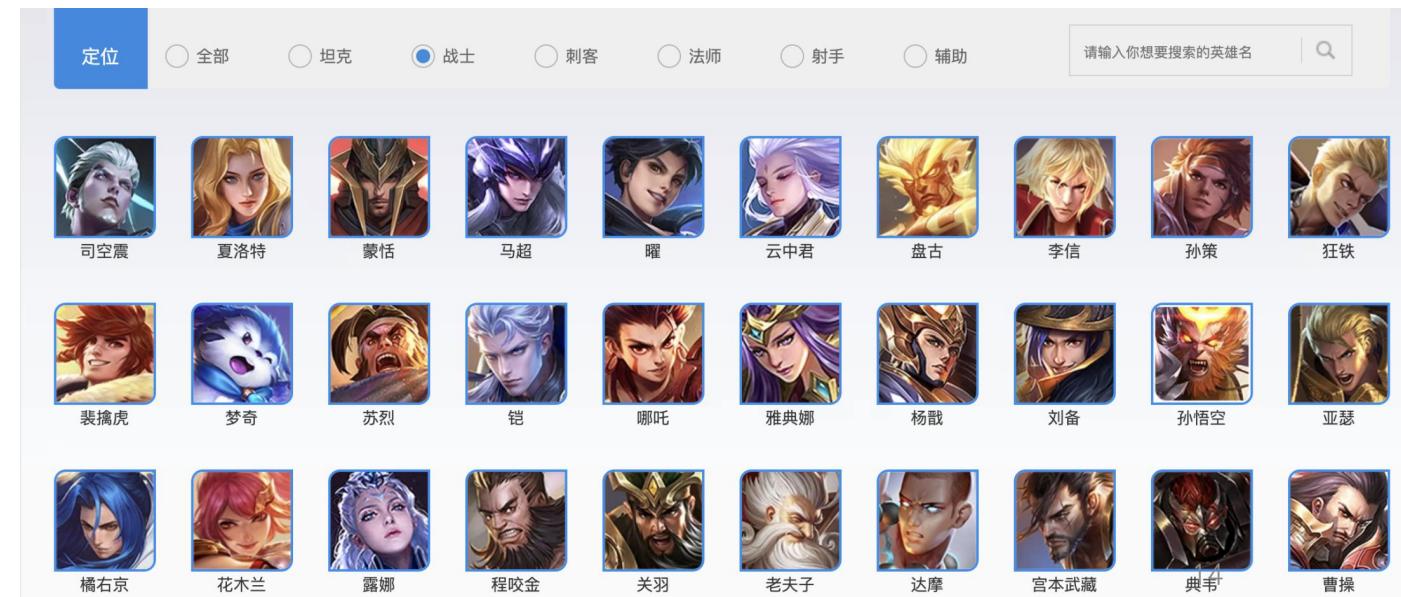
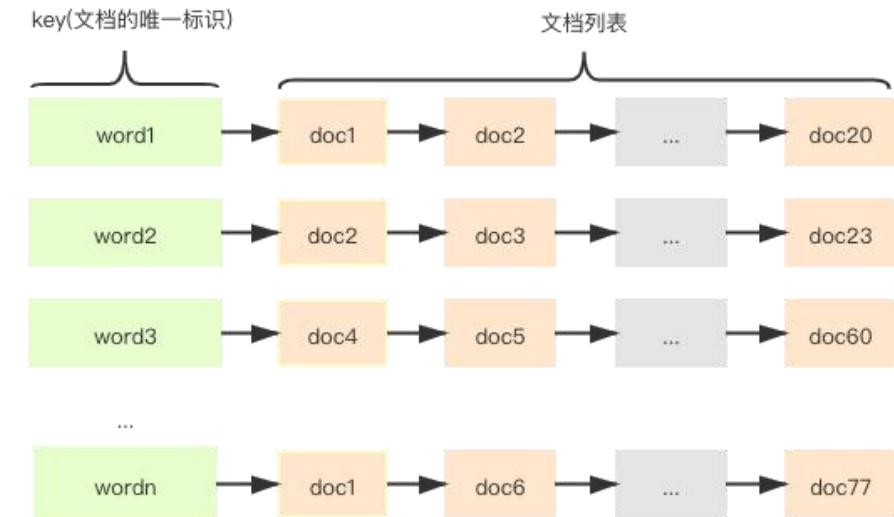
第二个问题：

