**Mongodb**

## MongoDB概述

**1 MongoDB介绍**

MongoDB是面向文档的非关系型数据库，不是现在使用最普遍的关系型数据库，其放弃关系模型的原因就是为了获得更加方便的扩展、稳定容错等特性。面向文档的基本思路就是：将关系模型中的“行”的概念换成“文档（document）”模型。面向文档的模型可以将文档和数组内嵌到文档中。因此，实际中可以用一条数据表示非常复杂的结构。

**MongoDB没有预定义模式：文档的键(key)和值(value)不再是固定的类型和大小，而且根据需求要添加或者删除字段变得更容易了。**由于没有模式需要更改，通常不需要迁移大量数据。不必将所有数据都放到一个模子里面，应用层可以处理新增或丢失的键。这样开发者可以非常容易地变更数据模型。实际应用中，随着数据量的增大，数据库都要进行扩展。扩展有纵向扩展和横向扩展。纵向扩展是使用计算能力更强的机器，也是最省力的方法，但是很容易达到物理极限，无论花多少钱也买不到最新的机器了。横向扩展就是通过分区将数据分散到更多的机器上。MongoDB的设计采用横向扩展。面向文档的数据模型使它很容易地在多台服务器之间进行数据分割。还可以自动处理跨集群的数据和负载，自动重新分配文档，以及将用户请求路由到正确的机器上。开发者根本不用考虑数据库层次的扩展问题，需要扩展数据库时，在集群中添加机器即可，MongoDB会自动处理后续的事情。MongoDB有如上各种特性，但为了达到这些，他也放弃了关系型数据库的某些功能如表连接join和复杂的多行事务。





**2 MongoDB使用场景**

mongodb的主要目标是在键/值存储方式（提供了高性能和高度伸缩性）以及传统的RDBMS系统（丰富的功能）架起一座桥梁，集两者的优势于一身。

**（1）MongoDB适用于以下场景**

  a.网站数据：mongo非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。

  b.缓存：由于性能很高，mongo也适合作为信息基础设施的缓存层。在系统重启之后，由mongo搭建的持久化缓存可以避免下层的数据源过载。

  c.大尺寸、低价值的数据：使用传统的关系数据库存储一些数据时可能会比较贵，在此之前，很多程序员往往会选择传统的文件进行存储。

  d.高伸缩性的场景：mongo非常适合由数十或者数百台服务器组成的数据库。

  e.用于对象及JSON数据的存储：mongo的BSON数据格式非常适合文档格式化的存储及查询。

**（2）MongoDB不适合的场景：**

  a.高度事物性的系统：例如银行或会计系统。传统的关系型数据库目前还是更适用于需要大量原子性复杂事务的应用程序。

  b.传统的商业智能应用：针对特定问题的BI数据库会对产生高度优化的查询方式。对于此类应用，数据仓库可能是更合适的选择。

  c.需要SQL的问题。

**3 物理内存和虚拟内存**

物理内存所指的就是你主板上所插的可以看到的内存条，它的容量有64M、128M、256M、512M等不同规格。

内存在计算机中的作用很大，电脑中所有运行的程序都需要经过内存来执行，如果执行的程序很大或很多，就会导致内存消耗殆尽。

为了解决这个问题，Windows中运用了虚拟内存技术，即拿出一部分硬盘空间来充当内存使用，当内存占用完时，电脑就会自动调用硬盘来充当内存，以缓解内存的紧张。举一个例子来说，如果电脑只有128MB物理内存的话，当读取一个容量为200MB的文件时，就必须要用到比较大的虚拟内存，文件被内存读取之后就会先储存到虚拟内存，等待内存把文件全部储存到虚拟内存之后，跟着就会把虚拟内存储存的文件释放到原来的安装目录里了。

虚拟内存是作业系统在硬盘上建立一个档案，把物理内存中不常用的部分拷贝起来，再把那个部分的物理内存清空，方便别的程序写入。虚拟内存的大小是可以自定，但是大小通常在物理内存的1到2倍之间，太大的话，虚拟内存的效能会下降。因为虚拟内存在硬盘上，所以它的速度是取决于硬盘的存取速度、碎片的多少等。另外，因为它不像物理内存一样，资料与停机后消失，所以虚拟内存是可以透过解密方法读取其中的资料。

**4 Memeory-Mapped Files**

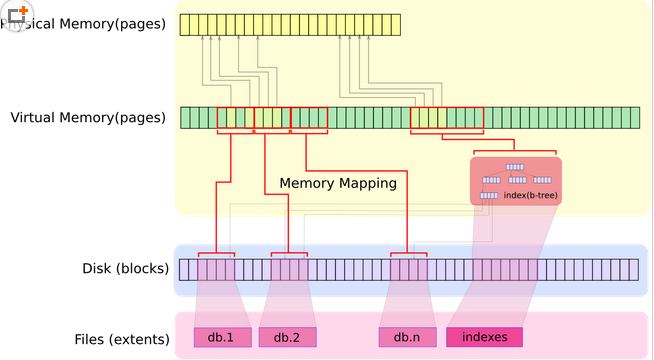
@内存映射文件是OS通过mmap在内存中创建一个数据文件，这样就把文件映射到一个虚拟内存的区域。

@虚拟内存对于进程来说，是一个物理内存的抽象，寻址空间大小为2^64

@操作系统通过mmap来把进程所需的所有数据映射到这个地址空间(红线)，然后再把当前需要处理的数据映射到物理内存(灰线)

