Messaging with RabbitMQ and .NET review part 5: one way messaging with an event based consumer

AUGUST 10, 2016 4 COMMENTS (HTTPS://DOTNETCODR.COM/2016/08/10/MESSAGING-WITH-RABBITMQ-AND-NET-REVIEW-PART-5-ONE-WAY-MESSAGING-WITH-AN-EVENT-BASED-CONSUMER/#COMMENTS)

Introduction

In the previous post (https://dotnetcodr.com/2016/08/08/messaging-with-rabbitmq-and-net-review-part-4-one-way-messaging-with-a-basic-consumer/) we saw how to process messages from a queue using a receiver we derived from a default basic consumer. We implemented the HandleBasicDeliver function for that purpose. We also discussed two message exchange patterns (MEPs), one-way and and worker queues. The two are practically identical in code but the worker queues MEP implies that we have 2 or more consumers competing for the messages from the queue. That way we can spread out the message load across multiple consumer instances.

In this short post we'll look at an alternative way to consume messages from a queue in code.

The event based queue consumer

In the previous post we built the OneWayMessageReceiver class which derived from the DefaultBasicConsumer class built into the .NET RabbitMq driver. There is an additional class called EventingBasicConsumer which exposes the message handling functions as events. If you are not sure what events and delegates are in C# you can start here (https://dotnetcodr.com/2014/02/13/events-delegates-and-lambdas-in-net-c-part-1-delegate-basics/). The end result is the same as we had previously but the syntax is different.

At this point we have a method called ReceiveSingleOneWayMessage in our RabbitMq.OneWayMessage.Receiver console application which is called from the Main method. We'll now save the channel in a private field and start consuming the messages in an event handler. Here's the entire code for clarity where I ignored the ReceiveSingleOneWayMessage function:

```
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RabbitMq.OneWayMessage.Receiver
{
    class Program
    {
        private static IModel channelForEventing;

        static void Main(string[] args)
        {
            ReceiveMessagesWithEvents();
        }

        private static void ReceiveMessagesWithEvents()
        {
            ConnectionFactory connectionFactory = new ConnectionFactory();

            connectionFactory.Port = 5672;
            connectionFactory.HostName = "localhost";
            connectionFactory.UserName = "accountant";
            connectionFactory.Password = "accountant";
            connectionFactory.VirtualHost = "accounting";

            IConnection connection = connectionFactory.CreateConnection();
            channelForEventing = connection.CreateModel();
            channelForEventing.BasicQos(0, 1, false);
            EventingBasicConsumer eventingBasicConsumer = new EventingBasicConsumer(channelForEventin
            eventingBasicConsumer.Received += EventingBasicConsumer_Received;
            channelForEventing.BasicConsume("my.first.queue", false, eventingBasicConsumer);
        }

        private static void EventingBasicConsumer_Received(object sender, BasicDeliverEventArgs e)
        {
            IBasicProperties basicProperties = e.BasicProperties;
            Console.WriteLine("Message received by the event based consumer. Check the debug window f
            Debug.WriteLine(string.Concat("Message received from the exchange ", e.Exchange));
            Debug.WriteLine(string.Concat("Content type: ", basicProperties.ContentType));
            Debug.WriteLine(string.Concat("Consumer tag: ", e.ConsumerTag));
            Debug.WriteLine(string.Concat("Delivery tag: ", e.DeliveryTag));
            Debug.WriteLine(string.Concat("Message: ", Encoding.UTF8.GetString(e.Body)));
            channelForEventing.BasicAck(e.DeliveryTag, false);
        }
    }
}
```

ReceiveMessagesWithEvents starts with the same connection and channel setup code as ReceiveSingleOneWayMessage. We save the channel in the private field to be reused in the event handler EventingBasicConsumer_Received for the acknowledgement. EventingBasicConsumer exposes a number of events of which Received is the most important. We can attach a handler to it which will be fired if there's a new message in the queue. The body of the event handler function is almost the same as the one we had in the overridden HandleBasicDeliver function of the OneWayMessageReceiver class. The parameters of HandleBasicDeliver are available in the incoming BasicDeliverEventArgs object.

