22.7 优化器 profiler

我们在 22.5 节“索引设置的技巧”中讲到可以使用 explain 分析 MongoDB 中的查询操作。除了使用 explain 之外，MongoDB 还提供了优化器 profiler 可以分析读写等操作，主要用于分析慢查询。默认情况下，慢查询指的是查询时间在 100ms 以上的操作，这个值可以更改。

profiler 优化器默认是关闭的，要使用 profiler 必须先开启它。有两种方式可以控制 profiler 的开关和级别。

（1）方式一：在启动 MongoDB 时加上--profile 参数，通过这个参数来设置 profiler 的级别。

（2）方式二：在 mongo 客户端使用 db.setProfilingLevel（级别）命令来实时配置，可以通过 db.getProfilingLevel()命令来获取当前的 profiler 级别。

level 有三种级别：

- 0：不开启。
- 1：记录慢命令，默认为>100ms。
- 2：记录所有命令。

参数为 1 的时候，默认的慢命令是大于 100ms，当然也可以进行设置，使用命令如下：

```
db.setProfilingLevel(1,120);
```

设置慢查询的标准为大于 120ms。

Mongodb Profile 记录是直接存在系统 db 里的，记录位置是集合 system.profile，system.profile 是一个固定集合。我们只要查询这个 Collection 的记录就可以获取到我们的 Profile 记录了。使用命令如下：

```
db.system.profile.find().pretty()
```

当然也可以搭配查询参数来查询，比如查询最新的记录使用命令：

```
db.system.profile.find().sort({$natural:-1}).pretty()
```

还有一种更简洁的查看方式是：

```
show profile
```

该命令可以查看最近的 5 条记录。

profile 提供的信息重要参数如下：

- ts：该命令在何时执行。
- millis：执行耗时，以毫秒为单位。
- op：什么操作。
- query：设置的查询条件。

- nReturned：返回的条数。
- docsExamined：文档扫描条数。

清空还原 profile 集合使用命令：

```
db.setProfilingLevel(0)
db.system.profile.drop()
db.createCollection("system.profile", { capped: true, size:4000000 } )
db.setProfilingLevel(1)
```

更多参数的含义请参考官网链接：

```
https://docs.mongodb.com/manual/tutorial/manage-the-database-profiler/
```

第 23 章

◂MongoDB管理的经验▸

23.1 MongoDB 安全管理

MongoDB 默认情况下没有开启用户认证，MongoDB 安装即用的方便性给我们留下了深刻的印象，分布式集群的功能也是它的特色之一。但是如果不开启用户认证，数据就得不到安全的保证，开启用户认证之后又会给 MongoDB 的使用和维护带来很大的不方便，而且因为 MongoDB 用户是指定到库的，每次创建连接时的密码验证会对 MongoDB 性能造成影响（即使 MongoDB 一直在优化用户认证）。因为开启了之后涉及客户端用户认证访问，以及分布式集群之间的相互访问，用起来就没那么方便了，集群搭建的难度也成几何倍数地上升。

那有没有好的解决办法呢？MongoDB 官方建议将 MongoDB 运行在一个可信任的网络环境中。

我们在生产环境中如果不想启用用户认证，可以使用 Linux 防火墙等功能配合使用进行管理，不把生产环境的数据库暴露在公网上，创造一个可信任的网络环境。

1. 方式一：iptalbes 设置

Linux 防火墙设置只有应用程序所在的服务器 ip 才能访问 MongoDB 所在的服务器。

Linux 禁止某个 IP 地址访问其实非常简单，最常用的办法就是使用 iptalbes 来操作。这个方法跟 MongoDB 本身没有关系，而是借用 Linux 的 iptalbes 功能，限制允许访问 MongoDB 端口的 ip 地址，具体做法（ip 和端口需要读者自己对应）如下：

```
# 拒绝所有访问27017端口的请求
sudo iptables -I INPUT -p tcp --dport 27017 -j DROP

# 允许123.123.123.123服务器访问 mongo 端口
sudo iptables -I INPUT -s 123.123.123.123 -p tcp --dport 27017 -j ACCEPT

sudo iptables-save
```

或者

```
vi /etc/sysconfig/iptables
```

把这两句：

```
iptables -I INPUT -p tcp --dport 27017 -j DROP
iptables -I INPUT -s 123.123.123.123 -p tcp --dport 27017 -j ACCEPT
```

加在这两句：

```
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
```

的前面，然后重启防火墙：

```
service iptables restart
```

查看防火墙状态：

```
service iptables status
```

这样就只允许 123.123.123.123 服务器访问 MongoDB 服务了。

注意命令的顺序不能反了。如果不只限制一个端口，而是限制所有端口的访问，把--dport 27017 去掉即可。

更多详情可参考官网链接：

```
https://docs.mongodb.com/manual/tutorial/configure-linux-iptables-firewall/
```

2. 方式二：hosts.allow 和 hosts.deny

Linux 中的配置文件/etc/hosts.allow 控制可以访问本机的 IP 地址，/etc/hosts.deny 控制禁止访问本机的 IP。执行的顺序是先 deny 再 allow，所以如果两个文件的配置有冲突，以/etc/hosts.allow 为准。

