面试竟被问到Redis事务，触及知识盲区，脸都绿了

程序员面试 7月4日

以下文章来源于非科班的科班，作者黎杜

前言

前几天有读者说自己面试被问到Redis的事务，虽然不常用，但是面试竟然被问到，平时自己没有注意Redis的事务这一块，面试的时候被问到非常不好受。

虽然，这位读者面试最后算是过了，但是薪资方面没有拿到自己理想的薪资。

其实这个也是正常的，一般面试被问到烂大街的，谁还问你啊，专门挑一些不常见的来问你，就是为了压你的薪资。

所以在这里写一篇文章对Redis的事务进行详细的讲解，估计对Redis事务从理解到原理深入这一篇就够了。

以后面试都不用担心了再被问道Redis的事务了，这一篇主要讲解Redis事务原理和实操的演练，理解理论的同时也通过实操来证实理论。

事务介绍

Redis事务是一组命令的集合，将多个命令进行打包，然后这些命令会被顺序的添加到队列中，并且按顺序的执行这些命令。

「Redis事务中没有像Mysql关系型数据库事务隔离级别的概念，不能保证原子性操作，也没有像Mysql那样执行事务失败会进行回滚操作」。

这个与Redis的特点：**「快速、高效」**有着密切的关联，**「因为一些列回滚操作、像事务隔离级别那这样加锁、解锁，是非常消耗性能的」**。所以，Redis中执行事务的流程只需要简单的下面三个步骤：

1. 开始事务（MULTI）
2. 命令入队
3. 执行事务（EXEC）、撤销事务（DISCARD ）

在Redis中事务的实现主要是通过如下的命令实现的：

命令	功能描述
MULTI	**「事务开始的命令」**，执行该命令后，后面执行的对Redis数据类型的**「操作命令都会顺序的放进队列中」**，等待执行EXEC命令后队列中的命令才会被执行
DISCARD	**「放弃执行队列中的命令」**，你可以理解为Mysql的回滚操作，**「并且将当前的状态从事务状态改为非事务状态」**。
EXEC	执行该命令后**「表示顺序执行队列中的命令」**，执行完后并将结果显示在客户端，**「将当前状态从事务状态改为非事务状态」**。若是执行该命令之前有key被执行WATCH命令并且又被其它客户端修改，那么就会放弃执行队列中的所有命令，在客户端显示报错信息，若是没有修改就会执行队列中的所有命令。
WATCH key	表示指定监视某个key，**「该命令只能在MULTI命令之前执行」**，如果监视的key被其他客户端修改，**「EXEC将会放弃执行队列中的所有命令」**
UNWATCH	**「取消监视之前通过WATCH 命令监视的key」**，通过执行EXEC 、DISCARD 两个命令之前监视的key也会被取消监视

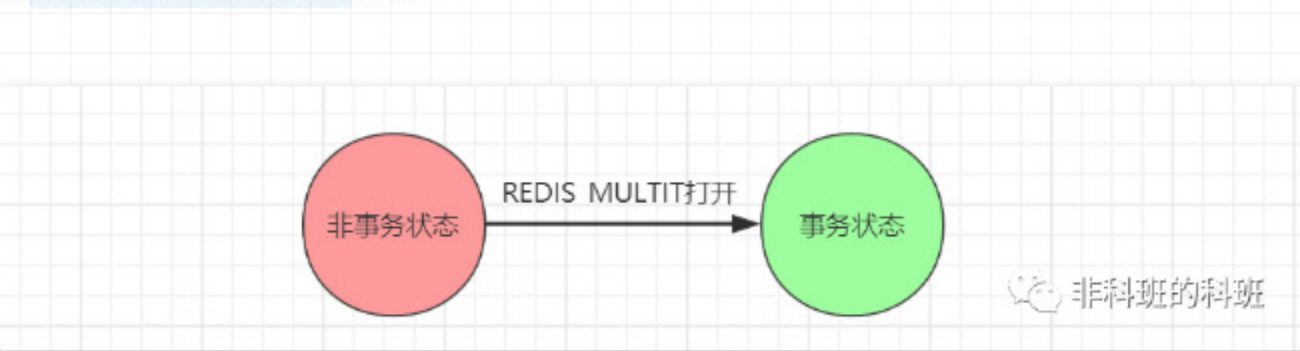
以上就是一个Redis事务的执行过程包含的命令，下面就来详细的围绕着这几个命令进行讲解。

