

1. ESTRUCTURAS DE DATOS

Redis Strings

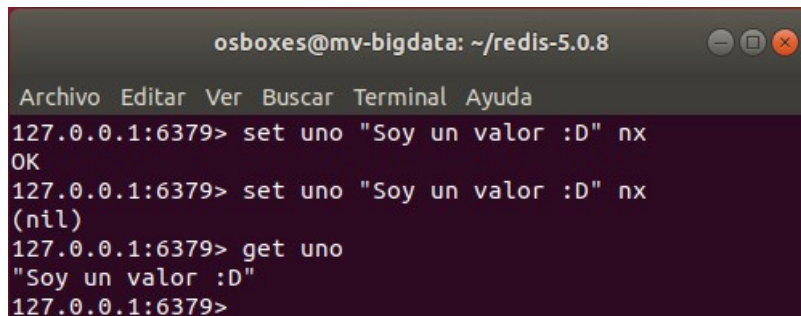
Pregunta: ¿Qué ocurre si ejecuto el siguiente comando?

redis> set uno "Soy un valor :D" nx

uno: Es el nombre de la clave que queremos establecer o actualizar.

"Soy un valor :D": Es el valor que queremos asociar a la clave.

nx: Es la opción que significa "establecer la clave sólo si no existe ya". Garantiza que la operación SET sólo se realice si la clave no existe ya.



```
osboxes@mv-bigdata: ~/redis-5.0.8
Archivo Editar Ver Buscar Terminal Ayuda
127.0.0.1:6379> set uno "Soy un valor :D" nx
OK
127.0.0.1:6379> set uno "Soy un valor :D" nx
(nil)
127.0.0.1:6379> get uno
"Soy un valor :D"
127.0.0.1:6379>
```

Al ejecutar el comando **GET** con la clave `"uno"`, podemos observar que su valor es "Soy un valor :D".

Pregunta: ¿Y si vuelvo a ejecutarlo de nuevo sigue devolviendo el mismo resultado? ¿Por qué?

Primera ejecución:

Establecemos la clave "uno" al valor "Soy un valor :D" con el comando **SET**.

La respuesta es "OK" porque la operación se ha realizado correctamente.

Segunda ejecución:

El comando **SET** intenta actualizar el valor de la clave "uno" a "Soy un valor :D".

Sin embargo, la opción nx impide que se actualice el valor de una clave si ya existe en la base de datos.

Como la clave "uno" ya existe, la operación falla y la respuesta es nil.

Pregunta: Y si ejecuto los siguientes comandos en orden, ¿Qué ocurre?

```
127.0.0.1:6379> set dos "Soy otro valor :3" xx
(nil)
127.0.0.1:6379> set dos "Soy otro valor :3"
OK
127.0.0.1:6379> set dos "Soy otro valor distinto :(" xx
OK
```

redis> set dos "Soy otro valor :3" xx

Este comando intenta establecer la clave "dos" al valor "Soy otro valor :3" con la opción xx.

La opción xx indica que el comando solo debe actualizar el valor de la clave si ya existe.

Como la clave "dos" no existe en la base de datos, la operación no se realiza y el comando devuelve nil.

redis> set dos "Soy otro valor :3"

Este comando sin ninguna opción establece la clave "dos" con el valor "Soy otro valor :3".

Como no se ha especificado ninguna opción, el comportamiento por defecto es actualizar o crear la clave si no existe.

Por lo tanto, el comando se ejecuta correctamente y devuelve OK.

redis> set dos "Soy otro valor distinto :(" xx

Este comando establece la clave "dos" al valor "Soy otro valor distinto :(" con la opción xx.

La opción xx significa "establecer la clave sólo si ya existe".

Como la clave "dos" ya existe en la base de datos (a partir del segundo comando), la opción xx permite que el comando actualice el valor de la clave existente.

Por lo tanto, el comando se ejecuta correctamente y devuelve OK.

Redis Hashes

Pregunta: ¿Qué comando tendríamos que utilizar si quisiésemos obtener todos los campos de una entrada?

redis> HKEYS user:1000

```
127.0.0.1:6379> hmset user:1000 username antirez birthyear 1977
OK
127.0.0.1:6379> hkeys user:1000
1) "username"
2) "birthyear"
```

El comando **HKEYS** devuelve todos los nombres de campo (claves) del hash especificado por la clave. Cada campo representa un par clave-valor en el hash.

