ElasticSearch使用基础

借助于LuceneAPI 在此之外构建了一层API 不仅仅添加了搜索组件 -----

---- SQL中数据存储到表中

每一个可被搜索的组件 叫文档 文档相当于SQL中表的行

Lucene中索引就是包含大量文档的集合 ----- 索引相当于SQL中的表

SQL中逻辑叫表 ---- 都会落地到本地磁盘文件

Lucene每一个索引有一个对应的物理文件 ----

SQL中的表定义好之后 schema定了 ---- 因为关系型 列一样 但是Lucene的索引的schema是free的

------ 但是 生产中 一般都是 人都放到同一个索引 兔子放到另一个索引 那这样schema-free还有意义么？

Schema-free的意义 --- 都是人 但是 字段可能不一样

\*\*索引就是可以被直接搜索的组件 这个和MySQL不一样 SQL中 索引是另外单独存储数据的组件 ---Lucene中 整个可以被搜索的文档 都被存储在索引中

索引本身就是倒排机制定义的 ----- 这里面 文档（就是数据）和倒排索引（类似SQL中的索引）是放在一起的 --- 都存放在成为索引（这个索引 是SQL中的表 里面包含了倒排索引+文档）的内部 （Innodb中 索引和数据放在一起）

===== 逻辑上也叫索引+文档

---------- es为了让Lucene并行存储和搜索 一个节点不能满足 Lucene本身就是一个库 本身不能完成复杂的功能 ---- es把lucene的机制扩展了 ---- 支持把一个大的索引（大的表）进行切片 --- 每一片叫shard ----- 把一个大的索引分散到多个物理机上 分布式 ===== 这样读和写就是分布式的

---------- 一个挂了 就都挂了 因为数据不完整的 --- 分布式不允许这样 ---- 每个节点高可用 这样要给一个共享存储 这样非常麻烦 （有时间弄清楚 主从 和 高可用）---- **替代就是 shard级别 每一个分片做冗余就可以了**

---------- 就有了主分片 和 副本分片 这个是主从机制

\*\*\*\*说到主从 就必须有以下几个反应：

&&&&第一个反应就是读写操作

----- 主能读能写 但是 从是只读

Es底层完成了分布式 利用了Lucene

搜索的时候 如何向那个节点发送查询？ 任何一个节点都可以发查询 ---- 因为es集群无中心 ---- 真正的发起读写操作 必须通过客户端或者API ---- 用es写一个程序 或者专用的接口 ----- 接口或者程序可以理解你的查询 可以帮你把查询转出去

写操作 写的请求发送到第三个节点 但是 第三个节点存储的是**第二个的从分片** ----- 所以你的写操作无法写到第三个节点的第二个分片中 ----- 这时候 把请求转发到主分片进行写入操作

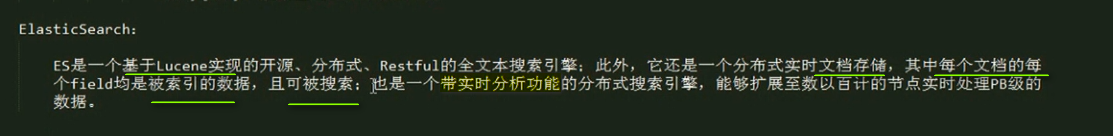
----- 完全借助于Lucene + 自己完成了分布式 ----- es自身能维护分片副本数量 --- 某个节点坏了等等 都可以底层自己实现了

===== es是一个**分布式的数据存储 搜索引擎** ---- 接近实时 ---- 也可以单节点运行

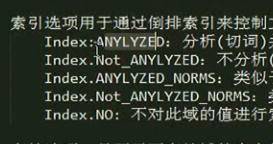
【存储的就是索引库 索引库中是倒排索引+文档】

配置简单 节点名 + 加入到集群的名称 剩下可以自己完成

=============== es的基本概念==========================



仅仅支持文本搜索 不仅仅仅是搜索引擎 还是一个分布式实时文档存储 其中每一个文档的每一个field都是被索引的数据 ---- 都可以被搜索

---- 刚才讲Lucene的时候 需要 每一个域都要指定被分词的方式  但是 es中 不用指定这些 都有默认来完成 每个域自动加了ANLYZED 这样每一个域都可以被搜索

