Database - P93

Redis

APPEND <key> <value> : append value to key
if <key> not exist, create and set as empty string

STRLEN <key> : get length of value stored in key
return error if <key> hold non-string value

although store string, recognize integer and provide simple operation
INCR <key> : increment integer value of key by one
INCRBY <key> <increment> : increment integer value of key by given amount
INCRBYFLOAT <key> <increment> : increment float value of key by given amount
DECR <key> : decrement integer value of key by one
DECRBY <key> <decrement> : decrement integer value of key by given amount
if <key> not exist, set to 0 before performing operation
return error if <key> contain wrong type value or string not represented as integer
limit to 64-bit signed integer

GETSET <key> <value> : set string value of key and return old value
return error when <key> exist but not hold string value

MULTI block atomic commands : similar concept as transaction support in other database
wrapping operations in single block will complete either successfully or not all
will never end up with partial operation
MULTI : make the start of transaction block
EXEC : execute all commands issued after MULTI
DISCARD : flush all previously queued commands in transaction
and restore connection state to normal
similar to ROLLBACK in SQL, clear transaction queue
unlike ROLLBACK, will not revert the database
simply will not run transaction at all
end effect identical, while underlying concept different
transaction rollback - operation cancellation
WATCH [<key>]+ : mark given key to be watched for conditional execution
UNWATCH : flush all previously watched key for transaction
no need to manually call if call EXEC / DISCARD

Database - P94

Redis

collection datatype can contain huge number (up to 2^{32}) of values per key

command pretty to decode with the pattern

set command begin with S

hash command begin with H

sorted set command begin with Z

list command begin with L (for left) / R (for right)

depending on direction of operation

Hashes: like nested Redis object that can take any number of key-value pairs

avoid storing data with artificial key prefix

":" for logical separation of key into segments

only matter of convention

HSET <key> <field> <value> : set string value of hash field

HMSET <key> [<field> <value>]+ :

set multiple hash fields to multiple values

HSETNX <key> <field> <value> :

set value of hash field only if field not exist

HGET <key> <field> : get value of hash field

HMGET <key> [<field>]+ : get values of all given hash fields

HGETALL <key> : get all fields and values in hash

HKEYS <key> : get all fields in hash

HVALS <key> : get all values in hash

HLEN <key> : get number of fields in hash

HSTRLEN <key> <field> : get length of value of hash field

(value): return nil when <field> not present or <key> not exist

(length): return 0 when <field> not present or <key> not exist

HEXISTS <key> <field> : determine if hash field exist

HDEL <key> [<field>]+ : delete given hash field

HINCRBY <key> <field> <increment> :

increment integer value of hash field by given number

HINCRBYFLOAT <key> <field> <increment> :

increment float value of hash field by given amount

Database - P95

Redis

List : contain multiple ordered values
act both as queue (FIFO) and as stack (LIFO)
more sophisticated actions
inserting somewhere in the middle / constraining size / moving between lists
Redis command return number of value pushed when inserting
list operation in Redis use zero-based index
negative position : number of step from the end
LPUSH <key> [<value>]+ : prepend one or more values to list
element inserted one after other to the head of list
RPUSH <key> [<value>]+ : append one or more values to list
element inserted one after other to the tail of list
if <key> not exist, created as empty list before performing push operation
LPUSHX <key> <value> : insert value at the head of list
RPUSHX <key> <value> : insert value at the tail of list
only if <key> already exist and hold list
LSET <key> <index> <value> : set value of element in list by index
return error for out of range indexes
LINSERT <key> BEFORE | AFTER <pivot> <value>
insert element before/after another element in list

LLEN <key> : get length of list
LINDEX <key> <index> : get element from list by index
negative indices can be used to destignate element starting at tail
-1 : last element, -2 : penultimate element, ...
return nil when index out of range
LRANGE <key> <start> <stop> : get range of elements from list
out of range indexes will not produce error

LPOP <key> : remore and return first element of list
RPOP <key> : remove and return last element of list
RPOPLPUSH <source> <destination> :
return and remove the last element of <source>
push the element at the first element of <destination>

Database - P96

Redis

List : LTRIM <key> <start> <stop> : trim list to specified range
out of range indexes will not produce error
useful when Redis to store logs

