Cursores

Tiempo estimado: 20min

En Redis, podemos recorrer los resultados de determinadas consultas usando cursores, valor a valor, o bien mediante la recuperación completa del resultado. La diferencia es sutil pero sencilla: los cursores son la opción más óptima, consumen menos recursos y sacan el mejor rendimiento de la instancia cuando el resultado es grande. Mientras que si el resultado es pequeño, se puede obtener todo él de una tacada.

La lección la comenzamos con la introducción del concepto de cursor y varias clasificaciones. La idea es conocer cuáles son los distintos tipos de cursores y qué implementa Redis al respecto. A continuación, exponemos cómo trabajar con los cursores en Redis y los comandos a usar. Finalmente, presentamos un ejemplo sencillo de uso de cursores en *scripts* de Lua.

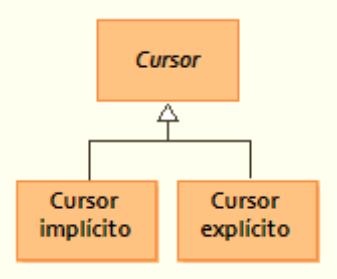
Al finalizar la lección, el estudiante sabrá:

- Qué es un cursor.
- Qué tipos de cursores hay.
- Cómo trabajar con cursores en Redis.

Introducción

Un cursor es un objeto para recorrer poco a poco el resultado de una consulta. Es una manera más óptima cuando el resultado puede ser grande, pues no solicita todo el resultado de golpe, sino que va haciéndolo poco a poco. Reduciendo así el consumo de recursos en el servidor y en el cliente. Se utiliza en muchos sistemas de gestión de bases de datos como, por ejemplo, MongoDB, Oracle, PostgreSQL, Redis y SQL Server.

Atendiendo a quién los crea y utiliza, se distingue entre cursores implícitos y explícitos.

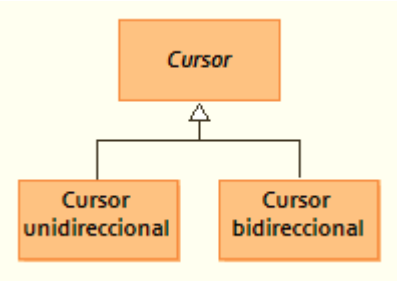


Un cursor implícito (*implicit cursor*) es aquel que define y administra el propio motor de bases de datos para recorrer el resultado. Un cursor explícito (*explicit cursor*) es aquel que creamos nosotros expresamente para iterar por un determinado resultado.

En Redis, los cursores son explícitos.

Direccionalidad de los cursores

Los cursores se pueden recorrer adelante o atrás. Atendiendo a las direcciones de recorrido, se distingue entre cursores unidireccionales y bidireccionales.



Un cursor unidireccional (*unidirectional cursor* o *forward-only cursor*) es aquel que permite recorrer cada grupo de valores del resultado una única vez; una vez el bloque ha sido accedido, ya no se puede volver a acceder con el mismo cursor. En cambio, un cursor bidireccional (*bidirectional cursor* o *scroll cursor*) es aquel que puede recorrerse adelante y atrás; se almacena en memoria cada bloque recorrido por si se desea volver a él más tarde.

En Redis, los cursores son unidireccionales.

Cursores explícitos

Un cursor explícito (*explicit cursor*) es aquel que creamos expresamente para recorrer un resultado. Es definido y administrado por el propio desarrollador. Se recomienda su uso cuando el resultado a recorrer es grande.

Para trabajar con cursores explícitos, en Redis, hay que hacerlo como sigue:

1. Solicitar la apertura del cursor y el primer bloque de datos mediante los comandos SCAN, HSCAN, SSCAN o ZSCAN.
2. Recorrer el resultado, bloque a bloque, mediante uno de los comandos anteriores.

En Redis, se utiliza un comando u otro para recorrer el resultado, atendiendo al tipo de los valores a recorrer. Por ejemplo, SCAN se utiliza para recorrer claves dadas de alta en la instancia; SSCAN para claves de tipo conjunto; ZSCAN para claves de tipo conjunto ordenado; y HSCAN para claves de tipo *array* asociativo. La sintaxis de los comandos es muy similar, tal como veremos en breve. Por otra parte, en Redis, los cursores se cierran automáticamente, a diferencia de otros motores donde hay que hacerlo explícitamente.

Es importante comprender cómo se trabaja con los cursores. Esto lo haremos con el comando SCAN y después lo extrapolaremos a los demás comandos cuyo comportamiento es muy similar, sólo que el resultado dependerá del tipo de valor almacenado en la clave.

