[一、基础概念 1](#_Toc6764)

[二、搭建Elasticsearch开发环境 2](#_Toc18569)

[2.1、单机单节点 2](#_Toc27765)

[2.2、为集群添加故障转移 2](#_Toc26205)

[2.3、为集群扩容 3](#_Toc13903)

[2.4、Elasticsearch的故障应对机制 4](#_Toc356)

[三、通过RESTful API与Elasticsearch交互 4](#_Toc21397)

[3.1、Curl请求示例 4](#_Toc22056)

[3.2、文档操作 5](#_Toc26797)

[四、通过JAVA客户端与Elasticsearch交互 8](#_Toc10751)

[五、Elasticsearch的存储与扩容机制 8](#_Toc20125)

[5.1、查看集群的健康信息 8](#_Toc24944)

[5.2、Elasticsearch的分片机制 9](#_Toc224)

[六、Elasticsearch的索引和文档 9](#_Toc19724)

[6.1、Elasticsearch的冲突处理机制 9](#_Toc4740)

[七、监控Elasticsearch 9](#_Toc2003)

[7.1、Elasticsearch提供的监控接口 9](#_Toc19374)

[7.1.1、Elasticsearch的集群监控接口 9](#_Toc30200)

[7.1.2、Elasticsearch的cat系列监控接口 12](#_Toc12791)

[7.1.3、Elasticsearch的索引监控接口 12](#_Toc14361)

[7.2、如何监控Elasticsearch 12](#_Toc30534)

[7.2.1、Elasticsearch性能监控 12](#_Toc1336)

[7.2.2、Elasticsearch指标采集 13](#_Toc30822)

[7.2.3、Elasticsearch常见性能问题解决方案 13](#_Toc2430)

[7.3、Kibana 13](#_Toc4415)

[7.4、ElasticsearchHq 13](#_Toc6157)

[7.5、Cerebro 13](#_Toc31710)

[7.6、ElasticsearchHead 13](#_Toc12398)

### 一、基础概念

1、什么是Elasticsearch：一个实时的分布式搜索分析引擎，可用于文档的分布式存储、字段索引搜索、全文搜索、结构化搜索和分析，并能够支持上百服务节点的扩展和PB级的数据。Elasticsearch建立于全文搜索引擎Lucene之上，使用Lucene做索引与搜索，并提供一套RESTful API。

2、Elasticsearch是面向文档的：Elasticsearch可以直接存储整个对象或是文档，而不必将对象或文档扁平化。Elasticsearch使用JSON作为文档额序列化格式。有别于高度结构化的关系型数据库，单个对象散布于多个表格中，文档数据库可以一次存储整个对象，并且每个对象可与其他被存储的对象不同。

3、Elasticsearch的数据组织方式：Elasticsearch中的索引(Index)类似于关系型数据库中的一个数据库，使用倒排索引来达到提升检索速度的目的。Elasticsearch中的类型(Type)类似于关系型数据库中的表格。Elasticsearch中的文档(Document)类似于关系型数据库中的一个数据行，将其存入的行为被称作索引（动词）。Elasticsearch中的字段(Field)类似于关系型数据库中的列。

4、Elasticsearch的集群组织方式：在分布式环境下，Elasticsearch由集群(cluster)、节点(node)以及分片(shard)组成。一个运行中的实例被称为一个节点，一个或多个拥有相同cluster.name配置的节点组成集群。新加入节点或是删除节点均会导致集群重新分配所有数据。集群中的主节点负责索引和其他节点的增删操作，不会涉及文档级别的数据操作和搜索。

### 二、搭建Elasticsearch开发环境

#### 2.1、单机单节点

1、执行bin/elasticsearch -d，后台运行Elasticsearch

2、访问http://localhost:9200/?pretty，测试Elasticsearch是否启动成功，若成功启动，可以收到类似结构的信息，如图2-1：

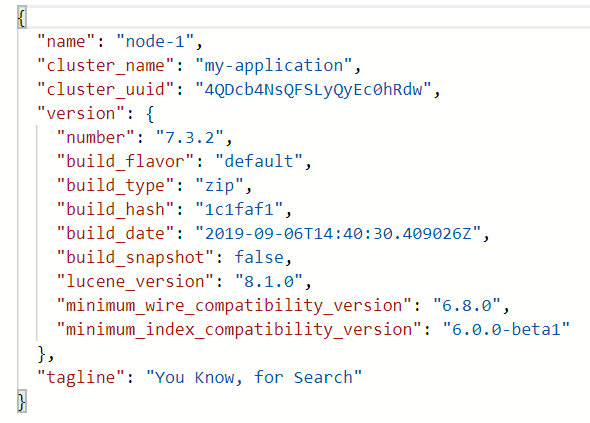


图2-1

可以在conf/elasticsearch.yml中配置相关信息。一个Elasticsearch集群中的节点拥有相同的cluster\_name。

#### 2.2、为集群添加故障转移

在同一台机器上Elasticsearch默认使用组播的方式发现新节点，集群中使用单播方式发现节点。通过修改elasticsearch.yml中的discovery.zen.ping.unicast.hosts可以修改单播列表，新节点连接单播列表中的任一成员均可获取到集群的整体状态和节点信息，之后新节点会联系master节点，并加入集群。集群的最小配置字段如图2-2所示：