LREM <key> <count> <value> : remove elements from list
remove the first <count> occurences of elements equal to <value>
<count> =0 : remove all elements equal to <value>
<count> >0 : remove elements equal to <value> from head to tail
<count> <0 : remove elements equal to <value> from tail to head
return number of removed elements

blocking list : BLPOP [<key>]+ <timeout>
BRPOP [<key>]+ <timeout>
remove and get first/last element in list
or block until one available
blocking version of LPOP/RPOP
block connection when no element to pop from any of given lists
element popped from head/tail of first non-empty list
return multi-bulk (<key>, <value>)
nil multi-bulk when no element popped and <timeout> expired
BRPOPLPUSH <source> <destination> <timeout>
blocking variant of RPOPLPUSH
<timeout>=0 : used to block indefinitely
behaves exactly like LPOP/RPOP/RPOPLPUSH
when used inside MULTI/EXEC block

Set : SADD <key> [<member>]+ : add one or more members to set
SCARD <key> : get cardinality of set
SMEMBERS <key> : get all members of set
SISMEMBER <key> <member> : determine if given value is member of set
SMOVE <source> <destination> <member> :
<member> removed from <source> and add to <destination> if exist
appear to be member of <source> or <destination>
for other clients in every given moment

Database - P97

Redis

Set: SUNION [<key>]+ : union multiple sets
SINTER [<key>]+ : intersect multiple sets
SDIFF [<key>]+ : substract multiple sets
左结合的：SDIFF A B C 等价于 (A\B)\C
SUNIONSTORE <destination> [<key>]+
SINTERSTORE <destination> [<key>]+
SDIFFSTORE <destination> [<key>]+
union / intersect / substract multiple sets and store resulting set
overwrite <destination> if exist

SRANDMEMBER <key> [<count>]
get one or more random members from set
<count> omitted : return one random element from set
<count> positive : return array of <count> distinct elements
<count> negative : return array of <count> elements
may return same element multiple times
return nil if <key> not exist
SPOP <key> [<count>]
remove and return one or more random members from set
SREM <key> [<member>]+
remove specified members from set
treat <key> as empty set if not exist

unlike list, no blocking command for set

Sorted set: ordered like list, but unique like set order of value
field-value pair like hashes, but field: numeric score and denote
as random access priority queue
ZADD <key> [NX|XX] [CH] [INCR] [<score> <member>]+
add one or more members to sorted set, or update score if exist
score value: string representation of double precision floating point number
NX|XX: always / never add new member, not / only update member exist
CH: return number of new member added and member score updated
INCR: act like ZINCRBY, only one score-member can be specified

Database - P98

Redis

sorted set : ZCARD <key> : return cardinality of sorted set

ZSCORE <key> <member> : get score associated with given member

return double precision floating point number represented as string

ZRANK <key> <member> : determine index of given member

scores ordered from low to high (ascending)

rank/index is 0-based

ZREVRANK <key> <member> :

get rank of member with scores ordered from high to low (descending)

return nil if <member> not exist / <key> not exist

ZCOUNT <key> <min> <max> :

count members with scores within given values

ZINCRBY <key> <increment> <member>

increment score of given member by given increment

if <member> not exist, added with <increment> as score

if <key> not exist, new sorted set created with specified <member>

ZRANGE <key> <start> <stop> [WITHSCORES] :

return range of members in sorted set by index

member considered to be ordered in ascending

ZREVRANGE <key> <start> <stop> [WITHSCORES] :

member considered to be ordered in descending

<start>, <stop> are inclusive ranges

WITHSCORES : return scores of members together with members

ZRANGEBYSCORE <key> <min> <max> [WITHSCORES] [LIMIT <offset> <count>]

ZREVRANGEBYSCORE <key> <min> <max> [WITHSCORES] [LIMIT <offset> <count>]

return range of members by score

ZRANGEBYLEX <key> <min> <max> [LIMIT <offset> <count>]

ZREVRANGEBYLEX <key> <min> <max> [LIMIT <offset> <count>]

<min>, <max> : (/[specify range item exclusive/inclusive

+/- special meaning for positive/negative infinite string

+inf/-inf no required to know highest/lowest score

LIMIT : only get range of matching member,

similar to SELECT LIMIT offset, count in SQL

Database - P99

Redis

sorted set: ZLEXCOUNT <key> <min> <max>:
count number of members between given lexicographical range
when all elements inserted with same score

ZREM <key> [<member>]+: remove one or more members from sorted set

ZREMRANGEBYRANK <key> <start> <stop>:
remove all members within given indexs

ZREMRANGEBYSCORE <key> <min> <max>:
remove all members with score between <min> and <max> (inclusive)

ZREMRANGEBYLEX <key> <min> <max>:
remove all members between given lexicographical range
when all elements inserted with same score
in order to force lexicographical ordering

