MongoDB 集群运维笔记

前面的文章介绍了 MongoDB 副本集和分片集群的做法，下面对 MongoDB 集群的日常维护操作进行小总结：

MongDB 副本集故障转移功能得益于它的选举机制。选举机制采用了 Bully 算法，可以很方便从分布式节点中选出主节点。Bully 算法是一种协调者(主节点)竞选算法，主要思想是集群的每个成员都可以声明它是主节点并通知其他节点。别的节点可以选择接受这个声称或是拒绝并进入主节点竞争。被其他所有节点接受的节点才能成为主节点。节点按照一些属性来判断谁应该胜出。这个属性可以是一个静态 ID，也可以是更新的度量像最近一次事务 ID(最新的节点会胜出)。

1）MongoDB 集群的节点数量

官方推荐 MongoDB 副本集的成员数量最好为奇数，且选举要求参与的节点数量必须大于成员数的一半。假设 MongoDB 集群有 3 个节点，那么只要有 2 个节点活着就可以选举；如果有 5 个，那么活 3 个节点就可以选举；如果有 7 个节点，那么活 4 个就可以选举.....
MongoDB 集群最多允许 12 个副本集节点，其中最多 7 个节点参与选举。这是为了减少心跳请求的网络流量和选举话费的时间，心跳每 2 秒发送一次。
MongoDB 集群最多 12 个副本集节点，是因为没必要一份数据复制那么多份，备份太多反而增加了网络负载和拖慢了集群性能；而最多 7 个节点参与选举是因为内部选举机制
节点数量太多就会导致 1 分钟内还选不出主节点，凡事只要适当就好。

2）MongoDB 心跳

整个 MongoDB 集群需要保持一定的通信才能知道哪些节点活着哪些节点挂掉。
MongoDB 节点会向副本集中的其他节点每两秒就会发送一次 pings 包，如果其他节点在 10 秒
钟之内没有返回就标示为不能访问。每个节点内部都会维护一个状态映射表，表明当前每个节点是什么角色、日志时间戳等关键信息。如果是主节点，除了维护映射
表外还需要检查自己能否和集群中内大部分节点通讯，如果不能则把自己降级为 secondary 只读节点。

3）MongoDB 同步

MongoDB 副本集同步分为初始化同步和 keep 复制。初始化同步指全量从主节点同步数据，如果主节点数据量比较大同步时间会比较长。而 keep 复制指初始化同步过后，
节点之间的实时同步一般是增量同步。初始化同步不只是在第一次才会被处罚，有以下两种情况会触发：

[1] secondary 第一次加入，这个是肯定的。

[2] secondary 落后的数据量超过了 oplog 的大小，这样也会被全量复制。

==========

何为 oplog？

oplog（应用日志）保存了数据的操作记录，oplog 主要用于副本，secondary 复制 oplog 并把里面的操作在 secondary 执行一遍。但是 oplog 也是 mongodb 的一个集合，保存在 local.oplog.rs 里；然而这个 oplog 是一个 capped collection，也就是固定大小的集合，新数据加入超过集合的大小会覆盖，所以这里需要注意，跨 IDC 的复制要设置合适的 oplogSize，避免在生产环境经常产生全量复制。oplogSize 可以通过--oplogSize 设置大小，对于 Linux 和 Windows 64 位，oplog size 默认为剩余磁盘空间的 5%。

在 mongodb 主从结构中，主节点的操作记录成为 oplog（operation log）。oplog 存储在一个系统数据库 local 的集合 oplog.$main 中，这个集合的每个文档都代表主节点上执行的一个操作。从服务器会定期从主服务器中获取 oplog 记录，然后在本机上执行！对于存储 oplog 的集合，MongoDB 采用的是固定集合，也就是说随着操作过多，新的操作会覆盖旧的操作！主节点通过--oplogSize 设置 oplog 的大小(主节点操作记录存储到 local 的 oplog 中)

==========

MongoDB 同步也并非只能从主节点同步，假设集群中 3 个节点，节点 1 是主节点在 IDC1，节点 2、节点 3 在 IDC2，初始化节点 2、节点 3 会从节点 1 同步数据。后面节点 2、节点 3 会使用就近原则从当前 IDC 的副本集中进行复制，只要有一个节点从 IDC1 的节点 1 复制数据。设置 mongodb、同步还要注意以下几点：

[1] secondary 不会从 delayed（延迟）和 hidden（隐藏）成员上复制数据。

[2] 要是需要同步，两个成员的 buildindexes 必须要相同无论是否是 true 和 false。

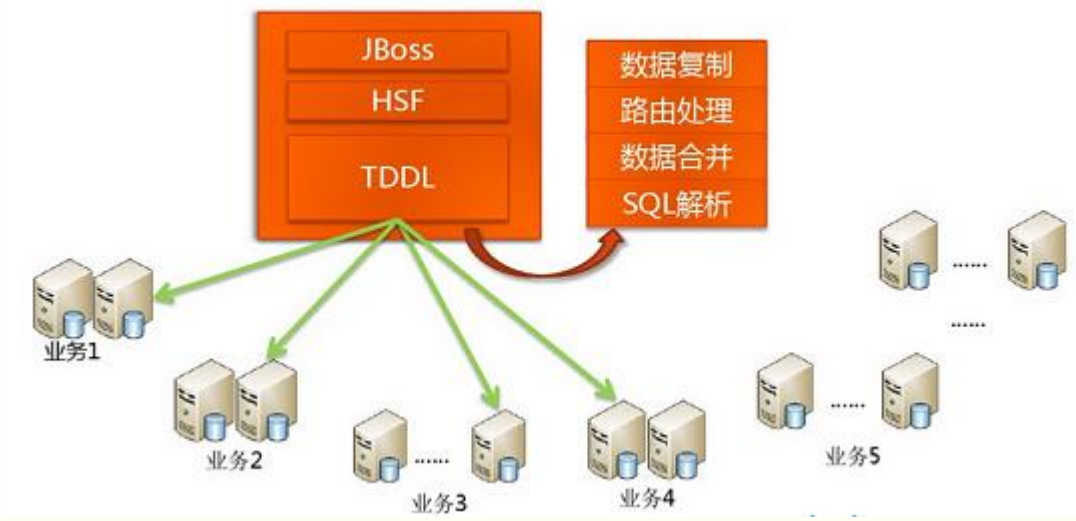
[3] buildindexes 主要用来设置是否这个节点的数据用于查询，默认为 true。

