Lecture 20 ElasticSearch
分布式搜索引擎

- ElasticSearch特点
- ES计算模型
- ES计算架构

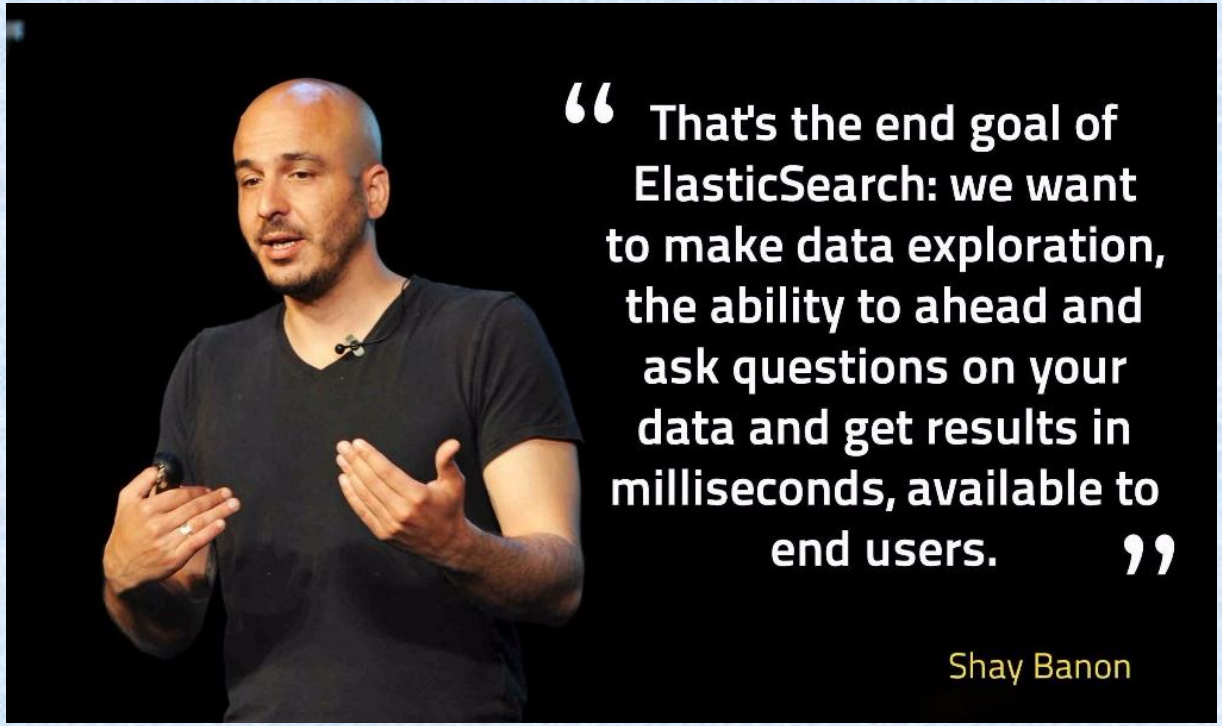
ElasticSearch

ElasticSearch（简称ES）是一个基于Lucene构建的分布式、低时延、高扩展性的开源分布式搜索引擎，也是一个分布式文档数据库，这一分布式搜索引擎具备实时分析功能，并能够方便地扩展至数以百计的服务器存储及处理PB级数据。

ES分布式数据库具有如下特点：

- 分布式文件存储，全文检索，即每一个字段都编入索引表检索
- 实时分布式搜索引擎
- 可以扩展到上百台服务器处理PB级别的结构化或非结构化数据

ElasticSearch创始人Shay Banon在2010年2月发布ElasticSearch第一个版本。2012年Elasticsearch BV成立，2015年3月ElasticSearch公司更名为Elastic，2018年6月Elastic提交了首次公开募股申请，估值在15亿到30亿美元之间。公司于2018年10月5日在纽约证券交易所挂牌上市。



ElasticSearch 资源

官网 https://www.elastic.co

文档 https://www.elastic.co/guide/index.html

GitHub https://github.com/elastic/elasticsearch

ELK 开发框架

ELK = ElasticSearch + Logstash + Kibana

ElasticSearch：后台分布式存储以及全文检索
Logstash：日志加工、“搬运工”
Kibana：数据可视化展示

ElasticSearch 核心技术

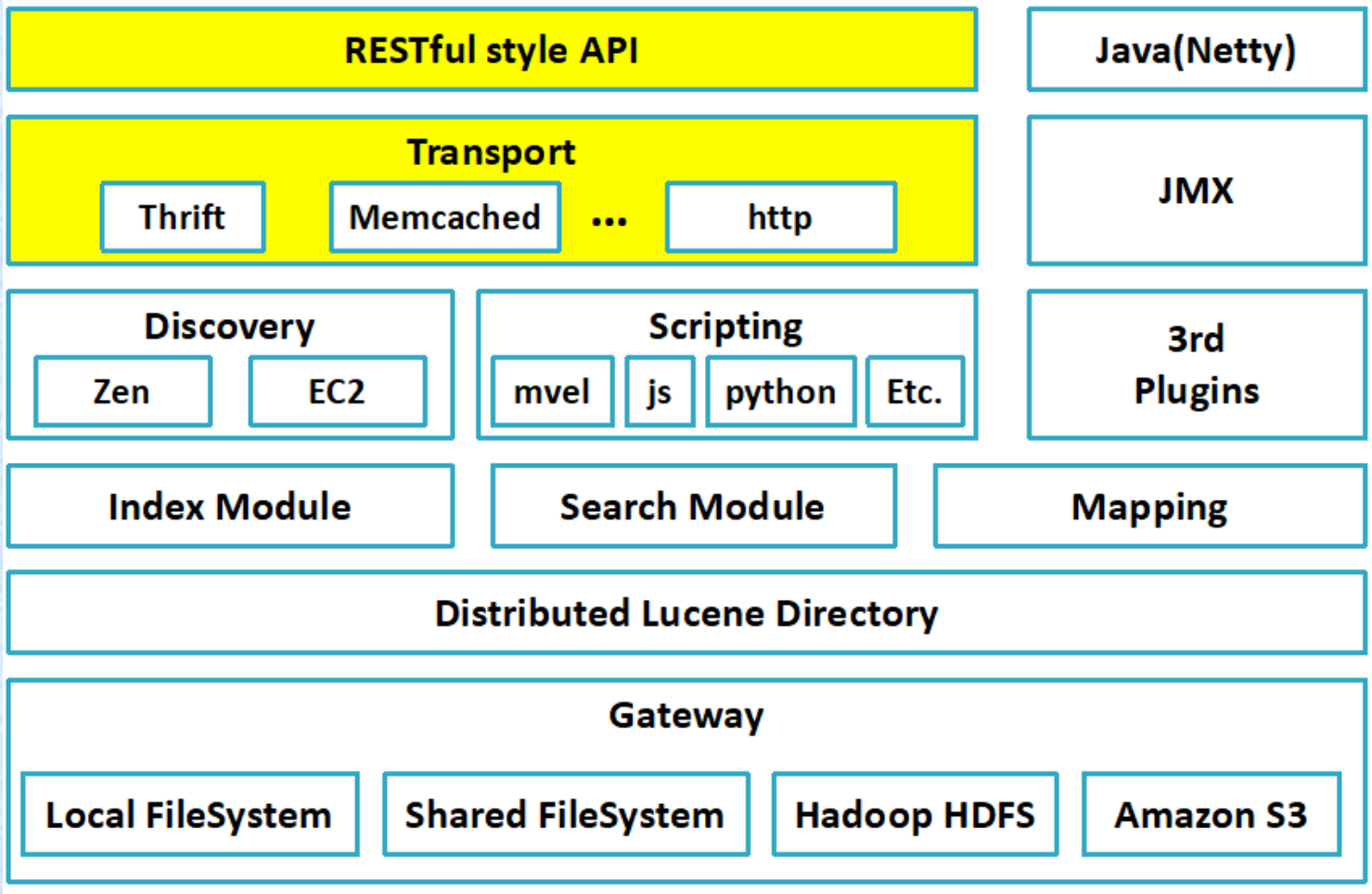
- **分布式文档数据库**，便于存储结构化和非结构化数据
- **倒排索引算法**，支持近实时（near real-time）快速查询
- **内存高效索引表压缩存储**，避免磁盘读写，高速查询
- **极好的水平扩展性**
- Restful格式数据库访问接口

ElasticSearch 计算模型

ES计算模型分为两层

- **ES逻辑架构：**索引(index)、类型(type)、文档(document)、分词(term)、字段(field)、倒排索引(inverted index)、倒排列表(posting list)、索引表内存压缩(term index hashed in memory)、FST(finite state transducers) ……
- **ES物理架构：**ES集群节点(node)、主节点选举(master)、分片(shard)、副本(replica)、路由算法(routing)、索引(write doc)、查询(query phase)、 获取(fetch phase) 、容错机制、……

ElasticSearch 软件架构



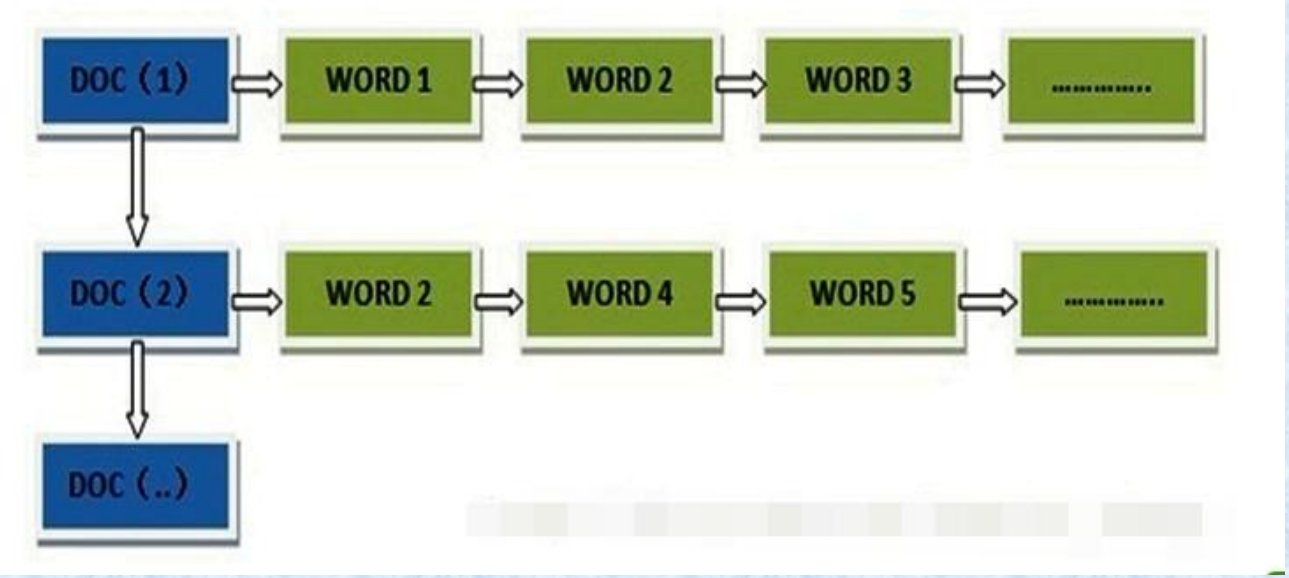
Gateway	ElasticSearch索引的持久化存储方式。默认先把索引存储在内存中，然后当内存满的时候，再持久化到Gateway（如LocalFileSystem和HDFS、AS3等）
Distributed Lucene Directory	Lucene索引文件组成的目录
River	插件形式的数据源
Mapping	类似于静态语言中的数据类型，
Search Module	这个很简单
Index Module	这个很简单
Discovery	主要是负责集群的master节点发现及分片机制（Zen, EC2）
Scripting	即脚本语言。包括mvel、js、python等
Transport	ElasticSearch节点与客户端的交互，包括Thrift、Memcached、Http
RESTful API	以RESTful方式来实现API编程
3rd plugins	代表第三方插件
Java(Netty)	是开发框架
JMX	系统监控

ElasticSearch 逻辑架构

正向索引（forward index）

正向索引结构如下：

“文档1”的ID > 单词1：出现次数，出现位置列表；单词2：出现次数，出现位置列表；……

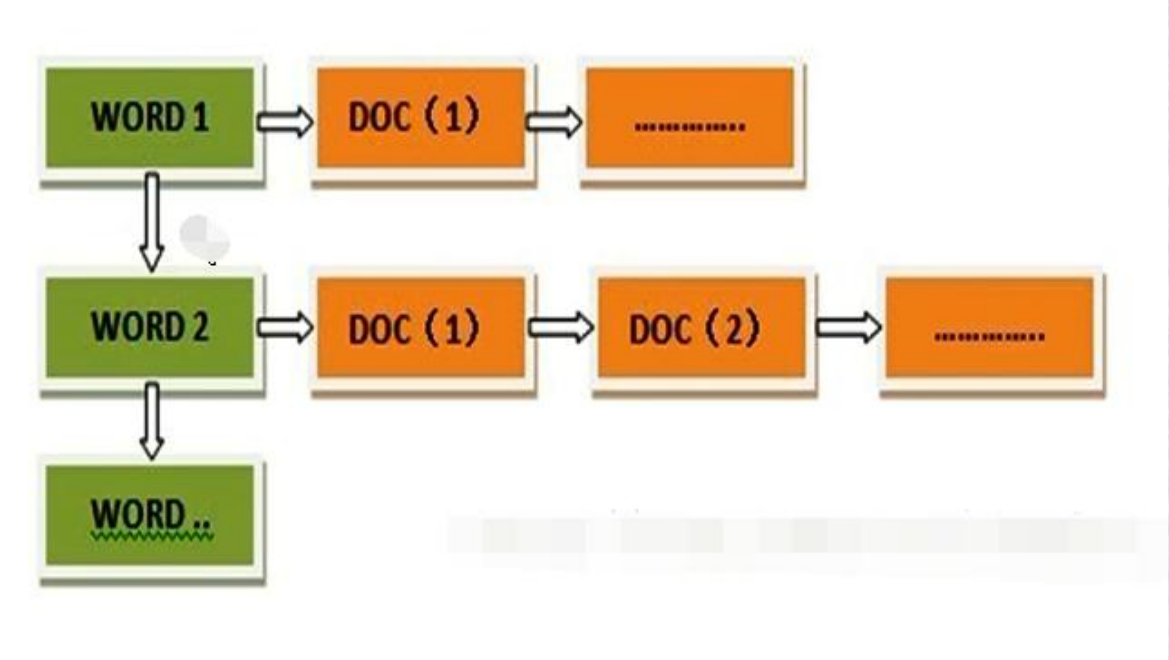


反向索引（inverted index）

转换为**关键词到文件ID的映射**，每个关键词都对应着一系列的文件，这些文件中都出现这个关键词。

反向索引（倒排索引）结构如下：

“关键词1”>“文档1”的ID，“文档2”的ID，……



单词—文档矩阵

单词-文档矩阵					
	文档1	文档2	文档3	文档4	文档5
词汇1	✓			✓	
词汇2		✓	✓		
词汇3				✓	
词汇4	✓				✓
词汇5		✓			
词汇6			✓		

搜索引擎的索引其实就是实现“单词-文档矩阵”的具体数据结构。

倒排索引概念

文档(Document)：以文本形式存在的存储对象，涵盖更多种形式，比如Word，PDF，html，XML等不同格式的文件都可以称之为文档。再比如一封邮件，一条短信，一条微博也可以称之为文档。