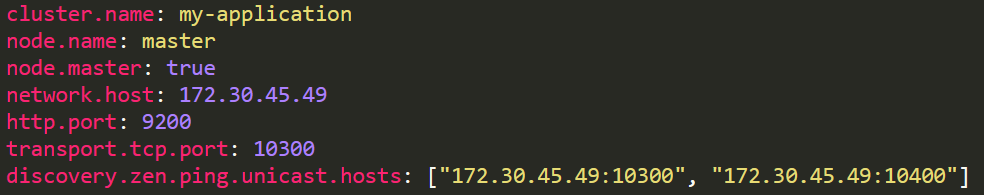


图2-2

同一集群内节点的cluster.name字段应该相同，Elasticsearch启动需要绑定两个端口（9200与9300），一个负责处理http请求，另一个负责处理来自客户端与集群其他节点的tcp连接。使用node.master字段标识主节点。

当配置并启动集群后，Elasticsearch会自动为每个节点分配主分片（P0-P2）或是副本分片（R0-R2），如图2-3，做到数据的一定程度冗余，防止出现数据丢失。若有一个新节点加入集群，那么Elasticsearch会重新调整集群上的分片分布，使其分布均匀，如图2-4所示。

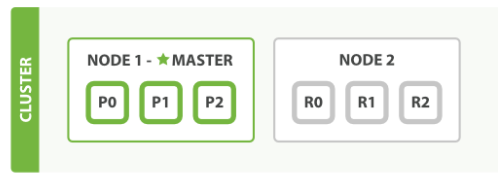


图2-3

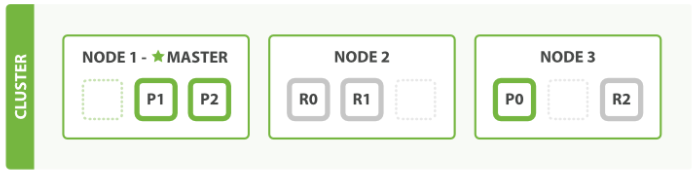


图2-4

#### 2.3、为集群扩容

在创建索引时可以指定主分片的个数，如图2-5所示，为索引megacorp指定3个主分片，每个主分片对应一个副本分片，共6个分片。6个分片最多可分布于6个节点上，每个节点上存放一个分片，若节点个数多于6个，则需要为集群扩容。

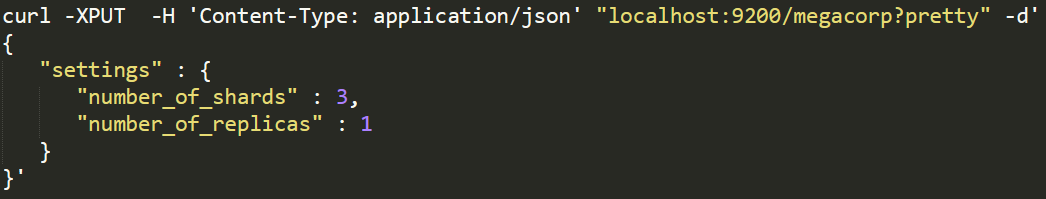


图2-5

由于索引的主分片数在创建时已经确定，可以通过调整每个主分片对应的副本分片数来动态伸缩集群，调整方式如图2-6所示。每个主分片对应2个副本分片，调整后集群一共包含9个分片，最多可扩容至9个节点上。调整后的结果如图2-7所示。

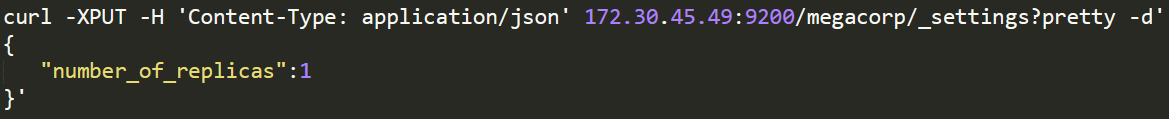


图2-6

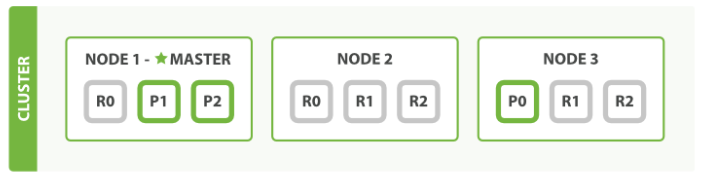


图2-7

在3个节点上存放9个分片可以保证在2个节点出故障的情况下，集群不丢失任何数据。若在3个节点上存放6个分片，如图2-4，那么最多1个节点出故障，集群才能不丢失任何数据。在完成集群扩容后，可通过如下命令查看分片状态：

curl -XGET -s 172.30.45.49:9200/\_cat/shards?h=index,shard,prirep,state,unassigned.reason