----- ES的基本组件 和Lucene重叠

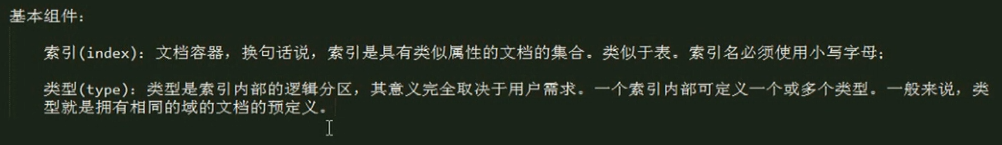
\*\*索引（Index）是具有类似属性的文档的集合 类似于关系型数据库中的表 es中的索引名**必须小写字符 不能大写**--- 是lucene索引和搜索的原子单位 --- 也就是你的Lucene构建索引 必须以索引为基础 你的文档也是放在索引中的

一个es可以创建任意数量的索引

\*\*类型（Type）是索引内部的逻辑分区 其意义完全取决于我们的需求 ---- 所以 一个索引内部可以定义一个或者多个类型 Type ---- Type就是拥有相同域的预定义 -----

文档是schema free的 为什么要定义结构type? 这个type没有约束性

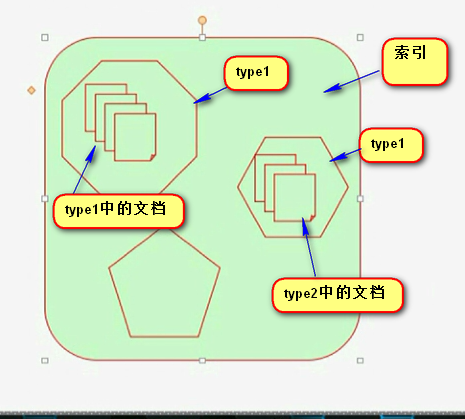
一个表只有一个结构 但是一个索引可以有一个或者多个类型



------ es中一个索引中可以定义存储用户日志类型 用户数据类型 等 这样一个索引可以存储2类数据 建议是一个索引一个Type

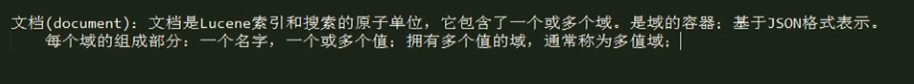
【所以 索引本身存储的是文档 但是 free-schema 现在es想加点限制 规范一下 就把一个索引中 意义相同的文档归类 就出现了索引的逻辑分区 Type】---- 一个索引中的类型就是存储**具有相同结构的一类文档的组件**

**---- 一般而言 一个索引内部仅仅有一个类型**



\*\*文档(document):是Lucene索引和搜索的原子单位 包含了一个或者多个域 是个域容器

文档基于json表示 ---- 每个域都一个名字 但是 可能有一个或者多个值 ---- 一个域的值还可能有其他文档【json嵌套】 ---文档可以嵌套 ---- 拥有多个值的域 叫做多值域

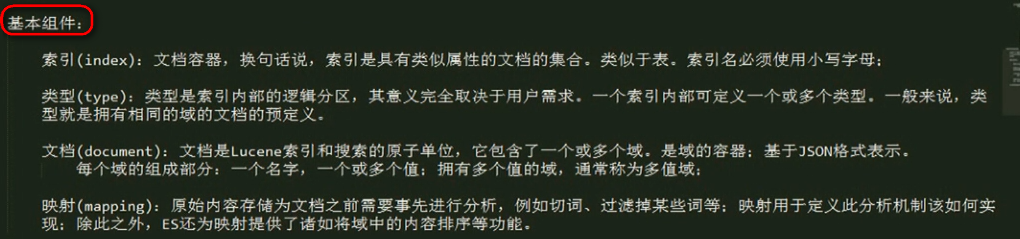


\*\*映射(mapping): 原始内容存储为文档之前需要事先进行分析 --- 映射就是用于定义**此分析机制该如何实现** 包括切词 过滤停词 等 ---- 除此之外 es还为映射提供了将域中的内容进行排序等

