

MongoDB的部署





Mongo要保证数据的可靠 有哪些模式 --- 主从模式 和 副本集模式





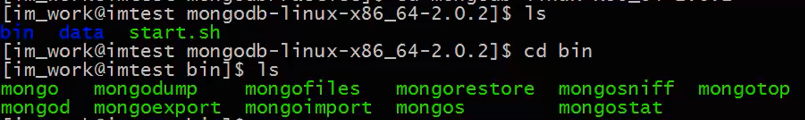
下载和解压



下载了2.0.2的版本



然后解压缩



Bin目录中 有好多程序

mongo 这个是mongoDB的二进制客户端 链接mongodb服务器

mongodump 实际上是对数据的导出和备份 在数据需要导出和备份 用的多 可以在mongoDB运行的时候 进行备份和恢复 这个备份不一定是数据库的实时快照 ---- -比如 数据库有实时更新的操作 这个dump未必能导出

mongorestore 是对mongodb进行导入和恢复 和mongodump配套使用

mongoexport 实际上对集合数据进行导出 cvs或者json格式 ----- 可以指定某些类型 或者 列的数据 指定条件对数据进行过滤

mongoimport 实际上对集合数据进行导入 导入到集合中去 可以是cvs或者json的格式

mongos 提供了统一的读写服务 屏蔽了mongodb底层的差异性

mongostat 实际上是Mongodb自带的mongo检测功率 ---- 固定一段时间 获取mongodb当前的状态 可以输出到终端上 后面mongodb监控会讨论这个工具

mongotop 是用来跟踪mongodb实例 比如集合的读取 写入的时间的花费

mongofiles 是用来操作gridfs 配合使用 在后面girdfs 详细探讨

mongosniff 类似于tcpdump 对网络请求包进行抓取 和 解释 ---- 和tcpdump不一样 tcpdump是可以抓取到所有发送到这台机器的请求 但是mongosniff只抓取和mongo相关的请求包