@当进程访问某个数据时，如果数据不在虚拟内存里，触发page fault，然后OS从硬盘里把数据加载进虚拟内存和物理内存

@如果物理内存满了，触发swap-out操作，这时有些数据就需要写回磁盘，如果是纯粹的内存数据，写回swap分区，如果不是就写回磁盘。



**5 MongoDB的优势与劣势**

**（1）优势**

快速！基于内存，将热数据存放在物理内存中（不仅仅只是索引和少部分数据），从而提高了整体速度和效率。

高扩展性！MongoDB的高可用和集群架构拥有十分高的扩展性。

自身的FailOver机制！在副本集中，当主库遇到问题，无法继续提供服务的时候，副本集将选举一个新的主库继续提供服务。

JSon格式的数据！MongoDB的Bson和JSon格式的数据十分适合文档格式的存储与查询。

**（2）劣势**

应用经验少！由于NoSQL兴起时间短，应用经验相比关系型数据库较少。

由于以往用到的都是关系型数据库，可能会造成使用者一开始的不适应。

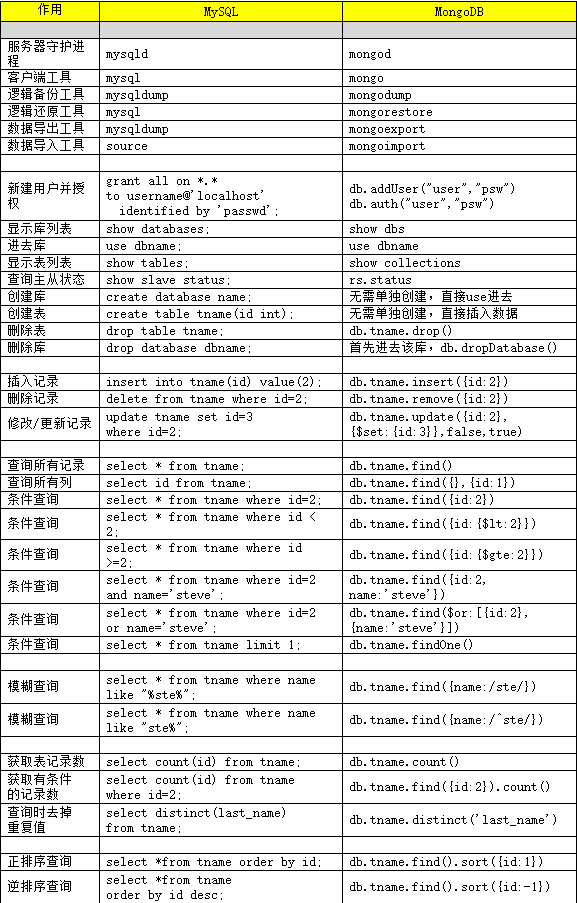
无事务机制！MongoDB本身没有自带事务机制，若需要在MongoDB中实现事务机制，需通过一个额外的表，从逻辑上自行实现事务。

**6 MongoDB与MYSQL对比**

（1）基础比较

| **数据库** | **MongoDB** | **MySQL** |
| --- | --- | --- |
| **数据库模型** | 非关系型 | 关系型 |
| **存储方式** | 以类JSON的文档的格式存储 | 不同引擎有不同的存储方式 |
| **查询语句** | MongoDB查询方式（类似JavaScript的函数） | SQL语句 |
| **数据处理方式** | 基于内存，将热数据存放在物理内存中，从而达到高速读写 | 不同引擎有自己的特点 |
| **成熟度** | 新兴数据库，成熟度较低 | 成熟度高 |
| **广泛度** | NoSQL数据库中，比较完善且开源，使用人数在不断增长 | 开源数据库，市场份额不断增长 |
| **事务性** | 仅支持单文档事务操作，弱一致性 | 支持事务操作 |
| **占用空间** | 占用空间大 | 占用空间小 |
| **join操作** | MongoDB没有join | MySQL支持join |

（2）SQL比较



## 二、分片技术

**1. 什么是分片**

（1）到目前为止，你都是把MongoDB当做一台服务器在用，每个mongod实例都包含应用程序数据的完整副本。就算使用了复制，每个副本也都是完整克隆了其他副本的数据。对于大多数应用程序而言，在一台服务器上保存完整数据集是完全可以接受的。但随着数据量的增长，以及应用程序对读写吞吐量的要求越来越高，普通服务器渐渐显得捉襟见肘了。尤其是这些服务器可能无法分配足够的内存，或者没有足够的CPU核数来有效处理工作负荷。除此之外，随着数据量的增长，要在一块磁盘或者一组RAID阵列上保存和管理备份如此大规模的数据集也变得不太现实。如果还想继续使用普通硬件或者虚拟硬件来托管数据库，那么这对这类问题的解决方案就是将数据库分布到多台服务器上，这种方法称之为分片。

（2）为数众多的Web应用程序，知名的如Flicker和LiveJournal，都实现了手动分片，将负载分布到多台MySQL数据库上。在这些实现中，分片逻辑都寄生于应用程序上。要明白这是如何实现的，想象一下，假如你有很多用户，需要将Users表分布到多台数据库服务器上。你可以指定一台数据库作为元数据库。这台数据库包含每个用户ID（或者用户ID范围）到指定分片映射关系的元数据。因此，要查询一个用户实际涉及两次查询：第一次查询访问元数据库以获得用户的分片位置，第二次查询直接访问包含用户数据的分片。对于这些Web应用程序而言，手动分片解决了负载问题，但气质并非无懈可击。最明显的问题就是迁移数据非常困难。如果单个分片负载过重，将其中的数据迁移到其他分片的过程完全是手动的。手动分片的第二个问题在于编写可靠的应用程序代码对于读写请求进行路由，并且将数据库作为一个整体进行管理，这也是非常困难的。最近也出现了管理手动分片的的框架，

最著名的就是Twitter的Gizzard。

（3）但正如那些手动分片数据库的人所说，要把事情做好并非易事。MongoDB中有一大块工作就是为了解决这个问题。因为分片是MongoDB的核心内容，所以用户无需担心在需求水平扩展时要自己设计外置分片框架。在处理困难的跨分片数据均衡问题时，这点尤为重要。这些代码并非那些大多数人在一个周末能够写出来的东西。也许最值得一提的是MongoDB在设计时为应用程序提供了统一接口，无论是在分片前，还是在分片后。也就是说，在数据库需要转换为分片架构时，应用程序几乎无需改变。

**2. 何时分片**

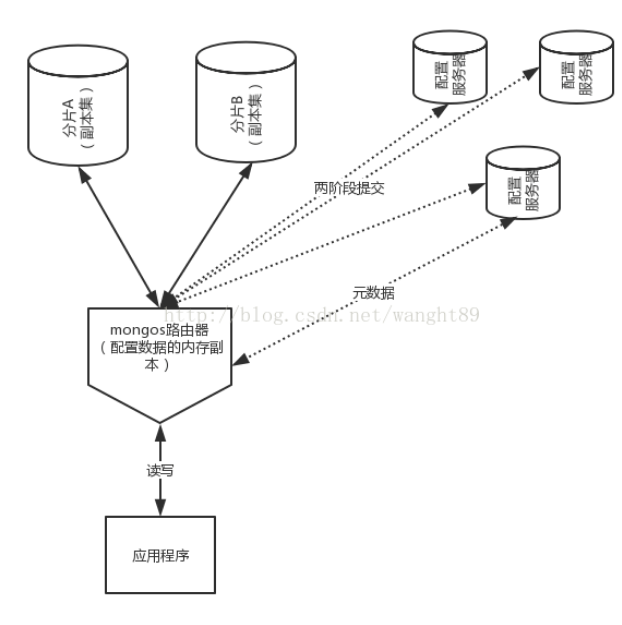
（1）这个问题的答案比你想的简单得多。我们之前已经说过把索引和工作数据集放在内存里时很重要的，这也是分片的主要原因。如果应用程序的数据集持续无限增长，那么迟早一天，内存会容纳不下这些数据。如果你正在使用亚马逊的EC2，那么这个阈值是68GB。或者你可以运行自己的硬件，并使用远高于68GB的内存，这样便能延后一段时间再做分片。但没有哪台机器内存时无限的，因此你早晚都会用到分片。

