Elastic通识

Elasticsearch是什么

lucene 和 elasticsearch 的关系

为什么不直接使用Luncene开发?

核心概念

- Node - 节点
- Cluster - 集群
- Shards - 分片
- Replicas - 副本 备份
- Index - 索引
- Mapping - 映射
- Settings - 配置
- Analyzer - 分词
- Type - 类型 （6.2中只能包含1个type ,7移除了 type的概念）
  - 为什么要移除Type呢?
- Document - 文档

Elasticsearch的架构设计

- 选主
- 分片和路由
- 数据写入过程
  1. 分段存储
  2. 延迟写策略
     - 基本方式
  3. 段合并
- WAL技术
- 主副分片数据一致性
- 自动管理索引生命周期（ILM）

Elasticsearch是什么

- Elasticsearch 是一个**近实时**的**分布式存储、搜索、分析**的引擎。

lucene 和 elasticsearch 的关系

- luncene是基于JAVA开发的搜索引擎类库。
- Elasticsearch是基于 lucene开发，隐藏复杂性。

为什么不直接使用Luncene开发?

1. luncene只能基于Java进行开发，可以使用更多语言进行开发
2. Api比较复杂，实现成本高, ES的API相对简单
3. 原生不支持水平扩展,ES实现了服务和数据两个维度 的高可用

