
retask Documentation

Release 0.1

Kushal Das

July 04, 2012

CONTENTS

1	Dependencies	3
2	User Guide	5
2.1	Introduction	5
2.2	Setting up the Redis Server	5
2.3	Quickstart	6
3	API Documentation	7
3.1	<code>retask.task</code>	7
3.2	<code>retask.queue</code>	7
4	Indices and tables	9

retask is a python module to create and manage distributed task queues.

It uses [Redis](#) to create task queues. User can enqueue and dequeue tasks in the queues they manage. Each task can contain any serializable python objects. We use *JSON* internally to store the tasks in the queues.

DEPENDENCIES

- python-redis
- mock
- A running Redis server

USER GUIDE

2.1 Introduction

2.1.1 First requirement

For various others projects I had to start looking for a simple task queue and solve kind of classical producer-consumer problems using them.

This project started from that idea.

2.1.2 Why Redis

I am following [Redis](#) development for a time and using it in various other projects. The simplicity it provides and rich datastructures are always a plus to use it. It is also very fast.

2.1.3 Retask License

Copyright (C) 2012, Kushal Das

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.2 Setting up the Redis Server

You can download and install [Redis](#) on your distro. In [Fedora](#) you can just `yum install redis` for the same.

To start the server in the local folder use the following command:

```
$ redis-server
```

On Fedora you can start the service as *root*:

```
# service redis start
```

2.3 Quickstart

For this example to work you should have your redis instance up and running.

2.3.1 producer.py

This code puts new task in the queue. We will have a dictionary as the information in this example.

```
from retask.task import Task
from retask.queue import Queue
queue = Queue('example')
info1 = {'user': 'kushal', 'url': 'http://kushaldas.in'}
info2 = {'user': 'fedora planet', 'url': 'http://planet.fedoraproject.org'}
task1 = Task(info1)
task2 = Task(info2)
queue.connect()
queue.enqueue(task1)
queue.enqueue(task2)
```

2.3.2 consumer.py

This code gets the tasks from the queue. Based on the actual requirement, the client will work on the information it received as the task. For now we will just print the data.

```
from retask.task import Task
from retask.queue import Queue
queue = Queue('example')
queue.connect()
while queue.length != 0:
    task = queue.dequeue()
    print task.data
```

API DOCUMENTATION

3.1 `retask.task`

This module contains generic task class, which can be used to create any kind of given task with serializable python objects as data.

class `retask.task.Task` (*data=None, raw=False*)

Returns a new Task object, the information for the task is passed through argument `data`

Parameters `data` – Python object which contains information for the task. Should be serializable through `JSON`.

data

The python object containing information for the current task

rawdata

The string representation of the actual python objects for the task

Note: This should not be used directly by the users. This is for internal use only.

3.2 `retask.queue`

This module contains the primary Queue which can be used to create and manage queues.

class `retask.queue.Queue` (*name, config={}*)

Returns the Queue object with the given name. If the user passes optional config dictionary with details for Redis server, it will connect to that instance. By default it connects to the localhost.

connect ()

Creates the connection with the redis server. Return `True` if the connection works, else returns `False`. It does not take any arguments.

Returns Boolean value

Note: After creating the Queue object the user should call the `connect` method so create the connection.

```
>>> from retask.queue import Queue
>>> q = Queue('test')
```

```
>>> q.connect()
True
```

dequeue()

Returns a `Task` object from the queue. Returns `None` if the queue is empty.

Returns `Task` object from the queue

```
>>> from retask.queue import Queue
>>> q = Queue('test')
>>> q.connect()
True
>>> t = q.dequeue()
>>> print t.data
{'name': 'kushal'}
```

enqueue(task)

Enqueues the given `Task` object to the queue and returns a tuple. Value in index 0 is `Boolean` explaining the enqueue operation is a success or not. Value at index 1 is string with error/success message (if any).

Parameters `task` – `Task` object

Returns Tuple with `Boolean` value and string message.

```
>>> from retask.queue import Queue
>>> q = Queue('test')
>>> q.connect()
True
>>> from retask.task import Task
>>> task = Task({'name': 'kushal'})
>>> q.enqueue(task)
(True, 'Pushed')
```

length

Gives the length of the queue. Returns `None` if the queue is not connected.

INDICES AND TABLES

- *genindex*