/etc/hosts.allow 和/etc/hosts.deny 两个文件是控制远程访问设置的，通过它们可以允许或者拒绝某个 ip 或者 ip 段的客户访问 Linux 的某项服务。服务用进程名来识别，比如 MongoDB 的服务进程名是 mongod，我们限制所有 ip 访问，除非 ip 是 123.123.123.123。

编辑 hosts.deny：

```
vi /etc/hosts.deny
```

拒绝一切 ip 访问 MongoDB 服务输入内容：

```
# no mongod
mongod:all:deny
```

按 Esc 键，输入:wq，保存离开。

mongod:all:deny 表示拒绝所有 ip 访问 mongod 服务，:deny 可以省略，写成 mongod:all。

编辑 hosts.allow：

```
vi /etc/hosts.allow
```

允许 123.123.123.123 访问 MongoDB 服务输入内容：

```
mongod:123.123.123.123
```

按 Esc 键，输入:wq，保存离开。

修改完后重启拦截器让刚才的更改生效：

```
service xinetd restart
```

经过设置之后需要进行校验，测试限制是否生效。如果是比较重要的数据，不满足于限制 ip 访问，那就把用户认证加上，这个需要读者自己权衡。

对于安全度要求高的数据库，还可以启用 SSL。

如果你没有使用 SSL，那么你在 MongoDB 客户端和 MongoDB 服务器之间的传输的数据就是明文的，容易受到窃听、篡改和“中间人”攻击。如果你是通过公网这样的非安全网络连接到 MongoDB 服务器，那么启用 SSL 就显得非常重要。

详细的 SSL 配置可以查看官网链接：

```
https://docs.mongodb.com/manual/tutorial/configure-ssl/
```

23.2 不要将 MongoDB 与其他服务部署到同一台机器上

MongoDB 的内存使用方式，导致了它很吃内存，所以如果其他服务与 MongoDB 放在同一服务器，会相互影响，导致性能下降。尤其是 NUMA（非统一内存访问），NUMA 是一种用于多处理器的电脑记忆体设计，内存访问时间取决于处理器的内存位置。在 NUMA 下，处理器访问它自己的本地存储器的速度比非本地存储器（本地存储器到另一个处理器之间共享的处理器或存储器）快一些。许多厂商都成功推出了基于 NUMA 架构的服务器，Linux 跟 NUMA 的搭配使用比较常见。NUMA 改变了系统原生的内存使用方式，如果是 Linux+NUMA+MongoDB 的服务器，MongoDB 性能会受到很大影响，所以如果开启有 NUMA 的服务器，最好不要用来安装 MongoDB。在生产环境中，一般用专门的服务器来作为 MongoDB 数据库的服务器。

在使用 NUMA 的机器上运行 MongoDB，MongoDB 日志中会提示警告信息如下：

```
WARNING: You are running on a NUMA machine.
We suggest launching mongod like this to avoid performance problems:
numactl --interleave=all mongod [other options]
```

这种情况下会严重影响性能，请在启动 MongoDB 时关闭 NUMA 功能，按照提示在启动

命令前加上 numactl --interleave 选项，启动时使用命令如下：

```
numactl --interleave=all  mongod  --dbpath=/data/db/ --fork --
logpath=/data/logs/db.log
```

如果系统中没有 numactl 命令，使用 yum 安装：

```
yum install -y numactl
```

再使用命令：

```
echo 0 > /proc/sys/vm/zone_reclaim_mode
vi /proc/sys/vm/zone_reclaim_mode
sysctl -w vm.zone_reclaim_mode=0
```

更多 NUMA 处理信息参考官网链接：

```
https://docs.mongodb.com/manual/administration/production-notes/#production-
numa
```

23.3 单机开启日志 Journal，多机器使用副本集

很多人抱怨 MongoDB 是内存数据库，也没有事务，容易丢失数据，其实这都是对 MongoDB 的误解；MongoDB 有完整的 redolog、binlog 和持久化机制，在正确使用 MongoDB 的情况下，不必太担心数据丢失问题。

Journal 是 MongoDB 中的 redolog，而 Oplog 则是负责副本集的 binlog。

不开启 Journal 的情况下，数据会写入内存，然后等待系统写入硬盘，一般每 60 秒写入一次到硬盘中，如果这段时间里断电或者机器崩溃了，就会损失内存中没有写入到磁盘中的数据。60 秒左右的数据，是难以承受的。

从 1.9.2+版本开始，MongoDB 默认打开 Journal 功能，以确保数据安全。

打开 Journal 的情况下，MongoDB 每 100 毫秒左右往 Journal 文件中写入一次数据（Journal 日志和 data 数据在一个磁盘上时每隔 100 毫秒刷新一次，不在一个磁盘上时 30 毫秒刷新一次，建议把 Journal 日志和 data 数据放在不同的磁盘上，提高数据的可靠性），那么即使机器崩溃了，日志文件还在的情况，经过恢复也只丢失了 100 毫秒左右的数据，很给力地保护了数据的安全。而且 Journal 的刷新时间是可以改变的，使用 --journalCommitInterval 命令修改，范围是 2~300 毫秒。值越低，刷新输出频率越高，数据安全度也就越高，但磁盘性能上的开销也更高。