You can set a breakpoint within the event handler and start the receiver application. Then send a message using the publisher we built before. The event handler should be fired with the same effect as in the previous post.

There's actually an alternative way of registering the event handler using a lambda expression as follows:

```
private static void ReceiveMessagesWithEvents()
{
    ConnectionFactory connectionFactory = new ConnectionFactory();

    connectionFactory.Port = 5672;
    connectionFactory.HostName = "localhost";
    connectionFactory.UserName = "accountant";
    connectionFactory.Password = "accountant";
    connectionFactory.VirtualHost = "accounting";

    IConnection connection = connectionFactory.CreateConnection();
    IModel channel = connection.CreateModel();
    channel.BasicQos(0, 1, false);
    EventingBasicConsumer eventingBasicConsumer = new EventingBasicConsumer(channel);

    eventingBasicConsumer.Received += (sender, basicDeliveryEventArgs) =>
    {
        IBasicProperties basicProperties = basicDeliveryEventArgs.BasicProperties;
        Console.WriteLine("Message received by the event based consumer. Check the debug window for d
        Debug.WriteLine(string.Concat("Message received from the exchange ", basicDeliveryEventArgs.E
        Debug.WriteLine(string.Concat("Content type: ", basicProperties.ContentType));
        Debug.WriteLine(string.Concat("Consumer tag: ", basicDeliveryEventArgs.ConsumerTag));
        Debug.WriteLine(string.Concat("Delivery tag: ", basicDeliveryEventArgs.DeliveryTag));
        Debug.WriteLine(string.Concat("Message: ", Encoding.UTF8.GetString(basicDeliveryEventArgs.Bod
        channel.BasicAck(basicDeliveryEventArgs.DeliveryTag, false);
    };

    channel.BasicConsume("my.first.queue", false, eventingBasicConsumer);
}
```

The above solution is based on an anonymous event handler. Moreover, we don't need to save the channel in a private field anymore. If you don't understand this lambda syntax then I encourage you to check out the link referenced above which leads you to a series on delegates, events and lambdas.

We'll explore the fanout exchange type in the next part (https://dotnetcodr.com/2016/08/15/messaging-with-rabbitmq-and-net-review-part-6-the-fanout-exchange-type/).

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

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

4 Responses to *Messaging with RabbitMQ and .NET review part 5: one way messaging with an event based consumer*

Jean-Paul says:
August 16, 2016 at 12:23 pm
Thanks, very interesting. What would you say is the differentiator to choose the Basic Consumer vs the Event Based one?

Reply

Andras Nemes says:
August 16, 2016 at 3:28 pm
Hi Jean-Paul, i don't think there is any real difference, both implementations work equally well. The event based variant saves you a new custom class, so that you have a bit less code, but that's it really.

Reply

granadacoder says:
August 18, 2016 at 10:22 pm
I figured out the hard way that "EventingBasicConsumer" does not seem to respect ".Priority" (a feature available to RabbitMQ since version 3.5).

I think I found the reason via the documentation below. I've marked the important sentence with ***.

http://rabbitmq.docs.pivotal.io/35/rabbit-web-docs/dotnet-api-guide.html.html

Retrieving Messages By Subscription ("push API")

Another way to receive messages is to set up a subscription using the IBasicConsumer interface. *** The messages will then be delivered automatically as they arrive, rather than having to be requested proactively. ***

var consumer = new EventingBasicConsumer(channel);
consumer.Received

Reply

Serkan Dede says:
January 21, 2021 at 8:44 pm
Hi Andras, first i should thank you cos you clarify many questions appears in mind after reading RabbitMQ official documentations.

I have a question to know whether you have same problem or not.
We are creating console applications and EventingBasicConsumer same as your post.
And we register this exe files as WindowsService. via nssm.

Everything is working very well when we run this services. But after some period of time, maybe 1-2 hours or more later, EventingBasicConsumer mechanism stops to listtening i guess.
(WindowsService continue to run)

Do you have any information about it?

Reply

Create a free website or blog at WordPress.com.

Advertisements

REPORT THIS AD