RabbitMQ

RabbitMQ是一个在AMQP基础上完整的，可复用的企业消息系统。他遵循Mozilla Public License开源协议。

MQ全称为Message Queue, 消息队列（MQ）是一种应用程序对应用程序的通信方法。应用程序通过读写出入队列的消息（针对应用程序的数据）来通信，而无需专用连接来链接它们。消 息传递指的是程序之间通过在消息中发送数据进行通信，而不是通过直接调用彼此来通信，直接调用通常是用于诸如远程过程调用的技术。排队指的是应用程序通过 队列来通信。队列的使用除去了接收和发送应用程序同时执行的要求。

**RabbitMQ安装**

**1.linux**

安装配置epel源

$ rpm -ivh http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm

安装erlang

$ yum -y install erlang

安装RabbitMQ

* yum -y install rabbitmq-server service rabbitmq-server start/stop

**2.安装python API**

pip install pika

or

easy\_install pika

先来一个基于Queue实现生产者消费者模型试试水

#!/usr/bin/env python3

#coding:utf8

import queue

import threading

message = queue.Queue(10)

def producer(i):

'''厨师,生产包子放入队列'''

while True:

message.put(i)

def consumer(i):

'''消费者,从队列中取包子吃'''

while True:

msg = message.get()

for i in range(12): 厨师的线程包子

* = threading.Thread(target=producer, args=(i,)) t.start()

for i in range(10): 消费者的线程吃包子

* = threading.Thread(target=consumer, args=(i,)) t.start()

**开始rabbitMQ**

**对于RabbitMQ来说，生产和消费不再针对内存里的一个Queue对象，而是某台服务器上的RabbitMQ Server实现的消息队列。**

**一、最基本的生产者消费者**

**1.生产者代码**

#!/usr/bin/env python

import pika

# ######################### 生产者 #########################

#链接rabbit服务器（localhost是本机，如果是其他服务器请修改为ip地址）

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

#创建频道

channel = connection.channel()

#创建一个队列名叫hello

channel.queue\_declare(queue='hello')

#exchange -- 它使我们能够确切地指定消息应该到哪个队列去。

#向队列插入数值 routing\_key是队列名 body是要插入的内容

channel.basic\_publish(exchange='',

routing\_key='hello',

body='Hello World!')

print("开始队列")

#缓冲区已经flush而且消息已经确认发送到了RabbitMQ中，关闭链接

connection.close()

**2.消费者代码**

#!/usr/bin/env python

import pika

* ########################## 消费者 ##########################

#链接rabbit

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

#创建频道

channel = connection.channel()

#如果生产者没有运行创建队列，那么消费者也许就找不到队列了。为了避免这个问题

#所有消费者也创建这个队列

channel.queue\_declare(queue='hello')

#接收消息需要使用callback这个函数来接收，他会被pika库来调用

def callback(ch, method, properties, body):

print(" [x] Received %r" % body)

#从队列取数据 callback是回调函数 如果拿到数据 那么将执行callback函数

channel.basic\_consume(callback,

queue='hello',

no\_ack=True)

print(' [\*] 等待信息. To exit press CTRL+C')

#永远循环等待数据处理和callback处理的数据

channel.start\_consuming()

**二、acknowledgment 消息不丢失的方法**

no-ack ＝ False，如果生产者遇到情况(关闭通道,连接关闭或TCP连接丢失))挂掉了，那么，RabbitMQ会重新将该任务添加到队列中。

**1.生产者不变，但是还是复制上来吧**

#!/usr/bin/env python

import pika

* ######################### 生产者 #########################

#链接rabbit服务器

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

#创建频道

channel = connection.channel()

#创建一个队列名叫hello

channel.queue\_declare(queue='hello')

#向队列插入数值 routing\_key是队列名 body是要插入的内容

channel.basic\_publish(exchange='',

routing\_key='hello',

body='Hello World!')

print("开始队列")

connection.close()

**2.消费者**

import pika

#链接rabbit

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

#创建频道

channel = connection.channel()

#如果生产者没有运行创建队列，那么消费者创建队列

channel.queue\_declare(queue='hello')

def callback(ch, method, properties, body):

print(" [x] Received %r" % body)

import time

time.sleep(10)

print 'ok'

ch.basic\_ack(delivery\_tag = method.delivery\_tag) #主要使用此代码

channel.basic\_consume(callback,

queue='hello',

no\_ack=False)

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

channel.start\_consuming()