ZPOPMAX <key> [<count>]

ZPOPMIN <key> [<count>]
remove and return up to count members with highest/lowest scores
<count> higher than cardinality will not produce error
one with highest/lowest will be the first when returning multiple elements

BZPOPMAX [<key>]+ <timeout>

BZPOPMIN [<key>]+ <timeout>
blocking variant of ZPOPMAX / ZPOPMIN primitive
block connection when no member to pop from any of given sorted set

ZUNIONSTORE <destination> <numberkeys> [<key>]+
[WEIGHTS [<weight>]+] [AGGREGATE SUM|MIN|MAX]

ZINTERSTORT <destination> <numberkeys> [<key>]+
[WEIGHTS [<weight>]+] [AGGREGATE SUM|MIN|MAX]
compute union/intersection of numkeys sorted set and store result in destination
WEIGHTS: possible to specify multiplication factor for each sorted set
score of every element multiplied by factor before AGGREGATE
AGGREGATE: possible to specify how result of union/intersection aggregated

Database - P100

Redis

fast-access cache for data expensive to retrieve or compute
- common use case for key-value system
- ensuring content expire after designated time period
  - essential function to keep the set from growing unboundedly
- EXPIRE <key> <seconds> : set key's time to live in seconds
- PEXPIRE <key> <milliseconds> : set key's time to live in milliseconds
  - key automatically deleted after timeout expired
  - key with associated timeout is volatile in Redis termindogy
- EXPIREAT <key> <timestamp>: set expiration of key as UNIX timestamp
- PEXEIREAT <key> <milliseconds-timestamp>:
  - set expiration for key as UNIX timestamp specified in milliseconds
  - absolute UNIX timestamp since January 1, 1970
  - key deleted rather than expired
    - if EXPIRE/PEXPIRE with non-positive timeout
    - or EXPIREAT/PEXPIREAT with timestamp in the past
  - return 1 if timeout set, 0 if key not exist

- EXISTS *[<key>]+ : return if key exist
  - return total number of given keys existing
- TTL <key> : get time to live for key in seconds
- PTTL <key> : get time to live for key in milliseconds
  - return -1 if key exist but no associated expire
  - return -2 if key not exist
- PERSIST <key>: remove expiration from key
  - turn key from volatile to persistent
  - return 0 if key not exist / no associated timeout
- DEL [<key>]+ : remove specified key

update expire time whenever retrieve value
- common trick for keeping only recently used key
- referred to most recently used (MRU) caching algorithm
- unused key just expire as normal

Database - P101

Redis

not support expiring individual item in collection
EXPIRE keyword only work on top-level key
provide sufficient functionality in sorted set to accomplish
high level library (e.o Redisson for Java) provide functionality as well
repurpose score of item inserted into sorted set
to be expiration time (UNIX timestamp)
e.o ZADD jobs (now() + 60) job1
priority equal to timestamp of 60s from now
future request can quickly determine from the set
cron job set up to expire data easily from the set
e.o ZREMRANGEBYSCORE jobs 0 now()

namespace: referred to database, keyed by number
default database known as database 0
SELECT <index>: change selected database for current connection
having specified 0-based numeric index
new connection always use database 0
all databases anyway persisted together in same RDB/AOF file
different databases can have key having same name
MOVE <key> <db>: move key to another database
do nothing if <key> already exist in destination database
or not exist in source database
possible to use as locking primitive
useful for different applications running against single Redis server
need to trade data between each other
SWAPDB <index> <index>: swap two Redis database
immediately all clients connected to given database
will see data of other database
FLUSHDB [ASYNC]: delete all keys of currently selected DB
never fail: don't do this in production
RANDOMKEY: return random key from currently selected database
RENAME <key> <newkey>: rename key to newkey
return error if <key> not exist
<newkey> overwritten if already exist

Database - P102

Redis

telnet : Streaming commands through TCP directly
interact without command-line interface
terminating command with CRLF (carriage return / line feed, 回车/换行)
output returned different to on the console
Redis streams OK status prefixed by "+"
number prefixed by "$" : number of characters in following string
number prefixed by ":" : number of values added successfully
"*" and number : number of complex values about to be returned

pipelining : netcat (nc) : BSD (Berkeley Software Distribution / Berkeley UNIX) command
stream own string one at a time
must specifically end line with CRLF
pipelining / streaming multiple commands into single request
should always be preferred if make sense
$ (echo -en "PING\r\nPING\r\nPING\r\n"; sleep 1) | nc localhost 6379

