MongoDB实战 第一课



课程目标概括如下：让同学真正掌握MongoDB 有哪些特性 &&有哪些坑 如何避免 最终应用到生产实践



随着大数据时代到来 Nosql兴起 ---- 从关系型数据库转到Mongodb 非常方便

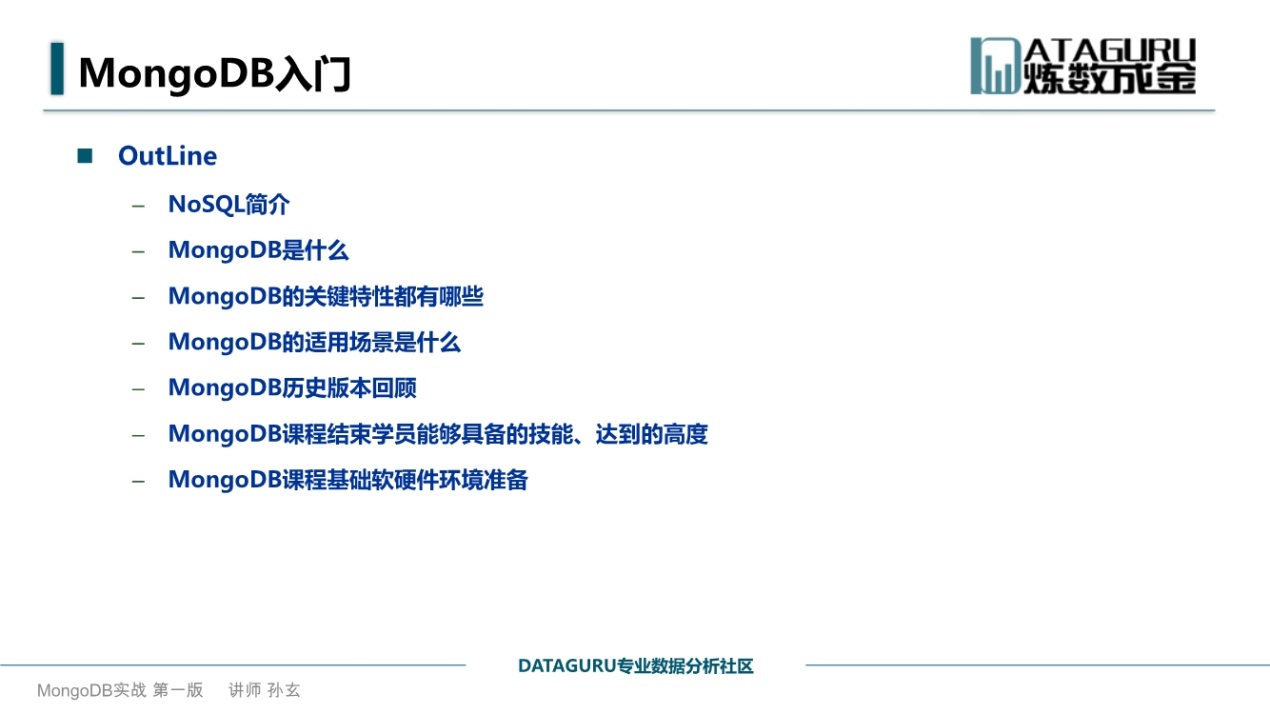
随着MongoDB3.0在写入效率 和 数据压缩率方面 进一步提升

现在使用的是2.0 也是比较稳定的 随着3.0的发布 有理由相信占有率会提升

课程会采用循序渐进 步步深入 会结合自己线上的情况来讲解

通过本次课程的学习 真正掌握Mongo并且应用到生产环境

下面是第一节课的学习 ---- mongodb的入门



首先介绍一下NoSQL简介：



NoSQL ---Not Only SQL 是SQL的有力补充

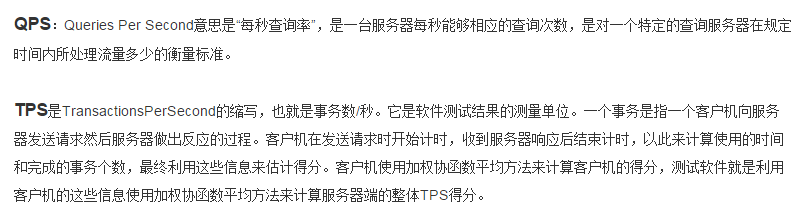
目前对NOSQL的解释是 是非关系型的数据库 --- 是key value的存储