开始事务

`MULTI` 命令表示事务的开始，当看到OK表示已经进入事务的状态：

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379>
```

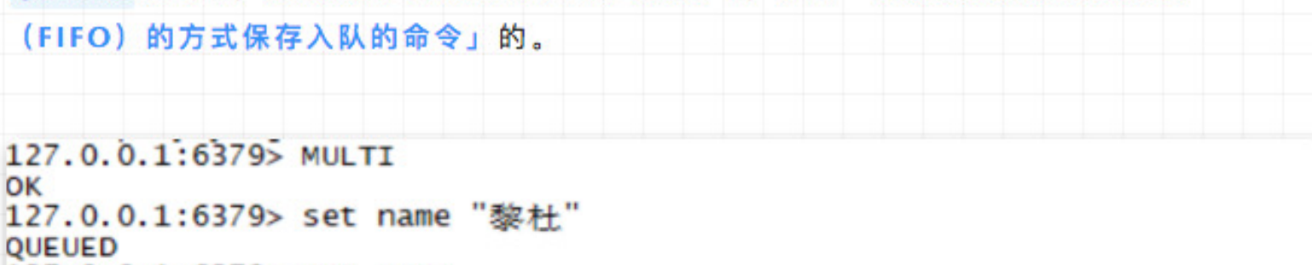
该命令执行后客户端会将**「当前的状态从非事务状态修改为事务状态」**，这一状态的切换是将客户端的 `flags` 属性中打开 `REDIS_MULTI` 来完成的，该命令可以理解关系型数据库Mysql的 `BEGIN TRANCATION` 语句：



命令入队

执行完MULTI命令后，后面执行的操作Redis五种类型的命令都会按顺序的进入命令队列中，该部分也是真正的业务逻辑的部分。

Redis客户端的命令执行后若是当前状态处于事务状态命令就会进入队列中，并且返回 `QUEUED` 字符串，表示该命令已经进入了命令队列中，并且**「事务队列是以先进先出（FIFO）的方式保存入队的命令」**的。

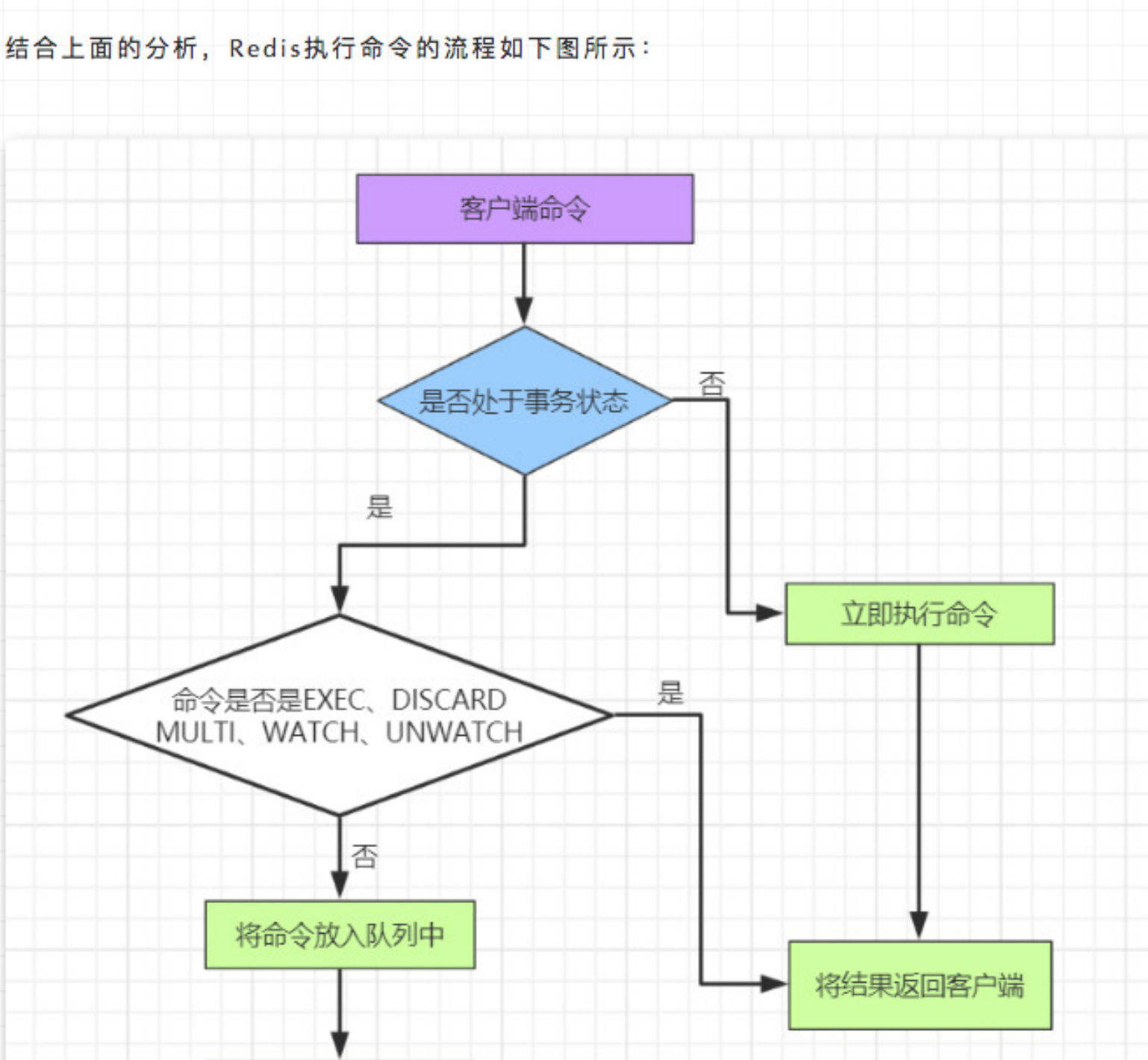


