redis事务及乐观锁

## 概念

[MULTI](http://redis.readthedocs.org/en/latest/transaction/multi.html#multi) 、 [EXEC](http://redis.readthedocs.org/en/latest/transaction/exec.html#exec) 、[DISCARD](http://redis.readthedocs.org/en/latest/transaction/discard.html#discard) 和 [WATCH](http://redis.readthedocs.org/en/latest/transaction/watch.html#watch) 是Redis事务的基础。

事务可以一次执行多个命令， 并且带有以下两个重要的保证：

* 事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。
* 事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。EXEC命令负责触发并执行事务中的所有命令：
  + 如果客户端在使用MULTI开启了一个事务之后，却因为断线而没有成功执行 EXEC ，那么事务中的所有命令都不会被执行。
  + 另一方面，如果客户端成功在开启事务之后执行EXEC，那么事务中的所有命令都会被执行。

当使用AOF方式做持久化的时候， Redis会使用单个write(2)命令将事务写入到磁盘中。然而，如果Redis服务器因为某些原因被管理员杀死，或者遇上某种硬件故障，那么可能只有部分事务命令会被成功写入到磁盘中。如果 Redis 在重新启动时发现 AOF 文件出了这样的问题，那么它会退出，并汇报一个错误。使用 redis-check-aof 程序可以修复这一问题：它会移除 AOF 文件中不完整事务的信息，确保服务器可以顺利启动。

从 2.2 版本开始，Redis 还可以通过乐观锁（optimistic lock）实现 CAS （check-and-set）操作，具体信息请参考文档的后半部分。

## 用法

[MULTI](http://redis.readthedocs.org/en/latest/transaction/multi.html#multi) 命令用于开启一个事务，它总是返回 OK 。

[MULTI](http://redis.readthedocs.org/en/latest/transaction/multi.html#multi) 执行之后， 客户端可以继续向服务器发送任意多条命令， 这些命令不会立即被执行， 而是被放到一个队列中， 当 [EXEC](http://redis.readthedocs.org/en/latest/transaction/exec.html#exec) 命令被调用时， 所有队列中的命令才会被执行。

另一方面， 通过调用 [DISCARD](http://redis.readthedocs.org/en/latest/transaction/discard.html#discard) ， 客户端可以清空事务队列， 并放弃执行事务。

以下是一个事务例子， 它原子地增加了 foo 和 bar 两个键的值：

|  |
| --- |
| > MULTI  OK  > INCR foo  QUEUED  > INCR bar  QUEUED  > EXEC  1) (integer) 1  2) (integer) 1 |

[EXEC](http://redis.readthedocs.org/en/latest/transaction/exec.html#exec) 命令的回复是一个数组， 数组中的每个元素都是执行事务中的命令所产生的回复。 其中， 回复元素的先后顺序和命令发送的先后顺序一致。

当客户端处于事务状态时， 所有传入的命令都会返回一个内容为 QUEUED 的状态回复（status reply）， 这些被入队的命令将在 [EXEC](http://redis.readthedocs.org/en/latest/transaction/exec.html#exec)命令被调用时执行。

## 事务中的错误

使用事务时可能会遇上以下两种错误：

* 事务在执行EXEC之前，入队的命令可能会出错。比如说，命令可能会产生语法错误（参数数量错误，参数名错误，等等），或者其他更严重的错误，比如内存不足（如果服务器使用maxmemory设置了最大内存限制的话）。
* 命令可能在 EXEC 调用之后失败。举个例子，事务中的命令可能处理了错误类型的键，比如将列表命令用在了字符串键上面，诸如此类。

对于发生在 [EXEC](http://redis.readthedocs.org/en/latest/transaction/exec.html#exec) 执行之前的错误，客户端以前的做法是检查命令入队所得的返回值：如果命令入队时返回 QUEUED ，那么入队成功；否则，就是入队失败。如果有命令在入队时失败，那么大部分客户端都会停止并取消这个事务。不过，从 Redis 2.6.5 开始，服务器会对命令入队失败的情况进行记录，并在客户端调用 [EXEC](http://redis.readthedocs.org/en/latest/transaction/exec.html#exec) 命令时，拒绝执行并自动放弃这个事务。在 Redis 2.6.5 以前， Redis 只执行事务中那些入队成功的命令，而忽略那些入队失败的命令。 而新的处理方式则使得在流水线（pipeline）中包含事务变得简单，因为发送事务和读取事务的回复都只需要和服务器进行一次通讯。至于那些在 [EXEC](http://redis.readthedocs.org/en/latest/transaction/exec.html#exec) 命令执行之后所产生的错误， 并没有对它们进行特别处理： 即使事务中有某个/某些命令在执行时产生了错误， 事务中的其他命令仍然会继续执行。从协议的角度来看这个问题，会更容易理解一些。 以下例子中， [LPOP](http://redis.readthedocs.org/en/latest/list/lpop.html#lpop) 命令的执行将出错， 尽管调用它的语法是正确的：

|  |
| --- |
| Trying 127.0.0.1...  Connected to localhost.  Escape character is '^]'.  MULTI  +OK  SET a 3  abc  +QUEUED  LPOP a  +QUEUED  EXEC  \*2  +OK  -ERR Operation against a key holding the wrong kind of value |

