redset Documentation

Release 0.4

jamesob, thekantian

January 13, 2014

Contents

CHAPTER 1

Introduction

Redset offers simple objects that mimic Python's builtin set, but are backed by Redis and safe to use concurrently across process boundaries. Time-sorted sets come included and are particularly interesting for use when time-sensitive and duplicate tasks are being generated from multiple sources.

Traditional queuing solutions in Python don't easily allow users to ensure that messages are unique; this is especially important when you're generating a lot of time-consumptive tasks that may have overlap. Redset can help with this, among other things.

Redset doesn't mandate the use of any particular consumer process or client. Production and consumption can happen easily from any piece of Python, making usage flexible and lightweight.

CHAPTER 2

Quick example

```
>>> import json

>>> ss = TimeSortedSet(redis.Redis(), 'json_biz', serializer=json)

>>> ss.add({'foo': 'bar1'}, score=123)
123

>>> {'foo': 'bar1'} in ss
True

>>> ss.score({'foo': 'bar1'})
123

>>> ss.pop()
{'foo': 'bar1'}
```

Redset is designed to be simple and pleasant to use. It was written at Percolate where it's used for all sorts of things, especially storing sets of time-sensitive tasks.

CHAPTER 3

Contents

3.1 Examples

3.1.1 Task system

Here's an example that shows how to construct a very basic multi-producer, multi-consumer prioritized task system.

```
import json
from collections import namedtuple
import redset, redis
from redset.serializers import NamedtupleSerializer

Task = namedtuple('Task', 'foo,bar,priority')

task_set = redset.SortedSet(
    redis.Redis(),
    'tasks',
    scorer=lambda task: task.priority,
    serializer=NamedtupleSerializer(Task),
)
```

Now we can produce from anywhere:

```
task_set.add(Task('yo', 'baz', 1))
task_set.add(Task('hey', 'roar', 0))
```

And maybe have a daemon that consumes:

```
def process_tasks():
    while True:
        for task in task_set.take(10):
            do_work_on_task(task)
        sleep(1)
```

3.2 API

3.2.1 Introduction

Redset offers a narrow interface consisting of a few objects. Most often, you'll be using an object that resembles a set.

There are two interesting components to the sets in redset: *scorers* and *serializers*.

Scorers determine what score the inserted item will be assigned (lower means popped sooner). Serializers determine what transformations happen on an item going into and coming out of redis.

Note: If an Exception is thrown while deserialization is attempted on a particular item, `None` will be returned in its stead and an exception will be logged. In the case of `SortedSet.take()`, the failed item will simply be filtered from the returned list.

3.2.2 Sets

class `redset.`**`SortedSet`**(*redis_client*, *name*, *scorer=None*, *serializer=None*, *lock_timeout=None*, *lock_expires=None*)

A Redis-backed sorted set safe for multiprocess consumption.

By default, items are stored and returned as str. Scores default to 0.

A serializer can be specified to ease packing/unpacking of items. Otherwise, items are cast to and returned as strings.

`__init__`(*redis_client*, *name*, *scorer=None*, *serializer=None*, *lock_timeout=None*, *lock_expires=None*)

Parameters

- **redis_client** (*redis.Redis instance*) – an object matching the interface of the redis.Redis client. Used to communicate with a Redis server.
- **name** (*str*) – used to identify the storage location for this set.
- **scorer** (*Callable, arity 1*) – takes in a single argument, which is the item to be stored, and returns a score which will be used for the item.
- **serializer** (`interfaces.Serializer`) – must match the interface defined by *redset.interfaces.Serializer*. Defines how objects are marshalled into redis.
- **lock_timeout** (*Number*) – maximum time we should wait on a lock in seconds. Defaults to value set in `locks.Lock`
- **lock_expires** (*Number*) – maximum time we should hold the lock in seconds Defaults to value set in `locks.Lock`

`__len__`()

How many values are the in the set?

Returns int

`__contains__`(*item*)

`add`(*item*, *score=None*)

Add the item to the set. It the item is already in the set, update its score.

Parameters

- **item** (*str*) –
- **score** (*Number*) – optionally specify the score for the item to be added.

Returns Number – score the item was added with

`clear`()

Empty the set of all scores and ID strings.

Returns bool

discard(*item*)
Remove a given item from the set.

Parameters **item** (*object*) –

Returns bool – success of removal

name
The name of this set and the string that identifies the redis key where this set is stored.

Returns str

peek()
Return the next item eligible for processing without removing it.

Raises KeyError – if no items left

Returns object

peek_score()
What is the score of the next item to be processed? This is interesting if you are trying to keep your set at real-time consumption.

Returns Number

pop()
Atomically remove and return the next item eligible for processing in the set.

If, for some reason, deserializing the object fails, None is returned and the object is deleted from redis.

Raises KeyError – if no items left

Returns object.

score(*item*)
See what the score for an item is.

Returns Number or None.

take(*num*)
Atomically remove and return the next `num` items for processing in the set.

Will return at most `min(num, len(self))` items. If certain items fail to deserialize, the falsey value returned will be filtered out.

Returns list of objects

3.2.3 Specialized sets

The only builtin concrete subclasses of `SortedSet` are sorted sets relating to time. One class maintains order based on time (`TimeSortedSet`) and the other does the same, but won't return items until their score is less than or equal to now (`ScheduledSet`).

class `redset.`**TimeSortedSet**(**args*, ***kwargs*)
Bases: `redset.sets.SortedSet`

A distributed, FIFO-by-default, time-sorted set that's safe for multiprocess consumption.

Implemented in terms of a redis ZSET where UNIX timestamps are used as the score.

__init__(**args*, ***kwargs*)
See *redset.sets.SortedSet*. Default scorer will return the current time when an item is added.

This ScheduledSet allows you to schedule items to be processed strictly in the future, which allows you to easily implement backoffs for expensive tasks that can't be repeated continuously.

class redset.**ScheduledSet**(**args*, ***kwargs*)
Bases: redset.sets.TimeSortedSet

A distributed, FIFO-by-default, time-sorted set that's safe for multiprocess consumption. Supports scheduling item consumption for the future.

Implemented in terms of a redis ZSET where UNIX timestamps are used as the score.

A ScheduledSet will only return results with a score less than time.time() - to enable you to schedule jobs for the future and let redis do the work of defering them until they are ready for consumption.

__init__(**args*, ***kwargs*)
See *redset.sets.SortedSet*. Default scorer will return the current time when an item is added.

3.2.4 Interfaces

The Serializer interface is included as a guideline for end-users. It need not be subclassed for concrete serializers.

class redset.interfaces.**Serializer**
This is a guideline for implementing a serializer for redset. Serializers need not subclass this directly, but should match the interface defined here.

dumps(*obj*)
Serialize a Python object into a *str*

Parameters **obj** – the Python object to be stored in a sorted set

Returns str

loads(*str_from_redis*)
Deserialize a str item from redis into a Python object.

Parameters **str_from_redis** (*str*) – the str corresponding with an item in redis

Returns object

3.2.5 Builtin serializers

One serializer is included for convenience, and that's redset.serializers.NamedtupleSerializer, which allows seamless use of namedtuples.

class redset.serializers.**NamedtupleSerializer**(*NTClass*)
Serialize namedtuple classes.

__init__(*NTClass*)

Parameters **NTClass** (*type*) – the namedtuple class that you'd like to marshal to and from.

dumps(*nt_instance*)

loads(*str_from_redis*)

CHAPTER 4

Authors

Written by jamesob and thekantian.

r