RabbitMQ in .NET C#: basic error handling in Receiver

JUNE 16, 2014 1 COMMENT (HTTPS://DOTNETCODR.COM/2014/06/16/RABBITMQ-IN-NET-C-BASIC-ERROR-HANDLING-IN-RECEIVER/#COMMENTS)

Introduction

This post builds upon the basics of RabbitMQ in .NET. If you are new to this topic you should check out all the previous posts listed on this (https://dotnetcodr.com/messaging/) page. I won't provide any details on bits of code that we've gone through before.

Most of the posts on RabbitMQ on this blog are based on the work of RabbitMQ guru Michael Stephenson (http://geekswithblogs.net/michaelstephenson/Default.aspx).

It can happen that the Receiver is unable to process a message it has received from the message queue.

In some cases the receiver may not be able to accept an otherwise well-formed message. That message needs to be put back into the queue for later re-processing.

There's also a case where processing a message throws an exception every time the receiver tries to process it. It will keep putting the message back to the queue only to receive the same exception over and over again. This also blocks the other messages from being processed. We call such a message a Poison Message (https://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp?topic=%2Fcom.ibm.mq.xms.doc%2Fconcepts%2Fxms_cpoison_messages.html).

In a third scenario the Receiver simply might not understand the message. It is malformed, contains unexpected properties etc.

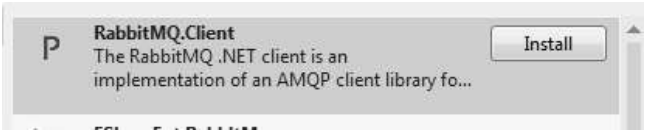
The receiver can follow 2 basic strategies: retry processing the message or discard it after the first exception. Both options are easy to implement with RabbitMQ .NET.

Demo

If you've gone through the other posts on RabbitMQ on this blog then you'll have a Visual Studio solution ready to be extended. Otherwise just create a new blank solution in Visual Studio 2012 or 2013. Add a new solution folder called FailingMessages to the solution. In that solution add the following projects:

- A console app called BadMessageReceiver
- A console app called BadMessageSender
- A C# library called MessageService

Add the following NuGet package to all three projects:



(https://dotnetcodr.files.wordpress.com/2014/04/rabbitmq-new-client-package-nuget.png)

Add a project reference to MessageService from BadMessageReceiverand BadMessageSender. Add a class called RabbitMqService to MessageService with the following code to set up the connection with the local RabbitMQ instance:

```
public class RabbitMqService
{
        private string _hostName = "localhost";
        private string _userName = "guest";
        private string _password = "guest";

        public static string BadMessageBufferedQueue = "BadMessageQueue";

        public IConnection GetRabbitMqConnection()
        {
            ConnectionFactory connectionFactory = new ConnectionFactory();
            connectionFactory.HostName = _hostName;
            connectionFactory.UserName = _userName;
            connectionFactory.Password = _password;

            return connectionFactory.CreateConnection();
        }
}
```

Let's set up the queue. Add the following code to Main of BadMessageSender:

```
RabbitMqService rabbitMqService = new RabbitMqService();
IConnection connection = rabbitMqService.GetRabbitMqConnection();
IModel model = connection.CreateModel();
model.QueueDeclare(RabbitMqService.BadMessageBufferedQueue, true, false, false, null);
```

Run the Sender project. Check in the RabbitMq management console that the queue has been set up.

Comment out the call to model.QueueDeclare, we won't need it.

Add the following code in Program.cs of the Sender:

```
private static void RunBadMessageDemo(IModel model)
{
    Console.WriteLine("Enter your message. Quit with 'q'.");
    while (true)
    {
        string message = Console.ReadLine();
        if (message.ToLower() == "q") break;
        IBasicProperties basicProperties = model.CreateBasicProperties();
        basicProperties.SetPersistent(true);
        byte[] messageBuffer = Encoding.UTF8.GetBytes(message);
        model.BasicPublish("", RabbitMqService.BadMessageBufferedQueue, basicProperties, messageBuffe
    }
}
```

This is probably the most basic message sending logic available in RabbitMQ .NET. Insert a call to this method from Main.

Now let's turn to the Receiver. Add the following code to Main in Program.cs of BadMessageReceiver:

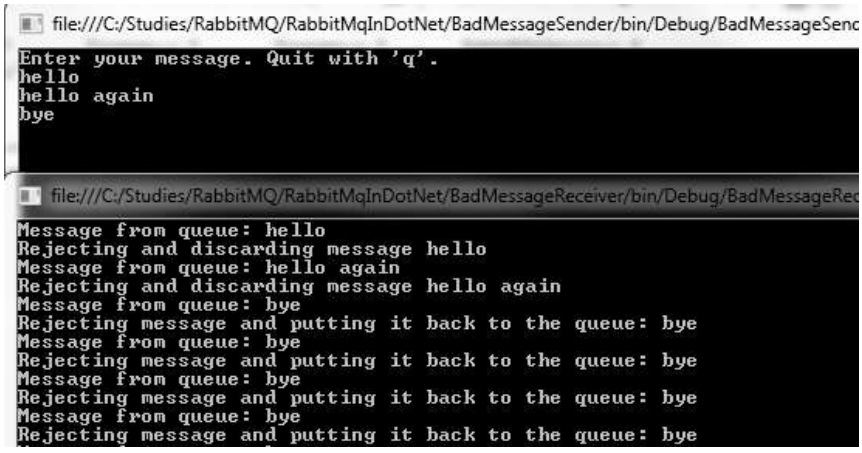
```
RabbitMqService messageService = new RabbitMqService();
IConnection connection = messageService.GetRabbitMqConnection();
IModel model = connection.CreateModel();
ReceiveBadMessages(model);
```

