Point To Point Channel

Exercise Notes

Introduction

- In this exercise you will create a Point To Point channel
- RMQ does not have a Point To Point channel
  - RMQ mediates all producer consumer interaction through an exchange – a dynamic router. We will cover that pattern later.
  - However, we can emulate a Point To Point channel by ensuring that one queue subscribes to one routing key
  - We could also use the Default Exchange, which maps queue names to a routing key name – but we want to build on the Direct Exchange later, so we are using convention instead.

Notes

- Much of this exercise is learning how to create AMQ model primitives
  - You will learn how to create an Exchange, a Queue and a binding between them
  - This forms the foundation for other exercises

C#

```
public PointToPointChannel(string queueName, string hostName = "localhost")
{
    //just use defaults: usr: guest pwd: guest port:5672 virtual host: /
    var factory = new ConnectionFactory() { HostName = hostName };
    factory.AutomaticRecoveryEnabled = true;
    _connection = factory.CreateConnection();
    _channel = _connection.CreateModel();

    //Because we are point to point, we are just going to use queueName for the routing key
    _routingKey = queueName;
    _queueName = queueName;

    _channel.ExchangeDeclare(ExchangeName, ExchangeType.Direct, durable: false);
    _channel.QueueDeclare(queue: _queueName, durable: false, exclusive: false, autoDelete: false, arguments: null);
    _channel.QueueBind(queue:_queueName, exchange: ExchangeName, routingKey: _routingKey);
}
```

We need to create the Exchange

We need to create the Queue

We need to route messages to the Queue

You'll need to do send as well!!

```
public string Receive()
{
    var result = _channel.BasicGet(_queueName, autoAck: true);
    if (result != null)
        return Encoding.UTF8.GetString(result.Body);
    else
        return null;
}
```

This polls for messages on the queue.

We set autoAck to true so that a message will be acked as soon as read.

This is convenient, but carries the danger that if we crash, the work will be lost!

Python

```
def __enter__(self) -> 'p2p':
    """
    We use a context manager as resources like connections need to be closed
    We return self as the channel is also the send/receive point in this point-to-point scenario
    :return: the point-to-point channel
    """
    self._connection = pika.BlockingConnection(parameters=self._connection_parameters)
    self._channel = self._connection.channel()
    self._channel.exchange_declare(exchange=p2p.exchange_name, exchange_type='direct', durable=False, auto_delete=False)
    self._channel.queue_declare(queue=self._queue_name, durable=False, exclusive=False, auto_delete=False)
    self._channel.queue_bind(queue=self._queue_name, exchange=p2p.exchange_name, routing_key=self._queue_name)

    return self
```

We need to create the Exchange

We need to create a Queue

We need to route messages to the Queue

You'll need to do send as well!!

```python
def receive(self) -> str:
    """
    We just use a basic get on the channel to retrieve the message, and return the body if it
    exists
    :return: The message or None if we could not read from the queue
    """
    method_frame, header_frame, body = self._channel.basic_get(queue=self._queue_name, no_ack=False)
    if method_frame is not None:
        self._channel.basic_ack(delivery_tag=method_frame.delivery_tag)
        return body
    else:
        return None
```

This polls for messages on the queue.

We set autoAck to false so that we must explicity acknowledge the message when done.

We ack the message.

JavaScript

```
conn.createConfirmChannel(function(err, channel){
    if (err) {
        console.error("AMDP", err.message);
        throw err;
    }

    //we don't usually use this for point-to-point which can be the default exchange
    channel.assertExchange(exchangeName, 'direct', {durable:true}, function (err, ok)
        if (err){
            console.error("AMQP", err.message);
            throw err;
        }

    });

    channel.assertQueue(me.queueName, {durable: false, exclusive: false, autoDelete:false}, function(err,ok){
        if (err){
            console.error("AMQP", err.message);
            throw err;
        }

    });

    //if we had used the default exchange, we always have a routing key equal to queue name,
    //which would be a more idiomatic way of representing point-to-point
    channel.bindQueue(me.queueName, exchangeName, me.queueName, {}, function(err, ok){
        if (err){
            console.error("AMQP", err.message);
            throw err;
        } else {
            cb(channel);
        }
    });
});
```

We need to create the Exchange

We need to create a Queue

We need to route messages to the Queue

You'll need to do send as well!!

```
//channel - the RMQ channel to make requests on
//cb a callback indicating success or failure
P2P.prototype.receive = function(channel, cb){
    channel.get(this.queueName, {noAck:true}, function(err, msgOrFalse){
        if(err){
            console.error("AMQP", err.message);
        }
        else if (msgOrFalse === false){
            cb({});
        }
        else {
            cb(msgOrFalse);
        }
    });
```

This polls for messages on the queue.

We set autoAck to false so that we must explicity acknowledge the message when done.

We ack the message.

Java

```
2 usages   iancooper
public PointToPointChannel(String queueName, String hostName) throws IOExcep
    //just use defaults: usr: guest pwd: guest port:5672 virtual host: /
    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost(hostName);
    factory.setAutomaticRecoveryEnabled(true);
    connection = factory.newConnection();
    channel = connection.createChannel();

    // Because we are point to point, we are just going to use queueName for the routing key
    routingKey = queueName;
    this.queueName = queueName;

    channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.DIRECT,  b: false);
    channel.queueDeclare(this.queueName,  b: false,  b1: false,  b2: false,  map: null);
    channel.queueBind(this.queueName, EXCHANGE_NAME, routingKey);
}
```

We need to create the Exchange

We need to create the Queue

We need to route messages to the Queue

You'll need to do send as well!!

```
public void send(String message) throws IOException {
    byte[] body = message.getBytes(StandardCharsets.UTF_8);
    channel.basicPublish(EXCHANGE_NAME, routingKey,  basicProperties: null, body);
}
```

This polls for messages on the queue.

We set autoAck to true so that a message will be acked as soon as read.

This is convenient, but carries the danger that if we crash, the work will be lost!

Go

```go
func NewChannel(qName string) *P2p {  2 usages  Ian Cooper

    channel := new(P2p)
    channel.xchng = exchange
    channel.queueName = qName
    channel.routingKey = qName

    conn, err := amqp.Dial( url: "amqp://guest:guest@localhost:5672/")
    failOnError(err,  msg: "Failed to connect to RabbitMQ", channel)
    channel.conn = conn

    ch, err := conn.Channel()
    failOnError(err,  msg: "Failed to open a channel", channel)
    defer ch.Close()

    err = ch.ExchangeDeclare(
        exchange, // name
        kind: "direct", // type
        durable: false,    // durable
        autoDelete: false,    // auto-deleted
        internal: false,    // internal
        noWait: false,    // no-wait
        args: nil,      // arguments
    )
    failOnError(err,  msg: "Failed to declare an exchange", channel)

    _, err = ch.QueueDeclare(
        qName, // name
        durable: false, // durable
        autoDelete: false, // delete when unused
        exclusive: false, // exclusive
        noWait: false, // no-wait
        args: nil,   // arguments
    )
    failOnError(err,  msg: "Failed to declare a queue", channel)

    err = ch.QueueBind(
        channel.queueName,  // queue name
        channel.routingKey, // routing key
        exchange,           // exchange
        noWait: false,
        args: nil)
    failOnError(err,  msg: "Failed to bind a queue", channel)

    return channel
}
```

We need to create the Exchange

We need to create a Queue

We need to route messages to the Queue