（2）不可否认，还有一些其他的应对措施。举例来说，如果你有自己的硬件，而且可以将所有的数据都保存在固态硬盘上，那么可以增加数据内存比，而不会为性能带来负面影响。还有一种情况，工作集是总数据量中的一部分，这是可以使用相对较小的内存。另一方面，如果有特殊的写负载要求，那么可以在数据达到内存大小之前先进行适当的分片，原因是需要将负载分到多台机器上，以便能够获得想要的吞吐量。五论哪种情况，对现有系统进行分片的决定都要基于以下几点--磁盘活动、系统负载以及最重要的工作集大小与可用内存的比例。

**3. 分片的工作原理**

**（1）分片组件**

**分片集群由分片、mongos路由器和配置服务器组成。**



**@分片**

MongoDB分片集群将数据分布在一个或多个分片上。每个分片部署成一个MongoDB副本集，该副本集保存了集群整体数据的一部分。因为每个分片都是一个副本集，所以他们拥有自己的复制机制，能够自动进行故障转移。你可以直接连接单个分片，就像连接单独的副本集一样。但是，如果连接的副本集是分片集群的一部分，那么只能看到部分数据。

**@mongos路由器**

如果每个分片都包含部分集群数据，那么还需要一个接口连接整个集群。这就是mongos。mongos进程是一个路由器，将所有的读写请求指引到合适的分片上。如此一来，mongos为客户端提供了一个合理的系统视图。mongos进程是轻量级且非持久化的。它们通常运行与与应用服务器相同的机器上，确保对任意分片的请求只经过一次网络跳转。换言之，应用程序连接本地的mongos，而mongos管理了指向单独分片的连接。

**@配置服务器**

如果mongs进程是非持久化的，那么必须有地方能持久保存集群的公认状态；这就是配置服务器的工作，其中持久化了分片集群的元数据，改数据包括：每个数据库，集合和特定范围数据的位置；一份变更记录，保存了数据在分片之间进行迁移的历史信息。配置服务器中保存的元数据是某些特定功能和集群维护是的重中之重。举例来说，每次有mongos进程启动，它都会从配置服务器中获取一份元数据的副本。没有这些数据，就无法获得一致的分片集群视图。该数据的重要性对配置服务器的设计和部署也有影响。如上面结构图中所示，有三个配置服务器，但它们并不是以副本集的形式部署的。它们比异步复制要求更严格；mongos进程向配置服务器写入时，会使用两阶段提交。这能保证配置服务器之间的一致性。在各种生产环境的分片部署中，必须运行三个配置服务器，这些服务器都必须部署在独立的机器上以实现冗余。

**4 分片核心**

**（1）分布式存储系统**

所谓分布式系统，就是利用多个独立的计算机来解决单个节点（计算机）无法处理的存储、计算问题，这是非常典型的分而治之的思想。每个节点只负责原问题（即整个系统需要完成的任务）的一个子集，那么原问题如何拆分到多个节点？在分布式存储系统中，任务的拆分即数据分片。需要解决的两个最主要的问题，即数据分片和数据冗余。

**（2）三个问题**

@如何做数据分片，即如何将数据映射到节点

@数据分片的特征值，即按照数据中的哪一个属性（字段）来分片

@数据分片的元数据的管理，如何保证元数据服务器的高性能、高可用，如果是一组服务器，如何保证强一致性

**（3）如何做数据分片，即如何将数据映射到节点**

三种分片方式：hash方式，一致性hash（consistent hash），按照数据范围（range based）。

为了后面分析不同的数据分片方式，假设有三个物理节点，编号为N0， N1， N2；有以下几条记录：

　　R0: {id: 95, name: 'aa', tag:'older'}

　　R1: {id: 302, name: 'bb',}

　　R2: {id: 759, name: 'aa',}

　　R3: {id: 607, name: 'dd', age: 18}

　　R4: {id: 904, name: 'ff',}

　　R5: {id: 246, name: 'gg',}

　　R6: {id: 148, name: 'ff',}

　　R7: {id: 533, name: 'kk',}

**（4）hash方式**

在哈希表中，最为简单的散列函数是 mod N（N为表的大小）。即首先将关键值计算出hash值（这里是一个整型），通过对N取余，余数即在表中的位置。

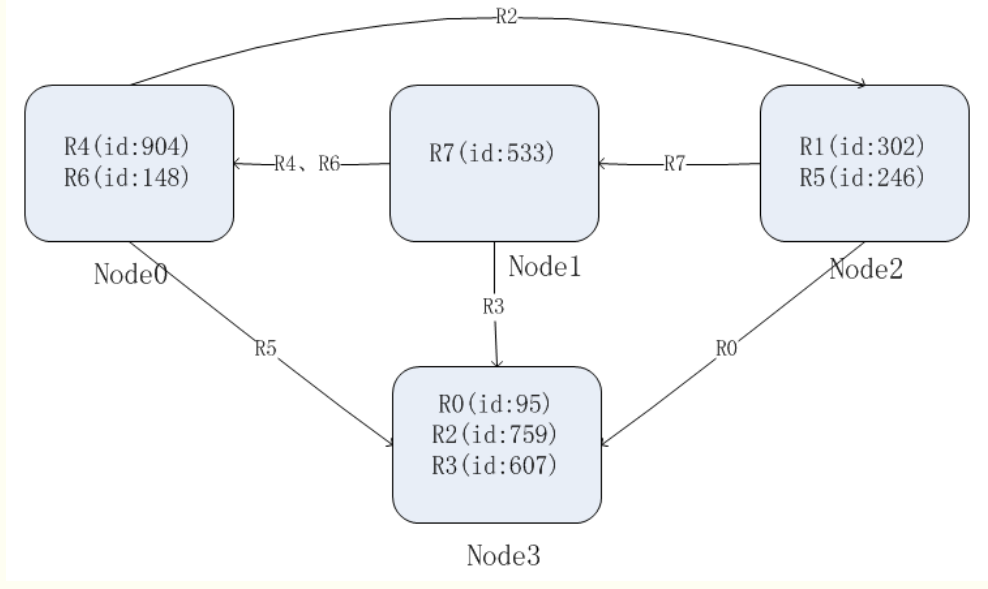
数据分片的hash方式也是这个思想，即按照数据的某一特征（key）来计算哈希值，并将哈希值与系统中的节点建立映射关系,从而将哈希值不同的数据分布到不同的节点上。

@优点：按照hash方式做数据分片，映射关系非常简单；需要管理的元数据也非常之少，只需要记录节点的数目以及hash方式就行了。

@缺点：但hash方式的缺点也非常明显：当加入或者删除一个节点的时候，大量的数据需要移动。比如在这里增加一个节点N3，因此hash方式变为了mod 4

在这种方式下，是不满足单调性（Monotonicity）的：如果已经有一些内容通过哈希分派到了相应的缓冲中，又有新的缓冲加入到系统中。哈希的结果应能够保证原有已分配的内容可以被映射到原有的或者新的缓冲中去，而不会被映射到旧的缓冲集合中的其他缓冲区。