mongod是mongodb的服务进程的启动工具



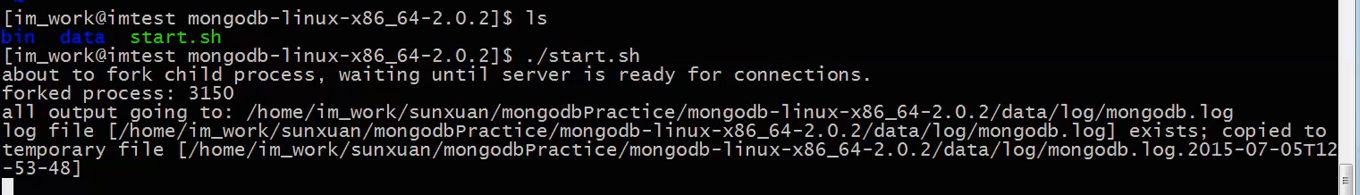
如何启动MongoDB



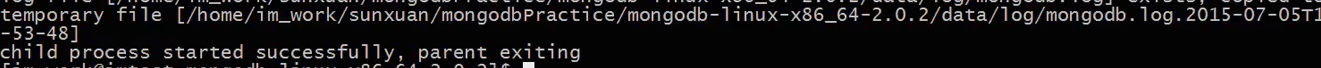
指定了日志的位置 和 数据文件的路径

其他可以使用 ./bin/mongod –help来进行查询

执行./start.sh 来启动mongodb

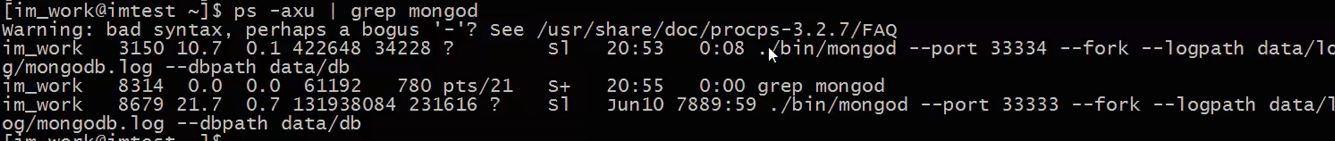


如果最后看到 started successful 就知道mongodb执行成功了



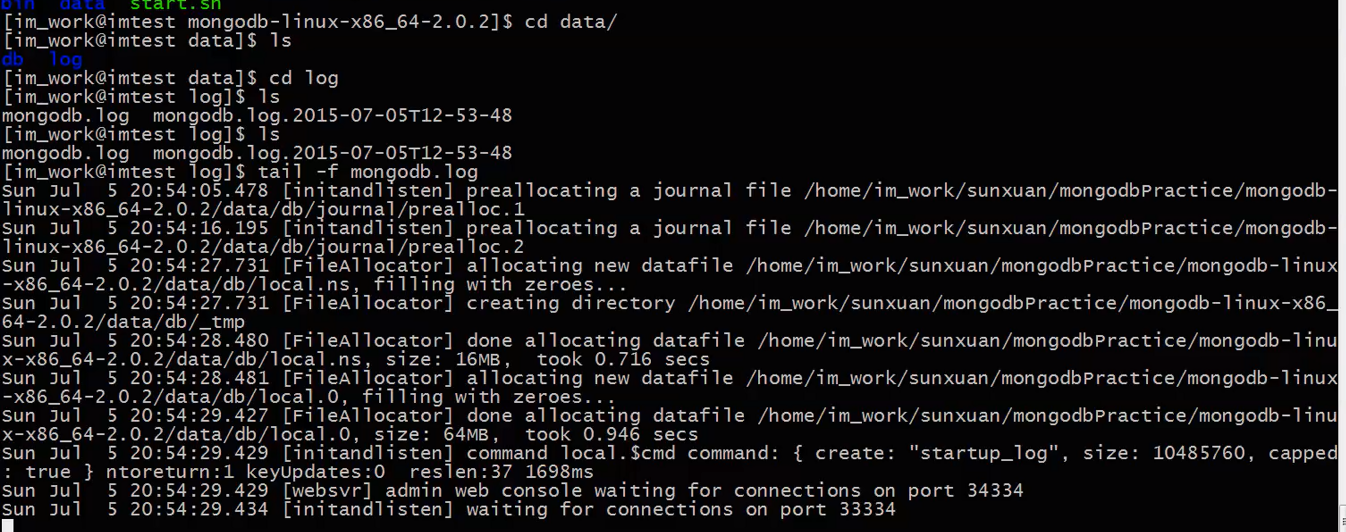
如何验证执行成功了呢？

ps –aux | grep mongod



这样就知道执行成功了

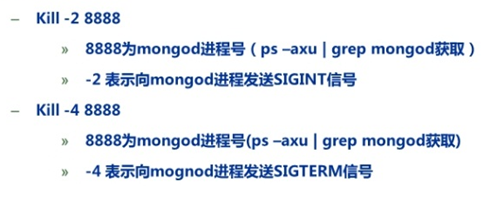
继续确认：指定的日志看下



日志上看到了 waiting for connections on port 34334

  
如何安全停止Mongo的进程？

两种方法





这两种方式为什么安全？

Mongo进程收到-4或者-2的时候 如果当前操作正在执行 收到这两个信号 会等待执行完毕之后 再结束进程 如果打开了一些链接 会先关闭这些链接 然后结束这个进程