publish-subscribe : several subscribers want to read announcements of single publisher
SUBSCRIBE [<channel>]+ :
listen for message published to given channels
cause CLI to output "Reading message ..."
block while subscriber listen for incoming message
not supposed to issue any other commands /PUNSUBSCRIBE
except for additional SUBSCRIBE/PSUBSCRIBE/UNSUBSCRIBE
PSUBSCRIBE [<pattern>]+
listen for message published to channel matching given patterns
glob-style pattern : ? : any single character
* : zero or more any characters
[..] : characters in list
\ : to escape special character
UNSUBSCRIBE [[<channel>]*+]
UNPSUBSCRIBE [[<pattern>]+]
unsubscribe client from given channels / patterns,
if none given, from all of channels / patterns

Database - P103

Redis

publish-subscribe : PUBLISH <channel> <message> : post message to given channel
return number of clients received message
all subscribes receive multibulk reply :
string "message" / channel name / published message value
have to CTRL-C to break connection on redis-cli

server : INFO [<section>] : get information and statistics about server
<section> : select specific section
server : general information about Redis server
clients : client connections section
memory : memory consumption related information
persistence : RDB and AOF related information
stats : general statistics
replication : master/replica replication information
cpu : CPU consumption statistics
commandstats : Redis command statistics
cluster : Redis cluster section
key space : database related statistics
all : return all sections
default : return only default set of section

durability : no persistence : Redis keep all value in main memory
good choice for basic cache server
durability increase latency
Redis built-in support for storing value to disk
rather than other fast-access cache
SAVE : perform synchronous save of dataset
producing point in time snapshot of all data in Radis instance
in form of RDB file
block all other clients in production environment
BGSAVE : asynchronously save dataset to disk (in background)
LASTSAVE : get UNIX time stamp of last successful save to disk
check if BGSAVE command succeeded

Database - P104

Redis

snapshotting : alter the rate of storage to disk
by adding / removing / altering one of save fields
each prefixed by "save" keyword
followed by time in second
and minumn number of keys must change before write occurs
ex : save 900 1 , any key change saved in at least 900 s
save 60 10000, if 10000 keys change, save in 60 second

append-only file : Redis eventually durable by default
asynchronous write value to disk in interval defined
can also be forced to write by client-initiated command
acceptable for [redacted] scenario : second-level cache / session server
insufficient for durable thing, like financial data
appendonly.aof : append-only file provided
keep record of all write command
like write-ahead log in HBase
"appendonly" must be enabled by [redacted] setting to "yes" to redis.conf
(slow) always : more durable because every command saved
(default) everyses : save update/write command only once a second
no : let OS handle flushing

security : not natively built to be fully secure server
AUTH <password> : request for authentication in password-protected server
can be instructed to require password before allowing execute command
done using "requirepass" directive in configuration file
can be safely ignored because merely scheme for setting plaintext password
also provide command-level security through obscurity
ex rename-command FLUSHALL c283d93ac95...
相当于用 c283d93ac95... 替换命令 FLUSHALL configuration
can disable command entirely by setting to blank string in
ex rename-command FLUSHALL ""
can set any number of commands to blank string
allow modicum of customization over command environment

Database - P105

Redis

replication : Redis support master-slave replication
- one server is master by default
- data replicated to any number of slave servers
- multiple ways to set server as slave of master redis node
  - $ redis.conf : slaveof 127.0.0.1 6379
  - command line : --slaveof 127.0.0.1 6379

data dump : can generate / insert data using redis Ruby gem
- can use hiredis gem to speed up data insertion
  - C driver considerably faster than native Ruby driver
    - $ %w{hiredis ...}
  - should be used in production environment
- huge performace improvement when using pipelined operation
  - pipelining reduce number of Redis connection required
    - but building pipelined dataset has overhead of its own
  - find balance between resource consumption and performance improvement

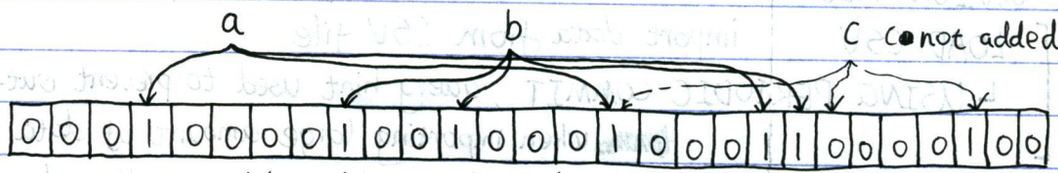
cluster : interface for building simple ad-hoc distributed Redis cluster
- unlike master-slave setup, all servers take master (default) configuration
- bridging between servers require minor changes to data dump client
  - need to require redis/distributed file from redis gem
  - Redis client replaced with Redis::Distributed
    - now take list of server URIs instead of one
- decrease in performance because more work done by client
- can validate key stored on seperate servers
  - by attempting to retrieve sam key from each server

bloom filter : simply never perform query for data that won't return any data
- good way to improve performance of any data retrieval system
- short-circuiting query for data known to have never seen inserted
- can return false-positive, cannot ever return false negative
  - like bloom filter with HBase