适用RDBMS就使用关系型数据库 否则就不使用

NoSQL相对于传统型数据库的优势：主要是表现在处理海量数据的能力上面



普通的集群 是处理每秒几万的QPS或者TPS 甚至是几十万的QPS是很轻松的

【】

传统的RDBMS提供了关联数据的处理 join操作 这些操作必须放在一个服务器上面处理

但是 对于NoSQL设计之初 不提供复杂的数据处理 数据是分布式存放的

写入的操作性就很高了

---- 同时 读取也可以并发读取 读取操作和写入操作同样高效

\*\*NoSQL的扩展性 通常 为了提高数据库的性能 ----- 两种选择 一种是Scanup----机器本身纵向扩展 比如增加内存 增加硬盘 增大CPU核数 提升性能 等等 ---- 纵向的提升有局限性 到达一种程度 就会成本非常高 另一种Scanout

NoSQL是天然的分布式数据库 通过扩展机器数量 来进行 ---- 理论上Scanout是无限扩展的

以上就是NoSQL在处理能力（读写能力 + 扩展方面）

市场上流行的NoSQL数据库：



键值存储 列存储 图形数据库 文档型

具体比较如下：

键值对的NoSQL产品：

数据库是kv存 ---- 通过key 快速获取value ---- 通常通过HashTable来实现

根据保存方式的不同 分成临时型 永久型 或者两者兼有的

临时存放型 风险就是内存中的数据会丢失 比如进程重启 机器宕机 内存满了—设置了4G或者8G 这样根据LRU就会把老的数据冲掉 ==== 这样通常是用来进行数据缓存 典型的代表就是memcached

另外就是永久的存储 ---- 把数据持久化到硬盘上 数据不会丢失 数据保存在磁盘上 读写速度 和 内存的读写速度有差距 ------ 典型的就是Tokyo Cabibnet

两者兼而有之的就是 Redis ---先把数据写到内存 在满足一定条件下 就会写到磁盘上

Redis相对于Memcached而言 提供更加丰富的数据类型 除了String 还有 list 等等

Redis在很大程度上补充了Memcached的不足 比如 计数 比如使用集合的场景

列存储数据库：

以列为单位进行数据的存取

RDBMS是以行位单位进行数据的存取

典型的代表 Cassendra和HBASe都是列存储数据库

图形数据库：

使用图形model

处理图的数据 理论很大 要求方便扩展到多台机器上 自己定义接口



文档型数据库：

同KV类型的数据库比较类似 可以看做key-value类型的升级版 ---- 除了普通的kv 还可以嵌套键值 ==== Value里面又可以嵌套新的kv的形式 支持多级嵌套 ---- 可以进行更加复杂的查询 ---- 非常容易和RDBMS对应

典型代表 就是MongoDB CouchDB





这样 以RDBMS来描述人这地址 要使用两张表 进行join才可以 但是 使用MongoDB 使用它这种灵活的可以嵌套的key-value 就是非常方便的 ---- 一个People的嵌套文档就可以了

 ---- C++开发的工具



 面向文档的存储 Mongo非常关键的特性

多种索引的支持

非常重要 就是复制和高可用

很好的卖点 就是自动分片

丰富的查询支持 ---- 很类似RMDBMS的查询 很相似

对更新的支持 ----- 对更新的支持 也类似于关系型数据库

 map-reduce功能 ----- 通过写一些Map或者Reduce的job来实现离线的计算和分析

 ---可以对大数据 或者大文件的存储 ---- mongo中的每一条文档 都是有大小限制的 当超过大小的时候 就没有办法存储在数据库中 ---- 这个时候 就可以使用GridFSB来进行存储【和Hadoop非常类似 就是我的分布式文件存储HDFS和分布式计算框架MapReduce是在一起的 Mongo中同样具有对应的GridFS+Map/Reduce ---- 从这里可以见Mongo可以提供大数据离线分析】



