PikaChewie

Release 1.1.0

August 19, 2015

Contents

1 pikachewie.broker – A RabbitMQ broker 3

2 pikachewie.publisher – PikaChewie base publisher class 5

3 pikachewie.consumer – PikaChewie base consumer class 7

4 pikachewie.message – A RabbitMQ message 9

5 pikachewie.agent – PikaChewie agent classes 11

6 pikachewie.helpers – Helpers for creating brokers and agents 15

7 pikachewie.data – PikaChewie data objects 17

8 pikachewie.utils – Utilities used internally by PikaChewie 19

9 Indices and tables 21

Python Module Index 23

PikaChewie is your pika co-pilot, providing RabbitMQ messaging tools with bandoliers included.

CHAPTER 1

pikachewie.broker – A RabbitMQ broker

class `pikachewie.broker.`**`Broker`**(*nodes=None*, *connect_options=None*)

A RabbitMQ broker.

nodes is a two-level `dict` containing the names and network locations of the clustered nodes that comprise the RabbitMQ broker, e.g.:

```
{
    'bunny1': {'host': 'localhost'},
    'bunny2': {'host': 'rabbit.example.com', 'port': 5678}
}
```

Valid netloc parameters are *host* and *port*. If not specified, *nodes* defaults to `{'default': {}}`, which represents a broker consisting of one node named "default" located at `localhost:5672`.

connect_options is a `dict` of keywords arguments to be passed to `pika.connection.ConnectionParameters`. If not specified, the `pika` default connection parameters will be used. For the list of supported keyword arguments and their default values, see `pika.connection.ConnectionParameters`.

Parameters

- **`nodes`** (`dict`) – two-level dict of nodenames and node netloc parameters
- **`connect_options`** (`dict`) – dict of `pika.connection.ConnectionParameters` keyword arguments

`connect`(*on_open_callback=None*, *stop_ioloop_on_close=False*, *connection_attempts=<object object>*, *on_failure_callback=None*, *cycle_delay=None*, *blocking=False*)

Return a new connection to this broker.

This method connects to each node of the broker in turn, until either a connection is successfully established, or the number of total *connection_attempts* is reached. The order in which the nodes are tried is random, and potentially different between different calls to `connect()`.

If *connection_attempts* is greater than the number of broker nodes, this method cycles through the list of nodes repeatedly as needed, pausing for *cycle_delay* seconds between each pass through the node list.

If *connection_attempts* is not specified, it defaults to the number of broker nodes, so that each node is tried at most once. If *connection_attempts* is explicitly set to *None*, there is no limit on the number of connection attempts, and `connect()` blocks until a connection is successfully made.

When *connection_attempts* is reached, the resultant behavior depends on whether an *on_failure_callback* has been specified. If it is defined, the *on_failure_callback* is called with a `BrokerConnectionError` as its sole argument, and `connect()` returns *None*. Otherwise, `connect()` raises the `BrokerConnectionError`.

Parameters

- **on_open_callback** (*callable*) – invoked once the AMQP connection is opened. Note: this parameter is ignored if *blocking* is True.
- **stop_ioloop_on_close** (`bool`) – whether to stop the IOLoop when the connection is closed (default: `True`). Note: this parameter is ignored if *blocking* is True.
- **connection_attempts** (`int` or `NoneType`) – maximum numbers of total connection attempts to make before failing
- **on_failure_callback** (*callable*) – invoked if the connection attempt fails
- **cycle_delay** (`float` or `NoneType`) – number of seconds to sleep between each cycle through the nodes list
- **blocking** (`bool`) – if True, return a `pika.BlockingConnection` (default: `False`)

Returns new connection to this broker (or `None`)

Return type `pika.adapters.tornado_connection.TornadoConnection` or `pika.BlockingConnection` or `NoneType`

Raises *`BrokerConnectionError`*

exception `pikachewie.broker.`**`BrokerConnectionError`**

Raised when a new connection cannot be opened to a `Broker`.

CHAPTER 2

pikachewie.publisher – PikaChewie base publisher class

class `pikachewie.publisher.`**`BlockingPublisher`**(*broker*)

Base class for synchronous RabbitMQ publishers.

`channel`

Return an open channel to the RabbitMQ broker.

If necessary, create and cache a new channel.

class `pikachewie.publisher.`**`JSONPublisherMixin`**

Publisher Mixin that JSON-serializes the message payload.

`publish`(*exchange*, *routing_key*, *payload*, *properties=None*)

Publish a message to RabbitMQ on the same channel the original message was received on, automatically serializing the message payload.