每个原始数据 先分析 然后存储文档 然后构建索引 所以 用户可以定义如何切词 哪些分词可以过滤掉 ---- 这个过程叫做分析 ---- es分析之后提供了额外的功能 就是把域中的功能进行排序等的 这种操作就称之为映射机制

**类型用于定义格式 映射就是用来定义格式中的数据如何被分析的**

Es上面四个基本的逻辑组件

这个是es的基本组件 其中mapping映射是相对于其他三个是外部的

ES的集群组件

\*\*Cluster ---- es的集群标示为集群名称 ---如果有多个es集群 你要加入哪一个 就凭借集群名称 ---- 所以集群名称是至关重要的 默认为elasticsearch

--- 一个节点只能属于一个集群

\*\*Node 运行了单个es实例的主机即为节点 用于存储数据分片 并参与索引及搜索操作

节点的标示通过节点名 默认会自动生成随机的字符串当作名字

\*\*Shard 讲索引切割成为的物理存储组件 --- 每一个shard都是一个独立且完整的索引 ---

创建索引 默认shard数目可为其分割5个分片



Shard有两种类型 primary shard和replica shard

主分片用于文档存储 每个索引都会被切成5个主分片 每个主shard都应该有一个副本 --- 有几个 用户可以设定

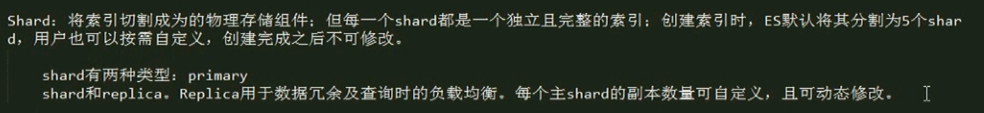
副本分片 可以提高搜索性能 + 冗余数据 ----- 比如 副本数有三个 冗余的三分肯定在三个节点上 这样就可以查询时候的负载均衡

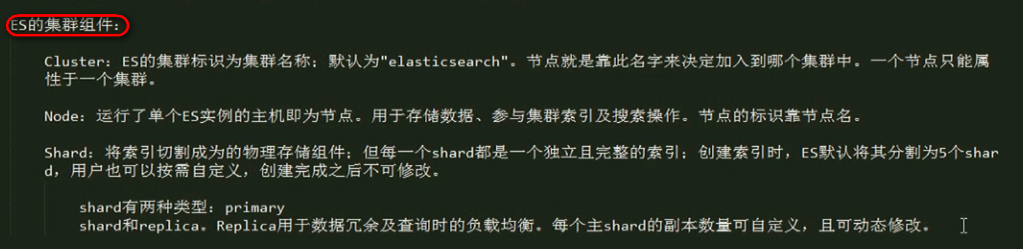
【主分片 和副本分片 有主从的 不是主分片和副本分片的地位是一样的】



每个主shard应该有副本数量 可以自定义

**每个索引被切分成多少个shard创建之后 不能变化 但是 每个shard多少个副本数量是可以动态改变的**





===== ES集群的工作过程 ============

ES启动的时候 通过多播（默认）或者单播 通过TCP协议的9300端口 查找同一个集群中的其他节点 并与之建立通信 ---- 集群中的所有节点会选举出一个主节点 负责管理整个集群状态 --- 也包括集群范围内 决定各个shard的分布方式

----- 站在用户角度来讲 每个节点都可以接受或者响应用户的各种请求 用户无需分别 何为主

----- 集群状态 green red yellow

----- 集群节点坏了 主节点会读取了集群状态的信息 会检查所有可用shard 和副本shard ---- 这个时候 集群状态转为yellow ---- 修复状态