核心概念

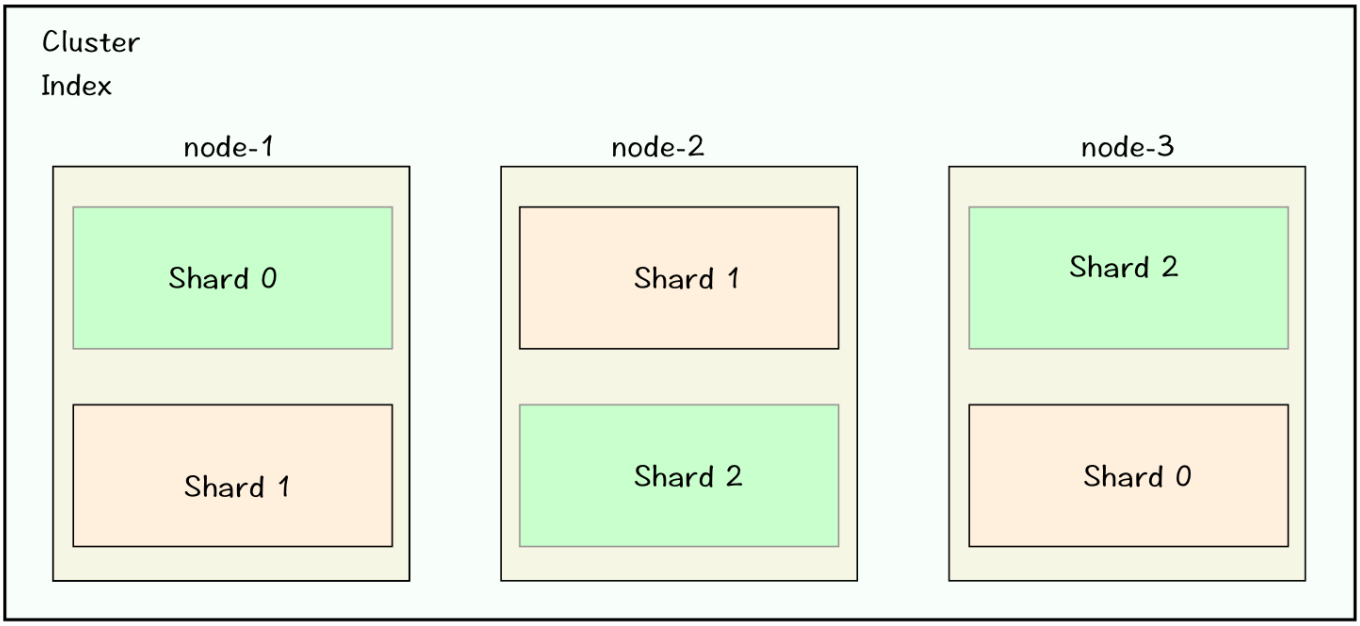
Node – 节点

- 即节点。节点是组成Elasticsearch集群的基本服务单元，集群中的每个运行中的Elasticsearch服务器都可称之为节点。
- 每个节点默认都可以参加master选举，可以禁止（比如配置比较低的机器）
- 每个节点都保存了集群的状态，但是只有master才可以修改
    - 集群状态：
        - 1. 所有节点的信息。
        - 2.所有的索引以及其相关的mapping和setting的信息
        - 3. 分片的路由信息
- 节点的类型：
    - data :数据节点，可以存储数据
    - coordinating : 协调节点，接受客户端请求，每个节点默认都可以作为协调节点
    - Ingest: 预处理节点
    - hot/warm: 冷热节点

Cluster – 集群

- 集群。Elasticsearch的集群是由具有相同cluster.name （默认值为elasticsearch）的一个或多个Elasticsearch节点组成的，各个节点协同工作，共享数据。同一个集群内节点的名字不能重复，但集群名称一定要相同。
- 在实际使用Elasticsearch集群时一般使用自定义集群名称，可以防止错误加入集群
- 集群状态：

- 绿色 表示节点运行状态为健康状态。所有的主分片和副本分片都可以正常工作，集群100%健康。
- 黄色 表示节点的运行状态为预警状态。所有的主分片都可以正常工作，但至少有一个副本分片是不能正常工作的。此时集群依然可以正常工作，但集群的高可用性在某种程度上被弱化。
- 红色 表示集群无法正常使用。此时，集群中至少有一个分片的主分片及它的全部副本分片都不可正常工作。虽然集群的查询操作还可以进行，但是也只能返回部分数据（其他正常分片的数据可以返回），而分配到这个有问题分片上的写入请求将会报错，最终导致数据丢失。



Shards – 分片

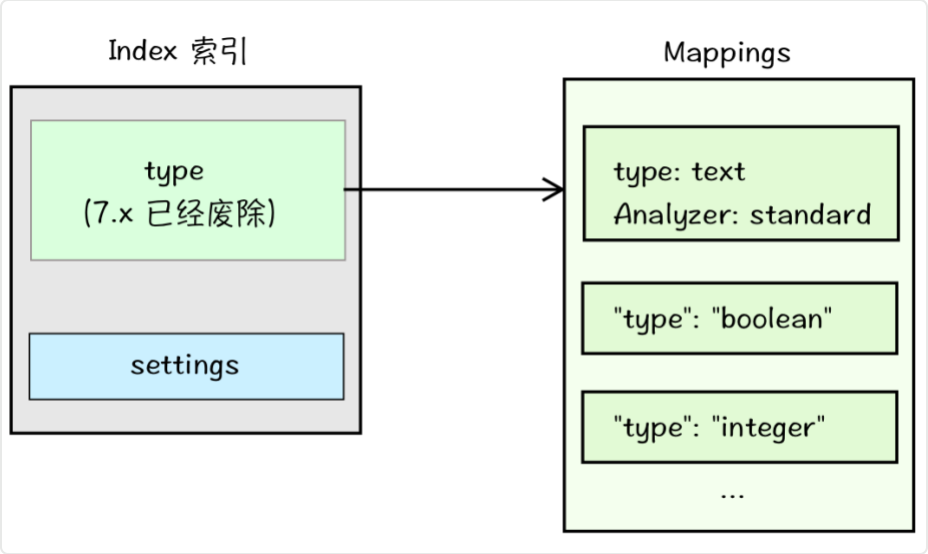
- 对于同一个索引进行水平拆分，拆分后的每一个数据部分被称为一个分片，一般来说，会将每个分片放入不同的节点上
- 感觉和mysql的分表有点像，不同部分的数据存放到不数据集合当中，这样的话，如果数量发生变化，自然需要把数据重新移动，所以es不允许修改分片数
- 索引创建的时候指定分片数量，分片的数量一旦确定就不能更改。
- 写入数据的时候，是通过路由来确定具体写入哪个分片中的
- 在Elasticsearch中，默认为一个索引创建5个主分片，并分别为每个主分片创建一个副本
- 一个分片就是一个 lucene 索引
- 分片越多搜索越慢

