Надежность

Подтверждение сообщений (acknowledgment)

Выполнение задачи может занять некоторое время. Вы можете задаться вопросом, что произойдет, если один из *consumer* начнет длинную задачу и упадет или завершится, выполнив ее лишь частично.

С таким кодом, как в прошлой карточке, RabbitMQ немедленно удаляет сообщение из памяти, как только доставляет его клиенту. В этом случае, если воркер неожиданно завершится, мы потеряем сообщение, которое он обрабатывал. Но мы, разумеется, не хотим терять задачи. Если *consumer* умирает, мы хотим, чтобы задание было передано *другому consumer*.

Чтобы сообщение никогда не потерялось, RabbitMQ поддерживает подтверждение сообщений. *Подтверждение (acknowledgment)* отправляется обратно от потребителя, чтобы сообщить RabbitMQ, что конкретное сообщение было получено, обработано, и RabbitMQ может его удалить.

Если потребитель умирает (его канал закрыт, RabbitMQ-соединение закрыто или TCP-соединение потеряно) без отправки подтверждения, RabbitMQ поймет, что сообщение не было обработано полностью, и повторно поставит его в очередь. Если в то же время есть *другие consumer*, он быстро перенаправит его другому потребителю. Таким образом, вы можете быть уверены, что ни одно сообщение не потеряется, даже если consumer иногда умирает.

Таймаутов у сообщений нет, и RabbitMQ будет ждать бесконечно долго, пока сообщение не подтвердится, либо не произойдет обрыв/закрытие соединения. В этом случае RabbitMQ повторно доставит сообщение другому consumer. Это нормально, даже если обработка сообщения занимает очень много времени.

Подтверждения сообщений включены по умолчанию. В предыдущем примере мы явно отключили их с помощью флага no_ack = True.

Пример: как работает подтверждение.

connectors/rabbit/ack/

receiver.py

```
import asyncio
import os

from aio_pika import connect, IncomingMessage


async def create_connection():
    return await connect(url=os.getenv("RABBITMQ_URL"))


async def on_message(_: IncomingMessage):
    print("Before sleep!")
    await asyncio.sleep(5)
    print("After sleep!")


async def receiver():
    connection = await create_connection()
    channel = await connection.channel()
    queue = await channel.declare_queue("ack")
    await queue.consume(on_message, no_ack=False)


if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.create_task(receiver())

    print(" [*] Waiting for messages.")
    loop.run_forever()
```

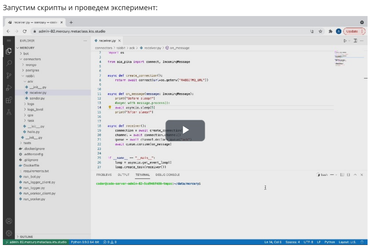
sender.py

```
import asyncio
import os

from aio_pika import connect, Message


async def create_connection():
    return await connect(url=os.getenv("RABBITMQ_URL"))


async def sender():
    connection = await create_connection()
    channel = await connection.channel()
    await channel.declare_queue("ack")
    await channel.default_exchange.publish(
        Message(f"Hello World!".encode()),
        routing_key="ack",
    )
    print(" [x] Sent 'Hello World!'")
    await connection.close()


if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(sender())
```

Код от предыдущего примера отличается только no_ack=False. По умолчанию no_ack равен False, поэтому этот параметр можно просто опускать. Но тут оставлено — для наглядности.

```
await queue.consume(on_message, no_ack=False)
```

Запустим скрипты и проведем эксперимент:



Для подтверждения сообщения нужно вызвать контекстный менеджер process:

```
async def on_message(message: IncomingMessage):
    print("Before sleep!")
    async with message.process():
        await asyncio.sleep(5)
    print("After sleep!")
```

Или вызвать ack:

```
async def on_message(message: IncomingMessage):
    print("Before sleep!")
    await asyncio.sleep(5)
    message.ack()
    print("After sleep!")
```

Нужно отметить, если произойдет исключение перед *message.ack()*, сообщение никогда не будет подтверждено и никогда не попадет другому consumer. Поэтому для этого следует использовать контекстный менеджер message.process()

Запустить пример нужно в разных терминалах в mercury:

```
python3 connectors/rabbit/ack/receiver.py
python3 connectors/rabbit/ack/sender.py
```

Персистентность

По умолчанию RabbitMQ хранит сообщения в оперативной памяти. Поэтому если сервер RabbitMQ упадет, сообщения будут утеряны.

Во избежание потерь в Rabbit есть механизм хранения сообщений на диске. Но при этом Rabbit по умолчанию нетранзакционный, он не гарантирует сохранность всех сообщений при внезапном выключении сервера. Часть сообщений, которые уже хранились на диске, сохранятся, но часть *недавно принятых сообщений* могут быть потеряны. Поэтому этот механизм плохо подходит для обеспечения надежности.

Подробно проблему надежности мы рассматривать не будем, но вот несколько способов, чтобы никогда не терять сообщения:

- получать подтверждения в стороны получателя — publisher-confirms
- создавать отказоустойчивый кластер RabbitMQ
- использовать транзакции в Rabbit (не рекомендуется, так как снижает производительность)
- использовать другие транзакционные брокеры сообщений