RabbitMQ

조영일

목차

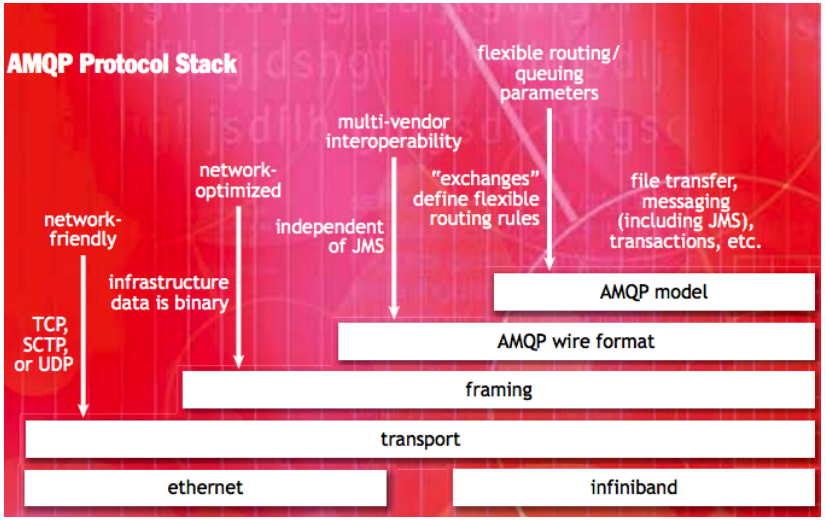
- 개요
  - AMQP와 RabbitMQ 소개
- 6 Patterns
  - Message Queue / Work Queue / Publish/Subscribe
  - Routing / Topic / RPC
- Java Usage
- 내부 동작방식
  - HA
  - Flow Control
- 성능

AMQP

- AMQP
  - Advanced Message Queuing Protocol
    - Open standard appilcation layer protocol for MOM
      - MOM: message-oriented middleware
  - 역사
    - 2003년에 JPMorgan Chase & Co.에 의해 시작됨
    - 2005년에 Cisco, IONA, iMatrix, Red Hat 등의 기술 회사와 함께 working group 결성
    - Red Hat은 **Apache Qpid** 개발 (Java/C++)
    - Rabbit Tech.는 **RabbitMQ** 개발 (Erlang)
    - 현재는 OASIS에서 관리
      - OASIS: Organization for the Advancement of Structured Information Standards

AMQP

- Architecture
  - TCP/SCTP/UDP 기반
  - JMS 개념을 확장하여 상호연동성을 높임
    - JMS: Java Message Service API
    - Elements
      - Provider / Client / Message / Queue / Topic
      - Producer/Publisher
      - Consumer/Subscriber
  - 유연한 Routing, Queuing이 가능하도록 파라미터 제공
  - 다양한 메시지 전송에 대응되는 응용 모델을 제시



RabbitMQ

- RabbitMQ
  - Erlang 기반의 AMQP 구현체
  - Rabbit Tech.가 개발
  - 2010년에 SpringSource에 인수됨 (VMware)
  - Java, .Net, Erlang 지원
    - PHP, Python, Ruby로도 개발 가능

Preliminary

- Python examples
  - with Pika library

sender

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

import pika

connection =
pika.BlockingConnection(pika.Connection
Parameters(host='localhost'))
channel = connection.channel()

channel.xxx_declare( . . . )
channel.basic_publish( . . . )

connection.close()
```

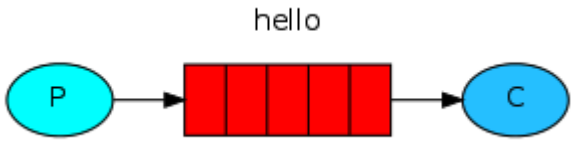
receiver

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

import pika

connection =
pika.BlockingConnection(pika.Connection
Parameters(host='localhost'))
channel = connection.channel()

channel.xxx_declare( . . . )
channel.basic_consume( . . . )
channel.start_consuming()

connection.close()
```

* xxx: queue or exchange

6 Patterns

- Simple Queue

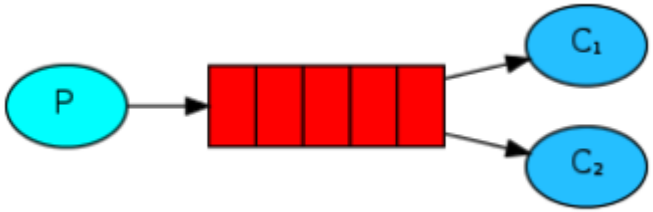



```
channel.queue_declare(queue='hello')
channel.basic_publish(exchange='', routing_key='hello', body='Hello World!')
print " [x] Sent 'Hello World!'"
```

