Listas

Tiempo estimado: 25min

Tras la presentación del tipo de datos cadena y el uso de módulos como medio para incrementar la funcionalidad de Redis, vamos a presentar el tipo de datos lista. El tipo cadena es simple, almacena un único valor formado por una secuencia de caracteres. En cambio, el tipo de datos lista es compuesto, permite almacenar varios valores, todos ellos de tipo cadena.

Para comenzar, introducimos el concepto de colección. A continuación, mostramos las colas y las pilas, colecciones que se pueden simular con las listas. Después, entramos en detalle con las listas. Y finalmente, presentamos varios comandos proporcionados por el módulo rxlists.

Al finalizar la lección, el estudiante sabrá:

- Cómo crear un par clave-valor de tipo lista.
- Cómo operar con una lista como si fuera una cola.
- Cómo operar con una lista como si fuera una pila.
- Cómo operar con una lista como si fuera una lista.

Introducción

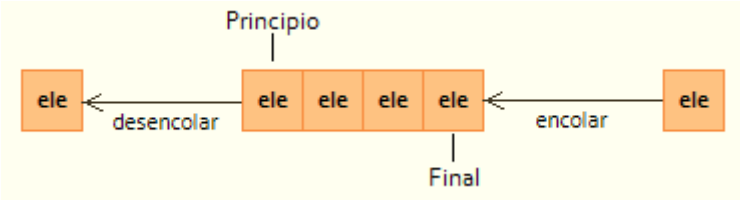
Una colección (*collection*) es un una estructura de datos o contenedor de valores. Cada uno de los valores almacenados se conoce formalmente como elemento (*element* o *item*). Atendiendo a cómo se trabaja y accede a los elementos, distinguimos principalmente entre listas, conjuntos, pilas y colas.

Redis sólo proporciona el concepto de lista, pero dota a esta estructura de datos de operaciones para que pueda funcionar como una lista así como una cola o pila. Vamos a analizar cada uno de estos tipos de colección, comenzando por la cola, siguiendo con la pila y finalizando con la lista. Los conjuntos tienen su propio tipo de datos y los veremos en la siguiente lección.

Colas

En Redis, una lista puede operarse como si se tratara de una cola (*queue*), un tipo de colección en el que sólo podemos acceder a los elementos usando sus extremos. Los nuevos elementos se añaden al final. Si lo que deseamos es acceder a uno, debemos hacerlo por el principio. Su funcionamiento se basa en quien primero entra, primero sale. Más o menos como el turno de espera de la carnicería.

La operación de añadir un nuevo elemento a la cola que, recordemos, siempre se debe hacer por el final, se conoce formalmente como encolamiento (*enqueue*). Mientras que la extracción de un elemento de la cola que, recordemos, debe ser siempre el primero de la cola, se conoce como desencolamiento (*dequeue*).



El último elemento que se encuentra en la cola, tras el cual se encolarán los nuevos, se conoce como final (*back*). El primero que se debe extraer, como principio (*front*).

Encolamiento

Visto lo visto, cuando trabajemos con una lista como si fuera una cola, utilizaremos los comandos siguientes para encolar nuevos elementos:


```
RPUSH clave valor
RPUSH clave valor valor valor...
RPUSHX clave valor
```
El comando RPUSH encola uno o más elementos a la lista. Si no existe la clave, la creará como vacía y después realizará el encolamiento. RPUSHX sólo encola el elemento indicado si la clave existe. Ambos comandos devuelven la longitud de la lista/cola tras realizar la operación. La R de los comandos hace referencia al lado derecho (*right*) de la lista.

Ejemplo:

```
127.0.0.1:6379> RPUSH cola dos tres cuatro
(integer) 4
127.0.0.1:6379>
```
Existe comandos equivalentes desde el punto de vista izquierdo, LPUSH y LPUSHX. En estos casos, la añadidura se hace por el comienzo, desplazando los elementos ya existentes una o varias posiciones a la derecha. Es importante no olvidar que, en las colas, siempre se hace por la derecha, esto es, por el final.