Parameters

- **`exchange`** (*str*) – the exchange to publish to
- **`routing_key`** (*str*) – the routing key to publish with
- **`pikachewie.data.Properties`** – the message properties
- **`dict|list`** – the message body to publish

class `pikachewie.publisher.`**`PublisherMixin`**

Mixin for publishing messages to RabbitMQ.

`publish`(*exchange*, *routing_key*, *body*, *properties=None*)

Publish a message to RabbitMQ.

properties should be an instance of pikachewie.data.Properties or None.

Parameters

- **`exchange`** (*str*) – the exchange to publish to
- **`routing_key`** (*str*) – the routing key to publish with
- **`body`** (*str|unicode*) – the message body to publish
- **`properties`** (pikachewie.data.Properties) – the message properties

CHAPTER 3

pikachewie.consumer – PikaChewie base consumer class

class `pikachewie.consumer.`**`Consumer`**

Base class for RabbitMQ consumers.

`process`(*message*)

Process the given RabbitMQ *message*.

To implement logic for processing messages in *Consumer* subclasses, override `process_message()`, not this method.

Parameters **`message`** (`pikachewie.message.Message`) – the message to process

`process_message`()

Subclasses must override this method to implement consumer logic.

CHAPTER 4

pikachewie.message – A RabbitMQ message

class `pikachewie.message.`**`Message`**(*channel*, *method*, *header*, *body*)

A RabbitMQ message.

`content_encoding`

Return the content encoding as a lowercase string.

Return type `str` or *NoneType*

`content_type`

Return the content-type as a lowercase string.

Return type `str` or *NoneType*

`created_at`

Return the message creation time as a *datetime.datetime*.

Return type `datetime.datetime`

`expires_at`

Return the message expiration time as a *datetime.datetime*.

Return type `datetime.datetime`

`is_expired`

Whether this message has expired.

Return type `bool`

`payload`

Return the decoded, deserialized contents of the message body.

Return type any

CHAPTER 5

pikachewie.agent – PikaChewie agent classes

class `pikachewie.agent.`**`ConsumerAgent`**(*consumer*, *broker*, *bindings*, *no_ack=False*, *config=None*)

A RabbitMQ client that passes Messages between a Broker and a Consumer.

`acknowledge`(*message*)

Acknowledge delivery of the given message.

Sends a Basic.Ack RPC method with the message's delivery tag to RabbitMQ.

Parameters **`message`** (`pikachewie.message.Message`) – the message to acknowledge

`add_on_cancel_callback`()

Add an on-cancel callback.

`add_on_channel_close_callback`()

Add an on-channel-close callback.

This method is invoked when the channel is opened.

`add_on_connection_close_callback`()

Add an on-connection-close callback.

This method is invoked when the connection is opened.

`bind`(*queue*, *exchange*, *routing_key*, *method_frame*)

Declare and bind the given queue to the given exchange.

The queue is bound to the exchange via the given routing_key.

`bind_queue`(*queue*, *exchange*, *routing_key*, *method_frame*)

Bind the given queue to the given exchange via the given routing_key.

`connect`()

Open a connection to RabbitMQ.

When the connection is established, the *on_connection_open* method will be invoked by pika.

`create_binding`(*queue*, *exchange*, *routing_key*)

Create a binding using the given queue, exchange, and routing_key.

This method will declare the exchange and queue.

`create_bindings`()

Create a queue binding for each of the Agent's declared bindings.

`declare_exchange`(*callback*, *exchange*)

Declare the given exchange in RabbitMQ.

declare_queue(*callback*, *queue*)
Declare the given queue in RabbitMQ.

disconnect()
Close the connection to RabbitMQ.

ensure_consuming(*queue*, *method_frame*)
Ensure that this agent is consuming from the given *queue*.

is_consuming_from(*queue*)
Whether this agent is currently consuming from the given queue.

Return type *bool*

on_channel_close(*channel*, *reply_code*, *reply_text*)
Callback invoked when the RabbitMQ channel is unexpectedly closed.

Channels are usually closed if you attempt to do something that violates the protocol, such as redeclare an exchange or queue with different paramters. In this case, we'll close the connection to shutdown the object.

Parameters **method_frame** (`pika.frame.Method`) – the Channel.Close method frame

on_channel_open(*channel*)
Callback invoked when a new channel has been opened.

Parameters **channel** (`pika.channel.Channel`) – the newly created channel

on_connection_close(*connection*, *reply_code*, *reply_text*)
Callback invoked when the RabbitMQ connection is closed unexpectedly.

on_connection_failure(*exc*)
Callback invoked when a RabbitMQ connection cannot be established.