若每个节点上均存放有索引的完整数据，如图2-7，而此时仍然有新的副本分片加入，则新加入的副本分片会成为未分配的分片（unassigned replica），可通过减少副本分片数来解决。

#### 2.4、Elasticsearch的故障应对机制

若集群中的主节点发生故障，则集群会重新在剩余的节点中选举出一个新的主节点，如图2-8。并检查集群中主分片的缺失情况，若存在缺失主分片对应的副本分片，则会将这些副本分片提升为主分片。

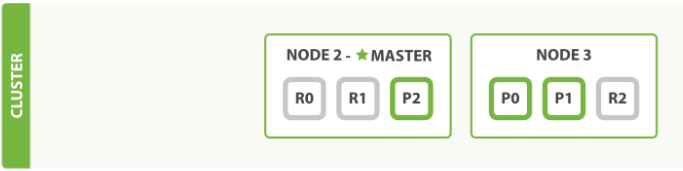


图2-8

### 三、通过RESTful API与Elasticsearch交互

#### 3.1、Curl请求示例

默认使用9200端口与Elasticsearch通信，请求示例：

curl -XGET -H "Content-Type: application/json" http://localhost:9200/\_count?pretty -d '{ "query": {"match\_all":{} } }'

向Elasticsearch插入一条数据，请求示例：

curl -X PUT -H 'Content-Type: application/json' "localhost:9200/megacorp/employee/1?pretty" -d' { "first\_name" : "John", "last\_name" : "Smith", "age" : 25, "about" : "I love to go rock climbing", "interests": [ "sports", "music" ] } '

其中megacorp为索引名，employee为类型名称，1为文档ID。

根据文档ID请求一条数据，请求示例：

curl -X GET "localhost:9200/megacorp/employee/1?pretty"

通过更换请求method为DELETE，可以删除指定数据，请求示例：

curl -X DELETE "localhost:9200/megacorp/employee/1?pretty"

可以通过使用PUT或POST方法对已存在的文档进行更新操作，请求示例：

curl -X PUT -H 'Content-Type: application/json' "localhost:9200/megacorp/employee/1?pretty" -d '{ "first\_name" : "John"}'

Elasticsearch在执行更新操作时，会先将旧文档标记为删除状态，然后添加新文档。

通过在请求路径中使用\_search端点，并配合q参数可以实现具体到字段的轻量搜索，请求示例：

curl -X GET "localhost:9200/megacorp/employee/\_search?q=last\_name:Smith&pretty"

也可以使用JSON构造一个更复杂的查询请求：

curl -X GET -H "Content-Type: application/json" "localhost:9200/megacorp/employee/\_search" -d'

{"query": {"bool": {"must": [{ "match": { "last\_name": "Smith" }},{ "match": { "age": 25 }}]}}}'

curl -X GET "localhost:9200/megacorp/employee/\_search?pretty" -H 'Content-Type: application/json' -d'

{"query" : {"bool": {"must": {"match" : {"last\_name" : "smith"}},"filter": {"range" : {"age" : { "gt" : 30 }}}}}}'

#### 3.2、文档操作

1. 索引文档，使用自定义ID

|  |  |  |
| --- | --- | --- |
| method | uri | data |
| PUT | /{index}/{type}/{id} | {"field":"value",...} |

返回格式为：

|  |
| --- |
| {  "\_index": {index},  "\_type": {type},  "\_id": {id},  "\_version": 1,  "created": true  } |

若仅需要在文档不存在时才创建新文档，防止现有文档被覆盖，则需要在uri后附上op\_type或是\_create参数，如/{index}/{type}/{id}?op\_type=create, /{index}/{type}/{id}/\_create。若请求成功，Elasticsearch会返回文档元数据以及201的HTTP响应码，若文档已经存在，会返回409的响应码。

1. 索引文档，使用自动生成ID

|  |  |  |
| --- | --- | --- |
| method | uri | data |
| POST | /{index}/{type}/ | {"field":"value",...} |

返回格式为：

|  |
| --- |
| {  "\_index": {index},  "\_type": {type},  "\_id": {generated\_id},  "\_version": 1,  "created": true  } |

Elasticsearch自动生成的ID为基于Base64编码的长度为20的GUID字符串。

1. 取回一个文档

|  |  |  |
| --- | --- | --- |
| method | uri | data |
| GET | /{index}/{type}/{id} |  |