文档集合(Document Collection)：由若干文档构成的集合称之为文档集合。

文档编号(Document ID)：文档集合内每个文档赋予一个唯一的编号。

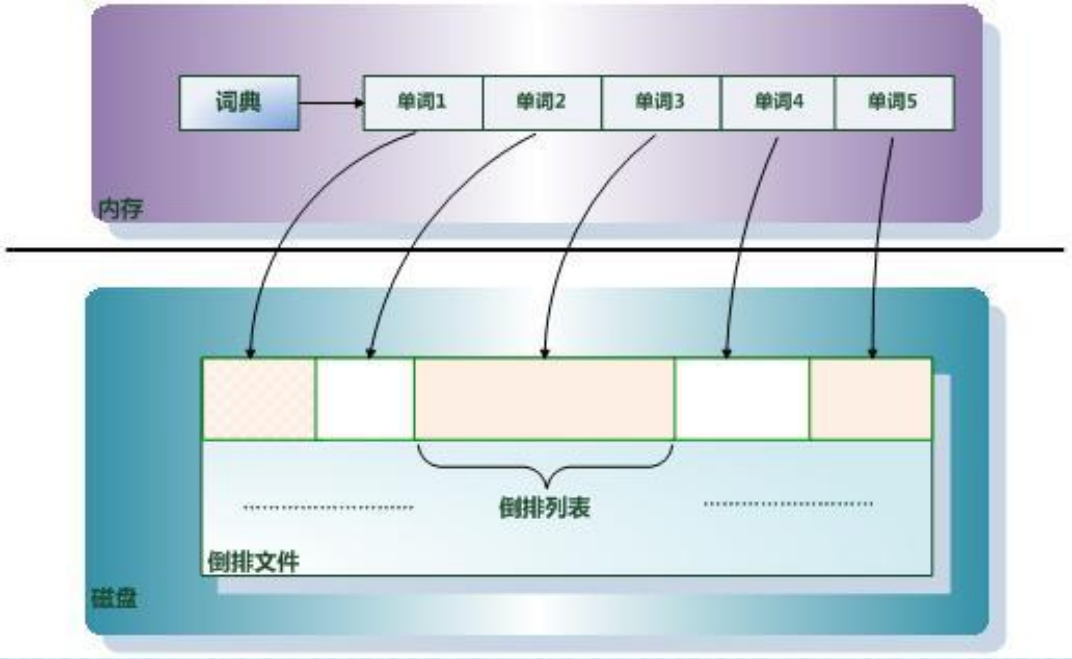
单词编号(Word ID)：以唯一的编号来表征某个单词。

倒排索引主要由两个部分组成：“单词词典”和“倒排文件”

单词词典(Lexicon)：搜索引擎的通常索引单位是单词，单词词典是由文档集合中出现过的所有单词构成的字符串集合，单词词典内每条索引项记载单词本身的一些信息以及指向“倒排列表”的指针。

倒排列表(PostingList)：倒排列表记载了出现过某个单词的所有文档的文档列表及单词在该文档中出现的位置信息，每条记录称为一个倒排项(Posting)。根据倒排列表，即可获知哪些文档包含某个单词。

倒排文件(Inverted File)：所有单词的倒排列表往往顺序地存储在磁盘的某个文件里，这个文件即被称之为倒排文件，倒排文件是存储倒排索引表的物理文件。



倒排索引实例：5个文档


文档编号
文档内容
1
谷歌地图之父跳槽Facebook
2
谷歌地图之父加盟Facebook
3
谷歌地图创始人拉斯离开谷歌加盟Facebook
4
谷歌地图之父跳槽Facebook 与Wave项目取消有关
5
谷歌地图之父拉斯加盟社交网站Facebook

分词后构建的简单倒排索引

含词频及位置信息的倒排索引表

单词ID	单词	文档频率	倒排列表（DocID;TF;<POS>）
1	谷歌	5	(1;1;<1>), (2;1;<1>), (3;2;<1;6>), (4;1;<1>), (5;1;<1>)
2	地图	5	(1;1;<2>), (2;1;<2>), (3;1;<2>), (4;1;<2>), (5;1;<2>)
3	之父	4	<1;1;<3>), (2;1;<3>), (4;1;<3>), (5;1;<3>)
4	跳槽	2	(1;1;<4>), (4;1;<4>)
5	Facebook	5	(1;1;<5>), (2;1;<5>), (3;1;<8>), (4;1;<5>), (5;1;<8>)
6	加盟	3	(2;1;<4>), (3;1;<7>), (5;1;<5>)
7	创始人	1	(3;1;<3>)
8	拉斯	2	(3;1;<4>), (5;1;<4>)
9	离开	1	(3;1;<5>)
10	与	1	(4;1;<6>)

以单词“拉斯”为例，其单词编号为8，对应的倒排列表信息为：{(3;1;<4>)，(5;1;<4>)}, 表示为在文档3和文档5出现过这个单词，单词频率都为1，在两个文档中的出现位置都是4，即文档中第四个单词是“拉斯”。

单词词典

是倒排索引中非常重要的组成部分，用来维护文档集合中出现过的所有单词的相关信息，同时记载某个单词对应的倒排列表在倒排文件中的位置信息。在支持搜索时，根据用户的查询词，去单词词典里查询，就能够获得相应的倒排列表，并以此作为后续排序的基础。

对于一个规模很大的文档集合来说，可能包含几十万甚至上百万的不同单词，所以需要高效的数据结构来对单词词典进行构建和查找，常用的数据结构包括哈希加链表结构和树形词典结构。

哈希加链表

主体部分是哈希表，每个哈希表项保存一个指针，指针指向冲突链表，在冲突链表里，相同哈希值的单词形成链表结构。之所以会有冲突链表，是因为两个不同单词获得相同的哈希值，如果是这样，在哈希方法里被称做是一次冲突，可以将相同哈希值的单词存储在链表里，以供后续查找。

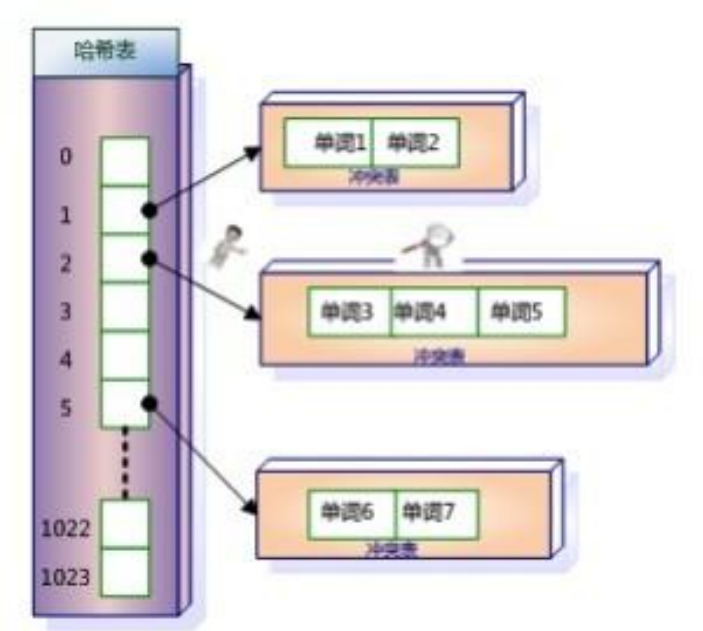


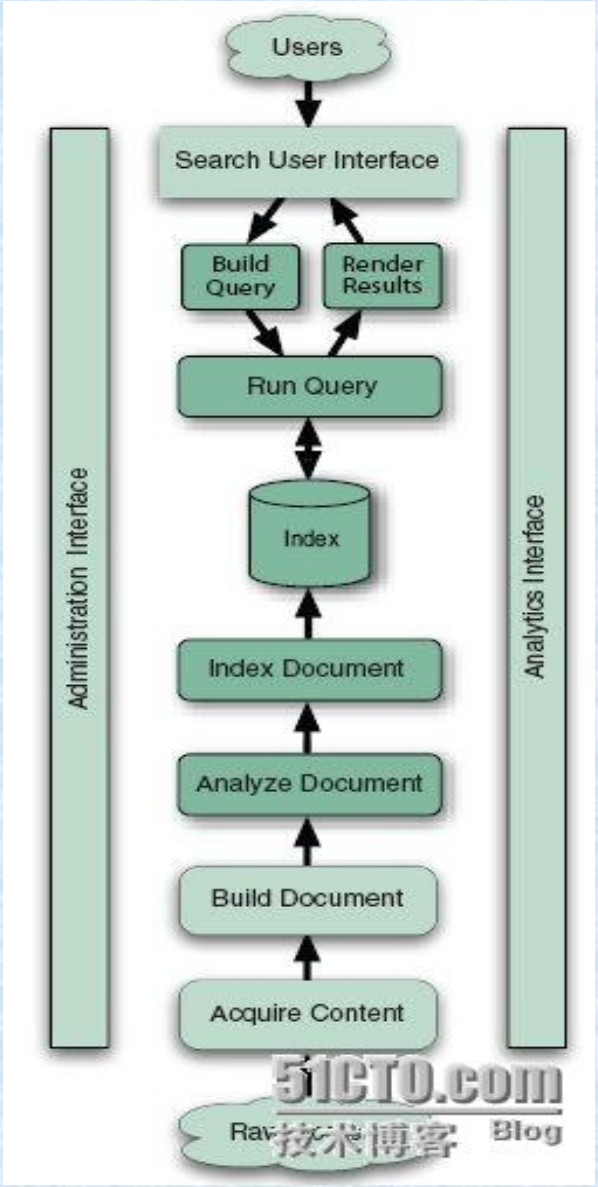
图 1-7 哈希加链表词典结构

Lucene搜索引擎原理

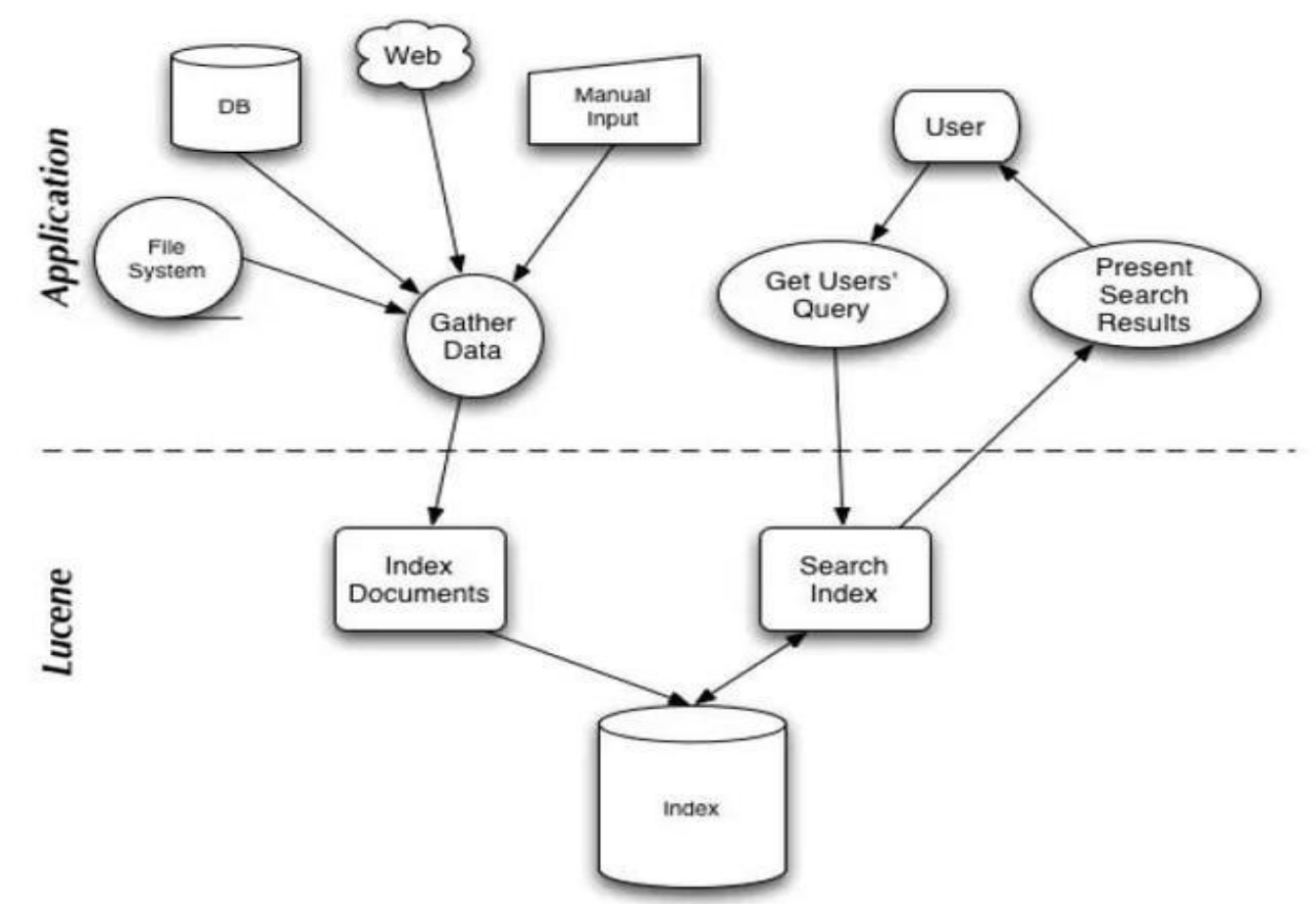
搜索程序一般由索引链及搜索组件组成。

索引链功能的实现需要按照几个独立的步骤依次完成：检索原始内容、根据原始内容来创建对应的文档、对创建的文档进行索引。

搜索组件用于接收用户的查询请求并返回相应结果，一般由用户接口、构建可编程查询语句的方法、查询语句执行引擎及结果展示组件组成。



Lucene索引和搜索过程



索引构建

索引是一种数据结构，它允许对存储在其中的单词进行快速随机访问。当需要从大量文本中快速检索文本目标时，必须首先将文本内容转换成能够进行快速搜索的格式，建立针对文本的索引数据结构，此即为索引过程。它通常由逻辑上互不相关的几个步骤组成。

第一步：获取内容。

通过网络爬虫程序等来搜集及界定需要索引的内容。Lucene并不提供爬虫组件，因此需要其它开源爬虫程序如Solr、Nutch、Grub及Aperture，获取到的内容建立为小数据块即文档Document。

第二步：建立文档。

获取的原始内容需要转换成文档（JSON格式）才能供搜索引擎使用。

一般来说，一个网页、一个PDF文档、一封邮件或一条日志信息可以作为一个文档。文档由带值(Value)的域(Field)组成，例如标题(Title)、正文(body)、摘要(abstract)、作者(Author)和链接(url)等。

索引构建（续）

第三步：文档分析。

搜索引擎不能直接对文本进行索引，确切地说，必须首先将文本分割成一系列被称为语汇单元(token)的独立原子元素，此过程即为文档分析。每个token大致能与自然语言中的“单词”对应起来，文档分析就是用于确定文档中的文本域如何分割成token序列，此即为切词，或分词。

文档分析中要解决的问题包括如何处理连接一体的各个单词、是否需要语法修正(例如原始内容存在错别字)、是否需要向原始token中插入同义词(例如laptop和notebook)、是否需要将大写字符统一转换为小写字符，以及是否将单数和复数格式的单词合并成同一个token等。这通常需要词干分析器等来完成此类工作，Lucene提供了大量内嵌的分析器，也支持用户自定义分析器，甚至联合Lucene的token工具和过滤器创建自定义的分析链。

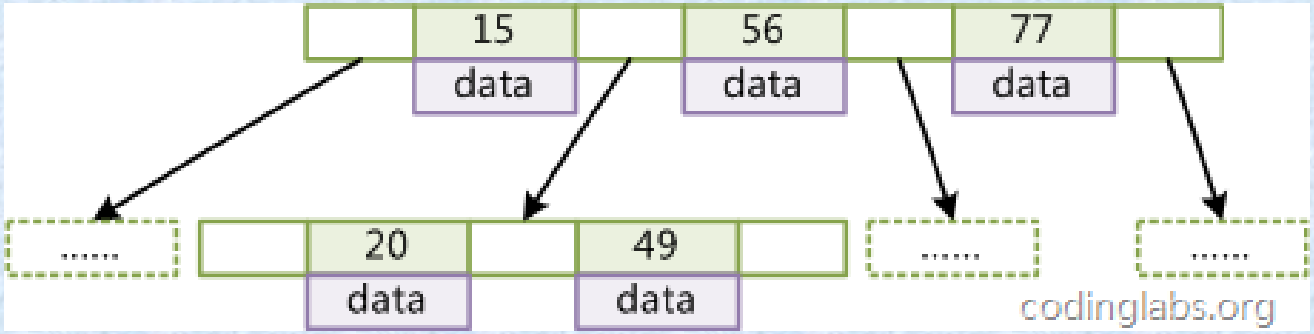
第四步：文档索引

在索引步骤中，文档将被加入到倒排索引表中。

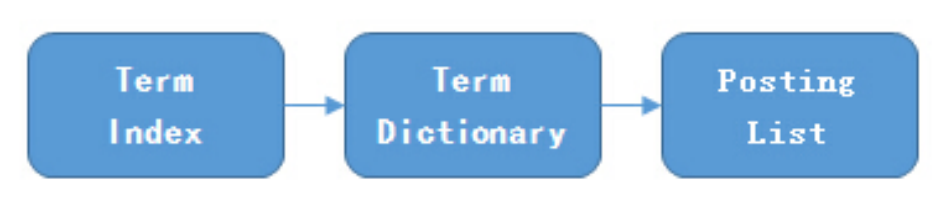
索引表内存压缩算法 – 内存计算

B-Tree / B+Tree结构

二叉树查找效率是logN，同时插入新节点不必移动全部节点，能同时兼顾插入和查询的性能，再结合磁盘的读取特性(顺序读/随机读)，传统关系型数据库采用了B-Tree/B+Tree这样的数据结构