会把内存中没有刷盘的数据 强制刷新到磁盘 保证数据可靠 不丢失

还有就是通过MongoDB自身的命令



这个admin是管理员的集合 db.shutdownServer()和刚才的两种发送信号的方式一样安全

下面是不安全的停止mongo的操作

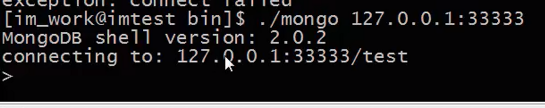


-2 SIGINIT -4 SIGTER -9 SIGKILL

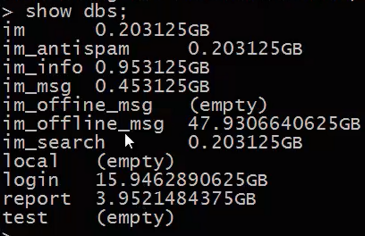
下面简单操作一下数据库



使用Mongo客户端mongo链接mongod启动的数据库

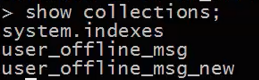


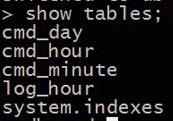
查看数据库

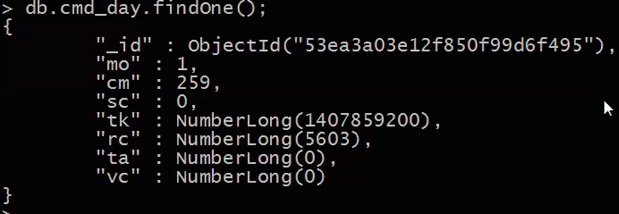


使用use指定要使用的数据库









Exit退出mongo客户端



 跨平台的数据库 都由对应的mongo的版本 不同的版本上 虽然有些差异 但是整体架构上 moingodb是一样的 也就是数据的逻辑结构和物理结构都是一样的



一般上一个mongDB Server包含一个实例 和 这个实例下的多个数据库

如果硬件投入成本有限 运行一个Server上部署多个实例

不建议一台机器上部署多个实例

Mongo中的物理文件的集合 对应的逻辑结果 一些文档 集合 都成为数据库

数据库是由一系列和磁盘文件的物理文件的组成

首先跨平台 + mongoDB server + 数据的逻辑结构 （文档 集合 数据库）+数据存储

下面讲下mongoDB的逻辑结构



逻辑结构是面向用户的 实际上就是层次关系图

和RDBMS对比如下：

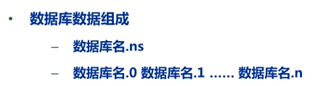


MongoDB最小单位是文档 但是在RDBMS就是行 row

下面是mongoDB数据的存储



每一个数据文件都包含一个.ns文件和若干个数据名开始的 .0 .1等数据文件

 随着存储量的增加 变得越来越大

上图是一个线上支持的例子

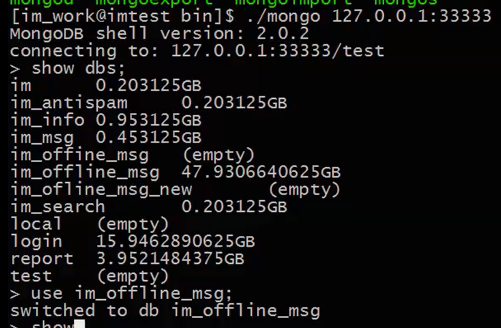
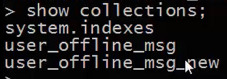


前面是 im\_offline\_msg是库名 后面.ns是一个文件 还有.数字结尾的文件

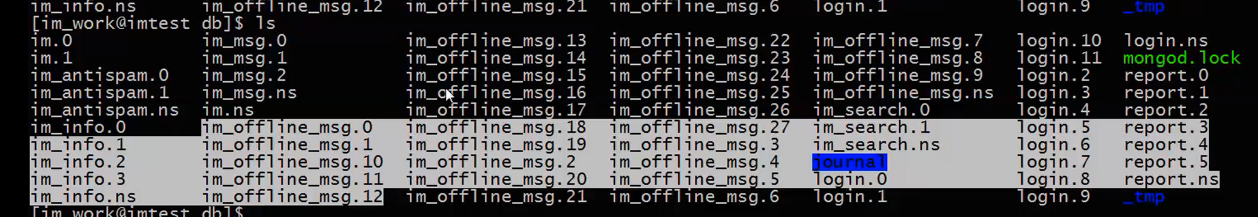
除此之外 大概有27个以数据库名 .数字结尾的文件 实际上就是数据文件 .ns实际上是元数据的文件

元数据文件只有一个 数据文件有多个

演示一下

现在看一下实际的存储



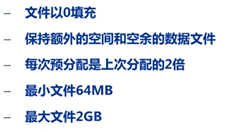
真正的物理磁盘就是这么存储的

怎么分配的？



Mongo内部 数据文件如何分配？

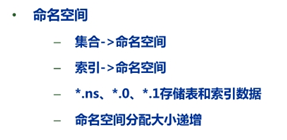
有一个预分配的机制 ----

避免数据库的暴增 导致磁盘压力过大

数据文件预分配一次 都是上一次的2倍

第一个文件是64M 每个文件的大小是2GB 为什么这样呢？这样的机制能有效防止较小的数据库浪费较多的数据库空间

比如我的数据库表很小 就用最小的64M 如果很大 因为预分配 所以 放到自己合适的空间



集合和索引都由命名空间的概念

以刚才的数据库为例：

im\_offline.ns Im\_offline.1 和 Im\_offline.0的文件存储了数据和索引数据

每个命名空间的大小也是随着盘区的变化而变化的

命名空间大致了解



这个是微观的生态系统 集群的构成



Mongod表示是mongod的进程 mongos是mongos的进程的意思

\*\*数据服务节点 就是mongod 真正的数据文件的存储 以数据库名.(ns 1 2 ) ----- 数据都存在这上面

\*\*路由节点 是mongo请求的入口 请求都要经过mongos进行协调 不需要在应用程序上添加路由选择器 mongos本身就是进行路由的 把数据请求分发到不同的数据服务节点上

线上通常多个mongos做路由 --- 实际上是mongos高可用的一种表现 ------ 保证任何一个mongos挂了 都不影响Mongo集群的路由服务 ---- 保证了集群的高可用性