Desencolamiento

Si el encolamiento consiste en añadir nuevos elementos por la derecha, está claro que desencolarlos consistirá en extraerlos por la izquierda (*left*):

```
LPOP clave
```
Si no existe la clave, devuelve nil; en otro caso, suprime el elemento que se encuentra al principio de la lista, o sea, el de índice cero, y lo devuelve.

Veamos un ejemplo ilustrativo:

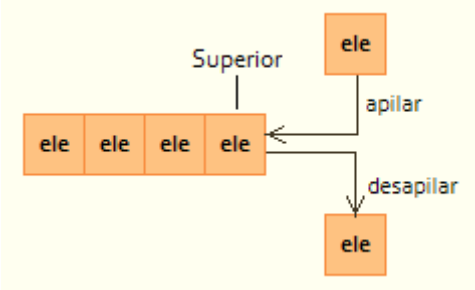
```
127.0.0.1:6379> LRANGE cola 0 -1
1) "uno"
2) "dos"
3) "tres"
4) "cuatro"
127.0.0.1:6379> LPOP cola
"uno"
127.0.0.1:6379> LRANGE cola 0 -1
1) "dos"
2) "tres"
3) "cuatro"
127.0.0.1:6379>
```
Existe un segundo comando de desencolamiento, BLPOP. Este comando se bloquea cuando la cola está vacía, a la espera de que encolemos algún elemento con otra sesión. Su sintaxis es como sigue:

```
BLPOP clave segundos
BLPOP clave clave clave... segundos
```
Podemos indicar una o más claves, separándolas por espacios. Hay que especificar el tiempo máximo que debe esperar el encolamiento. Si transcurrido éste no se ha añadido ningún elemento, se desbloqueará, devolviendo nil. Si indicamos cero como tiempo de espera, se bloqueará indefinidamente hasta que añadamos un elemento.

Pilas

Una pila (*stack*) es un tipo de colección en el que podemos acceder a los elementos usando sólo uno de sus extremos, el final. Su funcionamiento se basa en el último que entra, primero en salir, más o menos, como una pila de platos listos para fregar.

La operación para añadir nuevos elementos se conoce formalmente como apilamiento (*push*) y la de extracción de elementos como desapilamiento (*pop*).



Al último elemento apilado se le conoce como superior (*top*).

Apilamiento

El apilamiento es similar al encolamiento, siempre se hace por el final, por lo tanto, utilizaremos el comando RPUSH presentado en las colas.

Desapilamiento

En cambio, para realizar el desapilamiento, tenemos que extraer el último elemento añadido, no el primero como en las colas. Por lo que hay que utilizar RPOP (*right pop*) o BRPOP (*blocking right pop*):

```
RPOP clave
BRPOP clave segundos
BRPOP clave clave clave... segundos
```

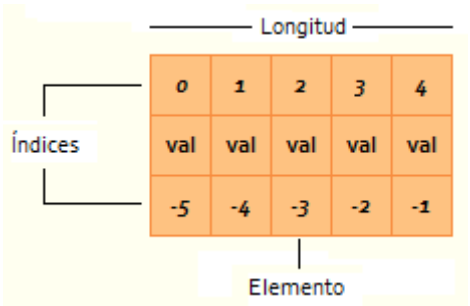
Ejemplo:

```
127.0.0.1:6379> LRANGE pila 0 -1
1) "uno"
2) "dos"
3) "tres"
4) "cuatro"
127.0.0.1:6379> RPOP pila
"cuatro"
127.0.0.1:6379> LRANGE pila 0 -1
1) "uno"
2) "dos"
3) "tres"
127.0.0.1:6379>
```

Listas

Ha llegado el momento de presentar formalmente el tipo de datos lista, con el cual podemos simular los dos anteriores. Una lista (*list*) no es más que un tipo de colección donde sus elementos se pueden acceder de cualquier manera. Podemos acceder a un elemento dado, podemos añadir un elemento entre dos, etc. Tal como hemos visto, tiene comandos que permiten trabajar con listas como si fueran colas o pilas.

Toda lista es una secuencia de cero, uno o más elementos de tipo cadena. Cada uno de los elementos se accede mediante su índice (*index*), esto es, su posición en la lista. Al igual que en las cadenas, los índices comienzan en cero y pueden ser positivos o negativos.



