# MongoDB中的基本概念及原理

MongoDB是一个基于分布式文件存储的数据库。由C++语言编写。旨在为WEB应用提供可扩展的高性能数据存储解决方案。MongoDB是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。在这里我们有必要先简单介绍一下非关系型数据库（NoSQL）

## 什么是NoSQL

NoSQL,指的是非关系型数据库。NoSQL有时也称作Not Only SQL 的缩写，是对不同于传统的关系型数据库的数据库管理系统的统称。NoSQL用于超大规模数据的存储。（例如谷歌或Facebook每天为他们的用户收集万亿比特的数据）。这些类型的数据存储不需要固定的模式，无需多余操作就可以横向扩展。

## 关系型数据库PK非关系型数据库

|  |  |
| --- | --- |
| **关系型数据库** | **NoSQL数据库** |
| 高度组织化结构化数据 | 代表着不仅仅是SQL |
| 结构化查询语言（SQL） | 没有声明性查询语言 |
| 数据和关系都存储在单独的表中 | 没有预定义的模式 |
| 数据操作语言，数据定义语言 | 键-值对存储，列存储，文档存储，图形数据库 |
| 严格的一致性 | 最终一致性，而非ACID属性 |
| 基础事务 | 非结构化和不可预知的数据 |
|  | CAP定理 |
|  | 高性能，高可用和可伸缩性 |

## NoSQL数据库分类

|  |  |  |
| --- | --- | --- |
| 类型 | **典型代表** | **特点** |
| 列存储 | Hbse  Cassandra  Hypertable | 顾名思义，是按照列存储数据的。最大的特点是方便存储结构化和半结构化的数据，方便做数据压缩，对针对某一列或某几列的查询有非常大的IO优势 |
| 文档存储 | MongoDB  CounchDB | 文档存储一般用类似json的格式存储，存储的内容是文档型的。这样也就有机会对某些字段建立索引，实现关系数据库的某些功能。 |
| Key-value存储 | Tokyo Cabinet/Tyrant  Memcache  Redis | 可以通过key快速查找其value。一般来说，存储不管value的格式，照单全收（Redis包含了其他功能） |
| 图存储 | Neo4J  FlockDB | 图形关系的最佳存储。使用传统关系数据库来解决的话新能低下，而且设计使用不方便 |
| 对象存储 | Db4o  Versant | 通过类似面向对象语言的语法操作数据库，通过对象的方式存储数据库 |
| XML数据库 | Berkeley DB XML  BaseX | 高效的存储XML数据，并存储XML内部查询语法，比如XQuery，Xpath |

## CAP原则

在计算机科学中，CAP定理（CAP theorem），又被称作布鲁尔定理（Brewer’s theorem），它指出对于一个分布式计算机系统来说，不可能同时满足以下三点：

一致性（Consistency）(所有节点在同一时间具有相同的数据)

可用性（Availability）(保证每个请求不管成功或者失败都有响应)

分区容错性（Partition tolerance）(系统中任意信息的丢失或失败不影响系统的继续运行)

CAP理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。因此，根据CAP原理将NoSQL数据库分成了满足CA原则、满足CP原则和满足AP原则三大类：

CA - 单点集群，满足一致性，可用性的系统，通常在扩展性上不太强大。

CP - 满足一致性，分区容错性的系统，通常性能不是特别高。

AP - 满足可用性，分区容错性的系统，通常可能对一致性要求低一些。

## MongoDB的数据结构与关系型数据库数据机构对比

|  |  |  |
| --- | --- | --- |
| 关系型数据库术语/概念 | **MongoDB属于/概念** | **解释/说明** |
| Databast | Database | 数据库 |
| Table | Collection | 数据库表/集合 |
| Row | DocuMent | 数据记录行/文档 |
| Column | Field | 数据列/数据字段 |
| Index | Index | 索引 |
| Table joins |  | 表关联/MongoDB不支持 |
| Primary Key | Object ID | 主键/MongoDB自动将\_id设置为主键 |