万一刚好有一个节点宕机了 这个节点刚好有某个主shard 此时 需要从副本shard中找一个 提升为主shard ---- yellow时候 各个副本shard都处于未分配模式 --- 就是各个副本shard不可用 ----只能使用主shard查询 不能负载均衡 ----- 吞吐能力很有限 yellow是残破不全的状态 ---- 主节点查询冗余shard 如果不够副本量 就重新复制一个 找到一个节点 --- 直到满足所有条件 --- 此时 yellow状态->green状态（状态转换）

----- 所以 主节点查询各个主shard是否在

如果在 检查副本shard量是否足够

不在 就把某一个副本提升 此时副本不够 再补全

**---- yellow状态的时候 es集群进入修复模式**

这个是es cluster的基本工作流程

Es工作过程 主节点会周期性检查各个节点是否可用 如果不可用 集群将进入修复模式 这个时候 集群进入重新均衡 ---- 一旦节点挂了 加了一个新节点 数据是要均衡的 ES这个rebalance过程是自动的

多个副本shard提升为主shard需要一个选举机制 ---- es都能自动完成

=============== es如何使用

\*\*安装es



Es后面收购了logstash

Es的官网中 有很多产品

 可以看到es可以和hadoop联合使用

Es的使用是基于ruby研发的 --- 就是在jvm上面运行ruby 叫jruby --- python也是 叫jpython --- cpython +jpython

有jruby和cruby

所以 es运行在jvm虚拟机上

\*\*安装jdk  


Es要求jdk 1.7以上





事实上 openjdk有多个版本



 仅仅有jre

如果要jdk 需要headless和devel

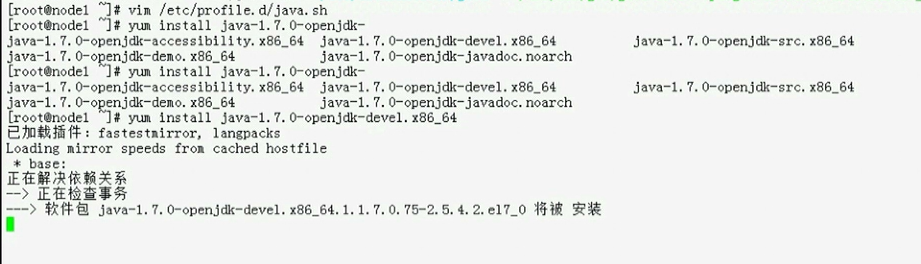


这个下面/etc实际上也是ln





 ---- 实际上不用指定全路径

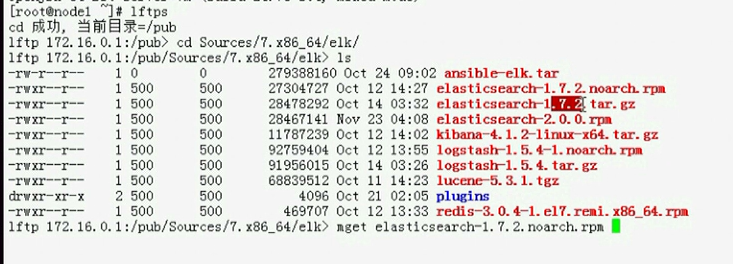




---- 安装elasticsearch

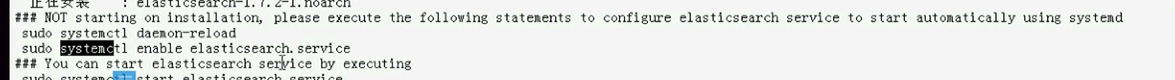


用1.7



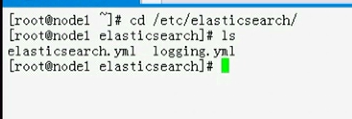






Yum install 过程就告诉你怎么启动 上面的截图就是

修改配置文件







如果只有一个结点 





Es的两个端口 ---- 参与机群的事务 9300(transport tcp port) ---- 接收请求的是 9200(http port)