ES构建索引例子



有这么一组文档，每个文档包含名字、年龄、性别等信息

文档ID	名字	年龄	性别
1	Kate	24	female
2	John	24	male
3	Bill	29	male

ElasticSearch建立的倒排索引（index）如下:

名字（type）:

Term	Posting List
Kate	1
John	2
Bill	3

年龄（type）:

Term	Posting List
24	（1，2）
29	3

性别（type）:

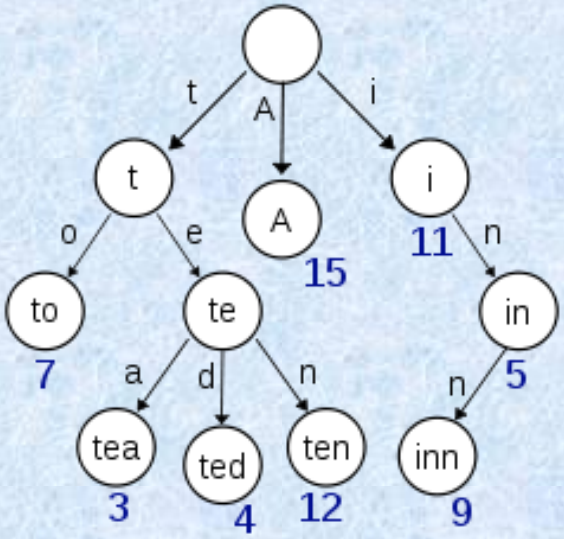
Term	Posting List
female	1
male	（2，3）

Term Dictionary

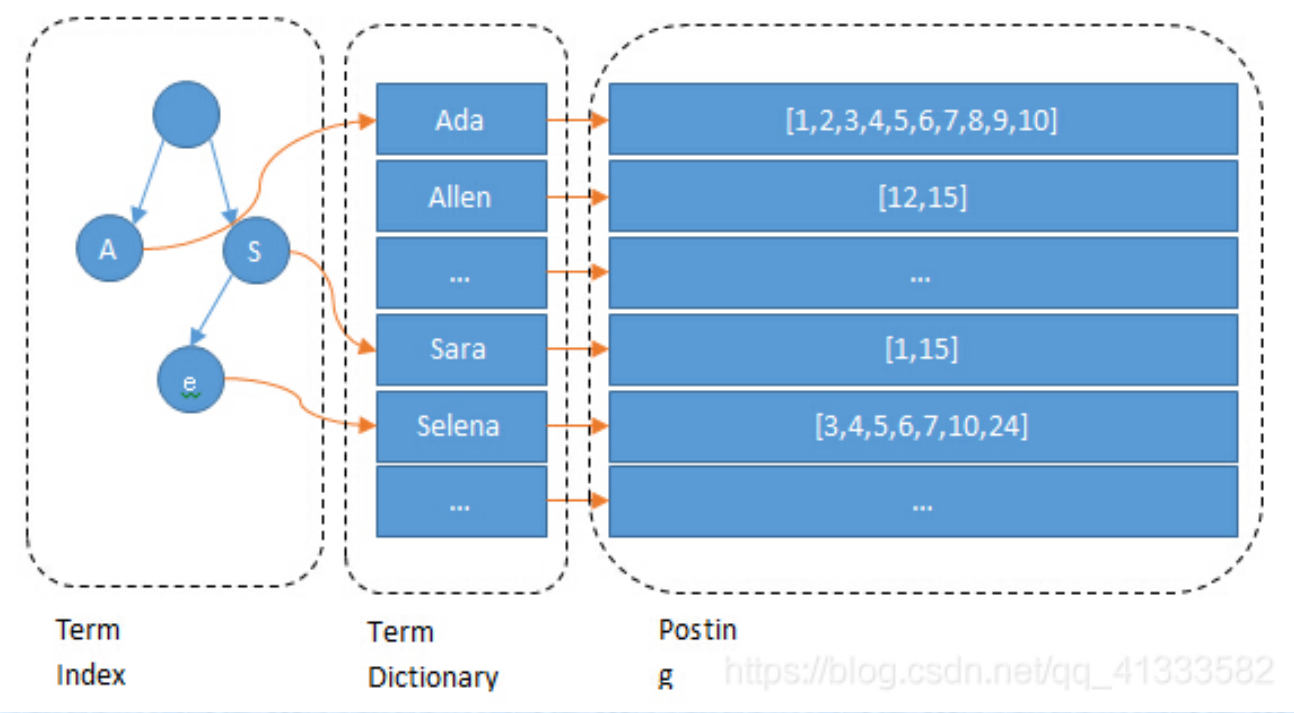
ES为了能快速找到某个term，将所有的term排序构成字典，用二分法查找term，具有logN的查找效率。

Term Index

ES采用存入内存查找term不读磁盘的思路， 但是如果Term Dictionary太大放内存不现实，于是有了Term Index，就像字典里的索引页一样，A开头的有哪些term，分别在哪页，可以理解Term Index是一颗树：



内存进一步压缩：这棵树不会包含所有的term，它只包含term的一些前缀。通过term index可以快速地定位到term dictionary的某个offset，然后从这个位置再往后顺序查找。



所以Term Index树不需要存下所有的term，而仅仅是它们的一些前缀与Term Dictionary的block之间的映射关系，再结合FST (Finite State Transducers)的压缩技术，可以使term index缓存到内存中。从Term Index查到对应的term dictionary的block位置之后，再去磁盘上找term，大大减少了磁盘随机读的次数。

FST (Finite State Transducers)

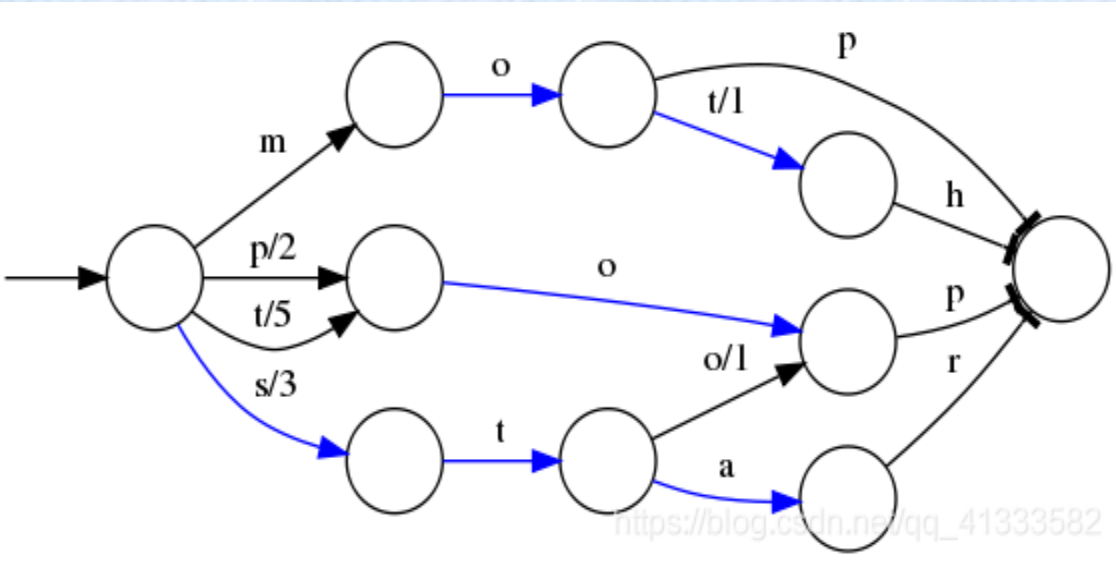
FST以字节的方式存储所有的term，这种压缩方式可以有效的缩减存储空间，使得term index足以放进内存，但这种方式也会导致查找时需要更多的CPU资源。

“ O “　表示一种状态

“ –> “　表示状态的变化过程，上面的字母/数字表示状态变化和权重

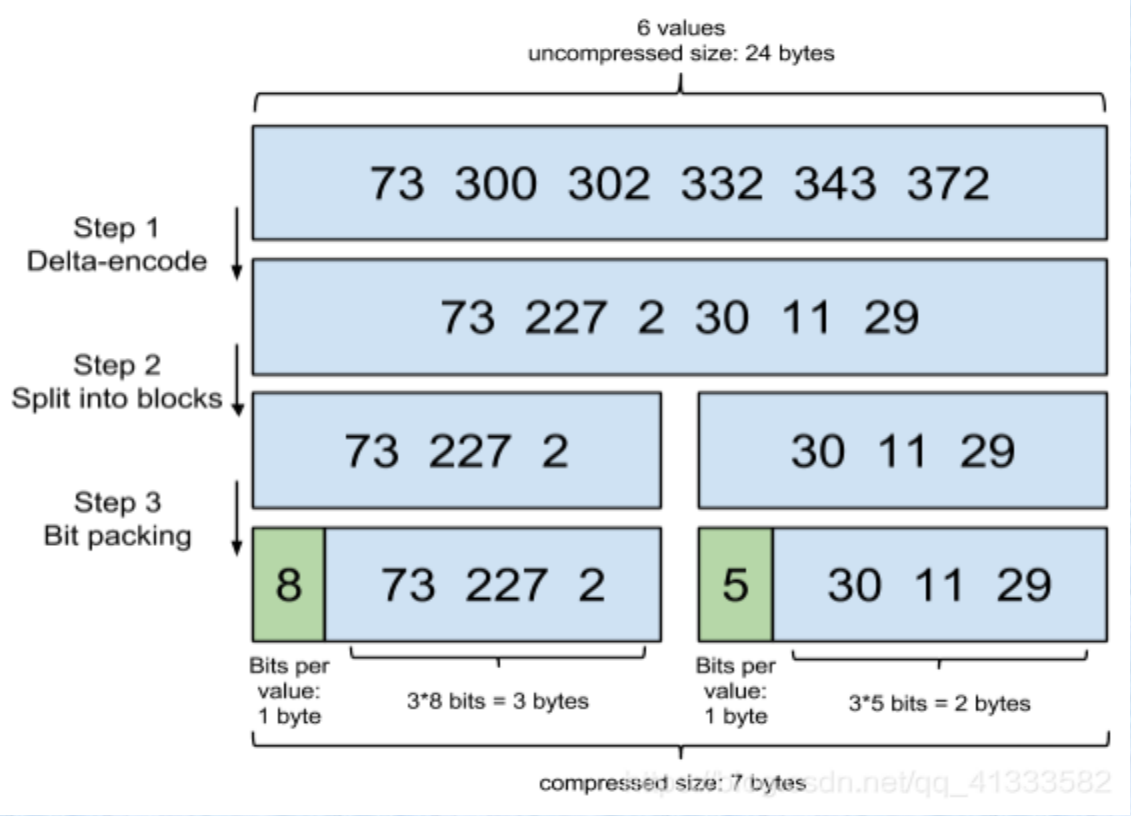
将单词分成单个字母通过 O 和 –> 表示出来，0权重不显示。如果O后面出现分支，就标记权重，最后整条路径上的权重加起来就是这个单词对应的序号。