```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> set name "黎杜"
QUEUED
127.0.0.1:6379> get name
QUEUED
127.0.0.1:6379> █
```

若是当前状态是非事务状态就会立即执行命令，并将结果返回客户端。在事务状态**「执行操作事务的命令就会被立即执行」**，如 `EXEC、DISCARD、UNWATCH`。

```
127.0.0.1:6379> set age 24
OK
127.0.0.1:6379> get age
"24"
```

结合上面的分析，Redis执行命令的流程如下图所示：



事务的命令队列中有三个参数分别是：**「要执行的命令」**、**「命令的参数」**、**「参数的个数」**。例如：通过执行如下的命令：

```
redis> MULTI
OK
redis> SET name "黎杜"
QUEUED
redis> GET name
QUEUED
```

那么对应上面的队列中三个参数如下表格所示：

执行的命令	命令的参数	参数的个数
SET	["name", "黎杜"]	2
GET	["name"]	1

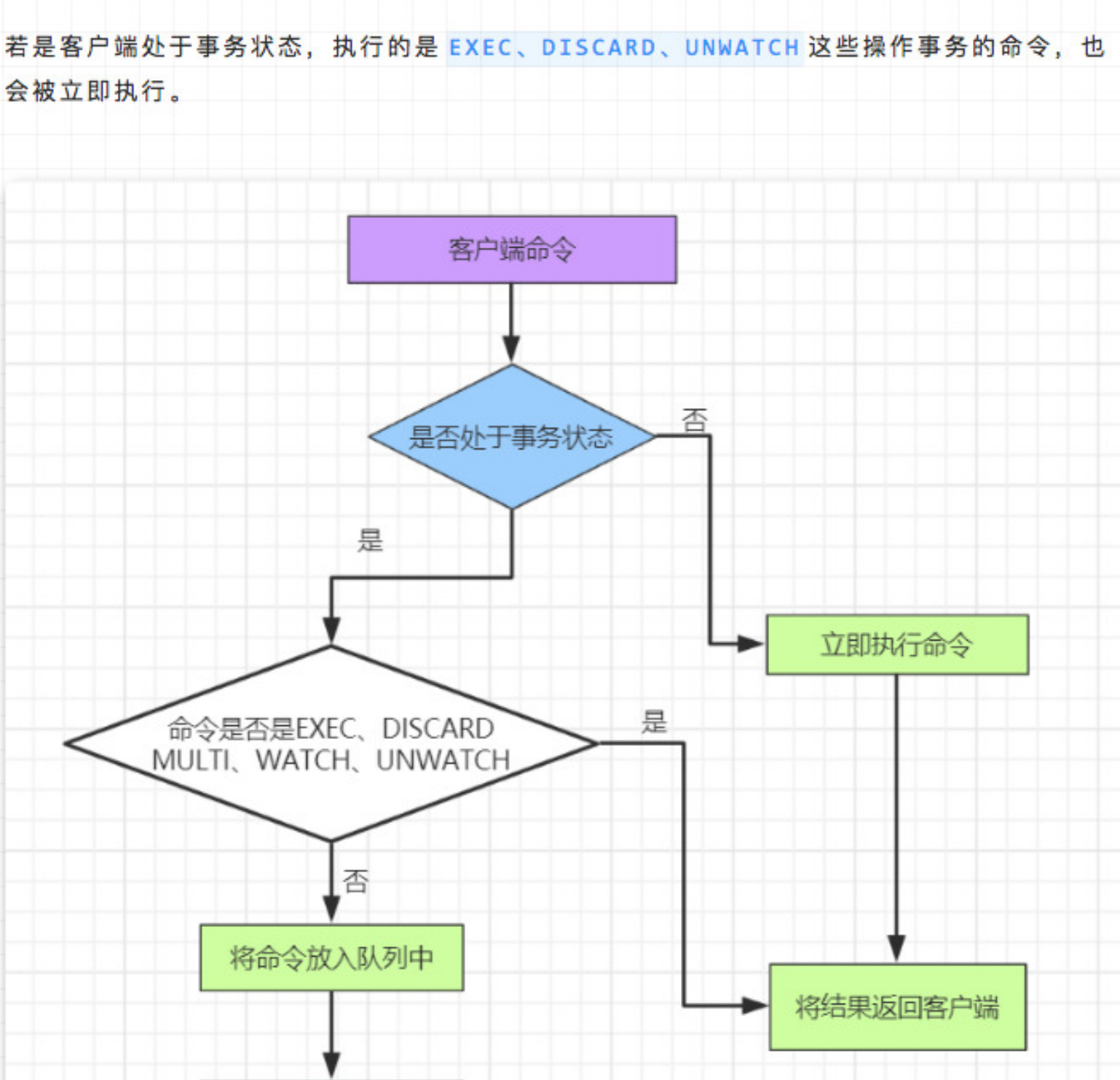
执行事务