\*\*配置节点 实际上存储了数据库元信息的配置和分片 因为mongos本身没有存储路由信息 和分片信息 只是缓存在内存中 ---配置服务器实际上存储了这些信息

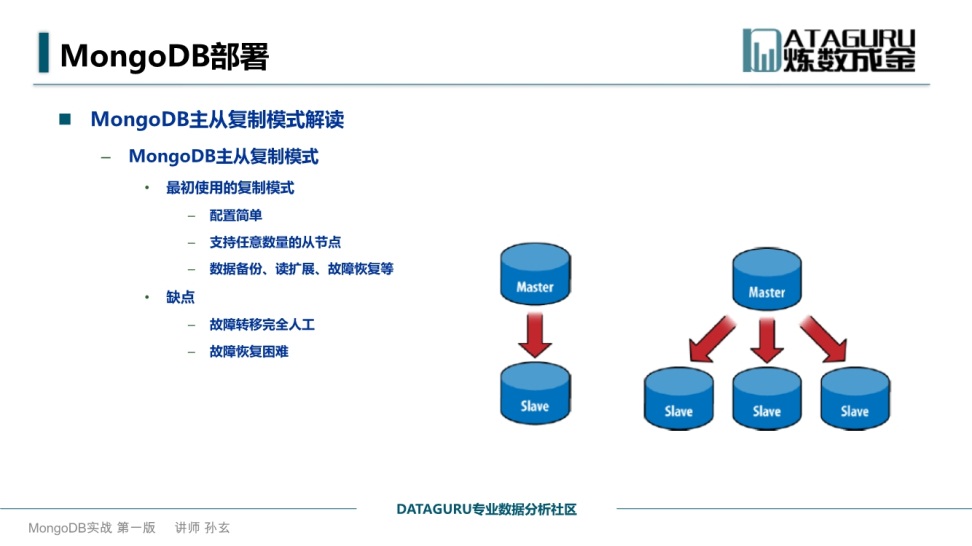
如果mongo关掉了 重启 就会冲配置节点加载moingo的配置 然后通知mongos重新加载配置 准确进行路由

实际生产环境 也是多个配置节点

\*\*投票节点 本身不存储数据 保证某一个主节点挂掉之后 哪一个备用节点提升为主节点 是一个备用作用

MongoDB生态系统就这些

下面是MongoDB集群的复制



主从复制模式 是最简单的复制模式 最初的复制模式

就是一个master 下面是一个或者多个slave

这种模式和关系型数据库很类似

这种复制模式配置简单 支持任意个数的从节点

但是不推荐 这种可以提供数据库的备份 扩展

Master提供故障恢复 和写服务 slave是读服务

【注意 复制的原因 就是 我的最初目的是分担一个数据库节点的压力 那么 无论几个节点 数据要做到一样 才能保证读操作的可靠性 那么多个机器之间怎么保证数据一样？就是复制 这就是复制的原因 复制在集群中的作用】

下面是主从复制模式致命的缺点：

\*\*故障转移完全要人工操作 --- 比如主节点出现故障 管理员必须关闭从节点 重启为主节点 ---- 应用要重新进行配置到主节点 这样 这个故障转移的过程完全人工 并且 对应用程序完全不透明 ---- 应用程序重新介入进来 修改配置 这样 应用程序因为修改配置可能重启 ---- 导致客户端发送来的请求的丢失

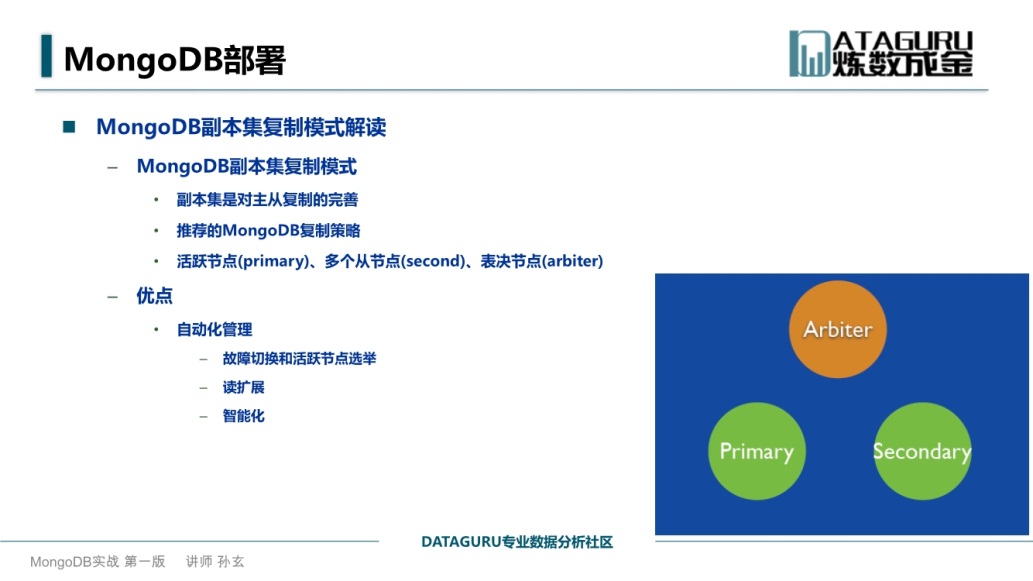
\*\*虽然可以提供故障恢复 但是 恢复比较困难

因为oplog仅仅存在于主节点上 一旦主节点发生了困难 就要在新的主节点上面创建新的oplog 意味着 发生故障的时候 所有的其他从节点都要从新的主节点上同步oplog –

