RabbitMQ

一、RabbitMQ 介绍

1.python的Queue和RabbitMQ

python消息队列

- 线程 queue（同一进程下线程之间进行交互）
- 进程 Queue（父子进程进行交互 或者 同属于同一进程下的多个子进程进行交互）

两个完全独立的python程序

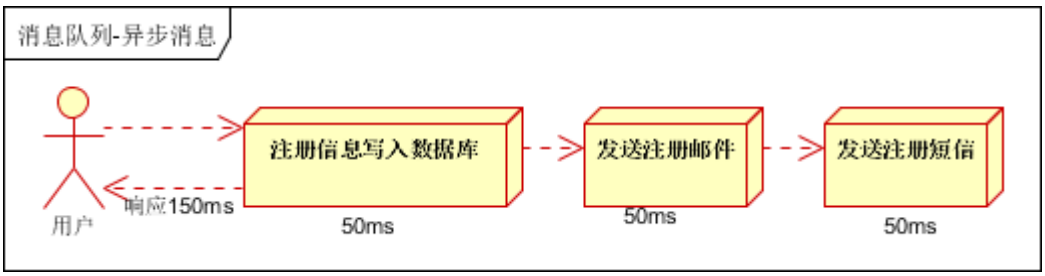
- 是不能用上面两个queue进行交互的，或者和其他语言交互有哪些实现方式呢。
- 【Disk、Socket、其他中间件】这里中间件不仅可以支持两个程序之间交互，可以支持多个程序，可以维护好多个程序的队列
-
- 像这种公共的中间件有好多成熟的产品： RabbitMQ 、ZeroMQ 、ActiveMQ等

2.消息队列应用场景

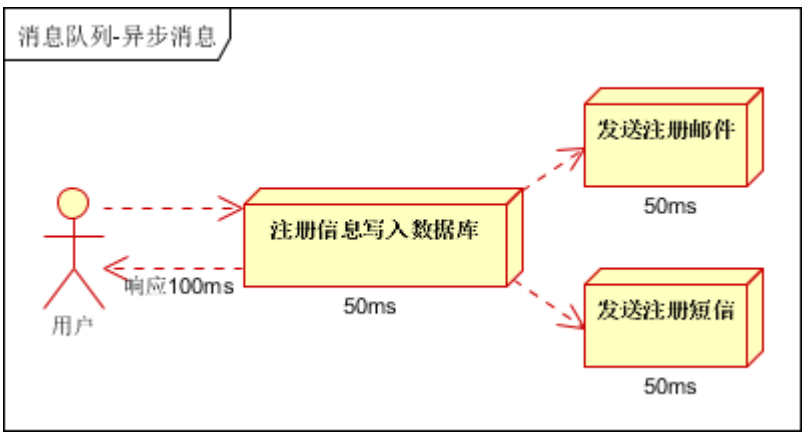
2.1 异步处理

场景说明：用户注册后，需要发注册邮件和注册短信。传统的做法有两种 1.串行的方式；2.并行方式

- **串行方式**：将注册信息写入数据库成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后，返回给客户端



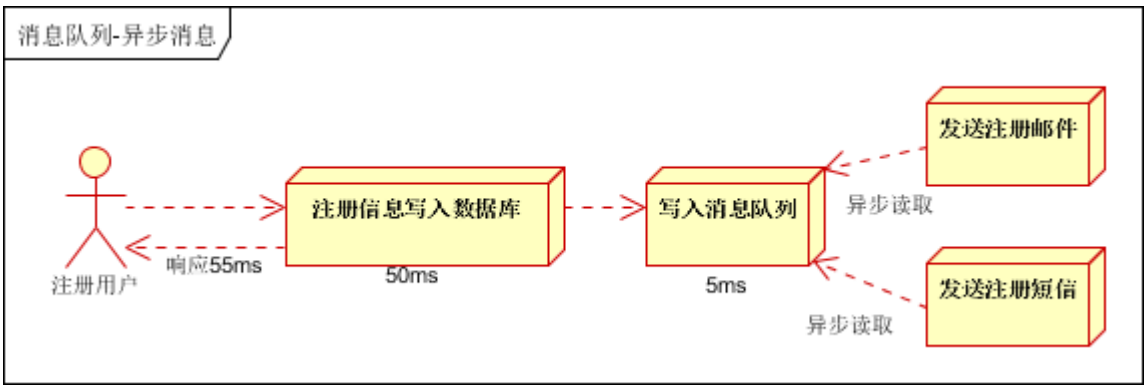
- **并行方式**：将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。与串行的差别是，并行的方式可以提高处理的时间