Para comenzar, hay que comprender que Redis no tiene el concepto de variable definido dentro del lenguaje como sucede con otros motores. Por ejemplo, en PostgreSQL, los cursores se pueden utilizar mediante PL/pgSQL; en SQL Server, mediante T-SQL; etc. Redis lo máximo que tiene es Lua, pero no tiene un lenguaje específico que le dé mayor riqueza a los usuarios para realizar consultas más complejas. Recordemos, las consultas complejas hay que hacerlas con Lua.

Así pues, si no tenemos una variable donde almacenar el cursor y a partir de ella recorrerlo, ¿cómo se hace en Redis? Mediante los comandos cursores. Recordemos que atendiendo al valor a recorrer, tenemos que usar un comando u otro. Vamos a ilustrar el uso de los cursores con SCAN.

Apertura de cursores

La apertura de cursor (*cursor opening*) es la operación mediante la cual se solicita el acceso a determinados valores. Mediante el comando cursor SCAN, la sintaxis es la siguiente:

```
SCAN 0
SCAN 0 MATCH patrón
SCAN 0 COUNT tamaño
SCAN 0 MATCH patrón COUNT tamaño
```

SCAN se utiliza para recorrer los nombres de clave de la base de datos. Cuando no se indica ningún patrón de nombre de clave mediante MATCH, se recorre *todas* las claves. En cambio, si sólo deseamos recorrer aquellas que cumplen un determinado patrón, utilizaremos MATCH. Por su parte, la cláusula COUNT es una sugerencia del tamaño de bloque de cada iteración. Es simplemente eso, una

sugerencia. Redis utilizará aquel que considere mejor para cada situación.

Esto se ve mejor mediante un ejemplo. Supongamos las siguientes claves:

```
127.0.0.1:6379> KEYS *
 1) "band:3:clicks"
 2) "band:4:website"
 3) "band:2:website"
 4) "band:3:name"
 5) "band:3:website"
 6) "band:2:clicks"
 7) "band:1:name"
 8) "band:1:website"
 9) "band:1:clicks"
10) "band:4:clicks"
11) "band:4:name"
12) "band:2:name"
127.0.0.1:6379>
```

Ahora, supongamos que deseamos recorrer los nombres de clave cuyo patrón es band:*:name:

```
127.0.0.1:6379> KEYS band:*:name
1) "band:3:name"
2) "band:1:name"
3) "band:4:name"
4) "band:2:name"
127.0.0.1:6379>
```

Veamos pues cómo hacerlo mediante SCAN. Supongamos que deseamos recorrerlo en bloques de dos claves cada vez:

```
127.0.0.1:6379> SCAN 0 MATCH band:*:name COUNT 2
1) "2"
2) (empty list or set)
127.0.0.1:6379>
```

Esto es un fiel ejemplo de que COUNT es tan sólo una sugerencia. En la primera iteración, no devuelve ninguna clave. Lo importante del ejemplo es comprender lo que devuelve. Una lista de dos elementos:

- El primero devuelve el identificador del cursor a utilizar para solicitar la siguiente iteración.
- El segundo los datos de esta primera iteración.

Observe que a continuación de SCAN se indica el valor cero. Es el valor con el que se indica la apertura del cursor.

A continuación, se muestra otro ejemplo de apertura, esta vez con devolución de un primer bloque de datos formado por un único valor:

```
127.0.0.1:6379> SCAN 0 MATCH band:*:name COUNT 2
1) "8"
2) 1) "band:5:name"
127.0.0.1:6379>
```

Recorrido del resultado

Una vez abierto el cursor, hay que recorrer o acceder al resultado de la consulta bloque a bloque. A esta operación se la conoce formalmente como extracción (*fetch*). Para este fin, se dispone de los mismos comandos que para la apertura, SCAN, HSCAN, SSCAN y ZSCAN.

En el caso de SCAN, su sintaxis cambia muy poco con respecto a la apertura:

```
SCAN identificador
SCAN identificador MATCH patrón
SCAN identificador COUNT tamaño
SCAN identificador MATCH patrón COUNT tamaño
```

Observe que, en la apertura, a continuación del SCAN se utiliza el valor 0; en cambio, en el recorrido, se utiliza el valor del identificador devuelto por SCAN en la anterior iteración. Recuerde, siempre que se desea abrir un cursor nuevo, se usa como identificador 0. A continuación, se indica el patrón si se indicó en la apertura.

