redis

python

Redis and redis-py

When and how to use Redis

10/09/2014

What is Redis

Has since changed his twitter pic

- Developed by Salvatore Sanfilippo (Italian) and initially released in April 2009
- In-memory key -> data structure datastore
- Five data structures including
  - Strings, Lists, Hashes, Sets, Sorted Sets + Publish Subscribe
- Built-in replication support
  - Master/Slave replication
  - Redis Cluster - in Beta now
- Support of persistence via snapshot (dump) or append only file
- Server-side Lua scripting
- Very basic string based protocol - demo?

What is Redis - Base Types

This is the real power!

- Strings
  - long int (int32, int64), double, char[] • think normal variable in Python
- Lists
  - Doubly-linked list of type char[] • think list in Python
- Hashes
  - Classical hash map of key char[] -> (long int, double, char[]) • think dictionary in Python
- Sets
  - Unordered, unique members of types char[] or long int • think set in Python
- Ordered Set (zset)
  - Implemented as a hash and a skip list*
  - Like a hash, but values are all doubles and returned in sorted order • think of collections.OrderedDict

This is as simple as it gets - very similar to memcached

- SET key value
  - sets key to value
- GET key
  - returns value of key
- SETEX key seconds value
  - sets key to value to expire in seconds seconds
- SETNX key value
  - sets key to value only if key does not already exist
- BITCOUNT, BITOP, BITPOS, GETBIT, SETBIT
  - bitwise operations
- INCR, INCRBY, DECR, DECRBY
  - increment and decrement value of key
- GETSET key value
  - set key to value and return the old value

Lists - useful commands

Doubly-linked list… perfect for a deque

- RPUSH key value [value …]
  - appends one or more values to key
- LPUSH key value [value …]
  - prepends one of more values to key
- RPOP / LPOP key
  - removes and gets the last / first element of key respectively
- LLEN key
  - get length of list
- LTRIM key start stop -> -1 as STOP
  - trim a list to the specified range
- BRPOP / BLPOP key timeout
  - blocking prop / lpop, waiting for value to be available - corresponding blpop

Hashes - useful commands

A single level dictionary for simple types

- HSET key field value
  - set the field of key to value
- HGET key field
  - get the value of the field of key
- HDEL key field
  - remove field from key
- HMSET key field value [field value …]
  - set multiple fields to corresponding values
- HMGET key field [field …]
  - get multiple fields from key
- HGETALL key
  - get all fields and values

Sets - useful commands

Super useful for any kind of membership

- SADD key member
  - add member to key
- SCARD key
  - get the number of members of a set
- SMEMBERS key
  - get all members of a set
- SREM key value
  - remove value from key
- SDIFF, SINTER, SUNION key [key …]
  - take the set difference, set intersection, or set union of 1-N sets
- SDIFFSTORE, SINTERSTORE, SUNIONSTORE source destination member
  - perform set operations and store the results
- SMOVE source destination member
  - move member from source to destination

Sorted Sets - useful commands

From personal experience: One of the most useful structures

- ZADD key score member
  - add member to key with score
- ZCARD key
  - get the number of members of a zset
- ZRANGE, ZREVRANGE key start stop WITHSCORES
  - get all members of a zset from index start to index stop (inclusive)
  - WITHSCORES adds the scores to the returned results
- ZRANGEBYSCORE, *ZREVRANGEBYSCORE key start stop WITHSCORES
  - get all members of a zest from scores ranging from start to stop
  - WITHSCORES adds the scores to the returned results
- ZREM key member
  - remove member from key
- ZRANK, *ZREVRANK key member
  - return an index position of a member

What is Redis - Pub / Sub

Speaking of pub, want to make sure I have a drink

- Clients subscribe to an explicit topic, or to a pattern based topic with wildcards
- Messages are ephemeral, if you aren't subscribed when it is published, you aren't going to receive it.
- Subscribe is blocking on the connection. When you are subscribed, you are only subscribed and can't do anything else with that connection.

When you should consider it

To be honest, you should always at least consider it…

- Task queues via lists
  - This is how the Celery implementation for a Redis backed message broker works
- Distributed locking
  - In use at Crowdstrike
- Analytics - counters
  - Timers too - especially async timers
- Caching - comparable in every sense to Memcached
  - Except you get nice data structures too!
- Messaging - via pub/sub
- Expiring date - via ttl
- Session storage
- Leaderboards with zsets. *Queue reminiscing over Call of Duty.

