FullStack.Cafe - Kill Your Tech Interview

Q1: What is Redis? ☆

Topics: Redis

Answer:

Redis, which stands for **Re**mote **Di**ctionary **S**erver, is a fast, open-source, in-memory key-value data store for use as a database, cache, message broker, and queue.

You can run **atomic** operations, like appending to a string; incrementing the value in a hash; pushing an element to a list; computing set intersection, union and difference; or getting the member with highest ranking in a sorted set.

In order to achieve performance, Redis works with an in-memory dataset. Depending on your use case, you can persist it either by dumping the dataset to disk every once in a while, or by appending each command to a log. Persistence can be optionally disabled, if you just need a feature-rich, networked, in-memory cache.

Redis is a popular choice for caching, session management, gaming, leaderboards, real-time analytics, geospatial, ride-hailing, chat/messaging, media streaming, and pub/sub apps.

Q2: Is Redis just a cache? ☆☆

Topics: Redis Caching

Answer:

Like a cache Redis offers:

- in memory key-value storage

But unlike a cash Redis:

- Supports multiple datatypes (strings, hashes, lists, sets, sorted sets, bitmaps, and hyperloglogs)
- It provides an ability to store cache data into physical storage (if needed).
- Supports pub-sub model
- Redis cache provides replication for high availability (master/slave)
- Supports ultra-fast lua-scripts. Its execution time equals to C commands execution.
- Can be shared across multiple instances of the application (instead of in-memory cache for each app instance)

Q3: Does Redis persist data? ☆☆

Topics: Redis

Answer:

Redis supports so-called "snapshots". This means that it will do a complete copy of whats in memory at some points in time (e.g. every full hour). When you lose power between two snapshots, you will lose the data from the

time between the last snapshot and the crash (doesn't have to be a power outage..). Redis trades data safety versus performance, like most NoSQL-DBs do.

Redis saves data in one of the following cases:

- automatically from time to time
- when you manually call `BGSAVE` command
- when redis is shutting down

But data in redis is not really persistent, because:

- crash of redis process means losing all changes since last save
- `BGSAVE` operation can only be performed if you have enough free RAM (the amount of extra RAM is equal to the size of redis DB)

Q4: What's the advantage of Redis vs using memory? ☆☆☆

Topics: Redis

Answer:

Redis is a *remote* data structure server. It is certainly slower than just storing the data in local memory (since it involves socket roundtrips to fetch/store the data). However, it also brings some interesting properties:

- Redis can be accessed by all the processes of your applications, possibly running on several nodes (something local memory cannot achieve).

- Redis memory storage is quite efficient, and done in a separate process. If the application runs on a platform whose memory is garbage collected (node.js, java, etc ...), it allows handling a much bigger memory cache/store. In practice, very large heaps do not perform well with garbage collected languages.

- Redis can persist the data on disk if needed.

- Redis is a bit more than a simple cache: it provides various data structures, various item eviction policies, blocking queues, pub/sub, atomicity, Lua scripting, etc ...

- Redis can replicate its activity with a master/slave mechanism in order to implement high-availability.

Basically, if you need your application to **scale on several nodes** sharing the same data, then something like Redis (or any other remote key/value store) will be required.

Q5: When to use Redis Lists data type? ☆☆☆

Topics: Redis

Answer:

Redis lists are ordered collections of strings. They are optimized for inserting, reading, or removing values from the top or bottom (aka: left or right) of the list.

Redis provides many commands for leveraging lists, including commands to push/pop items, push/pop between lists, truncate lists, perform range queries, etc.

Lists make great durable, atomic, queues. These work great for job queues, logs, buffers, and many other use cases.

Q6: When to use Redis Sets? ☆☆☆

Topics: Redis

Answer:

Sets are unordered collections of unique values. They are optimized to let you quickly check if a value is in the set, quickly add/remove values, and to measure overlap with other sets.

These are great for things like access control lists, unique visitor trackers, and many other things. Most programming languages have something similar (usually called a Set). This is like that, only distributed.

Redis provides several commands to manage sets. Obvious ones like adding, removing, and checking the set are present. So are less obvious commands like popping/reading a random item and commands for performing unions and intersections with other sets.

Q7: When to use Redis over MongoDB? ☆☆☆

Topics: Redis

Answer:

It depends on kind of dev team you are and your application needs but some notes when to use Redis is probably a good idea:

- **Caching**

  Caching using MongoDB simply doesn't make a lot of sense. It would be too slow.

- If you have enough time to think about your DB design.