基于这两个缺点 不建议使用主从复制模式 使用副本集的复制模式

【复制的是oplog 不是数据本身 这个oplog就类似于mysql主从保证读写分离的二进制操作文件 op本身就是operation的意思】

副本集的复制模式实际上是主从复制模式的完善：

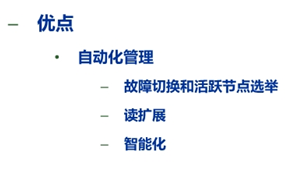


这个模式 除了刚才在主从复制模式的Primary主节点 和 Secondary从节点 ---- 又多了一个新的Arbiter的仲裁节点 这个节点的用处就是自动管理的功能

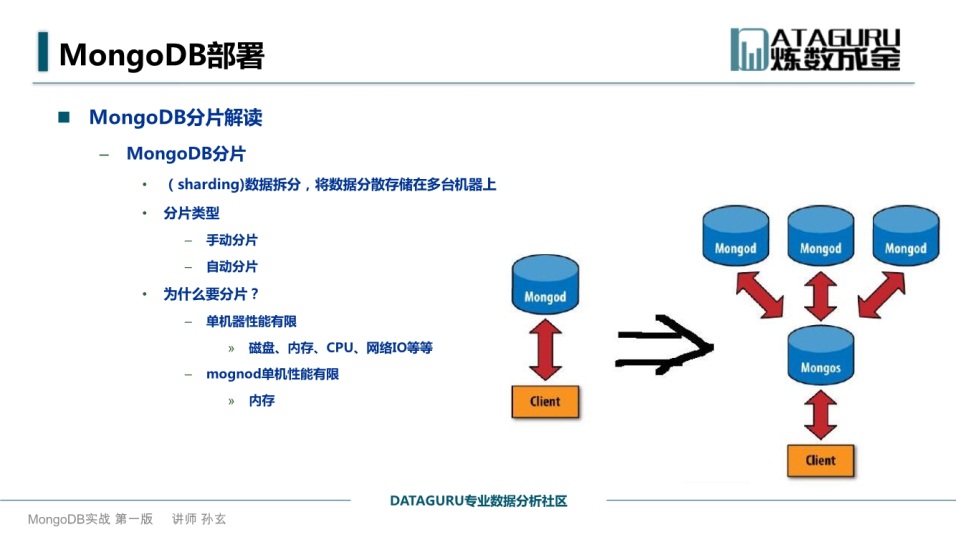
主从模式一旦出现故障吧 必须人工介入 才能完成故障恢复 现在表决节点做这个

一旦主节点挂掉 表决节点就会自己选出来一个从节点 提升为主节点 继续提供服务

Replica set模式自动化比较好



------- 另一个话题 Mongo的分片



分片：就是把数据做了一些拆分 sharding 就是把数据存储到多台机器上

前面的主从模式 或者 replica模式 都是单机上

如果数据量增大 数据要存储到多台机器上



在关系型数据库中 如果一个User表数据过大了 我进行一些水平拆分

Userid可能会拿出来 取模 分成32张表 ---- 应用程序读取的时候 要知道这个分表策略 不是透明的 --- 自己要读取两张表 这样应用程序就是要读取这32张表才行 ---- 所以 关系型数据库的手动分片 也就是手动sharding对应用程序不透明的