[4] 如果同步操作 30 秒都没有反应，则会重新选择一个节点进行同步。

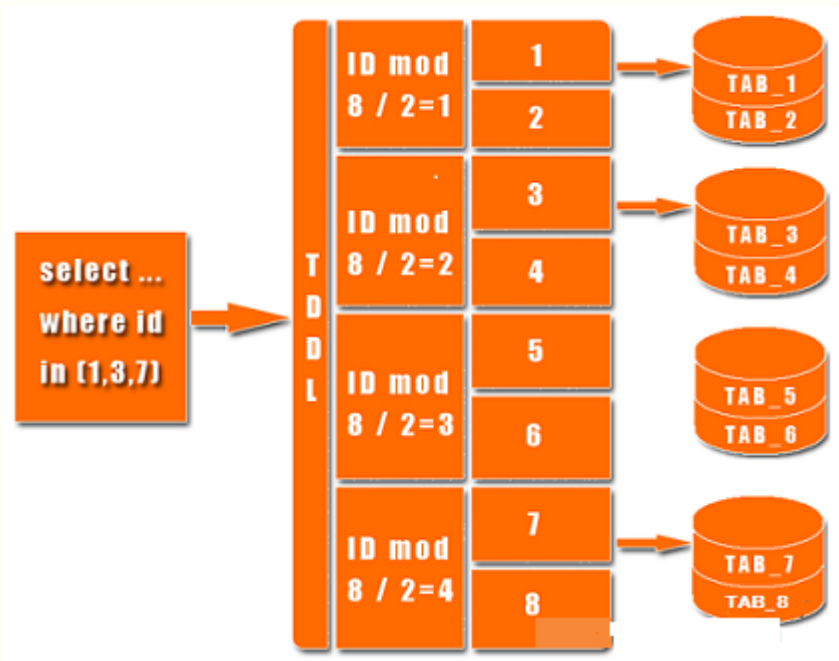
4）Mongodb 主节点的读写压力过大如何解决?

在系统早期，数据量还小的时候不会引起太大的问题，但是随着数据量持续增多，后续迟早会出现一台机器硬件瓶颈问题的。而 MongoDB 主打的就是海量数据架构，它不能解决海量数据怎么行! mongodb 的"分片"就是用来解决这个问题的。

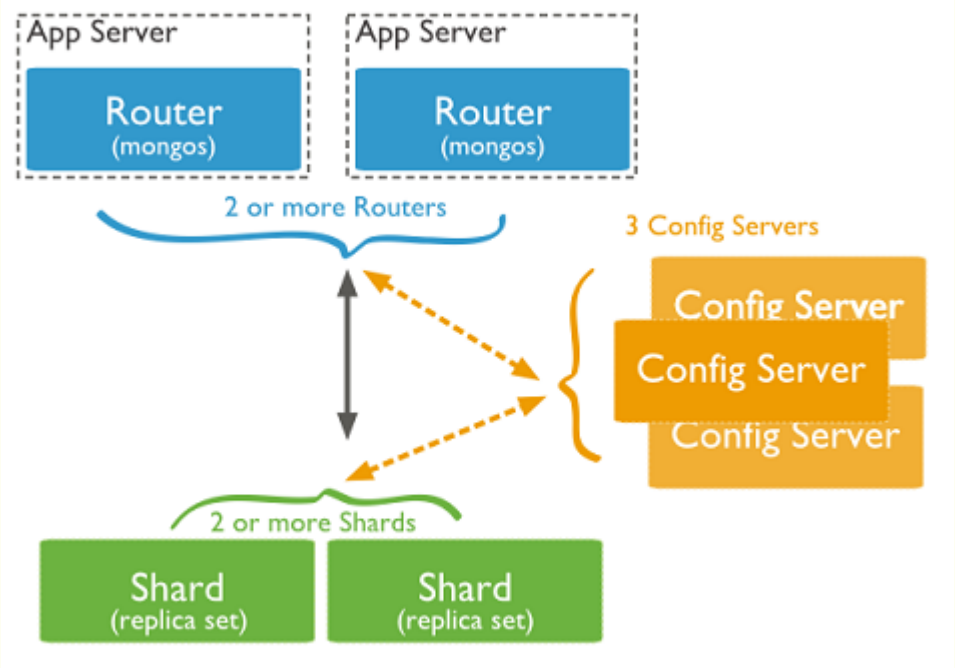
传统数据库怎么做海量数据读写?其实一句话概括：分而治之。如下 TaoBao 早期的一个架构图：



上图中有个 TDDL，是 TaoBao 的一个数据访问层组件，它主要的作用是 SQL 解析、路由处理。根据应用的请求的功能解析当前访问的 sql 判断是在哪个业务数据库、哪个表访问查询并返回数据结果。具体如图：



说了这么多传统数据库的架构，那 NoSQL 怎么去做到了这些呢？MySQL 要做到自动扩展需要加一个数据访问层用程序去扩展，数据库的增加、删除、备份还需要程序去控制。一但数据库的节点一多，要维护起来也是非常头疼的。不过 MongoDB 所有的这一切通过它自己的内部机制就可以搞定的了。如下图看看 MongoDB 通过哪些机制实现路由、分片：

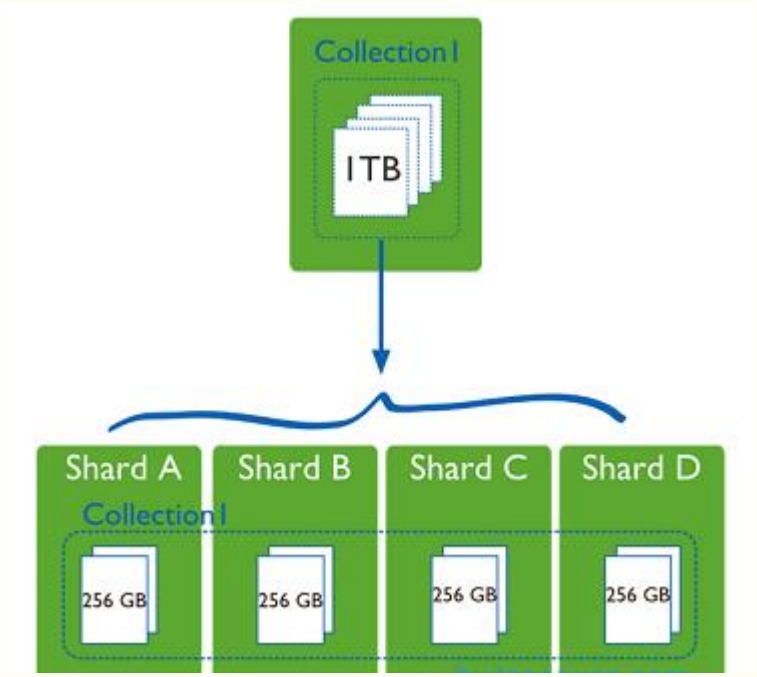


从图中可以看到有四个组件：mongos、config server、shard、replica set。

mongos，数据库集群请求的入口，所有的请求都通过 mongos 进行协调，不需要在应用程序添加一个路由选择器，mongos 自己就是一个请求分发中心，它负责把对应的数据请求请求转发到对应的 shard 服务器上。在生产环境通常有多 mongos 作为请求的入口，防止其中一个挂掉所有的 mongodb 请求都没有办法操作。

config server，顾名思义为配置服务器，存储所有数据库元信息(路由、分片)的配置。mongos 本身没有物理存储分片服务器和数据路由信息，只是缓存在内存里，配置服务器则实际存储这些数据。mongos 第一次启动或者关掉重启就会从 config server 加载配置信息，以后如果配置服务器信息变化会通知到所有的 mongos 更新自己的状态，这样 mongos 就能继续准确路由。在生产环境通常有多个 config server 配置服务器，因为它存储了分片路由的元数据，这个可不能丢失!就算挂掉其中一台，只要还有存货，mongodb 集群就不会挂掉。

shard，这就是传说中的分片了。上面提到一个机器就算能力再大也有天花板，就像军队打仗一样，一个人再厉害喝血瓶也拼不过对方的一个师。俗话说三个臭皮匠顶个诸葛亮，这个时候团队的力量就凸显出来了。在互联网也是这样，一台普通的机器做不了的多台机器来做，如下图：



