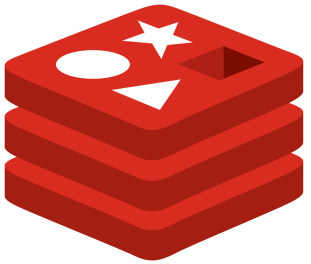
Fun with

and



redis

Javier Ramírez
@supercoco9

teowaki

WHAT?

Redis is an open source, BSD licensed, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets. (http://redis.io)

WHO?

Salvatore Sanfilippo
@antirez

Pieter Noordhuis
@pnoordhuis

95 contributors at
https://github.com/antirez/redis

THE REDIS MANIFESTO

1. A DSL for Abstract Data Types
2. Memory storage is #1
3. Fundamental data structures for a fundamental API
4. Two levels of API
5. Code is like a poem; it's not just something we write to reach some practical result
6. We're against complexity
7. We optimize for joy

```
javier@javier-teowaki:~/.fonts$ redis-benchmark -t get
====== GET ======
  10000 requests completed in 0.14 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

96.22% <= 1 milliseconds
98.97% <= 2 milliseconds
99.48% <= 3 milliseconds
99.51% <= 5 milliseconds
100.00% <= 5 milliseconds
68965.52 requests per second
```

```
javier@javier-teowaki:~/.fonts$ redis-benchmark -t set
====== SET ======
  10000 requests completed in 0.12 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

97.73% <= 1 milliseconds
99.67% <= 2 milliseconds
100.00% <= 2 milliseconds
80000.00 requests per second
```

```
====== LRANGE_100 (first 100 elements) ======
  10000 requests completed in 0.35 seconds
  50 parallel clients
  3 bytes payload
  keep alive: 1

75.53% <= 1 milliseconds
94.14% <= 2 milliseconds
97.86% <= 3 milliseconds
98.97% <= 4 milliseconds
99.51% <= 12 milliseconds
100.00% <= 12 milliseconds
28328.61 requests per second
```

REDIS IS FAST

UNSCIENTIFIC BENCHMARKS*

	PostgreSQL	Redis
1K inserts	3.9s	0.07s
1K reads	0.113s	0.06s
10K inserts	42.9s	0.67s
10K reads	1.15s	0.6s
100K inserts	6.8m	7s
100K reads	11.59s	6.06s
100K pipelined inserts	-	1.45s
100K pipelined reads	-	1.22s

*on an AWS micro instance with the default install for postgresql and redis

REDIS IS GAME CHANGING

Disclaimer: after using it, you'll find yourself doing things you wouldn't do before

SOME WILL NOT MAKE SENSE

As every other technology, Redis has its sweet spot, but it's not a silver bullet

ABUSING SIDEKIQ/RESQUE

```ruby
module ActivityLoggable
  extend ActiveSupport::Concern

  included do
    after_commit  :activity_log_create,  on: :create
    after_commit  :activity_log_update,  on: :update
    after_commit  :activity_log_destroy, on: :destroy
  end

  private
  def activity_log_create
    Activity.log_create(self)
  end

  def activity_log_update
    Activity.log_update(self)
  end

  def activity_log_destroy
    Activity.log_destroy(self)
  end
end
```

```ruby
def self.log(scope, type, model, user, params={})
  activity = Activity.new( { scope: scope, type: type,
                             model: model,
                             user: user, params: params}
  )

  Sidekiq::Client.enqueue( activity.queue, activity.fields_for_r
end
```

```ruby
class Activities::BaseQueue
  include Sidekiq::Worker

  def perform(activity_hash)
    Activities::Dispatcher.dispatch activity_hash
  end
end
```

```ruby
def api_stats_action
  #removing all the internal headers used by the rack/rails stack
  useful_headers=Hash[request.headers.entries].reject{|k,v| k =~ /rack|action_dispatch|action_controller|warden/}
  Activity.log('ApiStats', :monitor, nil, @current_user,
               {uri: request.original_fullpath, format: request.format.to_s, headers: useful_headers })
  nil
end
```

IN MEMORY DATA STORE

Redis stores all the data in memory all the time. Plan your architecture accordingly

PERSISTANCE: RDB

Compact binary format
Saves snapshots every few minutes
Good for backups and synchronizing
If Redis crashes, a few minutes worth of data will be lost