面向文档的存储特性

以下几点：丰富的数据类型 和 Schema的考虑

实际上通过mongoDB这一集的介绍 知道Mongo支持丰富的数据类型 整形 日期类型 数组 ---- 另外一点 文档不用事先定义Schema 但是传统的RDNMS必须先定义Schema 就是有哪些列 每一个列是什么数据类型 要严格遵守 但是 mongo中就是可以不做这个限制

Mongo就是把文档融合成集合 ----- 每一行是一个文档Record 整个是一个Collection

实际的工作中 不建议 每一行都不一样 后面的课程会讲到

实践中 建议一个集合中的文档 最好统一

简单举例 加入有一个QQ集合 有消息的文档 ---- 最好统一：消息的ID 消息的发送方 消息的接收方 便于进行维护和升级



Mongo的第二个特性：类似于关系型数据库的索引 支持丰富的索引类型

和关系数据库的所以很类似

存储的文档必须有一个\_ID字段 --- 理论上是任意的数据类型的 可以理解成主键 ---- 默认是ObjectID类型 使用了12字节的存储空间 实际上24bit的字符串 ---因为这个ObjectID非常占用内存 ----- 所以建议 如果 在应用端 能替换ObjectID 就替换

在一个集合Collection中，\_ID也保证了记录的唯一性 ---- 就是同一个集合中 文档的\_ID是不能够重复的【这个和集合的特性非常类似 集合中的元素具有唯一性】

但是不同的集合中 文档的\_ID是可以一样的

\_ID是用来在一个集合中唯一的 多个集合中可以相同 也可以不同

任何一个集合中 每一个文档必须有一个\_ID

----

Mongo中的一些索引 使用了BTree 针对多种查询 进行了优化 通过允许使用多个索引 对特定的查询进行优化

----- 注意Mongo中 每个集合最多支持64个索引 能够支持数据库的大多数索引 比如唯一索引 联合索引等等

实际上还有地理空间的索引 --- 为坐标平面查询 提供了查询 --- 比如经纬度 附近的餐馆有哪些美食的查询----- 基于坐标的索引



复制和高可用

Mongodb的集群的复制 提供了两种数据形式的复制：主从级别的复制 和 副本级级别的复制

主从复制模式 ---- mongo最常用的复制模式 最灵活 常见用途 比如线上压力太大 扛不住了 线上要求增加一台机器 分担主的压力 这个时候 使用主从复制模式 多挂几个从 来分担主的压力 ---- 这个就是主的扩展