在工程中，为了减少迁移的数据量，节点的数目可以成倍增长，这样概率上来讲至多有50%的数据迁移。



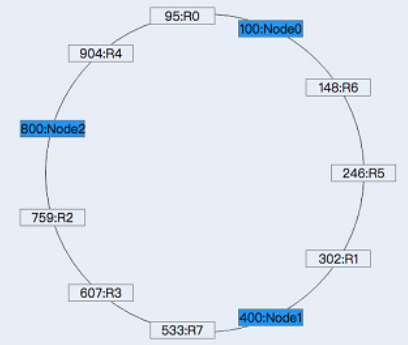
hash方式还有一个缺点，即很难解决数据不均衡的问题。有两种情况：原始数据的特征值分布不均匀，导致大量的数据集中到一个物理节点上；第二，对于可修改的记录数据，单条记录的数据变大。在这两种情况下，都会导致节点之间的负载不均衡，而且在hash方式下很难解决。

**（5）一致性hash**

一致性hash是将数据按照特征值映射到一个首尾相接的hash环上，同时也将节点（按照IP地址或者机器名hash）映射到这个环上。对于数据，从数据在环上的位置开始，顺时针找到的第一个节点即为数据的存储节点。这里仍然以上述的数据为例，假设id的范围为［0， 1000］，N0， N1， N2在环上的位置分别是100， 400， 800

一致性hash在增加或者删除节点的时候，受到影响的数据是比较有限的，比如这里增加一个节点N3，其在环上的位置为600，因此，原来N2负责的范围段（400， 800］现在由N3（400， 600] N2(600, 800]负责，因此只需要将记录R7(id:533) 从N2,迁移到N3:

不难发现一致性hash方式在增删的时候只会影响到hash环上响应的节点，不会发生大规模的数据迁移。



引入虚拟节点后的一致性hash需要维护的元数据也会增加：

第一，虚拟节点在hash环上的问题，且虚拟节点的数目又比较多；

第二，虚拟节点与物理节点的映射关系。但带来的好处是明显的，当一个物理节点失效是，hash环上多个虚拟节点失效，对应的压力也就会发散到多个其余的虚拟节点，事实上也就是多个其余的物理节点。在增加物理节点的时候同样如此。

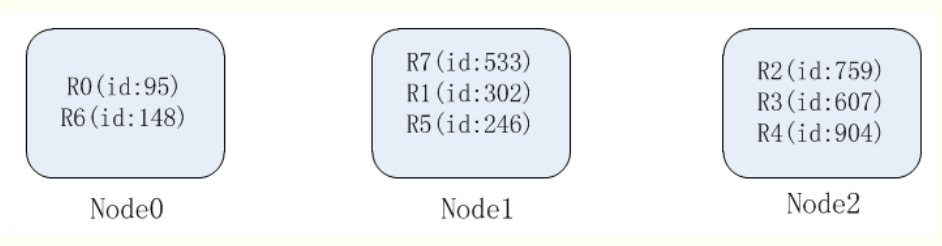
工程中，Dynamo、Cassandra都使用了一致性hash算法，且在比较高的版本中都使用了虚拟节点的概念。在这些系统中，需要考虑综合考虑数据分布方式和数据副本，当引入数据副本之后，一致性hash方式也需要做相应的调整， 可以参加cassandra的相关文档。

**（6）range based**

简单来说，就是按照关键值划分成不同的区间，每个物理节点负责一个或者多个区间。其实这种方式跟一致性hash有点像，可以理解为物理节点在hash环上的位置是动态变化的。

还是以上面的数据举例，三个节点的数据区间分别是N0(0, 200]， N1(200, 500]， N2(500, 1000]。

注意，区间的大小不是固定的，每个数据区间的数据量与区间的大小也是没有关系的。比如说，一部分数据非常集中，那么区间大小应该是比较小的，即以数据量的大小为片段标准。在实际工程中，一个节点往往负责多个区间，每个区间成为一个块（chunk、block），每个块有一个阈值，当达到这个阈值之后就会分裂成两个块。这样做的目的在于当有节点加入的时候，可以快速达到均衡的目的。

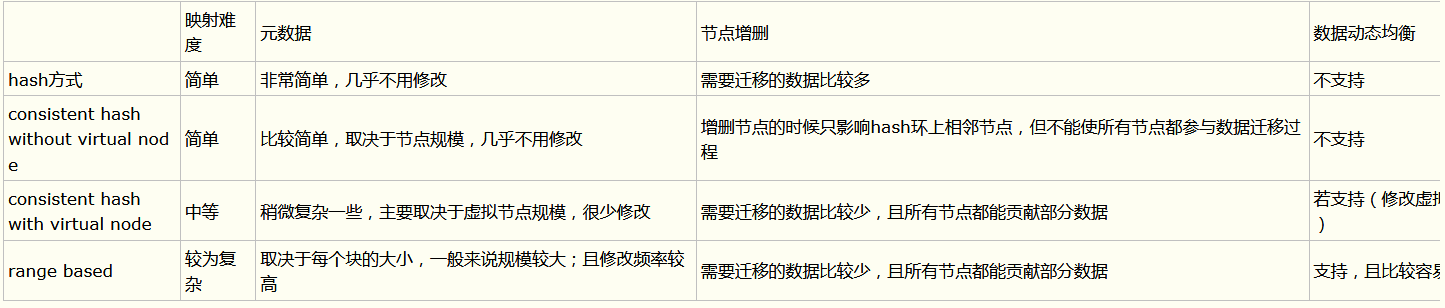


不知道读者有没有发现，如果一个节点负责的数据只有一个区间，range based与没有虚拟节点概念的一致性hash很类似；如果一个节点负责多个区间，range based与有虚拟节点概念的一致性hash很类似。

range based的元数据管理相对复杂一些，需要记录每个节点的数据区间范围，特别单个节点对于多个区间的情况。而且，在数据可修改的情况下，如果块进行分裂，那么元数据中的区间信息也需要同步修改。

range based这种数据分片方式应用非常广泛，比如MongoDB, PostgreSQL， HDFS

**（7）三种比较**



**（8）分片特征值的选择**

上面的三种方式都提到了对数据的分片是基于关键值、特征值的。这个特征值在不同的系统中有不同的叫法，比如MongoDB中的sharding key， Oracle中的Partition Key，不管怎么样，这个特征值的选择都是非常非常重要的。