- 将每个关键字当作 key；
- 将包含了某个关键字文档id的列表进行存储；

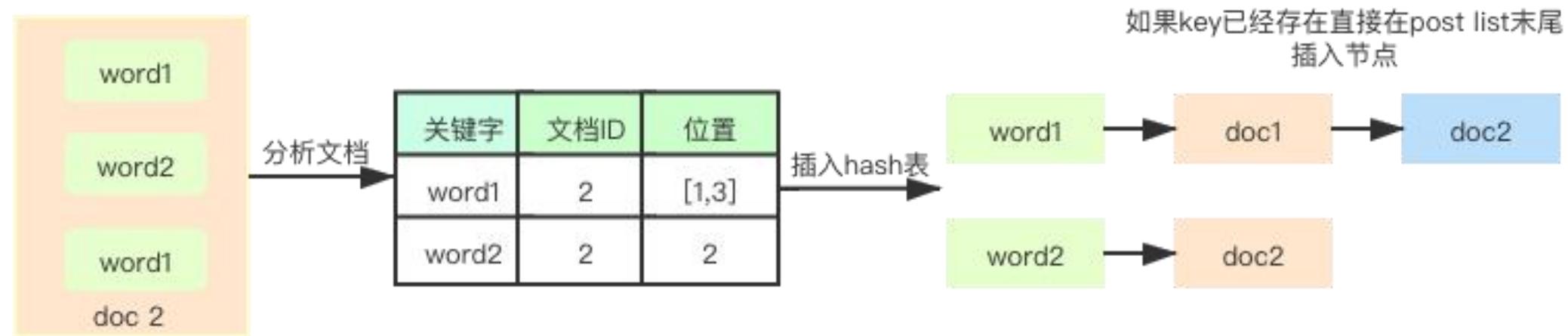
倒排索引

这种根据具体内容或属性反过来索引文档标题的结构，即为倒排索引（Inverted Index），其中：

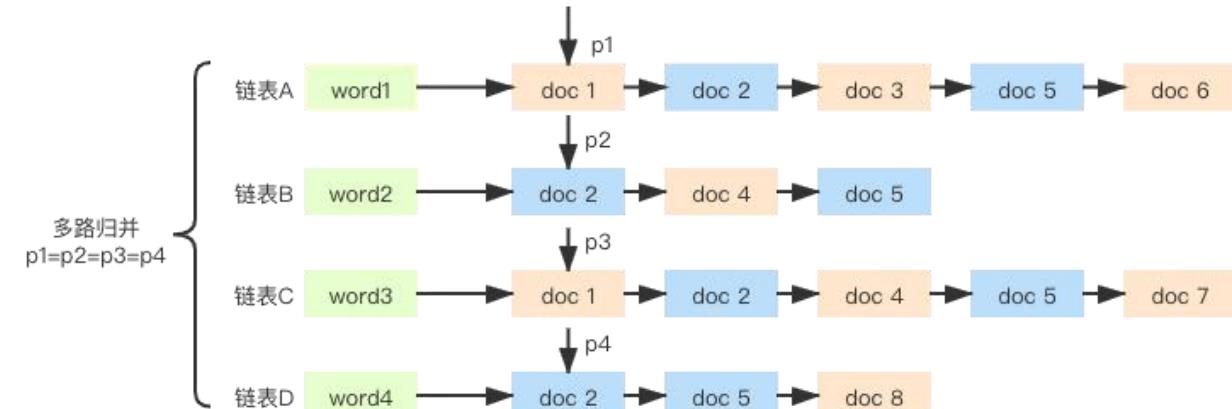
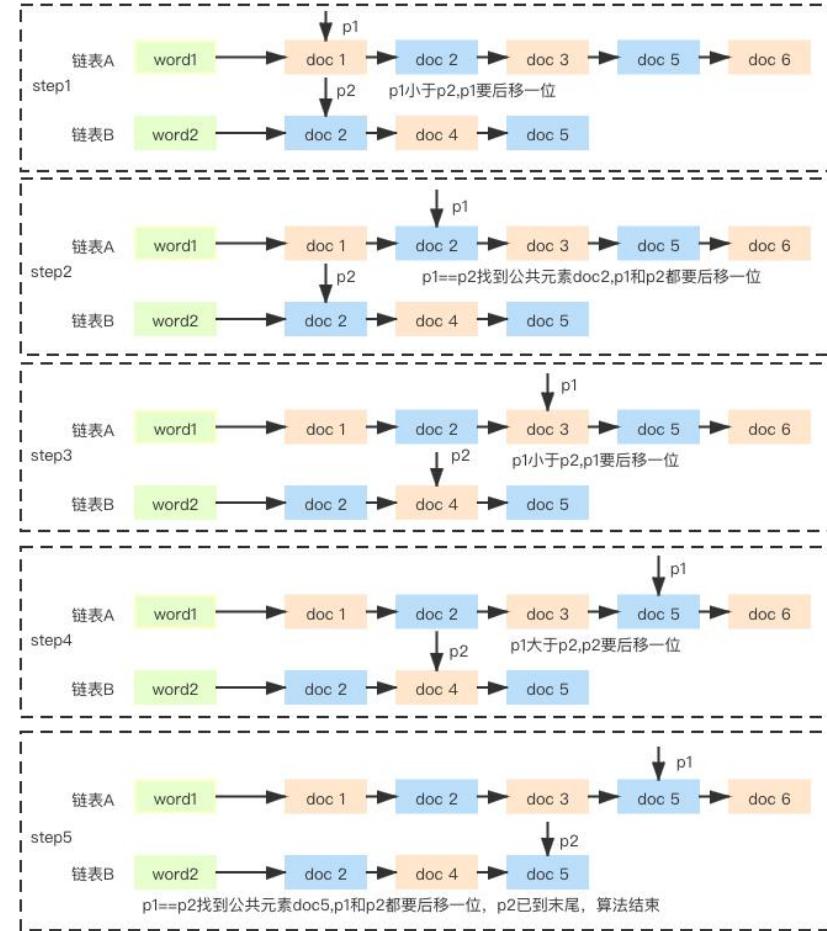
- key 的集合叫作字典（Dictionary）；
- key 后面对应的记录集合叫作记录列表（Posting List）。



