# Redis事務 - 緩存

mincoder.com/article/1266.shtml

## 事務（transaction）

本文檔翻譯自： http://redis.io/topics/transactions 。

MULTI 、 EXEC 、 DISCARD 和 WATCH 是 Redis 事務的基礎。

事務可以一次執行多個命令， 並且帶有以下兩個重要的保證：

- 事務是一個單獨的隔離操作：事務中的所有命令都會序列化、按順序地執行。事務在執行的過程中，不會被其他客戶端發送來的命令請求所打斷。
- 事務是一個原子操作：事務中的命令要麼全部被執行，要麼全部都不執行。

  EXEC 命令負責觸發並執行事務中的所有命令：

  - 如果客戶端在使用 MULTI 開啟了一個事務之後，卻因為斷線而沒有成功執行 EXEC ，那麼事務中的所有命令都不會被執行。
  - 另一方面，如果客戶端成功在開啟事務之後執行 EXEC ，那麼事務中的所有命令都會被執行。

  當使用 AOF 方式做持久化的時候， Redis 會使用單個 write(2) 命令將事務寫入到磁盤中。

  然而，如果 Redis 服務器因為某些原因被管理員殺死，或者遇上某種硬件故障，那麼可能只有部分事務命令會被成功寫入到磁盤中。

  如果 Redis 在重新啟動時發現 AOF 文件出了這樣的問題，那麼它會退出，並匯報一個錯誤。

  使用 redis-check-aof 程序可以修復這一問題：它會移除 AOF 文件中不完整事務的信息，確保服務器可以順利啟動。

從 2.2 版本開始，Redis 還可以通過樂觀鎖（optimistic lock）實現 CAS （check-and-set）操作，具體信息請參考文檔的後半部分。

## 用法

MULTI 命令用於開啟一個事務，它總是返回 OK 。

MULTI 執行之後， 客戶端可以繼續向服務器發送任意多條命令， 這些命令不會立即被執行， 而是被放到一個隊列中， 當 EXEC 命令被調用時， 所有隊列中的命令才會被執行。

另一方面， 通過調用 DISCARD ， 客戶端可以清空事務隊列， 並放棄執行事務。

以下是一個事務例子， 它原子地增加了 foo 和 bar 兩個鍵的值：

```
> MULTI
OK
> INCR foo
QUEUED
> INCR bar
QUEUED
```

```
> EXEC
1) (integer) 1
2) (integer) 1
```

EXEC 命令的回覆是一個數組， 數組中的每個元素都是執行事務中的命令所產生的回覆。 其中， 回覆元素的先後順序和命令發送的先後順序一致。

當客戶端處於事務狀態時， 所有傳入的命令都會返回一個內容為 QUEUED 的狀態回覆（status reply）， 這些被入隊的命令將在 EXEC 命令被調用時執行。

## 事務中的錯誤

使用事務時可能會遇上以下兩種錯誤：

- 事務在執行 EXEC 之前，入隊的命令可能會出錯。比如說，命令可能會產生語法錯誤（參數數量錯誤，參數名錯誤，等等），或者其他更嚴重的錯誤，比如內存不足（如果服務器使用 maxmemory 設置了最大內存限制的話）。
- 命令可能在 EXEC 調用之後失敗。舉個例子，事務中的命令可能處理了錯誤類型的鍵，比如將列表命令用在了字符串鍵上面，諸如此類。

對於發生在 EXEC 執行之前的錯誤，客戶端以前的做法是檢查命令入隊所得的返回值：如果命令入隊時返回 QUEUED ，那麼入隊成功；否則，就是入隊失敗。如果有命令在入隊時失敗，那麼大部分客戶端都會停止並取消這個事務。

不過，從 Redis 2.6.5 開始，服務器會對命令入隊失敗的情況進行記錄，並在客戶端調用 EXEC 命令時，拒絕執行並自動放棄這個事務。

在 Redis 2.6.5 以前， Redis 只執行事務中那些入隊成功的命令，而忽略那些入隊失敗的命令。 而新的處理方式則使得在流水線（pipeline）中包含事務變得簡單，因為發送事務和讀取事務的回覆都只需要和服務器進行一次通訊。