  You can't simply throw in your documents into Redis. You have to think of the way you in which you want to store and organize your data. One example are hashes in Redis. They are quite different from "traditional", nested objects, which means you'll have to rethink the way you store nested documents. One solution would be to store a reference inside the hash to another hash (something like *key: [id of second hash]*). Another idea would be to store it as JSON, which seems counter-intuitive to most people with a *SQL-background. Redis's non-traditional approach requires more effort to learn, but greater flexibility.

- If you need **really** high performance.

  Beating the performance Redis provides is nearly impossible. Imagine you database being as fast as your cache. That's what it feels like using Redis as a *real* database.

- If you don't care *that* much about scaling.

  Scaling Redis is not as hard as it used to be. For instance, you could use a kind of proxy server in order to distribute the data among multiple Redis instances. Master-slave replication is not *that* complicated, but distributing you keys among multiple Redis-instances needs to be done on the application site (e.g. using a hash-function, Modulo etc.). Scaling MongoDB by comparison is much simpler.

Q8: How are Redis pipelining and transaction different? ☆☆☆

Topics: Redis

Answer:

Pipelining is primarily a network optimization. It essentially means the client buffers up a bunch of commands and ships them to the server in one go. The commands are not guaranteed to be executed in a transaction. The benefit here is saving network round trip time for every command.

Redis is single threaded so an *individual* command is always atomic, but two given commands from different clients can execute in sequence, alternating between them for example.

Multi/exec, however, ensures no other clients are executing commands in between the commands in the multi/exec sequence.

Q9: Does Redis support transactions? ☆☆☆

Topics: Redis

Answer:

MULTI, **EXEC**, **DISCARD** and **WATCH** are the foundation of transactions in Redis. They allow the execution of a group of commands in a single step, with two important guarantees:

- All the commands in a transaction are serialized and executed sequentially. It can never happen that a request issued by another client is served **in the middle** of the execution of a Redis transaction. This guarantees that the commands are executed as a single isolated operation.
- Either all of the commands or none are processed, so a Redis transaction is also **atomic**.

Q10: How does Redis handle multiple threads (from different clients) updating the same data structure in Redis? ☆☆☆

Topics: Redis

Answer:

Redis is actually **single-threaded**, which is how every command is guaranteed to be atomic. While one command is executing, **no other command will run**.

A single-threaded program can definitely provide concurrency at the I/O level by using an I/O (de)multiplexing mechanism and an event loop (which is what Redis does). The fact that Redis operations are atomic is simply a consequence of the single-threaded event loop. The interesting point is atomicity is provided at no extra cost (it does not require synchronization between threads).

Q11: What is the difference between Redis replication and sharding? ☆☆☆

Topics: Redis

Answer:

- **Sharding**, also known as partitioning, is splitting the data up by key. Sharding is useful to increase performance, reducing the hit and memory load on any one resource.
- While **replication**, also known as mirroring, is to copy all data. Replication is useful for getting a high availability of reads. If you read from multiple replicas, you will also reduce the hit rate on all resources, but the memory requirement for all resources remains the same.

Any key-value store (of which Redis is only one example) supports sharding, though certain cross-key functions will no longer work. Redis supports replication out of the box.

Q12: When to use Redis or MongoDB? ☆☆☆☆

Topics: MongoDB Redis

Answer:

- **Use MongoDB if you don't know yet how you're going to query your data or what schema to stick with.** MongoDB is suited for Hackathons, startups or every time you don't know how you'll query the data you inserted. MongoDB does not make any assumptions on your underlying schema. While MongoDB is schemaless and non-relational, this does not mean that there is no schema at all. It simply means that your schema needs to be defined in your app (e.g. using Mongoose). Besides that, MongoDB is great for prototyping or trying things out. Its performance is not that great and can't be compared to Redis.

- **Use Redis in order to speed up your existing application.** It is very uncommon to use Redis as a standalone database system (some people prefer referring to it as a "key-value"-store).

Q13: When to use Redis Hashes data type? ☆☆☆☆

Topics: Redis

Answer:

Hashes are sort of like a key value store within a key value store. They map between string fields and string values. Field->value maps using a hash are slightly more space efficient than key->value maps using regular strings.

Hashes are useful as a *namespace*, or when you want to logically group many keys. With a hash you can grab all the members efficiently, expire all the members together, delete all the members together, etc. Great for any use case where you have several key/value pairs that need to grouped.

One example use of a hash is for storing user profiles between applications. A redis hash stored with the user ID as the key will allow you to store as many bits of data about a user as needed while keeping them stored under a single key. The advantage of using a hash instead of serializing the profile into a string is that you can have different applications read/write different fields within the user profile without having to worry about one app overriding changes made by others (which can happen if you serialize stale data).