假设三个业务节点每个使用50毫秒钟，不考虑网络等其他开销，则串行方式的时间是150毫秒，并行的时间可能是100毫秒。 因为CPU在单位时间内处理的请求数是一定的，假设CPU1秒内吞吐量是100次。则串行方式1秒内CPU可处理的请求量是7次（1000/150）。并行方式处理的请求量是10次（1000/100） 小结：如以上案例描述，传统的方式系统的性能（并发量，吞吐量，响应时间）会有瓶颈。如何解决这个问题呢?

- **引入消息队列**

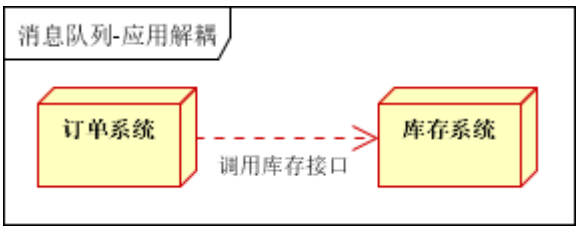
  将不是必须的业务逻辑，异步处理。改造后的架构如下



  按照以上约定，用户的响应时间相当于是注册信息写入数据库的时间，也就是50毫秒。注册邮件，发送短信写入消息队列后，直接返回，因此写入消息队列的速度很快，基本可以忽略，因此用户的响应时间可能是50毫秒。因此架构改变后，系统的吞吐量提高到每秒20 QPS。比串行提高了3倍，比并行提高了两倍。

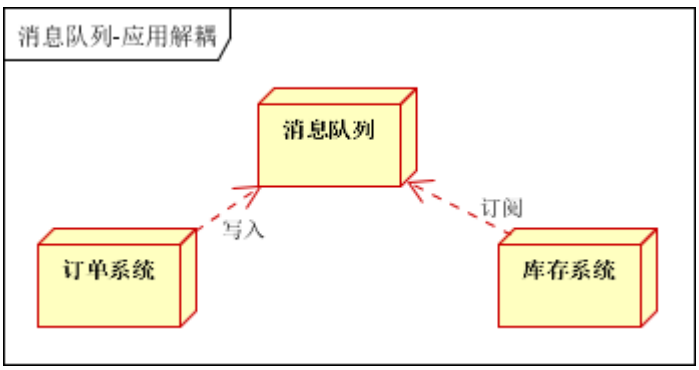
2.2 应用解耦

场景说明：用户下单后，订单系统需要通知库存系统。传统的做法是，订单系统调用库存系统的接口。如下图：



传统模式的缺点：假如库存系统无法访问，则订单减库存将失败，从而导致订单失败，订单系统与库存系统耦合

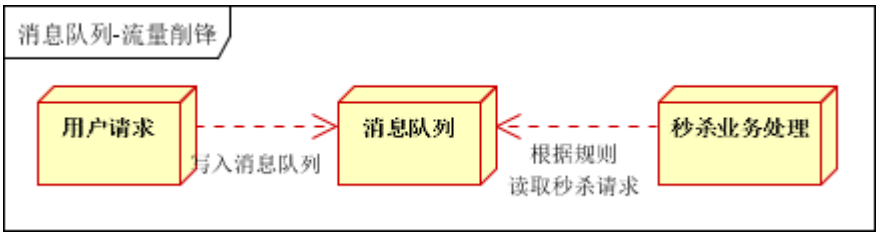
- 引入消息队列



订单系统：用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功 库存系统：订阅下单的消息，采用拉/推的方式，获取下单信息，库存系统根据下单信息，进行库存操作 假如：在下单时库存系统不能正常使用。也不影响正常下单，因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。实现订单系统与库存系统的应用解耦