基于最常用的访问模式。访问时包括对数据的增删改查的。比如上面的列子，我们选择“id”作为分片的依据，那么就是默认对的数据增删改查都是通过“id”字段来进行的。

另外一个问题，如果以单个字段为特征值（如id），那么不管按照什么分布方式，在多条数据拥有相同的特征值（如id）的情况下，这些数据一定都会分布到同一个节点上。在这种情况下有两个问题，一是不能达到节点间数据的均衡，二是如果数据超过了单个节点的存储能力怎么办？关键在于，即使按照分布式系统解决问题的常规办法 -- 增加节点 --也是于事无补的。

在这个时候，单个字段做特征值就不行了，可能得再增加一个字段作为“联合特征值”，类似数据库中的联合索引。比如，数据是用户的操作日志，可以使用id和时间戳一起作为hash函数的输入，然后算出特征值；但在这种情况下，如果还想以id为查询关键字来查询，那就得遍历所有节点了。

**（9）元数据服务器**

在上面讨论的三种数据分片分式中，或多或少都会记录一些元数据：数据与节点的映射关系、节点状态等等。

我们称记录元数据的服务器为元数据服务器（metaserver），不同的系统叫法不一样，比如master、configserver、namenode等。

元数据服务器就像人类的大脑，一只手不能用了还没忍受，大脑不工作整个人就瘫痪了。因此，元数据服务器的高性能、高可用，要达到这两个目标，元数据服务器就得高可扩展 -- 以此应对元数据的增长。

元数据的高可用要求元数据服务器不能成为故障单点（single point of failure），因此需要元数据服务器有多个备份，并且能够在故障的时候迅速切换。

有多个备份，那么问题就来了，怎么保证多个备份的数据一致性？

多个副本的一致性、可用性是CAP理论讨论的范畴，这里简单介绍两种方案。

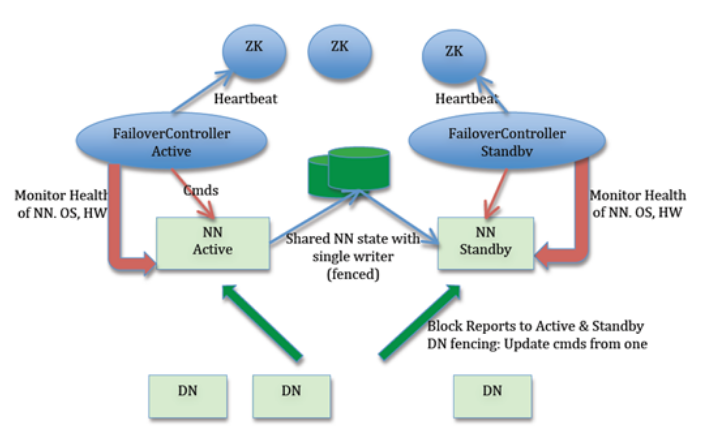
第一种是主从同步，首先选出主服务器，只有主服务器提供对外服务，主服务器将元数据的变革信息以日志的方式持久化到共享存储（例如nfs），然后从服务器从共享存储读取日志并应用，达到与主服务器一致的状态，如果主服务器被检测到故障（比如通过心跳），那么会重新选出新的主服务器。

第二种方式，通过分布式一致性协议来达到多个副本件的一致，比如大名鼎鼎的Paxos协议，以及工程中使用较多的Paxos的特化版本 -- Raft协议，协议可以实现所有备份均可以提供对外服务，并且保证强一致性。

**（10）HDFS元数据**

上图中NN即NameNode， DN即DataNode（即实际存储数据的节点）。从图中可以看到， 两台 NameNode 形成互备，一台处于 Active 状态，为主 NameNode，另外一台处于 Standby 状态，为备 NameNode，只有主 NameNode 才能对外提供读写服务。

Active NN与standby NN之间的数据同步通过共享存储实现，共享存储系统保证了Namenode的高可用。为了保证元数据的强一致性，在进行准备切换的时候，新的Active NN必须要在确认元数据完全同步之后才能继续对外提供服务。



另外，Namenode的状态监控以及准备切换都是Zookeeper集群负责，在网络分割（network partition）的情况下，有可能zookeeper认为原来的Active NN挂掉了，选举出新的ActiveNN，但实际上原来的Active NN还在继续提供服务。这就导致了“双主“或者脑裂（brain-split）现象。为了解决这个问题，提出了fencing机制，也就是想办法把旧的 Active NameNode 隔离起来，使它不能正常对外提供服务。

**（11）元数据的缓存**

即使元数据服务器可以由一组物理机器组成，也保证了副本集之间的一致性问题。但是如果每次对数据的请求都经过元数据服务器的话，元数据服务器的压力也是非常大的。很多应用场景，元数据的变化并不是很频繁，因此可以在访问节点上做缓存，这样应用可以直接利用缓存数据进行数据读写，减轻元数据服务器压力。

怎么达到缓存的强一致性呢？比较容易想到的办法是当metadata变化的时候立即通知所有的缓存服务器（mongos），但问题是通信有延时，不可靠。

另外一种解决办法，就是大名鼎鼎的lease机制

**（12）lease机制**

@服务器向所有客户端发送缓存数据的同时，颁发一个lease，lease包含一个有限期（即过期时间）

@lease的含义是：在这个有效期内，服务器保证元数据不会发生变化

@因此客户端在这个有效期内可以放心大胆的使用缓存的元数据，如果超过了有效期，就不能使用数据了，就得去服务器请求。

@如果外部请求修改服务器上的元数据（元数据的修改一定在服务器上进行），那么服务器会阻塞修改请求，直到所有已颁发的lease过期，然后修改元数据，并将新的元数据和新的lease发送到客户端

@如果元数据没有发生变化，那么服务器也需要在之前已颁发的lease到期之间，重新给客户端颁发新的lease（只有lease，没有数据）

## 三、MongoDB基础知识与CRUD

| **mongodb概念** | **类比sql概念** |
| --- | --- |
| 文档document，拥有唯一的键值”\_id” | 复杂的行 |
| 集合collection | 没有模式的表 |
| 单个实例 - 容纳多个独立的数据库 |  |

**1 文档document**

文档：多个 键key 及其关联的 值value 有序地 放置在一起。

在各种编程语言中表现不同：python字典，ruby散列，Go等的映射。

{“key” : value}

文档中的 键值(k-v) 是有顺序的。

{"k1":v1,"k2":v2}不同于 {"k2":v2,"k1":v1}

值v 可以是多种数据类型：字符串，数值 或者另一个 文档 等。

键k 是字符串。utf-8字符。

键值(k-v) 是区分 键大小写 和 值类型 的。

键k 不能重复。

**2 键的命名**

满足下述条件的任意 utf-8 字符串。

不能含有结尾符 \0 。

不能含有保留字符 . 和 $ 。

不能以 \_ 开头。(键是被保留的)