一台机器的一个数据表 Collection1 存储了 1T 数据，压力太大了！在分给 4 个机器后，每个机器都是 256G，则分摊了集中在一台机器的压力。也许有人问一台机器硬盘加大一点不就可以了，为什么要分给四台机器呢？不要光想到存储空间，实际运行的数据库还有硬盘的读写、网络的 IO、CPU 和内存的瓶颈。在 mongodb 集群只要设置好了分片规则，通过 mongos 操作数据库就能自动把对应的数据操作请求转发到对应的分片机器上。在生产环境中分片的片键可要好好设置，这个影响到了怎么把数据均匀分到多个分片机器上，不要出现其中一台机器分了 1T，其他机器没有分到的情况，这样还不如不分片！

replica set（副本集），其实上图 4 个分片如果没有 replica set 是个不完整架构，假设其中的一个分片挂掉那四分之一的数据就丢失了，所以在高可用性的分片架构还需要对于每一个分片构建 replica set 副本集保证分片的可靠性。生产环境通常是 2 个副本 + 1 个仲裁（即一主一从一仲裁）。

5）MongoDB 复制集节点增加移除及节点属性配置

复制集(replica Set)或者副本集是 MongoDB 的核心高可用特性之一，它基于主节点的 oplog 日志持续传送到辅助节点，并重放得以实现主从节点一致。再结合心跳机制，当感知到主节点不可访问或宕机的情形下，辅助节点通过选举机制来从剩余的辅助节点中推选一个新的主节点

从而实现自动切换。对于一个已经存在的 MongoDB Replica Set 集群，可以对其进行节点的增加，删除，以及修改节点属性等等。

1）查看当前版本环境
```
repSetTest:PRIMARY> db.version()
3.2.11
```

注意：利用 rs.add 和 rs.remove 是不用 rs.reconfig 来使用配置生效的。

2）删除节点
```
repSetTest:PRIMARY> rs.remove("192.168.10.220:27000")
{  "ok"  : 1 }
```

移除节点后的状态信息
```
repSetTest:PRIMARY> rs.status()
```

移除后查看配置文件
```
repSetTest:PRIMARY> rs.config()
-----------------------------------------------------------------------
repmore:PRIMARY> config = {_id:"repmore",members:[{_id:0,host:'192.168.10.220:27017'
repmore:PRIMARY> rs.reconfig(config);      //使配置生效
repmore:PRIMARY> rs.status();      //查看节点状态
-----------------------------------------------------------------------
```

3）增加节点
```
repSetTest:PRIMARY> rs.add("192.168.10.220:27000")
repSetTest:PRIMARY> rs.status()
repSetTest:PRIMARY> rs.config()
-----------------------------------------------------------------------
```
或者使用下面方法添加节点：
```
repmore:PRIMARY> config =
{_id:"repmore",members:[{_id:0,host:'192.168.10.220:27017',priority :2},{_id:1,host:
点
repmore:PRIMARY> rs.reconfig(config);         //使配置生效
repmore:PRIMARY> rs.status();                          //查看节点状态
```

注意：新增节点的 replSet 要和其他节点要一样
```
-----------------------------------------------------------------------
```

4）启用 Arbiter 节点
```
repSetTest:PRIMARY> rs.remove("192.168.10.220:27000")
repSetTest:PRIMARY> rs.add({host:"192.168.10.220:27000",arbiterOnly:true})
```

对于 Arbiter 也可以使用 rs.addArb 函数来添加
```
> rs.addArb("192.168.10.220:27000")
```

6）MongoDB 复制集状态查看

复制集状态查询命令
1）复制集状态查询：rs.status()
2）查看 oplog 状态： rs.printReplicationInfo()
3）查看复制延迟： rs.printSlaveReplicationInfo()
4）查看服务状态详情: db.serverStatus()

==========

1)rs.status()
self:只会出现在执行 rs.status()命令的成员里
uptime:从本节点 网络可达到当前所经历的时间
lastHeartbeat：当前服务器最后一次收到其心中的时间
Optime & optimeDate:命令发出时 oplog 所记录的操作时间戳
pingMs: 网络延迟
syncingTo: 复制源
stateStr:

可提供服务的状态：primary, secondary, arbiter
即将提供服务的状态：startup, startup2, recovering
不可提供服务状态：down,unknow,removed,rollback,fatal

2)rs.printReplicationInfo()
log length start to end: 当 oplog 写满时可以理解为时间窗口
oplog last event time: 最后一个操作发生的时间

3).rs.printSlaveReplicationInfo()
复制进度：synedTo
落后主库的时间：X secs(X hrs)behind the primary

4).db.serverStatus()
可以使用如下命令查找需要用到的信息
db.serverStatus.opcounterRepl
db.serverStatus.repl

5).常用监控项目：
QPS: 每秒查询数量
I/O: 读写性能
Memory: 内存使用
Connections: 连接数
Page Faults: 缺页中断
Index hit: 索引命中率
Bakground flush: 后台刷新
Queue: 队列

7）MongoDB 复制集常用监控工具

1） mongostat
-h, --host　　主机名或 主机名：端口
--port　　端口号
-u ,--uername　　用户名（验证）
-p ,--password　　密码（验证）
--authenticationDatabase　　从哪个库进行验证
--discover　　发现集群某个其他节点

[root@centos6-vm01 ~]# mongostat -h 192.168.10.220:27017
[root@centos6-vm01 ~]# mongostat -h 192.168.10.220:27017　--discover

mongostat 重点关注的字段:
getmore 大量的排序操作在进行
faults 需要的数据不在内存中
locked db 锁比例最高的库
index miss 索引未命中
qr|qw 读写产生队列，供求失衡

```
[root@centos6-vm01 ~]# mongostat -h 192.168.10.220:27017
insert query update delete getmore command flushes mapped  vsize
    *0    *0     *0     *0       0     1|0       0 320.0M 879.0M 11
    *0    *0     *0     *0       0     1|0       0 320.0M 879.0M 11
    *0    *0     *0     *0       0     1|0       0 320.0M 879.0M 11
    *0    *0     *0     *0       0     1|0       0 320.0M 879.0M 11
    *0    *0     *0     *0       0     2|0       0 320.0M 879.0M 11
    *0    *0     *0     *0       0     1|0       0 320.0M 879.0M 11
    *0    *0     *0     *0       0     1|0       0 320.0M 879.0M 11
    *0    *0     *0     *0       0     1|0       0 320.0M 879.0M 11
    *0    *0     *0     *0       0     1|0       0 320.0M 879.0M 11
    *0    *0     *0     *0       0     2|0       0 320.0M 879.0M 11
```

