Invalid Message Channel

Exercise Notes

Introduction

- In this exercise you will create a Invalid Message Channel
- RMQ calls this a Dead Letter Exchange
  - The problem with this moniker is that it is used for messages you have already received
  - This is an invalid message channel
    - And the reason to use it is that the type was not expected, or as-expected
- RMQ handles failure to route via an Alternate Exchange
  - We don’t cover setting one of those up here as it is usually a policy action

Notes

- As we are declaring everything in both consumer and producer we need to declare in both locations
  - We only show one in this deck as it is the same
  - You still need to implement both
- We use the expedient of deliberately sending the wrong message
    - This is used to force an error you can see in the Dead Letter Queue
    - It is representative however of versioning errors which we discuss in part 2
- You will need to use the Management Web Page to view the results
  - As we don’t have a client of the dead letter queue

C#

```
1 usage   ianCooper
public DataTypeChannelConsumer(Func<string, T> messageDeserializer, string hostName = "localhost")
{
    _messageDeserializer = messageDeserializer;
    //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();


     /* We choose to base the key off the type name, because we want tp publish to folks interested in this type
      We name the queue after that routing key as we are point-to-point and only expect one queue to receive
     this type of message */
    var routingKey :string  = "Invalid-Message-Channel." +
    _queueName = routingKey;

    var invalidRoutingKey :string  = "invalid." + routingKey;
    var invalidMessageQueueName :string  = invalidRoutingKey;

    _channel.ExchangeDeclare(ExchangeName, ExchangeType.Direct, durable: false);
    var arguments = new Dictionary<string, object>()
    {
        {"x-dead-letter-exchange", InvalidMessageExchangeName},
        {"x-dead-letter-routing-key", invalidRoutingKey}
    };
    _channel.QueueDeclare(queue: _queueName, durable: false, exclusive: false, autoDelete: false, arguments: arguments);
    _channel.QueueBind(queue:_queueName, exchange: ExchangeName, routingKey: routingKey);

    //declare a queue for invalid messages off an invalid message exchange
    //messages that we nack without requeue will go here
    _channel.ExchangeDeclare(InvalidMessageExchangeName, ExchangeType.Direct, durable: true);
    _channel.QueueDeclare(queue: invalidMessageQueueName, durable: true, exclusive: false, autoDelete: false);
    _channel.QueueBind(queue: invalidMessageQueueName, exchange: InvalidMessageExchangeName, routingKey: invalidRoutingKey);
}
```

When we create our queue we pass it in the dead letter exchange name

To route to our DLQ we have to send in the routing key

We can create the dead letter exchange after we identify it when creating the queue

We bind the DLQ

We create the DLQ

Python

```
def __enter__(self) -> 'Consumer':
    """
    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

    We establish an exchange to use for invalid messages (RMQ confusingly calls this dead-letter) and a routing key
    to use when we reject messages to this queue, so that we can create a subscribing queue on th
    up the invalid messages.
    :return: the point-to-point channel
    """
    self._connection = pika.BlockingConnection(parameters=self._connection_parameters)
    self._channel = self._connection.channel()
    self._channel.exchange_declare(exchange=exchange_name, exch                    auto_delete=False)

    invalid_routing_key = 'invalid.' + self._routing_key
    invalid_queue_name = invalid_routing_key

    args = {'x-dead-letter-exchange': invalid_message_exchange_name, 'x-dead-letter-routing-key': invalid_routing_key}

    self._channel.queue_declare(queue=self._queue_name, durable=False, exclusive=False, auto_delete=False, arguments=args)
    self._channel.queue_bind(exchange=exchange_name, routing_key=self._routing_key, queue=self._queue_name)

    self._channel.exchange_declare(exchange=invalid_message_exchange_name, exchange_type='direct', durable=True, auto_delete
    self._channel.queue_declare(queue=invalid_queue_name, durable=True, exclusive=False, auto_delete=False)
    self._channel.queue_bind(exchange=invalid_message_exchange_name, routing_key=invalid_routing_key

    return self
```

