Bibliotecas de Lua

Tiempo estimado: 20min

Tras abordar cómo ejecutar *scripts* en instancias de bases de datos Redis, vamos a presentar algunas de las bibliotecas que vienen incorporadas de fábrica en Redis.

Comenzamos la lección presentando el concepto de biblioteca. A continuación, describimos las bibliotecas con las que realizar operaciones matemáticas, operaciones con cadenas, con tablas y con datos de tipo JSON.

Al finalizar la lección, el estudiante sabrá:

- Cómo utilizar math para realizar operaciones matemáticas.
- Cómo utilizar string para manipular cadenas.
- Cómo utilizar table para manipular tablas.
- Cómo utilizar cjson para seriar y deseriar datos JSON.

Introducción

Una biblioteca (*library*) es un conjunto de objetos, como variables y funciones, reutilizables. El intérprete de Lua integrado en Redis carga las bibliotecas base, bitop, cjson, cmsgpack, math, string, struct y table, permitiéndonos así su uso.

A continuación, presentamos las más utilizadas en Redis.

Biblioteca math

La biblioteca math proporciona funciones y variables para realizar operaciones matemáticas como, por ejemplo, el cálculo del valor absoluto, del seno, del coseno, de la tangente, el redondeo, etc.:

- math.abs(n). Calcula el valor absoluto de un número.
- math.acos(n). Calcula el arco coseno de un número.
- math.asin(n). Calcula el arco seno de un número.
- math.atan(n). Calcula el arco tangente de un número.
- math.ceil(n). Calcula el valor entero superior a un número dado.
- math.cos(n). Calcula el coseno de un número.
- math.deg(a). Convierte el ángulo indicado de radianes a grados.
- math.exp(n). Calcula e^n.
- math.floor(n). Calcula el valor entero inferior a un número dado.
- math.fmod(x, y). Calcula el resto de la división de x por y.
- math.log(x [, base]). Calcula el logaritmo de x en la base dada.

  Si no se indica base, se asumirá e.
- math.max(x, ...). Devuelve el valor mayor de los indicados.
- math.maxinteger. Indica el mayor valor entero.
- math.min(x, ...). Devuelve el valor mínimo de los indicados.
- math.mininteger. Indica el mínimo valor entero.
- math.modf(x). Calcula las partes integral y fracional de x.

- math.pi. Valor π.
- math.rad(a). Convierte el ángulo indicado de grados a radianes.
- math.random(). Calcula un número aleatorio mayor o igual que 0 y menor que 1.
- math.random([min,] max). Calcula un número aleatorio entre dos números.

  Cuando no se indica el mínimo, se asume 1.
- math.randomseed(x). Fija la semilla para el cálculo de los números aleatorios generados por math.random().
- math.sin(n). Calcula el seno de un número.
- math.sqrt(n). Calcula la raíz cuadrada de un número.
- math.tan(n). Calcula la tangente de un número.
- math.tointeger(txt). Devuelve el número representado. En otro caso, nil.
- math.type(val). Devuelve el tipo de número en texto: integer si es un valor entero; float si es real.
- math.ult(m, n). Devuelve si m es menor que n, si se considera los valores como números enteros sin signo.

Biblioteca string

La biblioteca string contiene funciones para manipular cadenas como, por ejemplo, comprobar si una cadena sigue un determinado patrón, formatear una cadena según un patrón, etc.:

- string.byte(ch). Devuelve el código numérico de un carácter dado.
- string.byte(str, start [, end]). Devuelve los códigos numéricos de los caracteres de la cadena dada.
- string.char(...). Devuelve la cadena correspondiente a los caracteres dados por sus códigos numéricos.
- string.find(str, pattern). Devuelve dos valores: el primero indica la posición inicial donde comienza la primera aparición del patrón; y el segundo, donde termina. Si el patrón no aparece, devuelve un único valor, nil.
- string.find(str, pattern, init). Ídem a la anterior, pero comenzando a partir de la posición indicada.
- string.format(format, ...). Formatea una cadena con los argumentos dados.