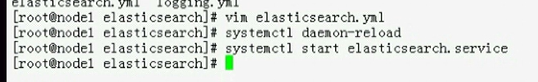


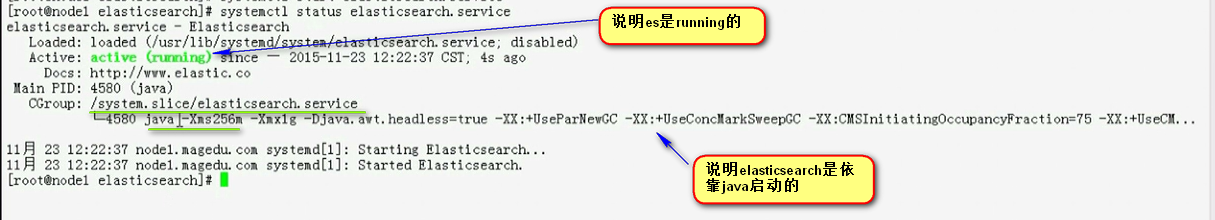
定义集群节点发现的

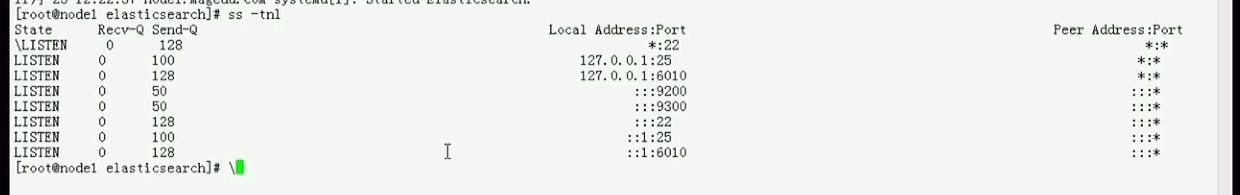
 集群最少的节点数量

 探测其他节点是否可用的超时时间

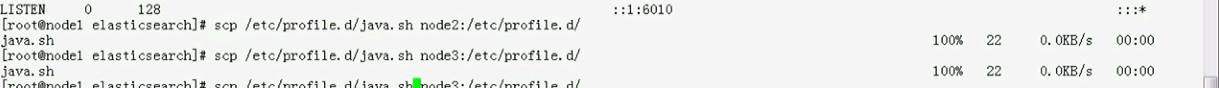
 多播 – false表示ping操作是单播



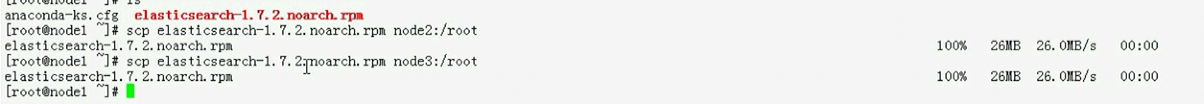




另一个也定义好java的启动流程



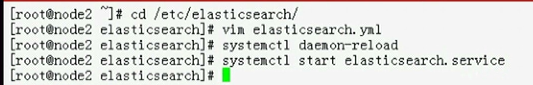
复制下配置文件

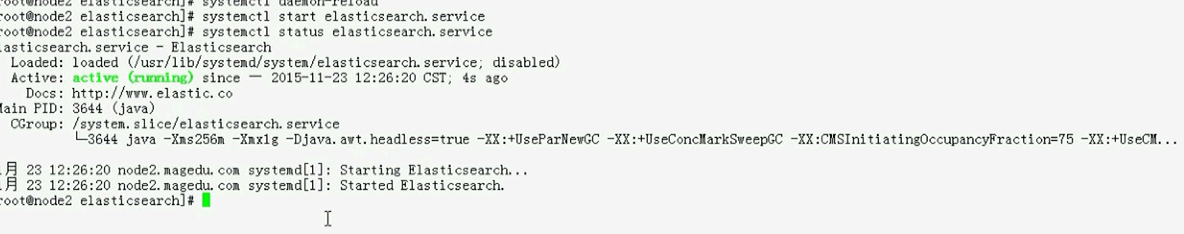




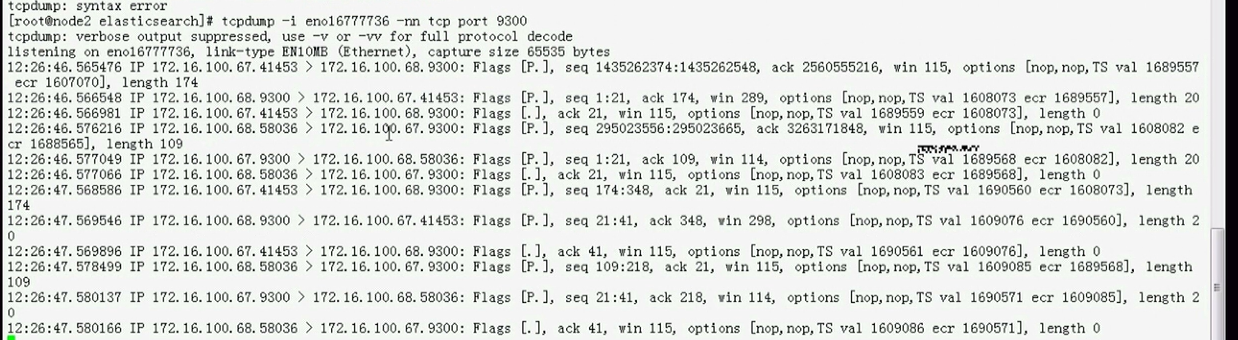