Q14: Explain a use case for Sorted Set in Redis ☆☆☆☆

Topics: Redis

Answer:

Sorted Sets are also collections of unique values. These ones, as the name implies, are ordered. They are ordered by a score, then lexicographically.

This data type is optimized for quick lookups by score. Getting the highest, lowest, or any range of values in between is extremely fast.

If you add users to a sorted set along with their high score, you have yourself a perfect leader-board. As new high scores come in, just add them to the set again with their high score and it will re-order your leader-board. Also great for keeping track of the last time users visited and who is active in your application.

Storing values with the same score causes them to be ordered lexicographically (think alphabetically). This can be useful for things like auto-complete features.

Many of the sorted set commands are similar to commands for sets, sometimes with an additional score parameter. Also included are commands for managing scores and querying by score.

Q15: What is Pipelining in Redis and when to use one? ☆☆☆☆

Topics: Redis

Answer:

If you have many redis commands you want to execute you can use **pipelining** to send them to redis all-at-once instead of one-at-a-time.

Normally when you execute a command, each command is a separate request/response cycle. With pipelining, Redis can buffer several commands and execute them all at once, responding with all of the responses to all of your commands in a single reply.

This can allow you to achieve even greater throughput on *bulk importing* or other actions that involve lots of commands.

Q16: How would you efficiently store JSON in Redis? ☆☆☆☆

Topics: Redis

Answer:

There are many ways to store an array of Objects in Redis:

1. Store the entire object as JSON-encoded string in a single key and keep track of all Objects using a set (or list, if more appropriate):

   ```
   INCR id:users
   SET user:{id} '{"name":"Fred","age":25}'
   SADD users {id}
   ```

   Use when:

- If you use most of the fields on most of your accesses.
- If there is variance on possible keys

2. Store each Object's properties in a Redis hash:

   ```
   INCR id:users
   HMSET user:{id} name "Fred" age 25
   SADD users {id}
   ```

   Use when:

- If you use just single fields on most of your accesses.
- If you always know which fields are available
- If there are a lot of fields in the Object
- Your Objects are not nested with other Objects
- You tend to only access a small subset of fields at a time

3. Store each Object as a JSON string in a Redis hash:

```
INCR id:users
HMSET users {id} '{"name":"Fred","age":25}'
```

Use when:

- really care about not polluting the main key namespace

4. Store each property of each Object in a dedicated key:

```
INCR id:users
SET user:{id}:name "Fred"
SET user:{id}:age 25
SADD users {id}
```

Use when:

- almost never preferred unless the property of the Object needs to have specific TTL or something

Option 4 is generally not preferred. Options 1 and 2 are very similar, and they are both pretty common. Option 1 (generally speaking) may be most preferable because it allows you to store more complicated Objects (with multiple layers of nesting, etc.) Option 3 is used when you really care about not polluting the main key namespace (i.e. you don't want there to be a lot of keys in your database and you don't care about things like TTL, key sharding, or whatever).

Also as a rule of the thumb, go for the option which requires fewer queries on most of your use cases.

Q17: How can I exploit multiple CPU/cores for Redis? ☆☆☆☆

Topics: Redis

Answer:

First Redis is single threaded but it's not very frequent that CPU becomes your bottleneck with Redis, as usually Redis is either memory or network bound.

For instance, using pipelining Redis running on an average Linux system can deliver even 1 million requests per second, so if your application mainly uses `O(N)` or `O(log n)` commands, it is hardly going to use too much CPU.

However, to maximize CPU usage you can start multiple instances of Redis in the same box and treat them as different servers. At some point a single box may not be enough anyway, so if you want to use multiple CPUs you can start thinking of some way to shard earlier.

With Redis 4.0 Redis team started to make Redis more threaded. For now this is limited to deleting objects in the background, and to blocking commands implemented via Redis modules. For future releases, the plan is to make Redis more and more threaded.

Q18: What do the terms "CPU bound" and "I/O bound" mean in context of Redis? ☆☆☆☆

Topics: Redis

Answer:

- A program is **CPU bound** if it would go faster if the CPU were faster, i.e. it spends the majority of its time simply using the CPU (doing calculations). A program that computes new digits of π will typically be CPU-

bound, it's just crunching numbers.

- A program is **I/O bound** if it would go faster if the I/O subsystem was faster. Which exact I/O system is meant can vary; I typically associate it with disk, but of course networking or communication in general is common too. A program that looks through a huge file for some data might become I/O bound, since the bottleneck is then the reading of the data from disk (actually, this example is perhaps kind of old-fashioned these days with hundreds of MB/s coming in from SSDs).

- **Memory bound** means the rate at which a process progresses is limited by the amount memory available and the speed of that memory access. A task that processes large amounts of in memory data, for example multiplying large matrices, is likely to be Memory Bound.