2.3 流量削锋

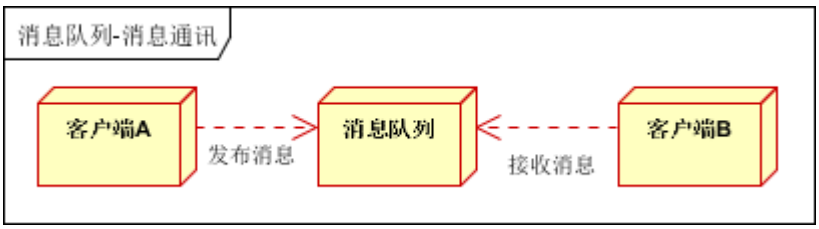
流量削锋也是消息队列中的常用场景，一般在秒杀或团抢活动中使用广泛。 应用场景：秒杀活动，一般会因为流量过大，导致流量暴增，应用挂掉。为解决这个问题，一般需要在应用前端加入消息队列。 a、可以控制活动的人数 b、可以缓解短时间内高流量压垮应用



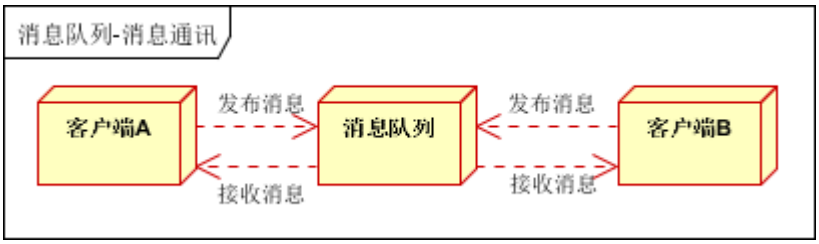
用户的请求，服务器接收后，首先写入消息队列。假如消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面。 秒杀业务根据消息队列中的请求信息，再做后续处理

2.4 消息通讯

- 点对点通讯

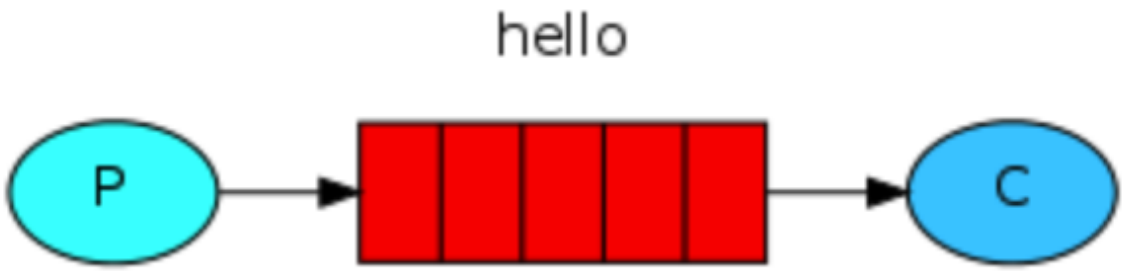


- 聊天时通讯



二、RabbitMQ基本示例

1. 单发送单接收 - 生产者消费者模型



生产者send.py

```
import pika
# 创建凭证，使用rabbitmq用户密码登录
# 去邮局取邮件，必须得验证身份
credentials = pika.PlainCredentials("s16","123")
# 新建连接，这里localhost可以更换为服务器ip
# 找到这个邮局，等于连接上服务器
connection =
pika.BlockingConnection(pika.ConnectionParameters('192.168.15.27',credentials=credentials))
# 创建频道
# 建造一个大邮箱，隶属于这家邮局的邮箱，就是个连接
channel = connection.channel()
# 声明一个队列，用于接收消息，队列名字叫“水许传”
channel.queue_declare(queue='水许传')
# 注意在rabbitmq中，消息想要发送给队列，必须经过交换(exchange)，初学可以使用空字符串交换(exchange='')，
它允许我们精确的指定发送给哪个队列(routing_key=''),参数body值发送的数据
channel.basic_publish(exchange='',
                      routing_key='水许传',
                      body='武松又去打老虎啦2')
print("已经发送了消息")
# 程序退出前，确保刷新网络缓冲以及消息发送给rabbitmq，需要关闭本次连接
connection.close()
```