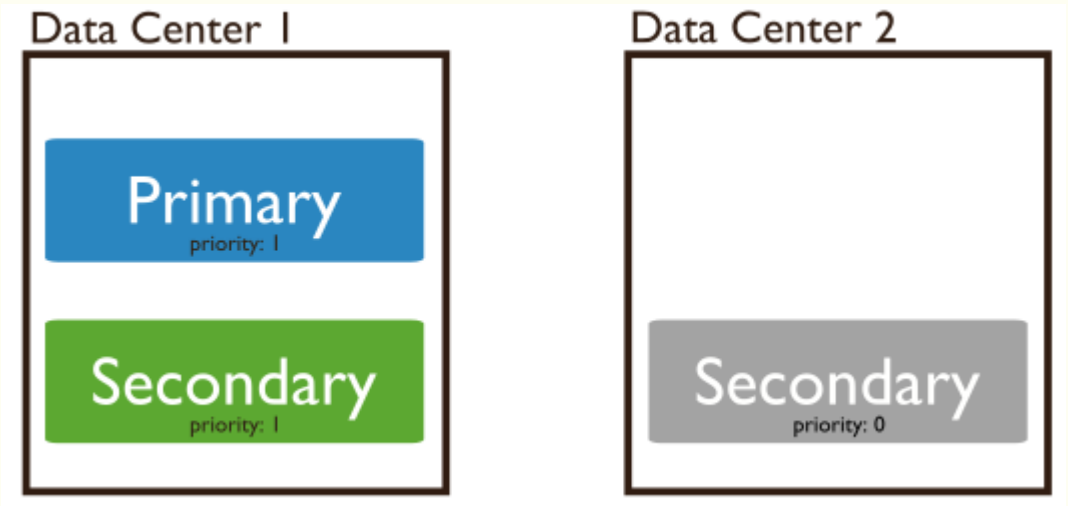
2) mongostop：与 mongostat 基本一样
-h, --host　　主机名或 主机名：端口
--port　　端口号
-u ,--uername　　用户名（验证）
-p ,--password　　密码（验证）
--authenticationDatabase　　从哪个库进行验证

```
[root@centos6-vm01 ~]# mongotop -h 192.168.10.220:27017
2018-01-03T09:31:15.675+0800    connected to: 192.168.10.220:27017

                                    ns    total    read    write    201
LuceneIndexDB.replicationColletion      0ms     0ms      0ms
     LuceneIndexDB.storeCollection      0ms     0ms      0ms
      LuceneIndexDB.system.indexes      0ms     0ms      0ms
   LuceneIndexDB.system.namespaces      0ms     0ms      0ms
             admin.system.indexes       0ms     0ms      0ms
          admin.system.namespaces       0ms     0ms      0ms
               admin.system.roles       0ms     0ms      0ms
               admin.system.users       0ms     0ms      0ms
            admin.system.version        0ms     0ms      0ms
                fragment.baseSe         0ms     0ms      0ms
```

8）对于 Secondary 来说有几个比较重要的属性：Priority 优先级、Vote 投票节点、Hidden 节点、Delayed 节点

设定节点的优先级别（Priority）

Priority=0 即优先级为 0 的节点。字面上来说，权限为 0。拥有最低的权限。已然是 Secondary 了，权限还最低，啥影响呢？之前说过，mongoDB 的副本集中是有投票机制的，如果一个 Primary 不可达，那么所有的 Secondary 会联合起来投票选举，选出心目中的新的 Primary。因为只有 Primary 才能接收 Writes 的操作，所以 Primary 在一个 mongoDB 的集群中是必须的。下图展示了一个在两个 IDC 中存放 Primary，Secondary，以及一个 Priority=0 的 Secondary 的场景(关于这个存放方式以及奇数偶数。



优先级为 0 的节点的特点

1）此节点丧失了当选 Primary 的机会。永远不会上位。
2）此节点正常参与 Primary 产生的 oplog 的读取，进行数据备份和命令执行。
3）此节点正常参与客户端对于数据的读取，进行担当负载均衡的工作。
4）此节点虽然不能当选 Primary 但是却可以投票，很民主。

Priority=0 在 mongoDB 中的解释就是一个 Standby，可投票不可参选，又干活又负载。有点像日本议会党派中刚入党派的小喽啰，可以参与自己党派党首的选举，还要干好多活，外面民众开骂还得挡着，但就是不可能当选党首。

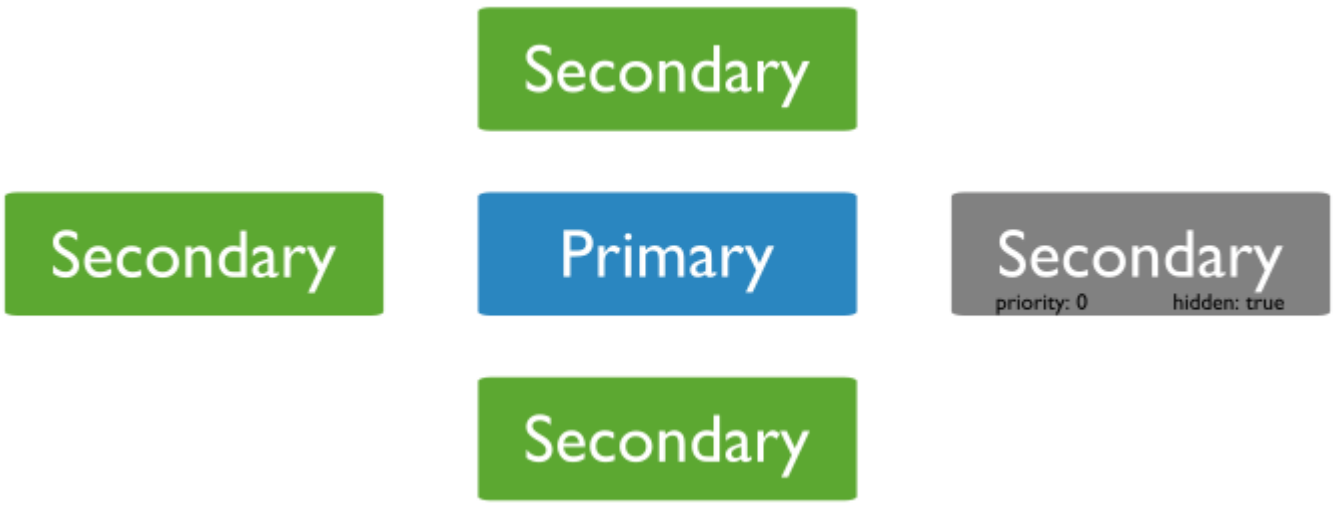
```
1）优先级用于确定一个倾向成为主节点的程度。取值范围为 0-100
2）Priority 0 节点的选举优先级为 0，不会被选举为 Primary，这样的成员称为被动成员
3）对于跨机房复制集的情形，如 A，B 机房，最好将『大多数』节点部署在首选机房，以确保能
4）对于 Priority 为 0 节点的情况，通常作为一个 standby，或由于硬件配置较差，设置为 0 以

如下示例，在新增节点的时候设定该节点的优先级别
repSetTest:PRIMARY> rs.add({"_id":3,"host":"localhost:27000","priority":1.5})

也可以通过下面的方式修改优先级别
repSetTest:PRIMARY> var config=rs.config()
repSetTest:PRIMARY> config.members[2].priority=2
2
repSetTest:PRIMARY> rs.reconfig(config)
{  "ok"  : 1 }
```