EXEC返回两条批量回复（bulk reply）： 第一条是OK，而第二条是-ERR。 至于怎样用合适的方法来表示事务中的错误， 则是由客户端自己决定的。

最重要的是记住这样一条，即使事务中有某条/某些命令执行失败了，事务队列中的其他命令仍然会继续执行 ——Redis不会停止执行事务中的命令。

以下例子展示的是另一种情况， 当命令在入队时产生错误， 错误会立即被返回给客户端：

|  |
| --- |
| MULTI  +OK  INCR a b c  -ERR wrong number of arguments for 'incr' command |

因为调用 INCR 命令的参数格式不正确， 所以这个 INCR 命令入队失败。

## 不支持回滚原因

如果你有使用关系式数据库的经验， 那么“Redis在事务失败时不进行回滚，而是继续执行余下的命令”这种做法可能会让你觉得有点奇怪。以下是这种做法的优点：

* Redis命令只会因为错误的语法而失败（并且这些问题不能在入队时发现），或是命令用在了错误类型的键上面：这也就是说，从实用性的角度来说，失败的命令是由编程错误造成的，而这些错误应该在开发的过程中被发现，而不应该出现在生产环境中。
* 因为不需要对回滚进行支持，所以Redis的内部可以保持简单且快速。

有种观点认为Redis处理事务的做法会产生bug， 然而需要注意的是，在通常情况下， 回滚并不能解决编程错误带来的问题。举个例子，如果你本来想通过INCR命令将键的值加上1，却不小心加上了2，又或者对错误类型的键执行了INCR，回滚是没有办法处理这些情况的。鉴于没有任何机制能避免程序员自己造成的错误，并且这类错误通常不会在生产环境中出现，所以Redis 选择了更简单、更快速的无回滚方式来处理事务。

## 放弃事务

当执行DISCARD命令时，事务会被放弃，事务队列会被清空，并且客户端会从事务状态中退出：

|  |
| --- |
| redis> SET foo 1  OK  redis> MULTI  OK  redis> INCR foo  QUEUED  redis> DISCARD  OK  redis> GET foo  "1" |

## 乐观锁

WATCH命令可以为Redis事务提供 check-and-set（CAS）行为。

被WATCH的键会被监视，并会发觉这些键是否被改动过了。如果有至少一个被监视的键在EXEC执行之前被修改了，那么整个事务都会被取消， EXEC返回空多条批量回复（null multi-bulk reply）来表示事务已经失败。

举个例子，假设我们需要原子性地为某个值进行增1操作（假设INCR不存在）。首先我们可能会这样做：

|  |
| --- |
| val = GET mykey  val = val + 1  SET mykey $val |

上面的这个实现在只有一个客户端的时候可以执行得很好。但是当多个客户端同时对同一个键进行这样的操作时，就会产生竞争条件。举个例子，如果客户端A和B都读取了键原来的值， 比如10，那么两个客户端都会将键的值设为11，但正确的结果应该是12才对。有了WATCH，我们就可以轻松地解决这类问题了：

|  |
| --- |
| WATCH mykey  val = GET mykey  val = val + 1  MULTI  SET mykey $val  EXEC |

使用上面的代码， 如果在WATCH执行之后， EXEC执行之前，有其他客户端修改了mykey 的值，那么当前客户端的事务就会失败。程序需要做的，就是不断重试这个操作，直到没有发生碰撞为止。这种形式的锁被称作乐观锁，它是一种非常强大的锁机制。并且因为大多数情况下，不同的客户端会访问不同的键，碰撞的情况一般都很少，所以通常并不需要进行重试。

## WATCH命令

WATCH使得EXEC命令需要有条件地执行：事务只能在所有被监视键都没有被修改的前提下执行，如果这个前提不能满足的话，事务就不会被执行。如果你使用WATCH监视了一个带过期时间的键，那么即使这个键过期了，事务仍然可以正常执行，关于这方面的详细情况，请看这个帖子：<http://code.google.com/p/redis/issues/detail?id=270>

WATCH命令可以被调用多次。对键的监视从WATCH执行之后开始生效， 直到调用 EXEC 为止。用户还可以在单个WATCH命令中监视任意多个键，就像这样：

|  |
| --- |
| redis> WATCH key1 key2 key3  OK |

当EXEC被调用时，不管事务是否成功执行，对所有键的监视都会被取消。另外当客户端断开连接时，该客户端对键的监视也会被取消。

使用无参数的UNWATCH 命令可以手动取消对所有键的监视。对于一些需要改动多个键的事务，有时候程序需要同时对多个键进行加锁，然后检查这些键的当前值是否符合程序的要求。当值达不到要求时，就可以使用UNWATCH命令来取消目前对键的监视，中途放弃这个事务，并等待事务的下次尝试。

## 附录

<https://blog.csdn.net/truong/article/details/47069631>

<https://www.zhihu.com/question/29397176>

http://www.roncoo.com/article/detail/130426