Replicas – 副本 备份

- 副本是对主分片的备份，每个主分片可以有零个或多个副本，主分片和备份分片都可以对外提供数

据查询服务

- 写入： 首先在主分片上完成数据的索引，然后数据会从主分片分发到备份分片上进行索引。
- 主分片不可用时，Elasticsearch会在备份分片中选举出一个分片作为主分片，从而避免数据丢失。
- 优点：增加集群可用性 ，缺点：如果分片过多，写操作时会增加数据同步的负担



Index – 索引

- 在Elasticsearch中，索引由一个和多个分片组成。在使用索引时，需要通过索引名称在集群内进行唯一标识。

Mapping – 映射

- Mapping表示中保存了定义索引中字段（Field）的存储类型、分词方式、是否存储等信息，有点类似于关系数据库（如MySQL）中的表结构信息。
- 一个索引的Mapping一旦创建，若已经存储了数据，就不可修改了，但是我们可以增加一个新的Mapping

Settings – 配置

- Settings是对集群中索引的定义信息，比如一个索引默认的分片数、副本数等。

Analyzer – 分词

- Analyzer表示的是字段分词方式的定义

~~Type – 类型~~（6.2中只能包含1个type ,7移除了 type的概念）

- type就是在一个索引可以创建多个Mapping，在老的版本有把索引比喻成数据库，type比喻成表的

说法。

为什么要移除Type呢?

- 不同类型的“记录”存储在同一个index中，会影响lucene的压缩性能，简单说，影响效率

Document – 文档

- 索引中的每一条数据叫作一个文档，与关系数据库的使用方法类似，一条文档数据通过_id在Type内进行唯一标识。

index table
mapping 字段
document 一行记录

Elasticsearch的架构设计

选主

- 在集群中配置一个相同的集群名称（即cluster.name），就能将不同的节点连接到同一个集群
- 它会对所有可以成为master的节点（node.master: true）根据节点Id的字典排序，取第一个，暂且认为它是master节点。如果对某个节点的投票数达到一定的值（可以成为master节点数n/2+1 discovery.zen.minimum_master_nodes ）并且该节点自己也选举自己，那这个节点就是master。否则重新选举一直到满足上述条件。
- 如果只有一个本地节点，则主节点就是它自己。
- **ES是如何避免脑裂现象的：**可以通过discovery.zen.minimum_master_nodes这个参数的设置来避免脑裂，设置为(N/2)+1。
- **ES的集群只有一个节点的话可以有副本吗?** Elasticsearch 禁止同一个分片的主分片和副本分片在同一个节点上，所以如果是一个节点的集群是不能有副本的。其实在一个节点也就没有了副本的意义