## MongoDB中的数据类型

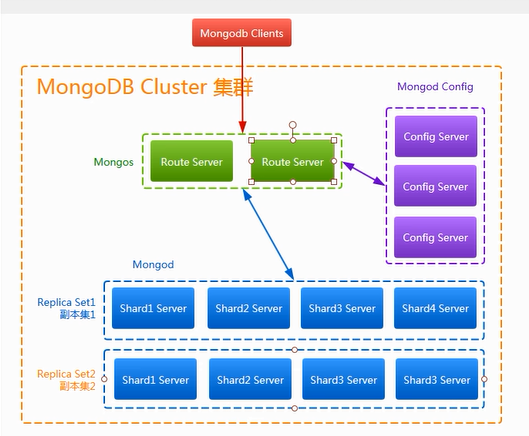
|  |  |  |  |
| --- | --- | --- | --- |
| **数据类型** | **说明** | **解释** | **举例** |
| Null | 空值 | 表示空值或未定义的对象 | {“x”:null} |
| Boolean | 布尔值 | 真或者假：true或者fasle | {“x”:true} |
| Integer | 整数 | 整型数值。用于存储数值。根据你所采用的服务器，可以分为32位或者64位。 |  |
| Double | 浮点数 | 双精度浮点值。 | {“x”:3.14,”y”:3} |
| String | 字符串 | UTF-8字符串 |  |
| Symbol | 符号 | 符号，该数据类型基本上等同于字符串类型，但不同的是，它一般用于采用特殊符号类型的语言。 |  |
| ObjectID | 对象ID | 对象ID，用于创建文档的ID | {“id”:ObjectId()} |
| Date | 日期 | 日期时间。用UNIX时间格式来存储当前日期或时间 | {“date”:new Date()} |
| Timestamp | 时间戳 | 从标准纪元开始的毫秒数 |  |
| Regular | 正则表达式 | 文档中可以包含正则表达式，遵循JavaScript的语法 | {“foo”:/foobar/i} |
| Code | 代码 | 可以包含JavaScript代码 | {“x”:function() {}} |
| Undefined | 未定义 | 已废弃 |  |
| Array | 数组 | 值的集合或者列表 | {“arr”:[“a”,”b”]} |
| BinaryData | 二进制 | 用于存储二进制数据 |  |
| Object | 内嵌文档 | 文档可以作为文档中某个key的value | {“x”:{“foo”:”bar”}} |
| Min/Max  Keys | 最大/小值 | 将一个值与BSON(二进制的JSON)元素的最低值和最高值相对比。 |  |

## 图解MongoDB底层原理

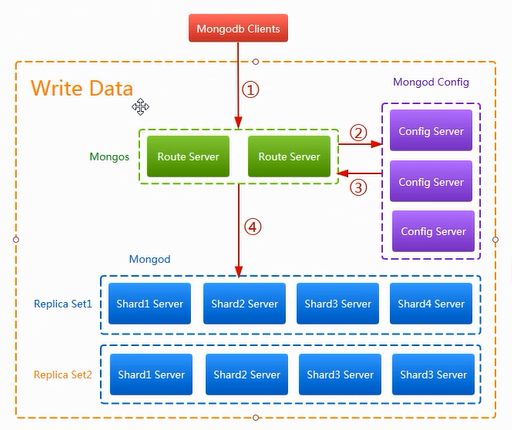
MongoDB的集群部署方案中有三种角色：实际数据存储结点、配置文件存储结点和路由接入结点。

连接的客户端直接与路由器的结点相连，从配置结点上查询数据，根据查询结果到实际的存储结点上查询和存储数据。MongoDB的部署方案有单机部署、复本集（主备）部署、分片部署、复本集与分片混合部署。