- **Cache bound** means the rate at which a process progress is limited by the amount and speed of the cache available. A task that simply processes more data than fits in the cache will be cache bound.

It's not very frequent that CPU becomes your bottleneck with Redis, as usually Redis is either memory or network bound.

Q19: Why Redis does not support roll backs? ☆☆☆☆

Topics: Redis

Answer:

Redis commands can fail during a transaction, but still Redis will execute the rest of the transaction instead of rolling back.

There are good opinions for this behavior:

- Redis commands can fail only if called with a **wrong syntax** (and the problem is not detectable during the command queueing), or against keys holding the **wrong data type**: this means that in practical terms a failing command is the result of a programming errors, and a kind of error that is very likely to be detected during development, and not in production.
- Redis is internally simplified and faster because it does not need the ability to roll back.

Q20: What is AOF persistence in Redis? ☆☆☆☆

Topics: Redis

Answer:

Redis AOF (Append Only Files) persistence logs every write operation received by the server, that will be played again at server startup, reconstructing the original dataset.

Redis must be explicitly configured for AOF persistence, if this is required, and this will result in a performance penalty as well as growing logs. It may suffice for relatively reliable persistence of a limited amount of data flow.

FullStack.Cafe - Kill Your Tech Interview

Q1: If there's a way to check if a key already exists in a Redis list?

☆☆☆☆

Topics: Redis

Answer:

Your options are as follows:

1. Using `LREM` and replacing it if it was found.
2. Maintaining a separate `SET` in conjunction with your `LIST`
3. Looping through the `LIST` until you find the item or reach the end.

Redis lists are implemented as a, hence the limitations. I think your best option is maintaining a duplicate `SET`. Regardless, make sure your actions are atomic with `MULTI` - `EXEC` or Lua scripts.

Q2: What are the underlying data structures used for Redis?

☆☆☆☆☆

Topics: Redis

Answer:

Here is the underlying implementation of every Redis data type.

- **Strings** are implemented using a C dynamic string library so that we don't pay (asymptotically speaking) for allocations in append operations. This way we have `O(N)` appends, for instance, instead of having quadratic behavior.
- **Lists** are implemented with *linked lists*.
- **Sets** and **Hashes** are implemented with *hash tables*.
- **Sorted sets** are implemented with skip lists (a peculiar type of balanced trees).
- **Zip List**
- **Int Sets**
- **Zip Maps** (deprecated in favour of zip list since Redis 2.6)

It shall be also said that for every Redis operation you'll find the time complexity in the documentation so if you are not interested in Redis internals you should not care about how data types are implemented internally really.

Q3: How much faster is Redis than MongoDB? ☆☆☆☆☆

Topics: Redis MongoDB

Answer:

- Rough results are **2x write, 3x read**.
- The general answer is that Redis 10 - 30% faster when the data set fits within working memory of a single machine. Once that amount of data is exceeded, Redis fails.

- But your best bet is to benchmark them yourself, in precisely the manner you are intending to use them. As a corollary you'll probably figure out the best way to make use of each.

Q4: What happens if Redis runs out of memory? ☆☆☆☆☆

Topics: Redis

Answer:

Redis will either be killed by the Linux kernel OOM killer, crash with an error, or will start to slow down. With modern operating systems malloc() returning NULL is not common, usually the server will start swapping (if some swap space is configured), and Redis performance will start to degrade, so you'll probably notice there is something wrong.

Redis has built-in protections allowing the user to set a max limit to memory usage, using the `maxmemory` option in the configuration file to put a limit to the memory Redis can use. If this limit is reached Redis will start to reply with an error to write commands (but will continue to accept read-only commands), or you can configure it to evict keys when the max memory limit is reached in the case where you are using Redis for caching.

Q5: RDB and AOF, which one should I use? ☆☆☆☆☆

Topics: Redis

Answer:

- The general indication is that you should **use both** persistence methods if you want a degree of data safety comparable to what PostgreSQL can provide you.

- If you care a lot about your data, but still can live with a few minutes of data loss in case of disasters, you can simply **use RDB** alone.

Q6: Is Redis a durable datastore ("D" from ACID)? ☆☆☆☆☆

Topics: Redis

Problem:

By "durable" I mean, the server can crash at any time, and as long as the disk remains in tact, no data is lost (see ACID).

Solution:

Redis is not *usually* deployed as a "durable" datastore (in the sense of the "D" in ACID.), even with journaling. Most use cases intentionally sacrifice a little durability in return for speed.

However, the "append only file" storage mode can optionally be configured to operate in a durable manner, at the cost of performance. It will have to pay for an fsync() on every modification.