另外就是数据备份 ----- 挂一台从服务器 用来进行数据备份

故障的恢复 ---- 主挂了 现在进行恢复 可以这个时候 把从库提升为主库 这个在故障恢复方面 也是mongo的卖点

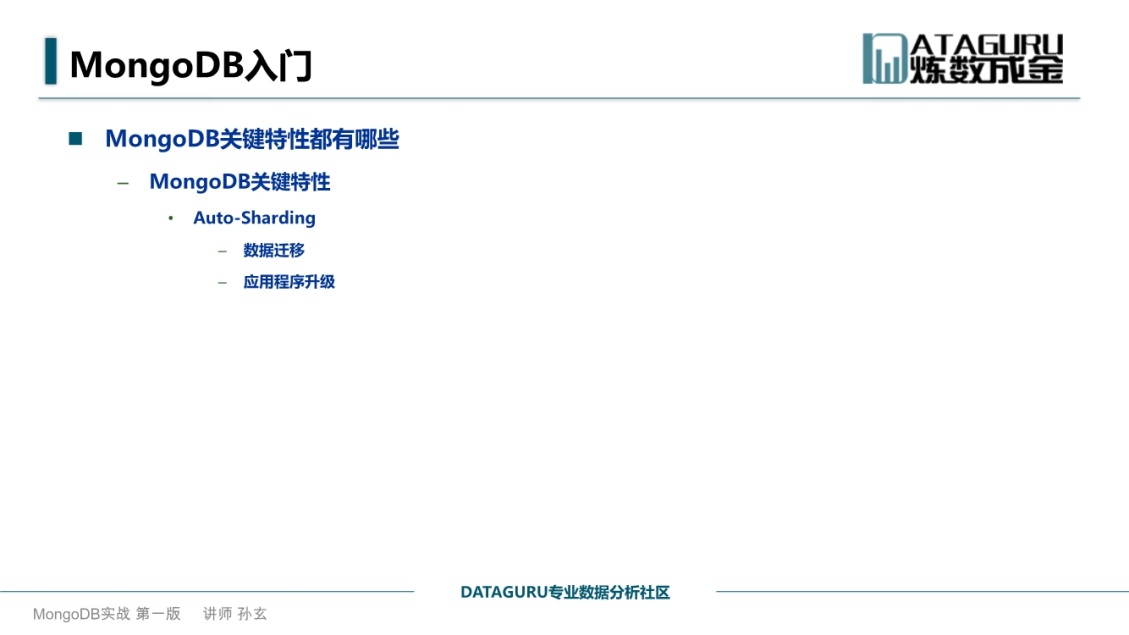
这个和传统数据库的主从 是没有本质区别

----- 下面是副本级的复制 相当于主从复制模式 提供了自动的故障恢复

在存储服务器或者网络发生故障的时候 提供自动选出主库 来对外恢复服务

Replication有哪两种？主从和Replica

【主要就是Nosql是集群的 mongo也是集群的 所以mongo的集群之间就要数据同步 就要复制 所以有Replication】



这个Auto-Sharding就是 应用程序不用升级就可以实现数据的自动迁移

实际的生产环境中 随着数据量的增加 在一台服务器存储完整数据库就变得不现实 ----- 这个时候 就需要分布在多台机器上 来减小这种压力 --- 这个就称为分片

在关系型数据库 进行手动分片 把数据分布在多台机器上 --- 手动分片的确解决了负载的问题 但是 最明显的问题 就是数据迁移非常困难

假设现在我的数据库遇到压力了，数据要求迁移 但是 又要求我线上的服务接着进行 这个时候要求数据迁移必须对应用程序无缝

当然在MySQL有无缝迁移的方案 ---- 有一定的成本 即使迁移成本高 也对应用程序不透明 必须应用程序对数据库的读写路由的改变 升级 这一点是比较困难的

以上这两点比较困难

但是对于mongoDB设计中 核心工作就是分片的问题 无需担心水平扩展的问题 该怎么迁移就怎么迁移 应用程序不需要进行任何的升级 都是Mongodb自动搞定 这就是应用程序的无缝和透明 ---- 看似完美 但是实际生产环境中靠谱么？数据量大 读写比较频繁 开启了Auto-sharding 会遇到什么问题呢？

【关系型数据库的分片是手动分片 但是对于Mongo是auto-sharding的 也就是自动分片】



丰富的查询支持和更细支持 --- 类SQL

和传统的关系型数据库是差不多的



左边的是关系型数据库的SQL 右边是Mongo中的类SQL

Find是和SQL中的Select对应的



投影查询

Select a,b from users < ---- > db.users.find({}, {a:1, b:1})



过滤查询

Select a, b from users where age =33 < --- > db.users.find({age:33}, {a:1, b:1})

查询结果的排序



Select \* from users order by name desc < --- > db.users.find().sort({name:-1})

模糊查询



Select \* from users where name like “%joe%” < --- > db.users.find({name:/Joe/})

【find中的第一个{}是对应where条件 第二个{}是对应sql中的投影结果的字段 ---- 可以把where+投影看成查询的输入 所以 先后是两个json ----- 注意一下模糊查询 SQL是%% 到这里面就是//进行对应 ----- 查询结果如何操作 就要通过新的函数 sort sort的参数同样是json】

在更新方面 提供了insert remove和update的操作 来实现

使用过关系型数据库的话 这个过度非常简单



这个是MongoDB中离线的分析和聚合的功能

实际上就是map reduce 提供了聚合的工具 在mongo中实现并行统计是非常方便的

Mapreduce是可以轻松并行化到多个服务器的方法

可以拆分问题 当所有的机器都完成的时候 再把结果汇集起来 再汇聚成完整的结果

一个是map阶段 另一个就是reduce

最开始就是map ---- 操作的就是集合中的多个文档 --- 一个文档要么无作为 要么就是产生一个key value

然后中间环节 shuffle ---- 对这些ks vs进行shuffle 也就是对这些kv进行洗牌：按照key进行分组 并将产生的键值组成列表 放在对应的键值中