序号	单词
0	mop
1	moth
2	pop
3	star
4	stop
5	top



Frame Of Reference 进一步内存压缩posting list

对于posting list数组，通过增量，将原来的大数变成小数仅存储增量值，变成字节(byte)存储，而不是整数型数组存储（耗内存）。

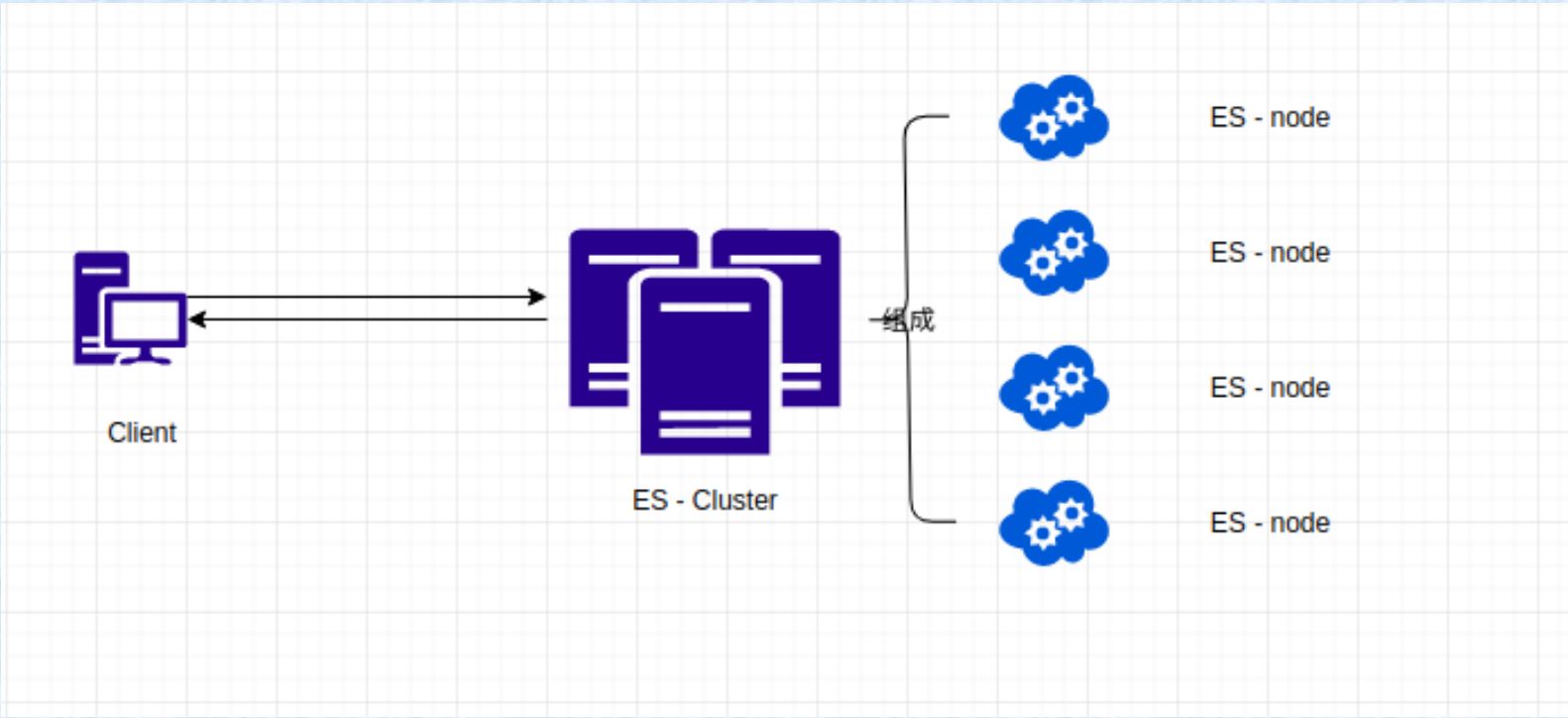


ElasticSearch 物理架构

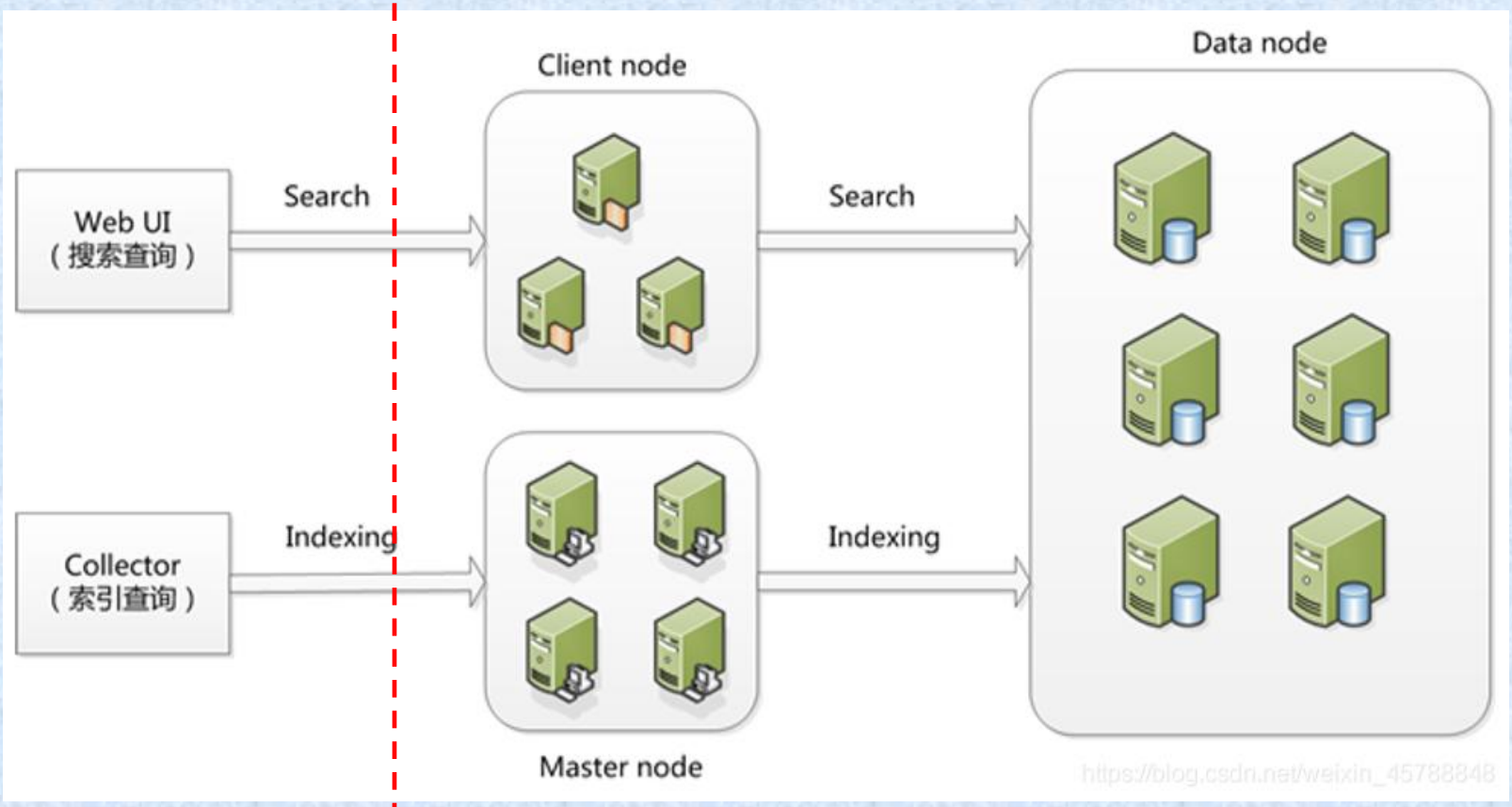
- **集群架构**：主从架构（Master/Slave）
- **存储架构**：索引(分片与副本)/类型/文件/字段
- **容错机制**：横向扩容与容错

系统架构：主从架构（Master/Slave）

ES集群由多个节点（node）组成，每个节点上运行一个ES进程。节点有主从之分。



ES集群架构


Client node
Data node
Web UI
（搜索查询）
Search
Search
Collector
（索引查询）
Indexing
Indexing
Master node

ES集群包含4种节点

- Master节点：负责保存更新集群的元数据信息（配置信息、节点信息、索引信息、关联信息、分片信息等），并同步到所有节点
- Datanode节点：负责数据存储和查询
- Coordinator节点：负责路由索引请求、分发批量索引请求、聚合搜索结果
- Ingestor节点：负责输入数据的处理和转换

集群节点配置

- 对于小规模集群（<10节点），让所有节点具备datanode属性并且是master candidate（在elasticsearch.yml中，将nodes.master属性设置为true）
- 对于大规模集群，再增配coordinator节点（也称为路由节点）

ES集群节点有三种角色

master node：主要用于元数据(metadata)的处理，比如索引的新增、删除、分片分配等。

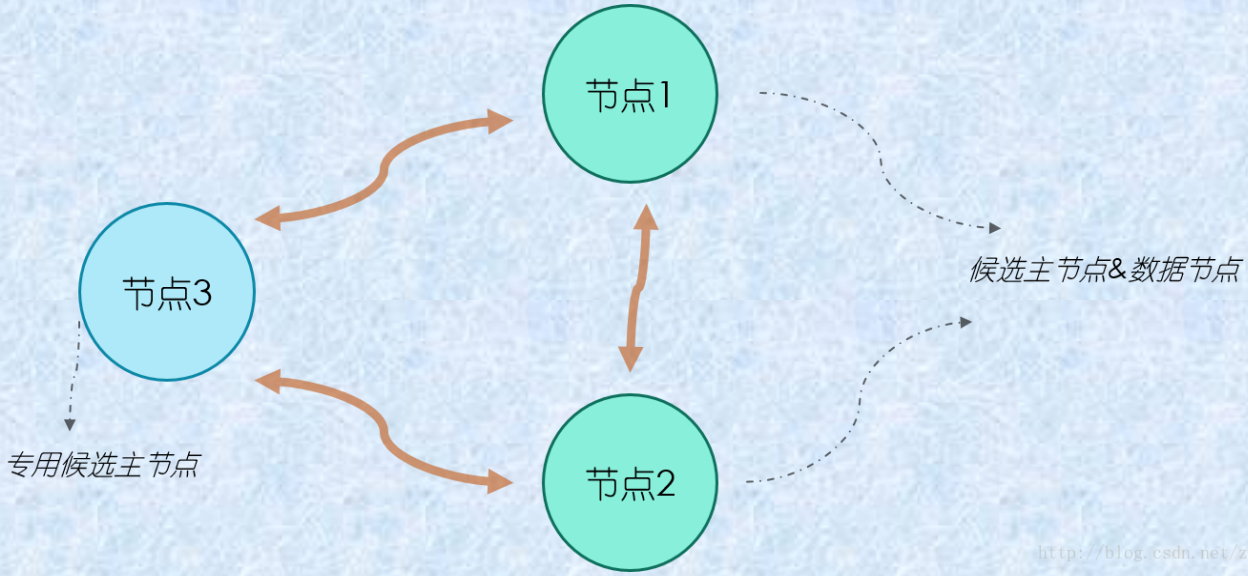
data node：保存数据分片，负责数据相关操作，比如分片的 CRUD，以及搜索和整合操作。这些操作都比较消耗 CPU、内存和 I/O 资源。

client node：起到路由请求的作用，实际上可以看做负载均衡器。

节点名称	IP地址	集群角色
server1	172.16.213.37	Master node、data node
server2	172.16.213.77	data node
server3	172.16.213.78	Master node、data node

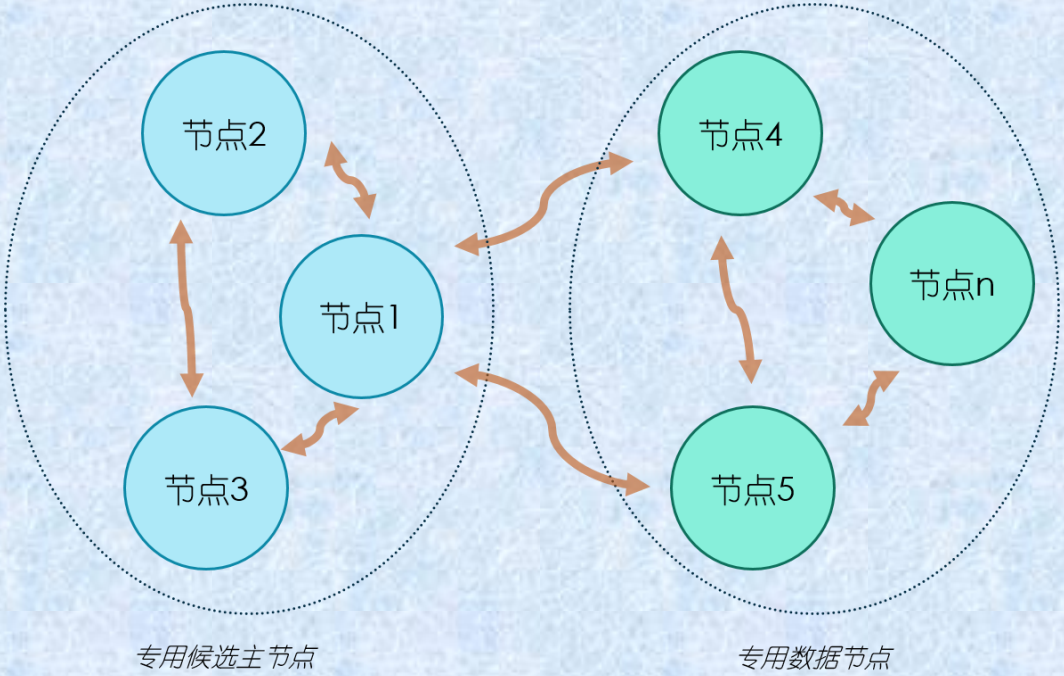
简单集群部署

3个节点都可作为候选主节点，可以设置最少候选主节点个数为2，节点1和2同时作为数据节点，两个节点的数据相互备份。后面根据需要逐步加入专用的数据节点。



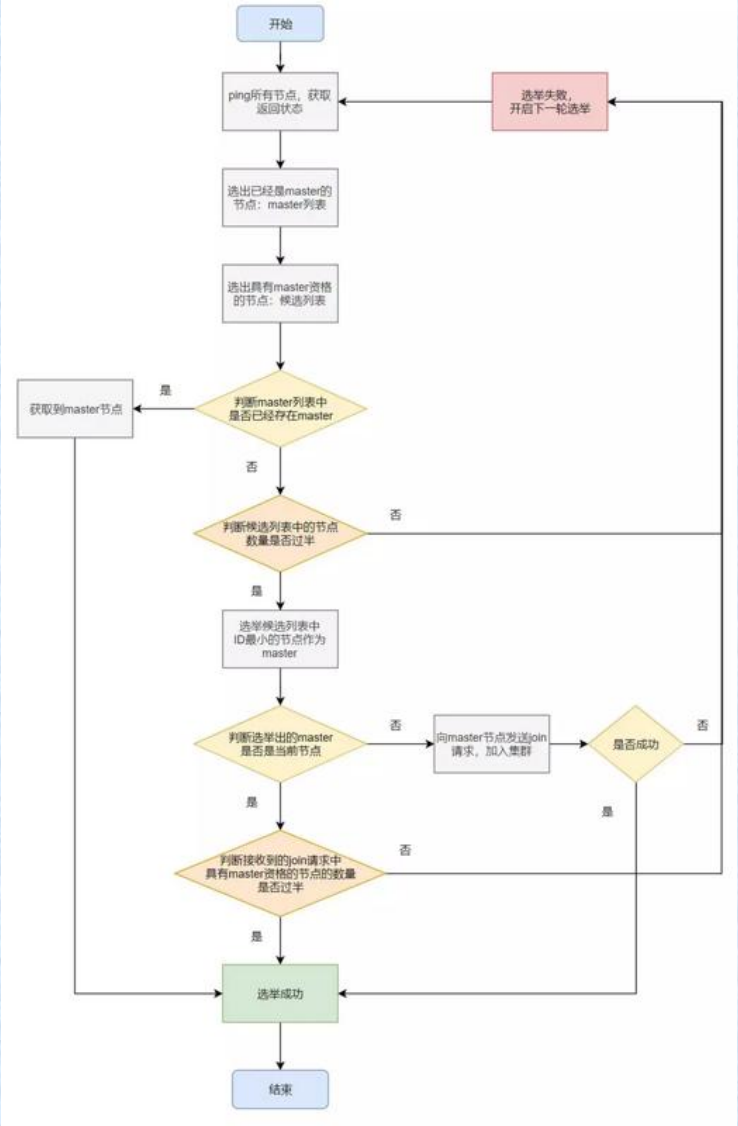
大规模集群部署

将候选主节点（节点1，节点2，节点3）和数据节点（节点4，节点5 。。。节点n）分离。候选主节点群（1， 2， 3组成）通过选举机制确定一个主节点。当一个新节点启动，会使用Zen Discovery机制去寻找主节点及集群其他节点，并在主节点处注册加入ES集群。

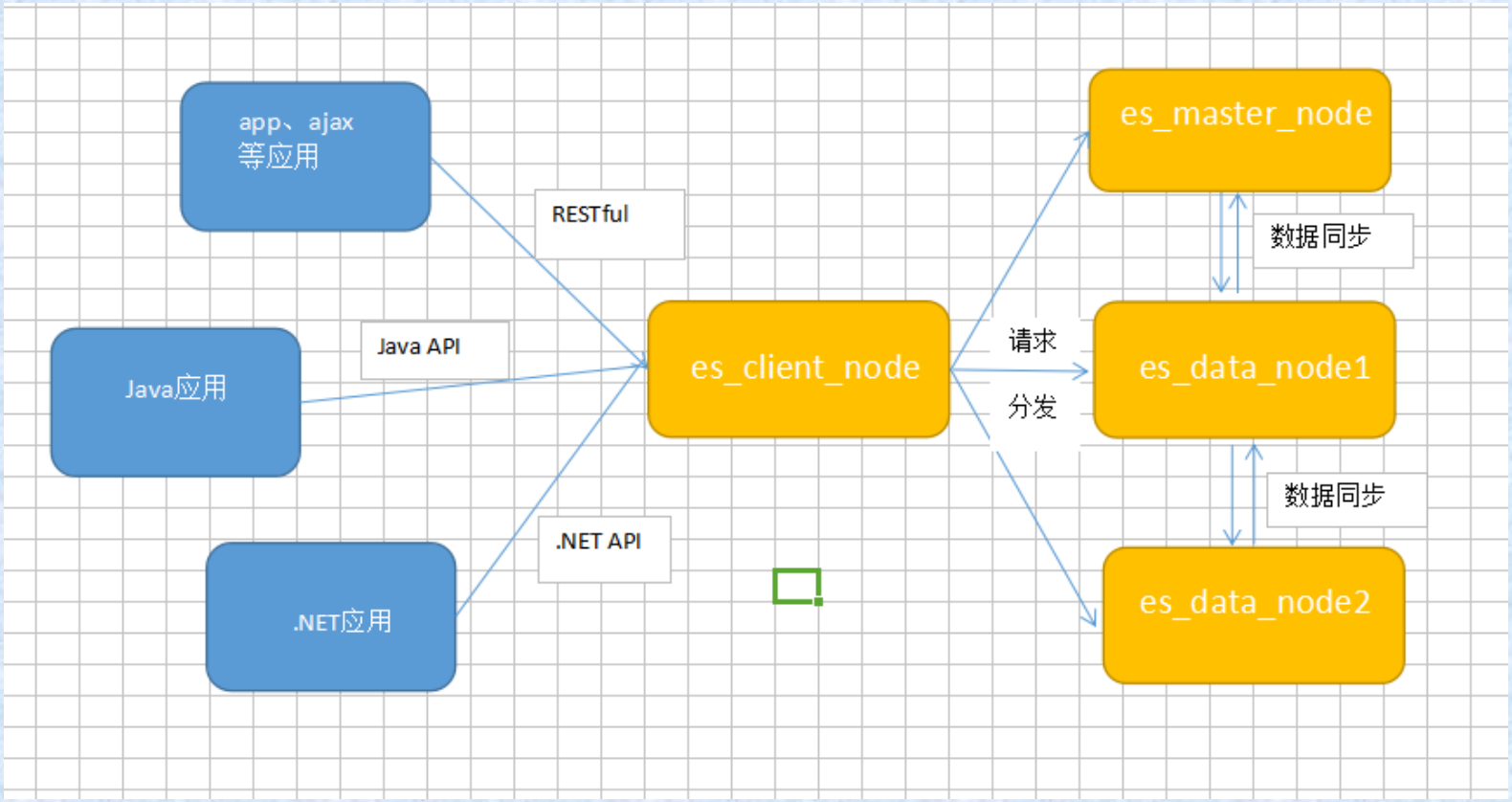


Master选举算法

- 如果集群有master，认可master并加入集群
- 如果集群没有master，从master candidates中选取节点ID最小者作为master节点



ES集群应用架构


app、ajax
等应用
RESTful
Java应用
Java API
.NET应用
.NET API
es_client_node
请求
分发
es_master_node
数据同步
es_data_node1
数据同步
es_data_node2

ElasticSearch 存储架构

索引（Index）

ES将数据存储于一个或多个索引中，索引是具有类似特性的文档的集合，类比传统关系型数据库的一个数据库（database），或者一个数据存储方案（schema）。索引由其名称（必须全小写字符）进行标识，并通过引用此名称完成文档的创建、搜索、更新及删除操作。

类型（Type）

类型是索引内部的逻辑分区（category/partition），一个索引内部可定义一个或多个类型（type）。类比传统关系型数据库的一张表。

文档（Document）

文档是es中可搜索的最小单位，由一个或多个字段组成，类似于关系型数据库中的一行记录。es的文档是以JSON格式保存，每个JSON对象由一个或多个字段组成，字段类型可以是布尔，数值，字符串、二进制、日期等数据类型。每个文档都有唯一的id, 可以由我们自己指定，也可以由es自动生成。