在Mongo引入了自动分片 并且对应用程序透明 只要告诉我你要查询啥 不用知道mongo哪些集群 等等 这样很炫酷

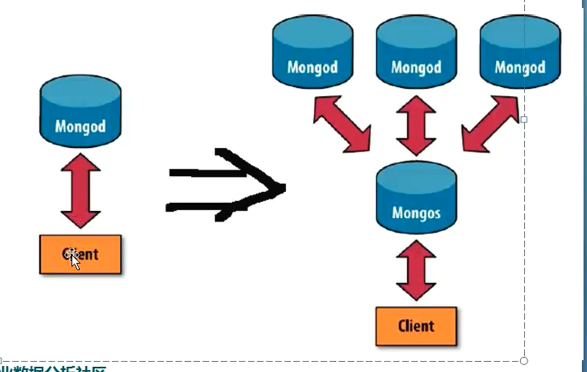
实际过程中 我们会自动分片么？靠谱么？最后一节再讲

为什么分片

(1).单机性能有限 --- 磁盘 内存 CPU 网络IO都有限 一旦达到单机的性能 这样更大的服务无法承受

(2) mongo的单机进程的性能有限

所以要分表或者分片



左边表示开始我的数据全部存储在一个mongo服务器上 后面 由于数据量的增加 数据存储到多个mongo服务器上面

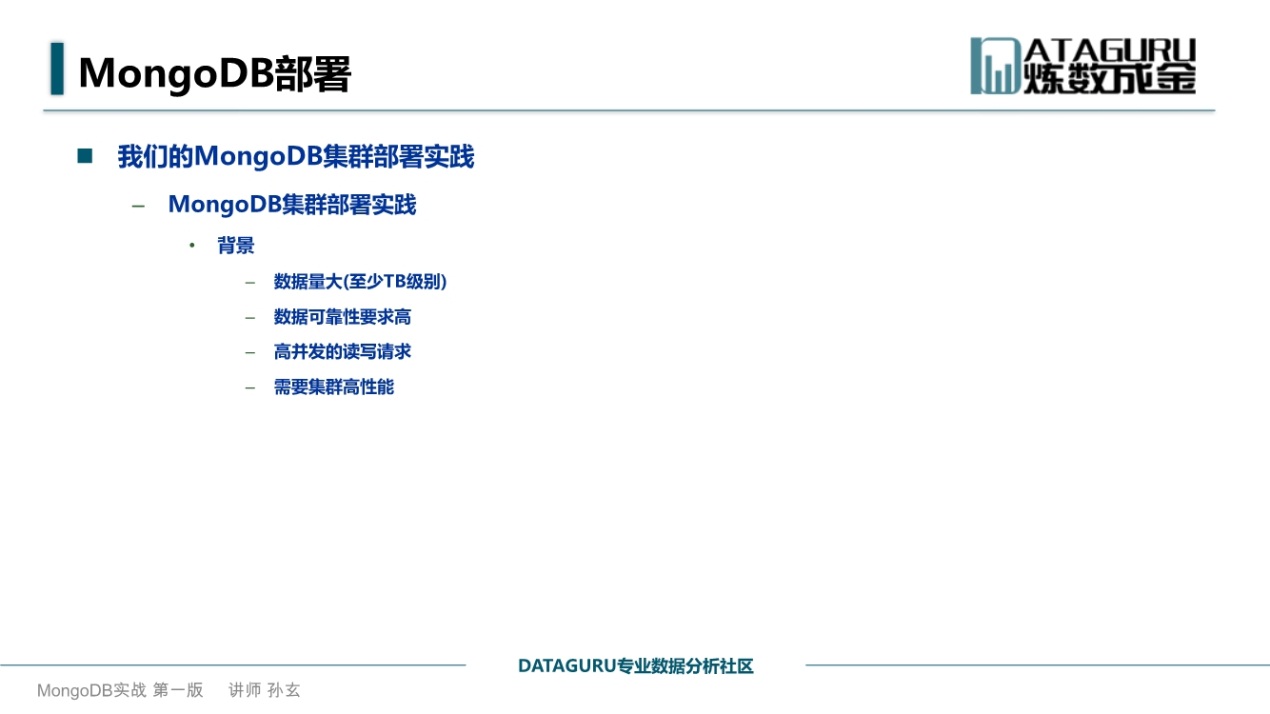
下面讲一下 我们mongodb通常的部署方式有哪些：



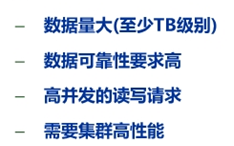
第一种是master-slave模式 这种模式问题多 线上生产不推荐 这种模式比较简单