当生产者生成一条数据，被消费者接收，消费者中断后如果不超过10秒，连接的时候数据还在。当超过10秒之后，重新链接，数据将消

失。消费者等待链接。

**三、durable 消息不丢失 （消息持久化）**

这个 queue\_declare 需要在 生产者（producer） 和消费方（consumer) 代码中都进行设置。

**1.生产者**

#!/usr/bin/env python

import pika

#链接rabbit服务器

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

#创建频道

channel = connection.channel()

#创建队列，使用durable方法

channel.queue\_declare(queue='hello', durable=True)

#如果想让队列实现持久化那么加上durable=True

channel.basic\_publish(exchange='',

routing\_key='hello',

body='Hello World!',

properties=pika.BasicProperties(

delivery\_mode=2,

#标记我们的消息为持久化的 - 通过设置 delivery\_mode 属性为 2

#这样必须设置，让消息实现持久化

))

#这个exchange参数就是这个exchange的名字. 空字符串标识默认的或者匿名的exchange：如果存在routing\_key, 消息路由到routing\_key指定的队列中。

print(" [x] 开始队列'")

connection.close()

**2.消费者**

#!/usr/bin/env python

* -\*- coding:utf-8 -\*-import pika

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

#创建频道

channel = connection.channel()

#创建队列，使用durable方法

channel.queue\_declare(queue='hello', durable=True)

def callback(ch, method, properties, body):

print(" [x] Received %r" % body)

import time

time.sleep(10)

print 'ok'

ch.basic\_ack(delivery\_tag = method.delivery\_tag)

channel.basic\_consume(callback,

queue='hello',

no\_ack=False)

print(' [\*] 等待队列. To exit press CTRL+C')

channel.start\_consuming()

注：标记消息为持久化的并不能完全保证消息不会丢失，尽管告诉RabbitMQ保存消息到磁盘，当RabbitMQ接收到消息还没有保存的时候仍然有一个短暂的时间窗口. RabbitMQ不会对每个消息都执行同步fsync(2) --- 可能只是保存到缓存cache还没有写入到磁盘中，这个持久化保证不是很强，但这比我们简单的任务queue要好很多，如果你想很强的保证你可以使用 publisher confirms

**四、消息获取顺序**

默认消息队列里的数据是按照顺序被消费者拿走，例如：消费者1 去队列中获取 奇数 序列的任务，消费者1去队列中获取 偶数 序列的任务。

channel.basic\_qos(prefetch\_count=1) 表示谁来谁取，不再按照奇偶数排列

1.生产者

import pika

import sys

connection = pika.BlockingConnection(pika.ConnectionParameters(

host='localhost'))

channel = connection.channel()

* 设置队列为持久化的队列

channel.queue\_declare(queue='task\_queue', durable=True)

message = ' '.join(sys.argv[1:]) or "Hello World!"

channel.basic\_publish(exchange='',

routing\_key='task\_queue',

body=message,

properties=pika.BasicProperties(

delivery\_mode = 2, # 设置消息为持久化的

))

print(" [x] Sent %r" % message)

connection.close()

2.消费者

#!/usr/bin/env python

* -\*- coding:utf-8 -\*-import pika

connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost')) channel = connection.channel() channel.queue\_declare(queue='hello'durable=True) # 设置队列持久化

def callback(ch, method, properties, body):

print(" [x] Received %r" % body)

import time

time.sleep(10)

print 'ok'

ch.basic\_ack(delivery\_tag = method.delivery\_tag)

#表示谁来谁取，不再按照奇偶数排列

channel.basic\_qos(prefetch\_count=1)# 消息未处理完前不要发送信息的消息

channel.basic\_consume(callback,

queue='hello',

no\_ack=False)

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

channel.start\_consuming()



**交换 (Exchanges)**

exchange类型可用： direct , topic , headers 和 fanout 。 我们将要对最后一种进行讲解 --- fanout **一、消息发布订阅**

发布订阅和简单的消息队列区别在于，发布订阅会将消息发送给所有的订阅者，而消息队列中的数据被消费一次便消失。所以，RabbitMQ实现发布和订阅时，会为每一个订阅者创建一个队列，而发布者发布消息时，会将消息放置在所有相关队列中。

**exchange type = fanout**

1.发布者

#!/usr/bin/env python

import pika

import sys

connection = pika.BlockingConnection(pika.ConnectionParameters(

host='localhost'))

channel = connection.channel()

channel.exchange\_declare(exchange='logs',

type='fanout')

message = ' '.join(sys.argv[1:]) or "info: Hello World!"

