

[VIP课程]

ElasticSearch分布式 高级特性

做技术人的指路明灯，职场生涯的精神导师！

➤ Tom老师QQ号：441221062



01

分布式特性

分布式特性

es支持集群模式，是一个分布式系统，其好处主要有两个：

增大系统容量，如内存，磁盘，使得es集群可以支持PB级的数据

提高系统可用性，即使部分节点停止服务，整个集群依然可以正常服务

es集群由多个es实例组成

不同集群通过集群名字来区分，可通过**cluster.name**来进行修改，默认为**elasticsearch**

每个es实例本质上是一个JVM进程，且有自己的名字，可以通过**node.name**来进行修改

可视化插件：

Elasticsearch-head : <https://github.com/mobz/elasticsearch-head>

Cerebro : <https://github.com/lmenezes/cerebro/releases>

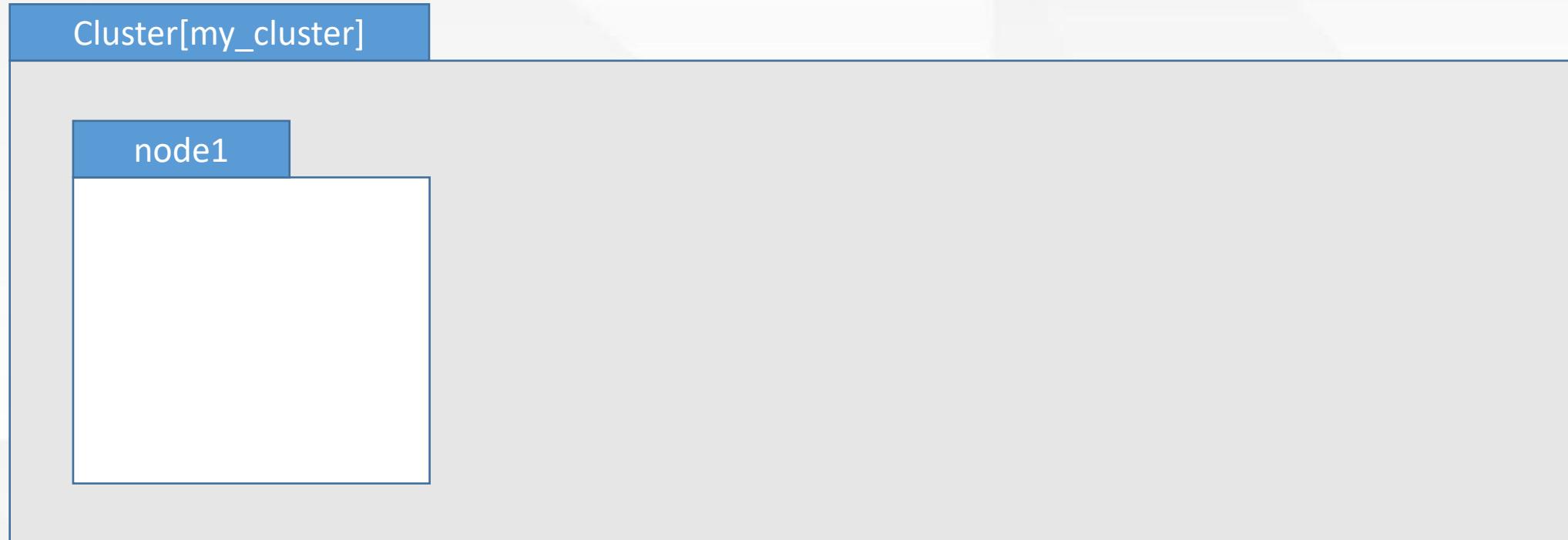


节点启动

运行如下命令可以快速启动一个es节点的实例

```
bin/elasticsearch
```

```
-Ecluster.name=my_cluster -Epath.data=my_cluster_node1 -Enode.name=node1  
-Ehttp.port=5100 -d
```



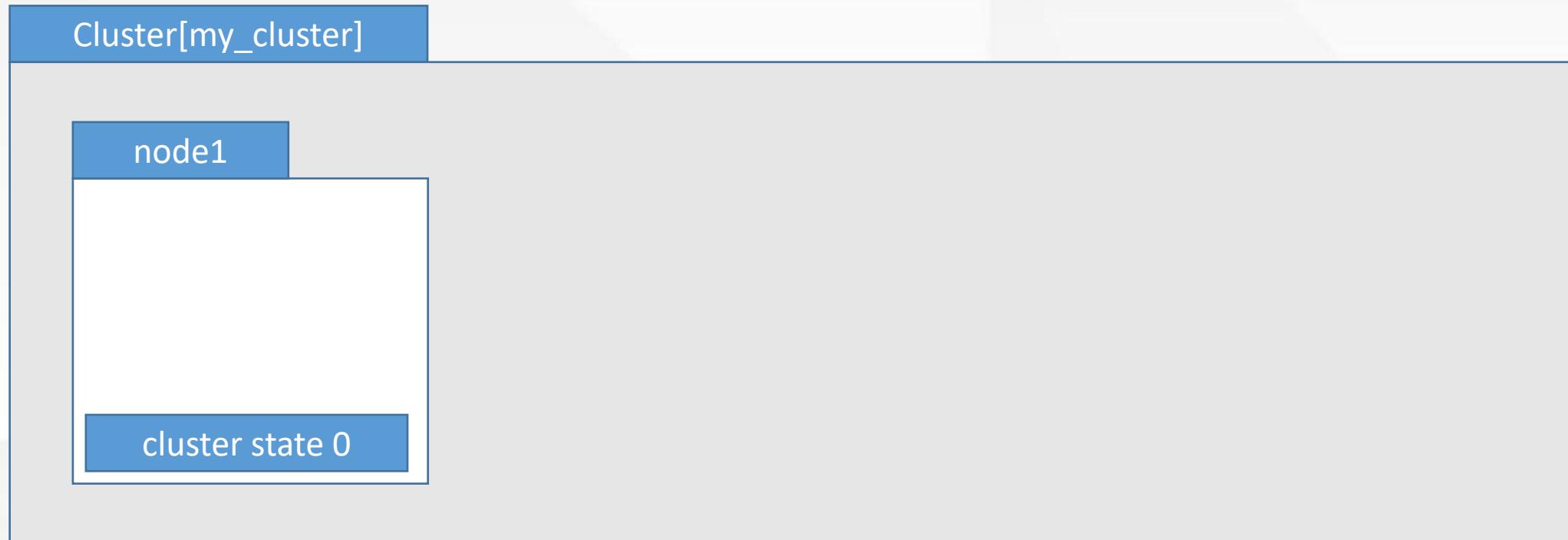
Cluster State

es 集群相关的数据称为 cluster state, 主要记录如下信息 :

节点信息 , 比如节点名称、链接地址等

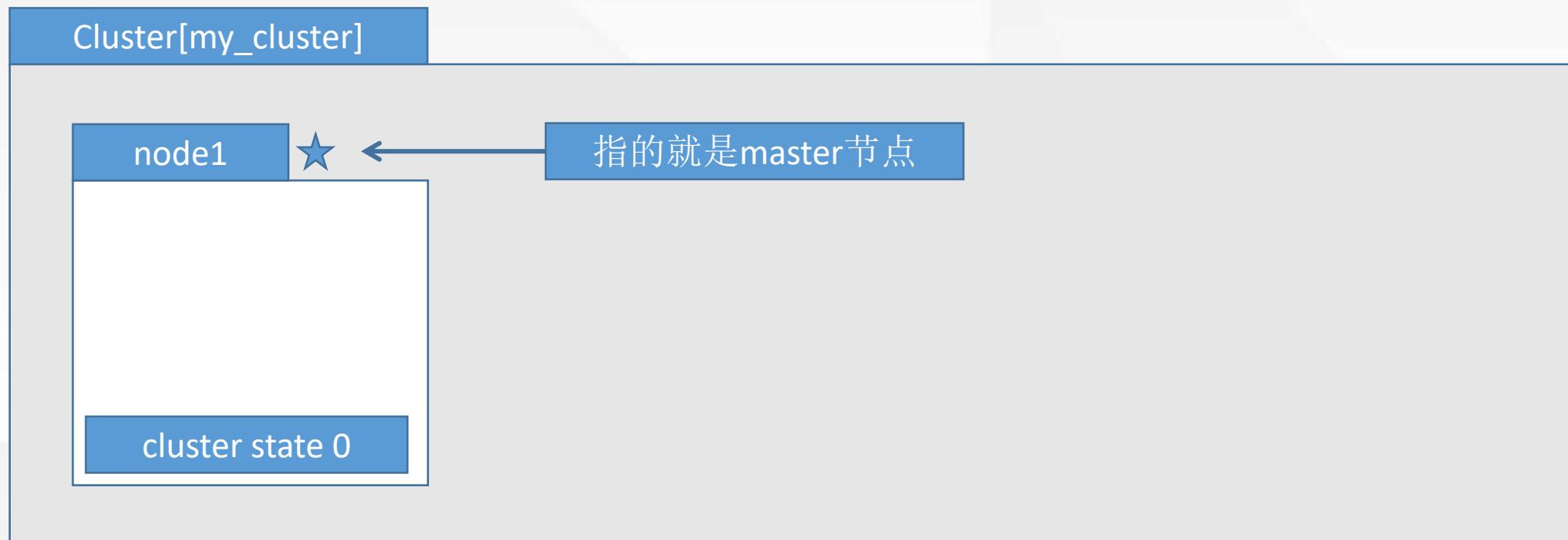
索引信息 , 比如索引名称、配置等

.....



Master Node

- 1、可以修改cluster state的节点称为master节点，一个集群只能有一个
- 2、cluster state存储在每个节点上，master维护最新版本并同步给其他节点
- 3、master 节点是通过集群中所有节点选取产生的，可以被选举的节点称为master-eligible节点
，相关配置：**node.master:true**