返回格式为（元数据+文档内容\_source）：

|  |
| --- |
| {  "\_index": {index},  "\_type": {type},  "\_id": {id},  "\_version": 1,  "found": true,  "\_source" : {...}  } |

若没有找到对应的文档，请求返回码将变为404，found字段将变为false。此uri接受\_source参数，如/{index}/{type}/{id}?\_source=field1,field2，则只返回field1和field2字段的查询结果。若\_source填空值，如/{index}/{type}/{id}?\_source或/{index}/{type}/{id}/\_source则返回值中不包含文档元数据。

1. 检查文档是否存在

|  |  |  |  |
| --- | --- | --- | --- |
| method | args | uri | data |
| HEAD | -I | /{index}/{type}/{id} |  |

返回格式为（只包含HTTP报头）：

|  |
| --- |
| HTTP/1.1 200 OK  Content-Type:text/plain; charset=UTF-8  Content-Length: 0 |

如文档不存在，则返回404.

1. 更新文档

|  |  |  |
| --- | --- | --- |
| method | uri | data |
| PUT | /{index}/{type}/{id} | {"field":"value",...} |

由于Elasticsearch中的文档不能修改，只能整体替换，更新文档的API与索引文档的API相同。

返回格式为：

|  |
| --- |
| {  "\_index": {index},  "\_type": {type},  "\_id": {id},  "\_version": 2,  "created": false  } |

1. 删除文档

|  |  |  |
| --- | --- | --- |
| method | uri | data |
| DELETE | /{index}/{type}/{id} |  |

返回格式为：

|  |
| --- |
| {  "\_index": {index},  "\_type": {type},  "\_id": {id},  "\_version": 3,  "found": true  } |

若文档不存在，则found字段为false。

1. 部分更新文档

|  |  |  |
| --- | --- | --- |
| method | uri | data |
| POST | /{index}/{type}/{id}/\_update | {"doc":{field:xxx,...}} |

返回格式为：

|  |
| --- |
| {  "\_index": {index},  "\_type": {type},  "\_id": {id},  "\_version": 3  } |

由于Elasticsearch的文档是不可变的，因此update操作包含了检索-修改-重建索引这三个处理过程，这三个过程均发生在分片内部。由于Elasticsearch使用乐观锁来处理冲突，因此update在执行检索-修改-重建索引这三个步骤的过程中存在冲突的可能。若发生冲突，update请求失败。对于部分字段的更新，执行顺序不重要，可以通过添加retry\_on\_conflict参数来指定冲突后的重试次数，如/{index}/{type}/{id}/\_update?retry\_on\_conflict=5。

1. 批量取回文档

未指定\_index、\_type和\_id：

|  |  |  |
| --- | --- | --- |
| method | uri | data |
| GET | /\_mget | {"docs":[  {"\_index":xx,"\_type":xx,"\_id":xx},  {"\_index":xx,"\_type":xx,"\_id":xx,"\_source":xx}  ]} |

指定\_index、\_type和\_id：

|  |  |  |
| --- | --- | --- |
| method | uri | data |
| GET | /{index}/{type}/\_mget | {"docs":[{"\_id":xx},{"\_id":xx,"\_source":xx}]} |
| GET | /{index}/{type}/\_mget | {"ids":["xx", "xx"]} |

返回格式为：

|  |
| --- |
| {  "docs":[  {  "\_index": {index},  "\_type": {type},  "\_id": {id},  "\_version": 3  "found": true,  "\_source":{...}  },  ...  ]  } |

1. 批量更新文档

|  |  |  |
| --- | --- | --- |
| method | uri | data |
| POST | /\_bulk | bulk\_request |
| POST | /{index}/\_bulk | bulk\_request |
| POST | /{index}/{type}/\_bulk | bulk\_request |

bulk\_request的格式为：

|  |
| --- |
| {"action01":{ doc\_metadata}}\n  {request body of action01}\n  {"action02":{ doc\_metadata}}\n  {request body of action02}\n  ...  {"actionN":{ doc\_metadata}}\n  {request body of actionN}\n |

action可以是create、index、update与delete四个操作之一，其中create操作是在文档不存在时创建，index操作是创建或替换文档。request body包含了对应操作所需的请求参数。

返回格式为：

|  |
| --- |
| {  "took":3,  "error": false,  "items":[  {"delete":{  "\_index": {index},  "\_type": {type},  "\_id": {id},  "\_version": 3  "status": 200,  "found": true  }},  ...  ]  } |