投票节点（Vote）

```
1  1）投票节点不保存数据副本，不可能成为主节点。
2  2）Mongodb 3.0 里，复制集成员最多 50 个，参与 Primary 选举投票的成员最多 7 个。
3  3）对于超出 7 个的其他成员（Vote0）的 vote 属性必须设置为 0，即不参与投票。
```

隐藏节点（Hidden）

字面上来说，隐藏。这个隐藏式对客户端的隐藏，客户端如果要读取 Secondary 的数据，永远无法读取 Hidden 节点的数据，因为设置了 Hidden 的这个节点对于客户端是透明的，不可见。但是，对于自己的 Secondary 的群体和 Primary 来说都是可见的，所以，Hidden 依然可以投票，依然要按照 oplog 进行命令的复制，只是，不参与负载了。Hidden 属性的前提是必须是一个 Priority=0 的节点，所以会具备一些优先级=0 的特点，具体如下。



Hidden 节点的特点

1）此节点丧失了当选 Primary 的机会。永远不会上位。
2）此节点正常参与 Primary 产生的 oplog 的读取，进行数据备份和命令执行。
3）此节点正常参与客户端对于数据的读取，进行担当负载均衡的工作。
4）此节点不参与客户端对于数据的读取，不进行负载均衡
5）此节点虽然不能当选 Primary 但是却可以投票，也很民主。

第 3 条特征体现出它与 Priority=0 的不同地方，第 4 条特征表现出它比 0 优先级多出来的特性。如果节点是 Hidden，它不参与负载，那么空闲出来的时间可以做一些赋予给它的特殊任务，比如数据备份等等。到应用场景的时候会有用处。

1）Hidden 节点不能被选为主（Priority 为 0），并且对 Driver 不可见。
2）因 Hidden 节点不会接受 Driver 的请求，可使用 Hidden 节点做一些数据备份、离线计算的任务
3）隐藏节点成员建议总是将其优先级设置为 0(priority 0)
4）由于对 Driver 不可见，因此不会作为 read preference 节点，隐藏节点可以作为投票节点
5）在分片集群当中，mongos 不会同隐藏节点交互

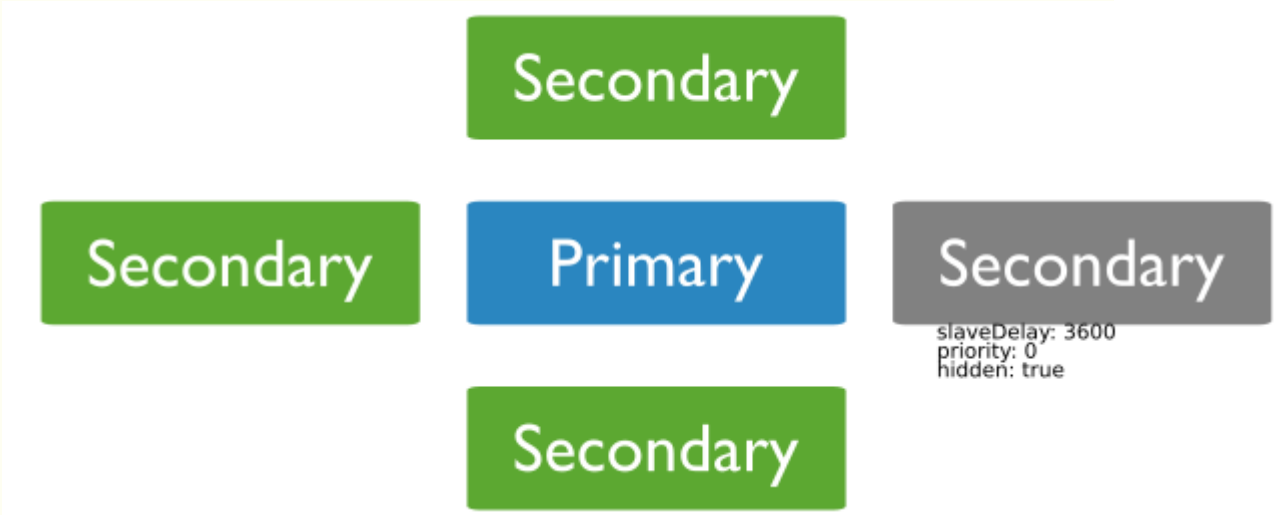
```
> cfg = rs.conf()
> cfg.members[2].priority = 0
> cfg.members[2].hidden =  true
> rs.reconfig(cfg)
```

查看设置为隐藏阶段后的属性

```
repSetTest:SECONDARY> db.isMaster()
{
"hosts"  : [
              "localhost:27000",
              "localhost:27001"
                                        ],
                                        "setName"  :  "repSetTest",
                                        "setVersion"  : 2,
                                        "ismaster"  :  false,
                                        "secondary"  :  true,
                                        "primary"  :  "localhost:27000",
                                        "passive"  :  true,
                                        "hidden"  :  true,          //此处表明当前节
                                        "me"  :  "localhost:27002",
                                        "maxBsonObjectSize"  : 16777216,
                                        "maxMessageSizeBytes"  : 48000000,
                                        "maxWriteBatchSize"  : 1000,
                                        "localTime"  : ISODate("2017-03-06T10:15:48.
                                        "maxWireVersion"  : 4,
                                        "minWireVersion"  : 0,
                                        "ok"  : 1
}
```