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

channel.queue_declare(queue='hello')
print ' [*] Waiting for messages. To exit press CTRL+C‘
channel.basic_consume(callback, queue='hello', no_ack=True))
channel.start_consuming()
```

6 Patterns

- Work Queue

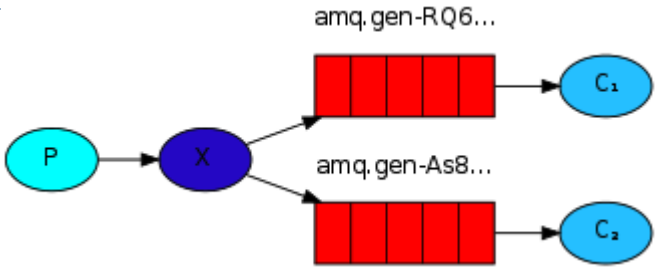



```
channel.queue_declare(queue='task_queue', durable=True)
message = ' '.join(sys.argv[1:]) or "Hello World!"
channel.basic_publish(exchange='', routing_key='task_queue',
                      body=message, properties=pika.BasicProperties(
                         delivery_mode=2, ))
print " [x] Sent %r" % (message,)
```

```
def callback(ch, method, properties, body):
    print " [x] Received %r" % (body,)
    time.sleep( body.count('.') )
    print " [x] Done"
    ch.basic_ack(delivery_tag = method.delivery_tag)

channel.queue_declare(queue='task_queue', durable=True)
print ' [*] Waiting for messages. To exit press CTRL+C'
channel.basic_qos(prefetch_count=1)
channel.basic_consume(...)
```

6 Patterns

- Publish/Subscribes

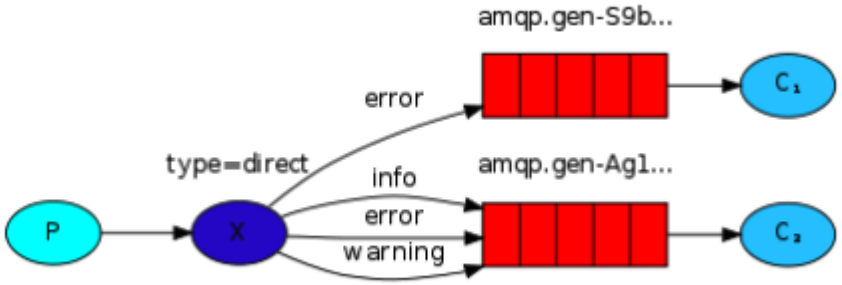



```
channel.exchange_declare(exchange='logs', type='fanout')
message = ' '.join(sys.argv[1:]) or "info: Hello World!"
channel.basic_publish(exchange='logs', routing_key='', body=message)
print " [x] Sent %r" % (message,)
```

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

channel.exchange_declare(exchange='logs', type='fanout')
result = channel.queue_declare(exclusive=True)
queue_name = result.method.queue
channel.queue_bind(exchange='logs', queue=queue_name)
print ' [*] Waiting for logs. To exit press CTRL+C'
channel.basic_consume(...)
```

6 Patterns

- Routing

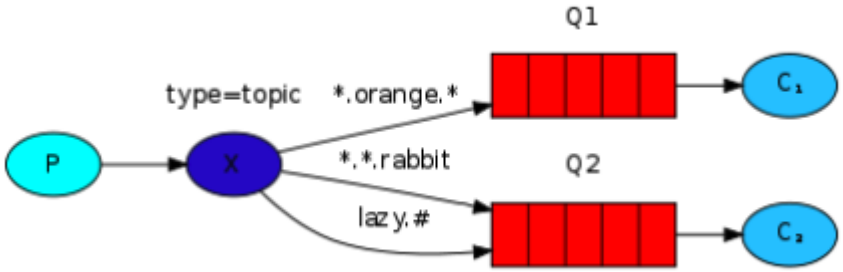



```
channel.exchange_declare(exchange='direct_logs', type='direct')
severity = sys.argv[1] if len(sys.argv) > 1 else 'info'
message = ' '.join(sys.argv[2:]) or 'Hello World!'
channel.basic_publish(exchange='direct_logs', routing_key=severity, body=message)
print " [x] Sent %r:%r" % (severity, message)
```