当客户端执行EXEC命令的时候，上面的命令队列就会被按照先进先出的顺序被执行，当然执行的结果有成功有失败，这个后面分析。

上面说到当客户端处于非事务的状态命令发送到服务端会被立即执行，若是客户端处于事务状态命令就会被放进命令队列。

命令入队的时候，会按照顺序进入队列，队列以先进先出的特点来执行队列中的命令。

若是客户端处于事务状态，执行的是 `EXEC、DISCARD、UNWATCH` 这些操作事务的命令，也会被立即执行。



正常执行

还是上面的例子，执行如下的代码：

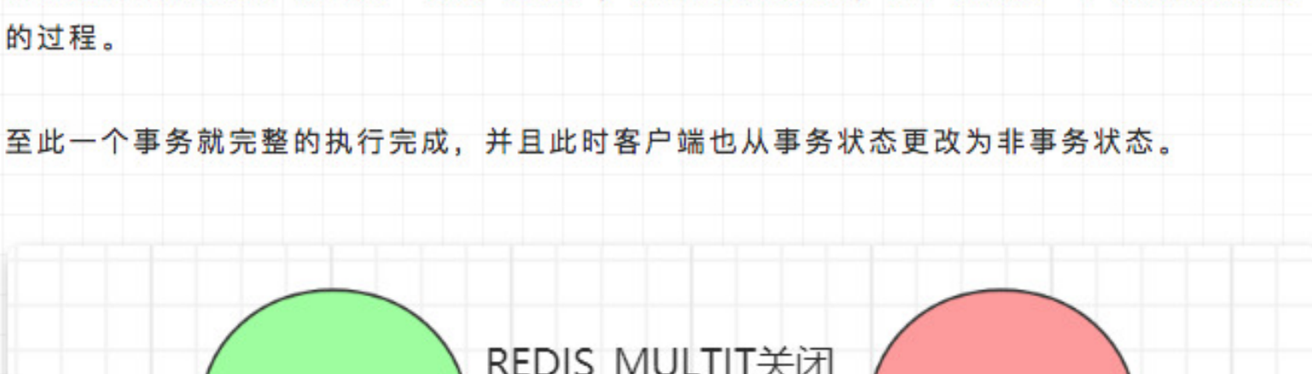
```
redis> MULTI
OK
redis> SET name "黎杜"
QUEUED
redis> GET name
QUEUED
```

所有的命令进入了队列，当最后执行EXEC，首先会执行SET命令，然后执行GET命令，并且执行后的结果也会进入一个队列中保存，最后返回给客户端：

回复的类型	回复的内容
status code reply	OK
bulk reply	"黎杜"

所以最后你会在客户端看到**「OK、黎杜」**，这样的结果显示，这个也就是一个事务成功执行的过程。

至此一个事务就完整的执行完成，并且此时客户端也从事务状态更改为非事务状态。



放弃事务

当然你也可以放弃执行该事务，只要你再次执行DISCARD操作就会放弃执行此次的事务。具体代码如下所示：

