**Title:** Redis SSCAN

**Excerpt:** Redis has in-built collection types. Hence, it has introduced the SSCAN command to iterate over a given set. This command returns the array of elements and the updated cursor per call. It is a flexible command which allows users to filter the returned elements based on a pattern as well.

**Permalink:** redis-sscan

**Category:** Redis

# 

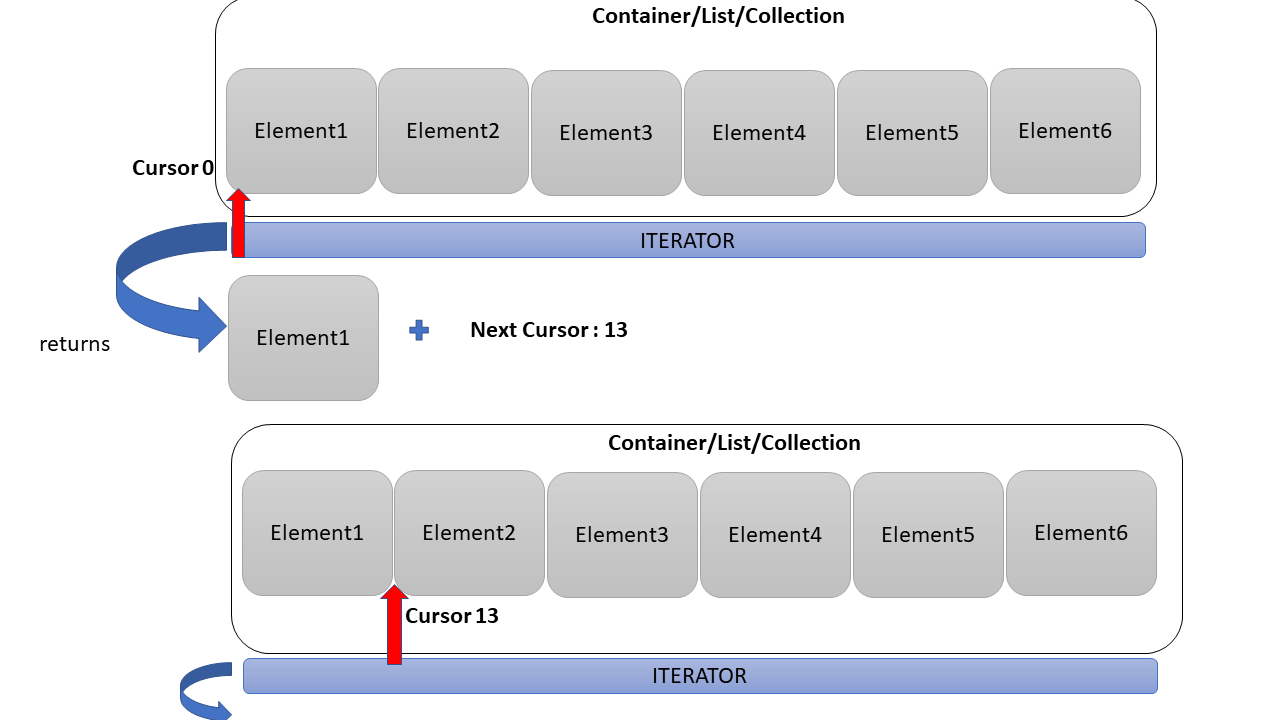
# 

# 

# Redis Iterator

Redis is a key-value store that comes with several in-built data types. It also includes collection types such as Sets, Hashes, and Sorted sets. Whenever you have collection types, it is a must to have a method to loop through the elements of a collection. Usually, in a programming context, we use iterators for this purpose.

Redis supports cursor-based iterators. These iterators can incrementally loop through the collection and return a limited number of elements. At the same time, it returns the updated cursor index/id as an integer to use in the next iteration. A cursor is a pointer maintained by the iterator to keep the new location after each iteration.



## The SCAN iterator

The SCAN command is the base for many other Redis iterators. This iterator doesn’t guarantee the number of elements returned in each loop. It might return 0 elements or all the elements in a given iteration. Also, it could produce the same element multiple times because the collection might change which is a drawback in SCAN iterator. The SCAN command returns two values in each execution. The first value is a reference id to the cursor and the second value is a collection of elements. It always starts from cursor 0. Several other iterators are derived from the SCAN iterator such as SSCAN, HSCAN, and ZSCAN. These commands have the same behavior but operate on different types of collections.

### The full iteration

Since the SCAN command doesn’t guarantee the number of elements returned in each iteration, the only way to notify that the iterator has been looped through the whole collection is to check the returned cursor reference id or index in each execution. If the returned cursor is 0, it indicates that all the elements have been iterated. It is called a full iteration.

# The SSCAN command

The SSCAN iterator is derived from the SCAN iterator. Hence, all the behaviors are extended from the SCAN command but it only iterates over set types.

## Syntax

| SSCAN set\_key cursor [MATCH pattern] [COUNT count] |
| --- |

***set\_key***: This is the key of the Set.

***cursor***: Reference id of the following cursor which is returned after each call.

The MATCH and COUNT are two optional parameters that can be used to filter out the returned elements based on a given pattern and limit the number of returned elements.

This command would return the next cursor and an array of elements. The returned cursor id can be passed to the SSCAN command in the next call.