DURABILITY: AOF

Log text format
Configurable durability
Large file, can slow down startup
If Redis crashes, typically one second of data could be lost

CONNECTING FROM RUBY

```
javier@javier-teowaki:~/work/teowaki/api/git$ telnet 127.0.0.1 6379
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
PING
+PONG
```

```ruby
require 'redis'
require 'hiredis'

r=Redis.new driver: :hiredis
r.ping
```

redis-rb exposes calls to all the methods in the redis protocol

other useful gems:

- **github.com/soveran/nest => helps with key naming**
- **https://github.com/soveran/ohm => "ORM" for redis**
- **github.com/obie/redis_props => properties for AR objects**
- **github.com/nateware/redis-objects => map redis types to ruby objects**
- **github.com/agoragames/amico => redis-backed friendships**

DATA TYPES

Strings
Hashes
Lists
Sets
Sorted Sets

Key handling
Transactions
Scripting
Pub/Sub
Server
Connection

UNTYPED OPERATIONS

STRING COMMANDS

```
2.0.0p195 :316 > r.set 'user.1234.name', 'javier'
 => "OK"
2.0.0p195 :317 > r.get 'user.1234.name'
 => "javier"
2.0.0p195 :318 > r.getrange 'user.1234.name', 0, 3
 => "javi"
2.0.0p195 :319 > r.incr 'user.counter'
 => 1
```

string commands

append, get, getset, set
getrange, setrange, strlen
bitcount, bitop, getbit, setbit
mget, mset, msetnx
setnx, setex, psetex
incr, incrby, incrbyfloat, decr, decrby, decrbyfloat

LIST COMMANDS

```
2.0.0p195 :328 > r.lpush 'users', ['javier', 'matz', 'diego']
 => 3
2.0.0p195 :329 > r.lrange 'users', 0, -1
 => ["diego", "matz", "javier"]
2.0.0p195 :330 > r.rpush 'users', 'nikos'
 => 4
2.0.0p195 :331 > r.lrange 'users', 0, -1
 => ["diego", "matz", "javier", "nikos"]
2.0.0p195 :332 > r.lpop 'users'
 => "diego"
2.0.0p195 :333 > r.lrange 'users', 0, -1
 => ["matz", "javier", "nikos"]
```

List commands

llen, lindex, linsert, lrange, lrem, lset, ltrim
lpop, lpush, lpushx, rpop, , rpush, rpushx
blpop, brpop
rpoplpush, brpoplpush

HASH COMMANDS

at the hash level

hkeys, hvals, hlen, hgetall, hexists

at the attribute level

hmget, hmset
hdel, hget, hset, hsetnx
hincrby, hincrbyfloat

SET COMMANDS

```
2.0.0p195 :391 > r.sadd 'team.euruko.users', ['javier', 'diego', 'matz', 'nikos']
 => 4
2.0.0p195 :392 > r.spop 'team.euruko.users'
 => "javier"
2.0.0p195 :393 > r.sismember 'team.euruko.users', 'matz'
 => true
2.0.0p195 :394 > r.sismember 'team.euruko.users', 'javier'
 => false
```

Set commands

sadd, scard, smembers,
sismember, srem, smove, srandmember, smove
sdiff, sinter, sunion
sdiffstore, sinterstore, sunionstore

team.euruko	location:greece	contact:nikos	edition:2013
team.teowaki	location:london	contact:diego	

team.euruko.users	javier	diego	matz	nikos
team.teowaki.users	javier	diego	alberto	

users.javier.teams	euruko	teowaki
users.alberto.teams	teowaki	
users.nikos.teams	euruko	
users.matz.teams	euruko	
users.diego.teams	euruko	teowaki

tags.matz	ruby	mruby
tags.nikos	ruby	css
tags.javier	ruby	
tags.diego	ruby	chef

```
r.sunion r.smembers('team.euruko').map{|m| "tags.#{m}"}
=>["mruby", "ruby", "css", "chef"]
r.sinter r.smembers('team.euruko').map{|m| "tags.#{m}"}
=>["ruby"]
r.diff r.smembers('team.euruko').map{|m| "tags.#{m}"}
=>["mruby", "css", "chef"]
```