第二种模式 是replica set模式 推荐的模式 数据量不大的时候 ----单机模式 所有的数据都放在一个机器上

第三种模式 是replica set + sharding  用于数据量比较大的方式



背景：



数据量大 --- 海量数据 mongo

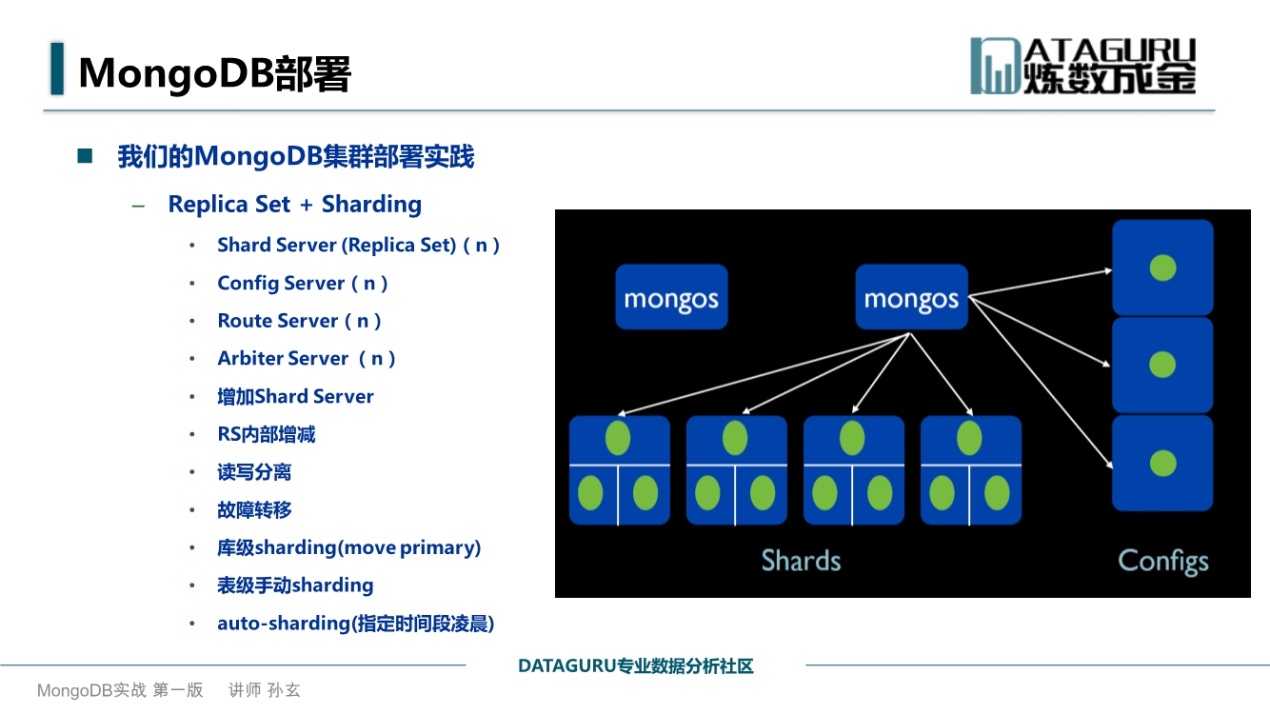
数据可靠性要求高 --- 不允许丢失 就要使用集群

高并发读写请求 ---- 每天请求几十亿次 整体需要集群的高性能 满足这样一个应用背景 如何进行集群的部署？

选择了replica set + sharding的模式

这种能满足业务场景下

这个背景下有以下角色



【replica set相当于hadoop的文件副本 还有 就是 读写分离在Hadoop中没有体现 因为hadoop不是高并发的 但是 对数据文件的分块实际上就是mongo中的sharding 每一个文件块有多个副本 实际上就是replica set ---- hadoop和mongo有很多方面比较类似】

按照hadoop的思想 我们一个数据集 或者或者数据库本身就很大 所以 先分片 编程shards 这样就对应图中左下方的四个shards --- 四个shards合并起来就是一个完成的mongo数据库的数据 ----- 这样图中就有四个sharding sever【每一个sharding sever内部其实是一个小集群而已 这个集群就是副本集架构】

