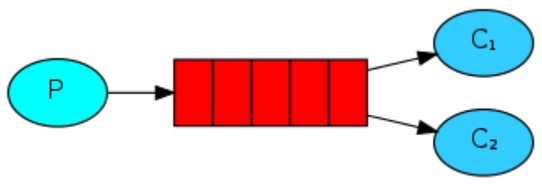
# Python RabbitMQ任务调度

一. 轮询调度

在这篇教程中, 我们将创建一个工作队列(Work Queue), 它会发送一些耗时的任务给多个工作者(Worker); 工作队列是为了避免等待一些占用大量资源、时间的操作, 当我们把任务(Task)当作消息发送到队列中, 一个运行在后台的工作者(worker)进程就会取出任务然后处理, 当你运行多个工作者(workers), 任务就会在它们之间共享.




在之前的教程中, 我们发送了一个包含"Hello World!"的字符串消息, 现在我们将发送一些字符串, 把这些字符串当作复杂的任务. 使用time.sleep()函数来模拟任务执行过程中的耗时情况, 我们在字符串中加上点号(.)来表示任务的复杂程度, 一个点(.)将会耗时1秒钟. 比如"Hello..."就会耗时3秒钟, 我们对之前教程的send.py做些简单的调整, 以便可以发送随意的消息. 这个程序会按照计划发送任务到我们的工作队列中.

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.queue_declare(queue='hello')
channel.basic_publish(exchange='',
                      routing_key='hello',
                      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.queue_declare(queue='hello')
def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)
    time.sleep(body.decode(encoding="utf-8").count('.'))
    print('done')
channel.basic_consume('hello', callback, auto_ack=True)
print(' [*] Waiting for messages. To exit press CTRL+C')
channel.start_consuming()
```

默认来说, RabbitMQ会按顺序得把消息发送给每个消费者(consumer), 平均每个消费者都会收到同等数量得消息, 这种发送消息得方式叫做—轮询(round-robin)

生产者发送数据:

```
(backup-platform) [root@backup-platform class2]# python demo1.py first msg.
 [x] Sent 'Hello World!'
(backup-platform) [root@backup-platform class2]# python demo1.py first msg..
 [x] Sent 'Hello World!'
(backup-platform) [root@backup-platform class2]# python demo1.py first msg...
 [x] Sent 'Hello World!'
(backup-platform) [root@backup-platform class2]# python demo1.py first msg....
 [x] Sent 'Hello World!'
(backup-platform) [root@backup-platform class2]# python demo1.py first msg.....
 [x] Sent 'Hello World!'
```

消费者对任务队列中任务消费的情况如下:

Worker1:

```
 [x] Received b'first msg.'
done
 [x] Received b'first msg...'
done
 [x] Received b'first msg.....'
done
```

Worker2:

```
 [x] Received b'first msg..'
done!
 [x] Received b'first msg....'
done!
```

二. 消息确认

当处理一个比较耗时得任务的时候，你也许想知道消费者(consumers)是否运行到一半就挂掉，当前的代码中当消息被RabbitMQ发送给消费者(consumers)之后，消息马上就会在内存中移除．这种情况下只要把一个工作者(worker)停止，正在处理的消息就会丢失．同时所有发送到这个工作者的还没有处理的消息都会丢失．我们不想丢失任何任务消息，如果一个工作者(worker)挂掉了，我们希望任务会重新发送给其他的工作者(worker)，为了防止消息丢失，RabbitMQ提供了消息响应(acknowledgments)，消费者会通过一个ack，告诉RabbitMQ已经收到并处理了某条消息，然后RabbitMQ就会释放并删除这条消息．如果消费者(consumer)挂掉了，没有发送响应，RabbitMQ就会认为消息没有被完全处理，然后重新发送给其他消费者(consumer)这样，即使工作者(workers)偶尔的挂掉，也不会丢失消息；消息响应默认是开启的，之前的例子中我们可以使用auto_ack=True标识把它关闭；现在将这个标识移除，当工作者(worker)完成了任务，就发送一个响应．

示例代码：

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

示例代码：

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.queue_declare(queue='hello')
def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)
    time.sleep(body.decode(encoding="utf-8").count('.'))
    print('done')
    ch.basic_ack(delivery_tag = method.delivery_tag)
channel.basic_consume('hello', callback)
print(' [*] Waiting for messages. To exit press CTRL+C')
channel.start_consuming()
```