安装yum install

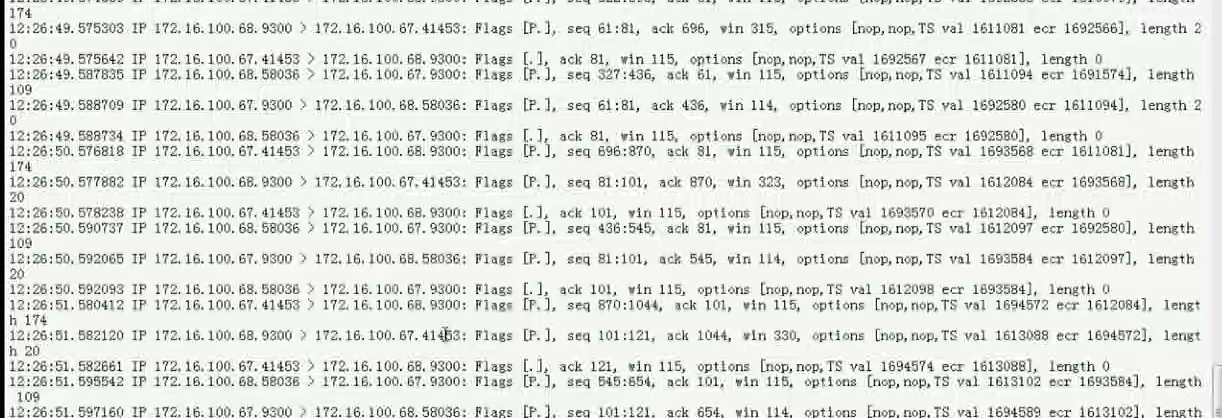
修改配置文件





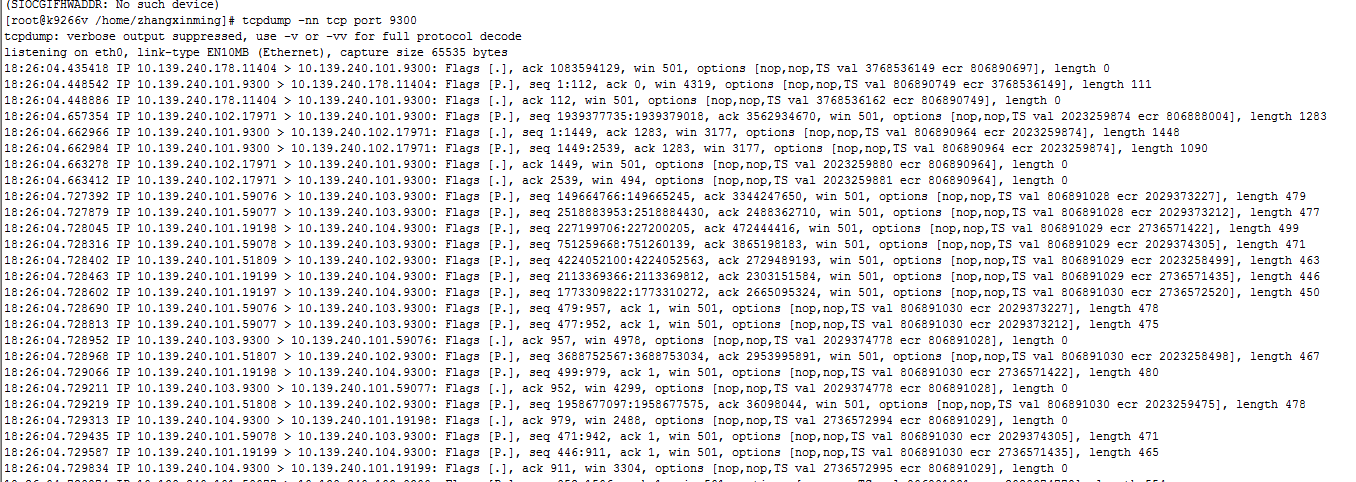
现在这两个节点应该再9300抓包





【

我们的es集群



】

----- es的访问接口 是restful的api

--------------------------------------

Api使用curl命令

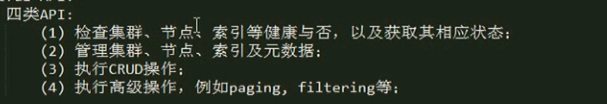
--- 四类APi

\*\***检查**集群、节点、索引等健康与否，以及获取其他相应状态；

\*\***管理**集群、节点、索引及元数据

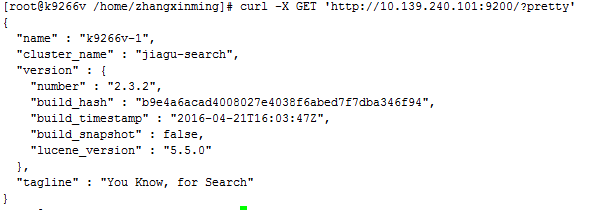
\*\***执行**CURD操作