消费者 receive.py

```
import pika
# 建立与rabbitmq的连接
credentials = pika.PlainCredentials("s16","123")
connection =
pika.BlockingConnection(pika.ConnectionParameters('192.168.25.27',credentials=credentials))
channel = connection.channel()
channel.queue_declare(queue="水许传")

def callback(ch,method,properties,body):
    print("消费者接收到了任务：%r"%body.decode("utf8"))
# 有消息来临，立即执行callback，没有消息则夯住，等待消息
# 老百姓开始去邮箱取邮件啦，队列名字是水许传
channel.basic_consume(callbak,queue="水许传",no_ack=True)
# 开始消费，接收消息
channel.start_consuming()
```

2. rabbitmq消息确认之ack

ack :

- 默认情况下，生产者发送数据给队列，消费者取出消息后，数据将被清除。
- 特殊情况，如果消费者处理过程中，出现错误，数据处理没有完成，那么这段数据将从队列丢失
- ACK机制用于保证消费者如果拿了队列的消息，`客户端`处理时出错了，那么队列中仍然还存在这个消息，提供下一位消费者继续取

no -ack

- **不确认机制**就是说每次消费者接收到数据后，不管是否处理完毕，rabbitmq-server都会把这个消息标记完成，从队列中删除

3. ack机制

send.py

```
import pika
# 创建凭证，使用rabbitmq用户密码登录
# 去邮局取邮件，必须得验证身份
credentials = pika.PlainCredentials("s14","123")
# 新建连接，这里localhost可以更换为服务器ip
# 找到这个邮局，等于连接上服务器
connection =
pika.BlockingConnection(pika.ConnectionParameters('192.168.119.10',credentials=credentials)
)
# 创建频道
# 建造一个大邮箱，隶属于这家邮局的邮箱，就是个连接
channel = connection.channel()
# 新建一个hello队列，用于接收消息
# 这个邮箱可以收发各个班级的邮件，通过
channel.queue_declare(queue='金品没')
# 注意在rabbitmq中，消息想要发送给队列，必须经过交换(exchange)，初学可以使用空字符串交换(exchange='')，
它允许我们精确的指定发送给哪个队列(routing_key=''),参数body值发送的数据
channel.basic_publish(exchange='',
                      routing_key='金品没',
                      body='潘金莲又出去。。。')
print("已经发送了消息")
# 程序退出前，确保刷新网络缓冲以及消息发送给rabbitmq，需要关闭本次连接
connection.close()
```

receive.py

拿到消息必须给rabbitmq服务端回复ack信息，否则消息不会被删除，防止客户端出错，数据丢失

```
import pika

credentials = pika.PlainCredentials("s14","123")
connection =
pika.BlockingConnection(pika.ConnectionParameters('192.168.119.10',credentials=credentials)
)
channel = connection.channel()

# 声明一个队列(创建一个队列)
channel.queue_declare(queue='金品没')

def callback(ch, method, properties, body):
    print("消费者接受到了任务: %r" % body.decode("utf-8"))
    # int('asdfasdf')
    # 我告诉rabbitmq服务端，我已经取走了消息
    # 回复方式在这
    ch.basic_ack(delivery_tag=method.delivery_tag)
# 关闭no_ack，代表给与服务端ack回复，确认给与回复
```

```
channel.basic_consume(callback,queue='金品没',no_ack=False)

channel.start_consuming()
```

三、RabbitMQ 消息持久化（durable、properties）

为了保证RabbitMQ在退出或者crash等异常情况下数据没有丢失，需要将queue，exchange和Message都持久化。

- **每次声明队列的时候，都加上durable**，注意每个队列都得写，客户端、服务端声明的时候都得写。

  ```
  # 在管道里声明queue
  channel.queue_declare(queue='hello2', durable=True)
  ```