创建一个索引

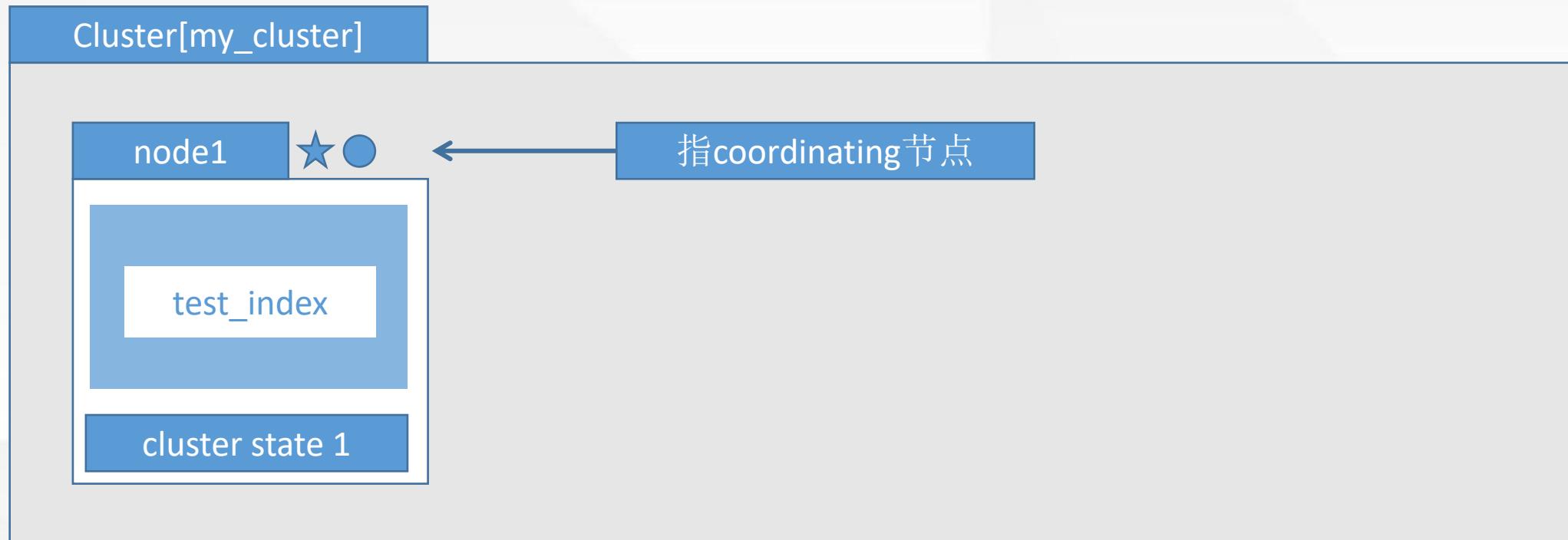
我们通过如下api创建一个索引

PUT test_index



处理请求 Coordinating Node

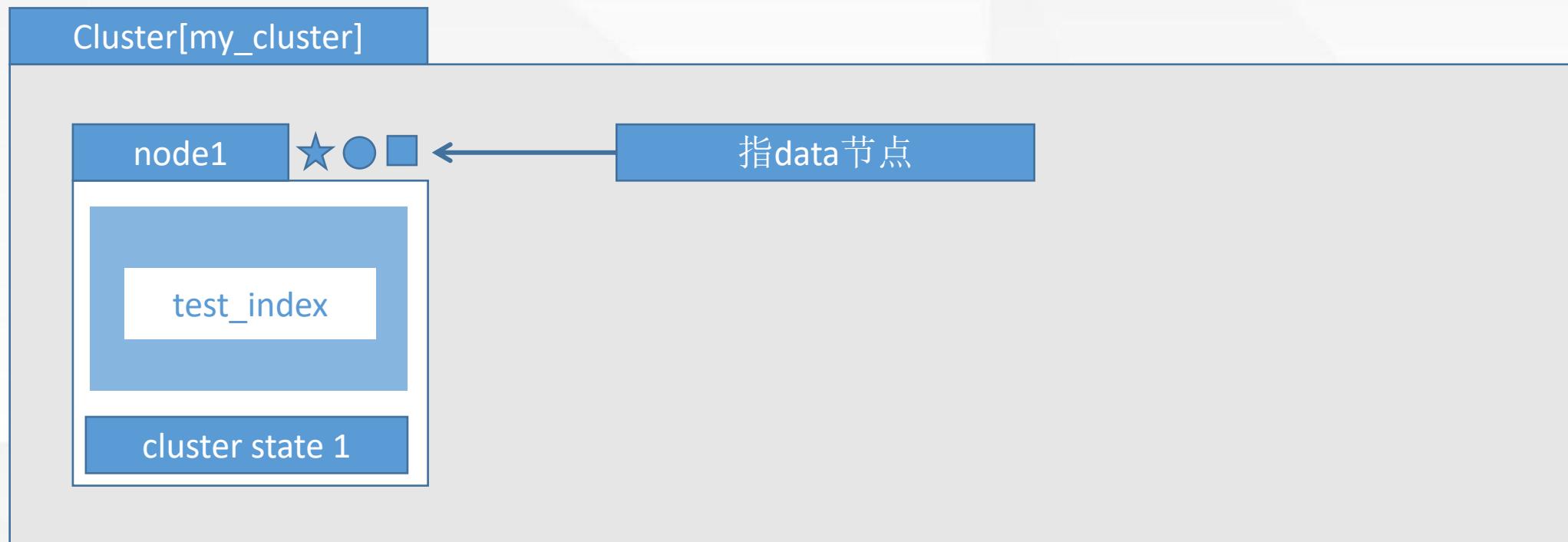
处理请求的节点称为**coordinating节点**，该节点为所有节点的默认角色，不能取消
路由请求到正确的节点处理，比如创建索引的请求到master节点



数据节点Data Node

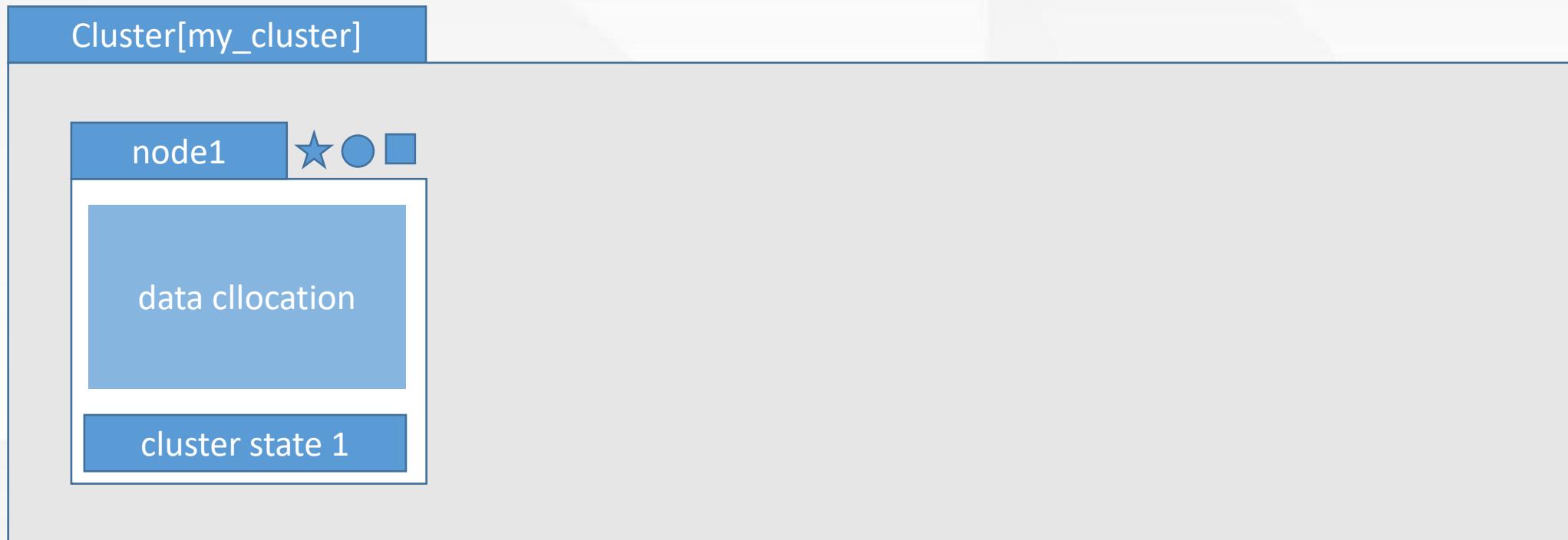
存储数据的节点称为 data节点，默认节点都是data类型，相关配置如下：

node.data:true



单点问题

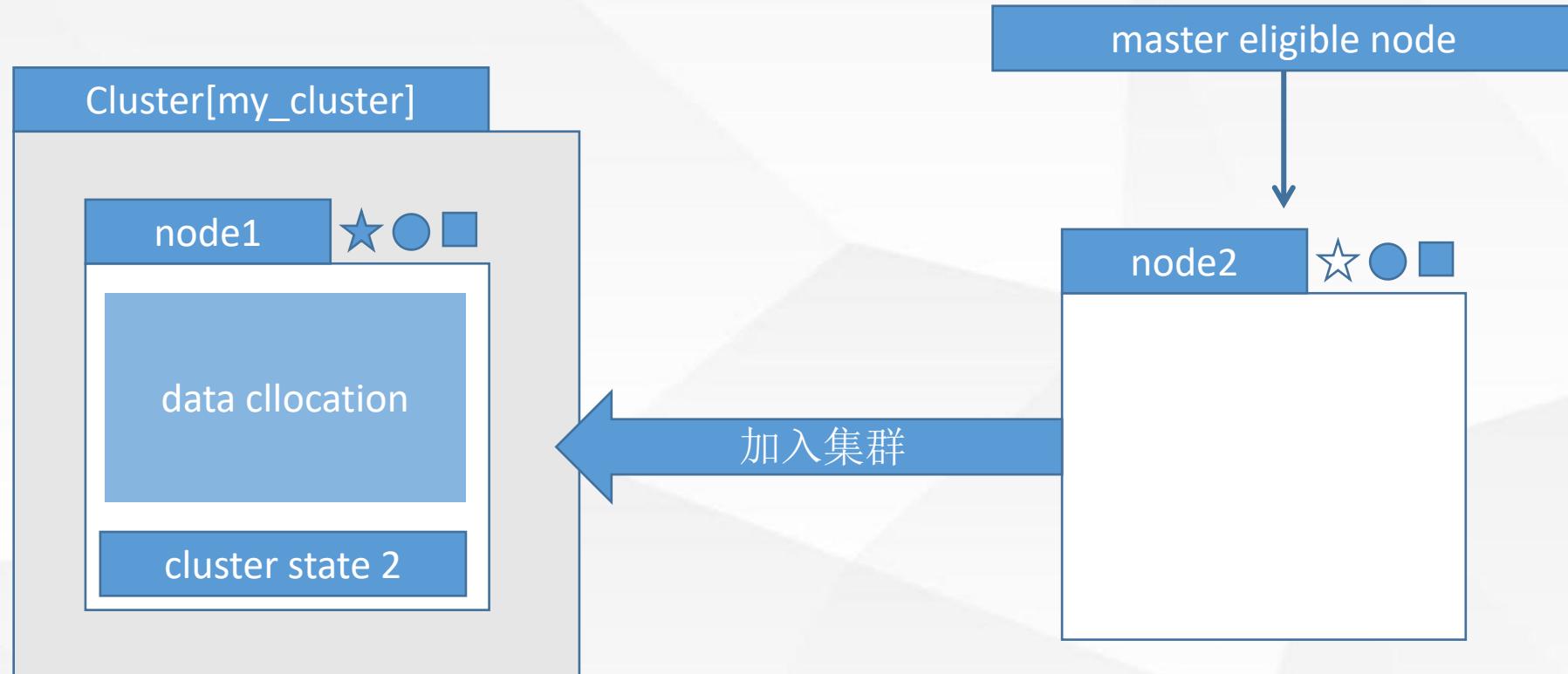
如果node1停止服务，集群就停止服务



新增节点

bin/elasticsearch

```
-Ecluster.name=my_cluster -Epath.data=my_cluster_node2 -Enode.name=node2  
-Ehttp.port=5200 -d
```