Al ejecutar el comando **HGETALL** devuelve todos los campos y valores en un hash.

```
127.0.0.1:6379> hgetall user:1000
1) "username"
2) "antirez"
3) "birthyear"
4) "1977"
```

Redis Sets

Pregunta: ¿Qué comando tendríamos que utilizar si quisiésemos saber si existe un elemento en mi set?

```
127.0.0.1:6379> sadd myset first second third
(integer) 3
127.0.0.1:6379> smembers myset
1) "third"
2) "first"
3) "second"
127.0.0.1:6379> sismember myset first
(integer) 1
127.0.0.1:6379> sismember myset amarillo
(integer) 0
```

Para determinar si un elemento existe en un conjunto en Redis, podemos utilizar el comando **SISMEMBER**.

Añadimos los elementos "first", "second" y "third" a un conjunto llamado "myset". Para obtener todos los valores (miembros) del conjunto, podemos usar el comando **SMEMBERS**:

redis> smembers myset

Para saber si un elemento existe en el conjunto, podemos usar el comando **SISMEMBER**:

redis> sismember myset first

Si el elemento existe, el comando **SISMEMBER** devuelve 1. El elemento "first" existe en el conjunto "myset", por lo que el comando devuelve 1.

redis> sismember myset amarillo

Si el elemento no existe, el comando **SISMEMBER** devuelve 0. El elemento "amarillo" no existe en el conjunto "myset", por lo que el comando devuelve 0.

Pregunta: El comando SPOP devuelve uno o varios elementos aleatorios de nuestro set. ¿Qué efectos tiene este comando sobre nuestro set? ¿Qué otro comando podríamos utilizar para evitar este efecto sobre nuestro set?

SPOP: Elimina y devuelve uno o varios elementos aleatorios del conjunto especificado por la clave.
Efecto sobre nuestro set: Elimina del conjunto el elemento o elementos seleccionados.

SRANDMEMBER: Devuelve uno o varios elementos aleatorios del conjunto especificado por la clave sin eliminarlos.

Efecto sobre nuestro set: No modifica el conjunto; sólo devuelve el elemento o elementos seleccionados.

```
127.0.0.1:6379> srandmember myset
"second"
127.0.0.1:6379> smembers myset
1) "third"
2) "first"
3) "second"
127.0.0.1:6379> spop myset
"second"
127.0.0.1:6379> smembers myset
1) "third"
2) "first"
```

redis> SRANDMEMBER myset

Observamos un elemento aleatorio del conjunto utilizando el comando SRANDMEMBER sin eliminarlo.

redis> SMEMBERS myset

Al consultar los elementos en el conjunto con el comando SMEMBERS, podemos verificar que SRANDMEMBER no modifica el conjunto.

redis> SPOP myset

Observamos un elemento aleatorio del conjunto utilizando el comando SPOP. Este comando sí elimina el elemento del conjunto.

redis> SMEMBERS myset

Al consultar los elementos en el conjunto con el comando SMEMBERS, podemos confirmar que SPOP ha eliminado el elemento del conjunto.

Redis Sorted Sets

Pregunta: ¿Cómo podemos saber los scores que tienen nuestros elementos asignados?

```
127.0.0.1:6379> zrange mysortedset 0 -1
1) "first"
2) "second"
3) "third"
127.0.0.1:6379> zrange mysortedset 0 -1 withscores
1) "first"
2) "1"
3) "second"
4) "2"
5) "third"
6) "3"
127.0.0.1:6379> zscore mysortedset first
"1"
127.0.0.1:6379> zscore mysortedset second
"2"
127.0.0.1:6379> zscore mysortedset third
"3"
127.0.0.1:6379> █
```

Tenemos un conjunto ordenado "mysortedset" con los siguientes elementos y sus scores:

first: 1
second: 2
third: 3

Para saber los scores de nuestros elementos, utilizamos el comando **ZRANGE**:

redis> ZRANGE mysortedset 0 -1 WITHSCORES

Este comando devuelve todos los elementos del conjunto ordenado mysortedset junto con sus scores.

0 -1: Especifica el rango de elementos. Aquí, 0 indica el primer elemento y -1 indica el último elemento, por lo que se incluirán todos los elementos del conjunto ordenado.