SORTED SET COMMANDS

```
2.0.0p195 :368 > r.zadd 'users.karma', [5000, 'javier', 12000, 'diego']
 => 2
2.0.0p195 :370 > r.zrangebyscore 'users.karma', '-inf', 'inf', withscore:true
 => ["javier", "diego"]
2.0.0p195 :371 > r.zrank 'users.karma', 'diego'
 => 1
2.0.0p195 :372 > r.zscore 'users.karma', 'diego'
 => 12000.0
```

Sorted set commands

zadd, zcard, zcount
zincrby, zrank, zrem,zrevrank, zscore
zrange, zrangebyscore, zremrangebyrank, zrevrangebyscore, zrevrange, zrevrangebyscore
zinterstore, zunionstore

COUNTERS

Atomic counters can be safely invoked concurrently from anywhere

```ruby
class Tag < ActiveRecord::Base
  counter :global_items_counter

  (...)

  global_items_counter.increment
end
```

```ruby
module Guidable
  extend ActiveSupport::Concern
  include Redis::Objects

  included do
    @@guid_counter = Redis::Counter.new('guid_sequence')

    scope :by_guid, ->(guid) { where guid: self.raw_guid(guid) }

    before_save :assign_guid

  def assign_guid
    self.guid=next_guid unless self.guid.present?
  end

  def next_guid
    @@guid_counter.increment
  end

  #if a block is passed, the gui will be passed to the block.
  # Raise an exception in  the block to rewind the sequence
  def with_guid(&block)
    @@guid_counter.increment do |guid|
      yield guid
      guid
    end
  end
```

You can create global sequences with counters. Redis-objects block semantics allow for rewinding on errors

SCRIPTING WITH LUA*

You can use Lua for scripting Redis when you need to atomically execute a sequence of commands in which the output of a command is used as input for another

It reduces the need to use complex lock mechanisms and simplifies dependencies between clients

You can even extend the functionality of Redis by using Lua scripts

* no, mruby won't be supported

LUA EXAMPLE

Expiring attributes inside a Redis hash*

```
-- COMMAND: silenced_user
--
-- KEYS[1]: User nickname
-- ARGV[1]: Silenced user nickname
-- ARGV[2]: Timestamp

local user_key = "user:"..KEYS[1]..":silenced_users"
local nickname = ARGV[1]
local timestamp = tonumber(ARGV[2])
local res = redis.call('hget', user_key, nickname)
if res then
  if tonumber(res) > timestamp then
    return true
  else
    redis.call('hdel', user_key, nickname)
  end
end
return false
```

1

```
class User < ActiveRecord::Base
  def has_silenced?(nickname)
    ::Teowaki::Redis::Lua.silenced_user(self.id, nickname)
  end
```

```
module Teowaki
  module Redis
    class Lua
      # Based on https://github.com/cyx/redis-lua-playground/
      @cache = Hash.new { |h, cmd| h[cmd] = File.read(
                                File.join(Rails.root, 'lua', "#{cmd}.lua"))
                          }

      # Call lua script. If not exists, load first
      def self.call(command, keys, args=nil)
        begin
          redis.evalsha(sha(command), keys, args)
        rescue RuntimeError
          redis.eval(@cache[command], keys, args)
        end
      end

      def self.silenced_user(id, nickname)
        call('silenced_user', [id], [nickname, Time.now.to_i])
      end
```

5

3

4

2

TEMPORARY DATA

It's possible to set self-expiring keys

```
=> nil
2.0.0p195 :277 > r.setex 'lattest_tweet', 30, '@redis is cool'
 => "OK"
2.0.0p195 :278 > r.ttl 'lattest_tweet'
 => 26
2.0.0p195 :279 > r.persist 'lattest_tweet'
 => true
2.0.0p195 :280 > r.ttl 'lattest_tweet'
 => -1
```

```
2.0.0p195 :305 > r.setex 'user.1234.quota', 900, 90
 => "OK"
2.0.0p195 :306 > r.decr 'user.1234.quota'
 => 89
2.0.0p195 :307 > r.ttl 'user.1234.quota'
 => 876
```

Very simple to implement usage quota patterns
(even better if done in Lua)

REDIS AS A CACHE

Expire keys individually or turn off persistence and use Redis as a cache system with automatic eviction