To route to our DLQ we have to send in the routing key

When we create our queue we pass it in the dead letter exchange name

We can create the dead letter exchange after we identify it when creating the queue

We create the DLQ

We bind the DLQ

JavaScript

```
//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;
    }

});

let invalidQueueName = "invalid." + me.queueName;

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

});

channel.bindQueue(me.queueName, exchangeName, me.queueName, {}, function(err, ok){
    if (err){
        console.error("AMQP", err.message);
        throw err;
    }
    else{
        cb(channel);
    }
});

channel.assertExchange(invalidMessageExchangeName, 'direct', {durable:true}, function(err, ok){
    if (err){
        console.error("AMQP", err.message);
        throw err;
    }
});

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

channel.bindQueue(invalidQueueName, invalidMessageExchangeName, invalidQueueName, {}, function(err, ok){
    if (err){
        console.error("AMQP", err.message);
    }
});
```

When we create our queue we pass it in the dead letter exchange name

To route to our DLQ we have to send in the routing key

We can create the dead letter exchange after we identify it when creating the queue

We create the DLQ

We bind the DLQ

Java

```
public DataTypeChannelConsumer(Function<String, T> messageDeserializer, String routingKey, String hostName) throws IOException, TimeoutException {
    this.messageDeserializer = messageDeserializer;
    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost(hostName);
    factory.setAutomaticRecoveryEnabled(true);
    connection = factory.newConnection();
    channel = connection.createChannel();

    queueName = routingKey;
    var invalidRoutingKey = "invalid" + routingKey;
    var invalidMessageQueueName = invalidRoutingKey;

    channel.exchangeDeclare(EXCHANGE_NAME, BuiltinExchangeType.DIRECT, b: false);

    var arguments = new HashMap<String, Object>();
    arguments.put("x-dead-letter-exchange", INVALID_EXCHANGE_NAME);
    arguments.put("x-dead-letter-routing-key", invalidRoutingKey);

    channel.queueDeclare(queueName, b: false, b1: false, b2: false, arguments);
    channel.queueBind(queueName, EXCHANGE_NAME, routingKey);

    channel.exchangeDeclare(INVALID_EXCHANGE_NAME, BuiltinExchangeType.DIRECT, b: false);
    channel.queueDeclare(invalidMessageQueueName, b: false, b1: false, b2: false, map: null);
    channel.queueBind(invalidMessageQueueName, INVALID_EXCHANGE_NAME, invalidRoutingKey);
}
```

When we create our queue we pass it in the dead letter exchange name

To route to our DLQ we have to send in the routing key

We can create the dead letter exchange after we identify it when creating the queue

We bind the DLQ

We create the DLQ

Go

```
//We don't need a second exchange, but it's one option
err = ch.ExchangeDeclare(
    invalid_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 a invalid message exchange", channel)

_, err = ch.QueueDeclare(
    invalid_routing_key, // name
    durable: true,              // 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(
    invalid_routing_key, // queue name
    invalid_routing_key, // routing key
    invalid_exchange,    // exchange
    noWait: false,
    args: nil)
failOnError(err, msg: "Failed to bind a queue", channel)
```

```
_, err = ch.QueueDeclare(
    qName, // name
    durable: false, // durable
    autoDelete: false, // delete when unused
    exclusive: false, // exclusive
    noWait: false, // no-wait
    amqp.Table{"x-dead-letter-exchange": invalid_exchange, "x-dead-letter-routing-key": invalid_routing_key},
)
failOnError(err, msg: "Failed to declare a queue", channel)
```

When we create our queue we pass it in the dead letter exchange name

To route to our DLQ we have to send in the routing key

We can create the dead letter exchange after we identify it when creating the queue

We bind the DLQ

We create the DLQ