```
redis> MULTI
OK
redis> SET name "黎杜"
QUEUED
redis> GET name
QUEUED
redis> DISCARD    // 放弃执行事务
OK
```

DISCARD命令取消一个事务的时候，就会将命令队列清空，并且将客户端的状态从事务状态修改为非事务的状态。

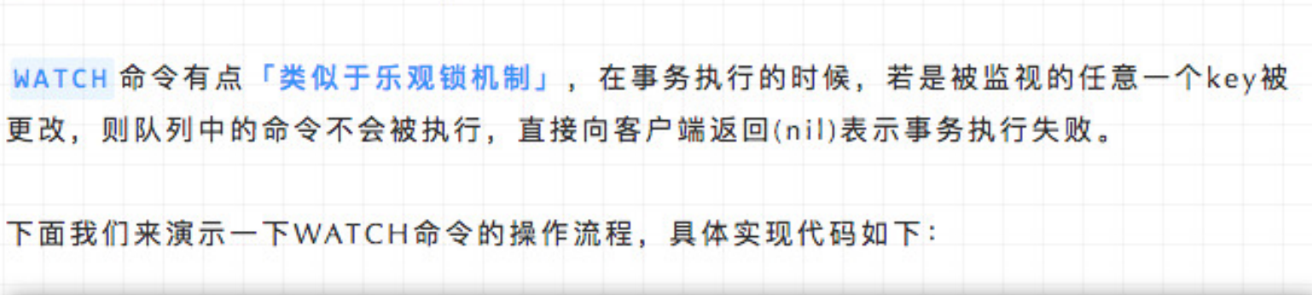
「Redis的事务是不可重复的」，当客户端处于事务状态的时候，再次向服务端发送MULTI命令时，直接就会向客户端返回错误。

WATCH 命令

`WATCH` 命令是在MULTI命令之前执行的，表示监视任意数量的key，与它对应的命令就是 `UNWATCH` 命令，取消监视的key。

`WATCH` 命令有点**「类似于乐观锁机制」**，在事务执行的时候，若是被监视的任意一个key被更改，则队列中的命令不会被执行，直接向客户端返回(nil)表示事务执行失败。

下面我们来演示一下WATCH命令的操作流程，具体实现代码如下：

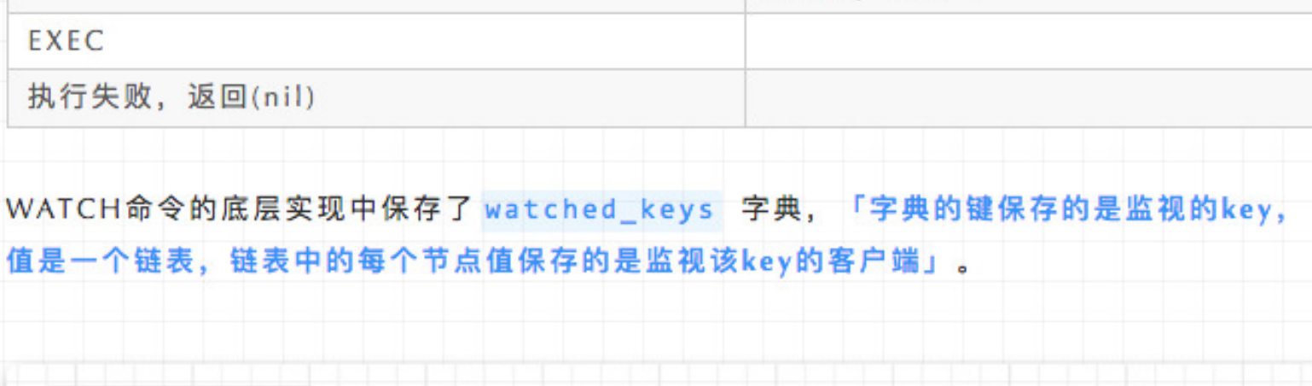