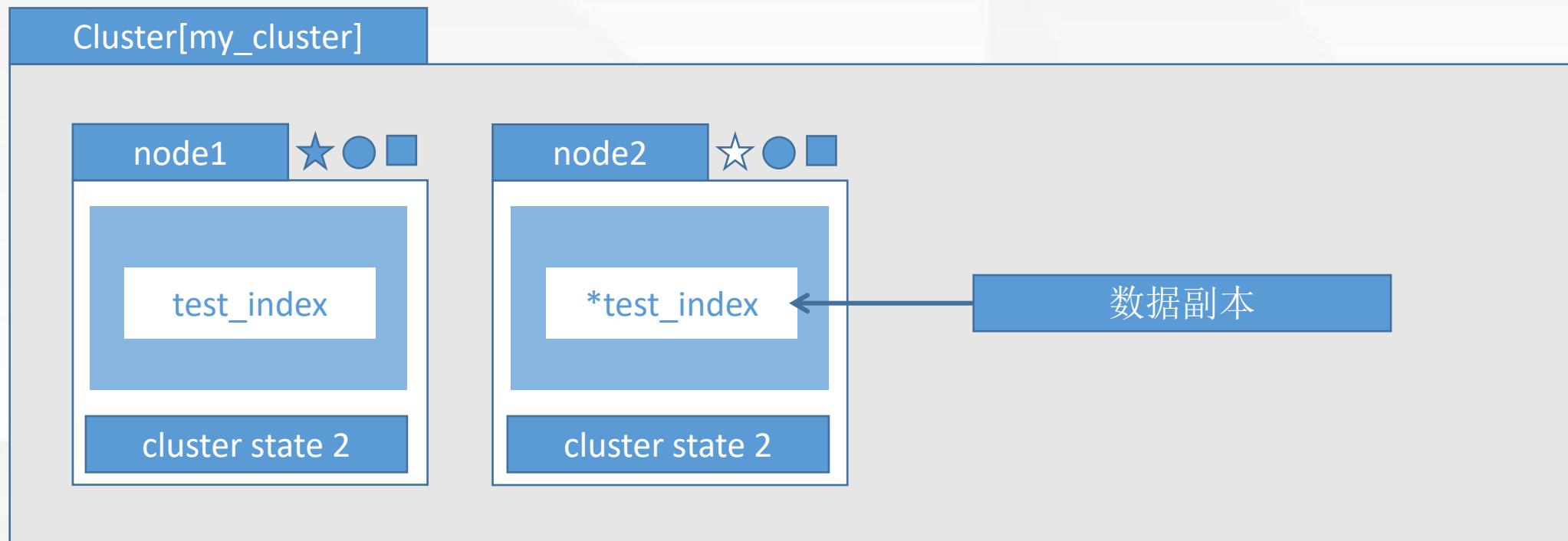


02

副本与分片

提高系统可用性

如下图所示，node2上是test_index的副本



数据扩容

如何将数据分布到所有节点上？

引入分片 (Shard) 解决问题

分片是es支持PB级数据的基石

分片存储了部分数据，可以分布于任意节点上

分片数在索引创建时指定且后续不允许再更改，默认为5个

分片有主分片和副本分片之分，以实现数据的高可用

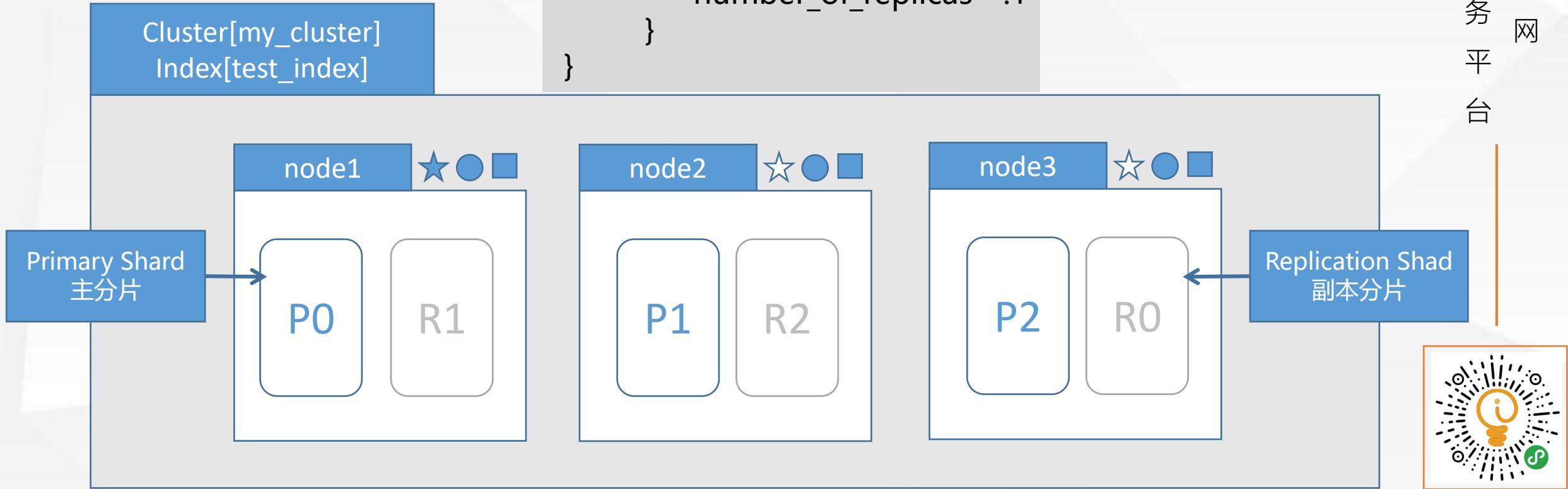
副本分片的数据由主分片同步，可以有多个，从而提高读取的吞吐量



分片

下图演示的是3个节点的集群中test_index的分片分布情况，创建时我们指定了3个分片和1个副本
api如下所示：

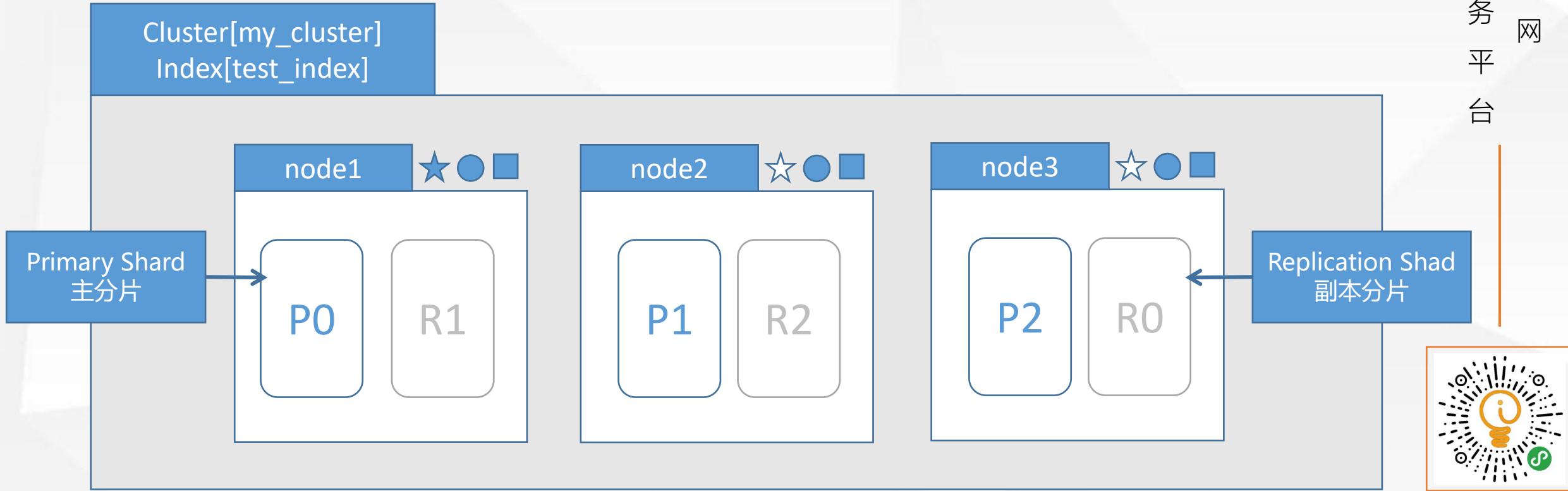
```
PUT test_index
{
  "settings" :{
    "number_of_shards":3,
    "number_of_replicas":1
  }
}
```