日志 Journal 能保证单个服务器的数据安全，所有的操作记录都被记录在日志中并定期写入磁盘。如果机器崩溃了，但硬盘还是好的，就能重启服务器，数据会自动根据日志完成修复。记住如果是硬盘出现问题，MongoDB 也无能为力了。

条件允许的话，最好使用多机器副本集对数据进行保障，尤其是重要的数据。

非常重要的数据建议副本集和日志 Journal 同时使用。

23.4 生产环境不要信任 repair 恢复的数据

如果数据库崩溃了，而且没有开启--journal 的情况下，千万不要将这些数据拿来就用，因为其中已经有损坏的文档了，另外，也可能由于索引混乱导致返回的结果不完整等。崩溃导致的问题比较严重，而且会潜伏很长一段时间不被发现。

所以需要对数据库进行处理，我们之前在 9.7 节修复未正常关闭的 MongoDB 中说过可以使用 mongod --repair 命令修复一次，再正常启动。修复过程是将所有的文档导出后马上导入，忽略无效的文档，完成后，会重建索引。但是这并不是最优选择，而且会丢失损坏了的文档。

这个过程很耗时，而且需要大量磁盘空间（至少与现在使用的空间相同）。修复之后数据库是正常了，但是可能很多损坏的文档都找不到了，这对重要数据来说无疑是致命的，而且你不知道它都丢弃了哪些文档。

所以我们建议当灾难发生后，生产环境中最好的恢复数据的方法是启动数据库清空它之后，使用完整备份下来的数据备份文件（例如 mongodump 等命令备份的文件）来进行数据库的恢复。mongod --repair 是不得已的最后一招。尽可能稳妥地备份数据库，经常做备份，这些才是最有效的管理数据的手段；不过遗憾的是这种方式的备份恢复还是会丢失备份时间到崩溃时间之间的所有数据。我们可以在清空旧数据库之前先备份一份，然后使用旧数据库中的数据与备份文件恢复的数据进行对比，尽可能地找回备份时间到崩溃时间之间的数据。

比较完美的方法是做好时间节点的完全备份和增量备份。

当然，最好的情况是数据库不崩溃，使用副本集是一个很好的容灾措施，一个实例崩溃之后有另一个实例可以无缝连接使用。

需要注意的是，如果数据量太大，使用 mongodump 备份时的查询会对 MongoDB 数据库的性能产生一些影响；而且如果在 mongodump 备份时还有大量的写入操作，则备份的数据可能缺失这部分新写入的数据。

所以，我们建议在应用程序用户少时才进行 mongodump 备份，而且有一种方式可以保证得到实时完整的所有数据备份，一条写入记录都不丢失。那就使用 fsync 和锁。

MongoDB 的 fsync 命令能够在 MongoDB 运行时复制数据目录还不会损坏数据。fsync 命令强制服务器把所有内存缓冲区的数据写入磁盘，还可以使用 lock 命令选择上锁，阻止对数据库的写入，直到释放锁为止。写入锁是让 fsync 在备份时发挥作用的关键。使用命令如下：

```
use admin
db.runCommand({"fsync":1,"lock":1});
```

这时，磁盘中的数据是实时最新的，而且上了写入锁，不会有新的写入产生了。这个时候使用 mongodump 进行备份数据即可。

备份完毕后需要解锁，使用命令：

```
db.$cmd.sys.unlock.findOne();
```

初次请求解锁会花点时间。

使用{"fsync":1,"lock":1}实现了备份实时的数据，不需要停掉服务器，要付出的代价就是一些写入操作被暂时阻塞了。解锁后被阻塞的写入操作会继续执行，服务器恢复正常工作。

那还有没有更好的备份方式呢，不需要耽误读写还能保证备份的是实时的数据？

那就是从属备份，也就是在从服务器上备份。因为从服务器的数据几乎与主服务器同步，而且不负责接受写入，所以随意使用 fsync 和锁备份，也不会影响到主服务器的读写，应用程序可以正常工作。

23.5 副本集管理

1. 诊断

副本集中的机子，可以使用 db.printReplicationInfo()查看主数据库的复制的状态。

输入如下：

```
configured oplog size:   192MB
log length start to end: 65422secs (18.17hrs)
oplog first event time:  Sun Apr 9 2017 17:47:18 GMT-0400 (EDT)
oplog last event time:   Mon Apr 10 2017 11:57:40 GMT-0400 (EDT)
now:                     Mon Apr 10 2017 14:24:39 GMT-0400 (EDT)
```

输出信息包括了 oplog 的大小和操作的时间范围。这里 oplog 的大小大约是 192MB，可以放置 18.17 小时的操作。日志的长度 log length 是根据 oplog 最早操作时间和最后的操作时间差值得到的。如果刚启动服务器，最早操作会比较新，这时，日志的长度就会很小，即便 oplog 可能还有空闲空间也是如此。所以说，才启动的副本集 log length 是不准确的，需要等服务器跑了一段时间后，日志已经转了几个来回，这时 log length 才能准确地度量记录的时间。