  测试结果发现，只是把队列持久化了，但是队列里的消息没了。 durable的作用只是把队列持久化

- **发送端发送消息时，加上properties**

  ```
  properties=pika.BasicProperties(
      delivery_mode=2,  # 消息持久化
      )
  ```

2. 示例

send.py

```
import pika
# 无密码
# connection = pika.BlockingConnection(pika.ConnectionParameters('123.206.16.61'))
# 有密码
credentials = pika.PlainCredentials("s14","123")
connection =
pika.BlockingConnection(pika.ConnectionParameters('192.168.119.10',credentials=credentials)
)
channel = connection.channel()
# 声明一个队列(创建一个队列)
# 默认此队列不支持持久化，如果服务挂掉，数据丢失
# durable=True 开启持久化，必须新开启一个队列，原本的队列已经不支持持久化了
'''
实现rabbitmq持久化条件
 delivery_mode=2
使用durable=True声明queue是持久化

'''
channel.queue_declare(queue='LOL',durable=True)
channel.basic_publish(exchange='',
                      routing_key='LOL', # 消息队列名称
                      body='德玛西亚万岁',
                      # 支持数据持久化
                      properties=pika.BasicProperties(
                          delivery_mode=2,#代表消息是持久的  2
                      )
```

```
                        )
connection.close()
```

receive.py

```
import pika
credentials = pika.PlainCredentials("s14","123")
connection =
pika.BlockingConnection(pika.ConnectionParameters('192.168.119.10',credentials=credentials)
)
channel = connection.channel()
# 确保队列持久化
channel.queue_declare(queue='LOL',durable=True)

'''
必须确保给与服务端消息回复，代表我已经消费了数据，否则数据一直持久化，不会消失
'''
def callback(ch, method, properties, body):
    print("消费者接受到了任务: %r" % body.decode("utf-8"))
    # 模拟代码报错
    # int('asdfasdf')    # 此处报错，没有给予回复，保证客户端挂掉，数据不丢失

    # 告诉服务端，我已经取走了数据，否则数据一直存在
    ch.basic_ack(delivery_tag=method.delivery_tag)
# 关闭no_ack，代表给与回复确认
channel.basic_consume(callback,queue='LOL',no_ack=False)
channel.start_consuming(
```

四、RabbitMQ 广播模式（exchange）

前面的效果都是一对一发，如果做一个广播效果可不可以，这时候就要用到exchange了 。exchange必须精确的知道收到的消息要发给谁。exchange的类型决定了怎么处理， 类型有以下几种：

- **fanout**:exchange将消息发送给和该exchange连接的所有queue；也就是所谓的广播模式；此模式下忽略routing_key
- **direct**: 通过routingKey和exchange决定的那个唯一的queue可以接收消息，只有routing_key为“black”时才将其发送到队列queue_name；
- **topic**： 所有符合routingKey(此时可以是一个表达式)的routingKey所bind的queue可以接收消息

1.fanout 纯广播、all

需要queue和exchange绑定，因为消费者不是和exchange直连的，消费者是连在queue上，queue绑定在exchange上，消费者只会在queue里读取消息

```
          |------------------------|
          |            /-- queue <-|-> consumer1
producer -|-exchange1 <bind        |
       \  |            \-- queue <-|-> consumer2
        \-|-exchange2    ……        |
          |------------------------|
```

发送端 publisher 发布、广播

```
import pika
import sys

connection = pika.BlockingConnection(pika.ConnectionParameters(
        host='localhost'))
channel = connection.channel()
# 注意：这里是广播，不需要声明queue
channel.exchange_declare(exchange='logs',  # 声明广播管道
                         type='fanout')

# message = ' '.join(sys.argv[1:]) or "info: Hello World!"
message = "info: Hello World!"
channel.basic_publish(exchange='logs',
                      routing_key='',  # 注意此处空，必须有
                      body=message)
print(" [x] Sent %r" % message)
connection.close()
```