Las listas se pueden clasificar en unidimensionales y multidimensionales. Una lista unidimensional (*one-dimensional list*) es aquella que tiene una única dimensión, esto es, ninguno de sus elementos puede ser a su vez otra lista. En cambio, una lista multidimensional (*multidimensional list*) es aquella que sí permite

que sus elementos sean a su vez otra lista. En Redis, los elementos de las listas deben ser de tipo cadena y, por lo tanto, son necesariamente unidimensionales.

Lectura de elementos

Para acceder al valor de un determinado elemento, hay que utilizar el comando LINDEX junto con su posición en la lista:

```
LINDEX clave índice
```

Con LINDEX, el elemento se mantiene en la lista. Simplemente, se consulta su valor.

Ejemplo:

```
127.0.0.1:6379> EXISTS lista
(integer) 0
127.0.0.1:6379> RPUSH lista cero
(integer) 1
127.0.0.1:6379> RPUSH lista uno dos
(integer) 3
127.0.0.1:6379> RPUSH lista tres
(integer) 4
127.0.0.1:6379> LINDEX lista 0
"cero"
127.0.0.1:6379> LINDEX lista 1
"uno"
127.0.0.1:6379> LINDEX lista 2
"dos"
127.0.0.1:6379> LINDEX lista -1
"tres"
127.0.0.1:6379> LINDEX lista 100
(nil)
127.0.0.1:6379>
```

LINDEX devuelve nil cuando el índice indicado se encuentra fuera de rango.

A veces, se hace necesario acceder a un fragmento de elementos consecutivos de la lista. Para esta función, tenemos el comando LRANGE:

```
LRANGE clave inicio fin
```

Hay que indicarle los índices inicial y final de los elementos del fragmento. Ejemplo:

```
127.0.0.1:6379> LRANGE lista 0 -1
1) "cero"
2) "uno"
3) "dos"
4) "tres"
127.0.0.1:6379> LRANGE lista 1 -1
1) "uno"
2) "dos"
3) "tres"
127.0.0.1:6379>
```

Longitud de la lista

Para conocer el tamaño (*size*) o longitud (*length*) de la lista, esto es, el número de elementos que tiene, hay que utilizar el comando LLEN:

```
LLEN clave
```

Ejemplo:

```
127.0.0.1:6379> LLEN lista
(integer) 4
127.0.0.1:6379>
```

Si la clave no existe, devuelve 0.

Asignación de elementos

Si necesitamos cambiar el valor de un elemento, utilizaremos el comando LSET:

```
LSET clave índice valor
```

He aquí un sencillo ejemplo ilustrativo:


```
127.0.0.1:6379> LRANGE lista 0 -1
1) "cero"
2) "uno"
3) "dos"
4) "tres"
127.0.0.1:6379> LSET lista 1 UNO
OK
127.0.0.1:6379> LRANGE lista 0 -1
1) "cero"
2) "UNO"
3) "dos"
4) "tres"
127.0.0.1:6379>
```

Si el índice indicado se encuentra fuera de rango, propagará error.

Añadidura de elementos

La añadidura de un elemento consiste en insertar un elemento en un determinado lugar desplazando a derecha o izquierda los que le siguen o preceden. Ya hemos visto comandos para añadir elementos al comienzo, LPUSH y LPUSHX; y al final, RPUSH y RPUSHX. Si lo que necesitamos es añadir entre dos elementos, podemos usar LINSERT:

```
LINSERT clave BEFORE pivote valor
LINSERT clave AFTER pivote valor
```

En Redis, la añadidura se hace con respecto a un valor, no a un índice. El valor a buscar y que usaremos como punto de inserción, se conoce como pivote (*pivot*). La añadidura del nuevo valor se puede hacer antes (*before*) o después (*after*) del pivote.