**3 集合collection**

集合：文档的集合。类似没有模式的表。

**4 无模式**

值可不同，键也可不同。

{“greeting” : "hello",

"foo" : 3}

为什么使用多个集合？

查询简易度 和 查询速度 会提升。

把同类型的文档放在一起，这样数据会更加集中。

创建索引时，文档会有附加的结构。索引更有效。

**5 命名**

满足下述条件的任意 utf-8 字符串。

不能是空字符串 "" 。

不能含有结尾符 \0 。

不能含有保留字符 $ 。

不能以 system. 开头。

**6 子集合**

使用 "." 分开的按命名空间划分的子集合,更好的组织数据。如，

blog.posts

blog.authors

**7 数据库 database**

数据库：集合的集合。

**8 命名**

满足下述条件的任意 utf-8 字符串。

不能是空字符串 "" 。

不能含有结尾符 \0 。

不能含有 单引号''，空 格， 点.，钱$，正反斜杠 / \。

全部 小写 。

最多 64 字节。

不能是保留名：admin，local，config

因为数据库名最终会变成文件系统里的文件，所以限制较多。

**9命名空间**

命名空间：database\_name.collection\_name 组成命名空间。

命名空间长度 <= 121 字节。实际中应 <= 100 字节。

**10 mongodb shell**

mongo.exe ，一个自带的 javascript shell 。从命令行和mongodb交互。

在运行shell mongo.exe 前，请确保启动数据库服务器 mongod.exe 。

**11 shell操作**

选择数据库

use foobar // db = foobar

查看数据库

db

（1）增 / 创建文档

db.collection\_name.insert(new\_doucment)

（2）增 / 增加键值

db.collection\_name.update(限定条件\_如\_{"title":"foo"}, new\_key\_value)

（3）删 / 删除

db.collection\_name.remove(限定条件\_如\_{"title":"foo"})

（4）查 / 读取

db.collection\_name.find() // 返回集合里所有的文档

db.collection\_name.findOne() // 返回一个文档

（5）帮助 help

shell中键入 help 获得帮助

help

db.help() // 数据库相关的帮助

db.collection\_name.help() // 集合相关的帮助

function\_name // 函数不加括号，显示该函数的js代码

（6）db.集合名

db.name

当js只有在db中找到同名属性时，会将其作为方法而非集合名。如，db.version

当集合名包含运算符时，如 db.foo-bar 。会做减法运算而报错。

解决方法，使用getCollection方法：

db.getCollection("collection\_name")

db.getCollection("version")

db.getCollection("foo-bar")

**14 变量访问子集合**

在js中 x.y = x['y'] ，也就是说，

db.collection\_name.sub\_name = db.collection\_name[sub\_name]

可以如下优雅的书写，

var sub\_collections = ["foo", "bar", "zzz"];

for (i in collections) {

doStuff(db.collection\_name[sub\_collections]);

}

而不用如此笨拙，

doStuff(db.collection\_name.foo);

doStuff(db.collection\_name.bar);

doStuff(db.collection\_name.zzz);

**15 数据类型**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **示例** |
| **bool** | true or false | {“x” : true} |
| **null** | 空值 | {“x” : null} |
| **i32** | shell不支持 | 会被转换为64bit float |
| **i64** | shell不支持 | 会被转换为64bit float |
| **f64** | shell中数字都是此类型 | {“x” : 3.14} |
|  | 上述的整数也是此类型 | {“x” : 3} |
| **string** | 字符串，utf-8 | {“x” : “foobar”} |
| **ObjectId** | 文档的 12字节 唯一ID | {“x” : ObjectId()} |
| **date** | 日期，从标准纪年的毫秒数 | {“x” : new Date()} |
| **re** | 正则表达式，采用js语法 | {“x” : /foobar/i} |
| **code** | js代码 | {“x” : function() {/\* … \*/}} |
| **符号** | shell不支持 | shell将db中的符号转化为字符串 |
| **二进制数据** | shell不支持 |  |
| **最大值** | shell不支持 |  |
| **最小值** | shell不支持 |  |
| **undefined** | js中null和undefined是不用类型 | {“x” : underfined} |
| **array[]** | 值和列表的集合 | {“x” : [“a”, “b”, “c”]} |
| **内嵌文档{}** | 文档包含文档 | {“x” : {“y”: “bar”}} |

**16 基本概念**

（1）文档(document)是MongoDB中数据的基本单元，非常类似于关系型数据库系统中的行(但是比行要复杂的多)。

（2）集合(collection)就是一组文档，如果说MongoDB中的文档类似于关系型数据库中的行，那么集合就如同表。

（3）MongoDB的单个计算机可以容纳多个独立的数据库，每一个数据库都有自己的集合和权限。

（4）MongoDB自带简洁但功能强大的JavaScript shell，这个工具对于管理MongoDB实例和操作数据作用非常大。

（5）每一个文档都有一个特殊的键”\_id”,它在文档所处的集合中是唯一的，相当于关系数据库中的表的主键。

## 四、Java操作MongoDB

**1 连接数据库**

连接数据库，你需要指定数据库名称，如果指定的数据库不存在，mongo会自动创建数据库。

所需jar包： mongo-java-driver-3.2.2.jar

连接数据库的Java代码如下：

1import com.mongodb.MongoClient;

2 import com.mongodb.client.MongoDatabase;

3 public class MongoDBJDBC{

4 public static void main( String args[] ){

5 try{

6 // 连接到 mongodb 服务

7 MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

8 // 连接到数据库

9 MongoDatabase mongoDatabase = mongoClient.getDatabase("test");

10 System.out.println("Connect to database successfully");

11 }catch(Exception e){

12 System.err.println( e.getClass().getName() + ": " + e.getMessage() );

13 }

14 }

15 }

**2 创建集合：**

我们可以使用 com.mongodb.client.MongoDatabase 类中的createCollection()来创建集合

import com.mongodb.MongoClient;

import com.mongodb.client.MongoDatabase;

public class MongoDBJDBC{

public static void main( String args[] ){

try{

// 连接到 mongodb 服务

MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

// 连接到数据库

MongoDatabase mongoDatabase = mongoClient.getDatabase("test");

System.out.println("Connect to database successfully");

mongoDatabase.createCollection("test");

System.out.println("集合创建成功");

}catch(Exception e){

System.err.println( e.getClass().getName() + ": " + e.getMessage() );

}

}

}

**3 插入文档**

//插入文档

/\*\*

\* 1. 创建文档 org.bson.Document 参数为key-value的格式

\* 2. 创建文档集合List<Document>

\* 3. 将文档集合插入数据库集合中 mongoCollection.insertMany(List<Document>) 插入单个文档可以用 mongoCollection.insertOne(Document)

\* \*/

Document document = new Document("title", "MongoDB").

append("description", "database").

append("likes", 100).

append("by", "Fly");

List<Document> documents = new ArrayList<Document>();

documents.add(document);

collection.insertMany(documents);

System.out.println("文档插入成功");