接收端 subscriber 订阅

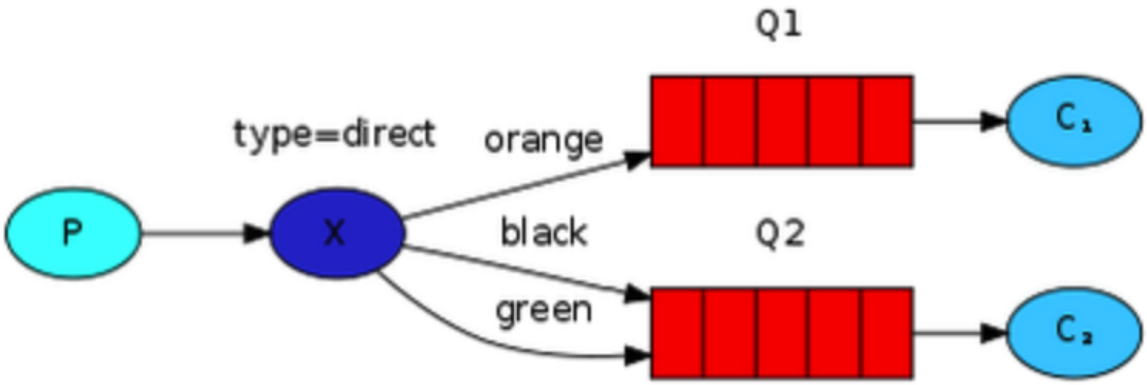
```
import pika

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

channel.exchange_declare(exchange='logs',
                         type='fanout')
# 不指定queue名字,rabbit会随机分配一个名字,exclusive=True会在使用此queue的消费者断开后,自动将queue删除
result = channel.queue_declare(exclusive=True)
# 获取随机的queue名字
queue_name = result.method.queue
print("random queuename:", queue_name)

channel.queue_bind(exchange='logs',  # queue绑定到转发器上
                   queue=queue_name)

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

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

channel.basic_consume(callback,
                      queue=queue_name,
                      no_ack=True)

channel.start_consuming()
```

注意：广播，是实时的，收不到就没了，消息不会存下来，类似收音机

2、direct 有选择的接收消息

路由模式，通过routing_key将消息发送给对应的queue; 如下面这句即可设置exchange为direct模式，只有routing_key为“black”时才将其发送到队列queue_name；

`channel.queue_bind(exchange=exchange_name,queue=queue_name,routing_key='black')`



在上图中，Q1和Q2可以绑定同一个key，如绑定routing_key=‘KeySame’，那么收到routing_key为KeySame的消息时将会同时发送给Q1和Q2，退化为广播模式；

发送端publisher

```
import pika
import sys

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

channel.exchange_declare(exchange='direct_logs',
                         type='direct')
# 重要程度级别，这里默认定义为 info
severity = sys.argv[1] if len(sys.argv) > 1 else 'info'
message = ' '.join(sys.argv[2:]) or 'Hello World!'
channel.basic_publish(exchange='direct_logs',
                      routing_key=severity,
                      body=message)
print(" [x] Sent %r:%r" % (severity, message))
connection.close()
```

接收端subscriber

```
import pika
import sys

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

channel.exchange_declare(exchange='direct_logs',
                         type='direct')
```