```
redis> WATCH num
OK
redis> MULTI
OK
redis> incrby num 10
QUEUED
redis> decrby num 1
QUEUED
redis> EXEC   // 执行成功
```

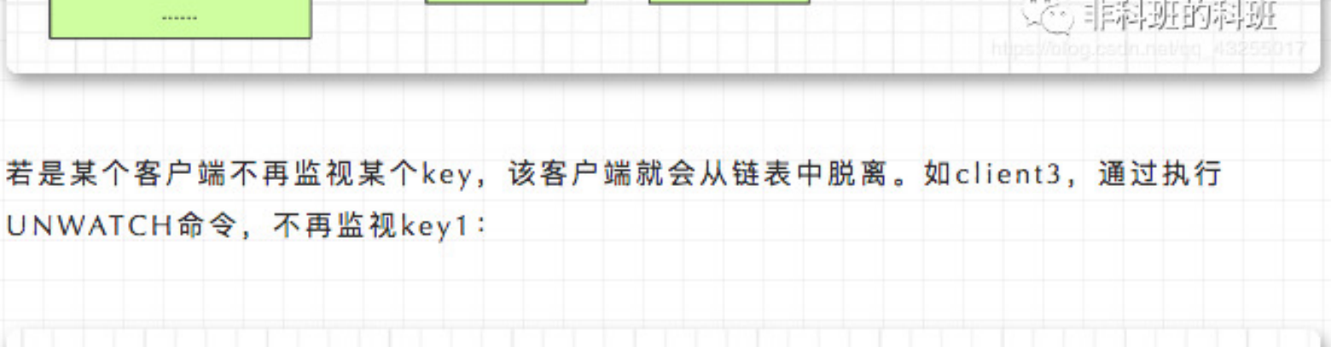
这个是 `WATCH` 命令的正常的操作流程，若是在其它的客户端，修改了被监视的任意key，就会放弃执行该事务，如下图所示：

客户端一	客户端二
WATCH num	
MULTI	
incrby num 10	get num
	decrby num 1
EXEC	
执行失败，返回(nil)	

WATCH命令的底层实现中保存了 `watched_keys` 字典，**「字典的键保存的是监视的key，值是一个链表，链表中的每个节点值保存的是监视该key的客户端」**。



若是某个客户端不再监视某个key，该客户端就会从链表中脱离。如client3，通过执行UNWATCH命令，不再监视key1：



错误处理

上面说到Redis是没有回滚机制的，那么执行的过程，若是不小心敲错命令，Redis的命令发送到服务端没有被立即执行，所以是暂时发现不到该错误。

那么在Redis中的错误处理主要分为两类：**「语法错误」**、**「运行错误」**。下面主要来讲解一下这两类错误的区别。

语法错误

比如执行命令的时候，命令的不存在或者错误的敲错命令、参数的个数不对等都会导致语法错误。

下面来演示一下，执行下面的四个命令，前后的两个命令是正确的，中间的两个命令是错误的，如下所示：

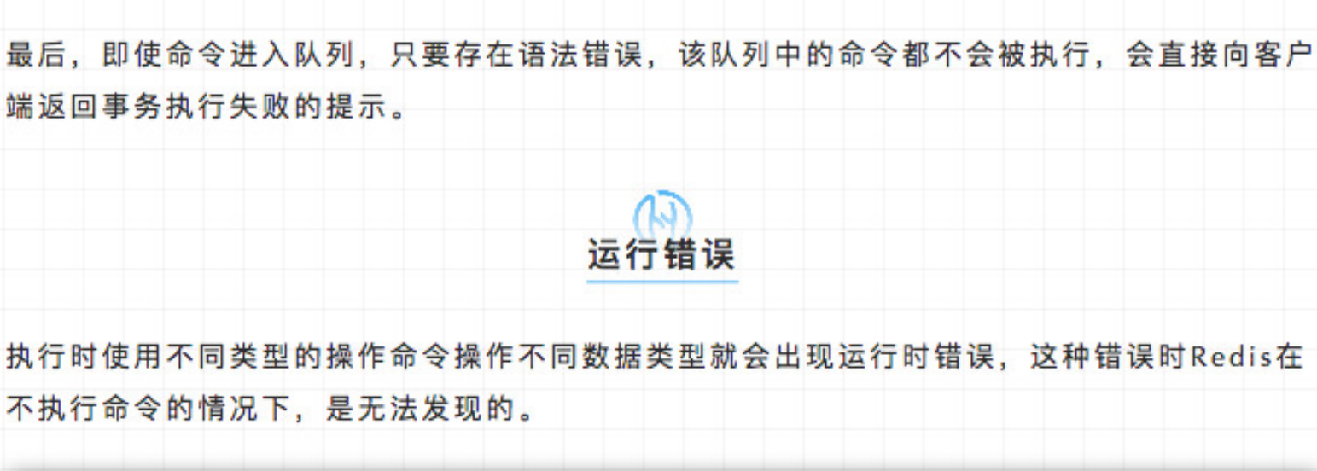
```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set num 1
QUEUED
127.0.0.1:6379> set num
(error) ERR wrong number of arguments for 'set' command
127.0.0.1:6379> ssset num 3
(error) ERR unknown command 'ssset'
127.0.0.1:6379> set num 2
QUEUED
127.0.0.1:6379> exec
(error) EXECABORT Transaction discarded because of previous errors.
```

语法错误是在Redis语法检测的时候就能发现的，所以当你执行错误命令的时候，也会即使的返回错误的提示。

最后，即使命令进入队列，只要存在语法错误，该队列中的命令都不会被执行，会直接向客户端返回事务执行失败的提示。

运行错误

执行时使用不同类型的操作命令操作不同数据类型就会出现运行时错误，这种错误时Redis在不执行命令的情况下，是无法发现的。



