Interview Questions on Redis for Developers

Nitish Prabhu Follow
Jun 9, 2018 · 7 min read

1. What is Redis?

Redis is an open source, in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries.

2. What is in-memory Database?

The in-memory database is a database where it keeps the dataset in RAM. Means that for every interaction with database, you will only access the Main memory; No DISK operations involved during this interaction. Hence the operation will be faster as it directly aceess main memoery instead of any disk operation.

3. Isn't the Redis data lost if there's a system crash?

To avoid data loss in case of system crash Redis provides persistence by Snapshotting & AOF(Append-only files).

4. How Redis achieve persistence by Snapshotting?

Snapshot takes the data as it exists at one moment in time & writes it to the disk, will be written to the file referenced as "**dbfilename**".

There are 5 steps to initiate snapshots.

a. BGSAVE: When redis client initiate BGSAVE, redis will create a fork so that child process will write a the snapshot to the disk where master interacts with commands.

b. SAVE: When redis client initiate SAVE, redis will stop responding any commands until snapshot completes

c. From Configuration Files:

Example: Configuration as SAVE 60 10000

BGSAVE will be called if 10000 writes occure within 60 seconds

d. SHUTDOWN: When redis client sends a SHUTDOWN, redis will perform SAVE operations, blocks all operations, shutdowns later.

e. Sync with other Redis: If a Redis server connects to another server, it issues SYNC, then master Redis will start BGSAVE.

5. How redis achive persistence by AOF?

AOF copies incoming write command to disk as they happen. It provides file synch options as follows.

a. always—Every writes to redis, writes to disk which usually affect redis' performance.

b. everysec—Writes to disk every second

c. no—Let OS control syncing to disk

6. Files will keep on growing as long as AOF executed. Won't it consume more memory? What is the solution provided by redis?

Redis provides command BGREWRITEAOF which rewrites the existing AOF with the shortest sequence of commands needed to rebuild the current dataset in memory.

7. Redis String data-type.

Redis Strings are binary safe, this means that a Redis string can contain any kind of data, for instance a JPEG image or a serialized Ruby object.

A String value can be at max 512 Megabytes in length.

8. Explain SET & GET command.

SET command used to set a value to the key and GET command to get a value of the key as follows, of data-type STRING.

127.0.0.1:6379> SET TEST_KEY TEST_VALUE

```
OK [TEST_VALUE is being set to TEST_KEY]
```

```
127.0.0.1:6379> GET TEST_KEY
```

```
"TEST_VALUE"
```

SET Command with NX|XX arguments

NX argument indicates that SET a value to the key if the key is not exist;

XX argument indicates that SET a value to the key only if the key already exists;

Example:

NX: key TEST_KEY is already exist; If we try to update TEST_KEY value, it will not accept and returns (nill) status

127.0.0.1:6379> SET TEST_KEY TEST_VALUE_UPDATED NX

```
(nil)
```

```
127.0.0.1:6379> GET TEST_KEY
```

```
"TEST_VALUE"
```

Value of TEST_KEY remain same;

XX: key TEST_KEY is alrady exists; Hence so, when we try to udpate TEST_KEY value, its updated and ack OK

127.0.0.1:6379> SET TEST_KEY TEST_VALUE_XX_UDPATED XX

```
OK
```

```
127.0.0.1:6379> GET TEST_KEY
```

```
"TEST_VALUE_XX_UDPATED"
```

9. Redis List data-type.

Internal Implementation: Linked-List

Redis Lists are simply lists of strings, sorted by insertion order. It is possible to add elements to a Redis List pushing new elements on the head (on the left) or on the tail (on the right) of the list.

The max length of a list is 2^{32}—1 elements (4294967295, more than 4 billion of elements per list).

10. Explain List operations.

LPUSH—Inserts a new element on the head

RPUSH—Inserts a new element on the tail

LPOP—Removes an element from the head

RPOP—Removes an element from tail

LRANGE—Prints an element from the specified position

Example:

```
127.0.0.1:6379> LPUSH A 1
```

```
127.0.0.1:6379> RPUSH A 2
```

```
127.0.0.1:6379> RPUSH A 3
```

```
127.0.0.1:6379> RPUSH A 4
```

```
127.0.0.1:6379> LRANGE A 0 -1
```

```
1) "1"
```

```
2) "2"
```

```
3) "3"
```

```
4) "4"
```

```
127.0.0.1:6379> LPOP A
```

```
"1" ["1" element value is removed]
```

```
127.0.0.1:6379> LRANGE A 0 -1
```

```
1) "2"
```

```
2) "3"
```

```
3) "4"
```

```
127.0.0.1:6379> RPOP A
```

```
"4" ["4" element value is removed]
```

```
127.0.0.1:6379> LRANGE A 0 -1
```

```
1) "2"
```

```
2) "3"
```

11. Explain Blocking operations on lists.

This is a case when you want to do remove operations like RPOP/LPOP, but the list is already empty. In this case, remove operation will return null does nothing. In this case a consumer is forced to wait some time and retry again with RPOP.

So Redis implements commands called BRPOP and BLPOP which are versions of RPOP and LPOP able to block if the list is empty: This will return to the caller only when a new element is added to the list, or when a user-specified timeout is reached.

Example: If we want pop the elements from the list A, which is empty right now;

```
127.0.0.1:6379> brpop A {timeout}
```

```
...
```

```
...
```

waits until new elements is added to list key A

timeout—No of seconds to wait until; 0 indicates wait forever

12. Redis SET data-type.

Redis Sets are an unordered collection of Strings. It is possible to add, remove, and test for existence of members in O(1)

The max number of members in a set is 2^{32}—1 (4294967295, more than 4 billion of members per set).

