**Title:** Redis BRPOPLPUSH

**Excerpt:** Redis has several in-built blocking commands which operate on lists. The BRPOPLPUSH is one of the useful commands that enables popping elements from the tail of a source list and pushing them into the head of another list. This command holds the connection to the Redis server when no elements are available in the source list.

**Permalink:** redis-brpoplpush

**Category:** Redis

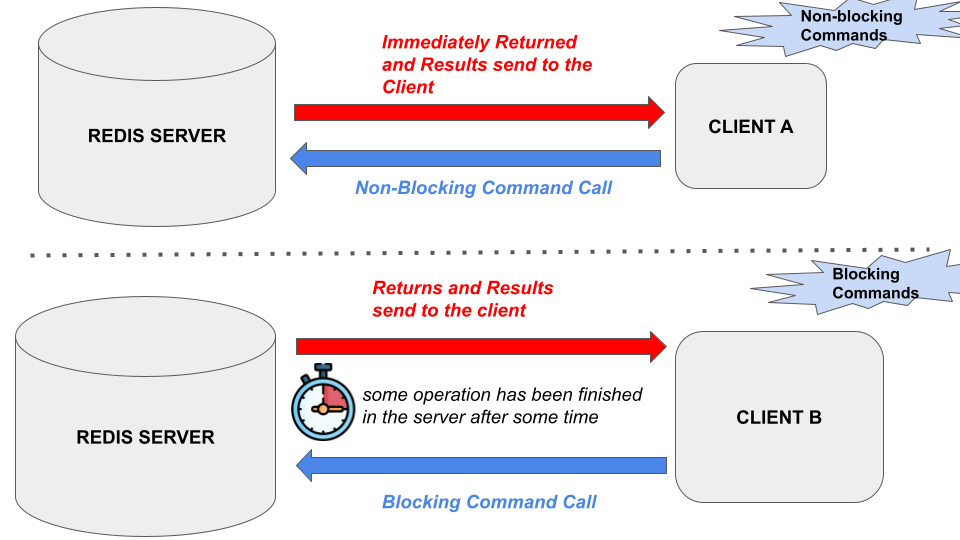
# 

# 

# 

# Introduction to Redis Blocking Commands

Redis comes with tens of in-built commands to operate on different data structures such as lists, hashes, sets, etc. Most of the commands perform a specific type of job and return immediately. In addition, a set of commands are available to use with blocking enabled. The blocking operations hold the connection to the Redis server until an external event finishes. The great thing about blocking commands is that they do not block the whole Redis server, instead they block only the calling client connection.



Redis has several blocking commands which operate on list data structures. The LPOP command is used to remove an element from the head of a given list. It will remove the head element from the list and immediately return and the removed element will reach the client. Whenever no elements are present in the list, the command will return immediately with a nil value. Its blocking variant is the BLPOP command which will hold the connection to the server whenever no elements are present in the list. The BLPOP command will wait until at least an element appears in the list to return. There are several blocking commands available to use with lists. In this guide, we will be focusing on the BRPOPLPUSH command.

# Redis BRPOPLPUSH Command

The BRPOPLPUSH command is the blocking variant of the RPOPLPUSH command. Hence, the BRPOPLPUSH command behaves the same as the RPOPLPUSH command when one or more elements are available in the source list. Whenever no single element is in the list, the BRPOPLPUSH command holds the connection to the Redis server until an element is available in the source list.

## Syntax:

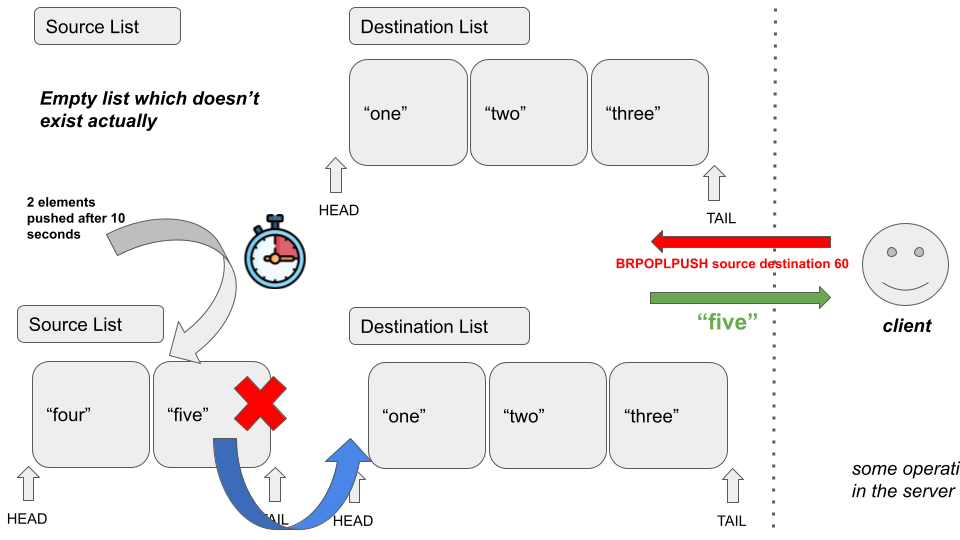
| BRPOPLPUSH source\_list destination\_list **timeout** |
| --- |