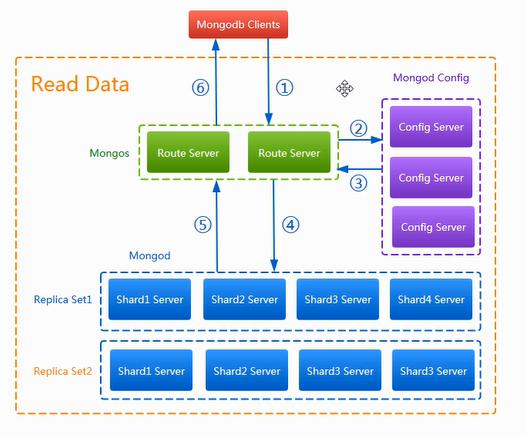
混合的部署方式如图：



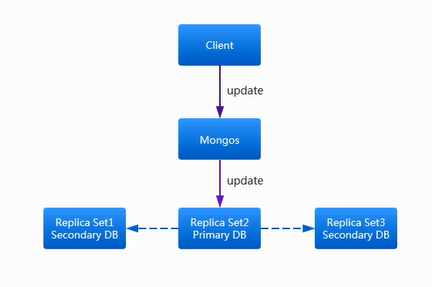
混合部署方式下向MongoDB写数据的流程如图:



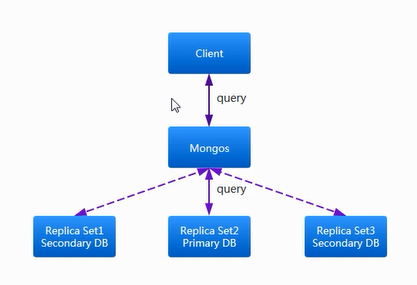
混合部署方式下读MongoDB里的数据流程如图：



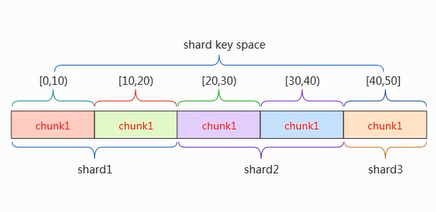
对于复本集又有主和从两种角色，写数据和读书节也是不同，写数据的过程只是写到主结点中，由主结点以异步的方式同步到从结点中：



而读数据只要从任一结点中读取，具体到哪个结点中读取是可以指定的：



对于MongoDB的分片，假设我们某一索引键（ID）为主键，ID的区间[0,50],划分成5个chunk，分别存储到3个片服务器中，如图所示：



假设数据量很大，需要增加片服务器时可以只移动chunk来均分数据即可。

配置结点：

存储配置文件的服务器其实存储的是片键与chunk以及chunk与server的映射关系，用上面的数据表示的配置结点存储的数据模型如下表：

Map1:

|  |  |
| --- | --- |
| Key range | **chunk** |
| [0,1] | chunk1 |
| [10,20] | chunk2 |
| [20,30] | chunk3 |
| [30,40] | chunk4 |
| [40,50] | chunk5 |

Map2:

|  |  |
| --- | --- |
| **chunk** | **shard** |
| chunk1 | Shard1 |
| chunk2 | Shard2 |
| chunk3 | Shard3 |
| chunk4 | Shard4 |
| chunk5 | Shard5 |

路由结点：

路由角色的结点在分片的情况下起到负载均衡的作用。

## MongoDB应用场景和不适用场景

1. 适用场景

对于MongoDB实际应用来讲，是否使用MongoDB需要根据项目的特定特点进行一一甄别，这就要去我们对MongoDB适用和不适用场景有一定了解。

根据MongoDB官网的说明，MongoDB适用场景如下：

1. 网站实用数据：MongoDB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。
2. 数据缓存：由于性价比很高，MongoDB也适合作为信息基础设施的缓存层。在系统重启之后，由MongoDB搭建的持久化缓存层可以避免下层的数据源过载。
3. 大尺寸、低价值数据存储：适用传统的关系型数据库存储一些数据时可能会比较昂贵，在此之前，很多时候程序员往往会选择传统的文件进行存储。
4. 高伸缩性场景：MongoDB非常适合由数十或数百台服务器组成的数据库。MongoDB的路线图中已经包含对MapReduce引擎的内置支持
5. 对象或JSON数据存储：MongoDB的BSON数据格式非常适合文档格式化的存储及查询。
6. 不适用场景