13. Explain Redis SET operations.

SADD—Adds a new element to the SET

SMEMBERS—To retrieve values from the KEY

SISMEMBER—To verify whether specified element is present in the SET

SPOP—To pop the element out from the SET

Example:

```
127.0.0.1:6379> SADD TEST_SET A
(integer) 1
127.0.0.1:6379> SADD TEST_SET B
(integer) 1
127.0.0.1:6379> SMEMBERS TEST_SET
1) "A"
2) "B"
127.0.0.1:6379> SISMEMBER TEST_SET A
(integer) 1 [1 indicates it's true; it's available]
127.0.0.1:6379> SPOP TEST_SET
```

```
"A"
```

```
127.0.0.1:6379> SMEMBERs TEST_SET
```

```
1) "B"
```

14. Explain SINTER & SUNION.

SUNION—Returns the members of the set resulting from the union of all the given sets.

SINTER—Returns the members of the set resulting from the intersection of all the given sets.

Example:

```
127.0.0.1:6379> SMEMBERS TEST_KEY_A
```

```
1) "1"
```

```
2) "2"
```

```
3) "3"
```

```
127.0.0.1:6379> SMEMBERS TEST_KEY_B
```

```
1) "2"
```

```
2) "6"
```

```
127.0.0.1:6379> SUNION TEST_KEY_A TEST_KEY_B
```

```
1) "1"
```

```
2) "2"
```

```
3) "3" [prints union of two sets]
```

```
4) "6"
```

```
127.0.0.1:6379> SINTER TEST_KEY_A TEST_KEY_B
```

```
1) "2" [print intersection of two sets]
```

15. Explain SUNIONSTORE & SINTERSTORE.

SUNIONSTORE—This command is equal to SUNION, but instead of returning the resulting set, it is stored in destination.

SINTERSTORE—This command is equal to SINTER, but instead of returning the resulting set, it is stored in destination.

Example:

127.0.0.1:6379> SUNIONSTORE TEST_KEY_UNION TEST_KEY_A TEST_KEY_B

```
(integer) 4

127.0.0.1:6379> SMEMBERS TEST_KEY_UNION

1) "1"

2) "2"

3) "3"

4) "6"

127.0.0.1:6379> SINTERSTORE TEST_KEY_INTER TEST_KEY_A TEST_KEY_B

(integer) 1

127.0.0.1:6379> SMEMBERS TEST_KEY_INTER

1) "2"
```

16. Redis Hashes data-type.

Redis Hashes are maps between string fields and string values, so they are the perfect data type to represent objects like user-details

Every hash can store up to 2^{32}—1 field-value pairs (more than 4 billion).

17. Explain HASH data-type operation.

HMSET—Sets the specified fields to their respective values in the hash stored at key.

Time complexity: O(N) where N is the number of fields being set.

HGET—Returns the value associated with field in the hash stored at key.

Time complexity: O(1)

HGETALL—Returns all fields and values of the hash stored at key.

Time complexity: O(N) where N is the size of the hash.

Example:

```
127.0.0.1:6379> HMSET TEST_HASH_KEY A 20
OK
127.0.0.1:6379> HMSET TEST_HASH_KEY B 10
OK
127.0.0.1:6379> HMSET TEST_HASH_KEY C 10
OK
127.0.0.1:6379> HGETALL TEST_HASH_KEY
1) "A"
2) "20"
3) "B"
4) "10"
5) "C"
6) "10"
127.0.0.1:6379> HMGET TEST_HASH_KEY A
```

```
1) "20"
```

18. Explain Redis Sorted sets.

Sorted sets are a data type which is similar to a mix between a Set and a Hash. Like sets, sorted sets are composed of unique, non-repeating string elements, so in some sense a sorted set is a set as well.

However while elements inside sets are not ordered, every element in a sorted set is associated with a floating point value, called the score.[in ascending order]

19. Explain redis Sorted Set operation.

ZADD—Adds all the specified members with the specified scores to the sorted set stored at key.

Time complexity: O(log(N)) for each item added, where N is the number of elements in the sorted set.

ZREM—Removes the specified members from the sorted set stored at key. Non existing members are ignored.

Time complexity: O(M*log(N)) with N being the number of elements in the sorted set and M the number of elements to be removed.

ZRANGE—Returns the specified range of elements in the sorted set stored at key.

Time complexity: O(log(N)+M) with N being the number of elements in the sorted set and M the number of elements returned.

ZRANGEBYSCORE—Returns all the elements in the sorted set at key with a score between min and max.

Time complexity: O(log(N)+M) with N being the number of elements in the sorted set and M the number of elements being returned. If M is constant (e.g. always asking for the first 10 elements with LIMIT), you can consider it O(log(N)).

127.0.0.1:6379> ZADD TEST_ZKEY 1 A

```
(integer) 1

127.0.0.1:6379> ZADD TEST_ZKEY 2 B

(integer) 1

127.0.0.1:6379> ZADD TEST_ZKEY 1 C

(integer) 1

127.0.0.1:6379> ZRANGE TEST_ZKEY 0 -1

1) "A"

2) "C"

3) "B"

127.0.0.1:6379> ZREM TEST_ZKEY B

(integer) 1

127.0.0.1:6379> ZRANGE TEST_ZKEY 0 -1

1) "A"

2) "C"

127.0.0.1:6379> ZRANGEBYSCORE TEST_ZKEY 1 2 {Here 1 is min rank, 2
is max rank- gets a data from rank of 1 to 2}

1) "A"
```