两个问题

此时增加节点是否能够提高test_index的数据容量吗？

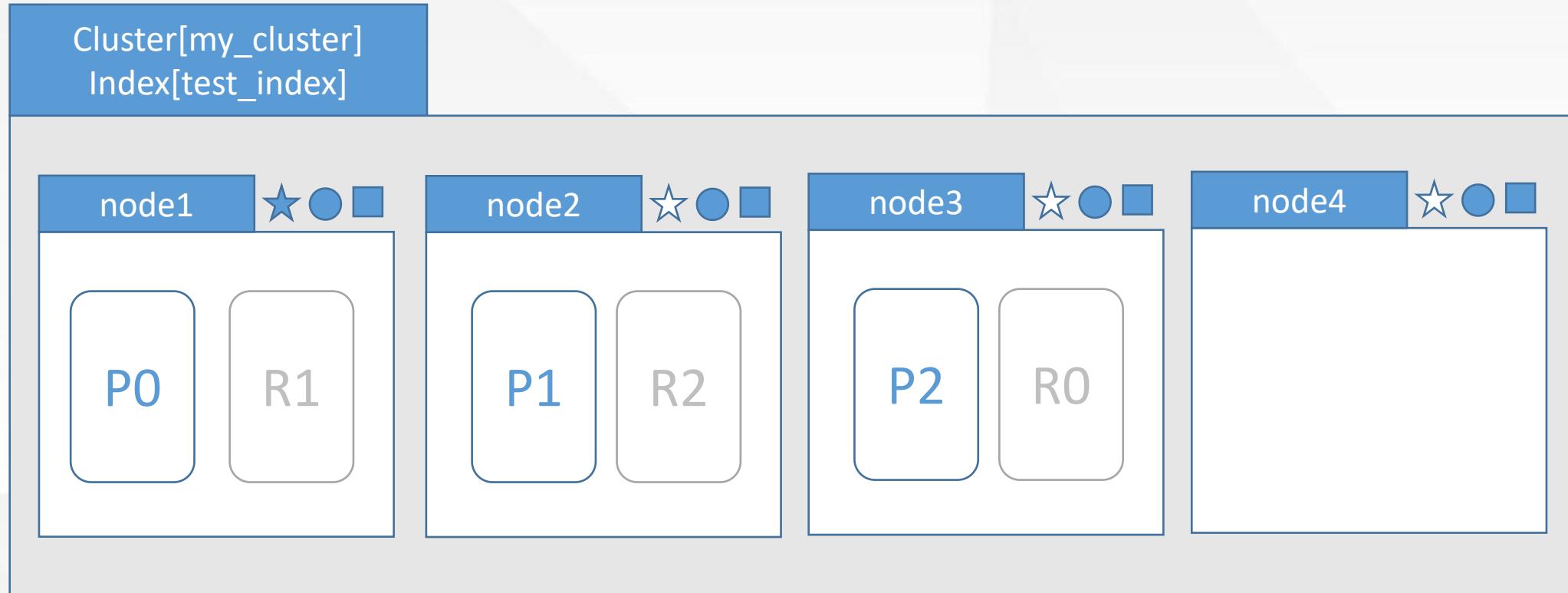
此时增加副本数是否能提高test_index的读取吞吐量呢？



两个问题

此时增加节点是否能够提高test_index的数据容量吗？

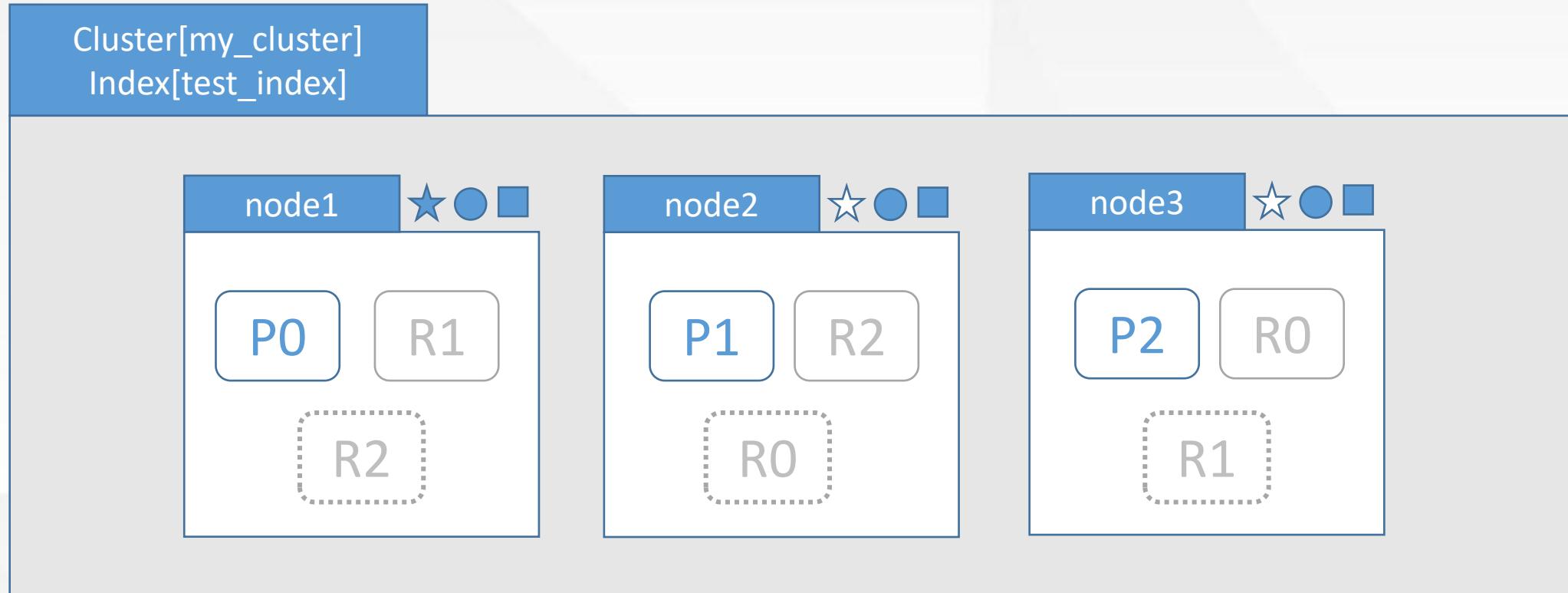
不能，因为只有3个分片，已经分布到3个节点上，新增的节点无法利用



两个问题

此时增加副本数是否能提高test_index的读取吞吐量呢？

不能，因为新增的副本是分布在3个节点上，还是利用了同样的资源，
如果要增加吞吐量，还需要增加节点。



如何解决这两个问题

分片数的设定非常重要，需要提前规划好

分片数太少，导致后续无法通过增加节点实现水平扩容

分片数过大，导致一个节点上分布多个分片，造成资源浪费，同时会影响查询性能



03

集群状态

Cluster Health

通过如下api可以查看集群健康状况，包括以下三种：

green 健康状态，指所有主副分片都正常分配

yellow 指所有主分片都正常分配，但是有副本分片未正常分配

red 有主分片未分配

三种状态只是代表分片的工作状态，并不是代表整个es集群是否能够对外提供服务

GET _cluster/health

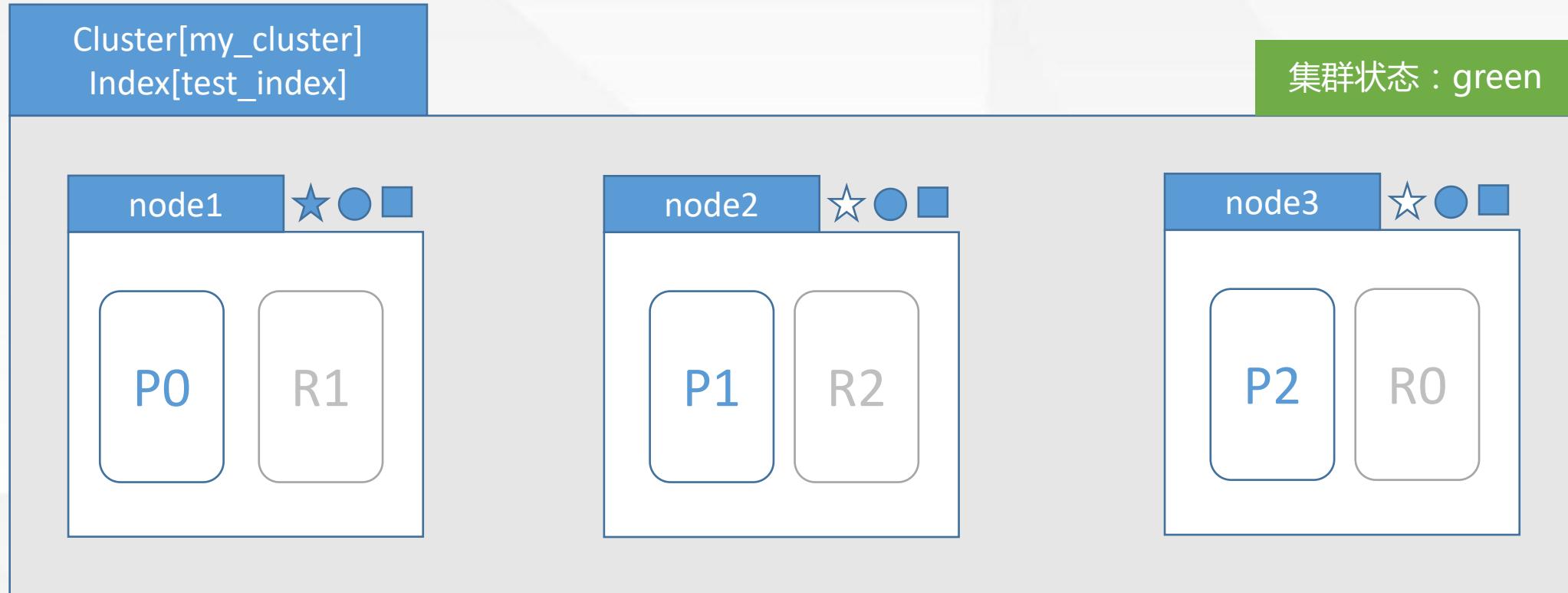


04

故障转移

故障转移

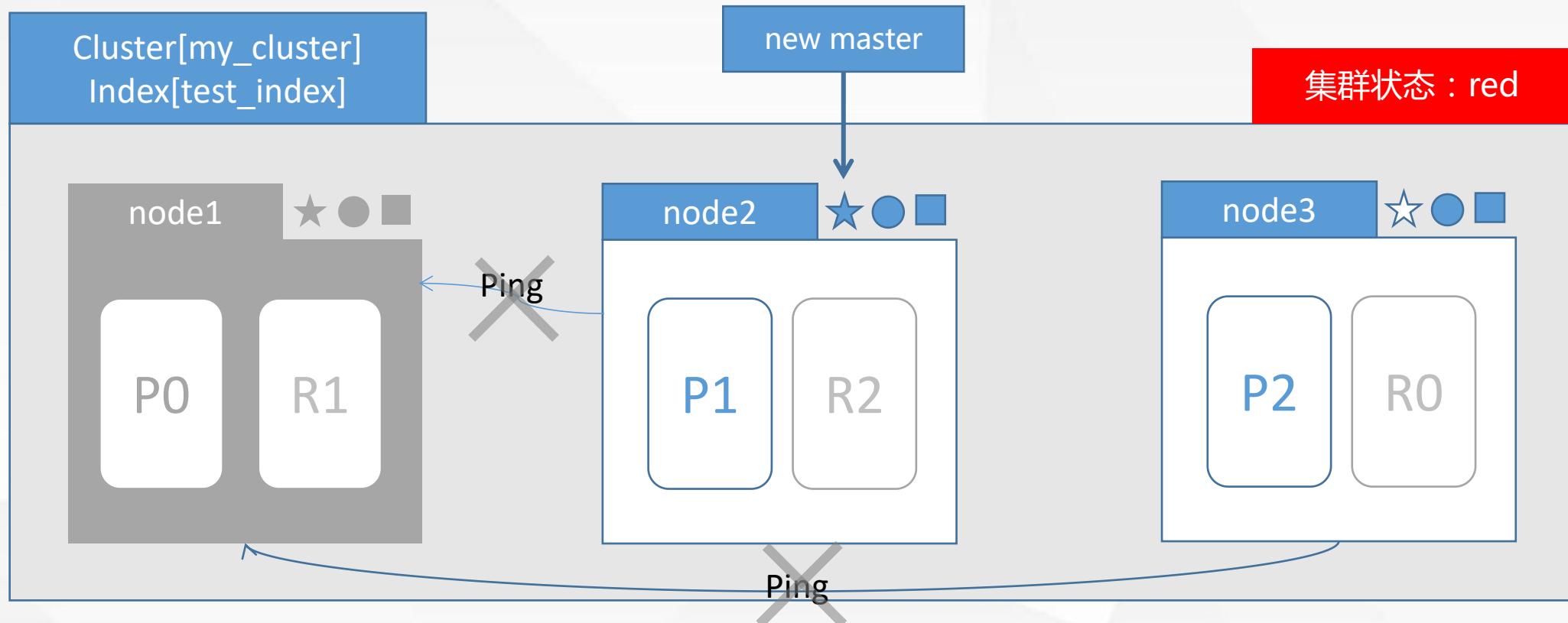
集群由3个节点组成，如下所示，此时集群状态是green



故障转移

node1所在机器宕机导致服务终止，此时集群会如何处理？

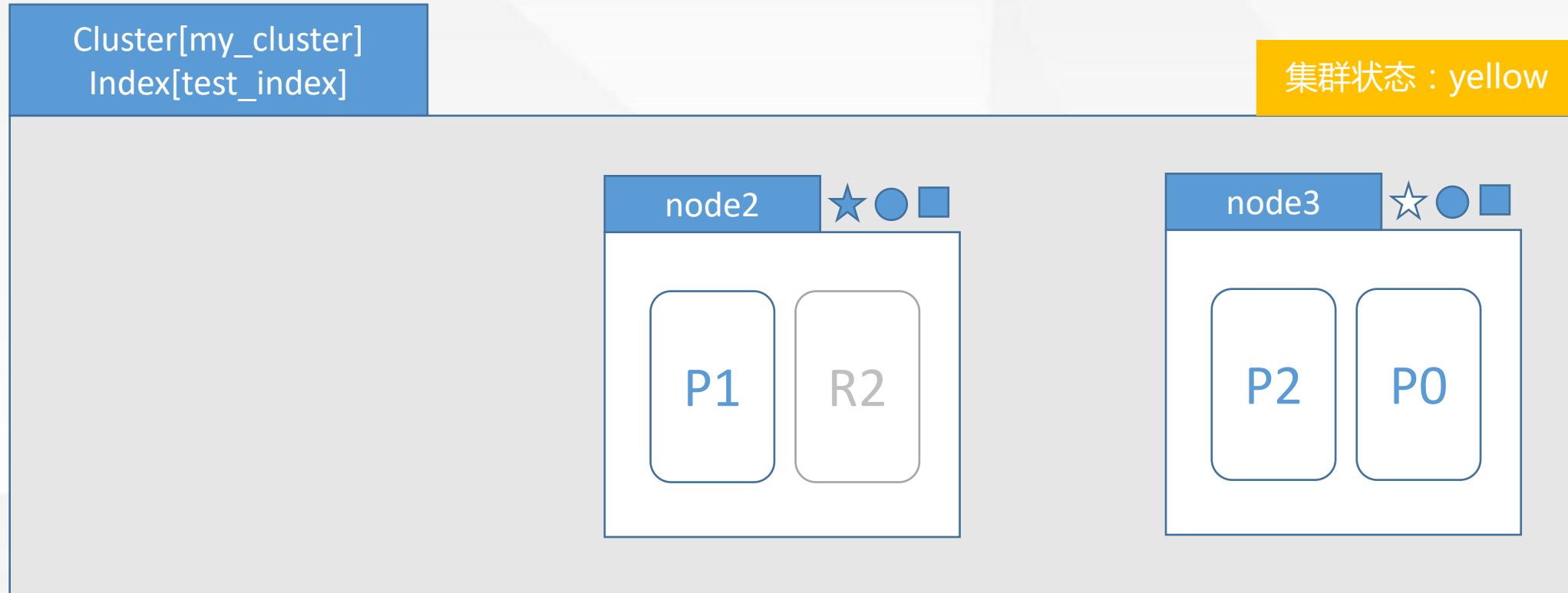
- 1、node2和node3发现node1无法响应一段时间后会发起master选举，
比如这里选举node2为master节点，此时由于主分片P0下线，集群状态变为red。