When you should avoid it

It doesn't solve all the things :(

- Data CAN NOT be lost
  - You can use AOF for every write, but this cripples Redis
- You have more data that you have RAM
  - There are some attempts to remove this requirement. But they all suck.
- Your data fits a relational model and requires JOINs
  - Simple relationships are ok, but complex JOINs are not going to happen
- You need ACID based transaction guarantees
  - WATCH/MULTI/EXEC get you somewhat there, and LUA scripting is kinda close, but none of it is ACID
- Required support - though this is becoming less an issue
- Multi-master required

Redis sharding options

More machines, more memory - hooray!

- Three ways to do sharing in Redis
- 1. Client side sharding.
  - Usually done via - crc32(key) % num_of_shards
- 2. Proxy based sharding. Proxy knows where to find the data. I even worked on one of these.
- 3. Redis Cluster. Not yet released, but in **Beta**

Redis cluster

RC1 Announced Today

Antirez weblog

Redis cluster, no longer vaporware.

antirez 8 hours ago.

The first commit I can find in my git history about Redis Cluster is dated March 29 2011, but it is a “copy and commit” merge: the history of the cluster branch was destroyed since it was a total mess of work-in-progress commits, just to shape the initial idea of API and interactions with the rest of the system.

Basically it is a roughly 4 years old project. This is about two thirds the whole history of the Redis project. Yet, it is only today, that I’m releasing a Release Candidate, the first one, of Redis 3.0.0, which is the first version with Cluster support.

An erratic run

—

What is Redis-Py

Besides being developed by Raven's fan

- Developed by Andy McCurdy and initially released almost 3 years ago
- 91 committers
- Supports connection pooling
  - And it does it well
- Unifies a somewhat disjoint API
  - Like the old zadd flip flop…
- Is thread safe
  - Doesn't support SELECT command
- Supports 100% of the Redis commands including MULTI / EXEC, PUBSUB, and LUA scripting

Strings - a small demo

Let's set some values… let's read some values. Nobody gets hurt.

```
from __future__ import print_function
import redis
import random

def demo_string(conn, key):
    mapping = {}
    for i in range(0, 1000):
        pkey = "%s_string_%d" % (key, i)
        val = random.randint(0, 10000)
        conn.set(pkey, val)
        mapping[pkey] = val
    for key, val in mapping.items():
        retrieved = conn.get(key)
        print("Retrieved %s for %s which should be %s" % (retrieved, key, val))


def main():
    conn = redis.Redis("vm")
    demo_string(conn, "foo_1")

if __name__ == "__main__":
    main()
```

Lists - a small demo

Doubly-linked list… perfect for a deque

```
def demo_list(conn, key):
    mapping = {}
    pkey = "%s_list" % (key,)
    vals = []
    for i in range(0, 10):
        val = random.randint(0, 10000)
        vals.append(val)
        conn.lpush(pkey, val)
    retrieved = []
    while conn.llen(pkey):
        right_item = conn.rpop(pkey)
        retrieved.append(int(right_item))
    print("Original Values: ", vals)
    print("Retrieved Values:", retrieved)

def main():
    conn = redis.Redis("vm")
    demo_list(conn, "foo_1")
```

Hashes - a small demo

A single level dictionary for simple types

```
from __future__ import print_function
import redis
import random

def demo_hash(conn, key):
    mapping = {}
    pkey = "%s_list" % (key,)
    for i in range(0, 10):
        val = random.randint(0, 10000)
        mapping[val*2] = val
    conn.delete(pkey)
    conn.hmset(pkey, mapping)
    retrieved = {int(k): int(v) for k, v in conn.hgetall(pkey).items()}
    print("Original Values: ", sorted(mapping.items()))
    print("Retrieved Values:", sorted(retrieved.items()))

def main():
    conn = redis.Redis("vm")
    demo_hash(conn, "foo_1")

if __name__ == "__main__":
    main()
```

