subpub

Release 1.0.0

Daniel Andersson

Jun 12, 2021

CONTENTS:

1 Example 3

2 Key features 5

3 Installation 7

4 Reference 9

Python Module Index 13

Index 15

subpub provides a minimalistic, thread-safe, publish-subscribe API for single-process Python applications.

- The source code is available on GitHub.

CHAPTER

ONE

EXAMPLE

The example below demonstrates basic usage.

```
# Create an instance of the message broker
>>> from subpub import SubPub
>>> sp = SubPub()

# Subscribe to a topic (= any string or regular expression).
# The returned queue `q` is used to retrieve published data:
>>> q = sp.subscribe(r'/food/(\w+)/order-(\d+)')

# Publish any data to topic:
>>> sp.publish('/food/pizza/order-66', "beef pepperoni")
True

# Get the published data from the queue:
>>> match, data = q.get()
>>> data
'beef pepperoni'

# The queue always receives the regexp `match` object as well.
# It can be used to see how the topic matched and get groups:
>>> match.groups()
('pizza', '66')

# Get the published topic:
>>> match.string
'/food/pizza/order-66'
```

See test cases in `test_subpub.py` for more examples.

CHAPTER

TWO

KEY FEATURES

- SubPub's methods `subscribe`, `unsubscribe`, `unsubscribe_all` and `publish` are **thread-safe**.
- Subscribers use **regular experssions** to filter on topic.
- Subscribers receive published data through **queues**. (There is no built-in mechanism to register callbacks.)
- When an queue is garbage collected, `unsubscribe` is executed **automatically** (because SubPub only keeps a weak reference to the subscribers' queues).
- Publishers can post any **Python object** as message.
- Publishers can use `retain=True` to **store** a message (as in MQTT).

CHAPTER

THREE

INSTALLATION

From PyPI:

```
$ python3 -m pip install subpub
```

CHAPTER

FOUR

REFERENCE

class subpub.**SubPub**(*queue_factory=<class '_queue.SimpleQueue'>*, ***, *timeout=None*)

A threadsafe message broker with publish/subscribe API.

This class implements four methods:

1. `subscribe` - listen to topic and retrieve data throuh a queue.
2. `unsubscribe` - stop listening on topic.
3. `unsubscribe_all` - stop listening on all topics.
4. `publish` - post data to subscribers' queues.

Example:

```
>>> sp = SubPub()
>>> q = sp.subscribe('helloworld')

>>> sp.publish('helloworld', 123)
True

>>> match, data = q.get()
>>> print(data)
123

>>> print(match.string)
helloworld
```

__init__(*queue_factory=<class '_queue.SimpleQueue'>*, ***, *timeout=None*)

Initialization of SubPub instance.

Example:

```
>>> sp = SubPub()
>>> print(sp)
SubPub(queue_factory=SimpleQueue, timeout=None)
```

Parameters

- **queue** (*Callable*) – Default queue factory. If used, this parameter must be a callable which returns an instance of a queue-like object (implements get/put with timeout keyword argument). Used whenever a client subscribes unless the client provides its own queue. Defaults to `queue.SimpleQueue`.

- **timeout** (*float*) – Default timeout used for subscribe/publish when not specified. Used when putting data in client's queues.

publish(*topic: str*, *data=None*, ***, *retain=False*, *timeout=None*)

Publish data to topic.

This method loops through the clients subscribed regex-topics and searches for a match on `topic`. If there is a match, the `re.Match` and `data` objects will be wrapped in a `Msg`, which then will be put in the client's subscription queue.

Examples:

```
>>> sp = SubPub()
>>> sp.publish('helloworld', data='Hi, there!')
False
>>> sp.publish('helloworld', 'Hi, new client', retain=True)
False
```

The boolean returned, in this case `False`, indicates if it existed at least one client that received the data.

Parameters

- **topic** (*str*) – Topic string the data will be published to.
- **data** (*any Python object*) – Data to be published.
- **retain** (*bool*) – If true, the published data will be remembered. Each client that subscribes to a regex-topic matching this topic, will immediately receive the retained data when they subscribe. To stop this behavior, make a publish to the same topic with data `None` and retain `True`.
- **timeout** (*float*) – Timeout when putting published data in subscribers' queues. The behavior is client specific and depends what type of queue the client uses. If timeout is a positive number, and the default queue `queue.SimpleQueue` is used, the publish blocks at most timeout seconds and raises the `queue.Full` exception if no free slot was available within that time. If timeout is `None`, block if necessary until a free slot is available. Defaults to `self.timeout`.

Returns Returns `True` if a subscribed client queue existed and the data was successfully put in that queue. If no receiver was found, return `False`.

Return type bool

subscribe(*topic: str*, ***, *queue=None*, *timeout=None*, ***args*)

Subscribe to topic and receive published data in queue.

If topic is a string, it will be compiled to a regular expression using `topic = re.compile(topic)`.

A custom receiver `queue` can be provided. If not, a new one will be created by `self.queue_factory` with the optional `**args` arguments.

When data is published to a topic which matches the this topic, the queue will receive an instance of `Msg` which contains the regex-match object and the data.