\*\***执行**高级操作 --- paging filtering等





 这个相当于es的hello world

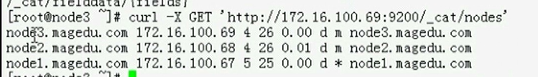




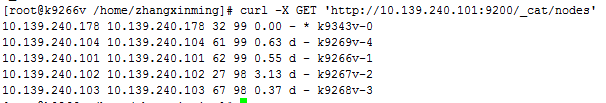
输出格式都是json 但是 这种json的的可读性不太好 ---- \_cat

\*\*es的api都通过\_开头 ---- 通过\_cat api

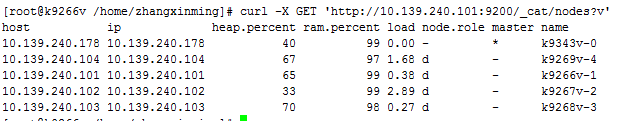
 这个就是 在\_cat这个操作下 支持这么多种操作

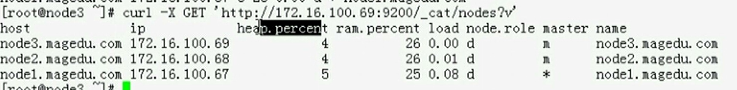


显示集群中的节点



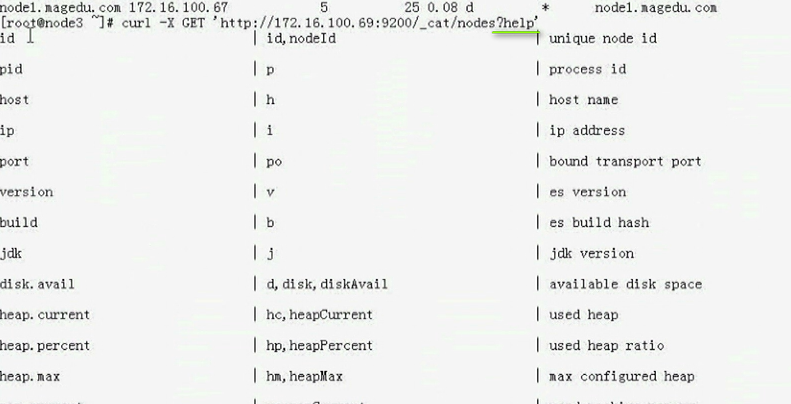
如果显示详细信息 加上?v --- verbose

