Cadenas

Tiempo estimado: 15min

Con esta lección, comenzamos la presentación de los distintos tipos de datos soportados por Redis. Empezamos con las cadenas (*strings*). En las siguientes, presentaremos otros tipos como las listas, los conjuntos y los *arrays* asociativos.

Primero, presentamos el tipo de datos cadena, mostrando los posibles tipos de valores que se puede almacenar con él. A continuación, mostramos cómo debemos acceder a un par cuyo valor es una cadena. Atendiendo al valor almacenado, después, presentamos qué comandos podemos usar para operar con secuencias de texto y números. Finalmente, presentamos los comandos múltiples y los adicionales proporcionados por el módulo rxstrings.

Al finalizar la lección, el estudiante sabrá:

- Qué tipos de valores puede almacenar en pares clave-valor de tipo cadena.
- Cómo crear y actualizar los valores de pares clave-valor de tipo cadena.
- Cómo leer los valores de pares clave-valor de tipo cadena.

Introducción

El tipo de datos simple cadena (*string*) permite el almacenamiento de una secuencia de caracteres, la cual puede representar un texto, un número o un valor binario como, por ejemplo, una imagen, un vídeo o un audio. Recordemos que en Redis, el tipo de datos del valor determina las operaciones que podemos utilizar contra el par clave-valor. Así, por ejemplo, si el valor almacenado es un número, podemos incrementarlo o decrementarlo mediante operaciones como INCR o DECR. Pero estas operaciones no podremos utilizarlas si el valor es, por ejemplo, de tipo lista.

El tamaño máximo de una cadena es 512MB.

Básicamente, con un valor de tipo cadena, podemos incrementar o decrementar su valor si es un número o bien añadirle un fragmento en cualquier caso.

Creación de un par clave-valor

Para crear un par clave-valor de tipo cadena, debemos utilizar cualquiera de los comandos siguientes:

```
SET clave valor
SETNX clave valor
SETEX clave segundos valor
PSETEX clave milisegundos valor
```

El comando SET crea la clave con el valor indicado; si ya existe, le fija su nuevo valor. SETNX, por su parte, se utiliza para crear la clave, si ésta no existe ya; en caso de existir, no hará nada. Finalmente, SETEX y PSETEX son similares a SET, pero crean o transforman el par clave-valor como temporal, fijando su tiempo de vida en segundos o milisegundos, respectivamente.

Sólo SETNX tiene una función meramente creadora, aunque si la clave ya existe, no hará nada. SET, SETEX y PSETEX, por su parte, pueden crear la clave o actualizar su valor actual. En caso de creación, SET y SETNX crean la clave como persistente, mientras que SETEX y PSETEX lo hacen como temporal. En caso de actualización, SET no toca el tipo del par clave-valor, pero SETEX y PSETEX sí lo hacen: si es persistente, lo convierten a temporal; y si es temporal, lo dejan como tal.

He aquí unos ejemplos ilustrativos:

```
127.0.0.1:6379> SET x 1
OK
127.0.0.1:6379> SETNX x 2
(integer) 0
127.0.0.1:6379> SETEX x 123 2
```



```
OK
127.0.0.1:6379>
```
SET y SETEX devuelven OK si llevan a cabo la creación o actualización de la clave. Por su parte, SETNX devuelve 1 si lleva a cabo la creación y 0 si no lo hace.

Cuando el valor contiene espacios, hay que delimitarlo por comillas simples (') o dobles ("). Ejemplo:

```
127.0.0.1:6379[2]> SET clave "Hola Mundo!"
OK
127.0.0.1:6379[2]>
```

Lectura de un par clave-valor

Para consultar el valor de una clave cuyo valor es una cadena, hay que utilizar los siguientes comandos:

```
GET clave
GETSET clave nuevoValor
```
El comando GET devuelve el valor de la clave. GETSET también, pero además actualiza su valor actual, eso sí, devolviendo el valor que tiene antes de la actualización.

Veamos unos ejemplos ilustrativos:

```
127.0.0.1:6379> GET x
"1"
127.0.0.1:6379> GETSET x 2
"1"
127.0.0.1:6379> GET x
"2"
127.0.0.1:6379>
```
Si la clave no existe, devuelve nil:

```
127.0.0.1:6379> GET noexiste
(nil)
127.0.0.1:6379>
```
Si la clave no existe, GETSET devuelve nil y la crea con el valor indicado.

Operaciones de texto o binarias

Cuando el valor almacenado en el par clave-valor lo consideramos un texto o un contenido binario, podemos utilizar varios comandos para acceder a su contenido, ya sea en modo lectura o escritura. Vamos a verlos detenidamente.

Concatenación

Para añadir un contenido al final de un valor, se utiliza el comando APPEND:

```
APPEND clave textoAAñadir
```
Ejemplo:

```
127.0.0.1:6379> GET despedida
"bye"
127.0.0.1:6379> APPEND despedida " bye"
(integer) 7
127.0.0.1:6379> GET despedida
"bye bye"
127.0.0.1:6379>
```
El comando devuelve el tamaño de la cadena tras la actualización. Si la clave no existe, la crea como vacía y a continuación realiza la concatenación.

Fragmentos de contenido

Para acceder a un fragmento del contenido, se utiliza el comando GETRANGE:

```
GETRANGE clave inicio fin
```
Los índices comienzan en cero al igual que en C/C++, Java o JavaScript. Si se desea, se puede indicar índices negativos, donde -1 representa el último carácter; -2, el penúltimo; y así sucesivamente.

