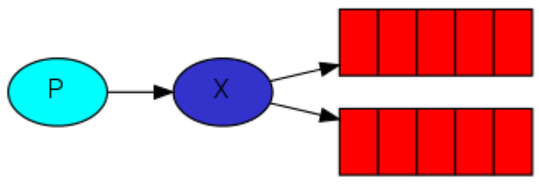
Python RabbitMQ发布/订阅

前面搭建的工作队列，每个任务只分发给一个工作者(worker)；在例中我们要做的跟之前完全不一样，分发一个消息给多个消费者(consumers)，这种模式被称为"发布/订阅"．为了描述这种模式，我们将会构建一个简单的日志系统，它包括一个负责发送日志消息的程序和一个负责获取消息并输出内容的程序，在我们的这个日志系统中，所有正在运行的接收方程序都会接受消息．我们用其中一个接收者(receiver)把日志写入硬盘中，另外一个接受者(receiver)把日志输出到屏幕上.最终日志消息被广播给所有的接受者(receivers)．

概念：
交换机(Exchanges)前面的例子中，我们发送消息到队列并从中取出消息，现在我们先介绍一下RabbitMQ中完整的消息模型了．发布者(producer)只需要把消息发送给一个交换机(exchange)，交换机一边从发布者方接收消息，一边把消息推送到队列．交换机必须知道如何处理它接收到的消息，是应该推送到指定的队列还是是多个队列还是是直接忽略消息．这些规则是通过交换机类型(exchange type)来定义的．



交换机类型：
直连交换机(director)、主题交换机(topic)、头交换机(headers)、扇型交换机(fanout)．本例中我们以扇型交换机(fanout)为例．先创建一个fanout类型的交换机，命名为logs：

```
channel.exchange_declare(exchange='logs', type='fanout')
```

扇型交换机(fanout)很简单，它把消息发送给它所知道的所有队列，这正是我们的日志系统所需要的．通过rabbitmqctl查看交换机信息：

```
[root@harbor-a sbin]# ./rabbitmqctl -n rabbit@localhost list_exchanges;
Listing exchanges
amq.rabbitmq.log        topic
logs    fanout
        direct
amq.rabbitmq.trace      topic
amq.topic       topic
amq.headers     headers
amq.fanout      fanout
amq.direct      direct
amq.match       headers
```

这个列表中有一些叫做amq.*的交换器，这些都是默认创建的先暂时不用理会．

匿名交换机：
前面的实验中我们没用到交换机，但仍然能够发送消息到队列中，因为我们使用了命名为空字符串("")默认的交换机．之前发布消息的方式如下：

```
channel.basic_publish(exchange='', routing_key='hello', body=message)
```

exchange参数就是交换机的名称，空字符串代表默认或者匿名交换机，消息将会根据指定的routing_key分发到指定的队列．现在我们创建一个命名交换机交换机代码如下：

```
channel.basic_publish(exchange='logs', routing_key='', body=message)
```

临时队列：

前面我们使用了队列(hello和task_queue)，在代码中将工作者(workers)指向正确的队列，在发布者(producers)和消费者(consumers)之间共享同队列时，需要为队列指定名字．但是前面的例子并不适用于我们现在的模拟日志系统代码．我们打算接收所有的日志消息，而不仅仅是一小部分，我们关心的是最新的消息而不是旧的，为了解决这个问题，我们需要做两件事情．

1．当我们连接上RabbitMQ的时候，我们需要一个全新的、空的队列．我们可以手动创建一个随机的队列名或者让服务器为我们选择一个随机的队列名(推荐)　此时只需要在调用queue_declare方法的时候，不提供queue参数就可以了．示例代码如下：

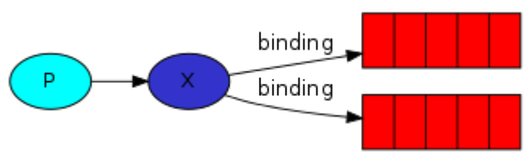
```
result = channel.queue_declare()
```

此时我们可以通过result.method.queue获得已经生成的随机队列名．它可能是这样子的：amq.gen-U0srCoW8TsaXjNh73pnVAw==

2．当与消费者(consumer)断开连接的时候，这个队列应当被立即删除．exclusive标识符即可达到此目的，示例代码：

```
result = channel.queue_declare(exclusive=True)
```

绑定(Bindings)：

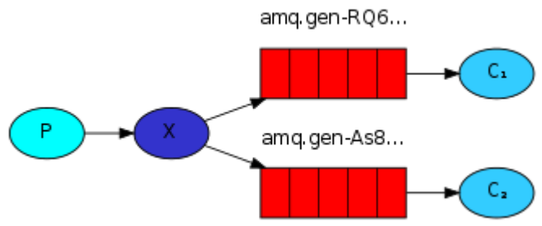


我们已经创建了一个扇型交换机（fanout）和一个队列，现在我们需要告诉交换机如何发送消息给我们的队列，交换器和队列之间的联系我们称之为绑定．

```
channel.queue_bind(exchange='logs', queue=result.method.queue)
```

现在logs交换机将会把消息添加到我们的队列中，绑定(binding)列表可以通过如下命令查看：

```
[root@harbor-a sbin]# ./rabbitmqctl -n rabbit@localhost list_bindings
Listing bindings
        exchange        amq.gen-1dsaufegRz1gp_BEQjZbSw  queue   amq.gen-1dsaufegRz1gp_BEQjZbS[
]
        exchange        amq.gen-nBW4mbGpGOmsFc5M13B6Gw  queue   amq.gen-nBW4mbGpGOmsFc5M13B6G[
]
        exchange        hello   queue   hello   []
        exchange        task_queue      queue   task_queue      []
logs    exchange        amq.gen-1dsaufegRz1gp_BEQjZbSw  queue   amq.gen-1dsaufegRz1gp_BEQjZbS[
]
logs    exchange        amq.gen-nBW4mbGpGOmsFc5M13B6Gw  queue   amq.gen-nBW4mbGpGOmsFc5M13B6G[
]
```



发布日志消息的程序看起来和之前的没有太大区别，最重要的改变就是我们把消息发送给logs交换机而不是匿名交换机。并且在消费者上创建了随机命名的队列并与扇形交换机相关联。

示例代码(生产者)：

In [ ]:
```
import pika
import sys
import random
credentials = pika.PlainCredentials('rabbit', 'rabbit')
connection = pika.BlockingConnection(pika.ConnectionParameters('172.16.70.251', 5672,'/',credentials))
message = ' '.join(sys.argv[1:])
channel = connection.channel()
channel.exchange_declare(exchange='logs', exchange_type='fanout')
channel.basic_publish(exchange='logs',
                      routing_key='',
                      body=message,
                     )
print(" [x] Sent 'Hello World!'")
connection.close()
```

示例代码(消费者)：

In [ ]:
```
import pika
import time
credentials = pika.PlainCredentials('rabbit', 'rabbit')
connection = pika.BlockingConnection(pika.ConnectionParameters('172.16.70.251', 5672,'/',credentials))
channel = connection.channel()
channel.exchange_declare(exchange='logs', exchange_type='fanout')
result = channel.queue_declare(queue='', exclusive=True)
queue_name = result.method.queue
channel.queue_bind(exchange='logs', queue=queue_name)
print(' [*] Waiting for messages. To exit press CTRL+C')

def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)
    time.sleep(body.decode(encoding="utf-8").count('.'))
    print('done')

channel.basic_consume(on_message_callback=callback, queue=queue_name, auto_ack=True)
channel.start_consuming()
```