运行上面的代码，我们发现即使使用CTRL+C杀掉了一个工作者(worker)进程，消息也不会丢失．当工作者(worker)挂掉后，所有没有响应的消息会重新发送．

消息释放：
一个很容易犯的错误就是忘了basic_ack，这将会导致严重的后果．如果它不能够释放没响应的消息，RabbitMQ就会占用越来越多的内存．

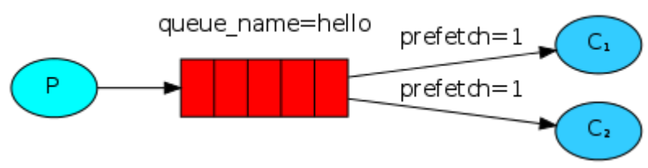
```
[root@harbor-a sbin]# ./rabbitmqctl -n rabbit@localhost list_queues  name  messages_ready messages_unacknowledged
Listing queues
hello   0       3
```

三．消息持久化
如果你没有特意告诉RabbitMQ，那么在它退出或者崩溃的时候，将会丢失所有队列和消息．为了确保信息不会丢失，有两个事情是需要注意的：我们必须把"队列"和"消息"设为持久化．

首先，为了不让队列消失，需要把队列声明为持久化(durable)：
channel.queue_declare(queue='hello', durable=True)
尽管这行代码本身是正确的，但是仍然不会正确运行，因为我们已经定义过一个叫hello的非持久化队列．RabbitMq不允许你使用不同的参数重新定义一个队列，它会返回一个错误．但我们现在使用一个快捷的解决方法——用不同的名字，例如task_queue．

channel.queue_declare(queue='task_queue', durable=True)
这个queue_declare必须在生产者(producer)和消费者(consumer)对应的代码中修改．此时我们就可以确保在RabbitMq重启之后queue_declare队列不会丢失，另外，我们需要把我们的消息也要设为持久化—将delivery_mode的属性设为2．将消息设为持久化并不能完全保证不会丢失，以上代码只是告诉了RabbitMq要把消息存到硬盘，但从RabbitMq收到消息到保存之间还是有一个很小的间隔时间，所以并不能保证真正的持久化，但已经足够应付我们的简单工作队列．如果你一定要保证持久化，你需要改写你的代码来支持事务(transaction)．

四.公平调度：
以上代码仍旧没有按照我们期望的那样进行分发，比如有两个工作者(workers)，处理奇数消息的比较繁忙，处理偶数消息的比较轻松．然而RabbitMQ并不知道这些，它仍然一如既往的派发消息.这时因为RabbitMQ只管分发进入队列的消息，不会关心有多少消费者(consumer)没有作出响应，它盲目的把第n-th条消息发给第n-th个消费者．




我们可以使用basic.qos方法，并设置prefetch_count=1．这样是告诉RabbitMQ，再同一时刻不要发送超过1条消息给一个工作者(worker)，直到它已经处理了上一条消息并且作出了响应，这样RabbitMQ就会把消息分发给下一个空闲的工作者(worker)

In [ ]:
```
channel.basic_qos(prefetch_count=1)
```

如果所有的工作者都处理繁忙状态，你的队列就会被填满．需要留意这个问题，要么添加更多的工作者(workers)，要么使用其他策略．

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.queue_declare(queue='task_queue', durable=True)
channel.basic_publish(exchange='',
                      routing_key='hello',
                      body=message,
                      properties=pika.BasicProperties(
                         delivery_mode = 2, # make message persistent
                      ))
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.queue_declare(queue='task_queue', durable=True)
def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)
    time.sleep(body.decode(encoding="utf-8").count('.'))
    print('done')
    ch.basic_ack(delivery_tag = method.delivery_tag)

channel.basic_qos(prefetch_count=1)
channel.basic_consume('hello', callback)
print(' [*] Waiting for messages. To exit press CTRL+C')
channel.start_consuming()
```