使用 db.printSlaveReplicationInfo()查看从数据库的复制的状态。

输入如下：

```
source: 192.168.199.9:27017
    syncedTo: Mon Apr 10 2017 14:24:27 GMT-0400 (EDT)
    12 secs (0 hrs) behind the primary

source: 192.168.199.10:27017
    syncedTo: Mon Apr 10 2017 14:24:39 GMT-0400 (EDT)
    0 secs (0 hrs) behind the primary
```

输出了从节点的数据源列表，其中有滞后时间，192.168.199.9 滞后了 12 秒，192.168.199.10 没有滞后。

2. oplog 设置和变更 oplog 大小

副本集是使用 oplog 在进行交互的，oplog 是一个固定集合，所以需要密切留意 oplog 大小是否足够完成一次完整的重新同步。否则如果数据写入非常多非常快时，oplog 中的日志有可能被顶掉，从节点想要同步时会发现找不到切入的 oplog 的点了，已经跟不上主节点了，从节点就会停止同步。

怎么判断 oplog 大小是否足够完成一次完整的重新同步呢？就是查看 log length start to end 参数，我们之前是 65422 秒（18.17 小时），还可以放置 18.17 小时的操作。如果这个 log length 变成了 30 秒，已经很接近 192.168.199.9 滞后的 12 秒了，说明就必须要增加 oplog 的大小了，否则 192.168.199.9 有可能跟不上节奏。跟不上节奏停止后的从节点需要手动重新同步，这个同步是完整同步，非常消耗时间，手动重新同步使用命令：

```
{"resync":1}
```

重新同步代价高昂，所以要尽量避免，方法就是配置足够大的 oplog。为了避免从节点跟不上，一定要确保主节点的 oplog 足够大，能存放相当长时间的操作记录，但是大的 oplog 会占用更多的磁盘空间，所以读者需要自己权衡设置多少比较合适。在 64 位系统上，oplog 的默认大小是空余磁盘空间的 5%。如果发现 oplog 的大小不合适了，最简单的做法就是停掉主节点，删除 local 数据库的文件，用新的设置重新启动，使用--oplogSize 参数设置更大的 oplog 大小。

假设 MongoDB 数据目录是/data/db，关闭主节点的 mongod 服务后使用命令：

```
rm /data/db/local.*
mongod --oplog=8038 --master
```

--oplog=8038 把 oplog 设置为 8GB，重启主节点之后，所有的从节点得用--autoresync 参数重启，否则需要手动重新同步。

需要注意的是，太大的 oplog 预分配空间非常耗费时间，有经验的读者可以在数据库开启前，使用 linux 中的/dev/zero 命令手动预分配空间。在启动 MongoDB 时使用了参数--noprealloc 可以关闭空间预分配。

例如，我们要生成 20GB 的文件空间，使用命令：

```
cd /tmp/local
for i in {0..9}
do
echo $i
head -c 2146435072 /dev/zero > local.$i
done
```

然后关闭 MongoDB 主节点进行数据文件移动：

```
mv /data/db/local.*  /safe/place
mv /tmp/local/* /data/db/
```

先将原 local 文件夹中的数据备份到/safe/place 目录下，然后把我们预分配的空间文件/tmp/local/*移到数据目录/data/db/中，这样就完成了 oplog 的空间手动预分配。

重启主节点时就可以把 oplog 的大小设置成 20GB 了。使用命令如下：

```
mongod --master --oplogSize=20000
```

3. 阻塞复制使用

从节点的复制跟不上主节点的写入操作时，除了变更 oplog 的大小之外，还有一种方式能够解决，那就是阻塞主节点的写入，直到从节点慢慢跟上来之后再放开阻塞，详情可查看 6.2.4 小节“阻塞复制”。使用命令如下：

```
db.runCommand({getLastError:1,w:2});
```

w 的值表示包括主节点在内，至少 2 个服务器记录了写入操作之后才返回写入的结果。w 的值可以修改，值越大阻塞越明显，写操作越慢。它的原理是安全写入机制，读者可以参考 22.4 节安全写入数据。

23.6 副本集回滚丢失的数据

使用副本集是提高系统可靠性及易维护的有效途径。这样的话，弄清节点间故障的发生及转移机制就变得至关重要。

副本集中的成员一般通过 oplog（记录了数据中发生增、删、改等操作的列表）来传递信息，当其中一个成员发生变化修改 oplog 后，其他的成员也将按照 oplog 来执行。如果你负责处理新数据的节点在出错后恢复运行，它将会被回滚至最后一个 oplog 公共点。然而在这个过程中：丢失的“新数据”已经被 MongoDB 从数据库中转移并存放到你的数据目录 rollback 里面等待被手动恢复。如果你不知道这个特性，你可能就会认为数据被弄丢了。所以每当有成员从出错中恢复过来，都必须要检查这个目录。而通过 MongoDB 发布的标准工具来恢复这些数据是件很容易的事情。