**4 查询文档**

//检索所有文档

/\*\*

\* 1. 获取迭代器FindIterable<Document>

\* 2. 获取游标MongoCursor<Document>

\* 3. 通过游标遍历检索出的文档集合

\* \*/

FindIterable<Document> findIterable = collection.find();

MongoCursor<Document> mongoCursor = findIterable.iterator();

while(mongoCursor.hasNext()){

System.out.println(mongoCursor.next());

}

**5 修改文档**

//更新文档 将文档中likes=100的文档修改为likes=200

collection.updateMany(Filters.eq("likes", 100), new Document("$set",new Document("likes",200)));

//检索查看结果

FindIterable<Document> findIterable = collection.find();

MongoCursor<Document> mongoCursor = findIterable.iterator();

while(mongoCursor.hasNext()){

System.out.println(mongoCursor.next());

}

**6 删除文档**

//删除符合条件的第一个文档

collection.deleteOne(Filters.eq("likes", 200));

//删除所有符合条件的文档

collection.deleteMany (Filters.eq("likes", 200));

//检索查看结果

FindIterable<Document> findIterable = collection.find();

MongoCursor<Document> mongoCursor = findIterable.iterator();

while(mongoCursor.hasNext()){

System.out.println(mongoCursor.next());

}

## 五、MongoDB体系结构

**1 MongoDB是一个可移植的数据库**，它在流行的每一个平台上都可以使用，即所谓的跨平台性，在不同的操作系统上虽然略有差别，但是从整体架构上来看，MongoDB在不同的平台上是一样的，如数据逻辑结构和数据存储等等。

**2 一个运行着的MongoDB数据库就可以看成是一个MongoDB Server**，该Server由实例和数据库组成，在一般情况下，一个MongoDB Server机器上包含一个实例或者多个与之对应的数据库，但是在特殊情况下，如硬件投入成本或者特殊的应用需求，也允许一个Server机器上可以有多个实例或者多个数据库。MongoDB中一系列物理文件(数据文件、日志文件等)的集合与之对应的逻辑结构(集合、文档等)被称之为数据库，简单的说，就是数据库是由一系列与磁盘有关系的物理文件的组成。

**3 数据逻辑结构**

很多人在学习MongoDB体系结构的时候会遇到各种各样的问题，我在这里给大家简单的介绍下MongoDB体系结构之一的逻辑结构，MongoDB的逻辑结构是一种层次结构，主要由：文档(Document)、集合(Collection)、数据库(database)这三部分组成，逻辑结构是面向用户的，用户使用MongoDB开发应用程序使用的就是逻辑结构。

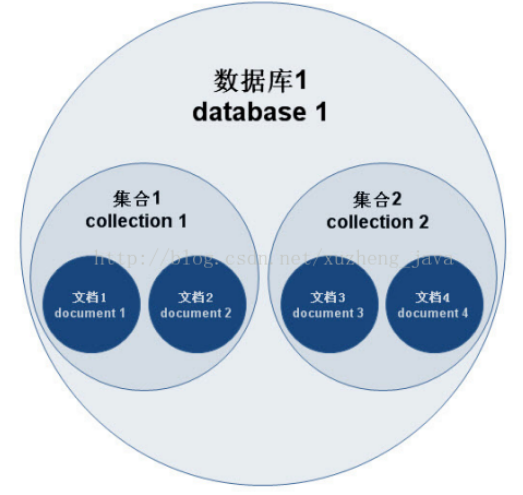
**MongoDB的文档(Document)，相当于关系数据库中的一行记录。**

**多个文档组成一个集合(Collection)，相当于关系数据库的表。**

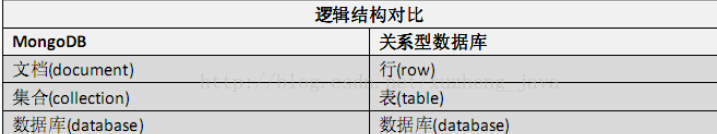
**多个集合(Collection)，扩及上组织在一起，就是数据库(database)。**

**一个MongoDB实例支持多个数据库(database)。**

文档(Document)、集合(Collection)、数据库(Database)的层次结构如下图：



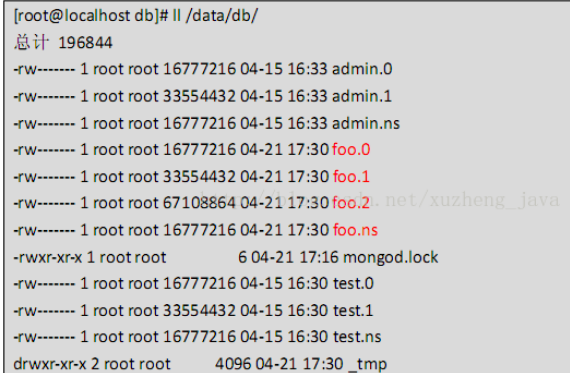
对于习惯了关系型数据库的朋友们，我将MongoDB与关系型数据库的逻辑结构进行了对比，以便让大家更深刻的理解MongoDB的逻辑结构：



**4 数据存储结构**

MongoDB对国内用户来说比较新，它就像是一个黑盒子，但是如果对于它内部的数据存储了解多一些的话，那么就会很快的理解和驾驭MongoDB，让它发挥更大的作用。

MongoDB的默认数据目录是/daba/db,它负责存储所有的MongoDB的数据文件，在MongoDB内部，每个数据库都包含一个.ns文件和一些数据文件，而且这些文件会随着数据量的增加变的越来越多，所以如果系统中有一个叫做foo的数据库，那么构成foo这个数据库的文件就会由foo.ns、foo.0、foo.1、foo.2等组成，具体如下：

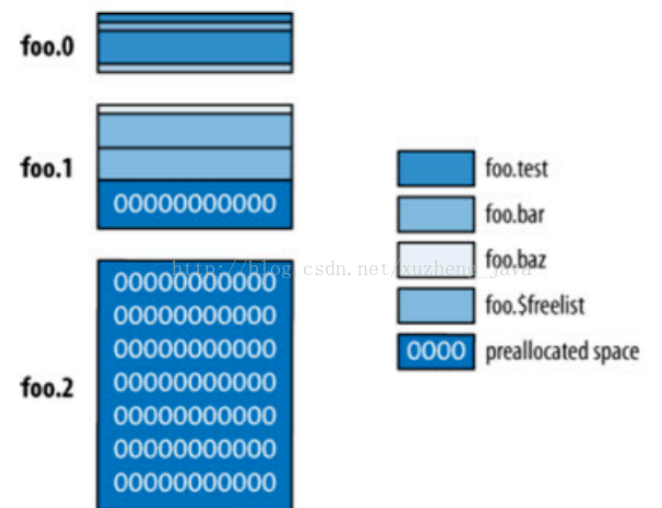


MongoDB内部有预分配空间的机制，每个预分配的文件都用0进行填充，由于有了这个机制，MongoDB始终保持额外的空间和空余的数据文件，从而有效避免了由于数据暴增而带来的磁盘压力过大的问题。