至於那些在 EXEC 命令執行之後所產生的錯誤， 並沒有對它們進行特別處理： 即使事務中有某個/某些命令在執行時產生了錯誤， 事務中的其他命令仍然會繼續執行。

從協議的角度來看這個問題，會更容易理解一些。 以下例子中， LPOP 命令的執行將出錯， 儘管調用它的語法是正確的：

```
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 返回兩條批量回覆（bulk reply）： 第一條是 OK ，而第二條是 -ERR 。 至於怎樣用合適的方法來表示事務中

的錯誤，則是由客戶端自己決定的。

最重要的是記住這樣一條，即使事務中有某條/某些命令執行失敗了，事務隊列中的其他命令仍然會繼續執行 —— Redis 不會停止執行事務中的命令。

以下例子展示的是另一種情況，當命令在入隊時產生錯誤，錯誤會立即被返回給客戶端：

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

因為調用 INCR 命令的參數格式不正確，所以這個 INCR 命令入隊失敗。

## 為什麼 Redis 不支持回滾（roll back）

如果你有使用關係式數據庫的經驗，那麼「Redis 在事務失敗時不進行回滾，而是繼續執行餘下的命令」這種做法可能會讓你覺得有點奇怪。

以下是這種做法的優點：

- Redis 命令只會因為錯誤的語法而失敗（並且這些問題不能在入隊時發現），或是命令用在了錯誤類型的鍵上面：這也就是說，從實用性的角度來說，失敗的命令是由編程錯誤造成的，而這些錯誤應該在開發的過程中被發現，而不應該出現在生產環境中。
- 因為不需要對回滾進行支持，所以 Redis 的內部可以保持簡單且快速。

有種觀點認為 Redis 處理事務的做法會產生 bug ，然而需要注意的是，在通常情況下，回滾並不能解決編程錯誤帶來的問題。舉個例子，如果你本來想通過 INCR 命令將鍵的值加上 1 ，卻不小心加上了 2 ，又或者對錯誤類型的鍵執行了 INCR ，回滾是沒有辦法處理這些情況的。

鑑於沒有任何機制能避免程序員自己造成的錯誤，並且這類錯誤通常不會在生產環境中出現，所以 Redis 選擇了更簡單、更快速的無回滾方式來處理事務。

## 放棄事務

當執行 DISCARD 命令時，事務會被放棄，事務隊列會被清空，並且客戶端會從事務狀態中退出：

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

## 使用 check-and-set 操作實現樂觀鎖

WATCH 命令可以為 Redis 事務提供 check-and-set （CAS）行為。

被 WATCH 的鍵會被監視，並會發覺這些鍵是否被改動過了。 如果有至少一個被監視的鍵在 EXEC 執行之前被修改了， 那麼整個事務都會被取消， EXEC 返回空多條批量回覆（null multi-bulk reply）來表示事務已經失敗。

舉個例子， 假設我們需要原子性地為某個值進行增 1 操作（假設 INCR 不存在）。

首先我們可能會這樣做：

```
val = GET mykey
val = val + 1
SET mykey $val
```

上面的這個實現在只有一個客戶端的時候可以執行得很好。 但是， 當多個客戶端同時對同一個鍵進行這樣的操作時， 就會產生競爭條件。

舉個例子， 如果客戶端 A 和 B 都讀取了鍵原來的值， 比如 10 ， 那麼兩個客戶端都會將鍵的值設為 11 ， 但正確的結果應該是 12 才對。

有了 WATCH ， 我們就可以輕鬆地解決這類問題了：

```
WATCH mykey
val = GET mykey
val = val + 1
MULTI
SET mykey $val
EXEC
```

使用上面的代碼， 如果在 WATCH 執行之後， EXEC 執行之前， 有其他客戶端修改了 mykey 的值， 那麼當前客戶端的事務就會失敗。 程序需要做的， 就是不斷重試這個操作， 直到沒有發生碰撞為止。

這種形式的鎖被稱作樂觀鎖， 它是一種非常強大的鎖機制。 並且因為大多數情況下， 不同的客戶端會訪問不同的鍵， 碰撞的情況一般都很少， 所以通常並不需要進行重試。

## 瞭解 WATCH

WATCH 使得 EXEC 命令需要有條件地執行： 事務只能在所有被監視鍵都沒有被修改的前提下執行， 如果這個前提不能滿足的話，事務就不會被執行。

如果你使用 WATCH 監視了一個帶過期時間的鍵， 那麼即使這個鍵過期了， 事務仍然可以正常執行， 關於這方面的詳細情況，請看這個帖子： http://code.google.com/p/redis/issues/detail?id=270

WATCH 命令可以被調用多次。 對鍵的監視從 WATCH 執行之後開始生效， 直到調用 EXEC 為止。

用戶還可以在單個 WATCH 命令中監視任意多個鍵， 就像這樣：

```
redis> WATCH key1 key2 key3
OK
```

當 EXEC 被調用時， 不管事務是否成功執行， 對所有鍵的監視都會被取消。

另外， 當客戶端斷開連接時， 該客戶端對鍵的監視也會被取消。

使用無參數的 UNWATCH 命令可以手動取消對所有鍵的監視。 對於一些需要改動多個鍵的事務， 有時候程序需要同時對多個鍵進行加鎖， 然後檢查這些鍵的當前值是否符合程序的要求。 當值達不到要求時， 就可以使用 UNWATCH 命令來取消目前對鍵的監視， 中途放棄這個事務， 並等待事務的下次嘗試。

## 使用 WATCH 實現 ZPOP

WATCH 可以用於創建 Redis 沒有內置的原子操作。

舉個例子， 以下代碼實現了原創的 ZPOP 命令， 它可以原子地彈出有序集合中分值（score）最小的元素：

```
WATCH zset
element = ZRANGE zset 0 0
MULTI
    ZREM zset element
EXEC
```

程序只要重複執行這段代碼， 直到 EXEC 的返回值不是空多條回覆（null multi-bulk reply）即可。

## Redis 腳本和事務

從定義上來說， Redis 中的腳本本身就是一種事務， 所以任何在事務裡可以完成的事， 在腳本裡面也能完成。 並且一般來說， 使用腳本要來得更簡單，並且速度更快。

因為腳本功能是 Redis 2.6 才引入的， 而事務功能則更早之前就存在了， 所以 Redis 才會同時存在兩種處理事務的方法。

不過我們並不打算在短時間內就移除事務功能， 因為事務提供了一種即使不使用腳本， 也可以避免競爭條件的方法， 而且事務本身的實現並不複雜。

不過在不遠的將來， 可能所有用戶都會只使用腳本來實現事務也說不定。 如果真的發生這種情況的話， 那麼我們將廢棄並最終移除事務功能。