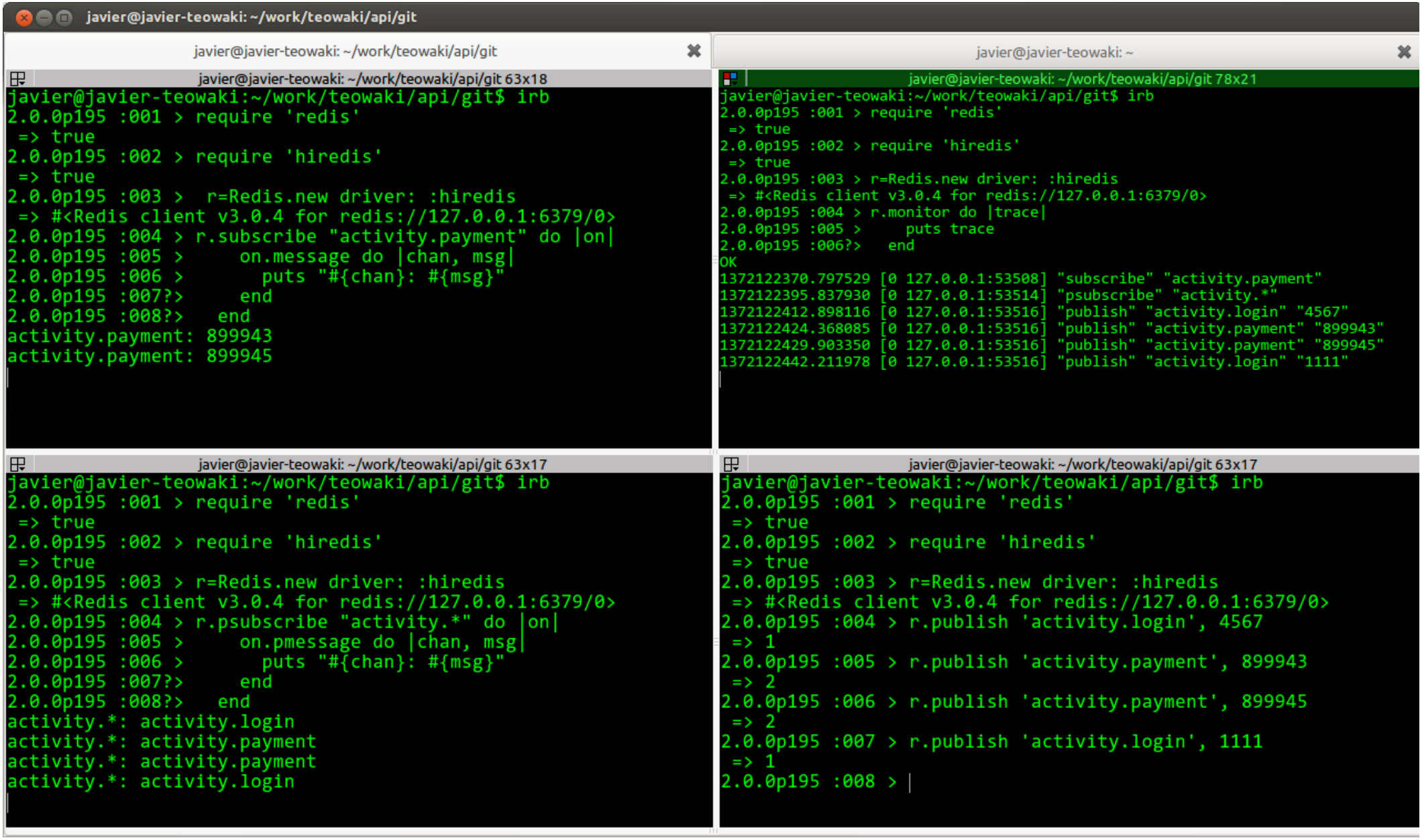
maxmemory 128mb
maxmemory-policy allkeys-lru