El comando **ZSCORE** devuelve el score asociada a un miembro en un conjunto ordenado.

Por ejemplo, queremos devolver el score asociada al miembro "first" del conjunto ordenado "mysortedset".

Para ello, podemos usar el siguiente comando:

redis> ZSCORE mysortedset first

Este comando devolverá el score 1, que es el score asociada al miembro "first".

Si queremos devolver el score asociada al miembro "second" del conjunto ordenado "mysortedset".

redis> ZSCORE mysortedset second

Este comando devolverá el score 2, que es el score asociada al miembro "second".

Si queremos devolver el score asociada al miembro "third" del conjunto ordenado "mysortedset".

redis> ZSCORE mysortedset third

Este comando devolverá el score 3, que es el score asociada al miembro "third".

He adjuntado a esta práctica todos los scripts de Python y Lua: hello.lua, multiply.lua, helloLUA.py y worcount.py.

También he adjuntado a esta práctica el script de Python lorem_text_generator.py que genera un archivo de texto llamado lorem_text.txt.

2. LUA

1. Implementar un script en LUA llamado hello.lua que:

```
127.0.0.1:6379> set greetkey "Hello"
OK
127.0.0.1:6379> get greetkey
"Hello"
```

Con el comando **SET greetkey "Hello"** asignamos el valor string "Hello" a la clave greetkey en Redis.



```
GNU nano 2.9.3 hello.lua
-- Retrieve the content stored in Redis associated with the key 'greetkey'
local greet = redis.call("get", KEYS[1])

-- Concatenate the retrieved content with the provided argument "Merve"
local result = greet .. " " .. ARGV[1]

-- Return the concatenated result
return result

[ 8 líneas leídas ]
^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Tex ^J Justificar ^C Posición
^X Salir ^R Leer fich. ^\ Reemplazar ^U Pegar txt ^T Ortografía ^_ Ir a línea
```

En el fichero hello.lua definimos una variable llamada greet que almacena el valor asociado a la clave greetkey en Redis.

Luego, concatenamos la variable greet con el argumento pasado como parámetro para formar un saludo en la variable result.

Finalmente, se devuelve el valor almacenado en la variable result.

```
osboxes@mv-bigdata:~/redis-5.0.8$ nano hello.lua
osboxes@mv-bigdata:~/redis-5.0.8$ redis-cli --eval hello.lua greetkey , "Merve"
"Hello Merve"
```

\$ redis-cli --eval hello.lua greetkey , "Merve"

Al ejecutar este comando, ejecutaremos el script Lua almacenado en "hello.lua" utilizando la opción --eval en Redis.

El script Lua devolverá el valor asociado a la clave "greetkey" de la base de datos Redis y lo concatenará con la cadena "Merve".

Finalmente, el resultado será devuelto por el script Lua y mostrado en la salida del comando como "Hello Merve"

2. Implementar un script en LUA llamado multiply.lua que:

```
127.0.0.1:6379> set multiply 3
OK
127.0.0.1:6379> get multiply
"3"
```

Con el comando **SET multiply 3** asignamos el valor integer 3 a la clave multiply en Redis.

```
GNU nano 2.9.3 multiply.lua

-- Retrieve the value stored in Redis associated with the key specified by KEYS[1]
local value = redis.call("get", KEYS[1])

-- Multiply the retrieved value by the provided number (ARGV[1])
local result = value * ARGV[1]

-- Return the result
return result
```

^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Text ^J Justificar ^C Posición
^X Salir ^R Leer fich. ^\ Reemplazar ^U Pegar txt ^T Ortografía ^_ Ir a línea

En el fichero multiply.lua definimos una variable llamada value que almacena el valor asociado a la clave multiply en Redis.

Luego, multiplicamos el valor asignado por el número proporcionado.

Finalmente, se devuelve el valor almacenado en la variable result.

```
osboxes@mv-bigdata:~/redis-5.0.8$ nano multiply.lua
osboxes@mv-bigdata:~/redis-5.0.8$ redis-cli --eval multiply.lua multiply , 5
(integer) 15
```

\$ redis-cli --eval multiply.lua multiply , 5

Al ejecutar este comando, ejecutaremos el script Lua almacenado en "multiply.lua" utilizando la opción --eval en Redis.