- 倒排索引的创建过程：



如何查询同时含有“坦克”字和“辅助”词两个 key 的文档？

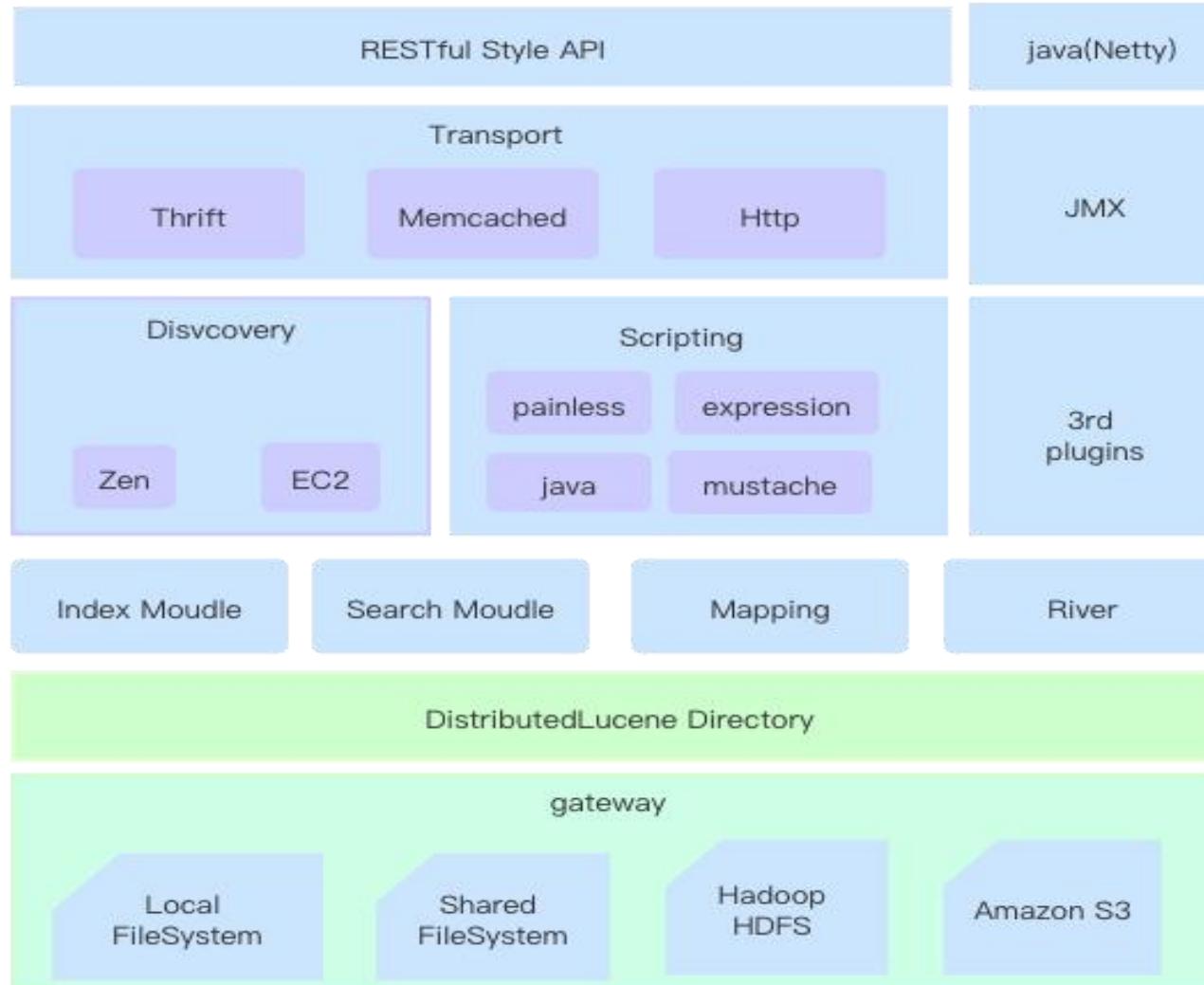


加速倒排索引的查找方法：
跳表、哈希表、位图、Roaring Bitmap等等；

2

原理与基本概念

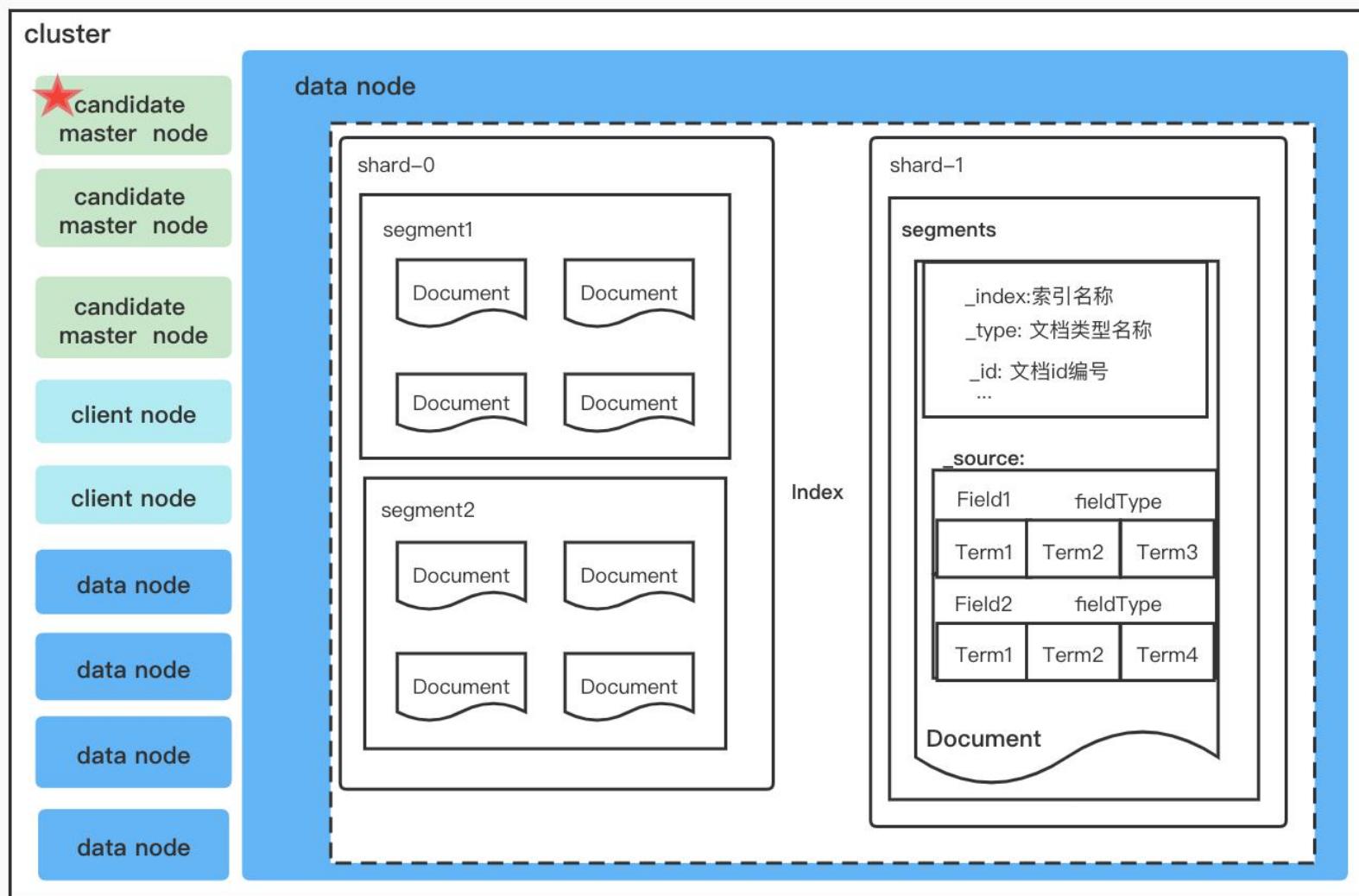
原理与基本概念一ES模块



Elasticsearch模块结构

参数配置

```
cluster.name: elasticsearch-study
node.name: ${HOSTNAME}-test
node.master: true
node.data: true
bootstrap.memory_lock: true
#Transport相关配置
network.host: 172.20.208.47
http.port: 9200
transport.tcp.port: 9300
http.cors.enabled: true
http.cors.allow-origin: "*"
#发现模块相关配置
discovery.zen.ping.unicast.hosts: ["172.20.208.47:9300","172.20.208.47:9301"]
discovery.zen.minimum_master_nodes: 2
discovery.zen.fd.ping_interval: 60s
discovery.zen.fd.ping_timeout: 10s
discovery.zen.fd.ping_retries: 5
#index模板相关配置
indices.fielddata.cache.size: 20%
indices.memory.index_buffer_size: 20%
indices.memory.min_index_buffer_size: 96mb
#分布式目录相关配置
path.data: /elasticsearch/data
path.logs: /home/logs/elasticsearch
path.repo: ["/elasticsearch/snapshot-back"]
#script相关配置
script.painless.regex.enabled: true
script.max_compilations_rate: 150/5m
```



ES集群组成

基本术语

- 集群 (cluster)
- 节点 (node)
- 索引 (index)
- 类型 (type)
- 文档 (document)
- 字段 (field)
- 映射 (mapping)
- 分片 (shards)
- 段 (segements)

- Analysis

即文本分析，把文档转换一系列单词(term/token)的过程，也叫分词；

Analysis是通过Analyzer来实现的。

- Analyzers

分析器，由三种构件块组成的：

- ✓ **character filter 字符过滤器**

过滤html标签 (好未来 --> 好未来), & --> and (I&you --> I and you) ;

- ✓ **tokenizers 分词器**

英文分词可以根据空格将单词分开,中文分词比较复杂,可以采用机器学习算法来分词;

- ✓ **Token filters 词条过滤器**

过滤停用词、非ASCII码值、转换为小写、转换为同义词等;



Analyzer = CharFilters (0个或多个) + Tokenizer(恰好一个) + TokenFilters(0个或多个)

Elasticsearch内置分词器：

- ✓ [Standard Analyzer](#) - 默认分词器，按词切分
- ✓ [Simple Analyzer](#) - 按照非字母切分(符号被过滤)
- ✓ [Stop Analyzer](#) - 小写处理，停用词过滤(the,a,is)
- ✓ [Keyword Analyzer](#) - 不分词，直接将输入当作输出
- ✓ [Whitespace Analyzer](#) - 按照空格切分
- ✓ [Language Analyzer](#) - 提供了30多种常见语言的分词器
- ✓ [Patter Analyzer](#) - 正则表达式，默认\W+(非字符分割)
- ✓ [Fingerprint Analyzer](#)
- ✓ [Customer Analyzer](#) 自定义分词器

开源分词插件：

- ✓ [elasticsearch-analysis-ansj](#)
- ✓ [elasticsearch-analysis-ik](#)
- ✓ [elasticsearch-analysis-pinyin](#)
- ✓ [jieba 分词器](#)

org.elasticsearch.indices.analysis.AnalysisModule.java#setupAnalyzers

```
NamedRegistry<AnalysisProvider<AnalyzerProvider<?>>> analyzers =  
new NamedRegistry<>("analyzer");  
analyzers.register("default", StandardAnalyzerProvider::new);  
analyzers.register("standard", StandardAnalyzerProvider::new);  
analyzers.register("simple", SimpleAnalyzerProvider::new);  
analyzers.register("stop", StopAnalyzerProvider::new);  
analyzers.register("whitespace", WhitespaceAnalyzerProvider::new);  
analyzers.register("keyword", KeywordAnalyzerProvider::new);  
analyzers.extractAndRegister(plugins, AnalysisPlugin::getAnalyzers);  
return analyzers;
```



自定义分词器实战

- [实战指导文档](#)
- [实战控制台](#)