***source\_list:*** The list that elements will be removed from

***destination\_list:*** The list that the elements will be pushed into

***timeout:*** The time to block the connection and keep it alive.

The BRPOPLPUSH command will remove an element from the tail of the *source\_list* first*. Then* the element will be pushed near the head of another *destination\_list*. Whenever the source is empty it will hold the connection as shown in the following.



This command has a constant time complexity and it has been deprecated from Redis version 6.2.0. It is recommended to use the *BLMOVE* command with *RIGHT* and *LEFT* arguments. The BRPOPLPUSH command returns the element being popped from the source and pushed into the destination. The output will be a null value if the timeout is reached.

# Example - Messaging Server with Reliability Queues

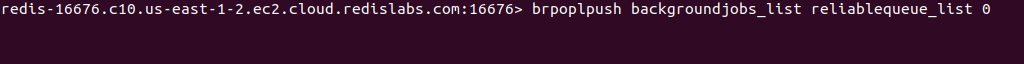
Let’s assume that a messaging server is using reliability queues to store tens of background jobs for a given application. The server-side list has been used to store the IDs of each background job. Several clients are consuming the list by fetching background jobs from the tail.

The biggest drawback of this process is if there is a network or the client application failure, the returned piece will be lost before the processing at the client-side. Hence, a separate reliable queue is needed to store the popped element from the source list while returning it to the client. Furthermore, the source list might be empty when there are no background jobs produced by publishers. Hence, we need a way that client connection will be held and alive until a background job is available at the source list. The BRPOPLPUSH command is the ideal candidate to achieve this.

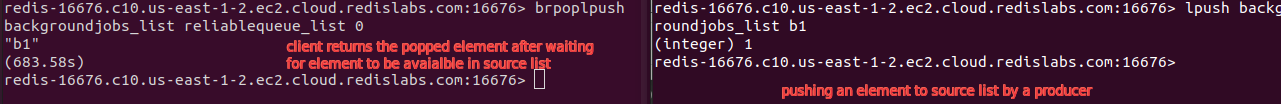
Let’s make a call to the Redis server with the BRPOPLPUSH command as shown in the following. The both source and destination keys do not exist.

| brpoplpush backgroundjobs\_list reliablequeue\_list 0 |
| --- |

We have specified the timeout argument to 0 which means the BRPOPLPUSH operation will be blocked forever until you terminate it manually. Now it is waiting for at least a single background job to be available in the list stored at key *backgroundjobs\_list*.



Next, we will be pushing a background job to the *backgroundjobs\_list* list as shown in the following. Let’s use the LPUSH command to push a single background job with the id of *“b1”.*



As expected the “b1” element has been returned to the client who held the connection until an element is available in the source list *backgroundjobs\_list.* Let’s inspect the destination queue *reliablequeue\_list*. Here we are using the *LRANGE* command as shown in the following.

| lrange reliablequeue\_list 0 10 |
| --- |



The destination queue contains the “b1” element that is popped from the source list. This command guarantees that the elements are stored in another queue without losing them when a client crashes. The BRPOPLPUSH command can be used as a circular list when both the source and destination are the same.

# Conclusion

To summarize, Redis has tens of built-in commands operating on different data types. Most of the commands are non-blocking where the command will be returned immediately after the job is finished. As discussed, several in-built blocking commands are available to operate on list types. The BRPOPLPUSH command is one of them. As you already know, it is the blocking variant of the RPOPLPUSH command. It behaves the same as the RPOPLPUSH command when at least a single element is stored in the source list. Whenever the source list doesn’t exist, the BRPOPLPUSH command will wait for a specified timeout until an element is present in the source list. As stated, since it has constant time complexity, very efficient to use with reliable queues in the messaging servers.