总结：故障恢复中丢失的数据将会出现在 rollback 目录里面。

更多回滚数据恢复的信息可查看官网链接：

```
https://docs.mongodb.com/manual/core/replica-set-rollbacks/#replica-set-
rollbacks
```

23.7 分片的管理

我们在第 7 章了解 MongoDB 分片、第 18 章分片部署中已经学习了分片的理论和实践，但是在日常工作中我们还有一些使用分片的注意事项。

1. 避免分片太迟

分片能把数据拆分到多台服务器上，通常用于单例服务器运行过慢时进行性能提升。MongoDB 支持自动分片。在日常工作中我们要提前考虑什么时候要进行分片，因为对数据的拆分和块的迁移需要时间和服务器资源（空间内存等），所以如果当服务器资源基本上耗尽时，很可能会导致在你最需要分片时却分不了片。

解决的方法很简单，使用一个工具对 MongoDB 进行监控。对 MongoDB 的服务器做最准确的评估，并且在占整体性能的 80%前进行分片。相关监控工具可参考 12.4 节使用第三方插件监控。

如果你确定从一开始就要分片处理，那么更好的建议是选用 AWS 或者阿里云服务器进行分片。

总结：尽早地分片才能有效地避免问题。

2. 片键大小和集合分片大小限制

MongoDB 规定片键 shard key 的大小不能超过 512bytes。

MongoDB 对进行分片的集合大小也有限制，随着版本的更新这个限制从 256GB 变成 512GB，目前 3.4 版本这个限制需要根据 shard key 的平均大小和设置的块的大小（chunkSize）来计算。每个 BSON 文档的大小限制是 16MB，约等于 16777216 bytes，如图 23-1 所示。

Average Size of Shard Key Values	512 bytes	256 bytes	128 bytes	64 bytes
Maximum Number of Splits	32,768	65,536	131,072	262,144
Max Collection Size (64 MB Chunk Size)	1 TB	2 TB	4 TB	8 TB
Max Collection Size (128 MB Chunk Size)	2 TB	4 TB	8 TB	16 TB
Max Collection Size (256 MB Chunk Size)	4 TB	8 TB	16 TB	32 TB

图 23-1　分片集合大小限制

分片大小限制计算公式如下：

```
maxSplits = 16777216 (bytes) / <average size of shard key values in bytes>
```

```
maxCollectionSize (MB) = maxSplits * (chunkSize / 2)
```

因为这个限制，我们应该尽早地进行分片，读者应该在集合数据量未达到限制前进行分片。

更多分片的大小限制详情可查看官网链接：

```
https://docs.mongodb.com/manual/reference/limits/#sharded-clusters
```

3. 选择正确的 shard key

MongDB 需要你选择一个 shard key 来将数据分片，如果选择了错误的 shard key，更改起来将是件很麻烦的事情。

关于 shard key 的选择可以参考 7.2.1 小节数据分流中的三种分片方式。

更多片键的信息可参考官网链接：

```
https://docs.mongodb.com/manual/sharding/#sharding-internals-shard-keys
```

4. 修改 shard key

对于分片设置，shard key 是 MongoDB 用来识别分块对应文件的凭证。当你插入一个文件后，你就不可以再对文件的 shard key 进行更改了，MongoDB 没有提供修改 shard key 的支持。如果非要修改的话，先把数据备份，再删除，然后重新建立 shard key，最后把数据恢复回来。这样就允许把它指定到对应的分块了。

总结：shard key 不可以修改，必要的时候可以删除文件重新建立。

5. 为每个分片部署足够的复制集成员

分片之间的数据互相不复制，每个分片的数据必须在分片内保证高可用。因此，建议对每一个分片至少部署 3 个数据节点副本集，以保证该分片在绝大部分时间都不会因为主节点宕机而造成数据不可用。

23.8 MongoDB 锁

（1）MongoDB 使用的锁

MongoDB 允许多个客户端读写同一部分的数据，为了保证数据的一致性，MongoDB 使用锁和并发控制等机制，防止同一部分数据被几个客户端同时修改。

这些机制确保这部分数据只能在一个客户端中被修改，所有的客户端不会读取到不一致的数据。

（2）锁的粒度

早期 MongoDB 只能提供库级粒度锁，这意味着当 MongoDB 一个写锁处于占用状态时，其他的读写操作都只能等待。

随着 MongoDB 版本的更新，目前 MongoDB 3.4 版本提供了多种粒度的 lock 机制：Global（全局）、Database（数据库）、Collection（集合）级别。在 WiredTiger 引擎中还支持 Document（文档）级别的锁。

每个类型的锁分为 read（读）和 write（写）锁，其中 read 为共享锁（S）、write 为排它锁（X）、意向共享锁（IS）和意向排它锁（IX）。

意向表示 read 或者 write 操作想要获取更细粒度的资源。

（3）锁的兼容

在 MongoDB 中，资源的锁定级别（或次序）依次为：Global→Database→Collection→Document，粒度逐渐变小，并发能力依次更强。 如果想获取 Collection 的 write 锁（X，排它锁），那么必须依次在 Global 和相应的 Database 上获取意向排它锁（IX），如果这两个级别上的 IX 锁获取成功，才能在 Collection 上尝试获取 X 锁。对于同一个数据库，可以同时被 IS、IX 两种模式锁定，但是 X 锁不能与其他模式兼容，S 锁只能与 IS 模式兼容。关于意向锁的兼容性，如图 23-2 所示。