Database - P106

Redis

bloom filter : succeed at discovering nonexistence
by converting value to very sparse sequence of bits
and comparing sequence to union of every value's bits
each time new value added
OR against current bloom filter bit sequence
when check whether value already in the system
perform AND against bloom filter bit sequence
if any true bit not true in bloom filter corresponding bucket
value never added



using bloomfilter-rb Ruby gem created by Ilya Grigorik
concept transferable to any programming language

BIT : bloomfilter function by flipping certain bit in sparse binary field

SETBIT <key> <offset> <value>
set/clear bit at offset in the string value
bet set/cleared depending on <value>, either 1 or 0
if <key> not exist, new string created
string grown to make sure can hold bit at <offset>
<offset> required to be no less than 0, and small than 2^{32}

GETBIT <key> <offset> : return bit value at offset in the string value
if <key> not exist, assumed to be empty string
if <offset> beyond string length
string assumed to be contiguous space with 0s

commonly used when faster to flip bit to describe option

BITCOUNT <key> [<start> <end>]
count number of set bit (population counting) in string
by default all bytes contained in string examined
possible to specify counting operation only in interval
passing additional argument <start> and <end>
if <key> not exist, treated as empty string, return 0

Database - P107

Redis

BIT : BITFIELD <key> [GET <type> <offset>]
[SET <type> <offset> <value>]
[INCRBY <type> <offset> <increment>]
[OVERFLOW WRAP | SAT | FAIL]

treat Redis string as array of bit
capable of addressing specific integer fields of varying bit width and arbitrary non (necessary) aligned offset

GET <type> <offset> : return specified bit field

SET <type> <offset> <value> :
set specified bit field and return old value

INCRBY <type> <offset> <increment> :
increment specified bit field and return new value

OVERFLOW WRAP | SAT | FAIL
sub command only change behavior of successive INCRBY
WRAP : wrap around, both with signed / unsigned integer
unsigned : modulo maximum value integer can contain
signed : overflow restart towards most negative value
underflow restart towards most positive value
used if not otherwise specified
SAT : use saturation arithmetic
underflow set to minimun integer value
overflow set to maximum integer value
FAIL : no operation performed on overflow / underflow detected
corresponding return value set to NULL, ep (nil)

<type> : prefixing with "i" for signed integer, "u" for unsigned integer
with number of bits of integer type
support up to 64-bit for signed integer
up to 63-bit for unsigned integer

<offset> : if number without any prefix specified
used as 0-based bit offset inside string
if number prefixed with character "#"
specified offset multipled by integer type width

Database - P108

Redis BIT: BITOP <operation> <destkey> [<key>]+

perform bitwise operation between multiple keys
store result in the destination key

operation: support AND / OR / XOR / NOT

特别地对于 NOT，只传入一个 key
即有 BITOP NOT <destkey> <key>

if operation performed between strings having different lengths
strings shorter than longest treated as zero-padded up to length of longest

return size of string stored in destination key

BITPOS <key> <bit> [<start>] [<end>]

find the first bit set/clear in string

by default all bytes contained are examined
specified interval passed by additional argument <start> and <end>
interpreted as range of byte rather than range of bit

if find set bit in empty string / string with all 0s, return -1

if find clear bit in string with all 1s
return first bit not part of string on the right

strength: obvious strength of Redis is speed

Redis allow store complex value (list / hash / set), unlike other key-value store
retrieve value based on operation specific to datatype

Redis' durability option allow trade speed for data safety

built-in master-slave replication is great way to ensure durability operation
without requiring slowness of syncing append-only file on disk on every
replication great for high-read system, such as cache

weakness: fast largely because reside in memory

inherent durability problem for main memory database
shut down before snapshot occur will lose data

risk with playing back expiry value even enable append-only file on disk

not support dataset larger than available RAM, partical limitation on size
although Redis clustering available and can help with RAM requirement
must roll own client to do so