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

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"
channel.basic_consume(...)
```

6 Patterns

- Topics

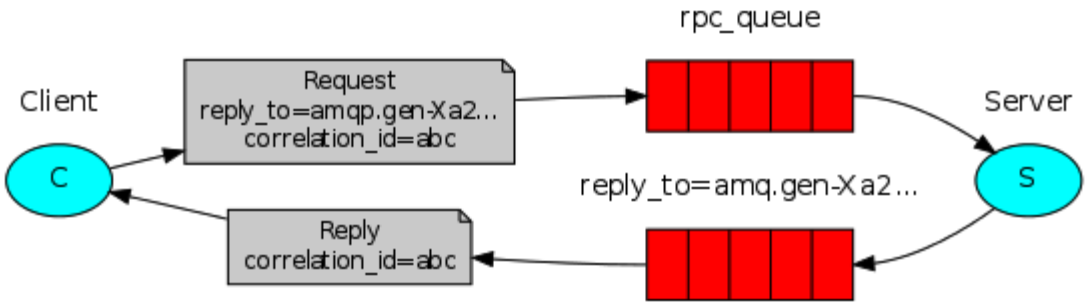



```
channel.exchange_declare(exchange='topic_logs', type='topic')
routing_key = sys.argv[1] if len(sys.argv) > 1 else 'anonymous.info'
message = ' '.join(sys.argv[2:]) or 'Hello world!'
channel.basic_publish(exchange='topic_logs', routing_key=routing_key, body=message)
print " [x] Sent %r:%r" % (routing_key, message)
```

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

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:
    print >> sys.stderr, "Usage: %s [binding_key]..." % (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 presss CTRL+C"
```

6 Patterns

- RPC (Remote Procedure Call)



6 Patterns

- RPC client

```
class FibonacciRpcClient(object):
    def __init__(self):
        result = self.channel.queue_declare(exclusive=True)
        self.callback_queue = result.method.queue
        self.channel.basic_consume(self.on_response, no_ack=True, queue=self.callback_queue)

    def on_response(self, ch, method, props, body):
        if self.corr_id == props.correlation_id:
            self.response = body

    def call(self, n):
        self.response = None
        self.corr_id = str(uuid.uuid4())
        self.channel.basic_publish(exchange='', routing_key='rpc_queue',
                                   properties=pika.BasicProperties(reply_to=self.callback_queue,
                                   correlation_id=self.corr_id,), body=str(n))
        while self.response is None:
            self.connection.process_data_events()
            return int(self.response)

fibonacci_rpc = FibonacciRpcClient()
print " [x] Requesting fib(30)"
response = fibonacci_rpc.call(30)
print " [.] Got %r" % (response,)
```

6 Patterns

- RPC server

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

def on_request(ch, method, props, body):
    n = int(body)
    print " [.] fib(%s)"  % (n,)
    response = fib(n)
    ch.basic_publish(exchange='', routing_key=props.reply_to,
                     properties=pika.BasicProperties(correlation_id=props.correlation_id),
                     body=str(response))
    ch.basic_ack(delivery_tag = method.delivery_tag)

channel.basic_qos(prefetch_count=1)
print " [x] Awaiting RPC requests"
channel.basic_consume(on_request, queue='rpc_queue')
```

Java Usage

- ShopN의 예제
  - Properties

```
rabbitmq.connection.url=localhost
rabbitmq.connection.virtualHost=/nmp
rabbitmq.connection.username=guest
rabbitmq.connection.password=guest
```

  - Spring Context

```
<bean id="connectionFactory"
class="org.springframework.amqp.rabbit.connection.SingleConnectionFactory">
    <constructor-arg value="${rabbitmq.connection.url}" />
    <property name="virtualHost" value="${rabbitmq.connection.virtualHost}" />
    <property name="username" value="${rabbitmq.connection.username}" />
    <property name="password" value="${rabbitmq.connection.password}" />
</bean>

<bean id="testProducer" class="org.springframework.amqp.rabbit.core.RabbitTemplate">
    <property name="connectionFactory" ref="connectionFactory" />
    <property name="routingKey" value="hello.queue" />
</bean>
```

Java Usage

- Producer