### 四、通过JAVA客户端与Elasticsearch交互

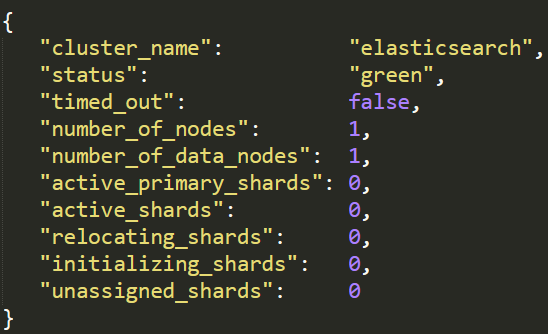
### 五、Elasticsearch的存储与扩容机制

5.1、查看集群的健康信息

请求示例：

curl -X GET "localhost:9200/\_cluster/health?pretty"

返回内容如下：

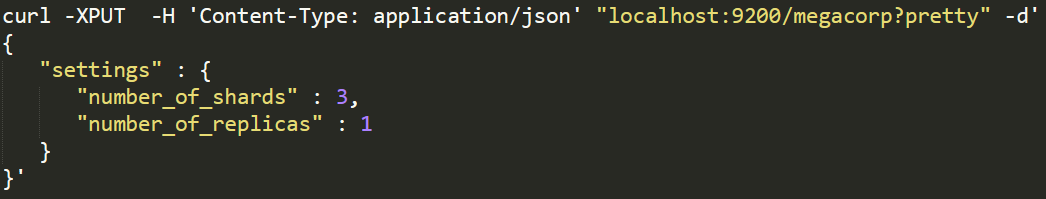


其中status字段表示集群的运行状态：green表示所有主分片和副本分片都正常运行，yellow表示所有主分片和部分副本分片正常运行，red表示部分主分片没有正常运行。

5.2、Elasticsearch的分片机制

分片是Elasticsearch的一个底层工作单元，仅保存一部分数据，一个分片即为一个完整的搜索引擎（一个Lucene实例）。当集群规模变化时，为了能够使数据均匀分布在集群里，Elasticsearch会迁移分片，利用分片将数据分发到集群各处。作为数据容器，当添加一个文档时，文档会被存储和索引到分片内。

分片可以被分为主分片和副本分片，副本分片只是主分片的拷贝，用于数据冗余备份以及提供数据读操作。在创建索引时，Elasticsearch会首先为索引分配多个主分片，每个主分片对应一个副本分片，可以通过如下方式在创建时指定索引的主分片数(number\_of\_shards)，以及每个主分片对应的副本分片数(number\_of\_replicas)：



同时也可以通过GET方法查看索引的状态信息。

### 六、Elasticsearch的索引和文档

#### 6.1、Elasticsearch的冲突处理机制

Elasticsearch使用乐观锁的方式进行并发控制。为每个文档分配一个版本号（\_version），当文档被修改时版本号依次递增，以此来确保数据变更能够以正确的方式进行。在更新文档时带上version参数，如curl -XPUT /{index}/{type}/{id}?version=1，此处version表示修改会被应用的文档版本，Elasticsearch会检测此次更新是否会有冲突，若冲突会返回409的HTTP响应码。

### 七、监控Elasticsearch

#### 7.1、Elasticsearch提供的监控接口

7.1.1、Elasticsearch的集群监控接口

1、集群健康监控

|  |  |  |
| --- | --- | --- |
| method | uri | desc |
| GET | /\_cluster/health |  |
| GET | /\_cluster/health/index1,index2 | 查看关于索引的更详细信息 |
| GET | /\_cluster/health?level=indices |
| GET | /\_cluster/health?level=shards | 查看关于每个分片的更详细信息 |
| GET | /\_cluster/health/index1,index2?level=shards |
| GET | /\_cluster/health?wait\_for\_status=green | 阻塞，直到达到指定状态 |

返回格式为：

|  |
| --- |
| {  "cluster\_name" : "my-application",  "status" : "green",  "timed\_out" : false,  "number\_of\_nodes" : 2,  "number\_of\_data\_nodes" : 2,  "active\_primary\_shards" : 8,  "active\_shards" : 16,  "relocating\_shards" : 0,  "initializing\_shards" : 0,  "unassigned\_shards" : 0,  "delayed\_unassigned\_shards" : 0,  "number\_of\_pending\_tasks" : 0,  "number\_of\_in\_flight\_fetch" : 0,  "task\_max\_waiting\_in\_queue\_millis" : 0,  "active\_shards\_percent\_as\_number" : 100.0  } |

字段解释：

·status :green表示所有主分片和副本分片都已分配；yellow表示所有主分片已被分配，但至少一个副本分片缺失；red表示至少一个主分片缺失，其对应的副本分片也缺失（若存在副本分片，则副本分片会被提升为主分片），搜索只能返回部分数据。

·timed\_out :

·number\_of\_nodes :集群中的节点个数。

·number\_of\_data\_nodes :集群中的数据节点个数，默认所有节点都是数据节点与主节点候选节点。但是可以通过node.data: false来声明此节点不存储数据，一般用于master节点和候选节点。同时也可以通过设置node.master: false来取消节点成为主节点的资格，只存储数据。两者都为false，则节点将会成为一个专用的客户端节点，起负载均衡的作用。