```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set num 3
QUEUED
127.0.0.1:6379> sadd num 4
QUEUED
127.0.0.1:6379> set num 6
QUEUED
127.0.0.1:6379> exec
1) OK
2) (error) WRONGTYPE Operation against a key holding the wrong kind of value
3) OK
127.0.0.1:6379> get key
"6"
```

这样就会导致，正确的命令被执行，而错误的命令不会不执行，这也显示出Redis的事务并不能保证数据的一致性，因为中间出现了错误，有些语句还是被执行了。

这样的结果只能程序员自己根据之前执行的命令，自己一步一步正确的回退，所谓自己的烂摊子，自己收拾。

Redis事务与Mysql事务

我们知道关系性数据库Mysql中具有事务的四大特性：**「原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）」**。

但是Redis的事务为了保证Redis除了客户端的请求高效，去除了传统关系型数据库的**「事务回滚、加锁、解锁」**这些消耗性能的操作，Redis的事务实现简单。

原子性中Redis的事务只能保证单个命令的原子性，多个命令就无法保证，如上面索道的运行时错误，即使中间有运行时错误出现也会正确的执行后面正确的命令，不具有回滚操作。

既然没有了原子性，数据的一致性也就无法保证，这些都需要程序员自己手动去实现。

Reids在进行事务的时候，不会被中断知道事务的运行结束，也具有一定的隔离性，并且Redis也能持久化数据。