故障转移

node1所在机器宕机导致服务终止，此时集群会如何处理？

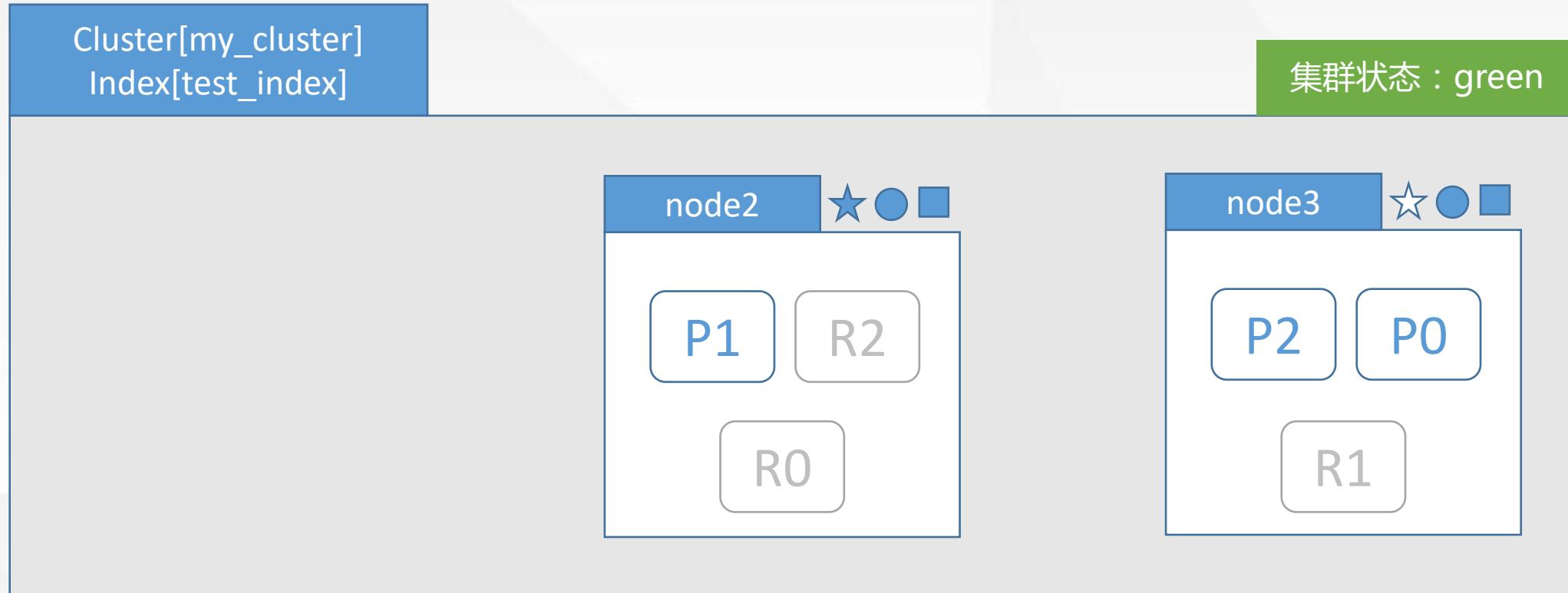
2、node2发现主分片P0未分配，将R0提升为主分片。此时由于所有主分片都正常分配，集群状态变为yellow。



故障转移

node1所在机器宕机导致服务终止，此时集群会如何处理？

3、node2发现主分片P0和P1生成新的副本，集群状态变为green。



05

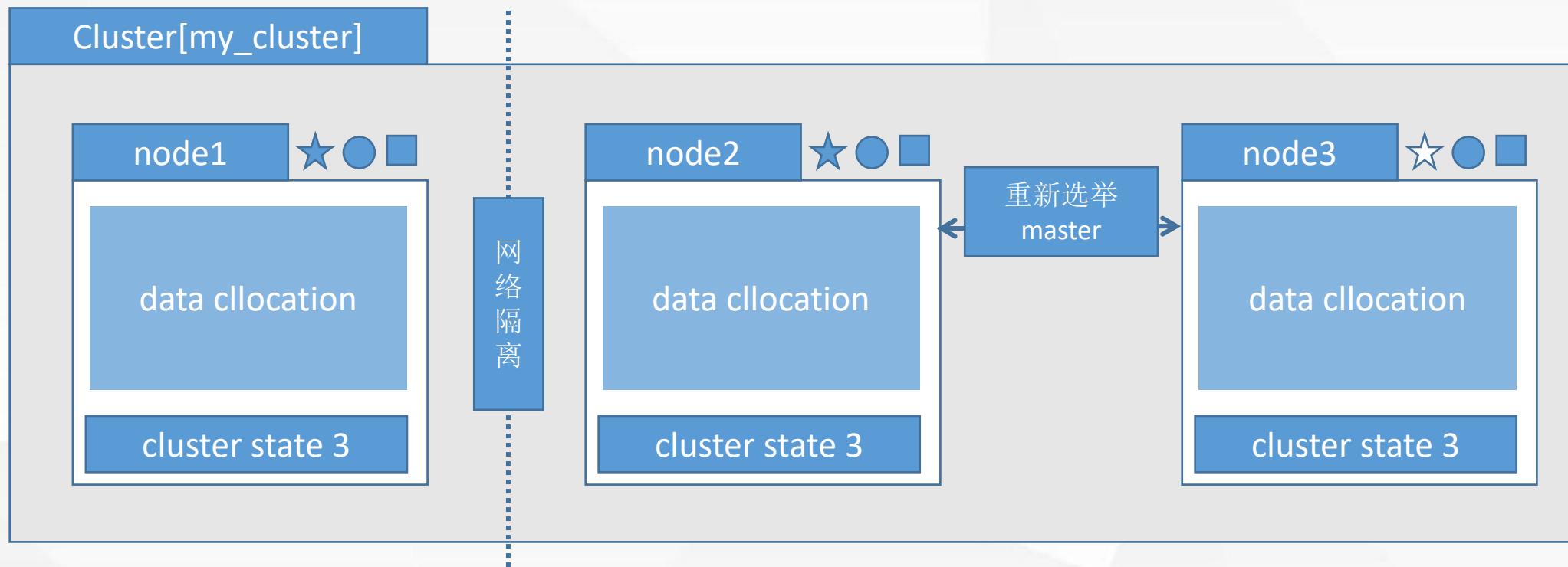
脑裂问题

脑裂问题

脑裂问题，英文为split-brain，是分布式系统中的经典网络问题，如下图所示：

node2与node3会重新选举master，比如node2成为了新master，此时会更新cluster state
node1自己组成集群后，也会更新cluster state