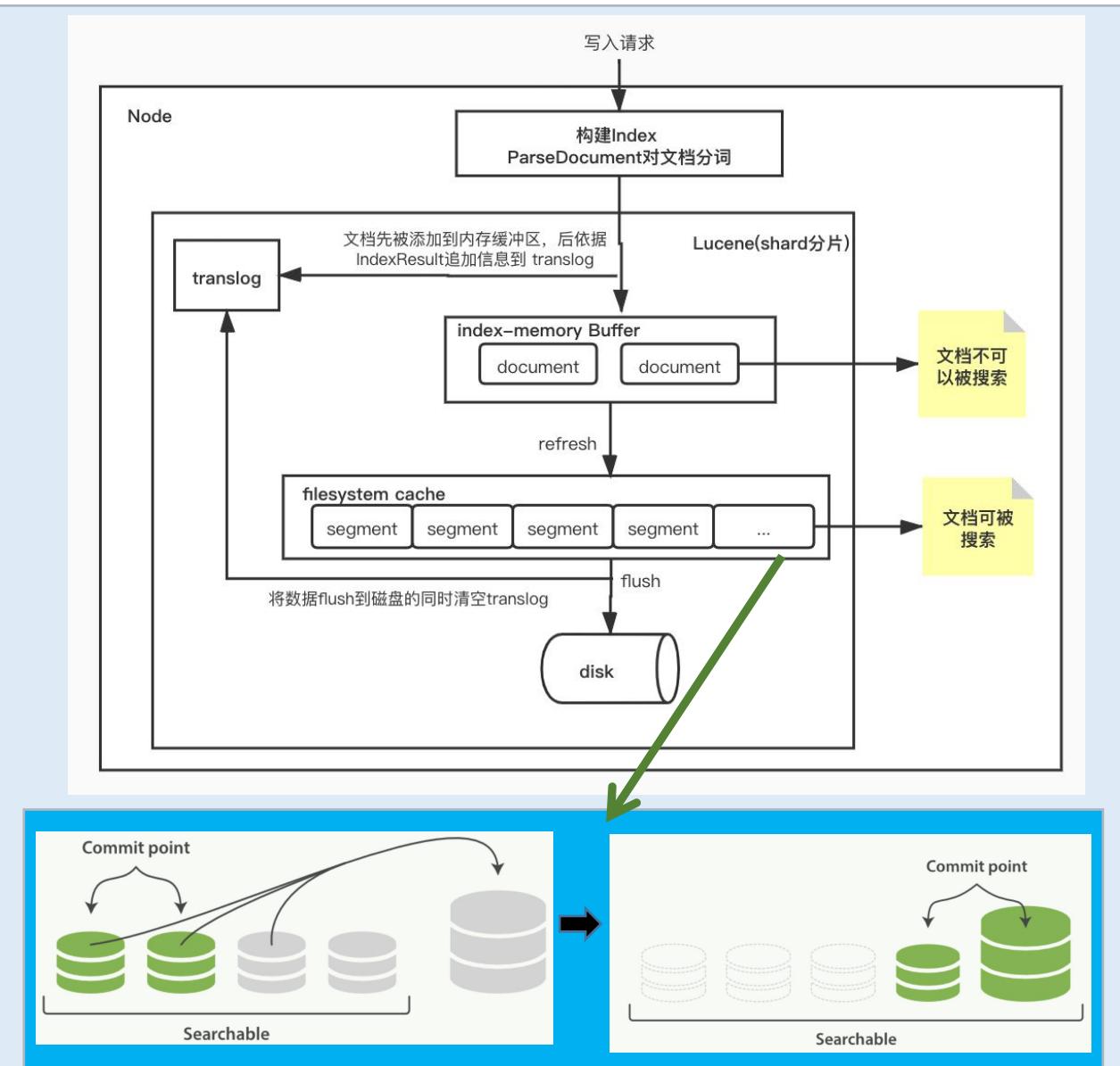
快闪开他要
开始装逼了



原理与基本概念-Lucene、写入流程

数据
写入
流程

段合并



关键点：

- 实时性保证---refresh

```
"index.refresh_interval": "1s"
```

- 安全性保证---translog

```
"index.translog.durability": "async",  
"index.translog.flush_threshold_size": "1024mb",  
"index.translog.sync_interval": "120s"
```

- 大量小Segments的，会影响查询性能，进行Semegent段的合并，减少段的个数；
- 段合并结束，老的段被删除；

可参考源码：
`org.elasticsearch.index.engine.InternalEngine.java`

Segment内部包含的数据结构：

- Inverted Index, 倒排索引
 - Stored Fields, 存储域
 - Document Values, 文档
 - Cache, 动态索引

段合并主要包括：

- 对正向信息的合并，如存储域，词向量，标准化因子等；
 - 对反向信息的合并，如词典，倒排表；

Segments是不可变的 (immutable) :

- **删除**：Lucene将文档标志位置为删除，但文件还是会在它原来的地方，不会改变；
 - **更新**：删除 +重新索引（Re-index）；

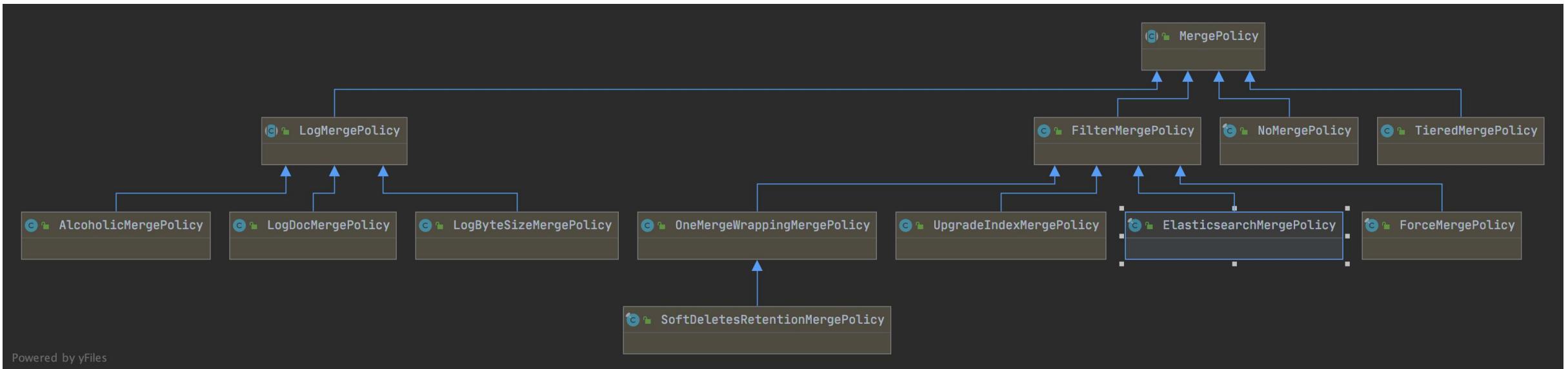
为什么Segments不可变？

- 不需要加锁，提升了并发性能；
 - 查询出的数据保存在缓存中，如过滤查询filter就使用了缓存；
 - 可以节省cpu和io的开销。

```
infoStream.message("IW", "merge codec=" + codec + " maxDoc=" + merge.info.info.maxDoc() + ";  
merged segment has " +  
                    (mergeState.mergeFieldInfos.hasVectors() ? "vectors" : "no vectors")  
+ " ; " +  
                    (mergeState.mergeFieldInfos.hasNorms() ? "norms" : "no norms") + " ; "  
+  
                    (mergeState.mergeFieldInfos.hasDocValues() ? "docValues" : "no  
docValues") + " ; " +  
                    (mergeState.mergeFieldInfos.hasProx() ? "prox" : "no prox") + " ; "  
                    (mergeState.mergeFieldInfos.hasProx() ? "freqs" : "no freqs") + " ; "  
+  
                    (mergeState.mergeFieldInfos.hasPointValues() ? "points" : "no  
points") + " ; " +  
String.format(Locale.ROOT,  
                "%1f sec%s to merge segment [%2f MB, %2f MB/sec]",  
                sec,  
                pauseInfo,  
                segmentMB,  
                segmentMB / sec));  
  
(mergeState.mergeFieldInfos.hasProx() ? "freqs" : "no freqs") wrong message
```

准备课程的时候发现个问题提的issue

<https://issues.apache.org/jira/browse/LUCENE-10009>



org.elasticsearch.index.MergePolicyConfig.java

```
index.merge.policy.expunge_deletes_allowed: "10.0"  
index.merge.policy.floor_segment: "2mb"  
index.merge.policy.max_merge_at_once: "10"  
index.merge.policy.max_merge_at_once_explicit:"30"  
index.merge.policy.max_merged_segment: "5gb"  
index.merge.policy.segments_per_tier: "10.0"  
index.merge.policy.reclaim Deletes_weight: "2.0"
```