El script Lua devolverá el valor asociado a la clave "multiply" de la base de datos de Redis, lo multiplicará por 5 (el argumento proporcionado) y devolverá el resultado.

En este caso, el producto sería 15, resultado de multiplicar el valor asociado a la clave "multiply" (que es 3) por el argumento proporcionado (que es 5).

3. Python

1. Implementar un script en Python llamado hello.py que:

```
127.0.0.1:6379> set greetkey "Hello"
OK
127.0.0.1:6379> get greetkey
"Hello"
```

Con el comando **SET greetkey "Hello"**, ya hemos asignado el valor de cadena "Hello" a la clave greetkey en Redis.

```
GNU nano 2.9.3                                hello.py
import redis

# Connect to Redis
r = redis.Redis(host='localhost', port=6379, db=0)

def get_greeting(key, name):
    # Retrieve the value associated with the key from Redis
    value = r.get(key)

    print("{} {}".format(value.decode('ascii'), name))

if __name__ == "__main__":
    import sys
    # Get the key and name from the command-line arguments
    key = sys.argv[1]
    name = sys.argv[2]

    # Call the function to retrieve the greeting and print it
    get_greeting(key, name)
```

^G Ver ayuda ^O Guardar ^W Buscar ^K Cortar Texto ^J Justificar ^C Posición
^X Salir ^R Leer fich. ^\ Reemplazar ^U Pegar txt ^T Corrector ^ Ir a línea

Importamos el módulo redis en el script para conectarse a una base de datos Redis en localhost (127.0.0.1) y puerto 6379, seleccionando la base de datos 0.

La función get_greeting recibe una clave y un nombre como argumentos, recupera el valor asociado a la clave de Redis y devuelve un mensaje de saludo.

El bloque del script (__main__) obtiene la clave y el nombre de la línea de comandos, llama a la función get_greeting con esos argumentos y muestra el mensaje de saludo resultante.

```
osboxes@mv-bigdata:~/redis-5.0.8$ nano hello.py
osboxes@mv-bigdata:~/redis-5.0.8$ python3 hello.py "greetkey" "Merve"
Hello Merve
osboxes@mv-bigdata:~/redis-5.0.8$
```

\$ python3 hello.py "greetkey" "Merve"

Con este comando ejecutamos el script de Python hello.py con los argumentos de la línea de comandos "greetkey" y "Merve".

El script recupera de Redis el valor asociado a la clave "greetkey" de Redis y lo combina con "Merve" para imprimir el mensaje "Hello Merve"

En resumen, el script funciona correctamente y muestra el saludo.

2. Implementar un script en Python llamado helloLUA.py que:

```
127.0.0.1:6379> set greetkey "Hello"
OK
127.0.0.1:6379> get greetkey
"Hello"
```

Con el comando **SET greetkey "Hello"**, ya hemos asignado el valor de cadena "Hello" a la clave greetkey en Redis.

```
GNU nano 2.9.3                                helloLUA.py

import redis

# Connect to Redis
r = redis.Redis(host='localhost', port=6379, db=0)

def execute_lua_script(redis_key, name):
    # Read Lua script from hello.lua file
    with open('hello.lua', 'r') as file:
        lua_script = file.read()

    # Upload Lua script to Redis cache and get the registered script
    script = r.register_script(lua_script)

    # Call the uploaded script using Redis key and ARGV as parameters
    result_bytes = script(keys=[redis_key], args=[name])

    # Decode byte string to regular string using ascii encoding
    result_string = result_bytes.decode('ascii')

    return result_string

if __name__ == "__main__":
    import sys

    redis_key = sys.argv[1]
    name = sys.argv[2]

    # Call the function to execute the Lua script and print the result
    result = execute_lua_script(redis_key, name)
    print(result)
```

Ver ayuda	Guardar	Buscar	Cortar Texto	Justificar	Posición
Salir	Leer fich.	Reemplazar	Pegar txt	Corrector	Ir a línea

Importamos el módulo redis en el script para conectarse a una base de datos Redis en localhost (127.0.0.1) y puerto 6379, seleccionando la base de datos 0.

La función execute_lua_script recibe dos parámetros: redis_key (vamos a usar en el script Lua) y name (un argumento a pasar al script Lua). La función lee el script Lua hello.lua, lo sube a la caché de Redis (mediante register_script) y lo ejecuta con los argumentos proporcionados.

