**redis的事务**

严格意义来讲,redis的事务和我们理解的传统数据库(如mysql)的事务是不一样的。

**redis中的事务定义**

Redis中的事务（transaction）是一组命令的集合。

事务同命令一样都是Redis的最小执行单位，一个事务中的命令要么都执行，要么都不执行。  
事务的原理是先将属于一个事务的命令发送给Redis，然后再让Redis依次执行这些命令。

Redis保证一个事务中的所有命令要么都执行，要么都不执行。如果在发送EXEC命令前客户端断线了，则Redis会清空事务队列，事务中的所有命令都不会执行。而一旦客户端发送了EXEC命令，所有的命令就都会被执行，即使此后客户端断线也没关系，因为Redis中已经记录了所有要执行的命令。

除此之外，***Redis的事务还能保证一个事务内的命令依次执行而不被其他命令插入***。试想客户端A需要执行几条命令，同时客户端B发送了一条命令，如果不使用事务，则客户端B的命令可能会插入到客户端A的几条命令中执行。如果不希望发生这种情况，也可以使用事务。

**事务的应用**

事务的应用非常普遍，如银行转账过程中A给B汇款，首先系统从A的账户中将钱划走，然后向B的账户增加相应的金额。这两个步骤必须属于同一个事务，要么全执行，要么全不执行。否则只执行第一步，钱就凭空消失了，这显然让人无法接受。

**和传统的事务不同**

和传统的mysql事务不同的事，即使我们的加钱操作失败,我们也无法在这一组命令中让整个状态回滚到操作之前

**事务的错误处理**

如果一个事务中的某个命令执行出错，Redis会怎样处理呢？要回答这个问题，首先需要知道什么原因会导致命令执行出错。

**语法错误**

语法错误指命令不存在或者命令参数的个数不对。比如：

redis＞MULTI

OK

redis＞SET key value

QUEUED

redis＞SET key

(error)ERR wrong number of arguments for 'set' command

redis＞ errorCOMMAND key

(error) ERR unknown command 'errorCOMMAND'

redis＞ EXEC

(error) EXECABORT Transaction discarded because of previous errors.

跟在MULTI命令后执行了3个命令：一个是正确的命令，成功地加入事务队列；其余两个命令都有语法错误。而只要有一个命令有语法错误，执行EXEC命令后Redis就会直接返回错误，连语法正确的命令也不会执行。

**这里需要注意一点：**  
Redis 2.6.5之前的版本会忽略有语法错误的命令，然后执行事务中其他语法正确的命令。就此例而言，SET key value会被执行，EXEC命令会返回一个结果：1) OK。

**运行错误**

运行错误指在命令执行时出现的错误，比如使用散列类型的命令操作集合类型的键，这种错误在实际执行之前Redis是无法发现的，所以在事务里这样的命令是会被Redis接受并执行的。如果事务里的一条命令出现了运行错误，事务里其他的命令依然会继续执行（包括出错命令之后的命令），示例如下：

redis＞MULTI

OK

redis＞SET key 1

QUEUED

redis＞SADD key 2

QUEUED

redis＞SET key 3

QUEUED

redis＞EXEC

1) OK

2) (error) ERR Operation against a key holding the wrong kind of value

3) OK

redis＞GET key

"3"

可见虽然SADD key 2出现了错误，但是SET key 3依然执行了。

Redis的事务没有关系数据库事务提供的***回滚（rollback）***功能。为此开发者必须在事务执行出错后自己收拾剩下的摊子（将数据库复原回事务执行前的状态等,这里我们一般采取日志记录然后业务补偿的方式来处理，但是一般情况下，在redis做的操作不应该有这种强一致性要求的需求，我们认为这种需求为不合理的设计）。

**Watch命令**

大家可能知道redis提供了基于incr命令来操作一个整数型数值的原子递增，那么我们假设如果redis没有这个incr命令，我们该怎么实现这个incr的操作呢？

那么我们下面的正主***watch***就要上场了。

**如何使用watch命令**

正常情况下我们想要对一个整形数值做修改是这么做的(伪代码实现)：

val = GET mykey

val = val + 1

SET mykey $val

但是上述的代码会出现一个问题,因为上面吧正常的一个**incr(原子递增操作)**分为了两部分,那么在***多线程(分布式)***环境中，这个操作就有可能不再具有**原子性**了。

研究过java的juc包的人应该都知道**cas**，那么redis也提供了这样的一个机制，就是利用watch命令来实现的。

**watch命令描述**

***WATCH命令可以监控一个或多个键，一旦其中有一个键被修改（或删除），之后的事务就不会执行。监控一直持续到EXEC命令（事务中的命令是在EXEC之后才执行的，所以在MULTI命令后可以修改WATCH监控的键值）***

**利用watch实现incr**

具体做法如下:

WATCH mykey

val = GET mykey

val = val + 1

MULTI

SET mykey $val

EXEC

和此前代码不同的是，新代码在获取mykey的值之前先通过WATCH命令监控了该键，此后又将set命令包围在事务中，这样就可以有效的保证每个连接在执行EXEC之前，如果当前连接获取的mykey的值被其它连接的客户端修改，那么当前连接的EXEC命令将执行失败。这样调用者在判断返回值后就可以获悉val是否被重新设置成功。

***注意点***

由于WATCH命令的作用只是当被监控的键值被修改后阻止之后一个事务的执行，而不能保证其他客户端不修改这一键值，所以在一般的情况下我们需要在EXEC执行失败后重新执行整个函数。

执行EXEC命令后会取消对所有键的监控，如果不想执行事务中的命令也可以使用UNWATCH命令来取消监控。

**实现一个hsetNX函数**

我们实现的hsetNX这个功能是：***仅当字段存在时才赋值***。

为了避免竞态条件我们使用watch和**事务**来完成这一功能（伪代码）：

WATCH key

isFieldExists = HEXISTS key, field

if isFieldExists is 1

MULTI

HSET key, field, value

EXEC

else

UNWATCH

return isFieldExists

在代码中会判断要赋值的字段是否存在，如果字段不存在的话就不执行事务中的命令，但需要使用UNWATCH命令来保证下一个事务的执行不会受到影响。