## Example 01 - Iterate over a Small set

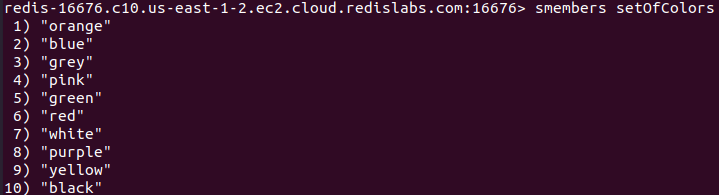
Let’s create a Redis set using the SADD command.

| **sadd** setOfColors red green **black** white **blue** pink grey orange purple yellow |
| --- |

The above command will create a new set *setOfColors* and add 10 members in an unordered manner. We can use the SMEMBERS command to verify the above operation.

| smembers setOfColors |
| --- |

As expected, the set *setOfColors* has been successfully created.

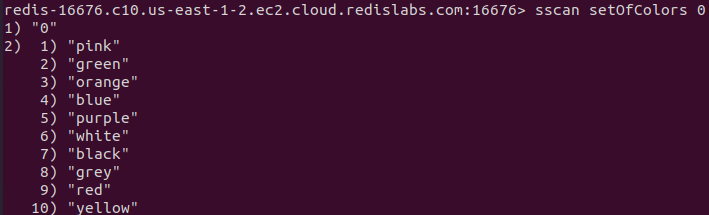


We can use the SSCAN command to iterate over the *setOfColors* set as shown in the following.

| sscan **set**OfColors 0 |
| --- |

As mentioned, the command should start with cursor 0. The set’s key has been passed too. Since the *setOfColors* contains only 10 members, the SSCAN command might return all the elements in the first iteration. Because the *setOfColors* is a comparatively small set.

Output:



As discussed, the command returned two values as the first one is 0. It means the above is a full iteration. There is no need to call the SSCAN again. All the ten members have been returned as the second return value of this command.

## Example 02 - Iterate over a big set

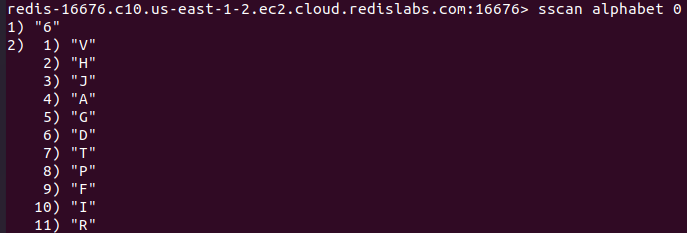
We will be creating a set with more than 20 members.

| sadd alphabet A B **C** **D** **E** F G H **I** J **K** L M **N** **O** P Q R S T U V W X Y Z |
| --- |

Let’s iterate over the *alphabet* set.

| sscan alphabet 0 |
| --- |

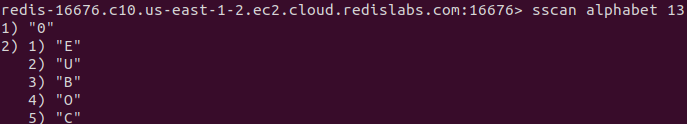
As usual, we have started with the cursor 0. Also, the set key has been passed as the *alphabet.* It only returned 11 members from the first iteration as shown in the following.



The next cursor is 6. Let’s call the SSCAN command again with the relevant arguments.



Now it returned another 10 members and the next cursor as 13. We have to call the SSCAN command again to finish the full iteration.



The returned cursor value is 0 which means the iterator has iterated through the whole set.

## Example 03 - Pattern matching with SSCAN

We can use the MATCH parameter to filter out the returned members based on a specified pattern. Let’s create a new set called *userNames* with some members.

| sadd userNames **john norek** pinku **juwana jorgia** noton derek desman **julia jumini** piter pinso demian |
| --- |

We will be calling the SSCAN command with the MATCH argument to fetch all the usernames starting with ‘ju’.

| sscan userNames 0 MATCH ju\* |
| --- |

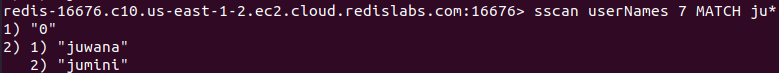
Output:



We got three usernames starting from ‘ju’ in the set. But it has returned only one in this iteration. Hence, we need to call the SSCAN command again with the updated cursor.

| sscan userNames 7 MATCH ju\* |
| --- |

Output:



The iterator has looped through all the elements in the set and we got three members matching our pattern.

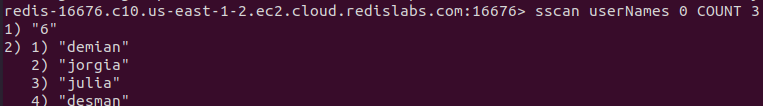
## Example 04 - Limit the number of elements per call

Redis SSCAN command returns around ten elements per call by default, but you can explicitly specify the number of elements to be returned per call. That can give a considerable performance boost to your application.

Let’s use the previous set and limit the number of elements returned per call to three. Usually, the returned element count will fluctuate around 3. We can specify the COUNT argument value to achieve this behavior.

| sscan userNames 0 COUNT 3 |
| --- |

Output:



Usually, it tends to return around 10 members, but now it's only 4 elements. The same happened with the following calls too.

| sscan userNames 6 COUNT 3 sscan userNames 1 COUNT 3 sscan userNames 7 COUNT 3 |
| --- |



This is very useful when you are dealing with large sets.

# Conclusion

Redis includes several collection types such as Sets, Hashes, and Sorted sets. Usually, the iterators can be used to loop through the collection types. The SCAN iterator is used to iterate over keys in a Redis database. It is a cursor-based iterator that returns two values per each call where the first is the updated cursor and the second is the array of elements. The SSCAN command has the same behaviors as the SCAN command but is specific to Set types. It can scan through a set based on a pattern too. Also, the command is capable of limiting the number of returned elements per call.