El bloque del script (__main__) obtiene la clave y el nombre de la línea de comandos, llama a la función execute_lua_script con esos argumentos y muestra el mensaje de saludo resultante.

```
osboxes@mv-bigdata:~/redis-5.0.8$ nano helloLUA.py
osboxes@mv-bigdata:~/redis-5.0.8$ python3 helloLUA.py "greetkey" "Merve"
Hello Merve
```

\$ python3 helloLUA.py “greetkey” “Merve”

Con este comando ejecutamos el script de Python helloLUA.py con los argumentos de la línea de comandos "greetkey" y "Merve".

El script se conecta al servidor Redis y lee el script Lua del archivo hello.lua.

El script Lua cargado en la caché de Redis se ejecuta en el servidor Redis con la clave "greetkey" y la cadena "Merve" como argumentos.

El resultado de la ejecución del script Lua, se imprime "Hello Merve" en la consola.

3. Implementar un script en Python llamado wordCount.py que:

Primero, creamos un fichero lorem_text.txt utilizando la librería lorem_text en Python. Luego, en este script de Python wordcount.py, extraemos las palabras del archivo y las convertimos todas a minúsculas. Después, eliminamos todos los símbolos de puntuación, como: (\n, \r, . , ? , etc). A continuación, establecemos una conexión a Redis y creamos un conjunto ordenado (sorted set) en Redis. En este conjunto, insertamos las palabras y obtenemos el ranking de las 10 palabras con mayor frecuencia en el texto. Finalmente, mostramos el ranking por pantalla usando el comando:

\$ python3 wordcount.py lorem_text.txt

```
GNU nano 2.9.3 wordcount.py

import sys
import re
import redis
from collections import Counter

# Function to clean text and extract words
def extract_words(file_path):
    with open(file_path, 'r') as file:
        text = file.read().lower()
        # Remove punctuation and special characters
        cleaned_text = re.sub(r'^a-zA-Z\s', '', text)
        # Split text into words
        words = cleaned_text.split()
    return words

# Function to connect to Redis and generate word ranking
def generate_word_ranking(words):
    # Connect to Redis
    r = redis.Redis(host='localhost', port=6379, db=0)
    # Create a sorted set in Redis
    sorted_set_key = 'word_ranking'
    for word, count in Counter(words).items():
        # Insert words into the sorted set with their count as score
        r.zadd(sorted_set_key, {word: count})

    # Retrieve the ranking of the 10 words with more occurrences
    ranking = r.zrevrange(sorted_set_key, 0, 9, withscores=True)
    return ranking

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python3 wordcount.py <file_path>")
        sys.exit(1)
```

```
def generate_word_ranking(words):
    # Connect to Redis
    r = redis.Redis(host='localhost', port=6379, db=0)
    # Create a sorted set in Redis
    sorted_set_key = 'word_ranking'
    for word, count in Counter(words).items():
        # Insert words into the sorted set with their count as score
        r.zadd(sorted_set_key, {word: count})

    # Retrieve the ranking of the 10 words with more occurrences
    ranking = r.zrevrange(sorted_set_key, 0, 9, withscores=True)
    return ranking

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python3 wordcount.py <file_path>")
        sys.exit(1)

    file_path = sys.argv[1]
    words = extract_words(file_path)
    ranking = generate_word_ranking(words)

    # Print the ranking on the screen
    print("Word Ranking:")
    for rank, (word, count) in enumerate(ranking, start=1):
        print(f"{rank}. {word.decode('utf-8')}: {int(count)}")
```

^G Ver ayuda	^O Guardar	^W Buscar	^K Cortar Texto	^J Justificar	^C Posición
^X Salir	^R Leer fich.	^_\ Reemplazar	^U Pegar txt	^T Corrector	^_ Ir a línea

```
osboxes@mv-bigdata:~/redis-5.0.8$ nano wordcount.py
osboxes@mv-bigdata:~/redis-5.0.8$ python3 wordcount.py lorem_text.txt
```

Word Ranking:

1. adipisci: 21
2. enim: 20
3. vero: 18
4. iste: 18
5. deleniti: 18
6. numquam: 17
7. tempore: 16
8. praesentium: 16
9. labore: 16
10. facilis: 16