Mode	NL	IS	IX	S	SIX	X
NL	Yes	Yes	Yes	Yes	Yes	Yes
IS	Yes	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	Yes	No	No	No
S	Yes	Yes	No	Yes	No	No
SIX	Yes	Yes	No	No	No	No
X	Yes	No	No	No	No	No

图 23-2　锁的兼容

（4）锁的执行顺序

因为锁的颗粒度有多个级别，通过意向锁对资源访问路径进行标记，对于解决锁冲突是非常必要的。锁的获取是公平的，如果锁被占用，那么获取锁的请求将被队列化，不过为了优化吞吐能力，当一个锁请求被准许，那么同时其他相容的锁请求也会一并被准许。例如，当 X 锁释放时，锁队列中有如下请求：

```
IS->IS->X->X->S->IS->入口
```

按照严格意义的 FIFO（公平锁）顺序，只有开头的 IS、S 两个请求被准许（相容）。不过 MongoDB 为了优化并发能力，将会把队列中所有与 IS 相容的请求全部准入（并从队列中移除），即 IS、S、IS、IS。相容锁的处理有计数器，当上述 4 个锁请求全部释放 lock 以后，计数器变为 0，此时即使队列中又有了新的 IS 或者 S 锁请求，但不会再次准许它们，而是开始检测队列的头部，首个请求为 X，不过此锁是排它锁，所以只能逐个执行。如此循环执行。

（5）并发控制

MongoDB 对 WiredTiger 引擎使用了乐观的并发控制，在 Global、Database 和 Collection 级别，只使用意向锁。当存储引擎检测到两个操作有锁冲突时，其中一个操作将会透明地重试（CAS 方式，tryLock），也就是会自动重试直到执行成功。对于有些特殊的操作，仍然会使用排它锁，比如删除 collection 仍会在 Database 上获取 X 锁。乐观的并发控制主要是针对 documents 数据的操作。

对于 MMAPv1 引擎，在 3.0 之后即支持 collection 级别的锁，此前只支持 Database 级别，因此新版本的并发能力将会提升很多。比如一个 Database 中有多个 Collection，那么这些 Collection 可以同时接收 write、read 请求。不过 Database 上如果有排它锁会阻止 Collections 上的读写操作。

所以我们在 MongoDB 的版本选择上尽量选择新版本的 MongoDB，使用最新的引擎，可以提供更好的并发性能。

（6）读写锁的让步 yield

在某些情况下，read 和 write 操作会让步锁。比如一些执行耗时的 update、delete 或者 query，操作执行一段时间或者每隔一定数量的 Documents（例如 100 条）之后会 yield 一次，允许其他操作获取 lock 并执行，那么当前操作重新尝试获取锁（锁请求进入队列尾部）。yield 操作用于减轻锁请求的饥饿程度。

对于某些特殊的操作，即使执行的时间较长，但是为了避免并发问题，仍然不会 yield，比如 index 索引文件的加载和刷新等。

（7）分片的并发

sharding 通过将 collections 数据分布在多个 mongod 实例上提高集群整体的并发能力和吞吐量，允许每个 Servers（比如 mongos、mongod）并行地执行读写操作。Lock 被应用在每个单独的 shard 节点上，而不是在整个 Cluster 上，即每个 mongod 单独维护各自的 locks，在 mongod 上的操作不会干扰其他实例上的 lock。

（8）副本集的并发

对于副本集，所有的 write 操作均首先在主节点上执行，操作也会被写入到 local 数据库的 oplog 中用于其他 secondary 同步，因此每个 write 操作将会同时锁定 collection 的数据库以及 local 数据库，以保证数据的一致性。注意此时 lock 发生在主节点上。

在副本集架构中，MongoDB 不会直接在 secondary 上执行 write 操作，多个 secondary 并行地同步（批量）主节点中的 oplog，然后在各自的本地重现这些操作。在应用程序对主节点 write 操作时，secondary 不允许对主节点的 oplong 进行 read 操作执行，secondary 严格根据同步好的 oplog 顺序对自己执行 write 操作（同步或者回滚）。

（9）哪些操作会对数据库产生锁

常见数据库操作产生的锁如表 23-1 所示。

表 23-1 常见数据库操作产生的锁

操作	锁的类型
查询	读锁
游标	读锁
写入	写锁
删除	写锁
更新	写锁
Map-reduce	读写锁
创建索引	前台模式创建地：库级锁
Db.eval()	写锁，版本 3.0 后不建议使用，db.eval()在执行 JavaScript 脚本时，会造成全局锁，如果要避免全局锁，可以使用参数 nolock: true
eval	写锁，版本 3.0 后不建议使用，db.eval()在执行 JavaScript 脚本时，会造成全局锁，如果要避免全局锁，可以使用参数 nolock: true
aggregate()	读锁