延迟节点（Delayed）

字面上来说，迟延。迟延代表此节点的数据与 Primary 的数据有一定的迟延，通过设定一个迟延的属性来确定。假设，Primary 的数据是 10:00 的最新数据，我们设置了一个 3600 秒的迟延参数，那么这个带有迟延的节点的数据或者说命令执行情况(在 oplog 中)应该只到 9:00 为止。与主节点有 1 小时的迟延。有些人可能会问，我们设计分布式数据库要的就是数据能够尽量避免迟延来达到一致，这样才能更好的提供服务，为什么要刻意制造迟延呢？试想这个场景：一个猪一样的队友在 mongoDB 的 Replica 集群上面执行了一个 Drop 操作，这个操作干掉了你的 Primary 的 Collection，这个 Drop 同时被记录到 oplog 中去，其他的 Secondary 看到这个 oplog 后争相执行，各自干掉了自己的 Collection，你苦心存储的数据就这么消失了。。。再怎么抽这个队友没用啊。所以，主动的过失避免就显得格外重要。如果你有一个 Delayed 节点，有一个 1000 秒的迟延，那么在你发现这个 miss 之后还有足够的时间可以响应去不让这个 Delayed 节点执行错误的 command，从而挽回你的损失。具体如下。



Delayed 节点的特点

1）此节点必须是一个 Priority=0 且为 Hidden 的节点，因为 Hidden 必须是 Priority=0 的，所以
2）此节点虽然又迟延又 Hidden 但是还是可以投票，也很民主。

Delayed 节点的最大作用是用来容人为的灾，猪一样的操作，驴一样的动作，在 Delayed 节点可以把损失降到最低。当然，如果你在 Delayed 时间经过后发现了错误，那么只能"呵呵"了。Delayed 的时间设定一定要大于响应时间，比如从 Primary 的 oplog 写到 Secondary 需要 1 秒，那么 Delayed 必须大于等于 1 秒，小于 1 秒的的话只能是一个不可及状态。

延迟节点包含复制集的部分数据，是复制集数据的子集
延迟节点上的数据通常落后于 Primary 一段时间（可配置，比如 1 个小时）。
当人为错误或者无效的数据写入 Primary 时，可通过 Delayed 节点的数据进行回滚

延迟节点的要求：
1）优先级别为 0(priority 0)，避免参与 primary 选举
2）应当设置为隐藏节点(以避免应用程序查询延迟节点)
3）可以作为一个投票节点，设置 members[n].votes 值为 1

延迟节点注意事项：
1）延迟时间应当等于和大于维护窗口持续期

2）应当小于 oplog 容纳数据的时间窗口

```
repSetTest:SECONDARY> cfg = rs.conf()
repSetTest:SECONDARY> cfg.members[2].priority = 0
repSetTest:SECONDARY> cfg.members[2].hidden =  true
repSetTest:SECONDARY> cfg.members[2].slaveDelay = 3600
repSetTest:SECONDARY> rs.reconfig(cfg)

repSetTest:SECONDARY> db.isMaster()
{
                                        "hosts"  : [
                                                "localhost:27000",
                                                "localhost:27001"
                                        ],
                                        "setName"  :  "repSetTest",
                                        "setVersion"  : 3,
                                        "ismaster"  :  false,
                                        "secondary"  :  true,
                                        "primary"  :  "localhost:27000",
                                        "passive"  :  true,
                                        "hidden"  :  true,
                                        "slaveDelay"  : 3600,          //此处表面当前
节点
                                        "me"  :  "localhost:27002",
                                        "maxBsonObjectSize"  : 16777216,
                                        "maxMessageSizeBytes"  : 48000000,
                                        "maxWriteBatchSize"  : 1000,
                                        "localTime"  : ISODate("2017-03-06T10:19:57.
                                        "maxWireVersion"  : 4,
                                        "minWireVersion"  : 0,
                                        "ok"  : 1
}
```

mongodb 副本集修改配置问题

因 ip 地址被占用，需要重新设置 ip 地址，这时需要修改副本集中的 IP 地址配置：
1）查看配置 rs.config()；需要找到 primary 主机，在该主节点服务器上才有权限修改配置
2）rs.remove("ip:port") 移除原配置文件中的已经变更地址的主机
3）rs.add("ip:port") 添加新的地址主机
4）设置 priority 优先级

```
> var config = rs.config()
> config.members[2].priority=2
> rs.reconfig(config)                    //重新更新配置
```

9）其他维护说明

一、新增副本集成员

1）登录 primary
2）
>use admin
>rs.add("new_node:port")
或者
>rs.add({"_id":4,"host":"new_node:port","priority":1,"hidden":false})
3）
>use admin
>rs.addArb("new_node:port")
或者
>rs.addArb({"_id":5,"host":"new_node:port"})
或者
>rs.add({'_id':5,"host":"new_node:port","arbiterOnly":true})

仲裁者唯一的作用就是参与选举，仲裁者并不保存数据，也不会为客户端提供服务。成员一旦以
它就永远只能是仲裁者，
无法将仲裁者重新配置为非仲裁者，反之亦然。最多只能有一个仲裁者每个副本集中。

温馨提示：
如果复制集中 priority=1 （默认），调用 rs.add("new_node:port") 该命令 会产生 主从切换
如果复制集中 priority=1 （默认），直接调用 rs.remove("new_node:port") 该命令 也会产生

二、删除副本集成员
1）登录要移除的目标 mongodb 实例；
2）利用 shutdownServer()命令关闭实例；即 db.shutdownServer()
3）登录复制集的 primary；
4）
primary>use admin
primary>rs.remove("del_node:port");

三、修改成员的优先级及隐藏性
1）登录 primary
2）
>use admin
>conf=rs.conf()
>conf.members[1].priority=[0-1000]
>conf.members[1].hidden=true　　#priority 必须为 0
>conf.members[9].tags={"dc":"tags_name1"}
>rs.reconfig(conf);　　# 强制重新配置 rs.reconfig(conf,{"force":true})

成员的属性有下列选项
　　[, arbiterOnly :　true]

```
            [, buildIndexes : <bool>]
            [, hidden :  true]
            [, priority: <priority>]
            [, tags: {loc1 : desc1, loc2 : desc2, ..., locN : descN}]
            [, slaveDelay : ]#秒为单位。
            [, votes :
```

如果该成员要设置为 隐藏(hidden:true) 或延迟(slaveDelay:30) 则其优先级 priority 必须设
也就是说 隐藏成员和延迟成员及 buildIndexs:false 的成员 的优先级别一定必须是 0 即 prior
优先级为 0 的成员不能发起选举操作。
只要优先级>0 即使该成员不具有投票资格，也可以成为主节点。
如果某个节点的索引结构和其他节点上的索引结构不一致，则该节点就永远不会变为主节点。
优先级用于表示一个成员渴望成为主节点的程度。优先级的取值范围是[0-100],默认为 1。优先
节点。
使用 rs.status()和 rs.config()能够看到隐藏成员，隐藏成员只对 rs.isMaster()不可见。客户
rs.isMaster()
来查看可用成员。将隐藏成员设定为非隐藏成员，只需将配置中的 hidden 设定为 false，或删除
每个成员可以拥有多个标签 tags {“dc":"tags_name2",qty:"tag_name3"}
votes：0 代表阻止这些成员在选举中投主动票，但是他们仍然可以投否决票。

