Redis Data Types

# Strings

Redis strings store **sequences of bytes**, including**:**

* Text
* serialized objects
* and binary arrays

As such, strings are the simplest type of value you can associate with a Redis key.

They're:

* **often used for caching** (like caching HTML fragments or pages)
* **but they support additional functionality that lets you implement counters**
* **and perform bitwise operations, too.**

Since Redis keys are strings, when we use the string type as a value too, we are mapping a string to another string.

 using the [SET](https://redis.io/docs/latest/commands/set/) and the [GET](https://redis.io/docs/latest/commands/get/) commands are the way we set and retrieve a string value. Note that [SET](https://redis.io/docs/latest/commands/set/) will replace any existing value already stored into the key, in the case that the key already exists, **even if the key is associated with a non-string value**.

> set bike:1 Deimos

OK

> get bike:1

"Deimos"

Values can be strings (**including binary data**) **of every kind**, for instance **you can store a jpeg image inside a value. A value can't be bigger than 512 MB.**

## set or retrieve the value of multiple keys in a single command for reduced latency

> mset bike:1 "Deimos" bike:2 "Ares" bike:3 "Vanth"

OK

> mget bike:1 bike:2 bike:3

1) "Deimos"

2) "Ares"

3) "Vanth"

## Strings as Counters

Even if strings are the basic values of Redis, there are interesting operations you can perform with them. For instance, one is **atomic increment:**

> set total\_crashes 0

OK

> incr total\_crashes

(integer) 1

> incrby total\_crashes 10

(integer) 11

The [INCR](https://redis.io/docs/latest/commands/incr/) command parses the string value as an integer, increments it by one, and finally sets the obtained value as the new value.

What does it mean that INCR is atomic?

**That even multiple clients issuing INCR against the same key will never enter into a race condition.** For instance, it will never happen that client 1 reads "10", client 2 reads "10" at the same time, both increment to 11, and set the new value to 11. The final value will always be 12 and **the read-increment-set operation is performed while all the other clients are not executing a command at the same time.**

**Practical point:**

if you have a system that increments a Redis key using [INCR](https://redis.io/docs/latest/commands/incr/) every time your web site receives a new visitor. You may want to collect this information once every hour, without losing a single increment. You can [GETSET](https://redis.io/docs/latest/commands/getset/) the key, assigning it the new value of "0" and reading the old value back.(This GETSET command has apparently been deprecated, use:

set key value get )

## Implementing Simple Locks with Redis Strings

SETNX sets the value of a key if and only if the key does not already exist. This property makes it ideal for implementing locks, as a successful SETNX operation indicates that the lock has been acquired.

More details:

<https://redis.io/glossary/redis-lock/>

<https://redis.io/docs/latest/develop/clients/patterns/distributed-locks/>

## Bitwise Operations

To perform bitwise operations on a string, see the [bitmaps data type](https://redis.io/docs/latest/develop/data-types/bitmaps/) section.

## Alternatives

If you're storing structured data as a serialized string, you may also want to consider Redis [hashes](https://redis.io/docs/latest/develop/data-types/hashes/) or [JSON](https://redis.io/docs/latest/develop/data-types/json/).

## Performance

Most string operations are O(1). However, be careful with the [SUBSTR](https://redis.io/docs/latest/commands/substr/), [GETRANGE](https://redis.io/docs/latest/commands/getrange/), and [SETRANGE](https://redis.io/docs/latest/commands/setrange/) commands, which can be O(n). These random-access string commands may cause performance issues when dealing with large strings.

## Limits

By default, a single Redis string can be a maximum of 512 MB.

# JSON