He aquí un ejemplo ilustrativo de apertura de cursor y de su recorrido:

```
127.0.0.1:6379> SCAN 0 MATCH band:*:name COUNT 1
```



```
1) "8"
2) (empty list or set)
127.0.0.1:6379> SCAN 8 MATCH band:*:name
1) "11"
2) 1) "band:4:name"
   2) "band:2:name"
   3) "band:3:name"
127.0.0.1:6379> SCAN 11 MATCH band:*:name
1) "0"
2) 1) "band:1:name"
127.0.0.1:6379>
```

El cursor se inicia con SCAN 0, el cual devuelve el identificador 8, a utilizar en la siguiente iteración, y un bloque vacío de cero claves. A continuación, solicitamos el siguiente bloque de datos. Para ello, volvemos a utilizar SCAN, pero esta vez pasamos 8, el valor devuelto por SCAN en la anterior iteración. Este valor le sirve a Redis para conocer el cursor en recorrido. En este caso, sí obtenemos claves en el resultado. Concretamente, tres. A continuación, solicitamos otra nueva iteración, para lo que usamos el identificador 11, devuelto por la anterior iteración. Sabemos que hemos alcanzado el final del cursor, cuando SCAN devuelve 0 como primer elemento, o sea, como identificador para la siguiente iteración.

Cierre de cursores

Los cursores se cierran automáticamente una vez recorridos.

Cursores de claves de tipo conjunto

Hemos utilizado el comando SCAN para recorrer los nombres de las claves, en vez de utilizar KEYS, cuando el posible resultado es grande. Cuando deseamos recorrer el resultado de una clave cuyo resultado es un conjunto con un número elevado de elementos, podemos utilizar SSCAN y ZSCAN. El primero con conjuntos desordenados y el segundo, ordenados:

```
SSCAN clave identificador
SSCAN clave identificador MATCH patrón
SSCAN clave identificador COUNT tamaño
SSCAN clave identificador MATCH patrón COUNT tamaño

ZSCAN clave identificador
ZSCAN clave identificador MATCH patrón
ZSCAN clave identificador COUNT tamaño
ZSCAN clave identificador MATCH patrón COUNT tamaño
```

Con ambos comandos sólo se puede acceder a una única clave. Se utiliza para recorrer conjuntos muy grandes de valores. No varias claves de tipo conjunto. Si lo deseamos, podemos filtrar qué elementos del conjunto recorrer, mediante el uso de la cláusula MATCH.

He aquí un ejemplo ilustrativo para recorrer una clave cuyo valor es un conjunto:

```
127.0.0.1:6379> SSCAN conjunto 0 MATCH *1*
1) "576"
2) 1) "812"
   2) "613"
   3) "512"
127.0.0.1:6379> SSCAN conjunto 576 MATCH *1*
1) "352"
2) 1) "561"
   2) "110"
   3) "1000"
127.0.0.1:6379> SSCAN conjunto 352 MATCH *1*
1) "400"
2) 1) "18"
   2) "51"
   3) "451"
127.0.0.1:6379> SSCAN conjunto 400 MATCH *1*
1) "464"
2) 1) "816"
   2) "180"
   3) "161"
   4) "391"
127.0.0.1:6379>
```


Esta clave contiene los primeros 1000 números enteros. De los cuales solicitamos sólo aquellos que contengan el dígito 1. No mostramos el recorrido completo porque el resultado es muy grande. Lo importante es comprender dos cosas: cómo funciona y que SSCAN se utiliza para recorrer claves de tipo conjunto.

Cursores de claves de tipo array asociativo

Para recorrer una clave cuyo valor es un array asociativo con muchos campos, se utiliza el comando HSCAN:

```
HSCAN clave identificador
HSCAN clave identificador MATCH patrón
HSCAN clave identificador COUNT tamaño
HSCAN clave identificador MATCH patrón COUNT tamaño
```

El patrón se utiliza para filtrar sólo aquellos campos que cumplen con él.

Ejemplo:

```
127.0.0.1:6379> HSCAN array 0 MATCH *o*
1) "13"
2)  1) "uno"
    2) "1"
    3) "dos"
    4) "2"
    5) "cuatro"
    6) "4"
127.0.0.1:6379> HSCAN array 13 MATCH *o*
1) "0"
2)  1) "cinco"
    2) "5"
    3) "ocho"
   4) "8"
127.0.0.1:6379>
```

Cursores en Lua

En un *script* de Lua, podemos utilizar los comandos SCAN, HSCAN, SSCAN y ZSCAN al igual que en redis-cli. Ejemplo:

```
127.0.0.1:6379> EVAL "local id, res = 0, {}; repeat local iter = redis.call('SCAN', id,
'MATCH', 'band:*:name'); id = iter[1]; for i = 1, #iter[2] do table.insert(res, iter[2][i]);
end until id == '0'; return res;" 0
1) "band:4:name"
2) "band:2:name"
3) "band:3:name"
4) "band:1:name"
127.0.0.1:6379>
```

El ejemplo anterior se comprende mejor si lo presentamos proposición a proposición:

```
local id, res = 0, {}

repeat
  local iter = redis.call('SCAN', id, 'MATCH', 'band:*:name')
  id = iter[1]
  for i = 1, #iter[2] do table.insert(res, iter[2][i]) end
until id == '0'

return res
```