Parameters **exc** (*exception*) – the exception raised

on_connection_open(*connection*)
Callback invoked when a connection to RabbitMQ is established.

Parameters **connection** (`pika.adapters.tornado_connection.TornadoConnection`) – the newly opened connection

on_consumer_cancel(*method_frame*)
Callback invoked when RabbitMQ sends a Basic.Cancel for a consumer.

Parameters **method_frame** (`pika.frame.Method`) – the Basic.Cancel method frame

open_channel()
Open a new channel on the current connection with RabbitMQ.

When RabbitMQ responds that the channel is open, the *on_channel_open* callback will be invoked.

process(*channel*, *method*, *header*, *body*)
Process a message received from RabbitMQ.

Parameters

- **channel** (`pika.channel.Channel`) – the channel the message was received on
- **method** (`pika.frame.Method`) – the method frame
- **header** (`pika.frame.Header`) – the header frame
- **body** (*str*) – the message body

reconnect()
Reconnect to RabbitMQ.

reject(*delivery_tag*, *requeue=True*)
Reject the message on the broker and log it.

Parameters

- **delivery_tag** (*str*) – delivery tag of the message to reject
- **requeue** (*bool*) – whether RabbitMQ should requeue the message

run()
Connect to RabbitMQ and start the connection's IOLoop.

By starting the IOLoop, this method will block, enabling the connection to operate.

start_consuming(*queue*)
Start consuming messages on the given queue.

stop()
Cleanly shutdown the connection to RabbitMQ.

CHAPTER 6

pikachewie.helpers – Helpers for creating brokers and agents

pikachewie.helpers.**broker_from_config**(*config*)

Create a *pikachewie.broker.Broker* from the given *config*.

pikachewie.helpers.**consumer_agent_from_config**(*config*, *name*, *broker='default'*, *section='rabbitmq'*)

Create a *pikachewie.agent.ConsumerAgent* from the given *config*.

The *config* dict should have a structure similar to the following example:

```
{
    'rabbitmq': {
        'brokers': {
            'default': {
                'nodes': {
                    'rabbit1': {
                        'host': 'rabbit1.example.com',
                        'port': 5672,
                    },
                    'rabbit2': {
                        'host': 'rabbit2.example.com',
                        'port': 5672,
                    },
                },
                'virtual_host': '/integration',
                'heartbeat_interval': 60,
            },
        },
        'consumers': {
            'message_logger': {
                'class': 'my.consumers.LoggingConsumer',
                'arguments': {
                    'level': 'debug',
                },
                'bindings': [
                    {
                        'exchange': 'message',
                        'queue': 'text',
                        'routing_key': 'example.text.#',
                    },
                ],
            },
        },
        'exchanges': {
            'message': {
```

```
                    'exchange_type': 'topic',
                    'durable': True,
                    'auto_delete': False,
                },
            },
            'queues': {
                'text': {
                    'durable': True,
                    'exclusive': False,
                    'arguments': {
                        'x-dead-letter-exchange': 'dead.letters',
                        'x-dead-letter-routing-key': 'omg.such.rejection',
                        'x-ha-policy': 'all',
                        'x-message-ttl': 1800000
                    },
                },
            },
        },
    }
```

`pikachewie.helpers.`**`consumer_from_config`**(*config*)

Create a `pikachewie.consumer.Consumer` from the given *config*.

CHAPTER 7

pikachewie.data – PikaChewie data objects

This module was forked from Gavin M. Roy's `rejected.data`.

ref: https://github.com/gmr/rejected/blob/master/LICENSE

class `pikachewie.data.`**`DataObject`**

Mixin that adds an object's attributes to its representation.

class `pikachewie.data.`**`Properties`**(*header=None*)

Class encapsulating attributes defined in AMQP's Basic.Properties.

CHAPTER 8

pikachewie.utils – Utilities used internally by PikaChewie

class `pikachewie.utils.`**`cached_property`**(*func*, *name=None*, *doc=None*)

A decorator that converts a function into a lazy property.

The function wrapped is called the first time to retrieve the result and then that calculated result is used the next time you access the value:

```
class Foo(object):

    @cached_property
    def foo(self):
        # calculate something important here
        return 42
```

The class has to have a *__dict__* in order for this property to work.

Cloned from *werkzeug.utils.cached_property*.

ref: https://github.com/mitsuhiko/werkzeug/blob/master/LICENSE

`pikachewie.utils.`**`delegate`**(*obj*, *attr*)

Define an instance property that is delegated to self.<obj>.<attr>.

