Vireo Documentation

Release 1.0a1

Juti Noppornpitak

Jun 15, 2018

Contents

A library and framework for event-driven application development

Warning: This library is currently under active development. Until version 1.0, the API signatures are subject to change. While the code should be stable (not easily throwing exceptions) even before reaching version 1.0, version pinning is highly recommended.

CHAPTER 1

API Reference

1.1 Core and Observer

class vireo.observer.**Observer**(*driver*)

Event Observer

id

Observer Identifier

join(*running_mode=2*)

Wait for all handlers to stop.

There are two mode: synchronous (`vireo.observer.SYNC_START`) and asynchronous (`vireo.observer.ASYNC_START`) joins.

```
app.join(ASYNC_START)
```

on(*event_name*, *callback*, *resumable=False*, *simple_handling=True*, *options=None*, *delay_per_message=0*, *max_retries=None*, *immediate_retry_limit=None*, *max_retry_timeout=None*)

Listen to an event with a callback function.

Parameters

- **event_name** (*str*) – the name of the event
- **callback** (*callable*) – the callback callable
- **resumable** (*bool*) – the flag to indicate whether the event consumption can be resumed (as the data stream will never be deleted).
- **simple_handling** (*bool*) – the flag to instruct the code to return the content of the message, instead of returning the whole `vireo.model.Message` object.
- **options** (*dict*) – the extra options to the method `observe` of the driver
- **delay_per_message** (*float*) – the delay per message (any negative numbers are regarded as zero, zero or any equivalent value is regarded as "no delay")

- **max_retries** (*int*) – maximum allowed retry count
- **immediate_retry_limit** (*int*) – allowed immediate retry count
- **max_retry_timeout** (*int*) – maximum retry timeout

The callback is a callable object, e.g., function, class method and lambda object, which takes only one parameter which is a JSON-decoded object.

For example,

```
def on_foo(self, message):
    print('on_foo:', message)

app.on('foo', on_foo)
app.on('foo.lambda', lambda x: print('foo_lambda:', x))
```

Here is an example for `error_handler`.

```
def error_handler(consumer, exception):
    ...
```

on_broadcast(*event_name*, *callback*, *simple_handling=True*, *options=None*, *delay_per_message=0*, *max_retries=None*, *immediate_retry_limit=None*, *max_retry_timeout=None*)

Listen to an distributed event with a callback function.

Parameters

- **event_name** (*str*) – the name of the event
- **callback** (*callable*) – the callback callable
- **simple_handling** (*bool*) – the flag to instruct the code to return the content of the message, instead of returning the whole `vireo.model.Message` object.
- **options** (*dict*) – the extra options to the method `observe` of the driver
- **delay_per_message** (*float*) – the delay per message (any negative numbers are regarded as zero, zero or any equivalent value is regarded as "no delay")
- **max_retries** (*int*) – maximum allowed retry count
- **immediate_retry_limit** (*int*) – allowed immediate retry count
- **max_retry_timeout** (*int*) – maximum retry timeout

The callback is a callable object, e.g., function, class method and lambda object, which takes only one parameter which is a JSON-decoded object.

For example,

```
def on_foo(self, message):
    print('on_foo:', message)

app.on('foo', on_foo)
app.on('foo.lambda', lambda x: print('foo_lambda:', x))
```

Here is an example for `error_handler`.

```
def error_handler(consumer, exception):
    ...
```

stop()
Send the signal to all handlers to stop observation.

> **Warning:** This method does not block the caller thread while waiting all handlers to stop.

```
app.stop()
```

exception vireo.observer.**UnknownRunningModeError**
Error for unknown running mode

1.2 Exceptions

exception vireo.exception.**InvalidURLError**
Invalid URL Error

exception vireo.exception.**NoConnectionError**
No connection error

exception vireo.exception.**ObservationError**
Observation error

1.3 RabbitMQ Driver

class vireo.drivers.rabbitmq.driver.**Driver**(*url*, *consumer_classes=None*, *unlimited_retries=False*, *on_connect=None*, *on_disconnect=None*, *on_error=None*, *default_publishing_options: dict = None*, *default_broadcasting_options: dict = None*, *default_consuming_shared_queue_options: dict = None*, *default_consuming_distributed_queue_options: dict = None*, *auto_acknowledge=False*, *send_sigterm_on_disconnect=True*)
Driver for RabbitMQ

Parameters

- **url** – the URL to the server (`str` for a single connection or `list` for rotation)
- **consumer_classes** (*list*) – the list of *consumer.Consumer*-based classes
- **unlimited_retries** (*bool*) – the flag to disable limited retry count.
- **on_connect** (*callable*) – a callback function when the message consumption begins.
- **on_disconnect** (*callable*) – a callback function when the message consumption is interrupted due to unexpected disconnection.
- **on_error** (*callable*) – a callback function when the message consumption is interrupted due to exception raised from the main callback function.
- **default_publishing_options** (*dict*) – the default options for publishing (normal)

- **default_broadcasting_options** (*dict*) – the default options for publishing (broadcast)
- **default_consuming_shared_queue_options** (*dict*) – the default options for consuming share queue
- **default_consuming_distributed_queue_options** (*dict*) – the default options for consuming distributed queue

`default_publishing_options` and `default_broadcasting_options` only take `exchange` to allow overriding the default exchange.

