Data types

Strings

Strings are the most basic kind of Redis value. 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.

You can do a number of interesting things using strings in Redis, for instance you can:

- Use Strings as atomic counters using commands in the INCR family: INCR, DECR, INCRBY.
- Append to strings with the APPEND command.
- Use Strings as a random access vectors with GETRANGE and SETRANGE.
- Encode a lot of data in little space, or create a Redis backed Bloom Filter using GETBIT and SETBIT.

Check all the available string commands for more information, or read the introduction to Redis data types.

Lists

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 LPUSH command inserts a new element on the head, while RPUSH inserts a new element on the tail. A new list is created when one of this operations is performed against an empty key. Similarly the key is removed from the key space if a list operation will empty the list. These are very handy semantics since all the list commands will behave exactly like they were called with an empty list if called with a non-existing key as argument.

Some example of list operations and resulting lists:

```
LPUSH mylist a   # now the list is "a"
LPUSH mylist b   # now the list is "b","a"
RPUSH mylist c   # now the list is "b","a","c" (RPUSH was used this t
```

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

The main features of Redis Lists from the point of view of time complexity are the support for constant time insertion and deletion of elements near the head and tail, even with many millions of inserted items. Accessing elements is very fast near the extremes of the list but is slow if you try accessing the middle of a very big list, as it is an O(N) operation.

You can do many interesting things with Redis Lists, for instance you can:

- Model a timeline in a social network, using LPUSH in order to add new elements in the user time line, and using LRANGE in order to retrieve a few of recently inserted items.
- You can use LPUSH together with LTRIM to create a list that never exceeds a given number of elements, but just remembers the latest N elements.
- Lists can be used as a message passing primitive, See for instance the well known Resque Ruby library for creating background jobs.
- You can do a lot more with lists, this data type supports a number of commands, including blocking commands like BLPOP.

Please check all the available commands operating on lists for more information, or read the introduction to Redis data types.

Sets

Redis Sets are an unordered collection of Strings. It is possible to add, remove, and test for existence of members in O(1) (constant time regardless of the number of elements contained inside the Set).

Redis Sets have the desirable property of not allowing repeated members. Adding the same element multiple times will result in a set having a single copy of this element. Practically speaking this means that adding a member does not require a *check if exists then add* operation.

A very interesting thing about Redis Sets is that they support a number of server side commands to compute sets starting from existing sets, so you can do unions, intersections, differences of sets in very short time.

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

You can do many interesting things using Redis Sets, for instance you can:

- You can track unique things using Redis Sets. Want to know all the unique IP addresses visiting a given blog post? Simply use SADD every time you process a page view. You are sure repeated IPs will not be inserted.
- Redis Sets are good to represent relations. You can create a tagging system with Redis using a Set to represent every tag. Then you can add all the IDs of all the objects having a given tag into a Set representing this particular tag, using the SADD command. Do you want all the IDs of all the Objects having three different tags at the same time? Just use SINTER.

- You can use Sets to extract elements at random using the SPOP or SRANDMEMBER commands.

As usual, check the full list of Set commands for more information, or read the introduction to Redis data types.

Hashes

Redis Hashes are maps between string fields and string values, so they are the perfect data type to represent objects (e.g. A User with a number of fields like name, surname, age, and so forth):

```
HMSET user:1000 username antirez password P1pp0 age 34
HGETALL user:1000
HSET user:1000 password 12345
HGETALL user:1000
```

A hash with a few fields (where few means up to one hundred or so) is stored in a way that takes very little space, so you can store millions of objects in a small Redis instance.

While Hashes are used mainly to represent objects, they are capable of storing many elements, so you can use Hashes for many other tasks as well.

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

Check the full list of Hash commands for more information, or read the introduction to Redis data types.

Sorted sets

Redis Sorted Sets are, similarly to Redis Sets, non repeating collections of Strings. The difference is that every member of a Sorted Set is associated with score, that is used in order to take the sorted set ordered, from the smallest to the greatest score. While members are unique, scores may be repeated.

With sorted sets you can add, remove, or update elements in a very fast way (in a time proportional to the logarithm of the number of elements). Since elements are *taken in order* and not ordered afterwards, you can also get ranges by score or by rank (position) in a very fast way. Accessing the middle of a sorted set is also very fast, so you can use Sorted Sets as a smart list of non repeating elements where you can quickly access everything you need: elements in order, fast existence test, fast access to elements in the middle!

In short with sorted sets you can do a lot of tasks with great performance that are really hard to model in other kind of databases.

With Sorted Sets you can:

- Take a leaderboard in a massive online game, where every time a new score is submitted you update it using ZADD. You can easily take the top users using ZRANGE, you can also, given a user name, return its rank in the listing using ZRANK. Using ZRANK and ZRANGE together you can show users with a score similar to a given user. All very *quickly*.
- Sorted Sets are often used in order to index data that is stored inside Redis. For instance if you have many hashes representing users, you can use a sorted set with elements having the age of the user as the score and the ID of the user as the value. So using ZRANGEBYSCORE it will be trivial and fast to retrieve all the users with a given interval of ages.

Sorted Sets are probably the most advanced Redis data types, so take some time to check the full list of Sorted Set commands to discover what you can do with Redis! Also you may want to read the introduction to Redis data types.

Bitmaps and HyperLogLogs

Redis also supports Bitmaps and HyperLogLogs which are actually data types based on the String base type, but having their own semantics.

Please refer to the introduction to Redis data types for information about those types.