Reduce ---- 就是把列表中的一些值化简为一个单值 --- 这个值被返回 –接着进行洗牌 直到每一个键的列表只有一个值 ---- 这个就是最后的一些结果

Map/Reduce虽然能做一些数据的统计和分析 但是map reduce是不能用于实时环境的 因为分析效率不是非常高 ---- 可以后台起一个mr 把mr的结果保存在一个集合中 –然后对这个集合进行实时的查询 ---- 如果对线上的服务使用Mr 很可能造成线上请求的超时

---- 如果有实时的请求 先离线 计算放到结果集中 最后 对结果集提供实时的查询



这个是Mongo对大文件的存储

为什么需要girdfs ---- mongo中的每一个文档 存储的上限时16M ----- 超过这个大小 就不能在集合中存储 --- 要考虑其他的存储方式

gridFS就是mongo中存储大型二进制文件的一种机制 ---- 怎么存储呢？

处理的思想：把大文件分块 每一块作为单独的文档存放 【单独的文档 大小是16M 但是 我们的HDFS 在hadoop1中 块的大小是64M 2中是128M 这个还是有差距的】

这样使用两张表存储数据

Fs.files 和fs.chunks

Fs.files包括了数据块的元数据对象 ---- 必须 \_ID length chuncksize（单位是字节）uploaddate文件存入gridfs的时间戳 文件的md5 ---- 存储了上传文件的元信息【类似于HDFS中的NameNode】

Fs.chunks 这个集合存储了大文件的分块二进制数据 【类似于HDFS的DataNode】

首先对文件进行分割 ----- 把分割的块存储在fs.chunks这个集合中 --- 对应的元数据 存储在fs.files

通过这两张表 可以方便进行大文件的上传 下载

下面是MongoDB的适用场景



MongoDB的适用场景 简单分析：



有些不知道Schema 这个时候 适用Mongo

下面分析一下应用场景：

第一个是Web应用程序



Web应用程序的特点 有很多数据类型 特定数据的上传 下载

MongoDB和关系数据库能力基本一样

比如 表量小 关系类型数据库是People和Address 但是mongo中由于Mongo的文档是嵌套的文档 所以 一个People --- 嵌套的json就可以表示了

支持丰富的查询

多种索引 丰富的查询+多路索引 让应用程序实现各种查询

在web应用合适的原因



对敏捷开发团队比较合适的地方：敏捷开发就是快速构建应用程序的渴望 这样 schema不限的mongo就节约了时间



用mongo进行日志分析越来越多流行

比如github

来源于速度和下面的两个关联特性：



原子更新 让客户端高效增加计数器

固定集合 存储空间大小固定

一旦写满 就会循环写

目标原子更新+固定集合 更加适合使用mongo进行日志的分析



下面是使用缓存的场景

使用MongoDB 因为内存使用MMP --- memory map ---- 就是热点数据已经存到内存中 这样本身就具备了缓存的功能 ----- mongo的读写速度非常快

使用mongo作为存储 把数据持久化到磁盘中 –保证了数据的可靠

使用Mongo 既提供了持久化的功能 又提供了缓存的功能

Mongo提供了写内存和写磁盘 不会出现脏数据 所以 不会出现数据一致性的问题

这种相对于传统的MySQL+Memcached的方案

--- 但是这种方案 MySQL数据变化了 就没有办法保证Memcached的数据是一致的 --- 这就出现了脏数据 数据就不一致了 非常难查 但是Mongo的数据就一份 所以 非常好



Schema变动的应用来说 mongo非常适合 这个场景使用mongo非常好

MongoDB的历史版本



历史不长 但是 Mongo诞生于一个比较宏伟的项目

就是Paas项目 既有服务器 又有web

这里面有一个数据库的技术 ------ 就有了mongoDB

mongoDB1.0发布于2009 --- 目标是高度伸缩性 + kv存储

每三个月就发布一个主要的版本

偶数发行版 就是稳定的版本 奇数版本 就是

在2011年 mongdb进入2.0版本时代

相比1.0 增加了一些压缩 以前是通过repair压缩 2.0 增加了compact进行压缩

还有就是写操作在内存中不能完成的时候 ---进行了写锁的优化

还有一个就是Mongo在并发的时候 为每一个客户端增加一个线程 ---线程量大的时候 线程消耗大量的内存 就要考虑进去

1:03