了解了MongoDB适用场景之后，还需要了解哪些场景不适合适用MongoDB，具体如下：

1. 高度事务性系统：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。
2. 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。
3. 需要复杂SQL查询的问题。

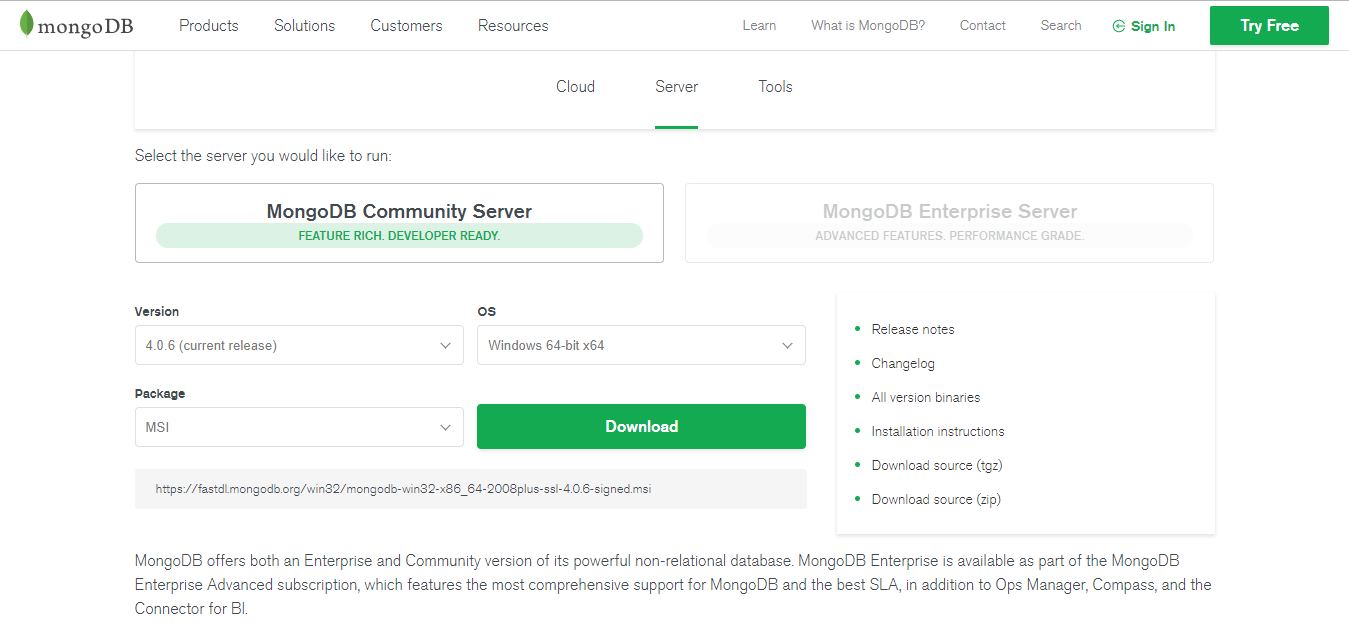
相信通过上面的说明，你已经大致了解了MongoDB的适用规则，需要说明一点的是，