Sets - a small demo

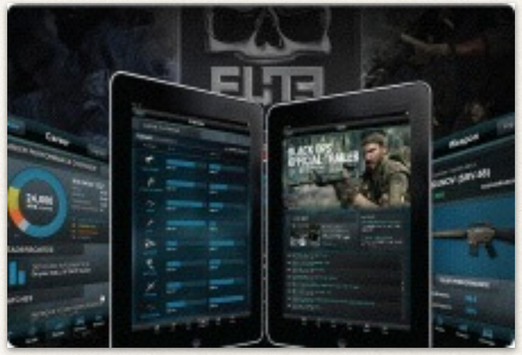
Super useful for any kind of membership

```
def demo_sets(conn, key):
    items_1, items_2 = [[random.randint(0, 1000) for _ in range(10)] for _ in range(2)]
    pkey_1, pkey_2 = ["%s_set_%d" % (key,i) for i in range(2)]
    conn.delete(pkey_1)
    conn.delete(pkey_2)
    conn.sadd(pkey_1, *items_1)
    conn.sadd(pkey_2, *items_2)
    retrieved_1 = set(map(int, conn.smembers(pkey_1)))

    assert(retrieved_1 == set(items_1))

    py_diff = set(items_1) - set(items_2)
    py_intersection = set(items_1) & set(items_2)
    py_union = set(items_1) | set(items_2)

    redis_diff = set(map(int, conn.sdiff(pkey_1, pkey_2)))
    redis_intersection = set(map(int, conn.sinter(pkey_1, pkey_2)))
    redis_union = set(map(int, conn.sunion(pkey_1, pkey_2)))

    assert(redis_diff == py_diff)
    assert(redis_intersection == py_intersection)
    assert(redis_union == py_union)
```

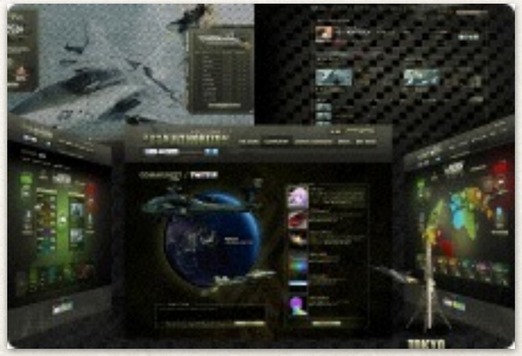
Sorted Sets - a small demo

From personal experience: One of the most useful structures

```
members = ["Sean", "Jim", "JK", "Jmac", "Leif", "Andrew", "Ross", "Adam", "Marcus"]

def demo_zsets(conn, key):
    pkey = "%s_zset" % (key,)
    conn.delete(pkey)
    conn.zadd(pkey, members[0], 10000)
    for m in members[1:]:
        conn.zadd(pkey, m, random.randint(0, 10000) - 1)
    leaderboards = conn.zrevrange(pkey, 0, -1, withscores=True)
    for i, (name, score) in enumerate(leaderboards, 1):
        print("{}: {:10} {}".format(i, name, score))
```



```
1: Sean       10000.0
2: Leif       8536.0
3: Jmac       7807.0
4: JK         7120.0
5: Adam       6823.0
6: Ross       4219.0
7: Andrew     3245.0
8: Marcus     2929.0
9: Jim        167.0
```



PubSub - a small demo

Notify all the things!

```
from __future__ import print_function
import redis
import random
import time

def demo_pub(conn, key):
    for i in range(0, 10):
        conn.publish(key, i)
        time.sleep(1)
    conn.publish(key, "END")

def main():
    conn = redis.Redis("vm")
    demo_pub(conn, "demo_pub")

if __name__ == "__main__":
    main()
```

```
from __future__ import print_function
import redis
import random

def demo_sub(conn, key):
    sub = conn.pubsub()
    sub.subscribe(key)
    for msg in sub.listen():
        print(msg)
        if msg.get('data') == 'END':
            print("Completing work")
            return

def main():
    conn = redis.Redis("vm")
    demo_sub(conn, "demo_pub")

if __name__ == "__main__":
    main()
```

Changed in newer version

We’re Hiring!

- Great pay, benefits, stock options, and bonus plan
- Top notch developers
- Challenging problems
- Large scale distributed cloud architecture
- We use Python
  - along with Scala, Go and C++
- “Hilarious quotes” wiki page from IRC logs

crowdstrike.com/about-us/careers/