amqppy

Release 0.0.19

Dec 29, 2017

Contents

amqppy is a very simplified AMQP client stacked over Pika. It has been tested with RabbitMQ, however it should also work with other AMQP 0-9-1 brokers.

The motivation of **amqppy** is to provide a very simplified and minimal AMQP client interface which can help Python developers to implement easily messaging patterns such as:

- Topic Publisher-Subscribers
- RPC Request-Reply

Others derivative messaging patterns can be implemented tunning some parameters of the Topic and Rpc objects.

CHAPTER 1

Installing amqppy

amqppy is available for download via PyPI and may be installed using easy_install or pip:

```
pip install amqppy
```

To install from source, run “python setup.py install” in the root source directory.

CHAPTER 2

Home of amqppy

https://github.com/marceljanerfont/amqppy

CHAPTER 3

Using amqppy

3.1 Usage Examples

It requires an accessible RabbitMQ and a Python environment with the **amqppy** package installed.

3.1.1 Topic Publisher-Subscribers

This is one of the most common messaging pattern where the publisher publishes message to an AMQP exchange and the subscriber receives only the messages that are of interest. The subscriber's interest is modeled by the *Topic* or in terms of AMQP by the **rounting_key**.

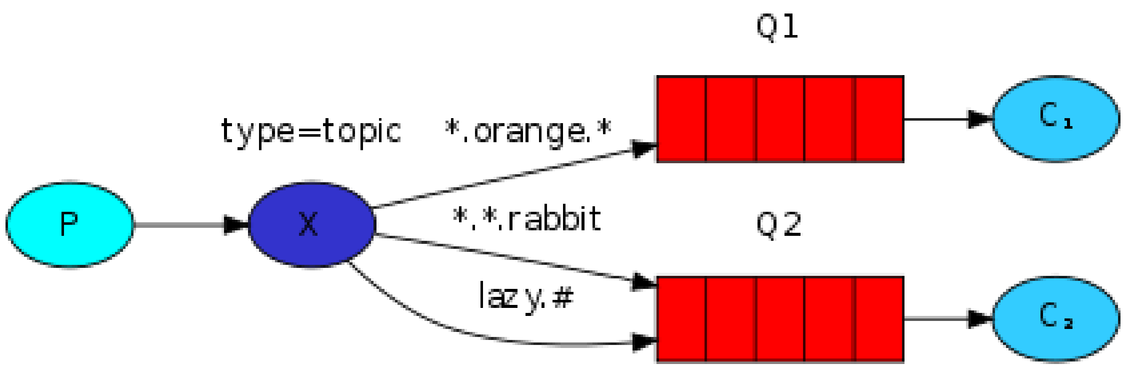

Image from RabbitMQ Topic tutorial.

Topic Subscriber

Firstly, we need to start the Topic Subscriber (*also known as Consumer*). In **amqppy** the class **amqppy.consumer.Worker** has this duty.

```
import amqppy
BROKER = 'amqp://guest:guest@localhost:5672//'

def on_topic_status(exchange, routing_key, headers, body):
    print('Received message from topic \'amqppy.publisher.topic.status\': {}'.
↪format(body))

# subscribe to a topic: 'amqppy.publisher.topic.status'
worker = amqppy.Worker(BROKER)
worker.add_topic(exchange='amqppy.test',
                 routing_key='amqppy.publisher.topic.status',
                 on_topic_callback=on_topic_status)
# it will wait until worker is stopped or an uncaught exception
worker.run()
```

The subscriber worker will invoke the *on_topic_callback* every time a message is published with a topic that matches with the specified **routing_key**: *'amqppy.publisher.topic.status'*. Note that **routing_key** can contain wildcards therefore, one subscriber might be listening a set of *Topics*.

Once the topic subscriber is running we able to launch the publisher.

Topic Publisher

```
import amqppy
BROKER = 'amqp://guest:guest@localhost:5672//'

# publish my current status
amqppy.Topic(BROKER).publish(exchange='amqppy.test',
                             routing_key='amqppy.publisher.topic.status',
                             body='RUNNING')
```

The topic publisher will send a message to the AMQP exchange with the Topic **routing_key**: *'amqppy.publisher.topic.status'*, therefore, all the subscribed subscribers will receive the message unless they do not share the same queue. In case they share the same queue a round-robin dispatching policy would be applied among subscribers/consumers like happens in 'work queues <https://www.rabbitmq.com/tutorials/tutorial-two-python.html>'_*.