·active\_primary\_shards :集群中主分片数量。

·active\_shards :集群的所有分片数量，包括副本分片。

·relocating\_shards :正在迁往其他节点的分片数量。

·initializing\_shards :当分片刚创建或是节点刚重启时，分片会短暂处于initializing状态。

·unassigned\_shards :未分配的分片数量，若分片数量过多，超过了满足集群每个节点保存完整数据的需要，就会出现未分配分片。若有节点出现故障也会出现未分配分片。

·delayed\_unassigned\_shards :

·number\_of\_pending\_tasks :

·number\_of\_in\_flight\_fetch :

·task\_max\_waiting\_in\_queue\_millis :

·active\_shards\_percent\_as\_number :

2、集群状态监控

|  |  |  |
| --- | --- | --- |
| method | uri | desc |
| GET | /\_cluster/state |  |
| GET | /\_cluster/state/{metrics}/{indices} | 查询指标和索引过滤 |

监控指标{metrics}包括了，这些指标可用逗号组合：

version：包含了集群名称，集群uuid和状态uuid等信息。

master\_node：

nodes：

routing\_table：

metadata：

blocks：

3、集群状态统计

|  |  |  |
| --- | --- | --- |
| method | uri | desc |
| GET | /\_cluster/stats | 集群范围的统计信息 |

返回内容包括了索引统计指标（分片数目、存储占用以及内存占用等信息）和集群节点（数目统计、操作系统和机器信息、jvm统计信息、插件信息、内存和cpu利用率等信息）。

1. 等待中任务信息获取

|  |  |  |
| --- | --- | --- |
| method | uri | desc |
| GET | /\_cluster/pending\_tasks | 统计能引起集群状态改变且尚未被执行的任务信息 |

返回格式为：

|  |
| --- |
| {  "tasks": [  {  "insert\_order": 101,  "priority": "URGENT",  "source": "create-index [foo\_9], cause [api]",  "time\_in\_queue\_millis": 86,  "time\_in\_queue": "86ms"  },  ...  ]  } |

5、集群节点监控

|  |  |  |
| --- | --- | --- |
| method | uri | desc |
| GET | /\_nodes/stats | 获取集群每个节点的统计数据 |
| GET | /\_nodes/nodeId1,nodeId2/stats | 获取集群指定节点的统计数据 |
| GET | /\_nodes/stats/{metrics} | 根据{metrics}过滤统计指标 |

返回格式为：

|  |
| --- |
| {  "\_nodes": {  "total": 2,  "successful": 2,  "failed": 0  },  "cluster\_name": "my-application",  "nodes": {  "82AFcZQDTzOAZc1U5i0LVQ": {  "timestamp": 1569415960231,  "name": "slave-2",  "transport\_address": "172.30.45.49:10400",  "host": "172.30.45.49",  "ip": "172.30.45.49:10400",  ...  },  "UjHGpQucRyaN6Jw2Wm7bRA": {  "timestamp": 1569415960231,  "name": "master",  ...  }  }  } |

监控指标{metrics}包括了，这些指标可用逗号组合：

indices：节点上索引的相关统计值，包括索引大小、文档数量、创建删除索引次数、搜索次数和字段缓存大小等信息。

fs：包括数据存储路径、剩余磁盘空间以及读写统计等信息。

http：http连接信息。

jvm：JVM统计信息，内存池信息，垃圾收集信息，缓存区以及类加载等信息。

os：描述整个操作系统的状况，包括负载、内存和内存交换等信息。

process：包括内存消耗、cpu占用和文件描述符占用等信息。

thread\_pool：线程池信息。

transport：网络传输数据统计。

breaker：field数据断路器相关信息。

discovery：

ingest：

7.1.2、Elasticsearch的cat系列监控接口

7.1.3、Elasticsearch的索引监控接口

#### 7.2、如何监控Elasticsearch

7.2.1、查询性能监控

Elasticsearch查询分为六个阶段：

Fetch

Query

Route

**Load&Return**

Response

MultiGet

Exec&Gather

Request

1. Request：客户端发送查询请求至集群某个节点NodeCoord。
2. Route：NodeCoord将查询请求发送至索引的一组完整分片上。
3. Execute&Gather：每个分片在本地执行查询操作并将结果传递至NodeCoord。
4. MultiGet：NodeCoord根据查询结果确定需要取回的文档和对应分片，然后向对应分片发送multi get请求。
5. Load&Return：收到请求的分片会加载文档并发送至NodeCoord。
6. Response：NodeCoord将查询结果发送至客户端。