分片和路由

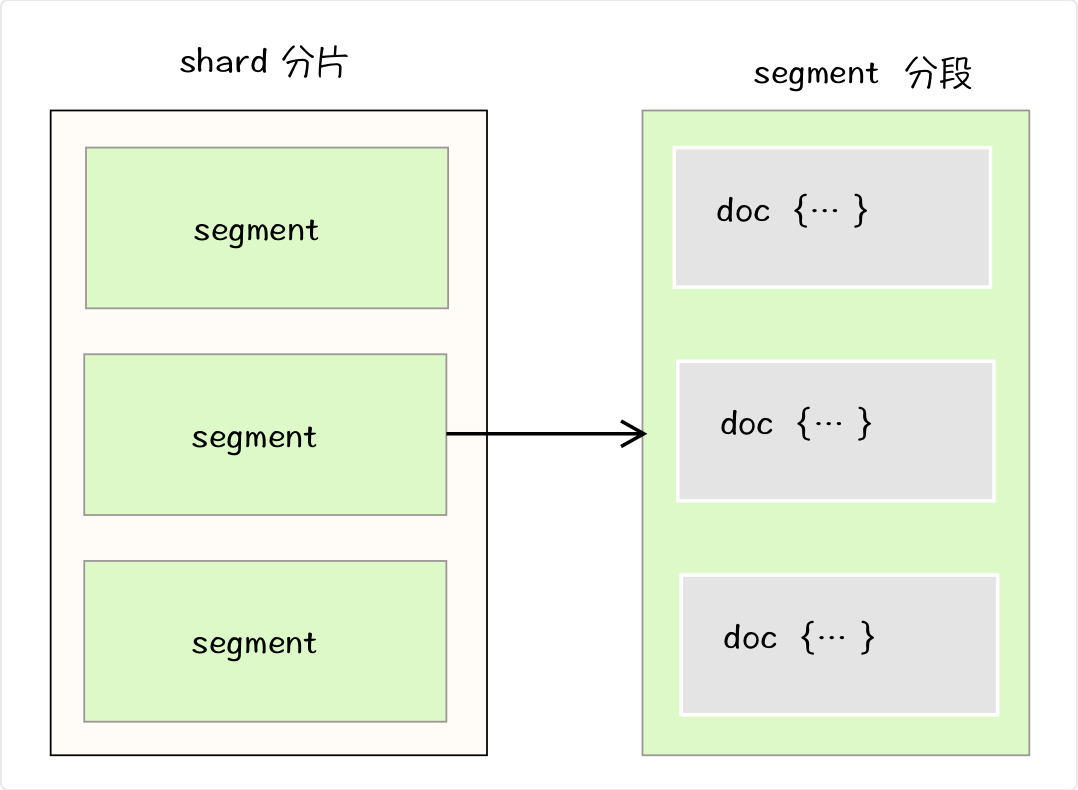
- 分片一旦确定，不能修改
- routing字段的取值默认是id字段，采用routing字段在Hash后之后再与有分片的数量取模。这个值可以更改
- shard_num = hash(_routing) % num_primary_shards
- 在做数据检索时，Elasticsearch默认会搜索所有分片上的数据，最后在主节点上汇总各个分片数据并进行排序处理后，返回最终的结果数据

- 通过路由的方式，还可以实现冷热数据架构
- 路由官方文档

数据写入过程

1. 分段存储

- 每个shard（分片）包含多个segment（段），每一个segment都会存放索引的部分文档。是Lucene分割的最小单位.分段数据一经创建，便不会被修改
- 新增数据：新增一个 数据段
- 删除数据：增加.del文件，被标记的可以查询到，但是在返回的时候会被移除
- 修改数据：先删除，然后新增新的数据段
- 优势： 不需要锁，从而提升Elasticsearch的读写性能，
- 缺点： 1. 分段越多 搜索越慢，因为需要过滤删除的数据 2. 在删除和更新数据时，存储空间会浪费



