# Redis and Ruby

**Brian Kaney**
IRC / github / twitter / quora / etc : **bkaney**
Developer at **Vermonster** <vermonster.com>

# Haven't we heard Redis Talks Before?

- Yes, Redis is popular
- Why is this different?
  - Not covering Resque, Caching, Sessions or even Pub/Sub
  - Usage of Redis to store Ruby objects
  - Why would you want to do that?

# Quick Redis 101 - Datatypes

- Keys are strings, no need to quote
- Strings
- Integers (not really- INCR is a string operation)
- Lists
- Sets
- Sorted Sets (sets with a score)
- http://redis.oi (interactive terminal; website written in cuba.io, incidentally)

# Let's talk Ruby

- Ruby has strings, hashes, arrays..
- And hashes are many times used as data storage for Ruby objects..
- Wouldn't it be nice to use Redis to persist Ruby objects?

# Redis-rb

- Download and install redis http://redis.io/download
- Install ruby gem

```
$ gem install redis
```

- Connection

```
require 'redis'

db = Redis.new
```

# Ruby Serialization

*

- Suppose a simple hash

```
movie = Hash[
  :name, 'Top Gun',
  :genre, 'Action'
]
```

- Might be represented in redis as:

```
db.hmset 'movie:1', 'name', 'Top Gun', 'genre', 'Action'
```

# Auto-increment ID

- Notice incrementing in the key name

```
db.hmset 'movie:1', 'name', 'Top Gun', 'genre', 'Action'
```

- Create tooling to track id's, we can use INCR

```
def get_pk(item, db)
    pk = db.incr "#{item}:pk"
    "#{item}:#{pk}"
end

db.hmset get_pk('movies', db), 'name', 'Top Gun', ...
```

# References

- A reference, like in AR 'belongs_to'

- Could be implemented similarly:

```
db.hmset 'character:1', 'movie_id', 'movie:1'
```

# Has Many

- Embed as one of the keys

```
# Make a few jets
db.hmset 'jet:1', 'manufacturer', 'Grumman', 'model', 'F-14'
db.hmset 'jet:2', 'manufacturer', 'Northrop', 'model', 'F-5'

# Make our movie
db.hmset 'movie:1', 'jets', %w(jet:1 jet:2).to_json
```

# Has Many (set)

- But that's not really “redisy”
- How about conceiving a new key name and creating a set?

```
# Make a few jets
db.hmset 'jet:1', 'manufacturer', 'Grumman', 'model', 'F-14'
db.hmset 'jet:2', 'manufacturer', 'Northrop', 'model', 'F-5'

# Make our movie
db.hmset 'movie:1', 'name', 'Top Gun', 'genre', 'Action'

# Add to a key to represent the set of jets
db.sadd 'movie:1:jets', 'jet:1'
db.sadd 'movie:1:jets', 'jet:2'
```

# Find / Indexing

- Finding items, need an index.
- Indexes could be sets with prescribed keys.

```
# Create a few movies
db.hmset 'movie:1', 'name', 'Top Gun', 'genre', 'Action'
db.hmset 'movie:2', 'name', 'Top Gun', 'genre', 'Action'
db.hmset 'movie:3', 'name', 'Airplane', 'genre', 'Comedy'


# Create an index on 'genre'
db.sadd 'movie:index:genre:Action', 'movie:1'
db.sadd 'movie:index:genre:Action', 'movie:2'
db.sadd 'movie:index:genre:Comedy', 'movie:3'
```

Yikes!

# We need a Key Strategy

- Among other things...

# Nest

- Very simple convention

  **<klass.underscore>:<identifier>**

  movie:1

  jet:10

  character:iceman

- https://github.com/soveran/nest/blob/master/lib/nest.rb

# We need tooling to manage relationships

- Nest gives nice convention for key, but what about sets, lists, references, indexes, collections, validation, callbacks, etc...
- Enter Ohm
- https://github.com/soveran/ohm

# Ohm: Redis-Ruby Object Hash Mapper

- Ohm models for movies, characters and jets. (see code)

# Using Ohm (example)

(code)

# Devops - AOF

- AOF is a newish strategy for managing data.
- Configuration (redis.conf):

```
appendonly yes
appendfilename appendonly.aof
```

- Process to save is:
  - issue 'BGREWRITEAOF'
  - loop until complete
- Process to load is:
  - stop redis
  - copy AOF file to prescribed location (redis.conf)
  - loop until loading is complete

# Devops – AOF save

```
%x{ echo "BGREWRITEAOF" | redis-cli }
loop do
    sleep 1
    puts " * Waiting for aof to save..."
    save_status = %x{ echo "INFO" | redis-cli | grep bgrewriteaof_in_progress }[/\:([0-9])/,1]
    break if save_status == "0"
end
puts "DONE!"
```

# Devops – AOF load

```
loop do
    sleep 1
    puts " * Waiting for aof to load..."
    load_status = %x{ echo "INFO" | redis-cli | grep loading }[/\:([0-9])/,1]
    break if load_status == "0"
end
puts "DONE"
```

# Devops - Sockets

- Faster, no TCP overhead
- Configuration (redis.conf)

```
# port 6379
unixsocket /tmp/redis.sock
```

- Redis-cli

```
$ redis-cli -s /tmp/redis.sock
```

- Redis.rb

```
Redis.new(:path => '/tmp/redis.sock')
```

# Ohm::Contrib

https://github.com/sinefunc/ohm-contrib

- Ohm::Boundaries
- Ohm::Callbacks
- Ohm::Timestamping
- Ohm::ToHash
- Ohm::WebValidations
- Ohm::NumberValidations
- Ohm::ExtraValidations
- Ohm::Typecast
- Ohm::Locking

# Ohm::Contrib is hacky

```
ruby-1.9.2-p180 :016 > os = Ohm::Types::String.new "key"; s = "key"
ruby-1.9.2-p180 :018 > os == s
 => true
ruby-1.9.2-p180 :019 > os === s
 => true
ruby-1.9.2-p180 :020 > hsh = { os => 'foo' }
 => {"key"=>"foo"}
ruby-1.9.2-p180 :021 > hsh[os]
 => "foo"
ruby-1.9.2-p180 :022 > hsh[s]
 => nil
```

# Hacks - Overriding redis_id

```
require 'ohm'
class Ohm
  class Model
    # initialize: added ability to pass redis_id and have it define the actual Ohm ID.  (new? is broken in Ohm, BTW)
    alias_method :initialize_without_redis_id, :initialize
    def initialize_with_redis_id(args={})
      @redis_id = args.delete(:redis_id)
      initialize_without_redis_id(args)
    end
    alias_method :initialize, :initialize_with_redis_id

    def initialize_id
      @id ||= (@redis_id || self.class.key[:id].incr.to_s)
    end
    private :initialize_id
end
```

# Hacks – Adding list/set by id

- Default to have the Ohm saved, use <<
- We wanted to pass in id, since we prescribed them anyway.

```
class List
  def push_id(model_id)
    key.rpush(model_id)
  end
end
class Set
  def push_id(model_id)
    key.sadd(model_id)
  end
end
```

# Thanks