Routing

(using the pika 0.9.8 Python client)

In the **previous tutorial** we built a simple logging system. We were able to broadcast log messages to many receivers.

In this tutorial we're going to add a feature to it - we're going to make it possible to subscribe only to a subset of the messages. For example, we will be able to direct only critical error messages to the log file (to save disk space), while still being able to print all of the log messages on the console.

Where to get help

If you're having trouble going through this tutorial you can **contact us** through the discussion list or directly.

Bindings

In previous examples we were already creating bindings. You may recall code like:

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

A binding is a relationship between an exchange and a queue. This can be simply read as: the queue is interested in messages from this exchange.

Bindings can take an extra `routing_key` parameter. To avoid the confusion with a `basic_publish` parameter we're going to call it a `binding key`. This is how we could create a binding with a key:

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

The meaning of a binding key depends on the exchange type. The `fanout` exchanges, which we used previously, simply ignored its value.

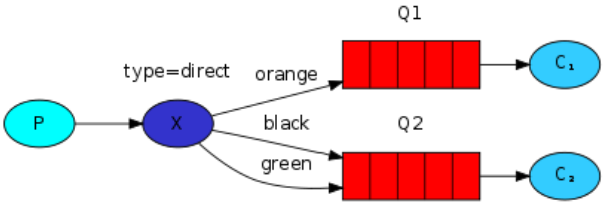
Direct exchange

Our logging system from the previous tutorial broadcasts all messages to all consumers. We want to extend that to allow filtering messages based on their severity. For example we may want the script which is writing log messages to the disk to only receive critical errors, and not waste disk space on warning or info log messages.

We were using a `fanout` exchange, which doesn't give us too much flexibility - it's only capable of mindless broadcasting.

We will use a `direct` exchange instead. The routing algorithm behind a `direct` exchange is simple - a message goes to the queues whose `binding key` exactly matches the `routing key` of the message.

To illustrate that, consider the following setup:

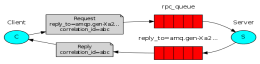


In this setup, we can see the `direct` exchange `X` with two queues bound to it. The first queue is bound with binding key `orange`, and the second has two bindings, one with binding key `black` and the other one with `green`.

In such a setup a message published to the exchange with a routing key `orange` will be routed to queue `Q1`. Messages with a routing key of `black` or `green` will go to `Q2`. All other messages will be discarded.

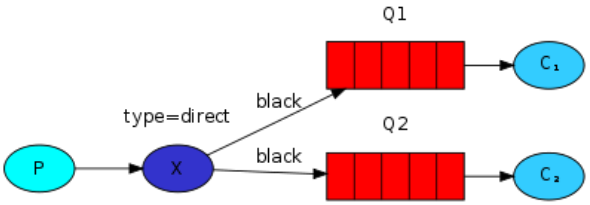
6 RPC

Remote procedure call implementation



Python | **Java**

Multiple bindings



It is perfectly legal to bind multiple queues with the same binding key. In our example we could add a binding between `X` and `Q1` with binding key `black`. In that case, the `direct` exchange will behave like `fanout` and will broadcast the message to all the matching queues. A message with routing key `black` will be delivered to both `Q1` and `Q2`.

Emitting logs

We'll use this model for our logging system. Instead of `fanout` we'll send messages to a `direct` exchange. We will supply the log severity as a `routing key`. That way the receiving script will be able to select the severity it wants to receive. Let's focus on emitting logs first.

Like always we need to create an exchange first:

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

And we're ready to send a message:

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

To simplify things we will assume that 'severity' can be one of 'info', 'warning', 'error'.

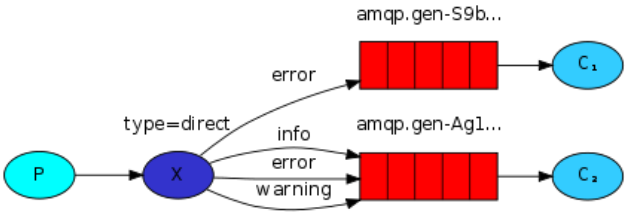
Subscribing

Receiving messages will work just like in the previous tutorial, with one exception - we're going to create a new binding for each severity we're interested in.

```
result = channel.queue_declare(exclusive=True)
queue_name = result.method.queue

for severity in severities:
    channel.queue_bind(exchange='direct_logs',
                       queue=queue_name,
                       routing_key=severity)
```

Putting it all together



The code for `emit_log_direct.py`:

```
#!/usr/bin/env python
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()
```

The code for `receive_logs_direct.py`:

```
#!/usr/bin/env python
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

severities = sys.argv[1:]
if not severities:
    print >> sys.stderr, "Usage: %s [info] [warning] [error]" % \
                         (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()
```

If you want to save only 'warning' and 'error' (and not 'info') log messages to a file, just open a console and type:

```
$ python receive_logs_direct.py warning error > logs_from_rabbit.log
```

If you'd like to see all the log messages on your screen, open a new terminal and do:

```
$ python receive_logs_direct.py info warning error
 [*] Waiting for logs. To exit press CTRL+C
```

And, for example, to emit an `error` log message just type:

```
$ python emit_log_direct.py error "Run. Run. Or it will explode."
 [x] Sent 'error':'Run. Run. Or it will explode.'
```

(Full source code for **emit_log_direct.py** and **receive_logs_direct.py**)

Move on to **tutorial 5** to find out how to listen for messages based on a pattern.

Copyright © 2013 VMware, Inc. All rights reserved. Terms of Use | Privacy