2. 延迟写策略

索引写入磁盘的过程是异步的

基本方式

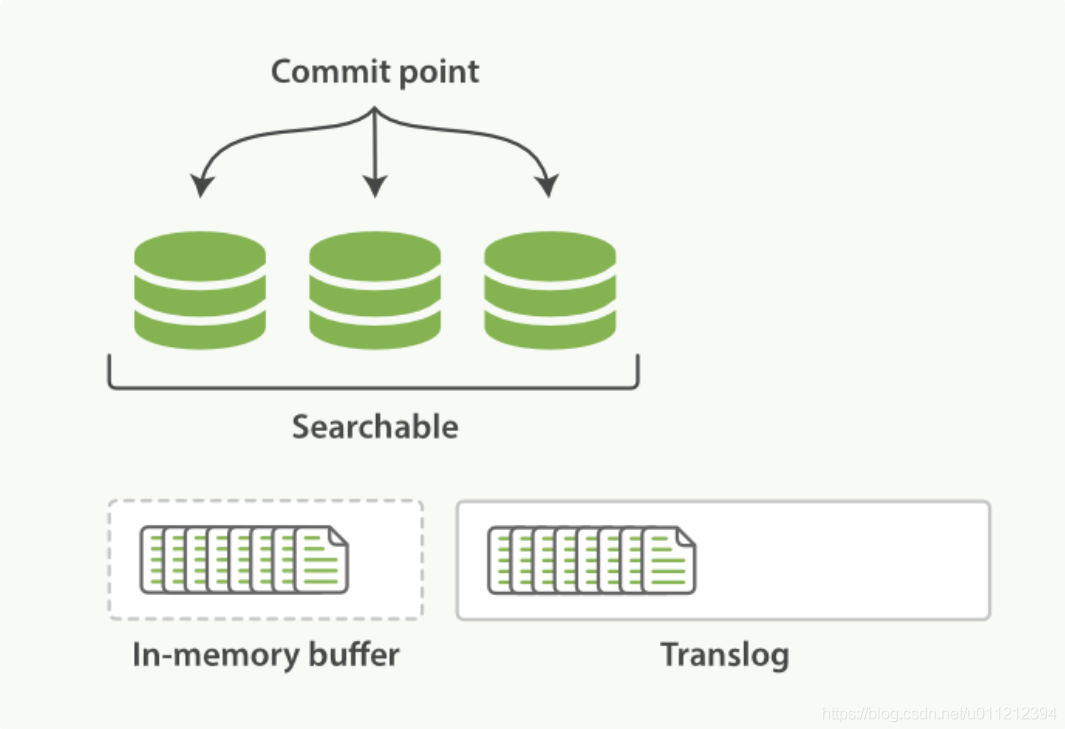
- 每当有新的数据产生的时候，会将数据先写入内存当中，当数据到达一定程度的时候，或者到达刷新时间，会触发一次刷新（Refresh）操作。刷新操作将内存中的数据生成到一个新的分段上并缓存

到文件缓存系统，稍后再被刷新到磁盘中并生成提交点。

- 当数据写入内存的时候，内存的数据并不是用段来存储的，因此不能提供检索功能。只有当数据由内存刷新到文件缓存系统的时候，并生成一个新的段的时候，才能被搜索到，而不用等刷新到磁盘
- 在es中，写入和打开一个新的段，叫做刷新。在默认情况下，每个分片会每秒自动刷新一次。这就是Elasticsearch能做到近实时搜索的原因，因为文档的变化并不是立即对搜索可见的，但会在一秒之内变为可见。也可以手动刷新。
- 可以通过 refresh_interval 来调整刷新频率， 当为-1的时候，表示关闭自动刷新