3.1.2 RPC Request-Reply

This pattern is commonly known as *Remote Procedure Call* or *RPC*. And is widely used when we need to run a function *request* on a remote computer and wait for the result *reply*.

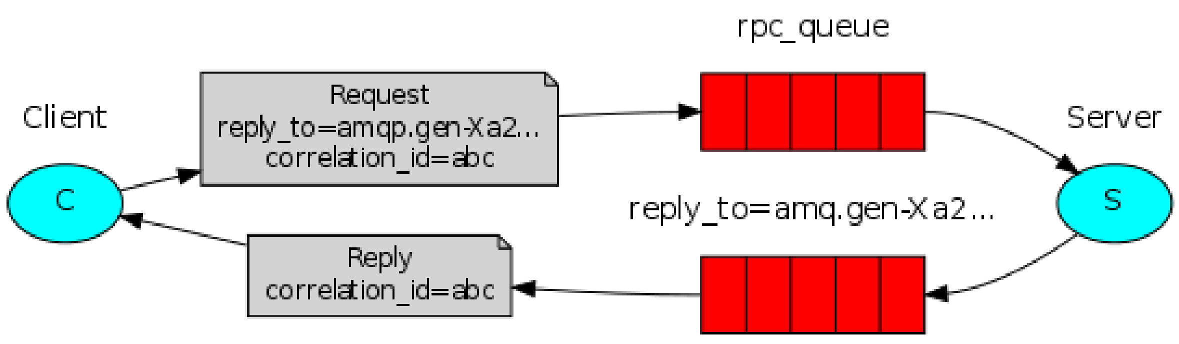

Image from RabbitMQ RPC tutorial

RPC Reply

An object of type **amqppy.consumer.Worker** listens incoming **RPC requests** and computes the **RPC reply** in the *on_request_callback*. In the example below, the RPC consumer listens on Request **rounting_key**:*'amqppy.requester.rpc.division'* and the division would be returned as the RPC reply.

```
import amqppy
BROKER = 'amqp://guest:guest@localhost:5672//'

def on_rpc_request_division(exchange, routing_key, headers, body):
    args = json.loads(body)
    return args['dividend'] / args['divisor']

# subscribe to a rpc request: 'amqppy.requester.rpc.division'
worker = amqppy.Worker(BROKER)
worker.add_request(exchange='amqppy.test',
                   routing_key='amqppy.requester.rpc.division',
                   on_request_callback=on_rpc_request_division)
# it will wait until worker is stopped or an uncaught exception
worker.run()
```

RPC Request

The code below shows how to do a **RPC Request** using an instance of class *amqppy.publisher.Rpc*

```
import amqppy
BROKER = 'amqp://guest:guest@localhost:5672//'

# do a Rpc request 'amqppy.requester.rpc.division'
result = amqppy.Rpc(BROKER).request(exchange='amqppy.test',
                                    routing_key='amqppy.requester.rpc.division',
                                    body=json.dumps({'dividend': 3.23606797749979,
↪'divisor': 2.0}))
print('RPC result: {}.'.format(result))
```

3.2 Core Class and Module Documentation

3.2.1 Topic

The *Topic* class for Topic publications.

class amqppy.publisher.**Topic**(*broker*)

This class creates a connection to the message broker and provides a method to publish messages on one topic also known as routing_key in AMQP terms. The class instance will create on single connection to the broker for all the topic published.

Parameters **broker** (*str*) – The URL for connection to RabbitMQ. Eg: 'amqp://serviceuser:password@rabbit.host:5672//'

publish(*exchange*, *routing_key*, *body*, *headers=None*, *persistent=True*)

Publish a message to the given exchange and a routing key.

Parameters

- **exchange** (*str*) – The exchange you want to publish the message.
- **rounting_key** (*str*) – The rounting key to bind on
- **body** (*str*) – The body of the message you want to publish. It should be of type unicode or string encoded with UTF-8.
- **headers** (*dict*) – Message headers.
- **persistent** (*bool*) – Makes message persistent. The message would not be lost after RabbitMQ restart.

3.2.2 Rpc

The *Rpc* class for Rpc requests.

class amqppy.publisher.**Rpc**(*broker*)

The duty of Rpc class is to make RPC requests and returns its responses. RPC pattern tutorial. The class instance will create on single connection to the broker for all the RPC requests.