ElasticSearch 存储架构（续）

字段（Field）

相当于数据库中的column。ES中，每个文档，其实是以json形式存储的。而一个文档可以被视为多个字段的集合。比如一篇文章，可能包括了主题、摘要、正文、作者、时间等信息，每个信息都是一个字段，最后被整合成一个JSON串，落地到磁盘。

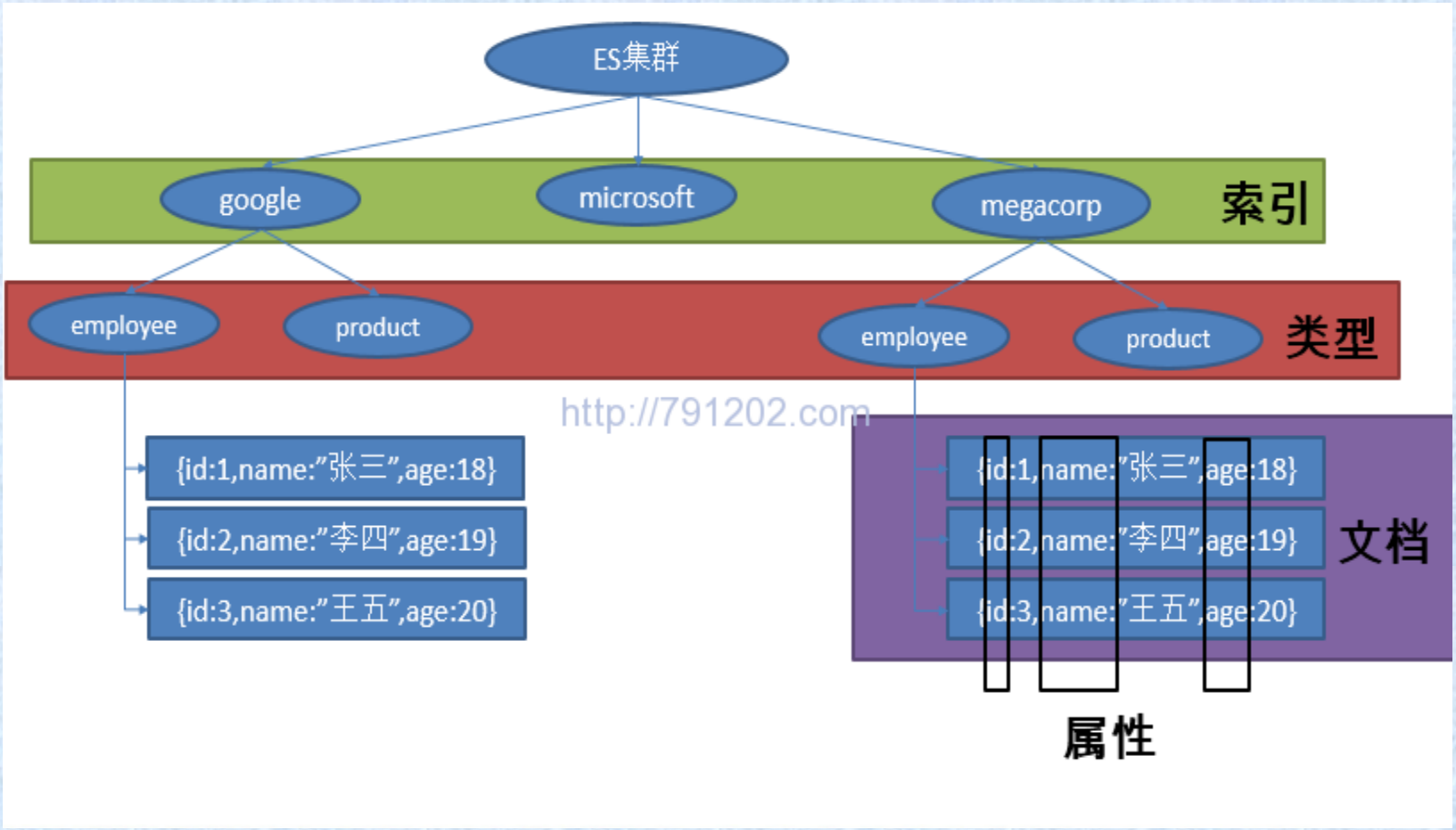
映射（Mapping）

相当于数据库中的schema，用来约束字段的类型，不过 Elasticsearch 的 mapping 可以不显示地指定、自动根据文档数据创建。

关系型数据库	ElasticSearch
Database	Index
Table	Type
Row	Document
Column	Field
Schema	Mapping

Relational DB -> Databases -> Tables -> Rows -> Columns

ElasticSearch -> Indices -> Types -> Documents -> Fields


ES集群
google
microsoft
megacorp
索引
employee
product
employee
product
类型
http://791202.com
{id:1,name:"张三",age:18}
{id:2,name:"李四",age:19}
{id:3,name:"王五",age:20}
{id:1,name:"张三",age:18}
{id:2,name:"李四",age:19}
{id:3,name:"王五",age:20}
文档
属性

类型举例：

比如一首诗，有诗题、作者、朝代、字数、诗内容等字段，那么首先可以建立一个名叫 Poems 的索引(index)。

然后创建一个名叫 Poem 的类型(type)，类型是通过 Mapping 来定义每个字段(field)的类型。

比如诗题、作者、朝代都是 keyword 类型，诗内容是 text 类型，而字数是 integer 类型，最后就是把数据以 JSON格式存入。

分词：keyword 类型不需分词的，直接根据字符串建立反向索引；Text 类型在存入时需要先分词，然后根据分词后的内容建立反向索引。

索引
poems

类型
```
"poem": {
  "properties": {
    "title": {
      "type": "keyword",
    },
    "author": {
      "type": "keyword",
    },
    "dynasty": {
      "type": "keyword"
    },
    "words": {
      "type": "integer"
    },
    "content": {
      "type": "text"
    }
  }
}
```

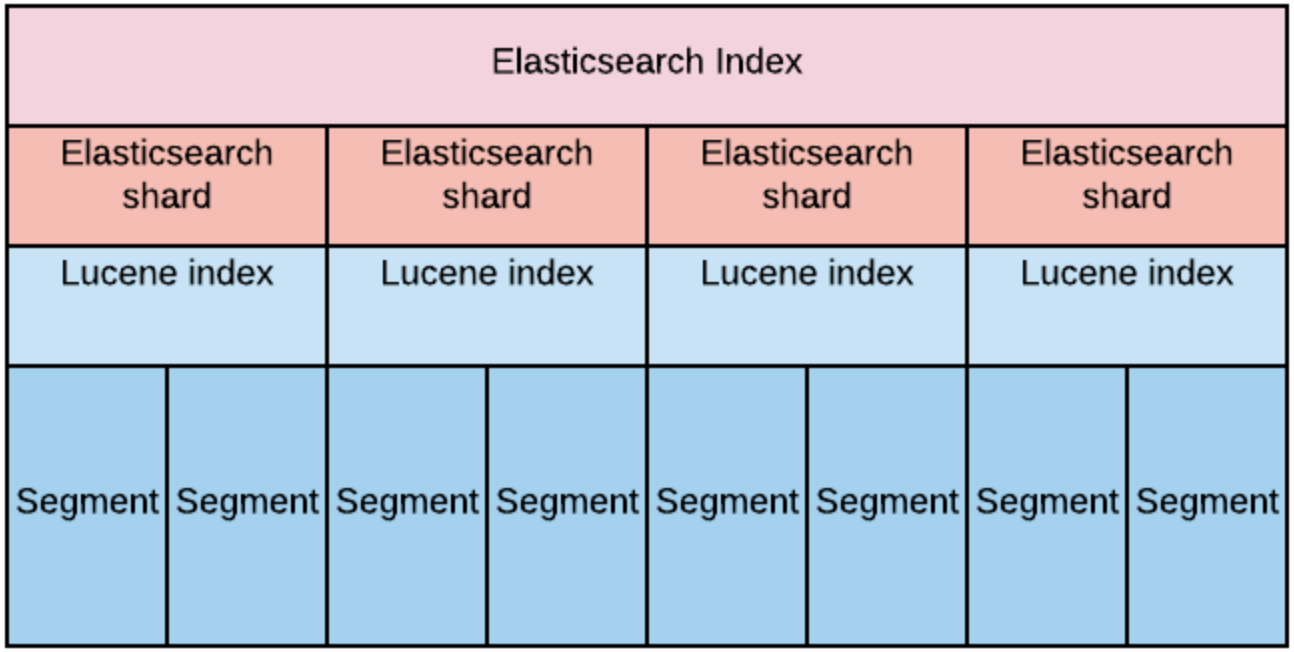
文档
```
{
  "title": "静夜思",
  "author": "李白",
  "dynasty": "唐",
  "words": "20",
  "content": "床前明月光，疑是地上霜。举头望明月，低头思故乡。"
}
```

文档举例：

ES中的一个document可以是一条客户数据，一条商品分类数据，一条订单数据，通常用JSON数据结构表示。一个document里面有多个field，每个field就是一个数据字段。

比如：

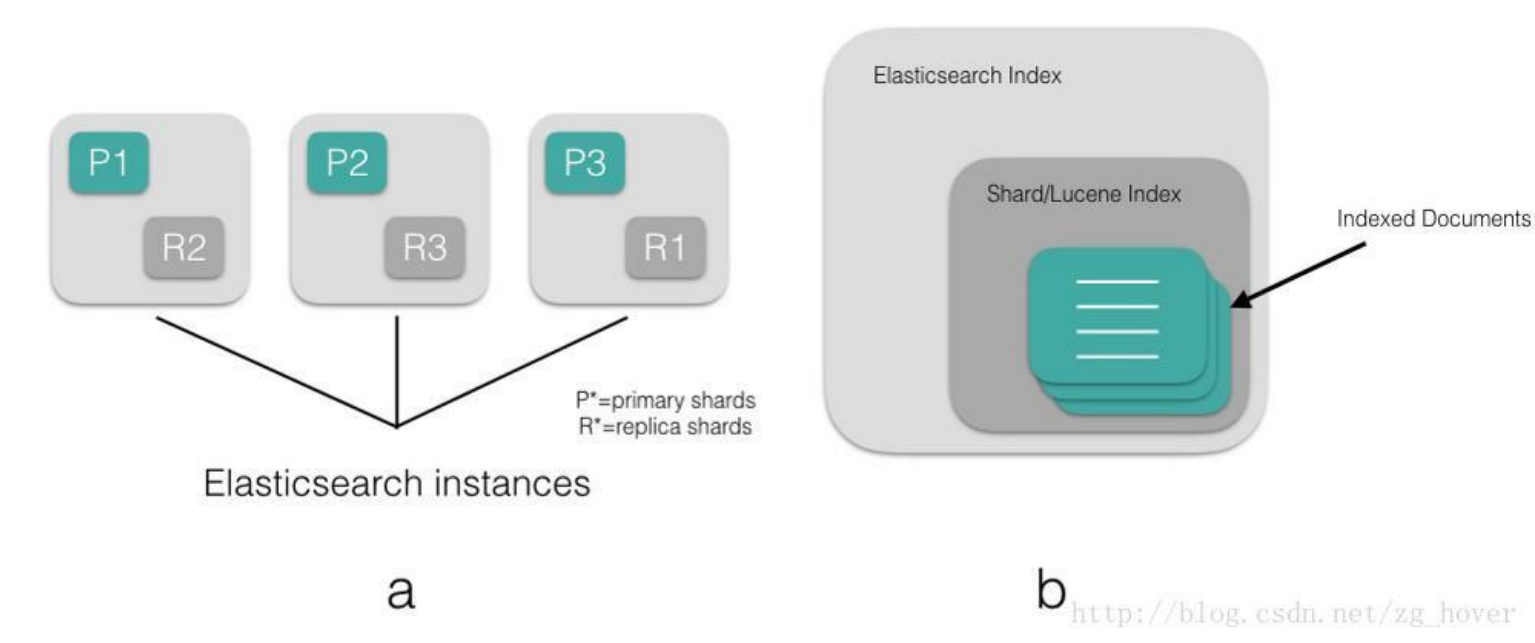
```
product document
{
  "product_id": "1",
  "product_name": "高露洁牙膏",
  "product_desc": "高效美白",
  "category_id": "2",
  "category_name": "日化用品"
}
```


Elasticsearch Index
Elasticsearch shard
Elasticsearch shard
Elasticsearch shard
Elasticsearch shard
Lucene index
Lucene index
Lucene index
Lucene index
Segment
Segment
Segment
Segment
Segment
Segment
Segment
Segment

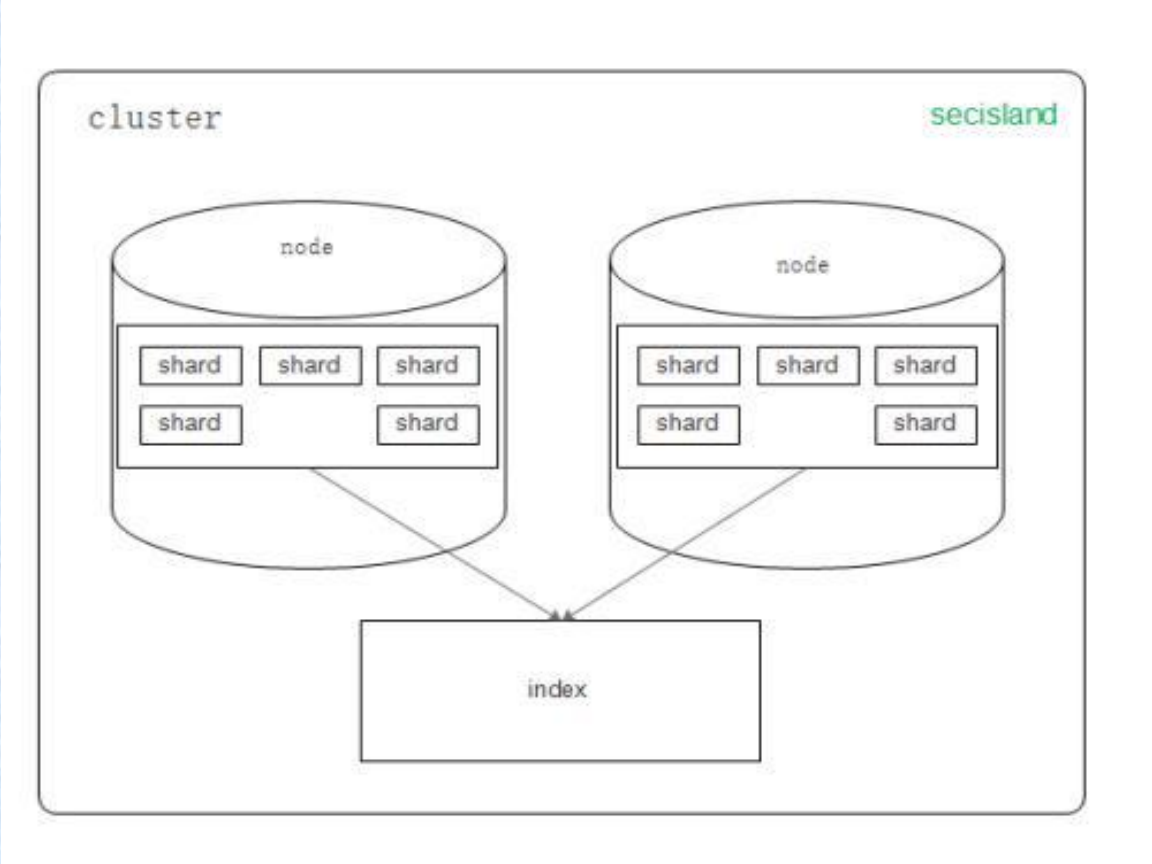
分片（Shard）与副本（Replica）

ES的索引数据按照分布式存储原理切分成1个或多个分片(shard)（默认为5）。shard就是实际存储数据的Lucene索引，它本身就是一个搜索引擎。每个shard可以有1个或多个副本(replicas)（默认为1）。