需要监控的指标：

|  |  |
| --- | --- |
| name | desc |
| indices.search.query\_total | 查询总数 |
| indices.search.query\_time\_in\_millis | 用于查询的总时间 |
| indices.search.query\_current | 目前正在进行的查询数量 |
| indices.search.fetch\_total | 执行文档取回的次数 |
| indices.search.fetch\_time\_in\_millis | 执行文档取回的总时间 |
| indices.search.fetch\_current | 目前正在执行文档取回的任务量 |

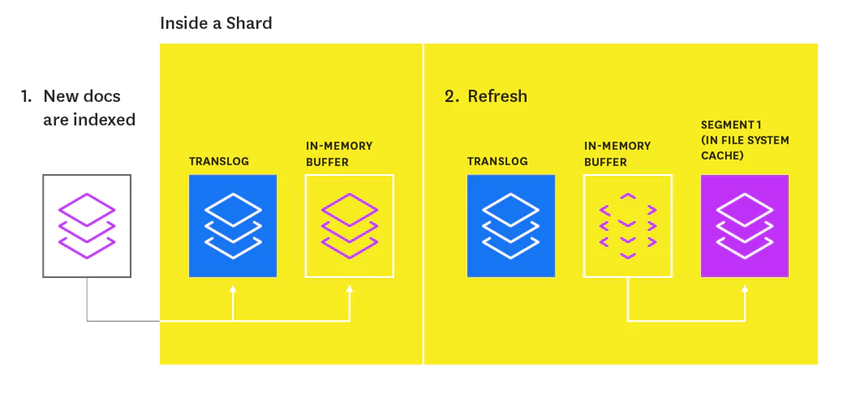
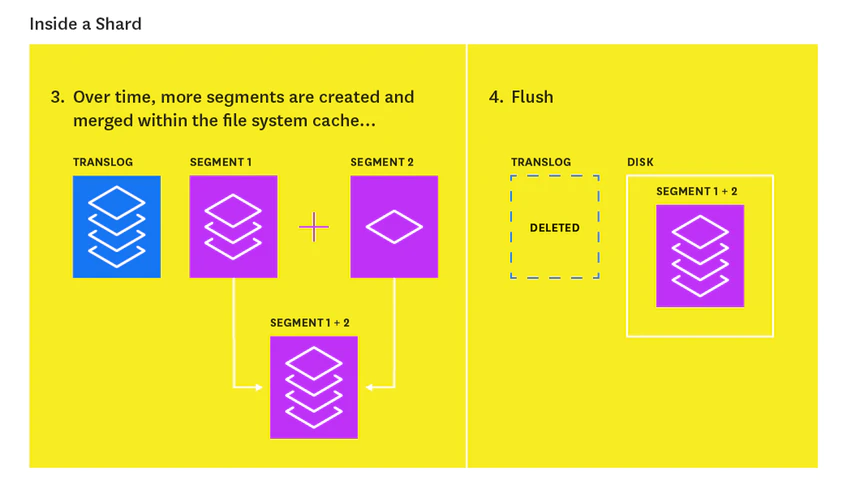
需要分析的指标：

|  |  |
| --- | --- |
| name | desc |
| query latency | 通过采样均匀时间间隔内的查询数和查询时间，计算出平均查询延迟 |
| fetch latency | 通过均匀间隔采样计算文档取回延迟 |

7.2.2、索引性能监控

每个分片内索引更新的过程分为两个阶段：refresh和flush。

1、新文档会首先被写入in-memory buffer中，同时附加在write-ahead log（translog）中。



1. 默认每隔一秒会进行一次索引刷新，使用in-memory buffer中的内容生成一个in-memory segment，然后清空in-memory buffer，此时文档可检索。
2. 每隔30分钟或是translog被写满，会触发flush操作。所有暂存于in-memory buffer中的文档会被刷新（refresh），生成新的in-memory segment。所有in-memory segment会被写入磁盘，同时清空translog。

在Elasticsearch中，每个分片由多个segment组成，每个segment是一个倒排索引，用于在词汇与文档之间建立映射关系。segment是不可变的，因此每当有文档被更改或删除，会在refresh期间将更新后的文档写入一个新的segment中，同时在旧的segment中标记该文档为删除状态。由于Elasticsearch默认每秒会生成一个新的segment，为了避免数量爆炸，会有专门的后台线程处理segment合并，小的segment会被合并为更大的segment。在合并期间，被标记为删除的文档不会合并进新的segment中。

需要监控的指标：

|  |  |
| --- | --- |
| name | desc |
| indices.indexing.index\_total | 被索引的文档总数 |
| indices.indexing.index\_time\_in\_millis | 索引文档花费的时间 |
| indices.indexing.index\_current | 正在被索引的文档数量 |
| indices.refresh.total | 索引刷新总数 |
| indices.refresh.total\_time\_in\_millis | 索引刷新花费的时间 |
| indices.flush.total | 写入到磁盘的索引数 |
| indices.flush.total\_time\_in\_millis | 索引写入磁盘花费的时间 |