Parameters **broker** (*str*) – The URL for connection to RabbitMQ. Eg: 'amqp://serviceuser:password@rabbit.host:5672//'

request(*exchange*, *routing_key*, *body*, *timeout=10.0*)

Makes a RPC request and returns its response. RPC pattern tutorial. This call creates and destroys a connection every time, if you want to save connections, please use the class Rpc.

Parameters

- **rounting_key** (*str*) – The routing key to bind on
- **body** (*str*) – The body of the message request you want to request. It should be of type unicode or string encoded with UTF-8.
- **exchange** (*str*) – The exchange you want to publish the message.
- **timeout** (*bool*) – Maximum seconds to wait for the response.

3.2.3 Worker

The `Worker` class for Topic subscription and Rpc Replies.

class `amqppy.consumer.`**`Worker`**(*broker*, *heartbeat_sec=None*)

This class handles a worker that listens for incoming Topics and Rpc requests.

Parameters **`broker`** (*`str`*) – The URL for connection to RabbitMQ. Eg: 'amqp://serviceuser:password@rabbit.host:5672//'

`add_request`(*routing_key*, *on_request_callback*, *exchange='amqppy'*, *durable=False*, *auto_delete=True*, *exclusive=False*)

Registers a new consumer for a RPC reply task. These tasks will be executed when a RPC request is invoked by publisher.Rpc.request().

Parameters

- **`rounting_key`** (*`str`*) – It defines the subscription interest. In terms of AMQP the routing key to bind on
- **`on_request_callback`** (*`method`*) – Called when a Rpc request is invoked, it should return the reply.
- **`exchange`** (*`str`*) – The exchange you want to publish the message.
- **`durable`** (*`bool`*) – Queue messages survives a reboot of RabbitMQ.
- **`auto_delete`** (*`bool`*) – Queues will auto-delete after use.
- **`exclusive`** (*`bool`*) – Ensures that is the unique consumer

`add_topic`(*routing_key*, *on_topic_callback*, *queue=None*, *exclusive=False*, *exchange='amqppy'*, *durable=False*, *auto_delete=True*, *no_ack=True*, ***kwargs*)

Registers a new consumer for a Topic subscriber. These tasks will be executed when a Topic is published by publisher.Topic.publish().

Parameters

- **`rounting_key`** (*`str`*) – The routing key to bind on.
- **`on_topic_callback`** (*`method`*) – Called when a topic is published.
- **`queue`** (*`str`*) – The name of the queue. If it is not provided the queue will be named the same as the 'routing_key'.
- **`exclusive`** (*`bool`*) – Only one consumer is allowed.
- **`exchange`** (*`str`*) – The exchange you want to publish the message.
- **`durable`** (*`bool`*) – Queue messages survives a reboot of RabbitMQ.
- **`auto_delete`** (*`bool`*) – Queues will auto-delete after use.
- **`no_ack`** (*`bool`*) – Tell the broker that ACK reply is not needed. If it is False, an ACK will be sent automatically each time a message is consumed unless a amqppy.AbortConsume or amqppy.DeadLetterMessage is raised.

`run`()

Start worker to listen. This will block the execution until the worker is stopped or an uncaught Exception

`run_async`()

Start asynchronously worker to listen. The execution thread will follow after this call, hence is not blocked.

`stop`()

Stops listening and close all channels and the connection

3.2.4 Exceptions

class amqppy.**RpcRemoteException**

It would be raised in the publisher.Rpc.request() when remote reply fails

class amqppy.**ResponseTimeout**

It would be raised in the publisher.Rpc.request() when remote reply exceeds its allowed execution time, the timeout.

class amqppy.**PublishNotRouted**

It would be raised in the publisher.Rpc.request() or publisher.Topic.publish() when there is no consumer listening those Topics or Rpc requests.

class amqppy.**ExclusiveQueue**

It would be raised in the consumer.Worker.add_topic() or consumer.Worker.add_request() when tries to consume from a queue where there is already a consumer listening. That happens when add_topic or add_request is called with 'exclusive=True'.

class amqppy.**ExchangeNotFound**

It will be raised when AMQP Exchange does not exist.

class amqppy.**AbortConsume**

This exception can be raised by the Topic callback or RPC reply callback. And indicates to amqppy to do not send ACK for that consuming message. For this, is required 'no_ack=False' in consumer.Worker.add_topic() or consumer.Worker.add_request()

class amqppy.**DeadLetterMessage**

This exception can be raised by the Topic callback or RPC reply callback. And indicates to amqppy to move this message is being consumed to the DeadLetter Queue. See: 'https://www.rabbitmq.com/dlx.html'

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index