  La cadena de formato debe seguir las reglas de la función sprintf() de C.
- string.gmatch(str, pattern). Devuelve un iterador para recorrer las subcadenas que cumplen con el patrón dado.
- string.gsub(str, pattern, repl [, n]). Reemplaza las apariciones del patrón por el valor repl.

  n indica el número máximo de reemplazos a realizar.
- string.len(str). Calcula la longitud de la cadena.
- string.lower(str). Devuelve la cadena en minúsculas.
- string.match(str, pattern [, init]). Devuelve la primera subcadena que cumple el patrón.

  init indica a partir de dónde comenzar la búsqueda.
- string.rep(str, n). Concatena una cadena consigo misma un determinado número de veces.
- string.reverse(str). Devuelve la cadena en orden inverso.
- string.sub(str, start [, end]). Devuelve una subcadena.
- string.upper(str). Devuelve la cadena en mayúsculas.

Patrones

Un patrón (*pattern*) describe un formato de texto. string permite trabajar con patrones muy útiles a la hora de realizar búsquedas dentro de cadenas o comprobar si una cadena cumple un determinado patrón o formato.

Los patrones son cadenas con caracteres especiales que representan uno o más caracteres adicionales. Estos caracteres especiales son:

- . Representa un carácter cualquiera.
- %a. Representa una letra cualquiera.
- %c. Representa un carácter de control cualquiera.
- %d. Representa un dígito cualquiera.
- %g. Representa un carácter imprimible cualquiera, a excepción del espacio en blanco.
- %l. Representa una letra cualquiera en minúscula.
- %p. Representa un carácter de puntuación cualquiera.
- %s. Representa un carácter de espacio cualquiera.
- %u. Representa una letra cualquiera en mayúscula.
- %w. Representa un carácter alfanumérico cualquiera.
- %x. Representa un carácter hexadecimal cualquiera.
- [caracteres]. Representa cualquiera de los caracteres indicados.
- [^caracteres]. Representa cualquiera de los caracteres no indicados.
- *. Indica que el carácter anterior debe aparecer cero, una o más veces.
- +. Indica que el carácter anterior debe aparecer una o más veces.
- ?. Indica que el carácter anterior debe aparecer cero o una vez.

Comprobación de patrones

Cuando deseamos comprobar si un valor cumple con un determinado patrón, podemos utilizar las funciones find(), match() y gmatch(). La diferencia es sutil, pero importante. find() devuelve la posición donde aparece la primera aparición del patrón. Por su parte, match() devuelve el texto de la primera aparición del patrón. Mientras que gmatch() devuelve un iterador para recorrer las distintas apariciones del patrón.

Esto se ve mejor con unos ejemplos. Comencemos con find():

```
> string.find("esto es un texto de ejemplo", ".t.")
2       4
> string.find("esto es un texto de ejemplo", "noexiste")
nil
>
```

Cuando el patrón aparece en la cadena, find() devuelve la posición de la primera aparición y como segundo valor la posición donde finaliza la aparición. En cambio, cuando no hay ninguna coincidencia, devuelve nil.

Ahora, veamos lo mismo, pero con match():

```
> string.match("esto es un texto de ejemplo", ".t.")
sto
> string.match("esto es un texto de ejemplo", "noexiste")
nil
>
```

Con match(), se obtiene el texto de la primera aparición del patrón.

Cuando la cadena presenta varias apariciones, se puede utilizar gmatch() para iterar por cada una de ellas:

```
> for txt in string.gmatch("esto es un texto de ejemplo", ".t.") do
>> print(txt)
```



```
>> end
sto
 te
xto
>
```

Sustitución de patrones

Por su parte, la función gsub() permite reemplazar apariciones de un patrón por un texto dado. Ejemplo:

```
> string.gsub("esto es un texto de ejemplo", ".t.", "XXX")
eXXX es unXXXXXX de ejemplo     3
>
```

Formateo

El formateo (*formating*) es la operación mediante la cual damos formato a una lista de valores según un determinado patrón o modelo. La cadena que contiene el formato se conoce formalmente como formato (*format*). Esta cadena debe seguir las reglas de la función sprintf() de C. Debe contener texto literal y especificadores que indican cómo dar formato a cada valor dentro del patrón.

Los especificadores de formato más utilizados son:

- %i. Como número entero.
- %f. Como número real.
- %s. Como un texto.