【我们shuaji项目 因为数据量小 所以 没有sharding 只有一个replica set而已】

每一个sharding server内部 又是一个replica set

为了保证高可用 左边的三个是config节点

还有上面两个routing server --- mongos

在sharding server内部 还有一个arbiter server 这边仲裁节点用于主从节点的切换 等等

包括主节点挂掉之后 从主节点提升为从节点

这样的架构的好处：



数据量大了 sharding server不够了 所以 可以添加sharding server集群

如果读的请求大 可以在一个sharding server中添加从节点 分担主压力

【Replica set既保证了数据的可靠性 又保证了高并发的读写分离】

读写请求压力大 可以在一个sharding sever内部进行读写分离

因为使用了replica set的模式 故障恢复很容易

现在 是库级别的sharding 更详细的后面介绍 表级别的sharding 没有做自动的sharding

表级别的是手动的sharding

指定某些时间段 可以开启自动sharding的功能 凌晨 压力小 --- 某些场景才使用auto sharding



可以是三台的物理机 也可以是三台虚拟机



延迟从 ---- 没有延迟从 也OK

下面是Mongos有三个

Config server也是三个

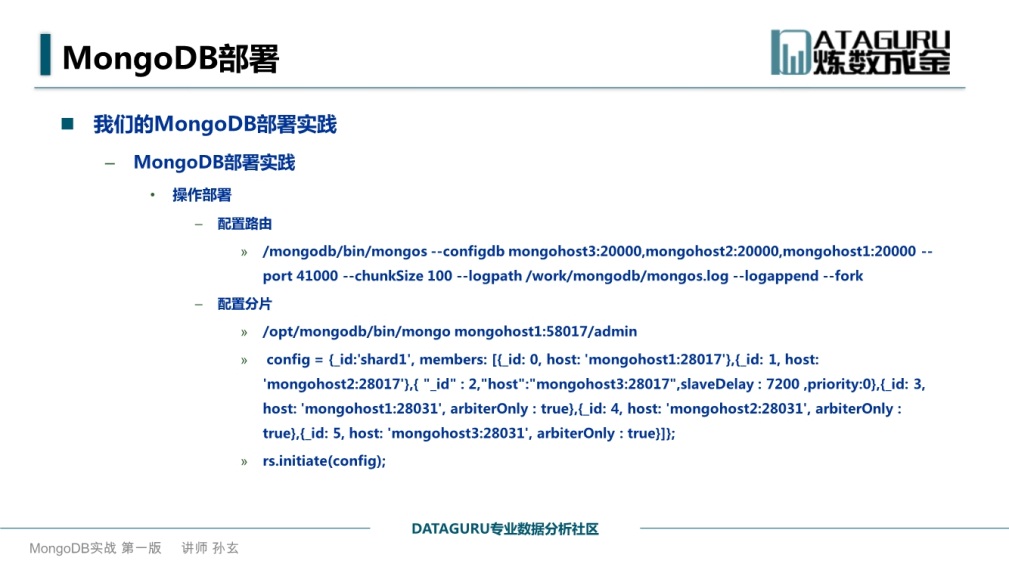
Arbiter server有6个 线上其中一台这么部署

以上是部署需要的资源



启动之前 配置Hosts和hostname

执行完上面两步之后





配置分片 就是在mongo的admin数据库中

生成下面的配置 每一个分片的数据节点 投票节点

Config =

{

\_id:shard1, members:[

{\_id:0, host:mongohost1:28017},

{\_id:1, host:mongohost2:28017},

{\_id:2, host:mongohost3:28017, slaveDelay:7200, priority:0},

{\_id:3, host:mongohost1:28031, arbiterOnly:true},

{\_id:4, host:mongohost2:28031, arbiterOnly:true},

{\_id:5, host:mongohost3:28031, arbiterOnly:true}]}

就配置了第一个分片 shard1 这个shard上面的Replica set模式由哪些机器组成（数据服务节点 + 投票节点）

配置好分片之后 执行rs.init(config)这条命令



配置好之后 增加一个库级分片

第一步 链接admin的数据库 接下来添加下面的分片 把所有的分片对应的机器放到一组

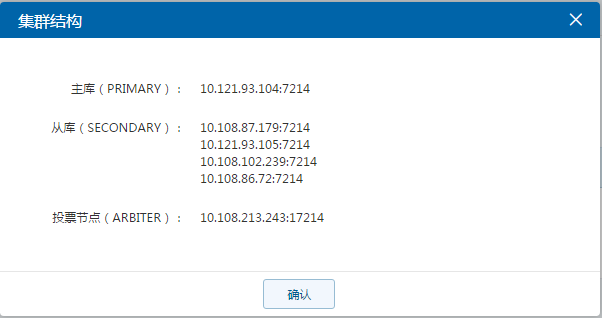
最终开启库级的sharding





【我们的shuaji的项目

Mongo的集群架构是：



可以看到 我们采用的部署模式 仅仅是replica set模式 不是replica set+sharding模式

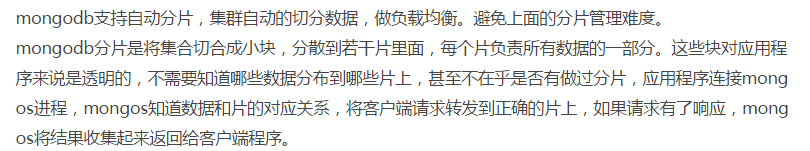
我们现在数据有20多G 不算大 所以 replica set就没有问题了

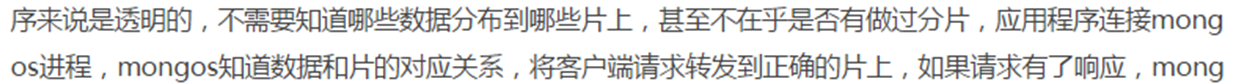
在我们的线上生产库 投票节点有一个 arbiter

一个主库 primary

】

摘抄出来的一算文字





分片相当于hadoop中的文件分块 ---- 每一个块有副本 存储在不同的DataNode上面

副本相当于mongo中的replica set

这样 一个源集合就对应多个片 ---- 是mongos进行路由

对应的是

一个hadoop文件对应多个块 ---- 是namenode进行路由

所以 mongo hadoop

文件块 集合或者库片

Mongos NameNode

【代码中的replica set name 这个就是一个标示 那么这个含义就是 副本集的名称 表示 你要使用的是哪一个副本集 因为一个副本集有多个mongod实例 之前 管理员一定是配置好的 这个replica set name就是会捆绑一堆IP 这些ip上面有mongod实例】