#save 900 1
#save 300 10
#save 60 10000

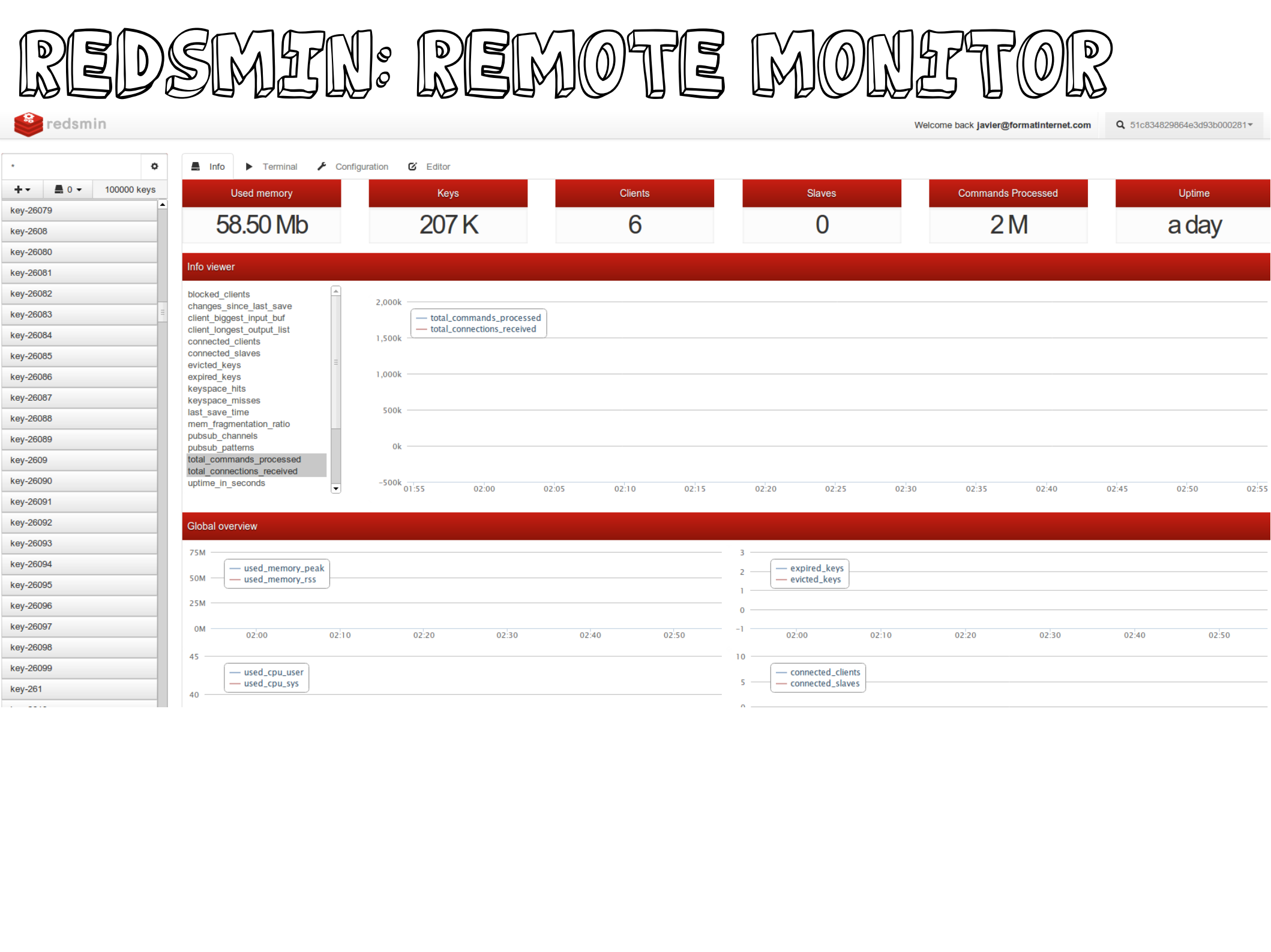
Multiple levels of cache by using Redis on the webserver/ middleware layer

http://wiki.nginx.org/HttpRedis
https://github.com/jodosha/redis-store

REDIS AS A PUBSUB SYSTEM



REDSMIN: REMOTE MONITOR



SHAMELESS SELF PROMOTION

If you enjoyed this presentation, please thank me by registering on **http://teowaki.com**

It's a site for developers, you can hang around for free, and I think it's quite cool

<3 <3 <3

Javier Ramírez
@supercoco9

teowaki