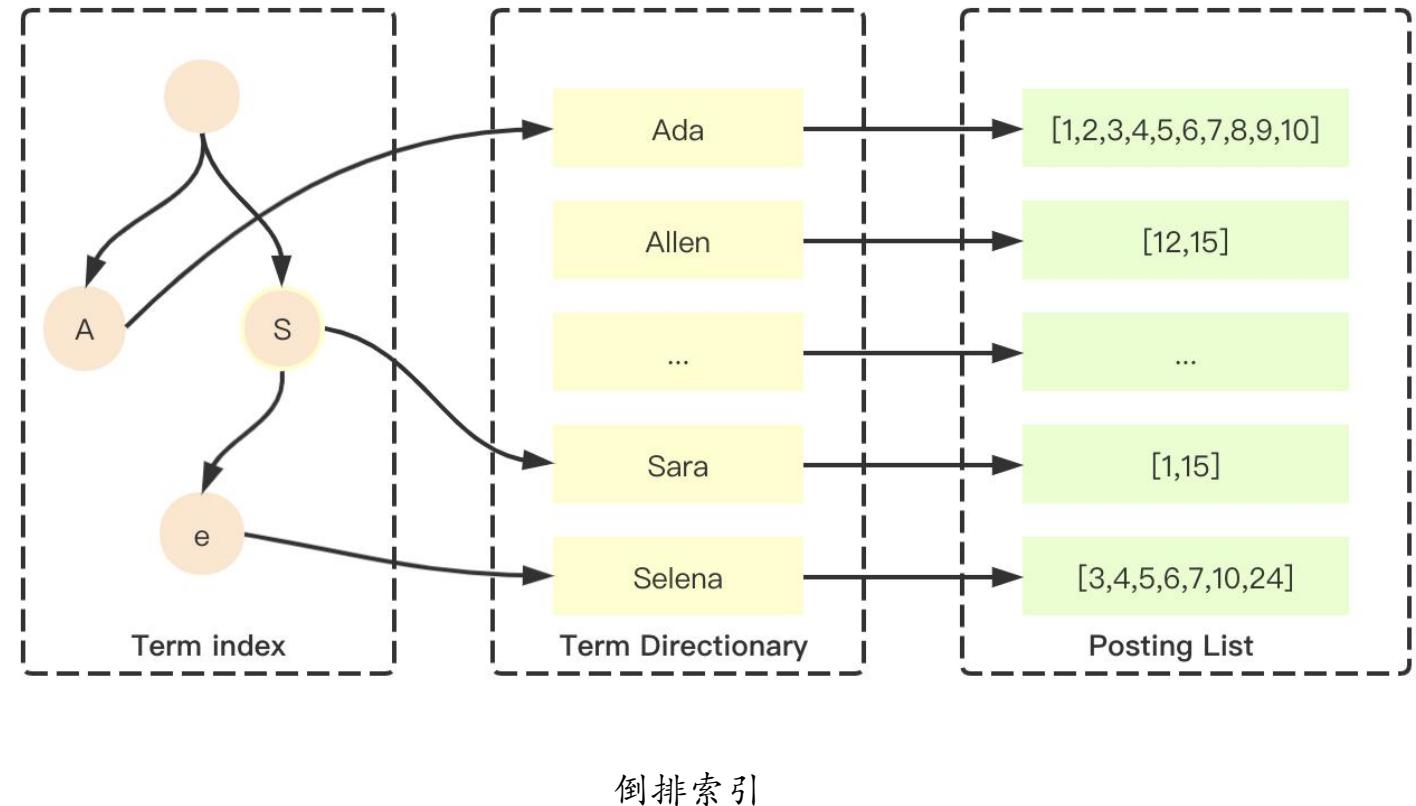
org.elasticsearch.index.MergeSchedulerConfig

```
index.merge.scheduler.max_thread_count:  
Math.max(1, Math.min(4, 配置的可使用CPU数/ 2))  
  
index.merge.scheduler.max_merge_count:  
MAX_THREAD_COUNT_SETTING.get(s) + 5  
  
index.merge.scheduler.auto_throttle: true
```

Elasticsearch为什么不用B+树索引？

倒排索引文件常见压缩方式：

- Roaring Bitmap
 - postlings list
 - numeric doc values
- LZ4
 - Stored fields, term vectors
- FST
 - terms index



分片策略

索引表的allocation主要由allocator与decider来实现：

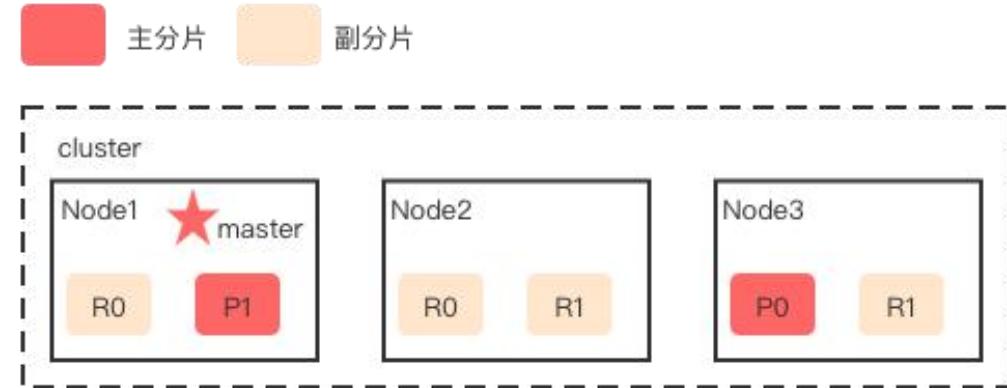
- allocator主要负责收集信息，寻找最优的节点来分配分片，源码所在包
`org.elasticsearch.cluster.routing.allocation.allocator`
- decider主要是做决策是否将分片进行分配，源码所在包
`org.elasticsearch.cluster.routing.allocation`

常见的分配策略

- | | |
|--------------------------------|---------------|
| • DiskThresholdDecider | 磁盘空间控制器 |
| • FilterAllocationDecider | 分配过滤控制器 |
| • MaxRetryAllocationDecider | 最大尝试次数决策器 |
| • ShardsLimitAllocationDecider | shard分配的限制控制器 |