He aquí un ejemplo ilustrativo:

```
> string.format("mi nombre es %s y tengo %i años", "Tom Petty", 65)
mi nombre es Tom Petty y tengo 65 años
>
```

Biblioteca table

La biblioteca table proporciona funciones para la manipulación de tablas, por ejemplo, para añadir o suprimir elementos:

- table.concat(list, [sep]). Devuelve una cadena con los elementos de la lista concatenados uno detrás de otro, separándolos con la cadena indicada.
- table.concat(list, sep, start [, end]). Ídem a la anterior, pero comienza la concatenación en la posición start y finaliza en end.
- table.insert(list, value). Añade un valor al final de la lista.
- table.insert(list, pos, value). Inserta un valor en la posición indicada, desplazando a la derecha los elementos que sea necesario.
- table.pack(...). Devuelve una lista con los elementos indicados.
- table.remove(list). Suprime y devuelve el último elemento de la lista.
- table.remove(list, pos). Suprime y devuelve el elemento de la posición indicada.
- table.sort(list). Ordena los elementos de la lista.
- table.sort(list, comp). Ordena los elementos de la lista usando la función indicada. Esta función se invocará con dos argumentos a comparar.
- table.unpack(list). Devuelve los elementos de una lista como si se pasará uno detrás de otro en una sentencia return.
- table.unpack(list, start [, end]). Ídem a la anterior, pero sólo de los elementos que se encuentran entre las posiciones start y end.

Las funciones más utilizadas son table.concat() para obtener la representación en forma de cadena de una lista:

```
> table.concat({"uno", "dos", "tres"}, ", ")
uno, dos, tres
```



```
>
```
También es común la inserción de elementos al final de una lista:

```
> table.concat(lista, ", ")
uno, dos, tres
> table.insert(lista, "cuatro")
> table.concat(lista, ", ")
uno, dos, tres, cuatro
>
```
O entre dos elementos de la lista:

```
> table.concat(lista, ", ")
uno, dos, tres, cuatro
> table.insert(lista, 2, "cinco")
> table.concat(lista, ", ")
uno, cinco, dos, tres, cuatro
>
```
La supresión también es frecuente:

```
> table.concat(lista, ", ")
uno, dos, tres, cuatro, cinco
> table.remove(lista)
cinco
> table.concat(lista, ", ")
uno, dos, tres, cuatro
> table.remove(lista, 2)
dos
> table.concat(lista, ", ")
uno, tres, cuatro
>
```
También es posible asignar los valores de una lista a varias variables mediante el operador de asignación múltiple:

```
> table.concat(lista, ", ")
uno, dos, tres, cuatro, cinco
> x, y, z = lista
> x
table: 0000000000431c50
> y
nil
> z
nil
> x, y, z = table.unpack(lista)
> x
uno
> y
dos
> z
tres
> table.concat(lista, ", ")
uno, dos, tres, cuatro, cinco
>
```
Cuando se indica una lista en una asignación múltiple, la lista se asigna a la variable que le corresponde por posición. Si deseamos asignar los elementos de la lista, podemos utilizar la función table.unpack(). Esta función devuelve los elementos de la lista, sin suprimirlos de ella, para que se asignen a varias variables.

No con tanta frecuencia, se ordena los elementos de una lista:

```
> table.concat(lista, ", ")
uno, dos, tres, cuatro, cinco
> table.sort(lista)
> table.concat(lista, ", ")
cinco, cuatro, dos, tres, uno
>
```
Desempaquetado de elementos

El desempaquetado de elementos (*element unpack*) es la operación mediante la que dada una lista de elementos, los devuelve mediante una operación return múltiple. Se utiliza principalmente cuando

deseamos pasar los elementos de una lista como argumentos de una función. Se lleva a cabo mediante la función table.unpack().

Por ejemplo, la operación math.min() espera un número variable de argumentos, su único parámetro es el parámetro de resto (...). Mediante table.unpack() podemos pasar los elementos de una lista como los argumentos de la función:

```
--math.min(2, 45, 101, 21, 203, 1)
math.min(table.unpack({2, 45, 101, 21, 203, 1}))
```

Librería cjson

La librería cjson proporciona funciones para seriar y deseriar valores JSON:

- cjson.encode(value). Serializa un valor Lua a una cadena en formato JSON.

  Los tipos de datos de Lua a seriar son: boolean, nil, number, string y table.
- cjson.decode(text). Deseria una cadena formateada como JSON a Lua.