需要分析的指标：

|  |  |
| --- | --- |
| name | desc |
| indexing latency | 通过index\_total与index\_time\_in\_millis可以计算出索引延迟 |
| flush latency | 数据持久化延迟 |

7.2.3、JVM垃圾回收监控

需要监控的指标：

|  |  |
| --- | --- |
| name | desc |
| jvm.gc.collectors.young.collection\_count | 年轻代垃圾回收次数 |
| jvm.gc.collectors.young.collection\_time\_in\_millis | 用于年轻代垃圾回收的总时间 |
| jvm.gc.collectors.old.collection\_count | 老年代垃圾回收次数 |
| jvm.gc.collectors.old.collection\_time\_in\_millis | 用于老年代垃圾回收的总时间 |
| jvm.mem.heap\_used\_percent | JVM堆使用百分比 |
| jvm.mem.heap\_committed\_in\_bytes | 由JVM保证能用的堆大小 |
| jvm.mem.heap\_used\_in\_bytes | 已使用的堆大小 |

需要分析的指标：

|  |  |
| --- | --- |
| name | desc |
| garbage collection duration | 通过对垃圾回收指标等间隔采样可计算出 |
| garbage collection frequency | 通过对垃圾回收指标等间隔采样可计算出 |

7.2.3、节点监控

需要监控的指标：

|  |  |
| --- | --- |
| name | desc |
| available disk space |  |
| I/O utilization | 磁盘I/O情况 |
| CPU usage |  |
| network bytes sent/received |  |
| open file descriptors | 文件描述符，用于节点间通信、客户端通信和文件操作 |
| http.current\_open | 目前打开的http连接数 |
| http.total\_opened | 历史上打开的http连接总数 |

7.2.4、集群健康程度监控

需要监控的指标：

|  |  |
| --- | --- |
| name | desc |
| cluster.health.status | green、yellow或red |
| cluster.health.number\_of\_nodes | 集群中节点总数 |
| cluster.health.initializing\_shards | 处于初始化状态的分片数，通常发生于新建索引或是节点重启后 |
| cluster.health.unassigned\_shards | 未分配的分片数 |

7.2.5、线程池监控

Elasticsearch提供了多种线程池的监控信息，其中search、merge和write线程池的信息最为常用。

需要监控的指标：

|  |  |
| --- | --- |
| name | desc |
| thread\_pool.search.queue  thread\_pool.merge.queue  thread\_pool.write.queue | 缓存于队列的任务数 |
| thread\_pool.search.rejected  thread\_pool.merge.rejected  thread\_pool.write.rejected | 被拒绝的任务数 |
| thread\_pool.search.active  thread\_pool.merge.active  thread\_pool.write.active | 正在执行的任务数 |

7.2.6、缓存监控

Elasticsearch包含了三种缓存，分别是query cache（filter cache），request cache，fielddata cache。query cache会对查询中包含了过滤器执行结果进行缓存，之后使用相同过滤器的查询可以直接使用缓存结果。request cache缓存分片查询的本地结果集，这些本地结果集之后会被发送至协调节点（NodeCoord），当查询的DSL语句一样时则命中缓存，分片每次执行完refresh操作后会被清空。字段排序与聚合会用到fielddata，其对倒排索引做了un-invert操作，可以查到每个文档包含的词语（terms），对倒排索引做逆转非常耗内存，因此Elasticsearch对fielddata做了缓存。

需要监控的指标：

|  |  |
| --- | --- |
| name | desc |
| indices.query\_cache.memory\_size\_in\_bytes  indices.request\_cache.memory\_size\_in\_bytes  indices.fielddata.memory\_size\_in\_bytes | 缓存大小 |
| indices.query\_cache.evictions  indices.request\_cache.evictions  indices.fielddata.evictions | 被移出缓存的条目数 |
| indices.query\_cache.hit\_count  indices.request\_cache.hit\_count | 缓存命中次数 |
| indices.query\_cache.miss\_count  indices.request\_cache.miss\_count | 缓存未命中次数 |

7.2.7、GET请求监控

需要监控的指标：

|  |  |
| --- | --- |
| name | desc |
| indices.get.total | GET请求总数 |
| indices.get.time\_in\_millis | 处理请求花费的总时间 |
| indices.get.exists\_total | 请求成功的GET请求总数 |
| indices.get.exists\_time\_in\_millis | 处理该类请求花费的总时间 |
| indices.get.missing\_total | 请求失败的GET请求总数 |
| indices.get.missing\_time\_in\_millis | 处理该类请求花费的总时间 |
| indices.get.current | 正在处理的GET请求数 |

7.2.2、Elasticsearch指标采集

7.2.3、Elasticsearch常见性能问题解决方案

#### 7.3、Kibana

#### 7.4、ElasticsearchHq

#### 7.5、Cerebro

#### 7.6、ElasticsearchHead