`index.routing.allocation.total_shards_per_node`

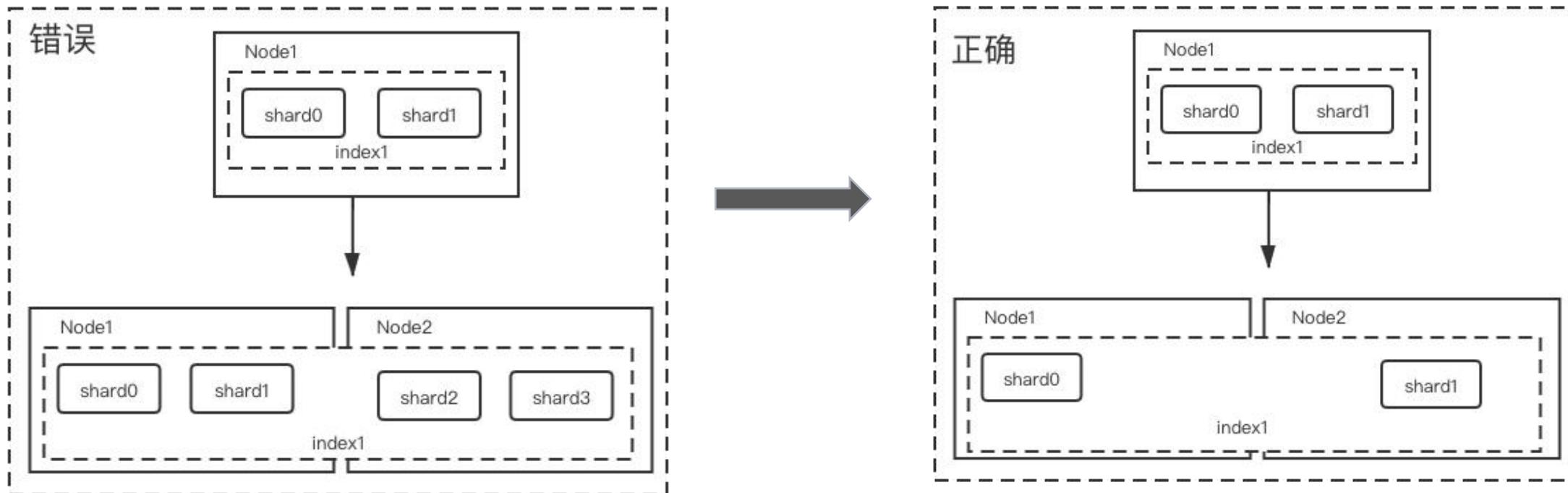
默认值不限制，限制单个索引在每个节点上最多允许的分片数目



诊断分片为什么没有分配成功

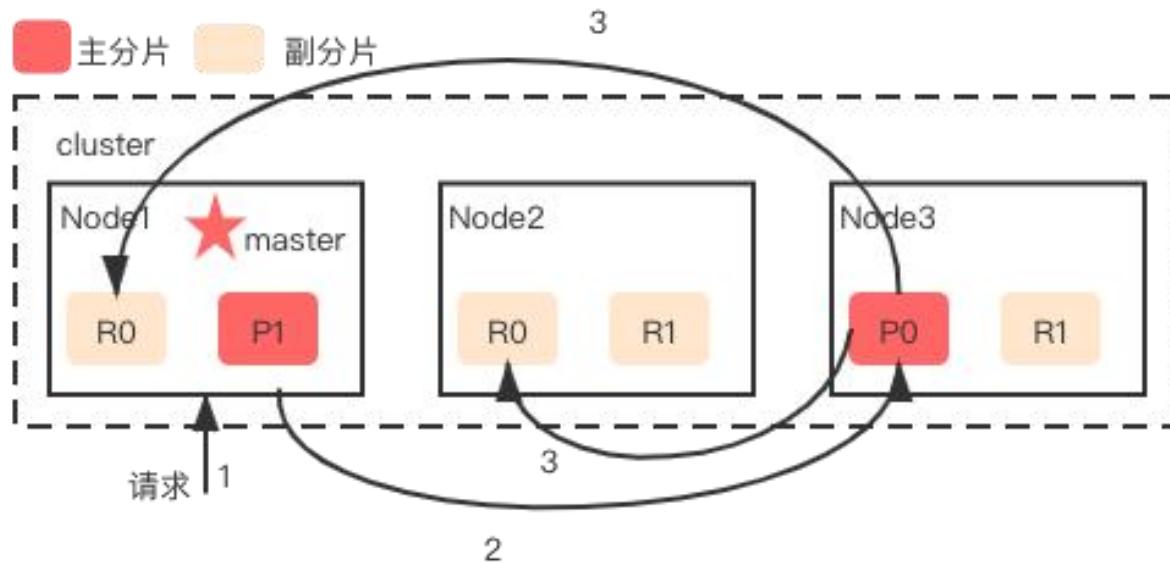
```
POST _cluster/allocation/explain
{
  "index": "homepage_package_index_v2.0.0",
  "shard": 3,
  "primary": false
}
```

Elasticsearch节点scale-out分片存储机制：



3

文档操作



新建, 索引和删除文档

两个重要参数:

✓ wait_for_active_shards

index.write.wait_for_active_shards:1
 $\text{int}((\text{primary} + \text{number_of_replicas}) / 2) + 1$

✓ timeout

默认情况下, 它最多等待1分钟

源码相关:

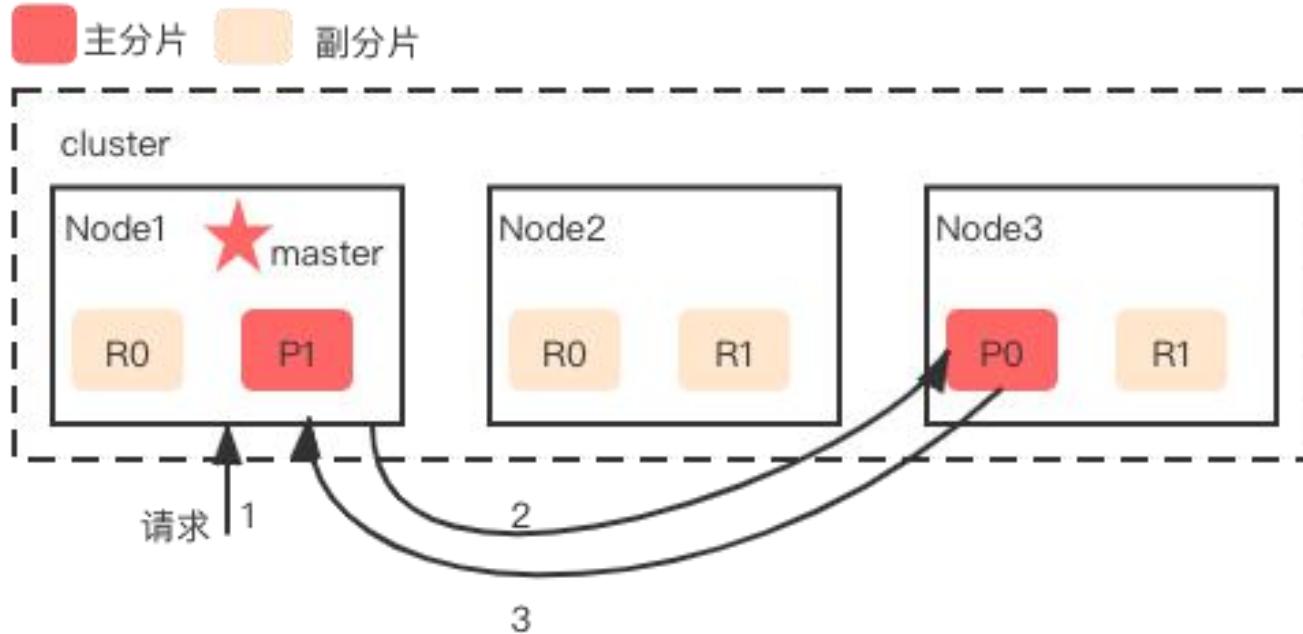
org.elasticsearch.cluster.routing.OperationRouting.java
calculateScaledShardId方法, 计算文档应该落在哪一个分片³⁰

hash = Murmur3HashFunction.hash(effectiveRouting) +
partitionOffset;

Math.floorMod(hash, indexMetaData.getRoutingNumShards()) /
indexMetaData.getRoutingFactor()

this.routingFactor = routingNumShards / numberOfShards;
routingNumShards 默认等于numberOfShards

文档的详细写入过程, 可参考 [Elasticsearch写过程源码浅析](#)



关键点:

取回一个文档



文档操作实战与分析

- [实战指导文档](#)
- [实战操作台](#)