同一个集群有两个master，而维护不同的cluster state，网络恢复后无法选择正确的master

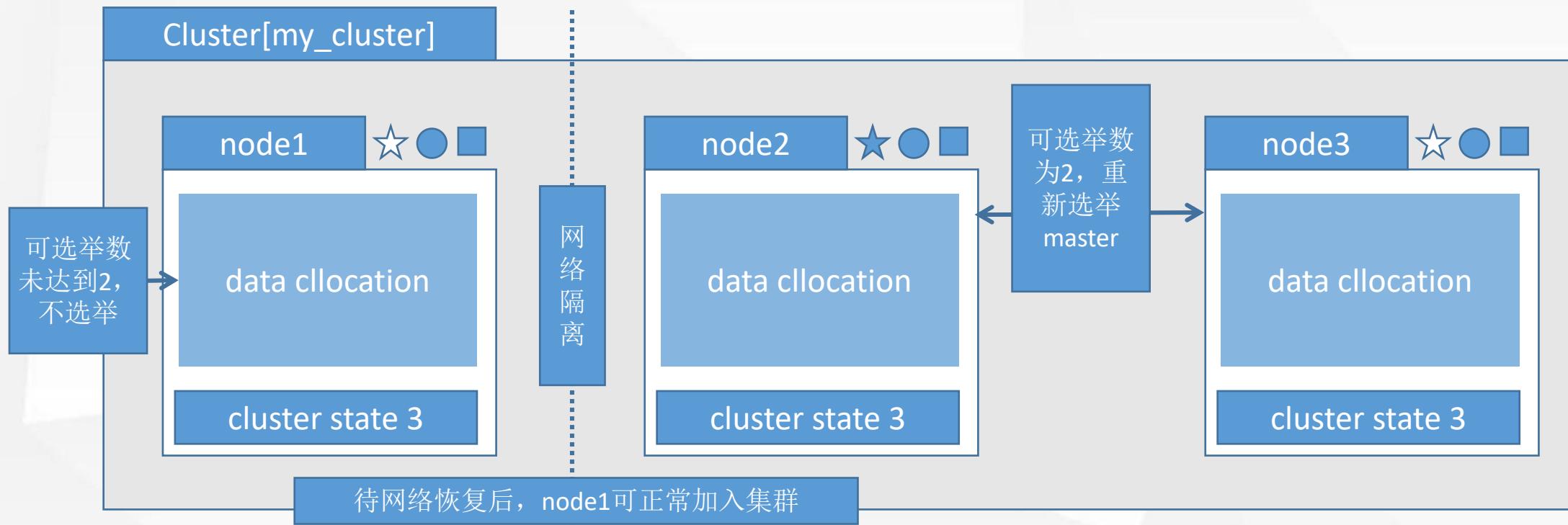


脑裂问题：解决方案

解决方案为仅在可选举master-eligible节点数据大于等于quorum时才可以进行master选举

$quorum = \text{master-eligible 节点数}/2 + 1$ ，例如 3个master-eligible节点时，quorum为2。

解决：`discovery.zen.minimum_master_nodes`为quorum即可避免脑裂



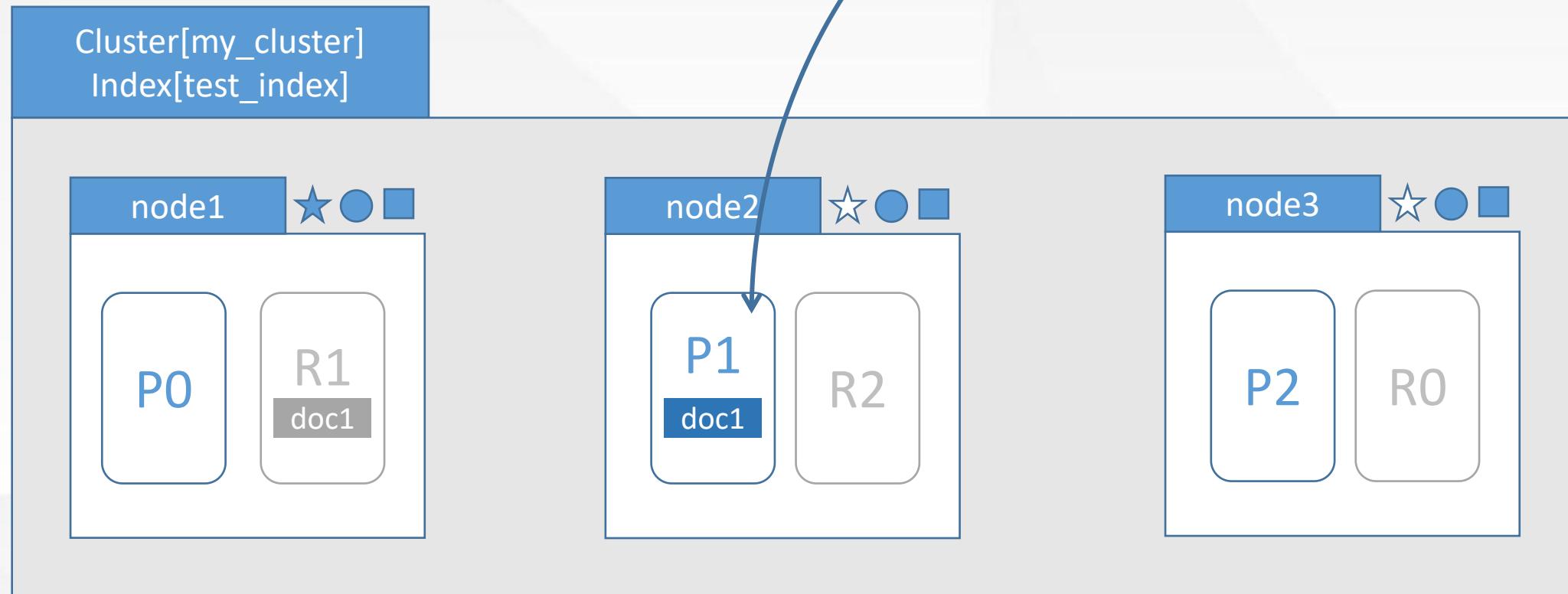
06

分布式存储

分布式文件存储

文档最终会存储在分片上，如图所示：

假设Doc1最终存储在分片P1上



分布式文件存储

Document1是如何存储到分片P1的呢？选择P1的依据是什么呢？

需要文档到分片的映射算法

目的

使得文档均匀分布在所有分片上，以充分利用资源

算法

随机选择或者round-robin算法

不可取：因为维护文档到分片的映射关系，成本巨大



文档到分片的映射算法

es通过如下公式计算文档对应的分片

```
shard = hash(routing) % number_of_primary_shards
```

hash算法保证可以将数据均匀地分散在分片中

routing是一个关键参数，默认是文档id，也可以自行指定

number_of_primary_shards 是主分片数

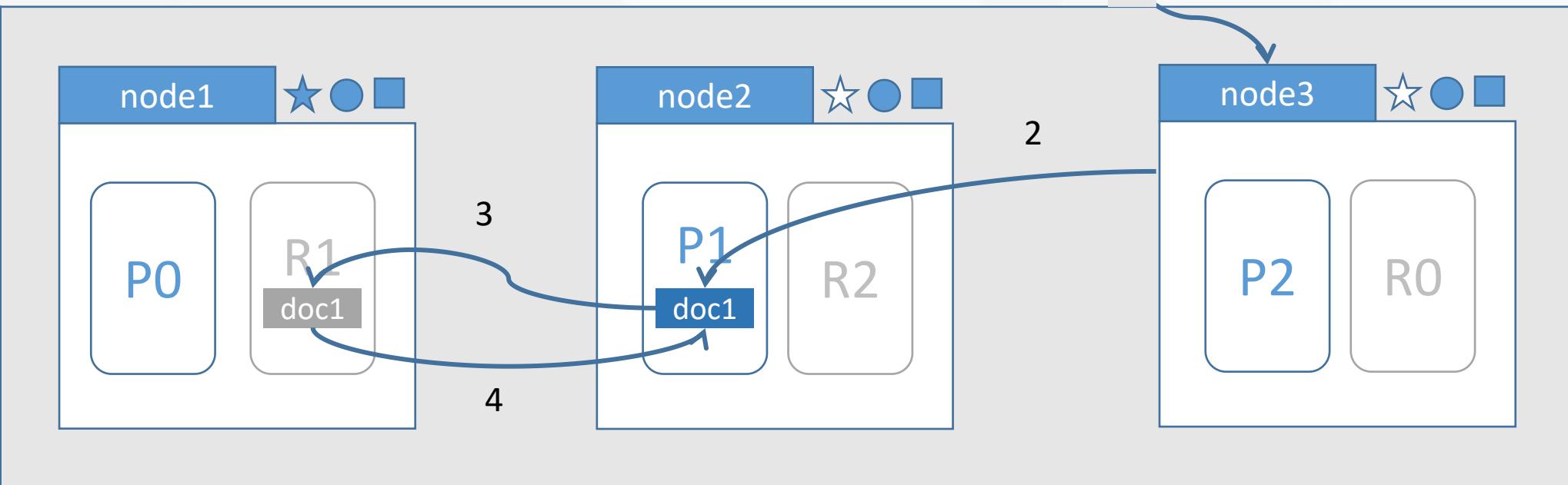
该算法与主分片数相关，这也是**分片数一旦确定后便不能更改**的根本原因



文档创建的流程

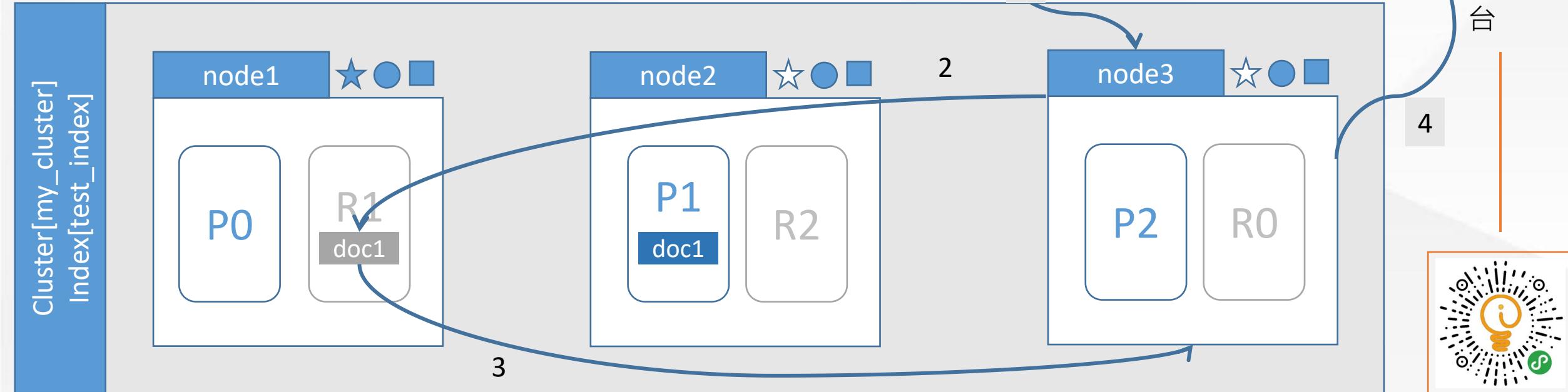
- 1、Client向**node3**发起创建文档的请求
- 2、**node3**通过routing计算该文档应该存储在**Shard1**上，查询cluster state后确认主分片**P1**在**node2**上，然后转发创建文档的请求到**node2**
- 3、**P1**接收并执行创建文档的请求后，将同样的请求发送到副本分片**R1**
- 4、**R1**接收并执行创建文档请求后，通知**P1**成功的结果

```
PUT test_index/doc/1
{
  "name" : "tom",
  "age" : 18
}
```



文档读取的流程

- 1、Client向**node3**发起创建文档的请求
- 2、**node3**通过routing计算该文档应该存储在**Shard1**上，查询cluster state后获取 **Shard 1**的主副分片列表，然后以轮询的机制获取一个shard，比如这里是**R1**,然后转发读取文档的请求到**node1**
- 3、**R1**接收并执行创建文档的请求后，将结果返回给**node3**
- 4、**node3**返回结果给Client