修改副本集成员配置时的限制：
1）不能修改_id；
2）不能讲接收 rs.reconfig 命令的成员的优先级设置为 0；
3）不能将仲裁者成员变为非仲裁者成员，反正亦然；
4）不能讲 buildIndexes：false 改为 true；

四、查看副本集成员数据同步(延迟)情况
>db.printReplicationInfo();
>db.printSlaveReplicationInfo();#最好在 secondary 上执行
>rs.printReplicationInfo()
>rs.printSlaveReplicationInfo()
>use local>db.slaves.find()

在主节点上跟踪延迟：
local.slaves 该集合保存着所有正从当前成员进行数据同步的成员，以及每个成员的数据新旧程
登录主节点
>use local
>db.slaves.find()、
查看其中每个成员对应的"syncedTo":{"t":9999999,"i":32} 部分即可知道数据的同步程度。
"_id"字段的值是每个当前成员服务器标识符。可以到每个成员的 local.me.find()查看本服务器

1）

如果多台服务器拥有相同的_id 标识符，则依次登录每台服务器成员，删除 local.me 集合（loca
mongodb，
重启 mongod 后，mongod 会使用新的“_id”重新生成 local.me 集合。

2）
如果服务器的地址改变但_id 没有变,主机名变了,该情况会在本地数据库的日志中看到重复键异
exception)。
解决方法是：删除 local.slaves 集合即可，不需要重启 mongod。

因为 mongod 不会清理 local.slaves 集合，故里面的数据可能不准确或过于老旧，可将整个集合
成员作为
复制源时，该集合会重新生成。如果在主节点中看到了某个特定的服务器在该集合中有多个文档
复制链，
该情况不影响数据同步，只是把每个备份节点的同步源告诉主节点。

删除 local.me 集合，需要重新启动 mongod，mongod 启动时会使用新的 _id 重新生成 local.me 集

删除 local.slaves 集合，不用重启 mongod。该集合中的数据是记录该成员被作为同步源的服务
本集状态。
删除后不久如果有新的节点成员将该服务器节点作为复制源，该集合就会重新生成。

五、主节点降为 secondary
```
>use admin
>rs.stepDown(60)#单位为 秒
```

六、锁定指定节点在指定时间内不能成为主节点（阻止选举）
```
>rs.freeze(120)#单位为 秒
```
释放阻止
```
>rs.freeze(0)
```

七、强制节点进入维护模式（recovering）
可以通过执行 replSetMaintenanceMode 命令强制一个成员进入维护模式。
例如自动检测成员落后主节点指定时间，则将其转入维护模式：
```
>function  maybeMaintenanceMode(){
var  local=db.getSisterDB("local");
if  (!local.isMaster().secondary)
{return;
)
var last=local.oplog.rs.find().sort({"$natural":-1}).next();
var lasttime=last['ts']['t'];
```

```
if  (lasttime<(new  date()).getTime()-30)
{db.adminCommand({"replSetMaintenanceMode":true});
}
};
```

将成员从维护模式转入正常模式。即恢复：

```
>db.adminCommand({"replSetMaintenanceMode":false});
```

八、阻止创建索引（不可再修改为可以创建索引，故慎重考虑）

通常会在备份节点的延迟节点上设置阻止创建索引。因为该节点通常只是起到备份数据作用。设
可。该选项是永久性的。

如果要将不创建索引的成员修改为可以创建索引的成员，那么必须将这个成员从副本集中移除，
再将其重新添加到副本集中。

并且允许其重新进行数据同步。该选项也要求成员的优先级为 0.

九、指定复制源（复制链） 查看复制图谱

使用 db.adminCommand({"replSetGetStatus":1})['syncingTo'] 可以查看复制图谱（每个节点

在备份节点上执行 rs.status()['sysncingTo'] 同样可以查看复制图谱（同步源）；

mongodb 是根据 ping 时间来选择同步源的，会选择一个离自己最近而且数据比自己新的成员作为

可以使用 replSetSyncFrom 命令来指定复制源或使用辅助函数 rs.sysncFrom()来修改复制源。

```
db.adminCommand({"replSetSyncFrom":"server_name:port"});
```

副本集默认情况下是允许复制链存在的，因为这样可以减少网络流量。但也有可能
花费是时间更长，因为复制链越长，将数据同步到全部服务器的时间有可能就越长（比如每个备
微旧点，这样就得
从主节点复制数据）。

解决方法：

1)手动改变复制源

登录备份节点：

```
>use admin
>db.adminCommand({"replSetSyncFrom":"新复制源"})
```

或者

```
>rs.syncFrom("新复制源")
```

副本集中的成员会自动选择其他成员作为复制源。这样就会产生复制链。

如果一个备份节点从另一个备份节点（而非主节点）复制数据时，就会形成复制链。

复制链是可以被禁用的，这样就可以强制要求所有备份节点都从主节点复制数据。

禁用复制链：即禁止备份节点从另一个备份节点复制数据。

```
>var config=rs.config()
>config.settings=config.settings ||{}
>config.settings.chainingAllowed=false
```

```
>rs.reconfig(config);
```

十、强制修改副本集成员

```
>var config=rs.config()
>config.member[n].host=...
>config.member[n].priority=...
.....
>rs.reconfig(config,{"force":true})
```

十一、修改 Oplog 集合的大小

如果是主节点，则先将 primary 降为 secondary。最后确保没有其他 secondary 从该节点复制数

1）关闭该 mongod 服务 use admin >db.adminCommand({shutdownServer:1});

2）以单机方式重启该 mongod（注释掉 配置文件中的 replSet shardsvr ，修改端口号）

3）将 local.oplog.rs 中的最后一条 insert 操作记录保存到临时集合中

```
> use  local
>var cursor=db.oplog.rs.find({"op":"i"});
>var lastinsert=cursor.sort({$natural:-1}).limit(1).next();
>db.templastop.save(lastinsert);
>db.templastop.findOne()#确保写入
```

4）将 oplog.rs 删除： db.oplog.rs.drop();

5）创建一个新的 oplog.rs 集合： db.createCollection("oplog.rs":{"capped":true,"size":

6）将临时集合中的最后一条 insert 操作记录写回新创建的 oplog.rs：

```
>var temp=db.templastop.findOne();
>db.oplog.rs.insert(temp);
>db.oplog.rs.findOne()  #确保写回，否则 该节点重新加入副本集后会删除该节点上所有数据
```

7）最后将该节点以副本集成员的身份重新启动即可。

十二、为复制集成员设置选项

即当运行 rs.initiate(replSetcfg) 或运行 rs.add(membercfg)选项时，需要提供描述复制集成

```
{
_id:replSetName,
members:
[
{_id:<number>,host:<hostname|ip[:port]>,
[priority:<priority>,]#默认值为 1.0.即选项的值是浮点型
[hidden:true,]#该元素将从 db.isMaster()的输出中隐藏该节点。
[arbiterOnly:true,]#默认值为 false
[votes:<n>,]#默认值为 1 。改选项的值为整形
[tags:{documents},]
[slaveDelay:<seconds>,]
[buildIndexes:true,]#默认值为 false。该选项用于禁止创建索引。
```