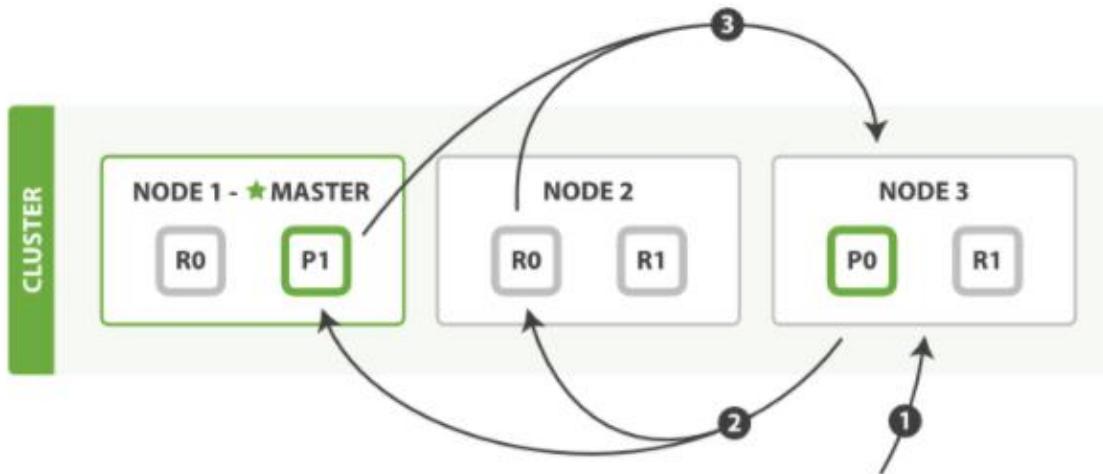
后退 我要装逼了



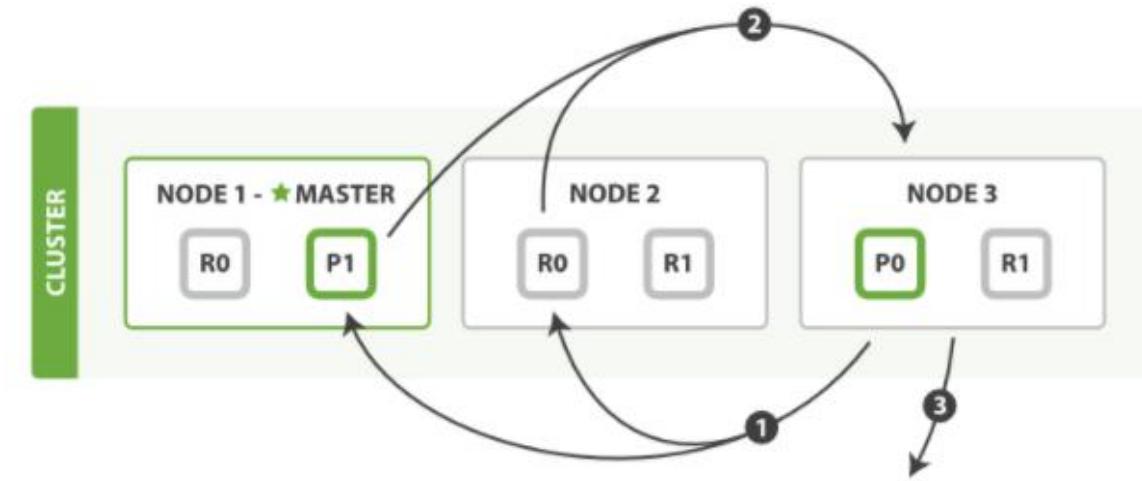


4 分布式搜索

分布式搜索查询阶段



分布式搜索的取回阶段



搜索类型：

`query_then_fetch`

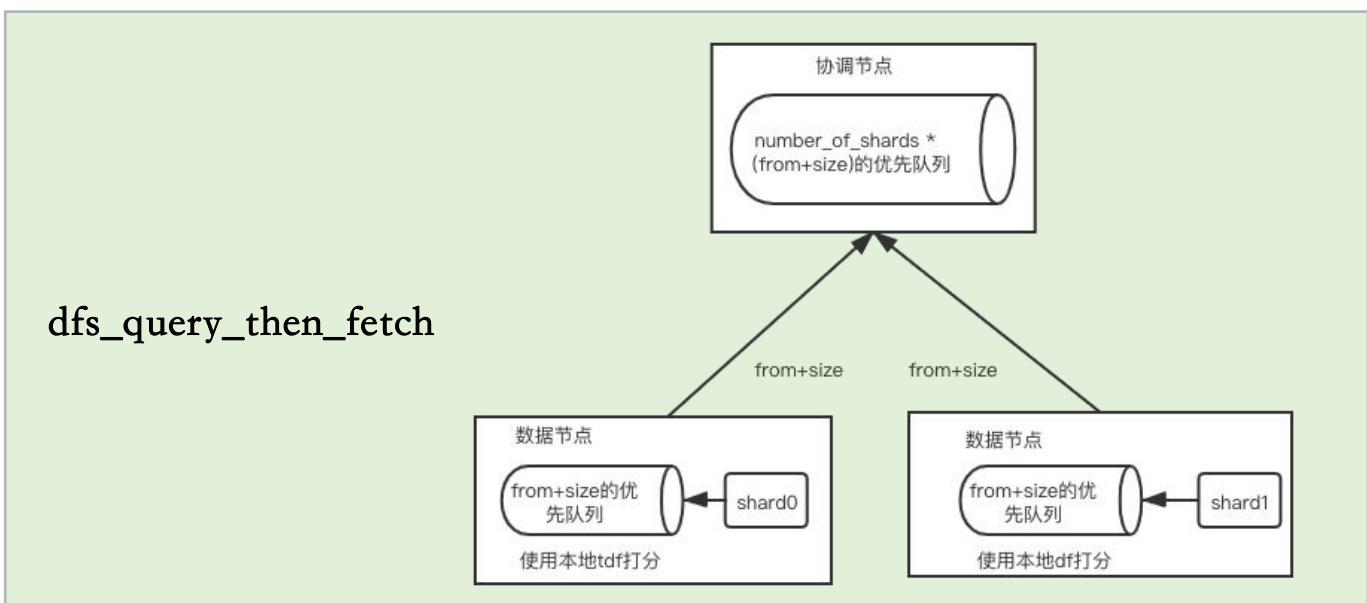
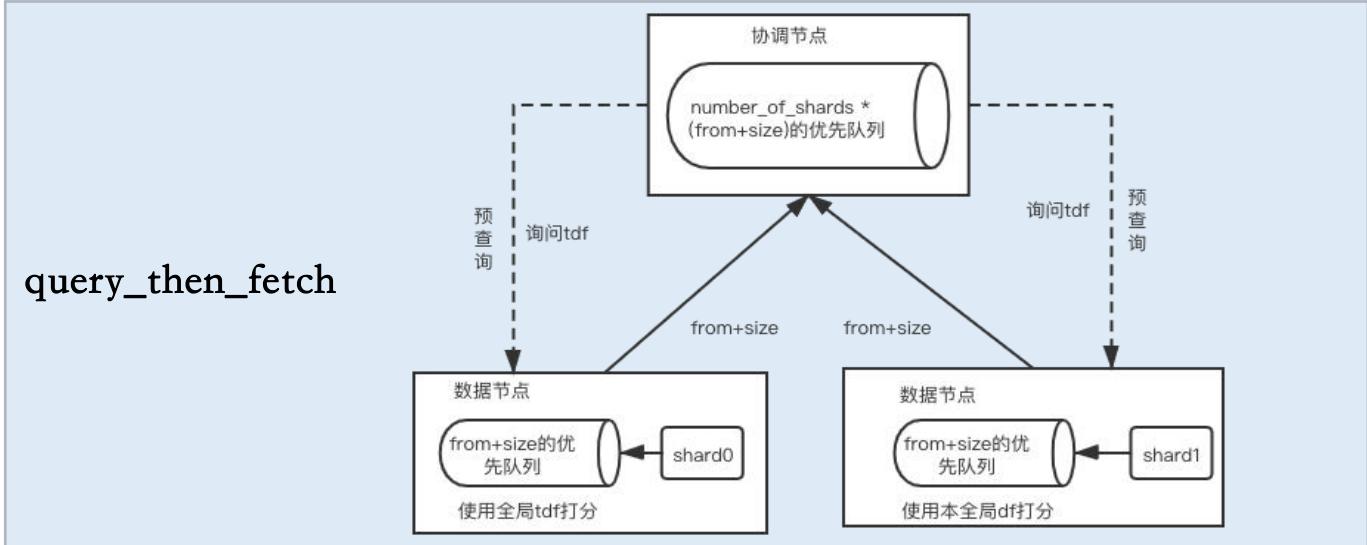
`dfs_query_then_fetch ;`

计算得分示例

```
GET my_store/products/2/_explain
{
  "query": {
    "match": {
      "productID": "KDKE"
    }
  }
}
```