Example:

```
>>> class MyClass(object):
...
...     def __init__(self, obj):
...         self.obj = obj
...
...     foo = delegate('obj', 'count')
...
>>> a = MyClass(obj=list())
>>> a.foo
<built-in method count of list object at 0x104ad15f0>
>>> a.obj.count
<built-in method count of list object at 0x104ad15f0>
```

CHAPTER 9

Indices and tables

- genindex
- search

Python Module Index

p

pikachewie.agent, 9
pikachewie.broker, 1
pikachewie.consumer, 5
pikachewie.data, 16
pikachewie.helpers, 13
pikachewie.message, 7
pikachewie.publisher, 4
pikachewie.utils, 17

A

acknowledge() (pikachewie.agent.ConsumerAgent method), 11
add_on_cancel_callback() (pikachewie.agent.ConsumerAgent method), 11
add_on_channel_close_callback() (pikachewie.agent.ConsumerAgent method), 11
add_on_connection_close_callback() (pikachewie.agent.ConsumerAgent method), 11

B

bind() (pikachewie.agent.ConsumerAgent method), 11
bind_queue() (pikachewie.agent.ConsumerAgent method), 11
BlockingPublisher (class in pikachewie.publisher), 5
Broker (class in pikachewie.broker), 3
broker_from_config() (in module pikachewie.helpers), 15
BrokerConnectionError, 4

C

cached_property (class in pikachewie.utils), 19
channel (pikachewie.publisher.BlockingPublisher attribute), 5
connect() (pikachewie.agent.ConsumerAgent method), 11
connect() (pikachewie.broker.Broker method), 3
Consumer (class in pikachewie.consumer), 7
consumer_agent_from_config() (in module pikachewie.helpers), 15
consumer_from_config() (in module pikachewie.helpers), 16
ConsumerAgent (class in pikachewie.agent), 11
content_encoding (pikachewie.message.Message attribute), 9
content_type (pikachewie.message.Message attribute), 9
create_binding() (pikachewie.agent.ConsumerAgent method), 11
create_bindings() (pikachewie.agent.ConsumerAgent method), 11
created_at (pikachewie.message.Message attribute), 9

D

DataObject (class in pikachewie.data), 17
declare_exchange() (pikachewie.agent.ConsumerAgent method), 11
declare_queue() (pikachewie.agent.ConsumerAgent method), 11
delegate() (in module pikachewie.utils), 19
disconnect() (pikachewie.agent.ConsumerAgent method), 12

E

ensure_consuming() (pikachewie.agent.ConsumerAgent method), 12
expires_at (pikachewie.message.Message attribute), 9

I

is_consuming_from() (pikachewie.agent.ConsumerAgent method), 12
is_expired (pikachewie.message.Message attribute), 9

J

JSONPublisherMixin (class in pikachewie.publisher), 5

M

Message (class in pikachewie.message), 9

O

on_channel_close() (pikachewie.agent.ConsumerAgent method), 12
on_channel_open() (pikachewie.agent.ConsumerAgent method), 12
on_connection_close() (pikachewie.agent.ConsumerAgent method), 12
on_connection_failure() (pikachewie.agent.ConsumerAgent method), 12

on_connection_open() (pikachewie.agent.ConsumerAgent method), 12
on_consumer_cancel() (pikachewie.agent.ConsumerAgent method), 12
open_channel() (pikachewie.agent.ConsumerAgent method), 12

P

payload (pikachewie.message.Message attribute), 9
pikachewie.agent (module), 9
pikachewie.broker (module), 1
pikachewie.consumer (module), 5
pikachewie.data (module), 16
pikachewie.helpers (module), 13
pikachewie.message (module), 7
pikachewie.publisher (module), 4
pikachewie.utils (module), 17
process() (pikachewie.agent.ConsumerAgent method), 12
process() (pikachewie.consumer.Consumer method), 7
process_message() (pikachewie.consumer.Consumer method), 7
Properties (class in pikachewie.data), 17
publish() (pikachewie.publisher.JSONPublisherMixin method), 5
publish() (pikachewie.publisher.PublisherMixin method), 5
PublisherMixin (class in pikachewie.publisher), 5

R

reconnect() (pikachewie.agent.ConsumerAgent method), 12
reject() (pikachewie.agent.ConsumerAgent method), 13
run() (pikachewie.agent.ConsumerAgent method), 13

S

start_consuming() (pikachewie.agent.ConsumerAgent method), 13
stop() (pikachewie.agent.ConsumerAgent method), 13