Брокер сообщений RabbitMQ

MQ = message queue

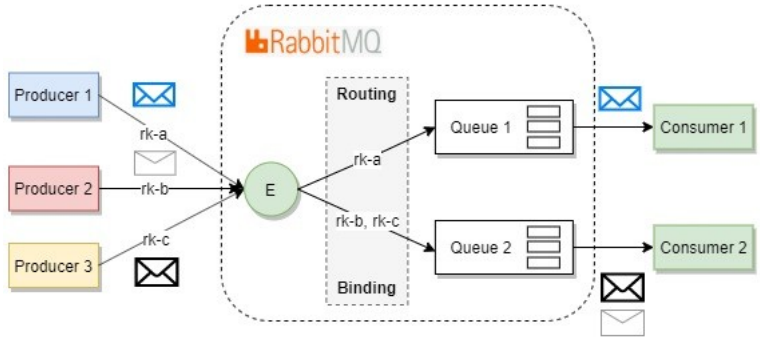
RabbitMQ — брокер сообщений с открытым кодом, реализующий протоколы AMQP, MQTT, WebSocket

Сайт проекта: www.rabbitmq.com
https://www.rabbitmq.com/getstarted.html

Интерфейс с Python: библиотека pika

Основные понятия:

- producer — отправитель сообщения;
- message — сообщение;
- exchange — точка обмена (определяет маршрутизацию сообщений);
- queue — очередь;
- consumer — исполнитель, принимающий сообщения из очередей



https://3-info.ru/post/10051

Основные функции

Создание очереди: queue_declare()
Параметры:

- название очереди;
- passive: true/false (passive = true - если обращаемся к очереди, не меняя её состояния, напр. проверка существования);
- durable: true/false (durable=true – есть периодическое сохранение на диск);
- exclusive: true/false (exclusive = true - очередь доступна только в одном соединении)
- auto_delete: true/false (delete=true - очередь автоматически очищается, когда соединение закрывается).

Очередь привязывается (bind) к некоторой точке обмена (exchange)

Создание точки обмена: exchange_declare()

- тип (fanout / direct / topic / headers);
- passive, durable, auto_delete – аналогично очереди

Задача точки обмена — определить, в какую из очередей должно попасть сообщение.

Виды точек обмена:

1. fanout - сообщение попадает во все доступные очереди;
2. direct — сообщение попадает в очередь с полностью совпадающим ключом;
3. topic — сообщение попадает в очередь с ключом, удовлетворяющим указанной маске;
4. headers — сообщение попадает в очередь с ключом, совпадающим с заголовком сообщения

Привязка очереди к точке обмена: queue_bind()
Указываем точку обмена и очередь: queue_bind(exchange, queue)

Параметры:

- exchange – точка обмена, через которую посылаем сообщение

Отправка сообщения в очередь: basic_publish()
basic_publish(exchange, routing_key, body, properties)

Параметры:

- exchange – точка обмена, через которую посылаем сообщение
- routing_key – ключ, определяющий очередь
- body – тело сообщения (строка, JSON,..)
- properties - свойства

Приём сообщений в цикле: basic_consume()

Параметры:

- queue – очередь, из которой принимаем
- on_message_callback – имя функции, которая будет вызываться при приёме очередного сообщения

Начало приёма: start_consuming()

Без параметров

Минимальный пример

https://www.rabbitmq.com/tutorials/tutorial-one-python.html

send.py

```python
import pika

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='hello')

channel.basic_publish(exchange='', routing_key='hello', body='Hello World!')
print(" [x] Sent 'Hello World!'")
connection.close()
```

receive.py

```python
import pika, sys, os

def main():
    connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
    channel = connection.channel()

    channel.queue_declare(queue='hello')

    def callback(ch, method, properties, body):
        print(" [x] Received %r" % body.decode())

    channel.basic_consume(queue='hello', on_message_callback=callback, auto_ack=True)

    print(' [*] Waiting for messages. To exit press CTRL+C')
    channel.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)
```

Дополнительные возможности

Выход из consume: basic_cancel()

Посылаем сообщение серверу RabbitMQ о том, что данный потребитель завершает приём сообщений. Пример: получено сообщение-маркер окончания связи (текст quit в body), нужно прервать цикл потребления сообщений.

Параметры:

- consumer_tag — идентификатор потребителя (он указывается в basic_consume)
- callback — обычно None

```
def recv_callback(ch, method, properties, body):
    print(" [x] Received %r" % body)

    if(body==’quit’)
            ch.basic_cancel()
```

Блокирующий приём одного сообщения: basic_get(queue_from)

Параметры

- queue_from – очередь, из которой принимаем

Отсылка подтверждения: basic_ack()

Параметры

- delivery_tag – действует внутри определённого канала, т. е. клиент не может получить сообщение в одном канале и подтвердить в другом
- multiple - если True, одним вызовом можно подтвердить все сообщения с тегом доставки меньше либо равным delivery_tag; если False можно подтвердить только одно сообщение с заданным тегом доставки; если multiple=1, delivery_tag=0 - подтверждаем все ожидающие сообщения

Свойства очередей, опции

- возможность сохранения (durable) – очередь время от времени сохраняется на диск
- время жизни (TTL) – сообщения удаляются из очереди при истечении TTL («теряют актуальность»)
- механизм подтверждений (ack) – если он включён, то сообщение удаляем из очереди только после получения подтверждения (по сути, это таймаут)
- задержка перед повторной обработкой (dead letter queue — очередь недоставленных сообщений)

«A dead-letter queue (DLQ), sometimes referred to as an undelivered-message queue, is a holding queue for messages that cannot be delivered to their destination queues, for example because the queue does not exist, or because it is full. Dead-letter queues are also used at the sending end of a channel, for data-conversion errors.. Every queue manager in a network typically has a local queue to be used as a dead-letter queue so that messages that cannot be delivered to their correct destination can be stored for later retrieval.»