Veamos unos ejemplos ilustrativos:

```
127.0.0.1:6379> LRANGE lista 0 -1
1) "cero"
2) "uno"
3) "dos"
4) "tres"
127.0.0.1:6379> LINSERT lista BEFORE uno "cero con cinco"
(integer) 5
127.0.0.1:6379> LRANGE lista 0 -1
1) "cero"
2) "cero con cinco"
3) "uno"
4) "dos"
5) "tres"
127.0.0.1:6379> LINSERT lista AFTER uno "uno con cinco"
(integer) 6
127.0.0.1:6379> LRANGE lista 0 -1
1) "cero"
2) "cero con cinco"
3) "uno"
4) "uno con cinco"
5) "dos"
6) "tres"
127.0.0.1:6379>
```

Supresión de elementos

Si necesitamos suprimir un determinado elemento, podemos hacerlo a través de su valor. El comando LREM se utiliza para suprimir un número máximo de elementos a través de su valor:

```
LREM clave veces valor
```

Si indicamos cero como número de veces a suprimirlo, suprimirá todos los elementos que presenten el valor. Si se indica un número positivo, sólo las primeras veces indicadas. Si deseamos que en vez de comenzar la búsqueda y supresión de los elementos por el inicio, se haga por el final, se indicará un valor negativo.

Ejemplo ilustrativo:

```
127.0.0.1:6379> LRANGE lista 0 -1
1) "cero"
```



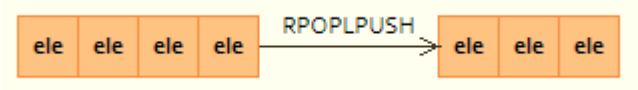
```
2) "uno"
3) "dos"
4) "tres"
5) "tres"
6) "uno"
7) "cuatro"
127.0.0.1:6379> LREM lista 1 tres
(integer) 1
127.0.0.1:6379> LRANGE lista 0 -1
1) "cero"
2) "uno"
3) "dos"
4) "tres"
5) "uno"
6) "cuatro"
127.0.0.1:6379> LREM lista 0 uno
(integer) 2
127.0.0.1:6379> LRANGE lista 0 -1
1) "cero"
2) "dos"
3) "tres"
4) "cuatro"
127.0.0.1:6379>
```

El comando devuelve el número de supresiones realizadas. Si la clave no existe, devolverá cero.

Comandos RPOPLPUSH y BRPOPLPUSH

Los comandos RPOPLPUSH y BRPOPLPUSH lo que hacen es desapilar un elemento, es decir, quitar el último elemento de una lista, y añadirlo al comienzo de otra:

```
RPOPLPUSH origen destino
BRPOPLPUSH origen destino
```



La versión BRPOPLPUSH es bloqueadora tal como vimos con los comandos BLPOP y BRPOP. He aquí un ejemplo ilustrativo:

```
127.0.0.1:6379> LRANGE origen 0 -1
1) "cero"
2) "uno"
3) "dos"
127.0.0.1:6379> LRANGE destino 0 -1
1) "UNO"
2) "CERO"
127.0.0.1:6379> RPOPLPUSH origen destino
"dos"
127.0.0.1:6379> LRANGE origen 0 -1
1) "cero"
2) "uno"
127.0.0.1:6379> LRANGE destino 0 -1
1) "dos"
2) "UNO"
3) "CERO"
127.0.0.1:6379>
```

Si el origen no existe o está vacío, no hará nada. Si el destino no existe, lo creará como una lista cuyo único elemento es el desapilado del origen.

Módulo rxlists

El módulo rxlists añade varios comandos adicionales para trabajar con listas.

LMPOP y RMPOP

Los comandos LMPOP y RMPOP suprimen varios elementos del inicio y el final de la lista, respectivamente, y los devuelven. Sus sintaxis son como sigue:


```
LMPOP clave número
RMPOP clave número
```

Ejemplo:

```
127.0.0.1:6379> LRANGE lista 0 -1
1) "uno"
2) "dos"
3) "tres"
4) "cuatro"
5) "cinco"
6) "seis"
7) "siete"
127.0.0.1:6379> LMPOP lista 3
1) "uno"
2) "dos"
3) "tres"
127.0.0.1:6379> LRANGE lista 0 -1
1) "cuatro"
2) "cinco"
3) "seis"
4) "siete"
127.0.0.1:6379>
```

LSPLICE

El comando LSPLICE suprime elementos del final de una lista origen y los añade al comienzo de otra:

```
LSPLICE origen destino númeroDeElementos
```