channel.basic\_publish(exchange='logs',

routing\_key='',

body=message)

print(" [x] Sent %r" % message)

connection.close()

2.订阅者

#!/usr/bin/env python

import pika

connection = pika.BlockingConnection(pika.ConnectionParameters(

host='localhost'))

channel = connection.channel()

channel.exchange\_declare(exchange='logs',

type='fanout')

result = channel.queue\_declare(exclusive=True) #队列断开后自动删除临时队列

queue\_name = result.method.queue # 队列名采用服务端分配的临时队列

channel.queue\_bind(exchange='logs',

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()

**六、关键字发送**

exchange type = direct

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。

1.生产者：

#!/usr/bin/env python3

#coding:utf8

#######################生产者#################

import pika

import sys

connection = pika.BlockingConnection(pika.ConnectionParameters(

host='localhost'))

channel = connection.channel()

channel.exchange\_declare(exchange='direct\_logs',

type='direct')

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()

2.消费者：

#!/usr/bin/env python3

#coding:utf8

import pika

import sys

############消费者####

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

severities = sys.argv[1:]

if not severities:

sys.stderr.write("Usage: %s [info] [warning] [error]\n" % sys.argv[0])

sys.exit(1)

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()

**七、模糊匹配**

exchange type = topic

在topic类型下，可以让队列绑定几个模糊的关键字，之后发送者将数据发送到exchange，exchange将传入”路由值“和 ”关键字“进行匹配，匹配成功，则将数据发送到指定队列。

* 表示可以匹配 0 个 或 多个 单词

 表示只能匹配 一个 单词

发送者路由值 队列中

old.boy.python old.\* -- 不匹配

old.boy.python old.# -- 匹配

1.消费者

#!/usr/bin/env python3

#coding:utf8

import pika

import sys

connection = pika.BlockingConnection(pika.ConnectionParameters(

host='localhost'))

channel = connection.channel()

channel.exchange\_declare(exchange='topic\_logs',

type='topic')

result = channel.queue\_declare(exclusive=True)

queue\_name = result.method.queue

binding\_keys = sys.argv[1:]

if not binding\_keys:

sys.stderr.write("Usage: %s [binding\_key]...\n" % sys.argv[0])

sys.exit(1)

for binding\_key in binding\_keys:

channel.queue\_bind(exchange='topic\_logs',

queue=queue\_name,

routing\_key=binding\_key)

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()

2.生产者

#!/usr/bin/env python3

#coding:utf8

import pika

import sys

connection = pika.BlockingConnection(pika.ConnectionParameters(

host='localhost'))

channel = connection.channel()

channel.exchange\_declare(exchange='topic\_logs',

type='topic')

routing\_key = sys.argv[1] if len(sys.argv) > 1 else 'anonymous.info'

message = ' '.join(sys.argv[2:]) or 'Hello World!'

channel.basic\_publish(exchange='topic\_logs',

routing\_key=routing\_key,

body=message)

print(" [x] Sent %r:%r" % (routing\_key, message))

connection.close()



更多内容：以下参考：

<http://blog.csdn.net/songfreeman/article/details/50945025>

**work queue （用来在多个workers之间分发消息）**

1.循环调度(Round-robin dispatching)

使用多个消费者来接收并处理消息

默认，RabbitMQ将循环的发送每个消息到下一个Consumer , 平均每个Consumer都会收到同样数量的消息。 这种分发消息的方式成为 循环调度（round-robin)

 生产者：

#!/usr/bin/env python3

#coding:utf8

import pika

import sys

#链接

connec = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))

channel = connec.channel()

#创建队列

channel.queue\_declare(queue='worker')

#插入数据

message = ' '.join(sys.argv[1:]) or "Hello World"

channel.basic\_publish(exchange='',

routing\_key='worker',

body=message,

properties=pika.BasicProperties(delivery\_mode = 2,)

)

print(" [x] Send %r " % message)

 消费者：

#!/usr/bin/env python3

#coding:utf8

import time

import pika

connect = pika.BlockingConnection(pika.ConnectionParameters (host='localhost'))

channel = connect.channel()

channel.queue\_declare('worker')

def callback(ch, method, properties,body):

print(" [x] Received %r" % body)

time.sleep(body.count(b'.'))

print(" [x] Done")

ch.basic\_ack(delivery\_tag = method.delivery\_tag)

channel.basic\_consume(callback,

queue='worker',

)

channel.start\_consuming()

执行的时候两个消费者等待接收消息，

第一次生产者产生消息的时候被消费者1接收

第二次生产者产生消息的时候被消费者2接收