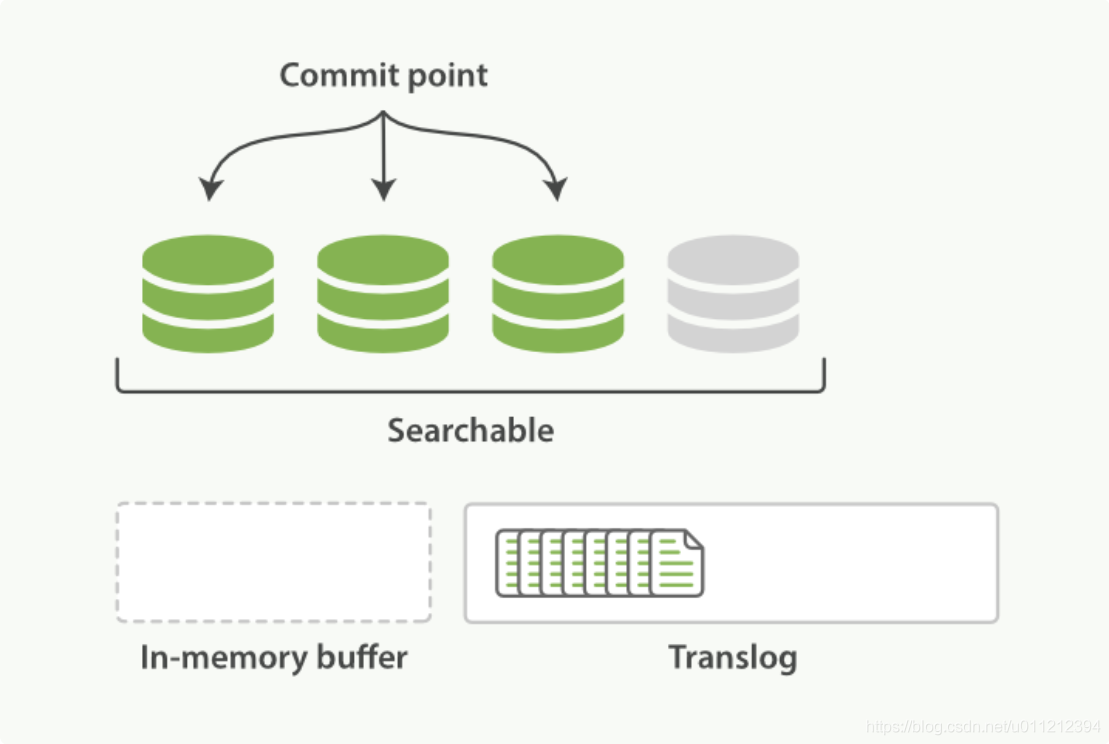
- 因为文件系统缓存有数据丢失的风险，所以引入了 Translog （事务日志）,所以最终的写入过程是这样的
  a. 新文档被索引之后，先被写入内存中。为了防止数据丢失，Elasticsearch会追加一份数据到事务日志中。此时的新数据还不能被检索和查询。
  b. 当达到默认的刷新时间或内存中的数据达到一定量后，Elasticsearch会触发一次刷新，将内存中的数据以一个新段形式刷新到文件缓存系统中并清空内存。这时新段虽未被提交到磁盘，但已经可以对外提供文档的检索功能且不被修改。
  c. 随着新文档索引不断被写入，当日志数据大小超过某个值（如512MB），或者超过一定时间（如30 min）时，Elasticsearch会触发一次Flush，提交到硬盘。
  d. 生成提交点。日志文件被删除，创建一个空的新日志。