07

Shard分片

倒排索引不可变更

倒排索引一旦生成，不能更改

有如下好处：

不用考虑并发写文件的问题，杜绝了锁机制带来的性能问题

由于文件不再更改，可以充分利用文件系统缓存，只需要载入一次，只要内存足够，

对该文件的读取都会从内存读取，性能高

利于生成缓存数据

利于对文件进行压缩存储，节省磁盘和内存存储空间

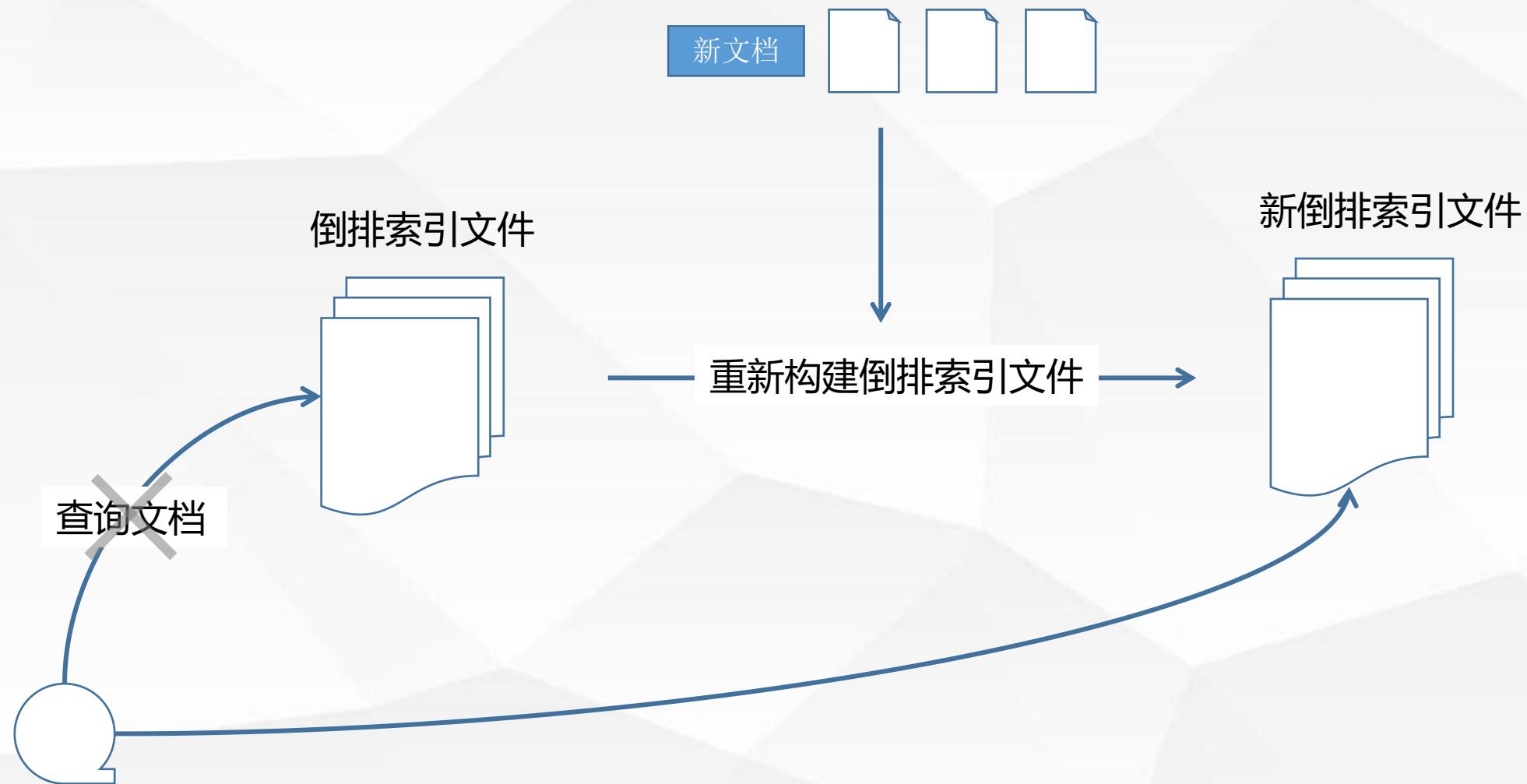
坏处：

写入新文档时，必须重新构建倒排索引文件，然后替换老文件后，新文档才能被检索，

导致文档实时性受到影响



倒排索引不可变更



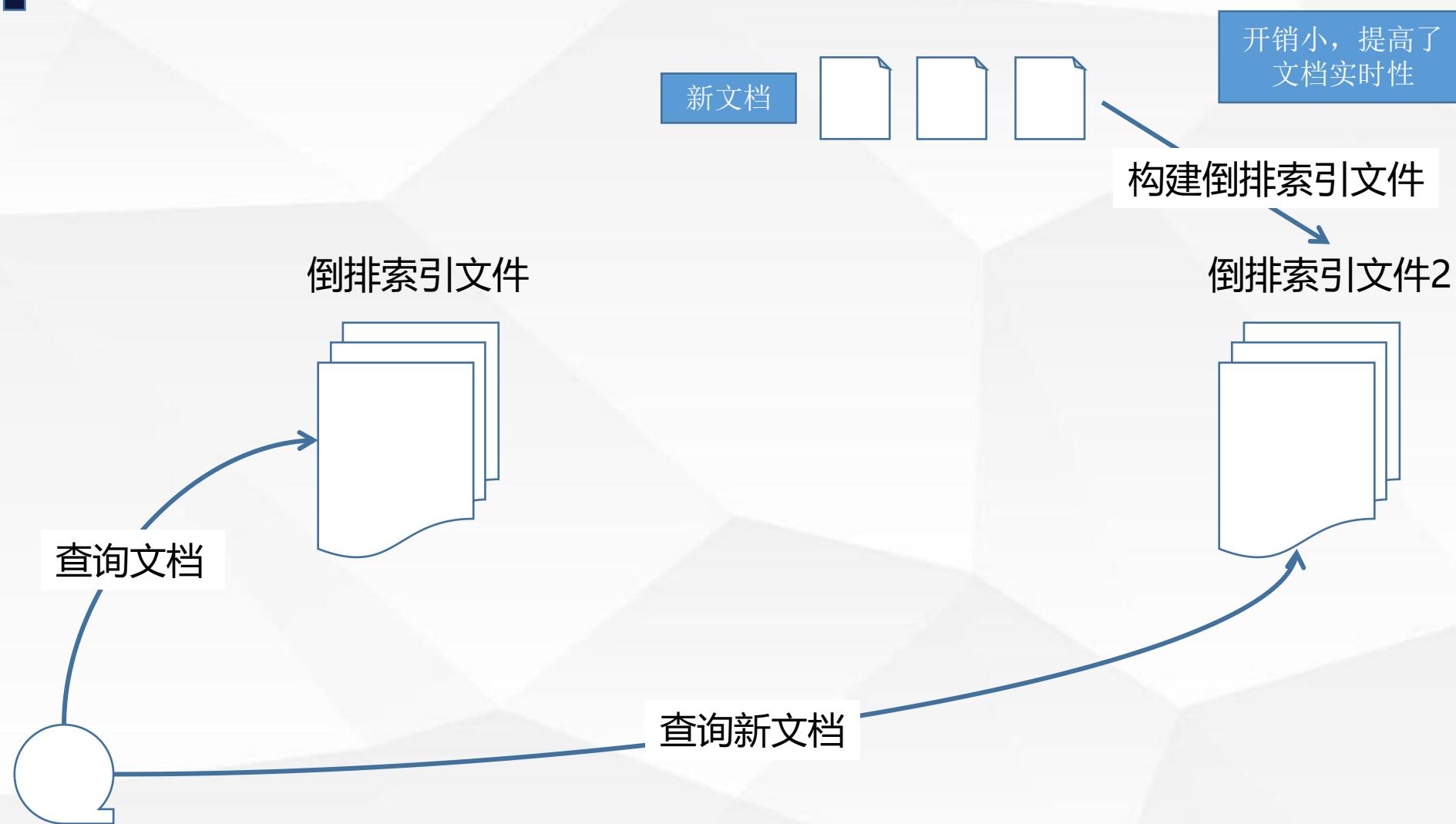
文档搜索实时性

解决方案：

新文档直接生成新的倒排索引文件，查询的时候同时查询所有的倒排文件，然后对查询结果做汇总计算即可



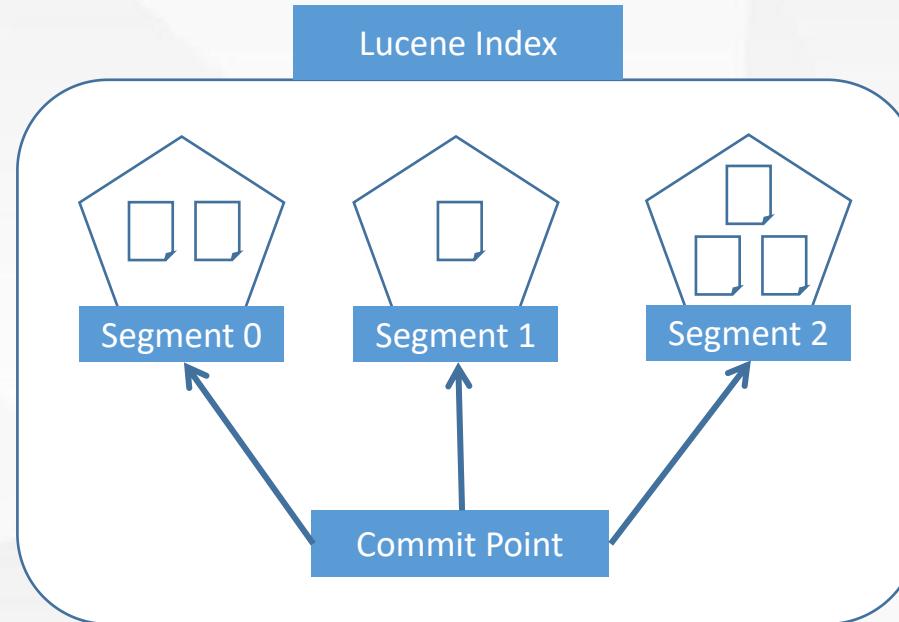
文档搜索实时性



文档搜索实时性

Lucene采用了这种方案，它构建的单个倒排索引称为**segment**，合在一起称为**Index**，与ES中的Index概念不同。ES中的一个**Shard**对应一个Lucene Index。

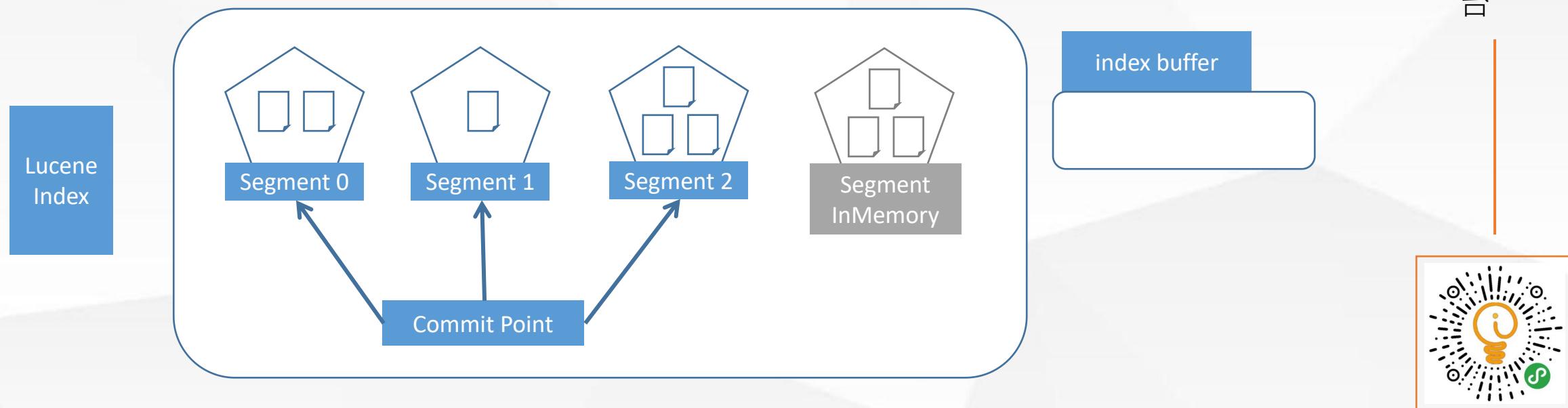
Lucene会有一个专门的文件来记录所有的**segment**信息，称为**Commit Point**



文档搜索实时性 - refresh

segment写入磁盘的过程依然很耗时，可以借助文件系统缓存的特性，先将segment在缓存中创建并开放查询来进一步提升实时性，该过程在es中被称为refresh。

在refresh之前文档会先存储在一个buffer中，refresh时将buffer中的所有文档清空并生成segment
es默认每1秒执行一次refresh，因此文档的实时性被提高到1秒，
这也是es被称为近实时（Near Real Time）的真正原因



文档搜索实时性 - translog

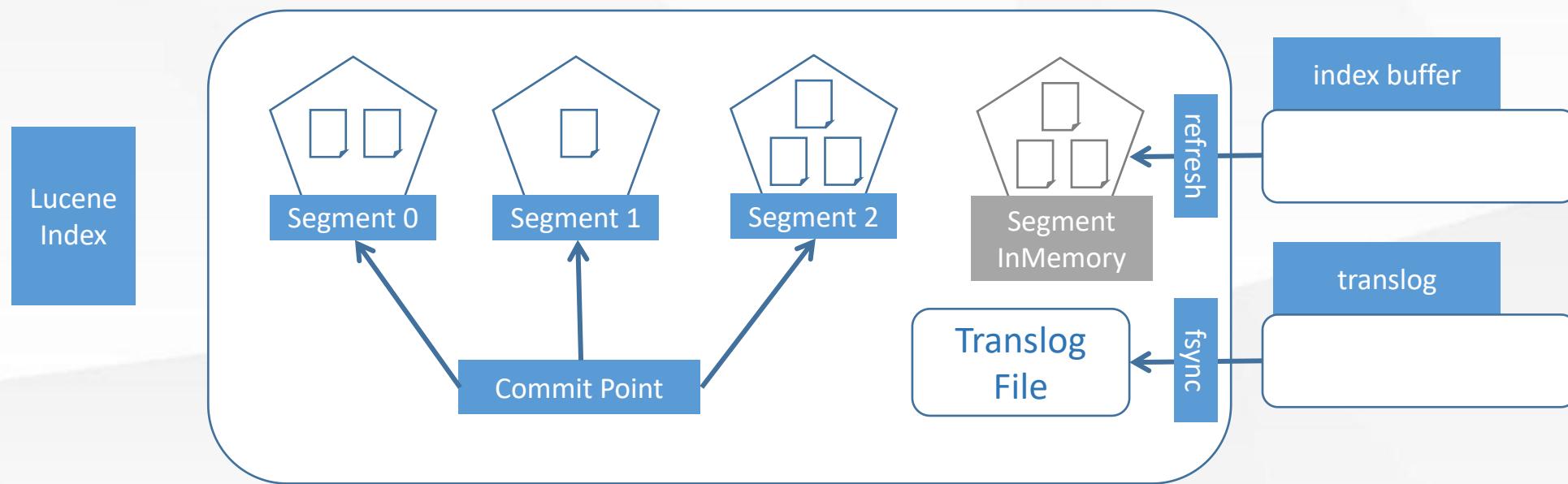
如果在内存中的segment还没有写入磁盘前发生了宕机，那么内存中的文档就无法恢复了。

那么如何解决这个问题呢？

es引入**translog**机制。写入文档到**buffer**时，同时将该操作写入**translog**。

translog 文件会即时写入磁盘（`fsync`），6.x默认每个请求都会落盘，可以修改为每5秒写一次，这样风险便是丢失5秒内的数据，相关配置为`index.translog.*`

es重新启动时会自动检查**translog**文件，并从中恢复数据



文档搜索实时性 - flush

flush负责将内存中的segment写入磁盘，主要做如下的工作：

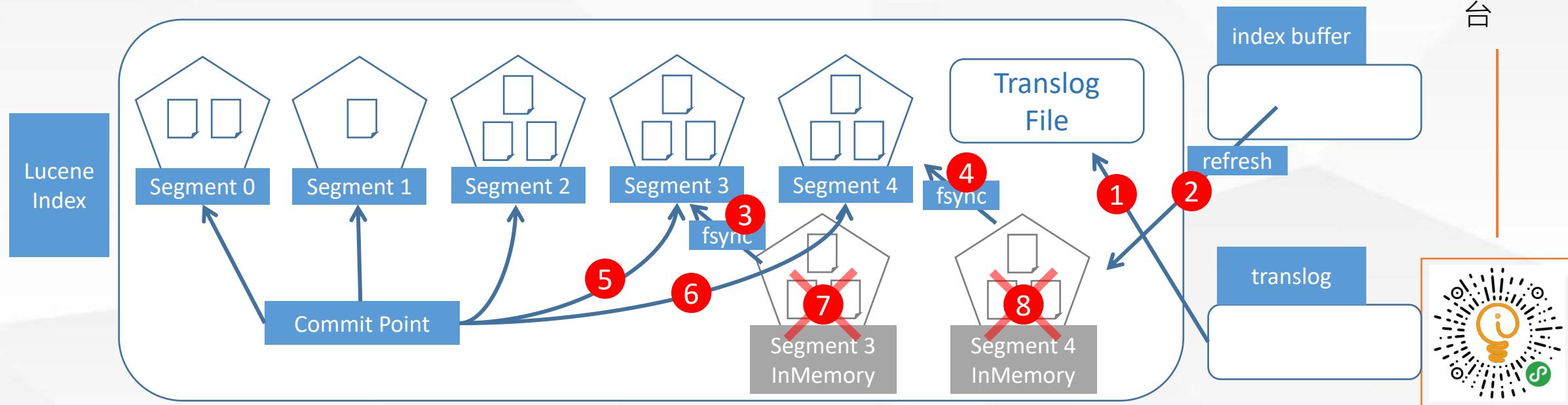
将translog写入磁盘

将index buffer清空，其中的文档生成一个新的segment，相当于一个refresh操作

更新commit point并写入磁盘

执行fsync操作，将内存中的segment写入磁盘

删除旧的translog文件



文档搜索实时性 - `reflush`

refresh发生的时机主要有以下几种情况：

间隔时间达到时，通过`index.settings.refresh_interval`来设定，默认是1秒

`index.buffer`占满时，其大小通过`indices.memory.index_buffer_size`设置，
默认为jvm heap的10%，所有shard共享

