
retask Documentation

Release 0.1

Kushal Das

December 10, 2012

CONTENTS

retask is a python module to create and manage distributed task queue/job queue.

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

Workers can execute tasks and return the result asynchronously and if required synchronously (wait until worker returns the result to the job).

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 long time and using it in various other projects. The simplicity it provides and rich datastructures are always a plus to use it.

Redis takes care of multithreading issues, it also helps to have data on disk for data persistence. It does not have any external dependencies and also very small in size, which helps to be used in enterprise world.

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*:

```
# systemctl enable redis.service
# systemctl start redis.service
```

2.3 Installation

This part of the documentation covers the installation of Retask. The first step to using any software package is getting it properly installed.

2.3.1 Distribute & Pip

Installing requests is simple with [pip](#):

```
$ pip install retask
```

or, with [easy_install](#):

```
$ easy_install retask
```

But, you really [shouldn't do that](#).

2.3.2 Get the Code

Retask is actively developed on GitHub, where the code is [always available](#).

You can either clone the public repository:

```
git clone git://github.com/kushaldas/retask.git
```

Download the [tarball](#):

```
$ curl -OL https://github.com/kushaldas/retask/tarball/master
```

Or, download the [zipball](#):

```
$ curl -OL https://github.com/kushaldas/retask/tarball/master
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

```
$ python setup.py install
```

2.3.3 Installing redis-py

You can install `redis-py` with `pip`:

```
$ pip install redis
```

2.4 Quickstart

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

2.4.1 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)
```

2.4.2 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
```

2.5 Tutorials

This section of the document we have in depth examples of various use cases.

2.5.1 Async data transfer between producer and worker

In many real life scenarios we need to send the result back from the worker instances to the producer. The following code examples shows how to achieve that.

async_producer.py

```
from retask.task import Task
from retask.queue import Queue
import time
queue = Queue('example')
info1 = {'user': 'Fedora planet', 'url': 'http://planet.fedoraproject.org'}
task1 = Task(info1)
queue.connect()
job = queue.enqueue(task1)
print job.result
time.sleep(30)
print job.result
```

Here `queue.enqueue` method returns a `Job` object. We can access `job.result` to see returned result from a worker. If there is no result yet came back from the worker, it will print *None*. If you don't need any returned data from the worker you can safely ignore the job object.

async_consumer.py

```
from retask.task import Task
from retask.queue import Queue
import time
queue = Queue('example')
queue.connect()
task = queue.wait()
print task.data
time.sleep(15)
queue.send(task, "We received your information dear %s" % task.data['user'])
```

In the above example we see two newly introduced methods `Queue.wait()` is a blocking call to wait for a new task in the queue. This is the preferred method over polling using `dequeue()`. To send the result back workers will use `send()` method, which takes an optional argument `wait_time` to specify timeout value in seconds.

2.5.2 Synchronous / blocking wait for the result

```
from retask.task import Task
from retask.queue import Queue
queue = Queue('example')
info1 = {'user': 'Fedora planet', 'url': 'http://planet.fedoraproject.org'}
task1 = Task(info1)
queue.connect()
job = queue.enqueue(task1)
job.wait()
print job.result
```

In this example we are using `wait()` function to do a blocking synchronous call to the worker.

API DOCUMENTATION

3.1 API

This part contains the API documentation of the module.

3.1.1 Submodules

`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

If the queue is not connected then it will raise `retask.ConnectionError`

```
>>> 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 `Job` object.

Parameters `task` – `Task` object

Returns `Job` object

If the queue is not connected then it will raise `retask.ConnectionError`.

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

find (*obj*)

Retruns the index of the given object in the queue, it might be string which will be searched inside each task.

Parameters `obj` – object we are looking

Returns -1 if the object is not found or else the location of the task

length

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

If the queue is not connected then it will raise `retask.ConnectionError`.

send (*task, result, expire=60*)

Sends the result back to the producer. This should be called if only you want to return the result in async manner.

Parameters

- `task` – `Task` object
- `result` – Result data to be send back. Should be in JSON serializable.
- `expire` – Time in seconds after the key expires. Default is 60 seconds.

wait (*wait_time=0*)

Returns a `Task` object from the queue. Returns `False` if it timeouts.

Parameters `wait_time` – Time in seconds to wait, default is infinite.

Returns `Task` object from the queue or `False` if it timeouts.

```
>>> from retask.queue import Queue
>>> q = Queue('test')
>>> q.connect()
True
>>> task = q.wait()
>>> print task.data
{'name': 'kushal'}
```

Note: This is a blocking call, you can specity `wait_time` argument for timeout.

class `retask.queue.Job(rdb)`

Job object containing the result from the workers.

Parameters `rdb` – The underlying redis connection.

result

Returns the result from the worker for this job. This is used to pass result in async way.

wait (`wait_time=0`)

Blocking call to check if the worker returns the result. One can use `job.result` after this call returns `True`.

Parameters `wait_time` – Time in seconds to wait, default is infinite.

Returns *True* or *False*.

Note: This is a blocking call, you can specify `wait_time` argument for timeout.

`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, urn=None)`

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.

3.1.2 Exceptions

exception `retask.RetaskException`

Some ambiguous exception occurred

exception `retask.ConnectionError`

A Connection error occurred.

INDICES AND TABLES

- *genindex*

PYTHON MODULE INDEX

r

retask, ??