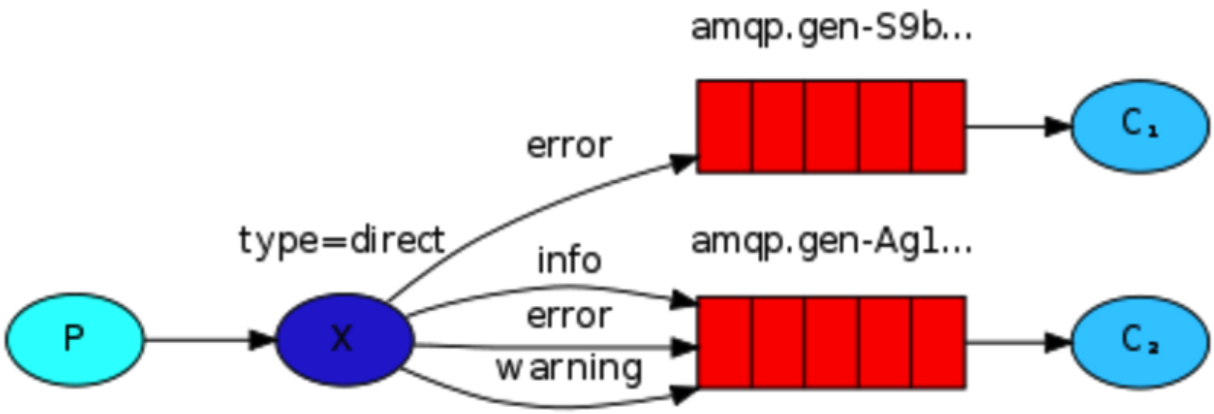
```
result = channel.queue_declare(exclusive=True)
queue_name = result.method.queue
# 获取运行脚本所有的参数
severities = sys.argv[1:]
if not severities:
    sys.stderr.write("Usage: %s [info] [warning] [error]\n" % sys.argv[0])
    sys.exit(1)
# 循环列表去绑定
for severity in severities:
    channel.queue_bind(exchange='direct_logs',
                       queue=queue_name,
                       routing_key=severity)

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

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

channel.basic_consume(callback,
                      queue=queue_name,
                      no_ack=True)

channel.start_consuming()
```

3.topic

topic模式类似于direct模式，只是其中的routing_key变成了一个有“.”分隔的字符串，“.”将字符串分割成几个单词，每个单词代表一个条件；

五、关键字发布Exchange

之前事例，发送消息时明确指定某个队列并向其中发送消息，RabbitMQ还支持根据关键字发送，即：队列绑定关键字，发送者将数据根据关键字发送到消息exchange，exchange根据 关键字 判定应该将数据发送至指定队列。



消费者1.py

路由关键字是sb,alex

```
```

```
import pika
credentials = pika.PlainCredentials("root","123")
connection =
pika.BlockingConnection(pika.ConnectionParameters('123.206.16.61',credentials=credentials))
channel = connection.channel()

# exchange='m1',exchange(秘书)的名称
# exchange_type='fanout' ， 秘书工作方式将消息发送给所有的队列
channel.exchange_declare(exchange='m2',exchange_type='direct')

# 随机生成一个队列,队列退出时，删除这个队列
result = channel.queue_declare(exclusive=True)
queue_name = result.method.queue

# 让exchange和queque进行绑定，只要
channel.queue_bind(exchange='m2',queue=queue_name,routing_key='alex')
channel.queue_bind(exchange='m2',queue=queue_name,routing_key='sb')

def callback(ch, method, properties, body):
    print("消费者接受到了任务: %r" % body)

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

消费者2.py

路由关键字sb

```
import pika

credentials = pika.PlainCredentials("root","123")
connection =
pika.BlockingConnection(pika.ConnectionParameters('123.206.16.61',credentials=credentials))
channel = connection.channel()

# exchange='m1',exchange(秘书)的名称
# exchange_type='fanout' ， 秘书工作方式将消息发送给所有的队列
channel.exchange_declare(exchange='m2',exchange_type='direct')

# 随机生成一个队列
result = channel.queue_declare(exclusive=True)
queue_name = result.method.queue

# 让exchange和queque进行绑定.
channel.queue_bind(exchange='m2',queue=queue_name,routing_key='sb')

def callback(ch, method, properties, body):
    print("消费者接受到了任务: %r" % body)
channel.basic_consume(callback,queue=queue_name,no_ack=True)
channel.start_consuming()
```

生产者.py

发送消息给匹配的路由，sb或者alex

```
import pika
credentials = pika.PlainCredentials("root","123")
connection =
pika.BlockingConnection(pika.ConnectionParameters('123.206.16.61',credentials=credentials))
channel = connection.channel()

# 路由模式的交换机会发送给绑定的key和routing_key匹配的队列
channel.exchange_declare(exchange='m2',exchange_type='direct')
# 发送消息，给有关sb的路由关键字
channel.basic_publish(exchange='m2',
                      routing_key='sb',
                      body='aaaalexlaolelaodi')

connection.close()
```