`flush`发生时也会发生`refresh`



文档搜索实时性 - flush

flush发生的时机主要有以下几种情况：

间隔时间达到时，默认是30分钟，5.x之前可以通过**index.translog.flush_threshold_period**修改
之后发布的版本无法设置

translog占满时，其大小可以通过**index.translog.flush_threshold_size**控制，默认是512MB
每个index有自己的translog



文档搜索实时性 - 删除与更新

segment一旦生成就不能更改，那么如果要删除文档该如何操作？

lucene会专门维护一个.del的文件，记录所有已经删除的文档，
注意.del上记录的是文档在Lucene的内部id

在查询结果返回前会过滤掉.del中所有的文档

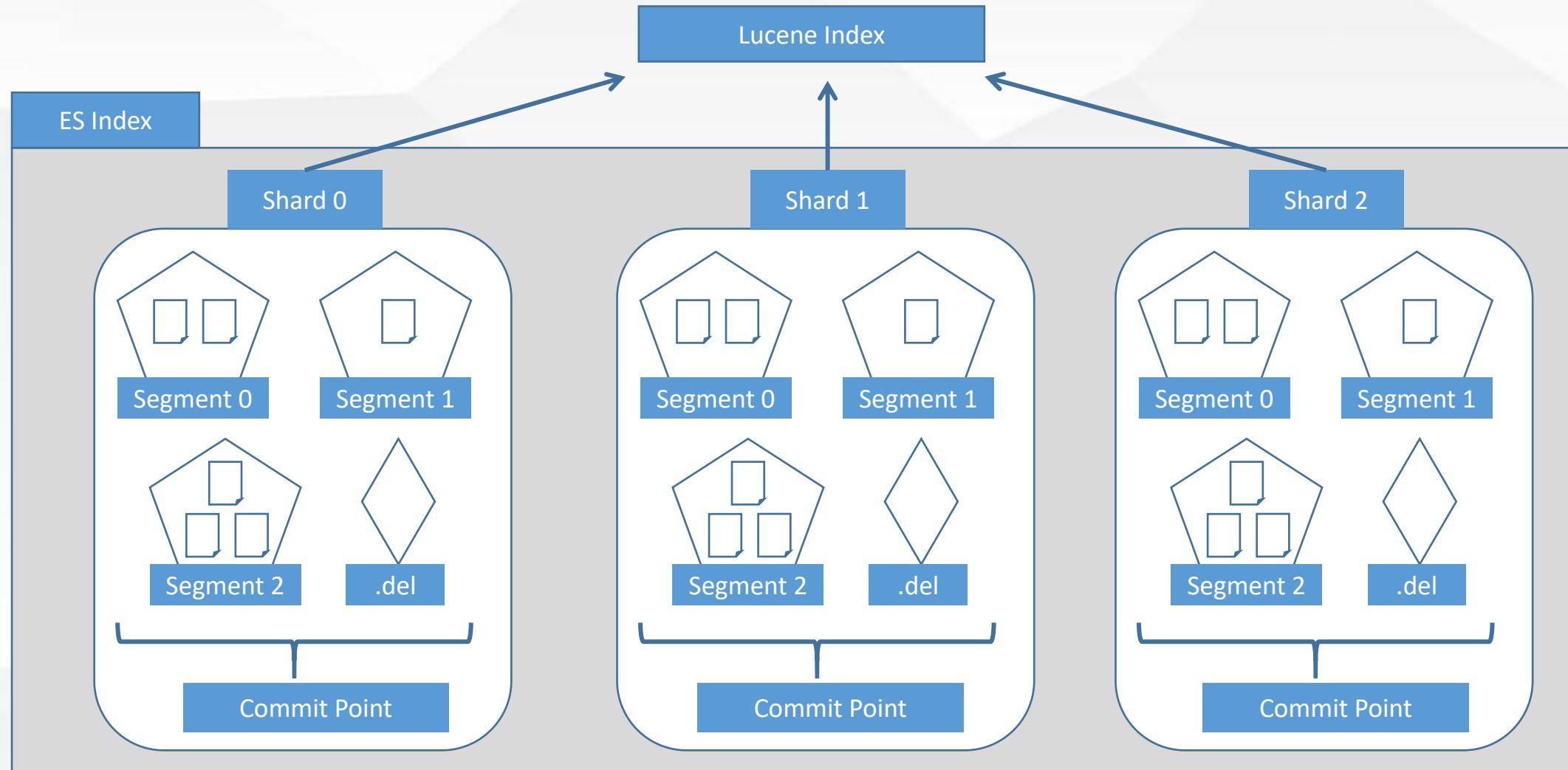
更新文档如何进行呢？

首先删除文档，然后再创建新的文档



整体视角

ES Index 与 Lucene Index 的术语对照如下所示：



Segment Merging

随着segment的增多，由于一次查询的segment数增多，查询速度会变慢
es会定时在后台进行segment merge的操作，减少segment的数量
通过force_merge api可以手动强制做segment merge的操作



08

Search的运行机制



Search的运行机制

Search执行的时候实际分两个步骤运行的

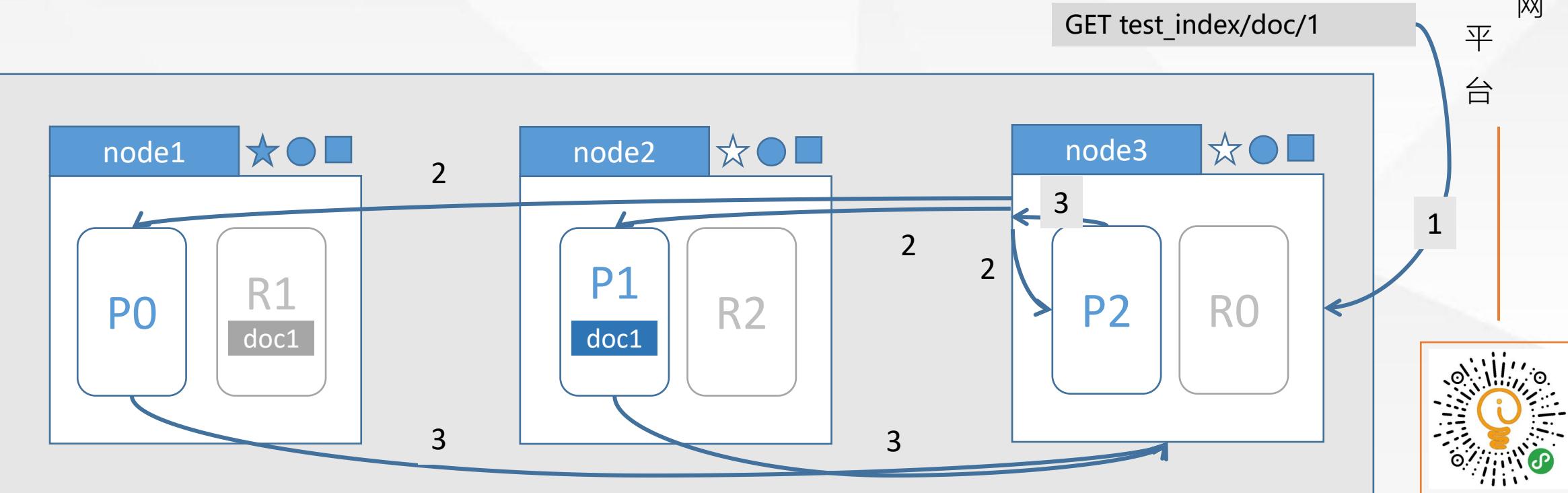
Query阶段

Fetch阶段

Query-Then-Fetch

Search的运行机制-Query阶段

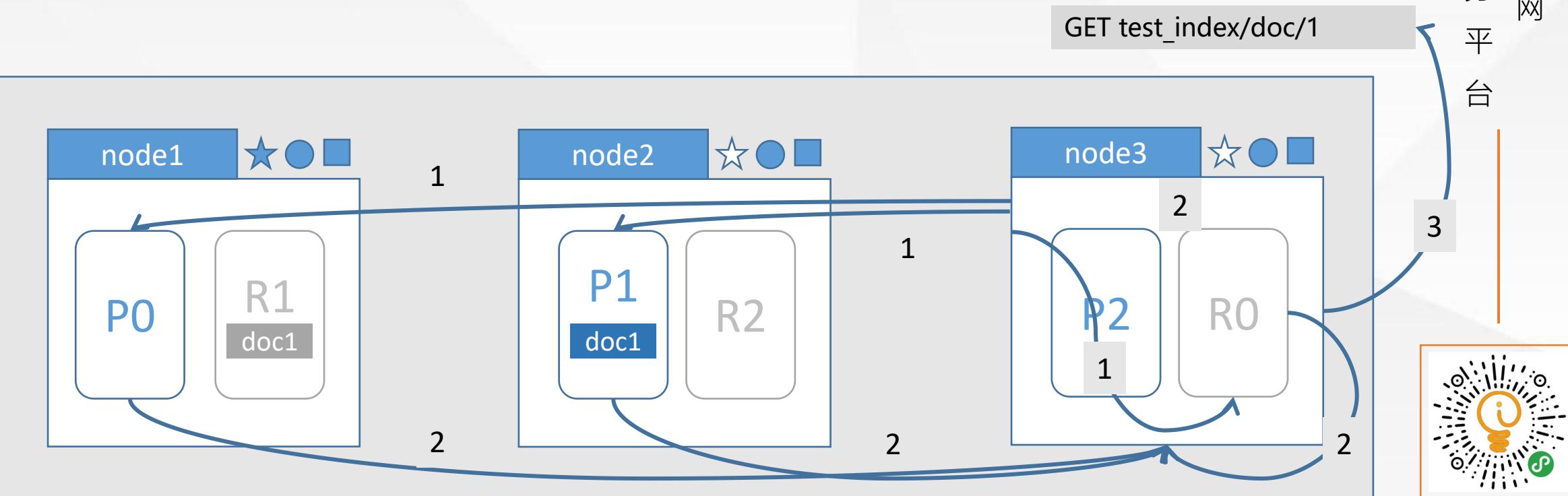
- 1、node3在接收到用户的search请求后，先会进行Query阶段（此时Coordinating Node角色）
- 2、node3在6个主副分片中随机选择3个分片，发送search request
- 3、被选中的3个分片会分别执行查询并排序，返回from+size个文档Id和排序值



Search的运行机制-Fetch阶段

node3根据Query阶段获取到文档Id列表对应的shard上获取文档详情数据

- 1、node3向相关的分片发送multi_get请求
- 2、3个分片返回文档详细数据
- 3、node3拼接返回的结果并返回给用户



09

相关性算分

相关性算分问题

相关性算分在shard与shard间是相互独立的，也就意味着同一个Term的IDF等值在不同shard上是不同的。文档的相关性算法和它所处的shard相关
在文档数量不多时，会导致相关性算分严重不准的情况发生

解决思路有两个：

- 1、设置分片数为1个，从根本上排除问题，在文档数量不多的时候可以考虑该方案，
比如百万到千万级别的文档数量
- 2、采用DFS Query-then-Fetch的查询方式



相关性算分问题

DFS Query-then-Fetch是拿到所有文档后再重新完整的计算一次相关性算法，耗费更多的cpu和内存，执行性能也比较低下，一般不建议使用。操作方式如下：

```
GET test_search_relevace/_search?search_type=dfs_query_then_fetch
{
  "query": {
    "match": {
      "name": "hello"
    }
  }
}
```



谢谢观看

咕泡学院，只为更好的你！

➤ Tom老师QQ号：441221062