`default_consuming_shared_queue_options` and `default_consuming_distributed_queue_options` will have the data structure like this:

```
{
    'exchange': {
        'name': str, # It is "exchange" in pika's exchange_declare.
        'type': str, # It is "exchange_type" in pika's exchange_declare.
    }
}
```

Here is an example for `on_connect`.

```
def on_connect(consumer = None, controller_id = None, route = None, queue_name =␣
↪None, summary = None):
    ...
```

Here is an example for `on_disconnect`.

```
def on_disconnect(consumer = None, controller_id = None, route = None, queue_name␣
↪= None, summary = None):
    ...
```

Here is an example for `on_error`.

```
def on_error(exception, consumer = None, controller_id = None, route = None,␣
↪queue_name = None, summary = None):
    ...
```

Where: * `exception` is the (raised) exception object. * `consumer` is the associate consumer object (optional). * `controller_id` is the associate ID (optional). * `route` is the affected route (optional). * `queue_name` is the affected queue name (optional). * `summary` is the summary of the event (optional).

broadcast(*route*, *message*, *options=None*, *allowed_retry_count=5*)

Broadcast a message to a particular route.

Parameters

- **route** (*str*) – the route
- **message** (*str*) – the message
- **options** (*dict*) – additional options for basic_publish

join()

Synchronously join all consumers.

publish(*route*, *message*, *options=None*, *allowed_retry_count=5*)

Synchronously publish a message

Parameters

- **route** (*str*) – the route
- **message** (*str*) – the message
- **options** (*dict*) – additional options for basic_publish
- **allowed_retry_count** (*bool*) – the flag to allow auto-retry on connection failure

setup_async_cleanup()
Prepare to cleanly join all consumers asynchronously.

stop_consuming()
Send the signal to stop consumption.

class vireo.drivers.rabbitmq.consumer.**Consumer**(*url*, *route*, *callback*, *shared_stream*, *resumable*, *distributed*, *queue_options*, *simple_handling*, *unlimited_retries=False*, *on_connect=None*, *on_disconnect=None*, *on_error=None*, *controller_id=None*, *exchange_options=None*, *auto_acknowledge=False*, *send_sigterm_on_disconnect=True*, *delay_per_message=0*, *max_retries=None*, *immediate_retry_limit=None*, *max_retry_timeout=None*)

Message consumer

This is used to handle messages on one particular route/queue.

Parameters

- **url** (*str*) – the URL to the server
- **route** (*str*) – the route to observe
- **callback** (*callable*) – the callback function / callable object
- **shared_stream** (*list*) – the internal message queue for thread synchronization
- **resumable** (*bool*) – the flag to indicate whether the consumption is resumable
- **resumable** – the flag to indicate whether the messages are distributed evenly across all consumers on the same route
- **queue_options** (*dict*) – additional queue options
- **exchange_options** (*dict*) – additional exchange options
- **unlimited_retries** (*bool*) – the flag to disable limited retry count.
- **on_connect** (*callable*) – a callback function when the message consumption begins.
- **on_disconnect** (*callable*) – a callback function when the message consumption is interrupted due to unexpected disconnection.
- **on_error** (*callable*) – a callback function when the message consumption is interrupted due to exception raised from the main callback function.
- **controller_id** (*str*) – the associated controller ID
- **exchange_options** – the additional options for exchange

- **auto_acknowledge** (*bool*) – the flag to determine whether the consumer should auto-acknowledge any delivery (default: `False`)
- **send_sigterm_on_disconnect** (*bool*) – the flag to force the consumer to terminate the process cleanly on disconnection (default: `True`)
- **delay_per_message** (*float*) – the delay per message (any negative numbers are regarded as zero, zero or any equivalent value is regarded as "no delay")
- **max_retries** (*int*) – the maximum total retries the consumer can have
- **immediate_retry_limit** (*int*) – the maximum immediate retries the consumer can have before it uses the exponential delay

Here is an example for `on_connect`.

```
def on_connect(consumer = None, controller_id = None, route = None, queue_name =␣
↪None, summary = None):
    ...
```

Here is an example for `on_disconnect`.

```
def on_disconnect(consumer = None, controller_id = None, route = None, queue_name␣
↪= None, summary = None):
    ...
```

Here is an example for `on_error`.

```
def on_error(exception, consumer = None, controller_id = None, route = None,␣
↪queue_name = None, summary = None):
    ...
```

static can_handle_route(*routing_key*)
Check if the consumer can handle the given routing key.

Note: the default implementation will handle all routes.

Parameters **routing_key** (*str*) – the routing key

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

stop()
Stop consumption

class vireo.drivers.rabbitmq.exception.**NoConnectionError**
No connection error

class vireo.drivers.rabbitmq.exception.**SubscriptionNotAllowedError**
Subscription not allowed

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

v

Index