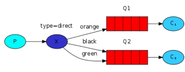
Routing

Наша система логирования из предыдущего примера транслирует все сообщения всем потребителям. Предположим, что мы хотим расширить ее, чтобы позволить фильтровать сообщения на основе их уровня. Критические сообщения будем складывать на диск, а все сообщения отображать в stdout.

В прошлом примере мы использовали тип FANOUT, который не дает нам особой гибкости — рассылаем всем связанным очередям сообщения. Вместо этого мы будем использовать exchange типа DIRECT.



В первом примере с default_exchange мы использовали параметр **routing_key** при отправке сообщения:

```
await channel.default_exchange.publish(Message(f"{i} - Hello World!".encode()), routing_key="hello")
```

Теперь мы будем использовать его более осознанно. Рассмотрим картинку выше. На ней отображен direct exchange, который привязан к двум очередям. Первая очередь *имеет routing_key*, равный *orange*, вторая *black* и *green*. То есть сообщения, которые отправляются с routing_key, равным orange, попадут в очередь *Q1*, а сообщения с *routing_key black или green* попадут в очередь *Q2*. Сообщения с *любыми другими значениями* routing_key *будут удалены*.

Чтобы привязать очередь к конкретному exchange и routing_key, нужно вызвать:

```
await queue1.bind(exchange, routing_key="orange")
await queue2.bind(exchange, routing_key="black")
await queue2.bind(exchange, routing_key="green")
```

Также можно привязать 2 разные очереди к одинаковому routing_key. Таким образом можно повторить поведения режима FANOUT, когда мы транслируем все пришедшие сообщения во все привязанные очереди.

Пример: connectors/rabbit/logs_level/

sender.py

```
import asyncio
import os
import sys

from aio_pika import connect, Message, ExchangeType


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


async def sender(routing_key):
    connection = await create_connection()
    channel = await connection.channel()
    logs_exchange = await channel.declare_exchange(
        "logs_level", ExchangeType.DIRECT
    )
    await logs_exchange.publish(
        Message(f"Hello World!".encode()),
        routing_key=routing_key,
    )
    print(f" [x] Sent 'Hello World!'")
    await connection.close()


if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    rout = sys.argv[1] if len(sys.argv) > 1 else "info"
    loop.run_until_complete(sender(rout))
```

Обратите внимание, что здесь мы объявляем exchange с типом DIRECT.

to_file.py

```
import asyncio
import os

from aio_pika import connect, IncomingMessage, ExchangeType


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


async def on_message(message: IncomingMessage):
    async with message.process():
        with open('logs.txt', 'a') as fd:
            fd.write(str(message))
            fd.write('\n')


async def receiver():
    connection = await create_connection()
    channel = await connection.channel()
    logs_exchange = await channel.declare_exchange(
        "logs_level", ExchangeType.DIRECT
    )
    queue = await channel.declare_queue(exclusive=True)
    await queue.bind(logs_exchange, routing_key="critical")
    await queue.consume(on_message)


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

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

Обратите внимание, что здесь мы соединяем очередь с ключем critical:

```
await queue.bind(logs_exchange, routing_key="critical")
```

to_stdout.py

```
import asyncio
import os

from aio_pika import connect, IncomingMessage, ExchangeType


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


async def on_message(message: IncomingMessage):
    async with message.process():
        print(message)


async def receiver():
    connection = await create_connection()
    channel = await connection.channel()
    logs_exchange = await channel.declare_exchange(
        "logs_level", ExchangeType.DIRECT
    )
    queue = await channel.declare_queue(exclusive=True)
    await queue.bind(logs_exchange, routing_key="info")
    await queue.bind(logs_exchange, routing_key="critical")
    await queue.consume(on_message)


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

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

Обратите внимание, что здесь мы соединяем очередь с ключем info и critical:

```
await queue.bind(logs_exchange, routing_key="info")
await queue.bind(logs_exchange, routing_key="critical")
```

Запустить пример нужно в разных терминалах в mercury:

```
python3 connectors/rabbit/logs_level/to_file.py
python3 connectors/rabbit/logs_level/to_stdout.py
python3 connectors/rabbit/logs_level/sender.py info
python3 connectors/rabbit/logs_level/sender.py critical
python3 connectors/rabbit/logs_level/sender.py something
```

Запустите to_file.py и sender.py. Далее запустите 3 раза sender.py:

```
# попадет только в stdout
python3 sender.py info
# попадет только в stdout и в файл logs.txt
python3 sender.py critical
# никуда не попадет
python3 sender.py something
```

В RabbitMQ есть более продвинутая система роутинга. Она основана на Topic (ExchangeType.Topic).

В примере выше мы присоединяли очереди к exchange на основе routing_key. Если routing_key при отправке сообщения *полностью* совпадал с routing_key, который был объявлен при соединении exchange и очереди, то сообщение попадало в указанную очередь. В RabbitMQ есть возможность привязывать очереди к exchange не по полному совпадению, а по шаблону:

- info.*
- *.info

Роутинг с помощью топиков мы рассматривать не будем, но о нем можно прочитать тут.