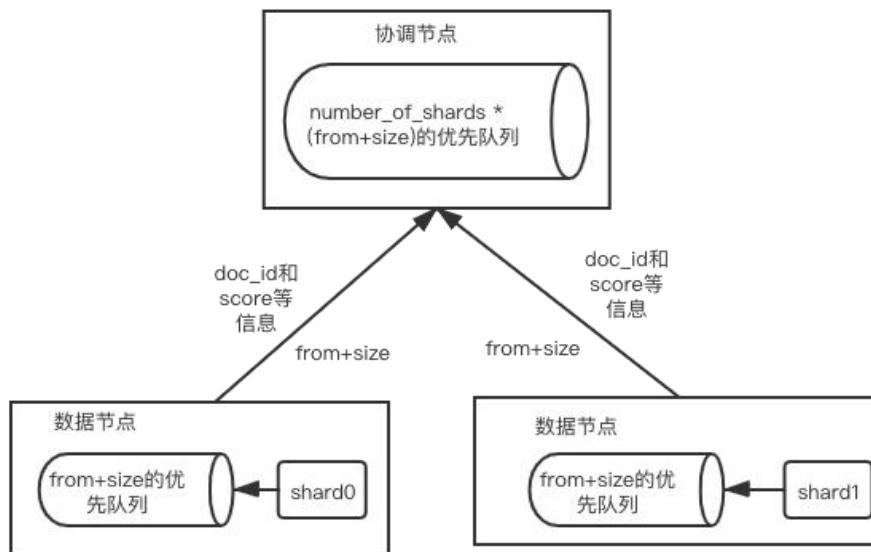
35

`org.elasticsearch.action.search.TransportSearchAction`
`searchAsyncAction方法`



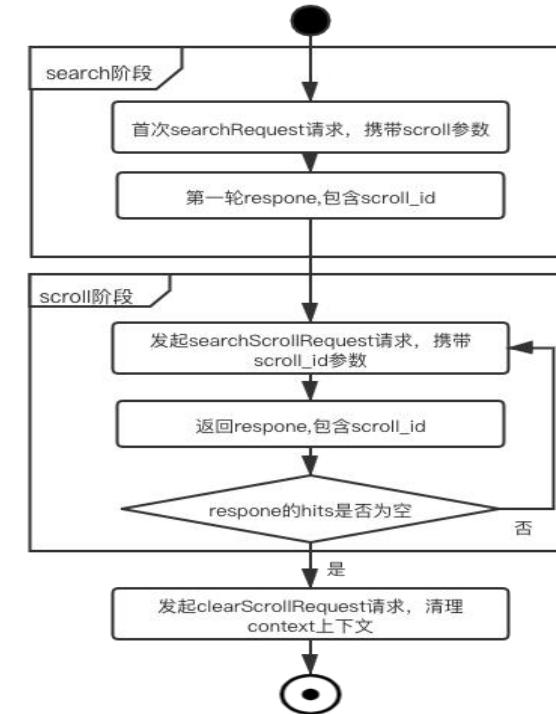
- 通过 from 、size 进行分页；不适合深分页；
- 通过 scroll 拉取大量数据；不支持跳页，不适合大量并发；
- 通过 search_after 拉取大量数据；不支持跳页。

深分页问题



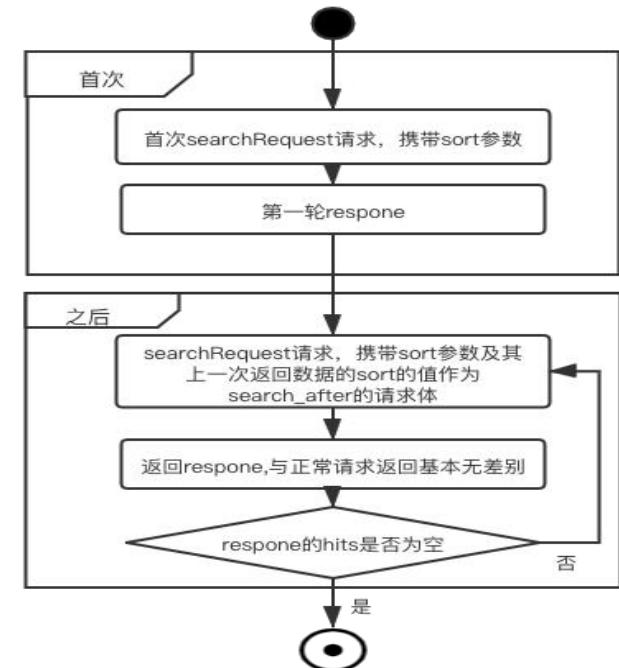
from+size的search查询阶段

快照数据，实时变更数据问题



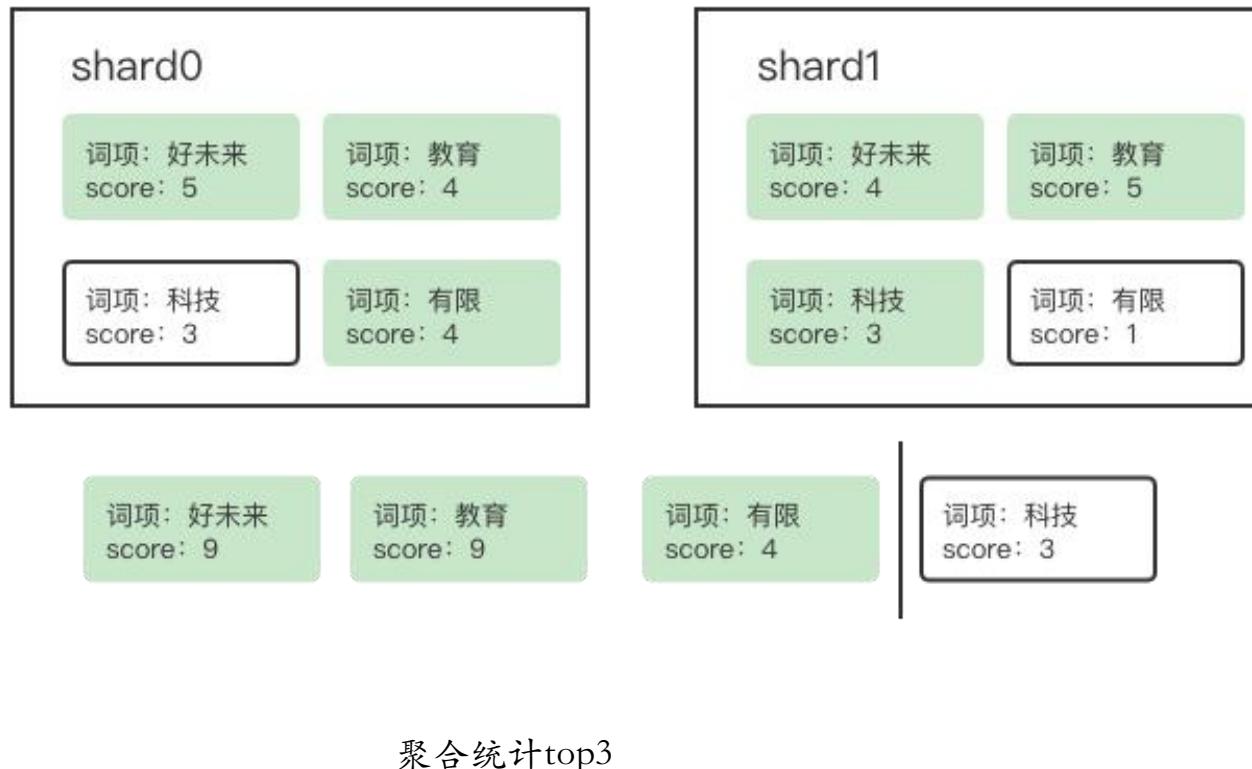
scroll查询

查询少数据问题



search_after查询

你的聚合查询准确吗？



关键点：

- **shard_size**

默认值 ($\text{size} * 1.5 + 10$)

- 提升准确性

GET elasticsearch_study/_search

```
{  
  "size": 0,  
  "aggs": {  
    "top_10_title": {  
      "terms": {  
        "field": "title.keyword",  
        "size": 10,  
        "shard_size": 100  
      }  
    }  
  }  
}
```

注意：shard_size 不能小于 size；也不能过大，增加资源的开销

[轻量搜索](#)



[过滤统计](#)

[精确值查询](#)

[组合过滤器](#)

资料推荐与技术交流

阿里云 Elasticsearch 技术实践

[《Elasticsearch官方最新文档》](#)

[《Elasticsearch: 权威指南》](#)

[《kibana使用手册官方中文文档》](#)

[《Elastic Stack 实战手册》](#)



分布式存储技术交流



知音楼扫描二维码，加入群组



好未来技术通道
TAL TECHNICAL CHANNEL

感谢聆听