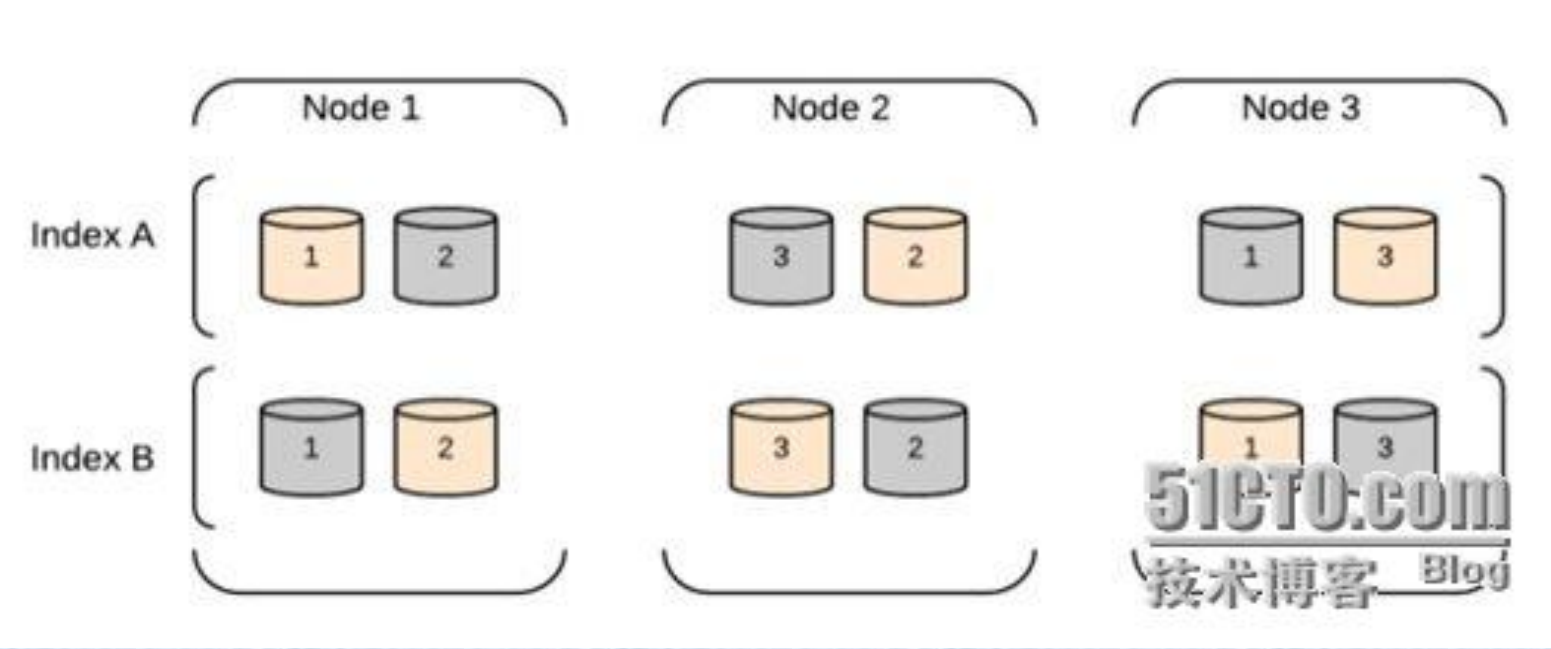
每个原分片称为primary shard（下图中的P1, P2, P3），用于存储索引信息；副本称为replica shard（R1, R2, R3），用于数据备份和提高查询效率。

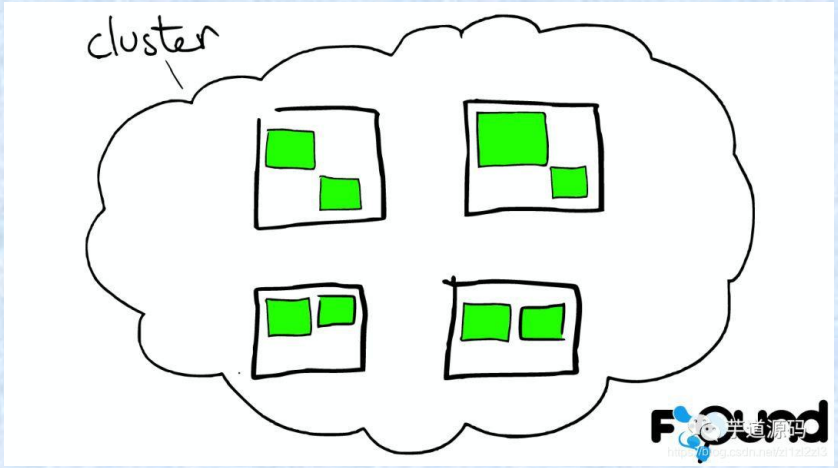


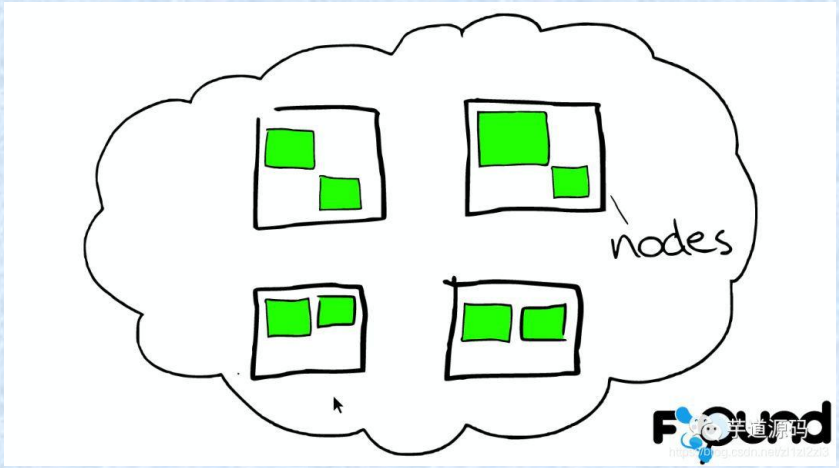
ES集群由多个节点(node) 组成，各shard分布存储于这些节点上。

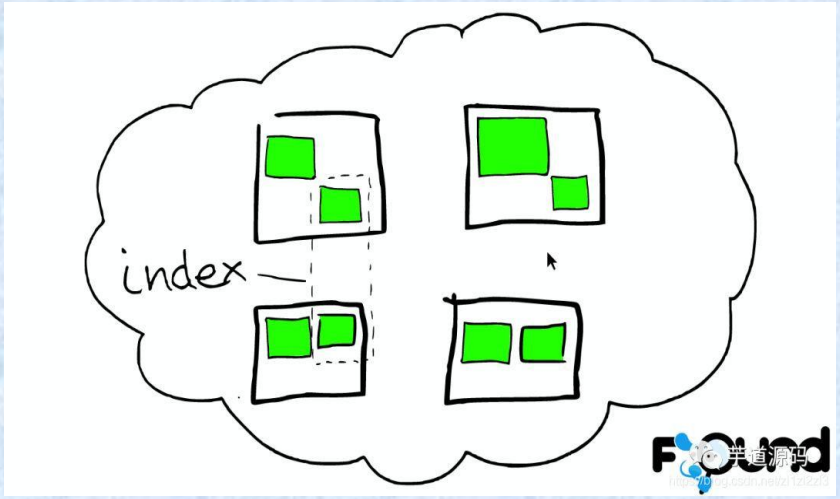


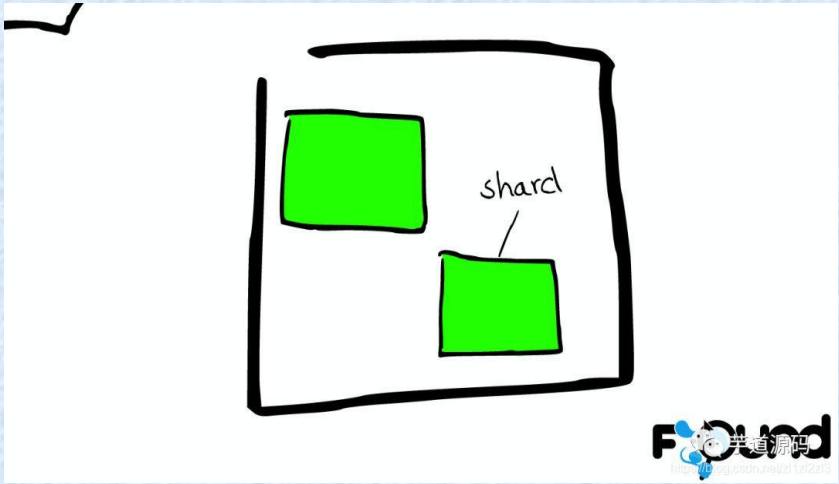
ES可自动在节点间按需要移动shards，例如增加新节点或节点发生故障时。



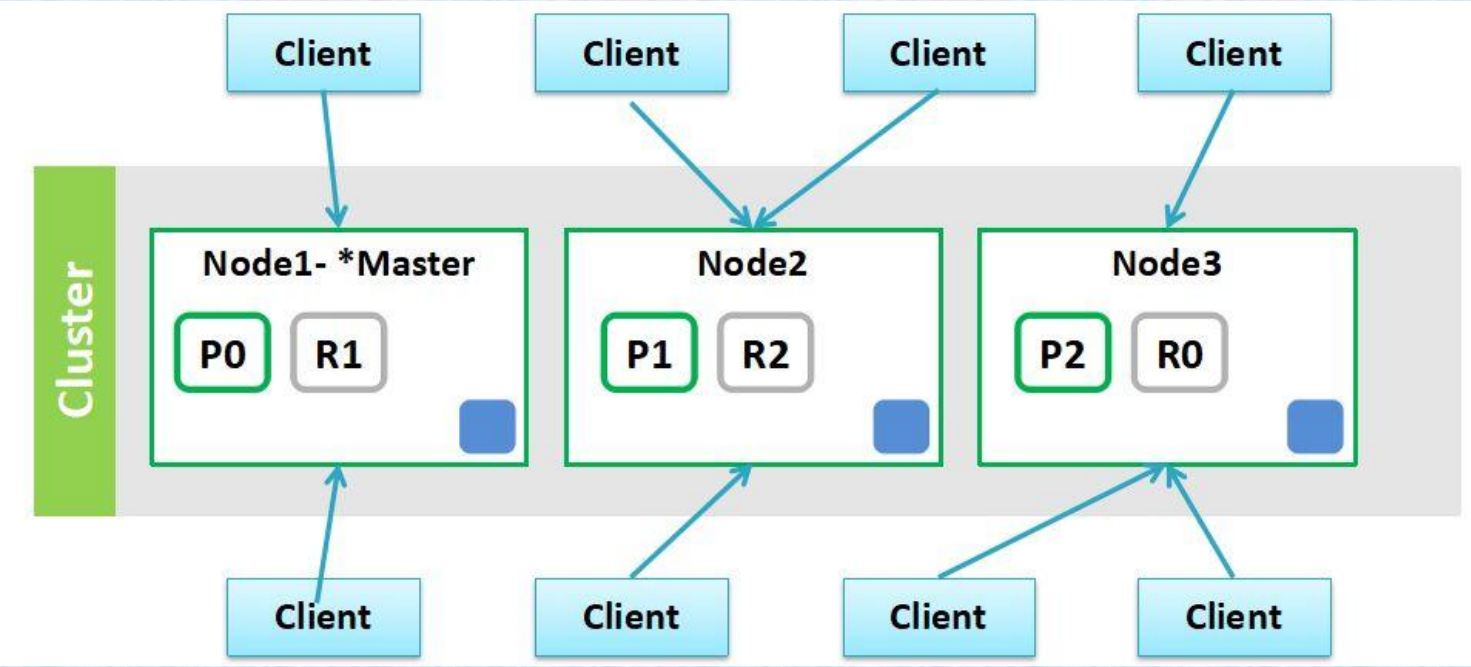

cluster


nodes

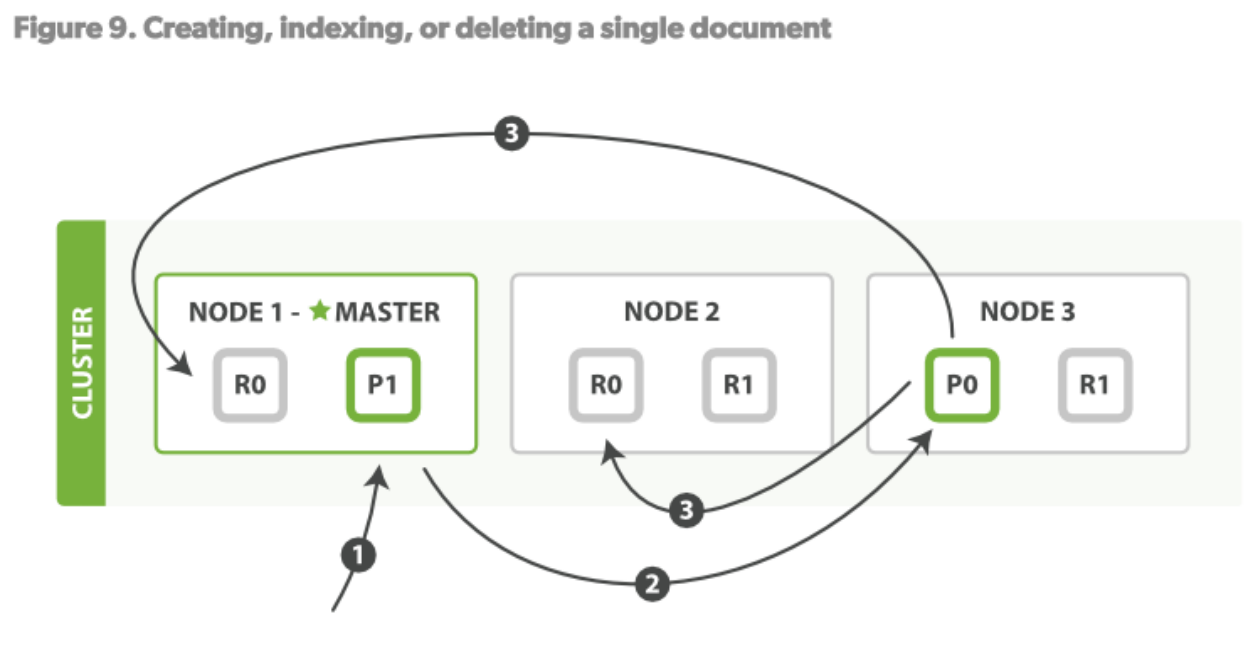

index


shard

- 客户端写数据（索引一个文档）需通过主节点（Node1）完成，主节点会将shard和replica分发到不同节点完成
- 客户端读数据（查询请求）可以通过任何一个节点来完成，提高查询效率



索引文档步骤



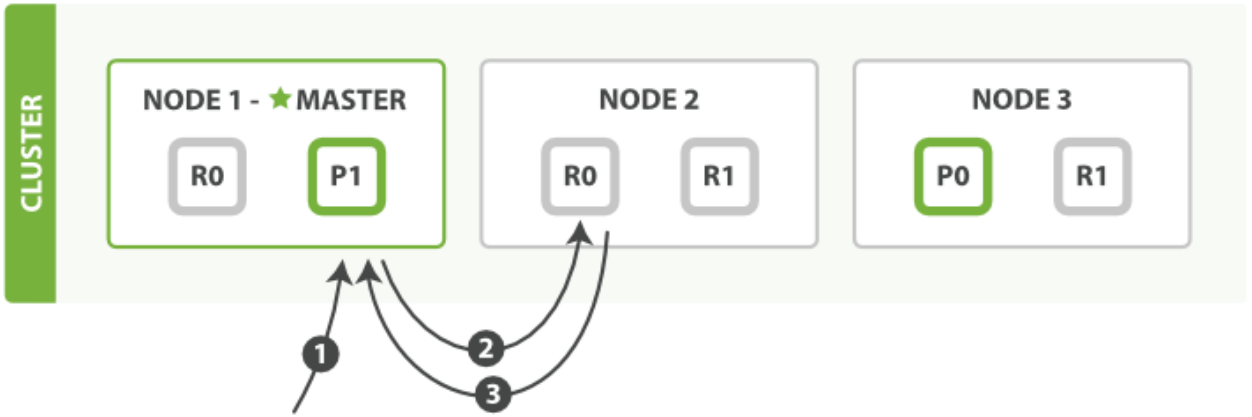
① 客户端向Node1 (master) 发送索引文档请求

② Node1 根据文档ID(_id字段)计算出该文档应该属于shard0，然后请求路由到Node3的P0分片上

③ Node3在P0上执行了请求。如果请求成功，则将请求并行的路由至Node1，Node2的R0上。当所有的Replicas报告成功后，Node3向请求的Node(Node1)发送成功报告，Node1再报告至Client。