（10）造成库级锁的 admin 命令

某些数据库管理操作会使用排它锁锁住数据库，以下命令需要申请排它锁，并锁定一段时间：

```
db.collection.ensureIndex();
reIndex;
compact;
db.repairDatabase();
db.createCollection();  当创建一个很大的固定集合时,空间预分配会花较长时间
db.collection.validate();
db.copyDatabase(); 这个命令会锁住所有的库
```

以下命令需要申请排它锁，锁定数据库，但锁定很短时间。

```
db.collection.dropIndex();
db.collection.getLastError();
db.isMaster();
rs.status();
db.serverStatus();
db.auth();
db.addUser();
```

（11）会锁住多个库的操作

db.copyDatabase()锁定整个 mongod 实例。

db.repairDatabase()会获取全局写锁，运行期间会阻塞其他操作。

journaling 内部操作，短时间锁定所有数据库，所有的数据库共享一个 journal。

查看锁的情况，使用如下命令：

```
db.serverStatus()
```

```
db.currentOp()
```

或者 MongoDB 提供的监控工具 mongotop、mongostat。

还可以查看 locks 集合或者使用第三方工具查看锁。

（12）解锁

使用 db.currentOp()找出有锁状态的执行操作，找到 opid。根据之前获取的 opid，使用 db.killOp(opid)来 kill 掉对应的操作。

更多关于锁的信息参考如下链接：

```
https://docs.mongodb.com/manual/faq/concurrency/

https://en.wikipedia.org/wiki/Multiple_granularity_locking
```

附录 A

MongoDB地理位置距离单位

MongoDB 查询地理位置默认有 3 种距离单位：

- 米（meters）。
- 平面单位（flat units，可以理解为经纬度的“一度”）。
- 弧度（radians）。

如果坐标以普通坐标对的格式保存，在不同的查询方式中默认的单位不同，如表 A-1 所示。

表 A-1 不同的查询方式中默认的单位不同

查询命令	距离单位	说明
$near	度	
$nearSphere	弧度	
$center	度	
$centerSphere	弧度	
$polygon	度	
$geoNear	度或弧度	指定参数 spherical 为 true 则为弧度，否则为度

如果坐标以 GeoJSON 格式，则单位都为米。

坐标以普通坐标对的格式保存的情况下：关于距离计算，MongoDB 的官方文档仅仅提到了弧度计算，未说明平面单位（度）计算。

关于弧度计算，官方文档的说明是：

```
To convert: distance to radians: divide the distance by the radius of the
sphere (e.g. the Earth) in the same units as the distance measurement. radians
to distance: multiply the radian measure by the radius of the sphere (e.g. the
Earth) in the units system that you want to convert the distance to.

The radius of the Earth is approximately 3,959 miles or 6,371 kilometers.
```

所以如果用弧度查询，则以千米数除以 6371，如“附近 200 米的餐厅”：

```
> db.runCommand( { geoNear: "places", near: [ 21.4905, 31.2646 ], spherical:
true,
```

```
$maxDistance: 0.2/6371 })
```

那如果不用弧度，以平面单位（度）查询时，距离单位如何处理？

答案是以千米数除以 111（推荐值），原因如下：

经纬度的一度，分为经度一度和纬度一度。

地球不同纬度之间的距离是一样的，地球子午线（南极到北极的连线）长度 39940.67 千米，纬度一度大约 110.9 千米。

但是不同纬度的经度一度对应的长度是不一样的：

在地球赤道，一圈大约为 40075 千米，除以 360 度，每一个经度大概是：40075/360=111.32 千米。

成都，大概在北纬 30.67 度，对应一个经度的长度是：40075*sin(90-31)/360=95.41 千米。

北京在北纬 40 度，对应的是 85 千米。

前面提到的参数 111，这个值是估算值，不完全准确，任意两点之间的距离，平均纬度越大，这个参数则误差越大。

所以“度”这个单位只用于平面，由于地球是圆的，在大范围使用时会有误差。

官方建议使用 sphere 查询方式，也就是说距离单位用弧度。

```
The current implementation assumes an idealized model of a flat earth, meaning
that an arcdegree of latitude (y) and longitude (x) represent the same
distance everywhere. This is only true at the equator where they are both
about equal to 69 miles or 111km. However, at the 10gen offices at { x : -74 ,
y : 40.74 } one arcdegree of longitude is about 52 miles or 83 km (latitude is
unchanged). This means that something 1 mile to the north would seem closer
than something 1 mile to the east.
```

$geoNear 返回的距离值 dis，如果指定了 spherical 为 true， dis 的值为弧度，不指定则为度。

指定 spherical 为 true，结果中的 dis 需要乘以 6371 换算为千米，因为地球的半径约为 6371 千米，所以 1 弧度约为 6371 千米。