Ejemplos ilustrativos:


```
127.0.0.1:6379> SET ejemplo "Esto es un ejemplo"
OK
127.0.0.1:6379> GET ejemplo
"Esto es un ejemplo"
127.0.0.1:6379> GETRANGE ejemplo 5 9
"es un"
127.0.0.1:6379> GETRANGE ejemplo 5 -12
"es"
127.0.0.1:6379>
```

Si lo que deseamos es actualizar un fragmento, podemos utilizar el comando SETRANGE:

```
SETRANGE clave inicio fragmento
```

Ejemplo:

```
127.0.0.1:6379> GET ejemplo
"Esto es un ejemplo"
127.0.0.1:6379> SETRANGE ejemplo 5 ES
(integer) 18
127.0.0.1:6379> GET ejemplo
"Esto ES un ejemplo"
127.0.0.1:6379>
```

Longitud del contenido

Para conocer el tamaño (*size*) o longitud (*length*) del contenido, esto es, el número de caracteres del valor, se utiliza el comando STRLEN:

```
STRLEN clave
```

Veamos un ejemplo:

```
127.0.0.1:6379> GET ejemplo
"Esto es un ejemplo"
127.0.0.1:6379> STRLEN ejemplo
(integer) 18
127.0.0.1:6379>
```

Si la clave no existe, devuelve cero.

Operaciones aritméticas

Cuando el valor almacenado en un par clave-valor es un número, podemos trabajar con el valor como sigue.

Incremento del valor

Para incrementar el valor, podemos utilizar los comandos siguientes:

```
INCR clave
INCRBY clave incremento
INCRBYFLOAT clave incremento
```

INCR incrementa en una unidad el valor almacenado. Por su parte, INCRBY e INCRBYFLOAT en las unidades indicadas. Los tres comandos devuelven el nuevo valor tras el incremento. Si la clave no existe, la crean a cero y a continuación realizan el incremento.

Ejemplos:

```
127.0.0.1:6379> GET x
"1"
127.0.0.1:6379> INCR x
(integer) 2
127.0.0.1:6379> INCRBY x 123
(integer) 125
127.0.0.1:6379> EXISTS y
(integer) 0
127.0.0.1:6379> INCR y
(integer) 1
127.0.0.1:6379>
```


Decremento del valor

Para decrementar un valor, podemos utilizar comandos similares a los de incremento:

```
DECR clave
DECRBY clave decremento
DECRBYFLOAT clave decremento
```

Comandos múltiples

Hasta el momento, hemos presentado comandos que sólo pueden trabajar a nivel de un único par clave-valor. Redis implementa varios comandos que permiten, mediante una única instrucción, trabajar sobre varias claves. Este tipo de comandos llevan el prefijo M en su nombre:

```
MGET clave clave clave...
MSET clave valor clave valor clave valor...
MSETNX clave valor clave valor clave valor...
```

MGET devuelve una lista con los valores de las claves indicadas, al igual que GET. Cuando una clave no existe, devolverá nil en su lugar. Ejemplo:

```
127.0.0.1:6379> GET x
"1"
127.0.0.1:6379> GET y
"2"
127.0.0.1:6379> GET z
(nil)
127.0.0.1:6379> MGET x y z
1) "1"
2) "2"
3) (nil)
127.0.0.1:6379>
```

MSET y MSETNX crean o actualizan claves al igual que SET y SETNX, respectivamente, pero de una vez:

```
127.0.0.1:6379> MSET x 1 y 2 z 3
OK
127.0.0.1:6379> MGET x y z
1) "1"
2) "2"
3) "3"
127.0.0.1:6379>
```

Para que MSETNX realice su trabajo, es necesario que ninguna de las claves indicadas exista. Si existe alguna, la operación no se realizará. Devuelve 0 si alguna de las claves ya existe y, por lo tanto, no hace nada; o bien 1 si crea las claves. Veámoslo mediante un ejemplo ilustrativo:

```
127.0.0.1:6379> EXISTS x y z
(integer) 0
127.0.0.1:6379> MSETNX x 1 y 2
(integer) 1
127.0.0.1:6379> EXISTS x y z
(integer) 2
127.0.0.1:6379> MSETNX x 111 y 222 z 333
(integer) 0
127.0.0.1:6379> MGET x y z
1) "1"
2) "2"
3) (nil)
127.0.0.1:6379>
```

Módulo rxstrings

Tal como veremos en la siguiente lección, un módulo (*module*) es un componente que extiende la funcionalidad de Redis añadiendo nuevos comandos y/o tipos de datos. El módulo rxstrings añade comandos adicionales para trabajar con cadenas.

Preposición de texto

Tal como hemos visto ya, si deseamos añadir texto al final de un valor de tipo cadena, podemos utilizar el comando APPEND. Para añadirlo al comienzo, podemos usar el comando PREPEND proporcionado

por el módulo rxstrings:

```
PREPEND clave textoAAñadir
```

Ejemplo:

```
127.0.0.1:6379> SET clave " Mundo!"
OK
127.0.0.1:6379> PREPEND clave Hola
(integer) 11
127.0.0.1:6379> GET clave
"Hola Mundo!"
127.0.0.1:6379>
```

Ejecución condicional

El comando CHECKAND comprueba si el valor de un par clave-valor es uno indicado y, si es así, ejecuta otro comando (APPEND, DECR, DECRBY, INCR, INCRBY, INCRBYFLOAT, PSETEX, SET, SETEX, SETNX). Su sintaxis es como sigue:

```
CHECKAND clave valor COMANDO
```

El siguiente ejemplo ilustrativo muestra cómo cambiar el valor de x a dos si éste es uno:

```
CHECKAND x uno SET x dos
```

El comando devuelve nil si la comprobación no se cumple. Y si se cumple, lo devuelto por el comando adjunto.