```
@Service
@Transactional
public class TestServiceImpl implements TestService {
    @Autowired
    @Qualifier("testProducer")
    RabbitTemplate rabbitTemplate;

    public void sendMessage() {
        MessageProperties prop = new MessageProperties();
        prop.setDeliveryMode(MessageDeliveryMode.PERSISTENT);
        prop.setReplyTo(new Address("http://10.66.19.204"));
        // 기타 필요한 프로퍼티 설정
        //entity (전송하고자 하는 Entity의 인스턴스; Serializable 구현할것)
        Message message = new Message(SerializationUtils.serialize(entity), prop);
        rabbitTemplate.send(message);
    }
}
```

Java Usage

- Consumer

```
<bean class="org.springframework.amqp.rabbit.listener.SimpleMessageListenerContainer">
    <property name="connectionFactory" ref="connectionFactory" />
    <property name="transactionManager" ref="transactionManager" />
    <property name="channelTransacted" value="true" />
    <property name="queueName" value="hello.queue" />
    <property name="messageListener" ref="testConsumer" />
</bean>
```

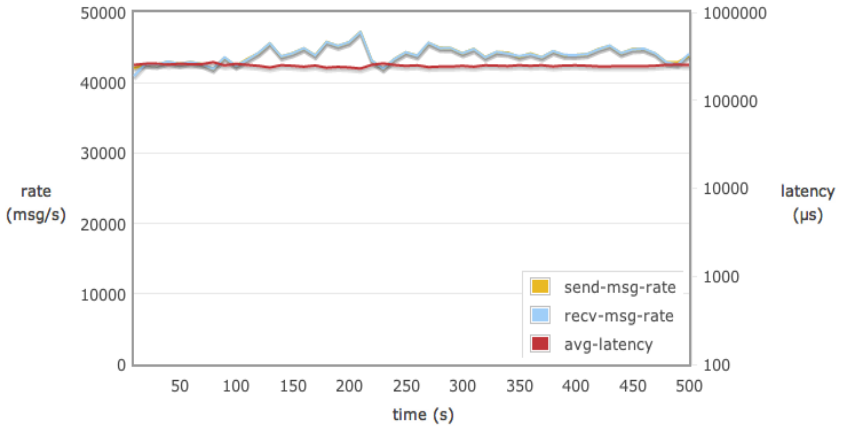
```
@Component("testConsumer")
public class TestConsumer implements MessageListener {
    @Override
    public void onMessage(Message message) {
        Category entity = (Category) SerializationUtils.deserialize(message.getBody());
        System.out.println(entity.getName());
    }
}
```

내부 동작방식

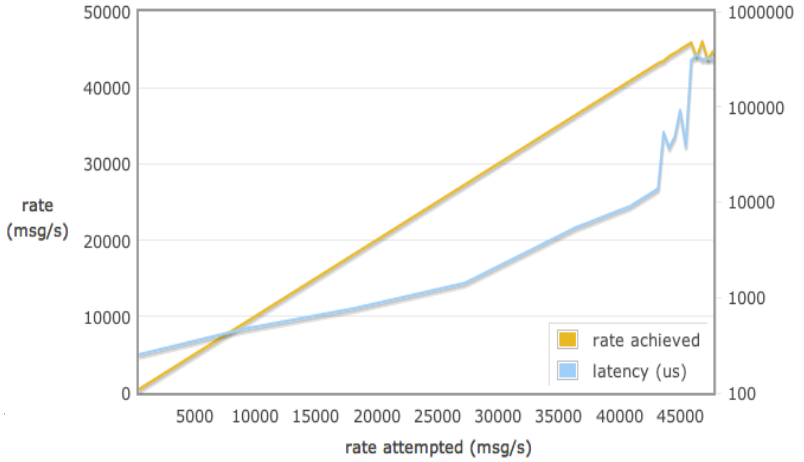
- High-availability
  - Active-Active & Mirroring
  - Master에 장애가 발생하면 동일한 상태를 유지하던 slave들 중 가장 오래된 slave를 master로 선출
- Flow Control
  - Per-connection
    - 메시지가 큐로 전달되는 속도보다 빠르게 publish되지 않게 함
  - Memory-based
    - 전체 메모리 또는 vm_memory_high_watermark의 일정 비율 (40%) 이상을 사용하지 않도록 함
  - Disk-based

성능

- RabbitMQ 2.8.1
  - Throughput: 4만 QPS
  - Response Time: 수백 milliseconds



  - RT를 수십 ms로 유지하려면 2만 5천 QPS 이하가 적당함



- Thank You!