坐标以 GeoJSON 格式保存的情况下，单位是米，不需要转化。

所以一般情况下建议使用 GeoJSON 格式保存地理位置信息。

附录 B

◂ 相关网址 ▸

1. MongoDB 的官方网址

https://www.mongodb.com/

2. 数据库知识网站 DB-Engines

https://db-engines.com/en/ranking

3. 谁在用 MongoDB

https://www.mongodb.com/who-uses-mongodb

4. MongoDB 数据类型

https://docs.mongodb.com/manual/reference/bson-types/

5. MongoDB3.4 版本的发布日志

https://docs.mongodb.com/manual/release-notes/3.4/

6. MongoDB 官方下载地址

http://www.mongodb.org/downloads

7. VMware Workstation 11 下载地址

https://my.vmware.com/cn/web/vmware/details?productId=462&rPId=11036&downloadGroup=WKST-1110-WIN

8. CentOS 6.4 下载地址

http://vault.centos.org/6.4/isos/x86_64/

9. CentOS 6.4 官网

https://www.centos.org/

10. MongoDB 安装步骤 Linux 版本

https://docs.mongodb.com/master/administration/install-on-linux/

11. MongoDB 仓库列表

https://repo.mongodb.org

12. MongoDB 的公钥 Key 查看

https://docs.mongodb.com/master/tutorial/install-mongodb-on-ubuntu/

13. openssl 更新包

http://mirrors.163.com/centos/6/os/x86_64/Packages/

14. MongoDB 安装步骤 MacOS 版本
http://docs.mongodb.org/manual/tutorial/install-mongodb-on-os-x/

15. MongoDB 启动配置参数
http://docs.mongodb.org/manual/reference/configuration-options/

16. 聚合运算 count
https://docs.mongodb.com/manual/reference/command/count/

17. 聚合运算 distinct
https://docs.mongodb.com/manual/reference/command/distinct/

18. 聚合运算 group
https://docs.mongodb.com/manual/reference/command/group/

19. 管道操作器
https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/

20. GeoJSON 信息
https://docs.mongodb.com/manual/reference/geojson/

21. 2d 索引
https://docs.mongodb.com/manual/core/2d/

22. 2dsphere 索引
https://docs.mongodb.com/manual/core/2dsphere/

23. 地理位置操作
https://docs.mongodb.com/manual/reference/operator/query-geospatial/

24. aggregate 管道表达式
https://docs.mongodb.com/manual/reference/operator/aggregation/#expression-operators

25. GUI 工具
https://docs.mongodb.com/ecosystem/tools/administration-interfaces/

26. Robomongo 官网
https://robomongo.org/

27. serverStatus 命令
https://docs.mongodb.com/manual/reference/command/serverStatus/

28. 用户认证
https://docs.mongodb.com/manual/tutorial/enable-authentication/

29. 用户权限参数
https://docs.mongodb.com/manual/reference/built-in-roles/

https://docs.mongodb.com/manual/tutorial/manage-users-and-roles/#view-a-role-s-privileges

30. TIOBE 官网

https://www.tiobe.com/tiobe-index/

31. MongoDB 驱动

https://docs.mongodb.com/ecosystem/drivers/

32. JDK 下载地址

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

33. Eclipse 官网

http://www.eclipse.org/downloads/

34. 第三方 jar 包管理网站

http://search.maven.org

35. 驱动操作示例

https://github.com/mongodb/specifications/blob/master/source/crud/examples/java/src/main/java/examples/MongoCollectionUsageExample.java

36. Tomcat 官网

http://tomcat.apache.org/

37. MongoDB 数据结构

https://docs.mongodb.com/manual/core/data-modeling-introduction/

38. MapReduce 与 Hadoop

https://docs.mongodb.com/ecosystem/use-cases/hadoop/

https://docs.mongodb.com/ecosystem/tools/hadoop/

39. 写关注

https://docs.mongodb.com/v2.4/core/write-concern/

40. 调优工具 profiler

https://docs.mongodb.com/manual/tutorial/manage-the-database-profiler/

41. 防火墙设置

https://docs.mongodb.com/manual/tutorial/configure-linux-iptables-firewall/

42. SSL 配置

https://docs.mongodb.com/manual/tutorial/configure-ssl/

43. 关闭 NUMA

https://docs.mongodb.com/manual/administration/production-notes/#production-numa

44. 回滚数据恢复

https://docs.mongodb.com/manual/core/replica-set-rollbacks/#replica-set-rollbacks

45. 分片的大小限制

https://docs.mongodb.com/manual/reference/limits/#sharded-clusters

46. 片键

https://docs.mongodb.com/manual/sharding/#sharding-internals-shard-keys

47. 锁

https://docs.mongodb.com/manual/faq/concurrency/

MongoDB游记

之轻松入门到进阶

作者简介

张泽泉，毕业于四川理工学院，数据工程师、中级职称软件设计师、CSDN博客专家。致力于数据采集、数据分析、数据分布式运算架构等技术的应用与研究。多年一线MongoDB数据库存储、部署、开发经验，以及将其应用于房地产数据分析、金融数据分析、基因数据分析等领域行业经验。

清华社官方微信号

扫 我 有 惊 喜

ISBN 978-7-302-47860-7

9 787302 478607 >

定价：59.00元