MongoDB不仅仅是数据库，更多的使用是将MongoDB作为一个数据库中间件在实际应用中合理划分使用细节，这一点对于MongoDB应用来讲至关重要！

# MongoDB安装配置、常用命令及客户端

## 安装MongoDB数据库（Windows和Linux环境）

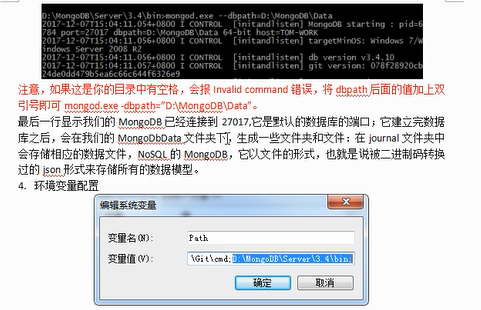
下载地址：<https://www.mongodb.com/download-center/community>



安装和启动

Windows版下载.zip

1. 在D盘创建安装目录，D:\MongoDB,将解压后的文件拷入新建的文件。
2. 在D盘创建一个目录，D:\MongoDB\Data,用于存放MongoDB的数据。
3. 执行安装，使用命令行，进入MongoDB的安装目录，执行安装命令，并指明存放MongoDB的路径。





# MongoDB高级应用

## MongoDB用户管理

1. 用户管理
   1. 添加用户

为testdb添加tom用户

|  |
| --- |
| use testdb  db.createUser({user:”tom”,pwd:”123”,roles:[{role:”dbAdmin”,db:”testdb”}]}) |

具体角色有

read：允许用户读取指定数据库

readWrite：允许用户读写指定数据库

dbAdmin：允许用户在指定数据库中执行管理函数，如索引创建、删除，查看统计或访问system.profile

userAdmin：允许用户向system.users集合写入，可以找指定数据库里创建、删除和管理用户

clusterAdmin：只在admin数据库中可用，赋予用户所有分片和复制集相关函数的管理权限

readAnyDatabase：只在admin数据库中可用，赋予用户所有数据库的读权限

readWriteAnyDatabase：只在admin数据库中可用，赋予用户所有数据库的读写权限

userAdminAnyDatabase：只在admin数据库中可用，赋予用户所有数据库的userAdmin权限

dbAdminAnyDatabase：只在admin数据库中可用，赋予用户所有数据库的dbAdmin权限

root：只在admin数据库中可用。超级账号，超级权限

* 1. 查看所有用户

|  |
| --- |
| db.system.users.find() |

和用户管理相关操作基本都要在admin数据库下运行，要先user admin;

如果在某个单一的数据库下，那只能对当前数据库的权限进行操作;

* 1. 用户删除操作

|  |
| --- |
| db.system.users.remove({user:”tom”}) |

* 1. 查看当前用户权限

|  |
| --- |
| db.runCommand({userInfo:”tom”,showPrivileges:true}) |

* 1. 单点

## MongoDB高可用方案实战演练

详解RouteServer(路由)、ConfigServer(配置)、Replica(副本集)、Shard(切片)、Chunk(分块)

## MongoDB启动与关闭

1. 命令行启动

|  |
| --- |
| $./mongodb --fork --dbpath=/data/program/mongodb-4.0.6/data/db |

1. 配置文件启动

|  |
| --- |
| $./mongodb --f mongodb.cfg |

1. MongoDB基本配置/data/program/mongodb-4.0.6/mongodb.cfg

|  |
| --- |
| dbpath=/data/program/mongodb-4.0.6/data/db  logpath=/data/program/mongodb-4.0.6/data/logs/mongodb.log  logappend=true  fork=true  bind\_ip=192.168.209.128  port=27017 |

1. 环境变量配置

|  |
| --- |
| export PATH=/data/program/mongodb-4.0.6/bin:$PATH |

MongoDB的三种集群方式的搭建：Master-Slaver/Replica Set/Sharding

## MongoDB主从搭建

这个是最简单的集群搭建，不过准确的说也不能算是集群，只能说是主备。并且官方已经不推荐这种方式，所以在这里只是简单的介绍下吧，搭建方式也相对简单。

主机配置 /data/program/mongodb-4.0.6/master-slave/master/mongodb.cfg