读取文档步骤

Figure 10. Retrieving a single document

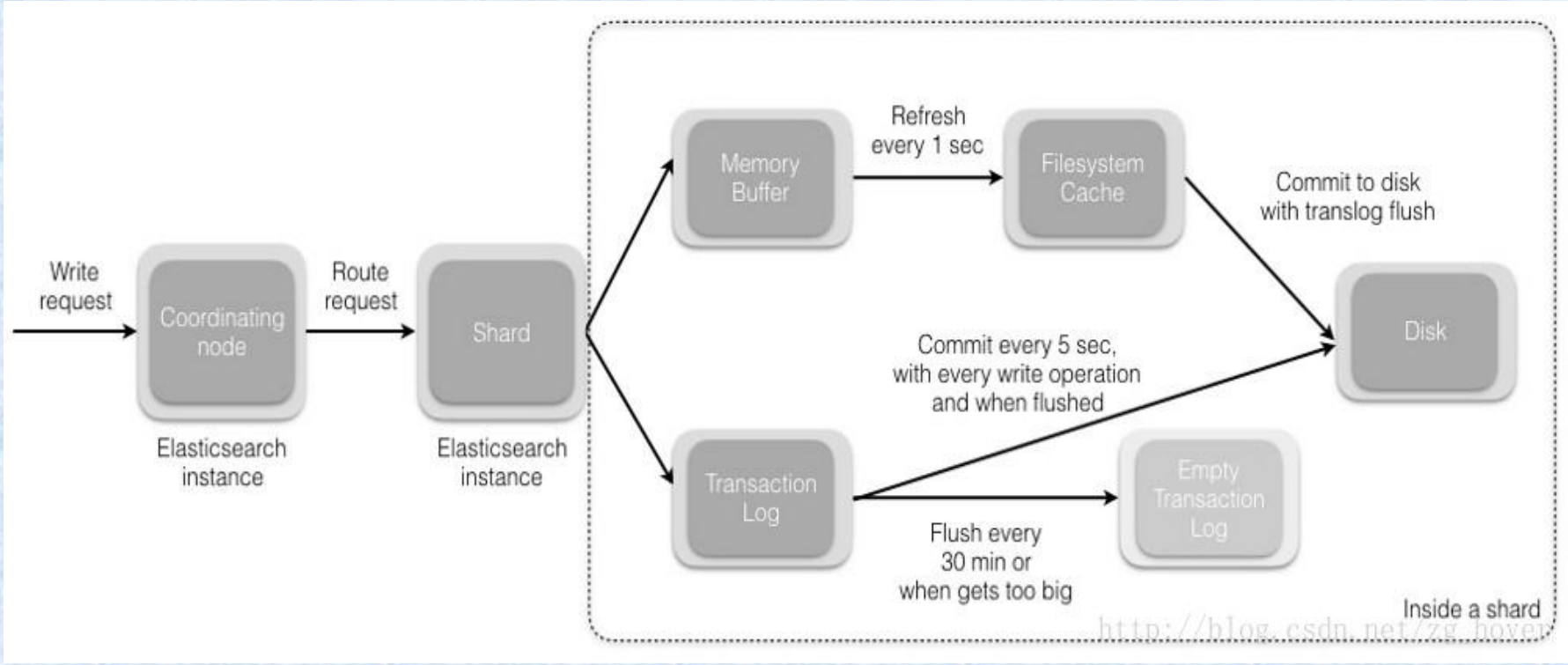


① 客户端发送Get请求到NODE1。

② NODE1使用文档的_id决定文档属于shard0. shard0的所有拷贝存在于所有3个节点上。这次，它将请求路由至NODE2。

③ NODE2将文档返回给NODE1，NODE1将文档返回给客户端。 对于读请求，请求节点(NODE1)将在每次请求到来时都选择一个不同的replica，来达到负载均衡

Shard上的write/create实现原理



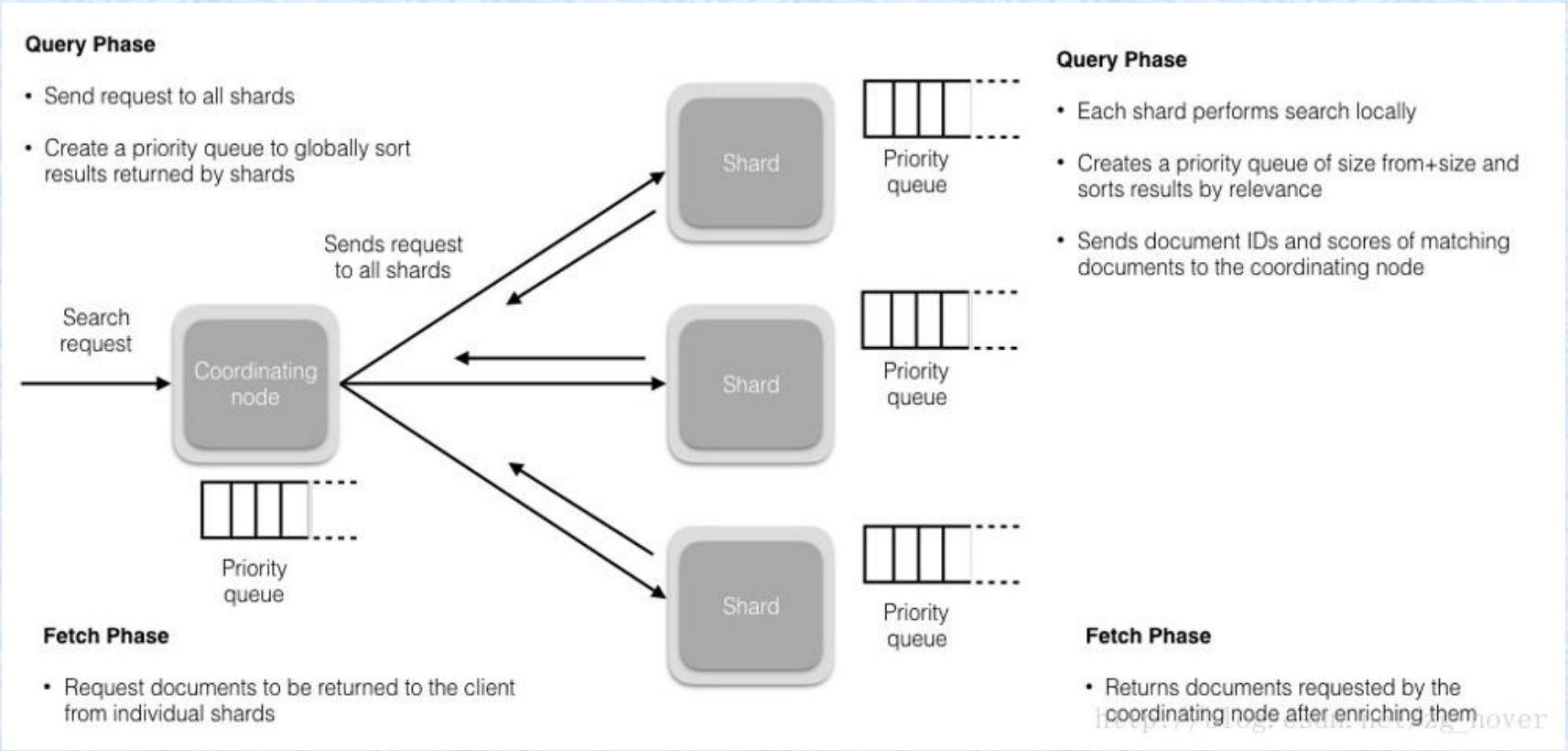
Transaction Log是ES的事务日志文件，它记录了所有对索引分片的事务操作（add/update/delete），用于分片的数据恢复。每个分片对应一个translog文件。

- 所有ES集群节点都包含哪个分片存在于哪个节点上的元数据。协调节点(coordinating node)使用文档ID（默认）将文档路由到对应的分片；
- ES将文档ID用主分片数量进行取模运算，以确定文档应被索引到哪个分片

  shard = hash(document_id) % (num_of_primary_shards)
- 当节点接收到来自协调节点的请求时，请求被写入到translog并将该文档添加到内存缓冲区。如果请求在主分片上成功，则请求将并行发送到副本分片。只有在所有主分片和副本分片上的translog被fsync’ed后，客户端才会收到该请求成功的确认；
- 内存缓冲区以固定的间隔刷新（默认为1秒），并将内容写入文件系统缓存中的新段。此分段的内容更尚未被fsync’ed(未被写入文件系统)，分段是打开的，内容可用于搜索；
- translog被清空，并且文件系统缓存每隔30分钟进行一次fsync，或者当translog变得太大时进行一次fsync。这个过程在ES中称为flush，过程中内存缓冲区被清除，内容被写入新的文件分段(segment)。当文件分段被fsync’ed并刷新到磁盘，会创建一个新的提交点（其实就是会更新文件偏移量，文件系统会自动做这个操作）。旧的translog被删除，一个新的开始。

Shard上的read实现原理

- 查询阶段（Query Phase）
- 获取阶段（Fetch Phase）



查询阶段 (Query Phase)

在此阶段，协调节点将搜索请求路由到索引(index)中的所有分片(shards)，分片独立执行搜索，并根据相关性分数创建一个优先级排序结果（稍后我们将介绍相关性分数）。所有分片将匹配的文档和相关分数的文档ID返回给协调节点。协调节点创建一个新的优先级队列，并对全局结果进行排序。可以有很多文档匹配结果，但默认情况下，每个分片将前10个结果发送到协调节点，协调创建优先级队列，从所有分片中分选结果并返回前10个匹配。

获取阶段 (Fetch Phase)

在协调节点对所有结果进行排序，已生成全局排序的文档列表后，它将从所有分片请求原始文档。所有的分片都会丰富文档并将其返回到协调节点。

搜索相关性 (Search Relevance)

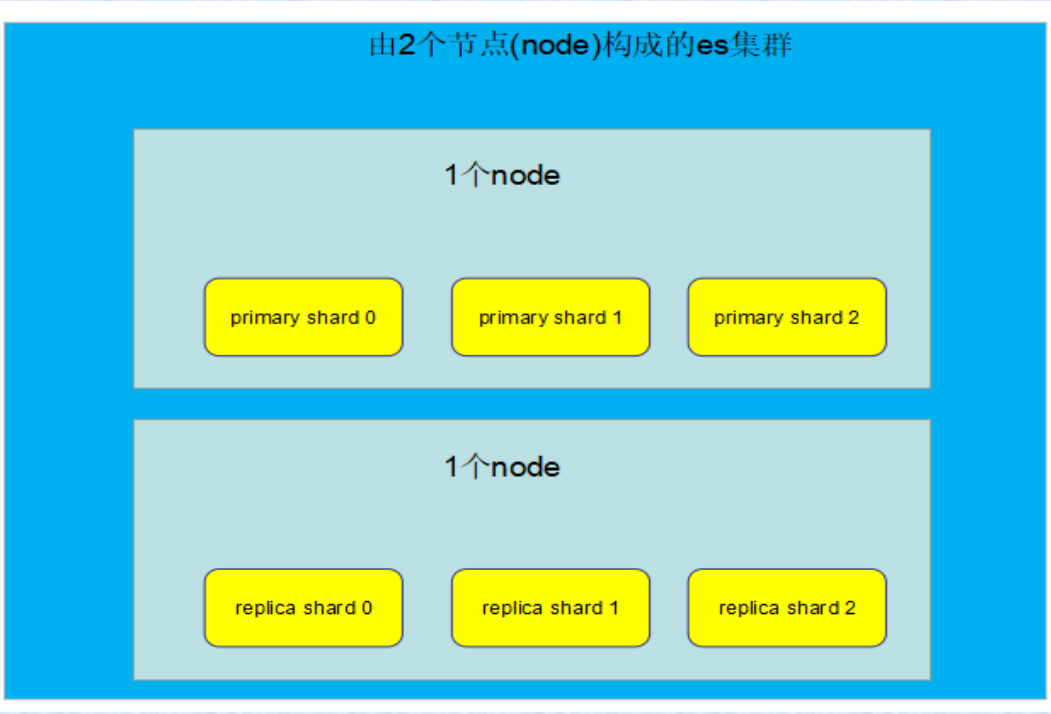
相关性由ES给搜索返回的每个文档的分数确定。用于评分的默认算法为tf / idf（术语频率/逆文档频率）。该术语频率测量术语出现在文档中的次数（更高频率=更高的相关性），逆文档频率测量术语在整个索引中出现的频率占索引中文档总数的百分比（更高的频率 ==较少的相关性）。最终得分是tf-idf分数与其他因素的组合。

ES容错机制

ES集群横向扩容

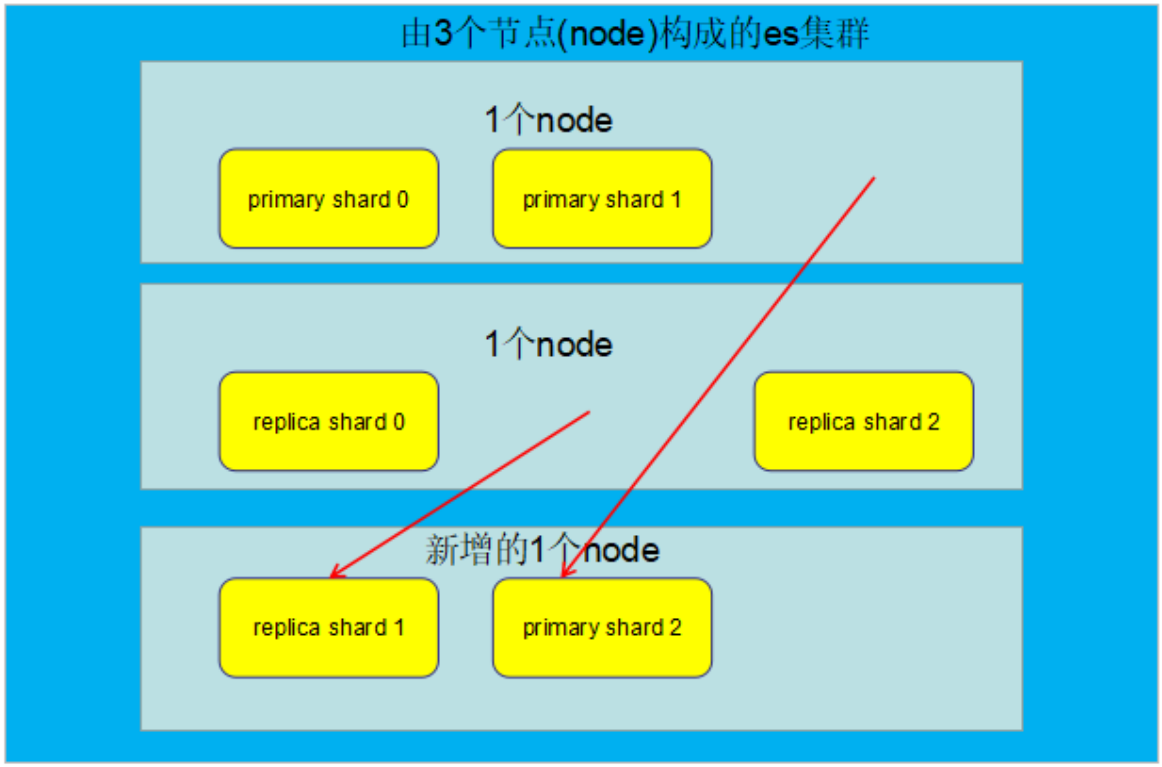
- **primary shard 和 replica shard自动负载均衡**

目前集群状态：2个node，3个primary shard，3个replica shard



◆ 给ES集群增加一个节点(node)