1. 新文档先被放入内存中，此时数据不能被检索

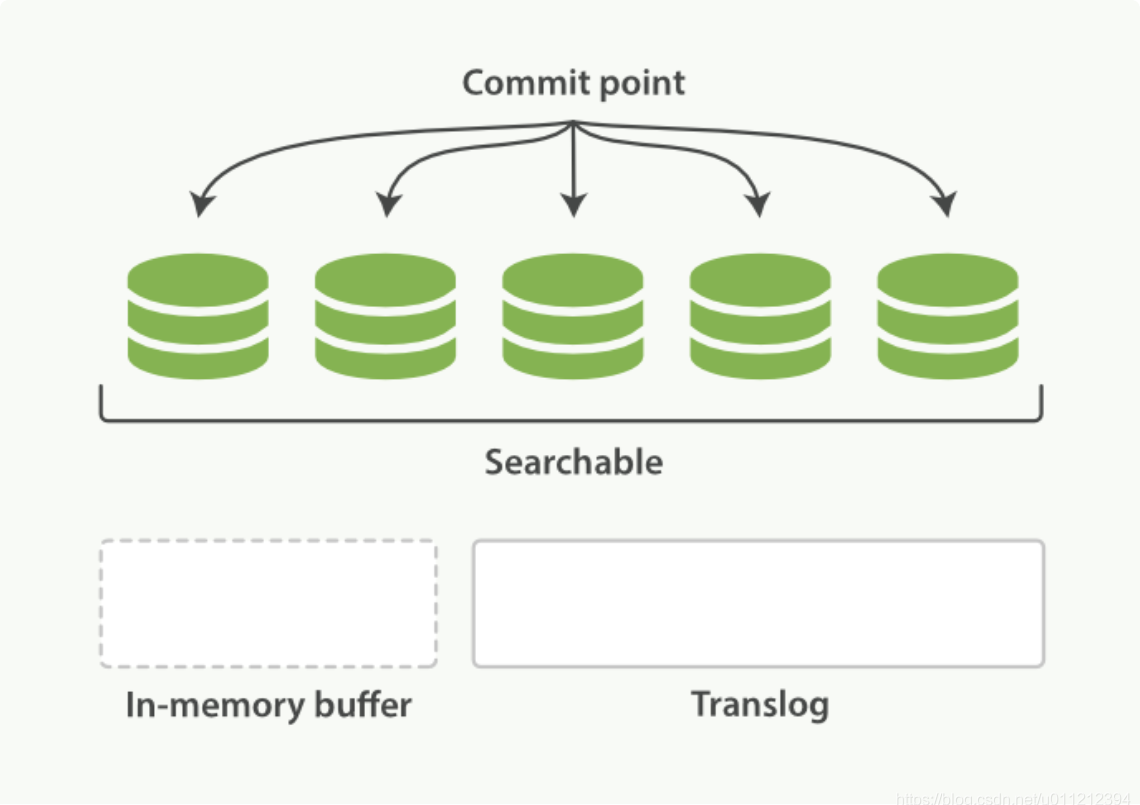


2. 默认刷新时间（1秒钟）到达，或者数据到达一定量的时候，会把内存中的数据创建一个新的段，此

时数据可以被检索，但是此时数据还在内存。



3. translog太大，或者默认*flush时间*（30分钟），进行*flush*。*段*数据被提交到硬盘当中，并且 translog 被清空



3. 段合并

- 因为分段的特点，es在增写删所以必然会产生很多的分段，对资源的消耗会剧增。而且这样的分段在查询出来数据之后，还需要与del，进行合并，这样的话，段越多，搜索越慢。所以就需要进行段合并。
- 段合并机制在后台定期进行，从而小的段被合并到大的段，然后这些大的段再被合并到更大的段。
- 段合并过程中，Elasticsearch会将那些旧的已删除文档从文件系统中清除。被删除的文档不会被拷贝到新的大段中，当然，在合并的过程中不会中断索引和搜索。
- 合并结束后，老的段会被删除，新的段被Flush到磁盘，同时写入一个包含新段且排除旧的和较小的段的新提交点。打开新的段之后，可以用来搜索。
- 默认情况下会对合并流程进行资源限制，防止影响搜索性能。
- 官方文档

WAL技术

此处为语雀内容卡片，点击链接查看：https://www.yuque.com/go/doc/42404465?view=doc_embed

主副分片数据一致性

- 同步：主分片处理完毕，然后同步进行数据同步到所有副本
- 异步：主分片处理完毕直接返回，副本 异步同步，会造成服务器压力过大

自动管理索引生命周期（ILM）

- 把一个索引的生命周期定义为4个部分

  - Hot：索引可写入，也可查询。
  - Warm：索引不可写入，但可查询。
  - Cold：索引不可写入，但很少被查询，查询的慢点也可接受。
  - Delete：索引可被安全的删除
  - 文档
- 策略：
  - rollover：滚动存储，建立新索引，数据新增引入新索引
  - delete： 自动删除

    - shrink: 缩减分片，读的时候分片少可以减少内存消耗，写的时候分片多可以提速
    - readonly: 设置为只读
    - focemerge : 强行合并分段到一个很小的规模
    - freeze: 把索引关闭，不占用内存，只占用磁盘，不可以检索
    - allocate: 分片感知，可以用来冷热分离
1. 完整的策略配置：完整策略

JSON | 复制代码

```
PUT _ilm/policy/full_policy
{
  "policy": {
    "phases": {
      "hot": {
        "actions": {
          "rollover": {
            "max_age": "7d",
            "max_size": "50G"
          }//当数量到达500g，或者存储时间超过7天的时候，就会进行滚动存储
        }
      },
      "warm": {
        "min_age": "30d",//进入这个阶段的条件
        "actions": {
          "forcemerge": {
            "max_num_segments": 1 //强行合并分段
          },
          "shrink": {
            "number_of_shards": 1 //分片缩减到1个
          },
          "allocate": {
            "number_of_replicas": 2 //增加两个副本
          }
        }
      },
      "cold": {
        "min_age": "60d",//进入冷阶段条件
        "actions": {
          "allocate": {
            "require": {//将索引分配给 冷数据节点
              "type": "cold"
            }
          }
        }
      },
      "delete": {
        "min_age": "90d",//进行删除的条件
        "actions": {
          "delete": {}//删除索引
        }
      }
    }
  }
}
```