# Redis 高级应用 -- 事务和分布式锁

作者：GaoWentian
原文链接：https://ld246.com/article/1576203775109
来源网站：链滴

## 简单用法

Redis使用MULTI、EXEC、DISCARD和WATCH等命令实现事务。下面是Redis事务的用法，使用MULTI命令开始后，Redis会判断输入的命令是否是MULTI、EXEC、DISCARD和WATCH中的一个，如果，则执行命令，否则会将命令保存在队列中，最后执行EXEC命令提交事务。Redis执行事务期间，服器不会去执行其他命令，等事务中所有命令执行完毕才会处理其他请求。

```
127.0.0.1:6379> MULTI  # 事务开始
OK
127.0.0.1:6379> SET name "11231" #命令入队
QUEUED
127.0.0.1:6379> GET name  #命令入队
QUEUED
127.0.0.1:6379> EXEC  #事务执行
1) OK
2) "11231"
```

Redis提供DISCARD命令取消事务：

```
127.0.0.1:6379> SET name "123"
OK
127.0.0.1:6379> GET name
"123"
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> DEL name
QUEUED
127.0.0.1:6379> DISCARD   # 取消事务
OK
127.0.0.1:6379> GET name
"123"
```

如果输入的命令语法错误，会直接报错.但是Redis无法判断输入的指令是否存在逻辑错误。例如下面例子，Redis在事务执行前可以判断出来"XPUSH"是不合法的命令，但是无法判断"name"是字符串类，而不是列表型，只有执行后才报错。

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> XPUSH 123 123
(error) ERR unknown command 'XPUSH'
127.0.0.1:6379> EXEC
(error) EXECABORT Transaction discarded because of previous errors.

127.0.0.1:6379> del name
(integer) 1
127.0.0.1:6379> SET name 123  # name是字符串类型
OK
127.0.0.1:6379> GET name
"123"
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> LPUSH name 123 321  # 将name作为列表型保存
QUEUED
```

```
127.0.0.1:6379> EXEC
1) (error) WRONGTYPE Operation against a key holding the wrong kind of value
```

## WATCH命令

WATCH命令是一个乐观锁，它可以在EXEC命令执行之前，监视任意数量的key，并在EXEC命令执行，检查被监视的键是否至少有一个已经修改，如果是的话，服务器拒绝执行事务，并返回空恢复。

```
127.0.0.1:6379> WATCH name  # 开始监视name键
OK
127.0.0.1:6379> SET name 321 # 这个命令修改了name的值，这里模拟的是其他客户端修改name值
OK
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> SET haha 321  # 这里可以执行任意指令，因为watch监视的键，和事务中操作的没有关联。
QUEUED
127.0.0.1:6379> EXEC # EXEC命令执行前，name的值已经被修改，所以返回nil
(nil)
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> SET name 321
QUEUED
127.0.0.1:6379> EXEC # 第二次执行事务，WATCH的监视已经消失，虽然前面name已经被修改，里还可以操作name
1) OK
127.0.0.1:6379> GET name
"321"
```

还可以使用UNWATCH命令取消所有键的监视。

## 回滚

Redis的事务和关系型数据库事务最大的区别是没有事务失败后的回滚操作。如下例子，如果事务执失败，在失败命令之前的命令都会执行，并且无法回滚，需要手动回滚。

```
127.0.0.1:6379> KEYS *
(empty list or set)
127.0.0.1:6379> SET name gavin # name是字符串类型
OK
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> SET age 1
QUEUED
127.0.0.1:6379> LPUSH name 123 321 # name是字符串型，这里当成数组使用，会报错
QUEUED
127.0.0.1:6379> EXEC
1) OK
2) (error) WRONGTYPE Operation against a key holding the wrong kind of value
127.0.0.1:6379> KEYS *  #事务执行后，虽然报错，但是age已经被添加进去，没有回滚
1) "age"
2) "name"
```

```
127.0.0.1:6379>
```

## 分布式锁

分布式使用SETNX（set if not exists）设置公共锁。SETNX命令，可以判断key是否有值，有值返回置失败，无值则返回设置成功。

```
SETNX key value
```

- 对于返回设置成功的，拥有控制权，进行下一步的业务操作
- 对于返回设置失败的，不具有控制权，则排队或者等待

## 死锁

Redis分布式锁中，使用SETNX命令进行简单的上锁，如果上锁的机器因为一些原因掉线，将会一直用锁，造成死锁，因此需要一些特殊处理。参考资料中Redisson给出了一个很好的解决办法：

首先使用正常的方式加锁，再为myLock设置30秒的生存时间，避免发生死锁。

```
SETNX myLock 客户端id #设置客户端id，这样客户端在下次进入时判断哪个客户端占用锁，实现可入锁
pexpire myLock 30000
.....    # 执行业务操作
del myLock   # 执行完毕删除锁
```

然后客户端通过一个线程每10s就去判断客户端是否释放锁，如果客户端还持有锁，然后就延长锁的存时间。这样就保证了客户端存活时一直占有锁，掉线时则自动过期，让别的客户端取占有锁。当然实业务中不可能持有锁30秒，一般使用如下规则：

- 例如：持有锁的操作最长执行时间127ms，最短执行时间7ms
- 测试百万次最长执行时间对应命令的最大耗时，测试百万次网络延迟平均耗时
- 锁时间设定推荐：最大耗时 * 120% + 平均网络延迟* 110%
- 如果业务最大耗时<<网络平均延迟，通常为2个数量级，取其中单个耗时较长即可

## 参考资料

Redis分布式锁