ES会自动对primary shard和replica shard进行负载均衡。

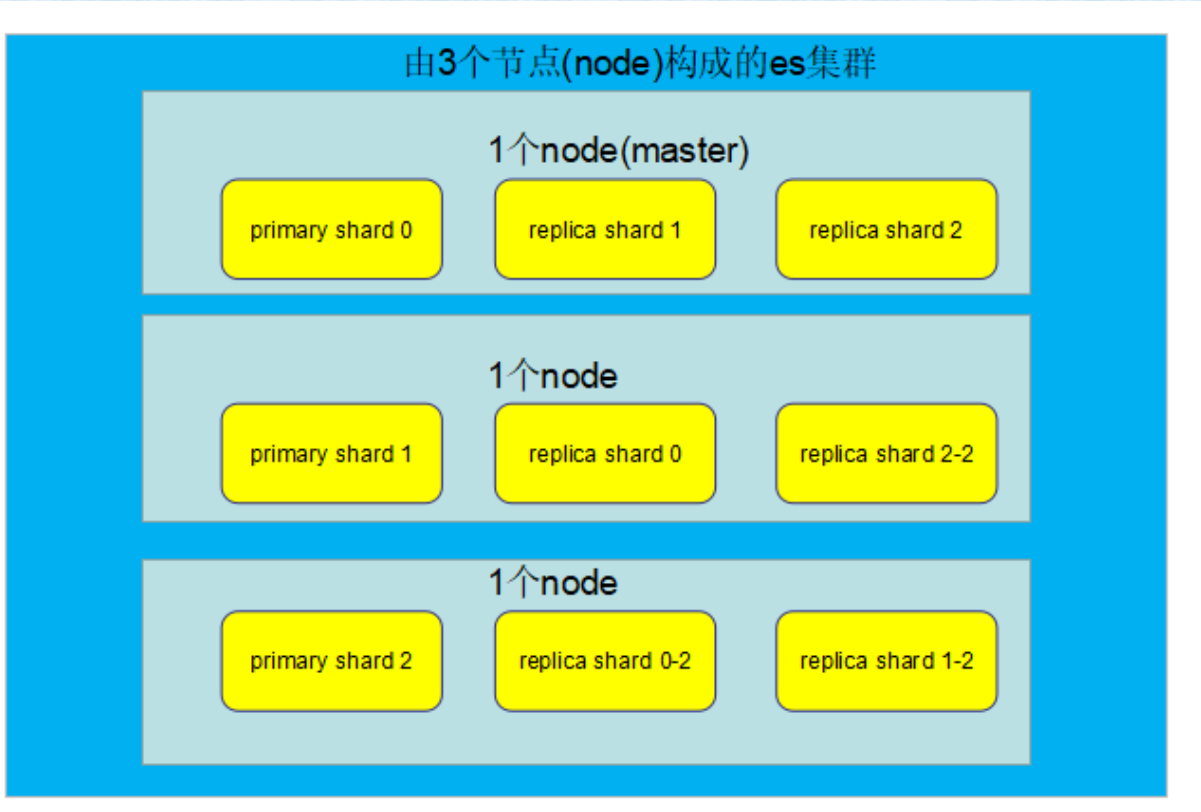


扩容之后，每个节点的shard数量更少，就意味着每个shard可以占用的节点上更多的资源，IO/CPU/Memory，整个系统，性能会更好。

ES容错机制（续）

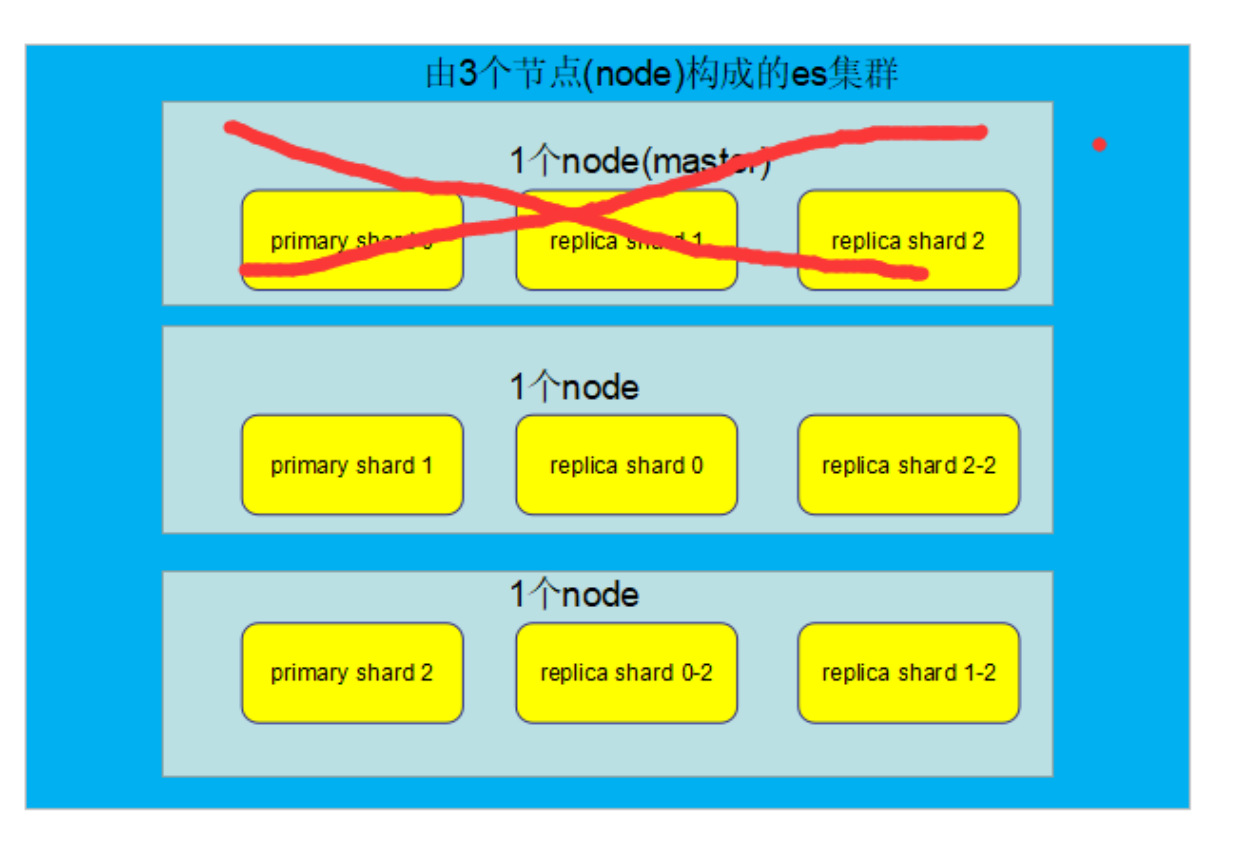
- **master选举、replica容错、数据恢复**

目前es集群状况：3个node，9个shard (3个primary，6个replica)

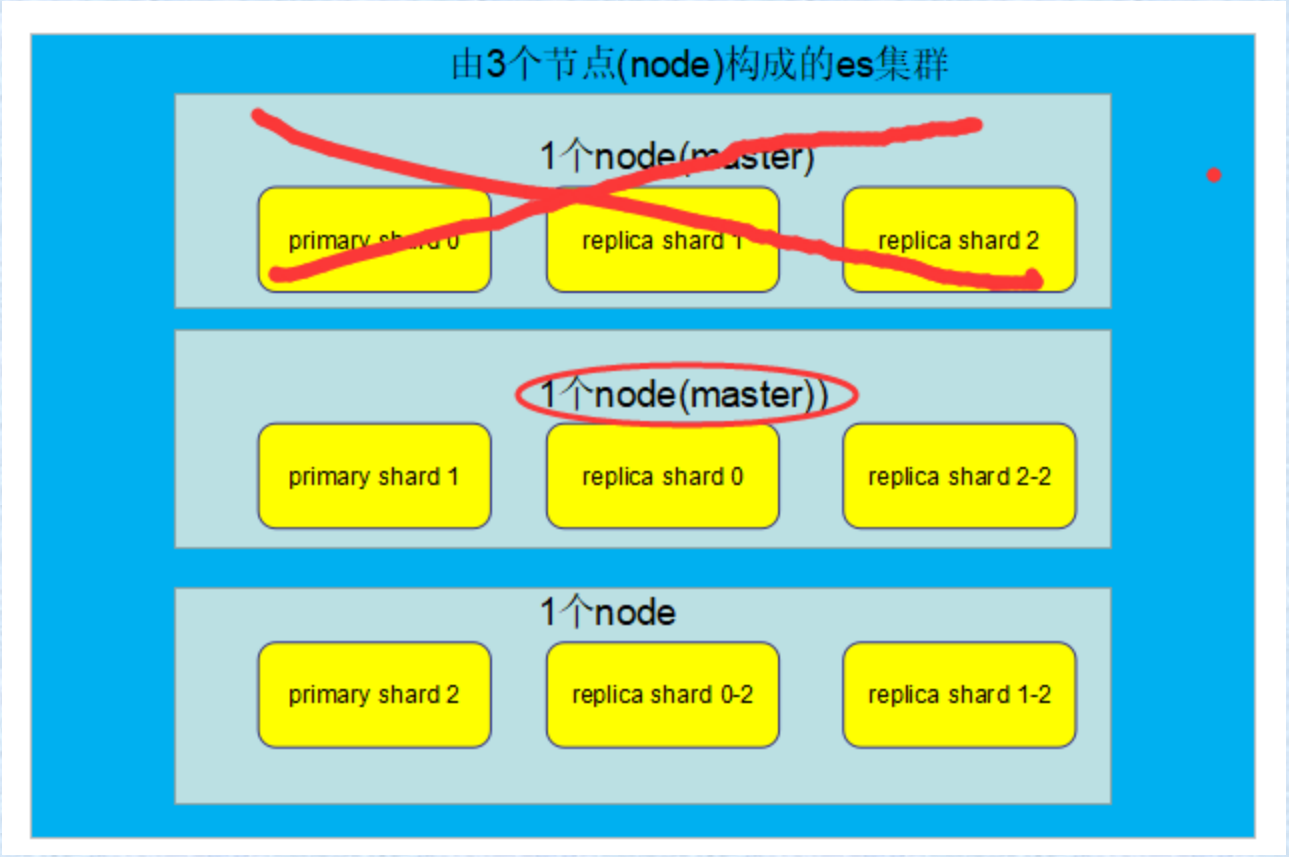


- 如果**master node**宕机

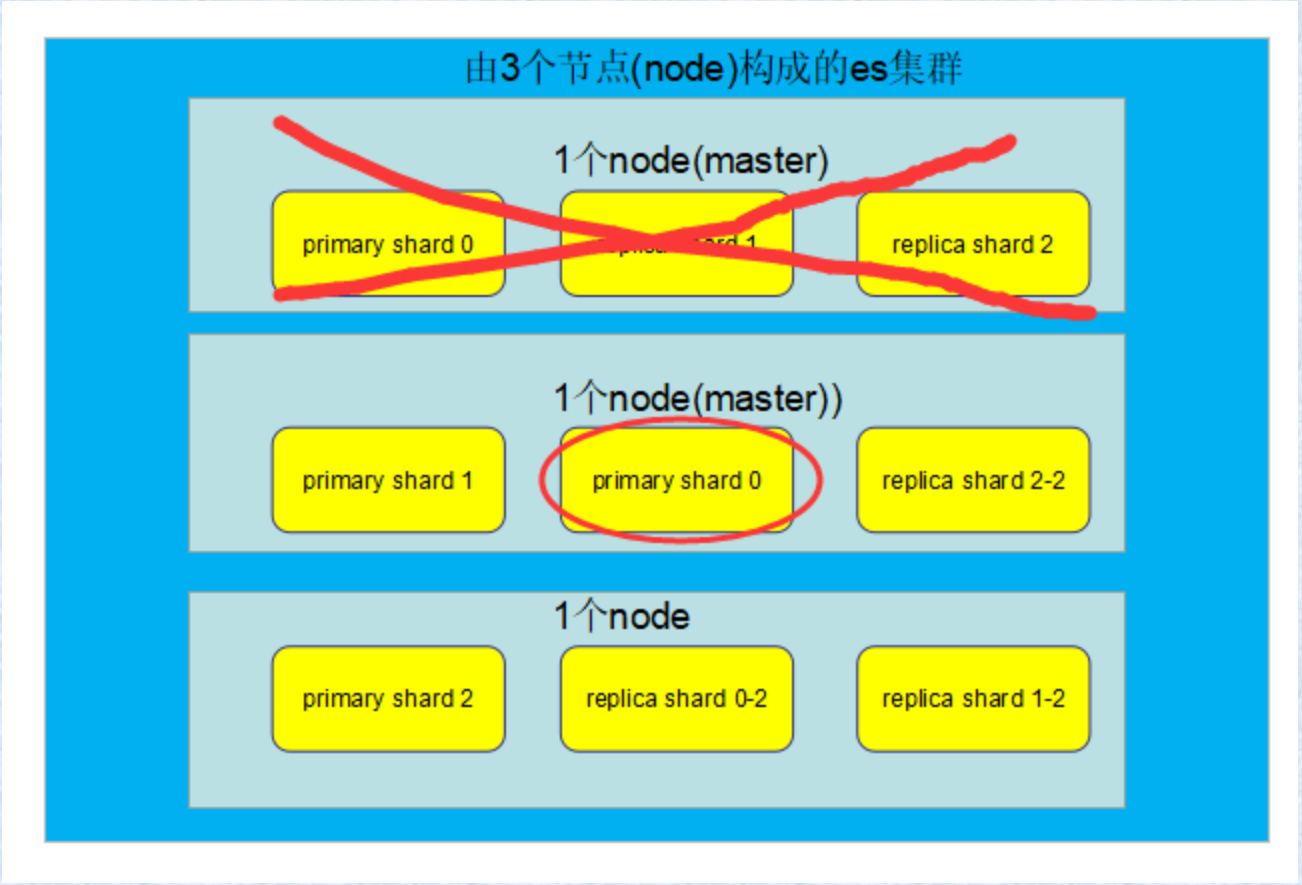
primary shard0、replica shard1、replica shard2 这三个shard就丢失了



◆ 容错第一步： **master选举，自动选举另外一个node成为新的master，承担起master的责任**

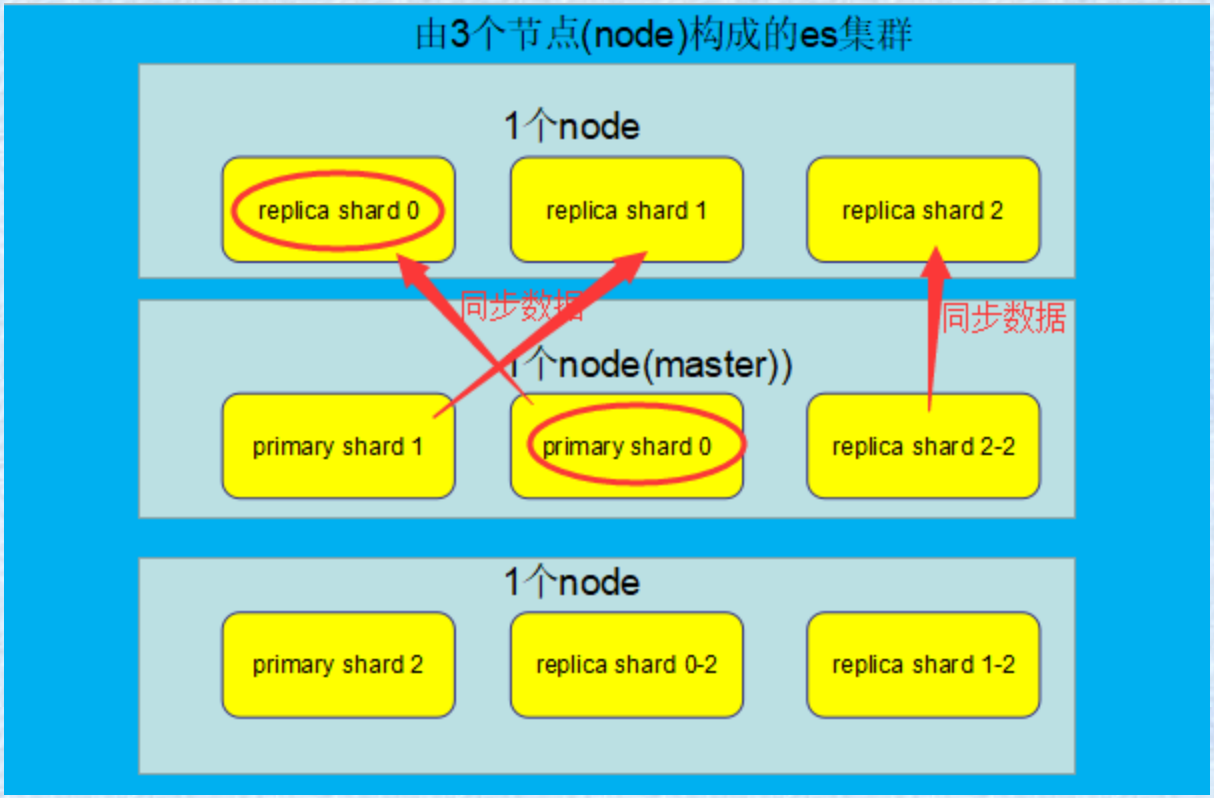


◆ 容错第二步：新master将丢失的primary shard的某个replica提升为primary



◆ 容错第三步：**重启故障node, new master节点会把缺失的副本都copy一份到该node上去**

而且该node会使用之前已有的shard数据，只是同步一下宕机之后发生的改变



Thank you!