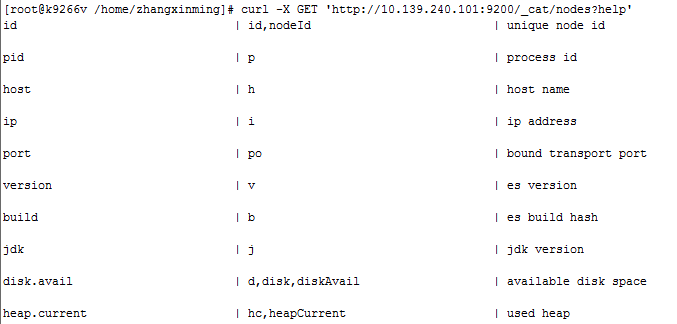




\*表示master

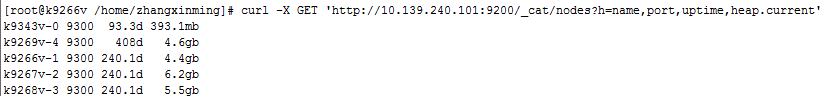
每个命令不会用 可以获取帮助 help







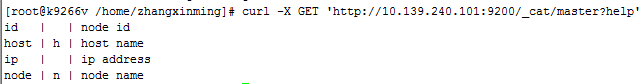
Heap是堆内存的使用量 uptime使用时长

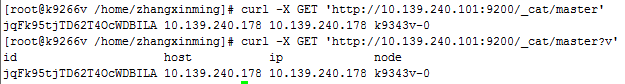


【我们这里 是 几百天了 几个G的内存】

看看谁是主节点



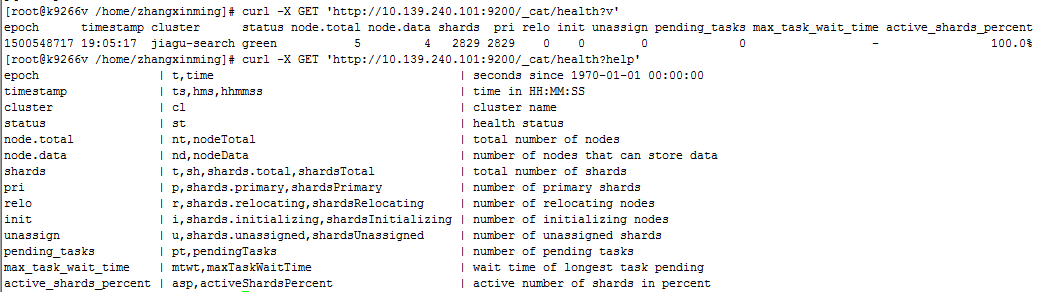




这个id就是随机串







\_cat 仅仅是一个

但是还有很多接口

----- es集群还有很多插件 对于扩充es的功能

这些插件种类繁多 ---- 展示集群状态 无非就那几种