由于表中数据量的增加，数据文件每新分配一次，它的大小都会是上一个数据文件大小的2倍，每个数据文件最大2G，这样的机制有利于防止较小的数据库浪费过多的磁盘空间，同时又能保证较大的数据库有相应的预留空间使用。

数据库的每张表都对应一个命名空间，每个索引也有对应的命名空间，这些命名空间的元数据集中在\*.ns文件中。

在下图中。foo这个数据库包含3个文件用于存储表和索引数据，foo.2文件属于预分配的空文件，foo.0和foo.1这两个数据文件被分为了相应的盘区对应不同的名字空间。



上图显示了命名空间和盘区的关系，每个命名空间可以包含多个不同的盘区，这些盘区并不是连续的，与数据文件的增长相同，每个命名空间对应的盘区大小的而是随着分配的次数不断增长的，这样做的目的是为了平衡命名空间浪费的空间与保持某一个命名空间中数据的连续性，上图中还有一个需要注意的命名空间：$freelist，这个命名空间用户记录不再使用的盘区(被删除的Collection或者索引)，每当命令空间需要分配新的盘区的时候，都会先查看$freelist是否有大小合适的盘区可以使用，这样就回收空闲的磁盘空间。

## MongoDB、Cassandra 和 HBase比较

**1 MongoDB：源于开发人员，为开发人员服务**

在众多NoSQL的方案中，MongoDB的Stirman指出，MongoDB的瞄准了适合各种应用的平衡的方法。它的功能接近于传统的关系型数据库，MongoDB的用户不仅可以利用其横向扩展机器的云基础架构的优势，并且，因为它能够轻松定义各种灵活的数据模型，所以可以支持不同类型的数据集存储。MongoDB通常是开发人员第一个尝试的NoSQL数据库，因为它是很容易学习。Will Shulman，MongoLab（一个MongoDB服务提供商）的CEO，是这样说的：MongoDB中的成功在很大程度上是因为它数据结构存储的创新，让我们更容易和更具表现力地定义我们应用程序中的数据模型。在通常开发和应用场景中，和原有数据库具有相同的基本数据模型是有极大好处的，因为它简化了应用程序开发的任务，另一方面，消除了复杂的数据格式代码转换层。当然，像任何其他技术一样，MongoDB中都有其长处和短处。 MongoDB是专门为OLTP（On-Line Transaction Processing，联机事务处理系统）模式。如果您需要复杂的事务处理，它不是一个好的选择。然而，MongoDB的简单性使其成为一个优秀的存储。（注：MongoDB以文档的形式存储数据，不支持事务和表连接。因此查询的编写、理解和优化都容易得多。）

**2 Cassandra：规模化安全运行**

三种数据库中，至少两种数据库具有简单特性：开发简单，操作简便。而MongoDB赢得人心的原因是简单的开发应用，Cassandra赢得人心是因为易于管理的规模。DataStax的McFadin告诉我，用户往往倾向于使用Cassandra ，是因为特别在大规模集群下，增强一个关系型数据的性能、可靠性是非常困难的。一位前甲骨文DBA，McFadin是兴高采烈地发现，“复制和可扩放性是基础”，Cassandra 特点是从一开始设计就解决这个问题。在RDBMS中的世界，数据库功能，拓展和复制对很多开发者用户来说，是一个难题。这个问题在过往的企业规模小的时候，不是一个大问题。而在今天，它很迅速地成为大问题。

我从McFadin和其他人那里获知，Cassandra在机器拓展部署上，表现特别出色。Cassandra自带的备份机制，保证各个数据中心的数据安全。至于增加容量到集群，“你只需启动一台新机器，并告诉Cassandra那里的新节点，”McFadin说，“然后，它完成其他剩下的事情。”

优秀的可拓展性，加上出色的写入和可观的查询性能，加起来成为Cassandra高性能的核心。NoSQL的一篇文章认为Cassandra在集群规模管理方面非常出色，但它需要一个博士学位才能上手。

事实并非如此，McFadin坚持认为：

在复制、读取和写入是故意简单。你可以在几个小时内学会Cassandra的核心功能。在部署这项新技术的时候，为给开发者带来很多的信心，因为比较少引入“黑盒子”内的技术细节和复杂的故障模式原理。这意味着主要的开发成本，是对Cassandra数据模型的理解，以及如何结合您的应用程序。鉴于Cassandra的CQL查询语言（类似于SQL，实际上不是SQL），McFadin说，学习这个也不困难。更重要的是，他告诉我，“Cassandra回报给你的是，在一个数据库中：没有戏剧性的场（故障）出现。这就是用户喜欢使用Cassandra的原因。”

**3 HBase：Hadoop的知心伙伴**

HBase，像Cassandra一样是个通过key-value面向列存储的服务。因为它和Hadoop有着“共同血统”，被广泛使用。事实上，正如Cloudera的Kestelyn所说的那样，“HBase提供了一个基于记录的存储层，能够快速随机读取和写入数据，正好弥补了Hadoop的缺陷，Hadoop侧重系统吞吐量，而牺牲I / O读取效率为代价。”Kestelyn接着说：更改有效录入到内存中，以达到最大的访问量，同时将数据保存到HDFS。这种设计使基于Hadoop的EDH（enterprise data hub，企业数据中心）服务，能够实时完成随机读写存储数据，但仍拥有HDFS的高容错性和耐用性。Hadoop的亲和力，不是HBase数据库中的人气排名不断上升的唯一原因。类似Cassandra，HBase是Google的Bigtable的开源实现转化成的数据库，天然被设计为高可扩展性。Hbase可以利用任何数量服务器的磁盘、内存和CPU资源，同时拥有极佳的扩展功能，如自动分片。当系统负载和性能要求不断增加，HBase的可通过简单增加服务器节点的方式无限拓展。 HBase从底层设计上保证，在确保数据一致性的同时，提供最佳性能。但规模不是它的唯一用途。Kestelyn指出，“由于它与Hadoop的生态系统紧密集成，对于用户和应用程序来说，数据是容易获取的，可以通过SQL的方式查询（使用Cloudera的Impala，Phoenix，或Hive），甚至自由文本搜索（使用Cloudera Search）。“

**4 因此，HBase为开发人员提供了一种方法，利用现有通用的SQL语**言，来建立在一个更成熟的分布式数据库。每种数据库技术都有自己的长处和不足，但这里评论的三种数据库，在大数据技术领域，占据了重要的位置。虽然未来可能还有一种全新的NoSQL数据库技术会挑战它们前三的位置，但目前的现实是，许多开发人员以及一批强大的成熟企业已经做出了它们的选择：MongoDB、Cassandra 和 HBase。