…where ReceiveBadMessages looks as follows:

```
1  private static void ReceiveBadMessages(IModel model)
2  {
3      model.BasicQos(0, 1, false);
4      QueueingBasicConsumer consumer = new QueueingBasicConsumer(model);
5      model.BasicConsume(RabbitMqService.BadMessageBufferedQueue, false, consumer);
6      while (true)
7      {
8          BasicDeliverEventArgs deliveryArguments = consumer.Queue.Dequeue() as BasicDeliverEventArgs;
9          String message = Encoding.UTF8.GetString(deliveryArguments.Body);
10         Console.WriteLine("Message from queue: {0}", message);
11         Random random = new Random();
12         int i = random.Next(0, 2);
13
14         //pretend that message cannot be processed and must be rejected
15         if (i == 1) //reject the message and discard completely
16         {
17             Console.WriteLine("Rejecting and discarding message {0}", message);
18             model.BasicReject(deliveryArguments.DeliveryTag, false);
19         }
20         else //reject the message but push back to queue for later re-try
21         {
22             Console.WriteLine("Rejecting message and putting it back to the queue: {0}", message);
23             model.BasicReject(deliveryArguments.DeliveryTag, true);
24         }
25     }
26 }
```

The only new bit compared to the basics is the BasicReject method. It accepts the delivery tag and a boolean parameter. If that's set to false then the message is sent back to RabbitMQ which in turn will discard it, i.e. the message is not re-entered into the queue. Else if it's true then the message is put back into the queue for a retry.

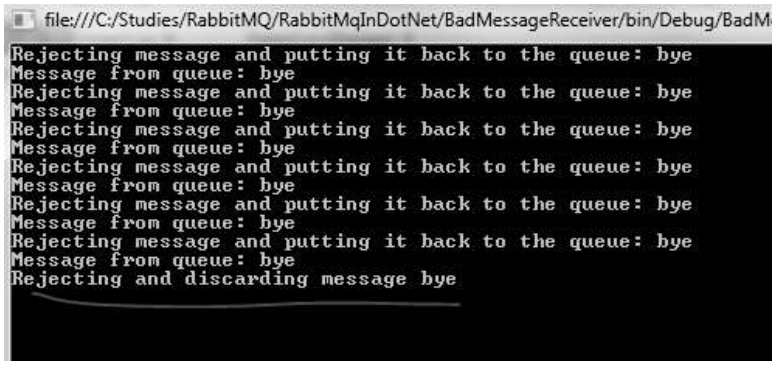
Let's run the demo. Start the Sender app first. Then right-click the Receiver app in VS, select Debug and Run new instance. You'll have two console windows up and running. Start sending messages from the Sender. Depending on the outcome of the random integer on the Receiver side you should see an output similar to this one:



(https://dotnetcodr.files.wordpress.com/2014/05/basic-retry-console-output-1.png)

In the above case the following has happened:

- Message "hello" was received and immediately discarded
- Same happened to "hello again"
- Message "bye" was put back into the queue several times before it was finally discarded – see the output below



(https://dotnetcodr.files.wordpress.com/2014/05/basic-retry-console-output-2.png)

Note that I didn't type "bye" multiple times. The reject-requeue-retry cycle was handled automatically.

The message "bye" in this case was an example of a Poison Message. In the code it was eventually rejected because the random number generator produced a 0.

This strategy was OK for demo purposes but you should do something more sophisticated in a real project. You can't just rely on random numbers. On the other hand if you don't build in any mechanism to finally discard a message then it will just keep coming back to the receiver. That will cause a "traffic jam" in the message queue as all messages will keep waiting to be delivered.

We'll look at some other strategies in the next post (https://dotnetcodr.com/2014/06/19/rabbitmq-in-net-c-more-complex-error-handling-in-the-receiver/).

View the list of posts on Messaging here (https://dotnetcodr.com/messaging/).
FILED UNDER .NET, MESSAGING TAGGED WITH C#, EXCEPTION HANDLING, RABBITMQ

About Andras Nemes
I'm a .NET/Java developer living and working in Stockholm, Sweden.

One Response to *RabbitMQ in .NET C#: basic error handling in Receiver*

dineshramitc says:
June 16, 2014 at 4:49 pm
Reblogged this on Dinesh Ram Kali..

Reply

Create a free website or blog at WordPress.com.

Advertisements

nucamp.co
22 Weeks for under $2,500

REPORT THIS AD