Subscribe to plain string:

```
>>> sp = SubPub()
>>> q1 = sp.subscribe('helloworld')
```

Subscribe to regex:

```
>>> q2 = sp.subscribe(r'sensor/\d+/temperature')
```

Subscribe to regex with named groups:

```
>>> q3 = sp.subscribe(r'worker/(?P<id>\d+)/(?P<status>done|error)')
```

The `MqttTopic` class can be used to build topics using MQTT syntax for wildcards:

```
>>> t = MqttTopic('sensor/+/temperature/#')
>>> t.as_regexp()
re.compile('sensor/([^/]*)/temperature/(.*)$')
>>> q4 = sp.subscribe(t)
```

Parameters

- **topic** (str or `re.Pattern`) – Regular expression that match topics of interest. If a string, the topic will be compiled to a regular expression with `topic = re.compile(topic)`.
- **queue** (*Queue like object.*) – An instance of a queue-like object (implements get/put with timeout keyword argument). Will be used as receiver queue for published data. If used not, a new one will be created by `self.queue_factory` with the optional `**args` arguments.
- **timeout** (*float*) – Timeout when putting retained data in subscriber's queue. The behavior is client specific and depends what type of queue the client uses. If timeout is a positive number, and the default queue `queue.SimpleQueue` is used, the publish blocks at most timeout seconds and raises the `queue.Full` exception if no free slot was available within that time. If timeout is `None`, block if necessary until a free slot is available. Defaults to `self.timeout`.

Returns Queue instance which will receive published data whenever the published topic matches the subscribers regex-topic. The data is wrapped together with the `re.Match` object in an instance of `Msg`.

Return type Queue-like object.

unsubscribe(*topic: str*) → bool
Unsubscribe to topic.

Parameters **topic** (str or `re.Pattern`) – Same as for `subscribe`.

Returns Returns False if the caller wasn't actually subscribed, otherwise True.

Return type int

unsubscribe_all() → int
Unsubscribe to all clients

Returns Returns the number of topics that got unsubscribed.

Return type int

class subpub.**Msg**(*match*, *data*)
Msg is the item sent/received in subscriber's queues.

It carries the regular experssion `re.Match` object and the published `data` (data can be any Python object).

property data
Alias for field number 1

property match
Alias for field number 0

class subpub.**ExceptionAwareQueue**
Raise exception instances when received in queue.

This is useful if you want to publish an exception instance to a client and have it raised automatically when the client receives it by calling `.get()`.

get(*block=True*, *timeout=None*)
If item retrived is an Exception instance - raise it.

get_nowait()
Same as `get()` but with `block=False`.

class subpub.**MqttTopic**(*seq*)
String which represents a topic in MQTT format.

Instead of normal Python regex, the MQTT wildcards, '+' and '#', can be used instead.

An instance of MqttTopic can be used as topic argument to the SubPub methods. It will be converted to a regular expression automatically:

```
>>> MqttTopic('room/3/sensor/+/temperature/#').as_regexp()
re.compile('room/3/sensor/([^/]*)/temperature/(.*)$')
```

as_regexp(*flags=0*)
Replace MQTT wildcards and return regular expression.

class subpub.**AsyncSubPub**(*queue_factory=<class 'asyncio.queues.Queue'>*, ***, *timeout=None*)
Asynchronous implementation of SubPub.

It has the same API as SubPub but is based on the asyncio paradigm.

__init__(*queue_factory=<class 'asyncio.queues.Queue'>*, ***, *timeout=None*)
Initialization of AsyncSubPub instance.

async publish(*topic: str*, *data=None*, ***, *retain=False*, *timeout=None*)
Publish data to topic.

async subscribe(*topic: str*, ***, *queue=None*, *timeout=None*, ***args*)
Subscribe to topic and receive published data in queue.

async unsubscribe(*topic: str*) → bool
Unsubscribe to topic.

async unsubscribe_all() → int
Unsubscribe to all clients

class subpub.**AsyncExceptionAwareQueue**(*maxsize=0*, ***, *loop=None*)
Asynchronous ExceptionAwareQueue

async get()
If item retrived is an Exception instance, raise it.

get_nowait()
Same as `get()` but not blocking.

PYTHON MODULE INDEX

S

subpub, 1

INDEX

Symbols

__init__() (*subpub.AsyncSubPub method*), 12
__init__() (*subpub.SubPub method*), 9

A

as_regexp() (*subpub.MqttTopic method*), 12
AsyncExceptionAwareQueue (*class in subpub*), 12
AsyncSubPub (*class in subpub*), 12

D

data (*subpub.Msg property*), 11

E

ExceptionAwareQueue (*class in subpub*), 12

G

get() (*subpub.AsyncExceptionAwareQueue method*), 12
get() (*subpub.ExceptionAwareQueue method*), 12
get_nowait() (*subpub.AsyncExceptionAwareQueue method*), 12
get_nowait() (*subpub.ExceptionAwareQueue method*), 12

M

match (*subpub.Msg property*), 11
module
    subpub, 1
MqttTopic (*class in subpub*), 12
Msg (*class in subpub*), 11

P

publish() (*subpub.AsyncSubPub method*), 12
publish() (*subpub.SubPub method*), 10

S

subpub
    module, 1
SubPub (*class in subpub*), 9
subscribe() (*subpub.AsyncSubPub method*), 12
subscribe() (*subpub.SubPub method*), 10

U

unsubscribe() (*subpub.AsyncSubPub method*), 12
unsubscribe() (*subpub.SubPub method*), 11
unsubscribe_all() (*subpub.AsyncSubPub method*), 12
unsubscribe_all() (*subpub.SubPub method*), 11