}],
settings:{
[chainingAllowed:<boolen>,]#指定该成员是否允许从其他辅助服务器复制数据。默认值为 tru
[getLastErrorModes:<modes>,]#模式：用于自定义写顾虑设置
[getLastErrorDefaults:<lasterrdefaults>,]#默认值：用于自定义写顾虑设置。
}
}
以上是复制集的完整的配置结构。最高级的配置结构包括 3 级：
_id、members、settings。
_id 是复制集的名称，与创建复制集成员时时候用的 --replSet 命令选项时提供的名称一样。
members 是数组，由一个描述每个成员的集合组成；这是添加单个服务器到集合中时，应该在 rs
构；
settings 也是数组，该 settings 数组包含应用到整个复制集的选项。这些选项可以设置复制集

十三、Rollback
mongodb 只支持小于 300M 的数据量回滚，如果大于 300M 的数据需要回滚或要回滚的操作在 30 分
会在 mongodb 日志中报以下错误：
[replica set sync] replSet syncThread: 13410 replSet too much data to roll back
经量避免让 rollback 发生。方法是：使用 复制的 写顾虑（Write Concern）规则来阻止回滚的
如果发生了回滚操作，则会在 mongodb 数据文件目录下产生一个以 database.collection.times
件的内容用 bsondump 工具来查看。

十四、读偏好设置（ReadPreferred）
读取偏好是指选择从哪个复制集成员读取数据的方式。可以为驱动指定 5 中模式来设置读取偏好
readPreference=primary|primaryPreferred|secondary|secondaryPreferred|nearest
setReadPreferred()命令设置读取偏好。

primary:只从主服务器上读取数据。如果用户显式指定使用标签读取偏好，该读取偏好将被阻塞
primaryPreferred:读取将被重定向至主服务器；如果没有可用的主服务器，那么读取将被重定向
secondary：读取将被重定向至辅助服务器节点。如果没有辅助服务器节点，该选项将会产生异常
secondaryPreferred：读取将被重定向至辅助服务器；如果没有辅助服务器，那么读取将被重定
的“slaveOK”方法；
nearest:从最近的节点读取数据，不论它是主服务器还是辅助服务器。该选项通过网络延迟决定

十五、写顾虑设置（Write Concern）
写顾虑类似读取偏好，通过写顾虑选项可以指定在写操作被确认完成前，数据必须被安全提交到
写顾虑的模式决定了写操作时如何持久化数据。参数“w”会强制 getLastError 等待，一直到给
的写入操作。w 的值是包含主节点的。
写顾虑的 5 中模式：
w=0 或不确定：单向写操作。写操作执行后，不需要确认提交状态。
w=1 或确认：写操作必须等到主服务器的确认。这是默认行为。

w=n 或复制集确认：主服务器必须确认该写操作。并且 n-1 个成员必须从主服务器复制该写入操
起延迟。
w=majority:写操作必须被主服务器确认，同时也需要集合中的大多数成员都确认该操作。而 w=
延迟引起问题。
j=true 日志：可以与 w=写顾虑一起共同指定写入操作必须被写入到日志中，只有这样才算是确
wtimeout：避免 getLastError 一直等待下去，该值是命令的超时时间值，如果超过这个时间还
值的单位是毫秒。如果返回失败，
值在规定的时间内没有将写入操作复制到"w"个成员。
该操作只对该连接起作用，其他连接不受该连接的"w"值限制。

```
db.products.insert(
{ item :  "envelopes" , qty : 100 ,  type  :  "Clasp"  },
{ writeConcern : { w : 2 , wtimeout : 5000 } }
)
wtimeout#代表 5 秒超时
```

修改默认写顾虑

```
cfg = rs.conf()
cfg.settings = {}
cfg.settings.getLastErrorDefaults = { w:  "majority"  , wtimeout: 5000 }
rs.reconfig(cfg)
```

设置写等待

```
db.runCommand({"getLastError":1,w:"majority"})
```

或

```
db.runCommand({"getLastError":1,"w":"majority","wtimeout":10000})
```

即，表示写入操作被复制到了多数个节点上(majority 或 数字），这时的 w 会强制 getLastEr
成员执行完了最后的写入操作。

而 wtimeout 是指超过这个时间没有返回则返回失败提示（即无法在指定时间内将写入
getLastError 并不代表写操作失败了，

而是代表在指定给定 wtimeout 时间内没有将写入操作复制到指定的 w 个成员中。w 是限
阻塞这个连接上的操作，其他连接上

的操作不受影响。

十六、读取偏好和写顾虑中使用标签(tags)

十七、选举机制

1）自身是否能够与主节点连通；
2）希望被选件为主节点的备份节点的数据是否最新；
3）有没有其他更高优先级的成员可以被选举为主节点；
4）如果被选举为主节点的成员能够得到副本集中“大多数”成员的投票，则它会成为主节点，
个否决了本次选举，则本次选举
失败即就会取消。一张否决票相当于 10000 张赞成票。

5）希望成为主节点的成员必须使用复制将自己的数据更新为最新；

十八、数据初始化过程

1）首先做一些记录前的准备工作：选择一个成员作为同步源，在 local.me 集合中为自己创建一
数据库，以一个全新的状态
开始进行同步；该过程中，所有的数据都会被删除。
2）然后克隆，就是将同步源的所有记录全部复制到本地。
3）然后就进入 oplog 同步的第一步，克隆过程中的所有操作都会被记录到 oplog 中。
4）接下来就是 oplog 同步过程的第二步，用于将第一个 oplog 同步中的操作记录下来。
5）截止当前，本地的数据应该与主节点在某个时间点的数据集完全一致了，可以开始创建索引
6）若当前节点的数据仍然落后同步源，那么 oplog 同步过程的最后一步就是将创建索引期间的
止该成员成为备份节点。
7）现在当前成员已经完成了初始化数据的同步，切换到普通状态，这时该节点就可以成为备份

十九、mongodb3.0 建议开启的设置

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
# echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

执行上面两命令后只是当前起作用。如果重启 mongod 服务后 就失效。永久起效则
写入到 /etc/rc.local
即

二十、修改服务器 hostname 名

1）首先修改 secondary 服务器的 hostname;然后 stop secondary;
2）重启 secondary 以修改后的新 hostname；
3）登录 primary ；
4）用 rs.reconfig()命令修改 复制集的配置信息；

```
>conf=rs.conf()
>conf.members[x].host='new_address:27017'
>rs.reconfig(conf);
```

二十一、生成 keyfile 文件

```
# openssl rand -base64 666 > /opt/mongo/conf/MongoReplSet_KeyFile
# chown mongod.mongod /opt/mongo/conf/MongoReplSet_KeyFile
# chmod 600 /opt/mongo/conf/MongoReplSet_KeyFile
```