Messaging with RabbitMQ and .NET C# part 4: routing and topics

MAY 8, 2014 1 COMMENT (HTTPS://DOTNETCODR.COM/2014/05/08/MESSAGING-WITH-RABBITMQ-AND-NET-C-PART-4-ROUTING-AND-TOPICS/#COMMENTS)

Introduction

In this post we'll continue our discussion of the message exchange via RabbitMQ. In particular we'll investigate the following topics:

- Routing
- Topics

We'll continue building on the demo solution we've been working on, so open it already now in Visual Studio. Also, log onto the RabbitMQ management UI on http://localhost:15672/ (http://localhost:15672/)

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).

Routing

Here the client sends a message to an exchange and attaches a routing key to it. The message is sent to all queues with the matching routing key. Each queue has a receiver attached which will process the message. We'll initiate a dedicated message exchange and not use the default one. Note that a queue can be dedicated to one or more routing keys.

As usual we'll set up the queues and exchanges first. Add the following code to AmqpMessagingService.cs:

```
public void SetUpExchangeAndQueuesForRoutingDemo(IModel model)
{
    model.ExchangeDeclare(_routingKeyExchange, ExchangeType.Direct, true);
    model.QueueDeclare(_routingKeyQueueOne, true, false, false, null);
    model.QueueDeclare(_routingKeyQueueTwo, true, false, false, null);
    model.QueueBind(_routingKeyQueueOne, _routingKeyExchange, "cars");
    model.QueueBind(_routingKeyQueueTwo, _routingKeyExchange, "trucks");
}
```

…with the following private variables:

```
private string _routingKeyExchange = "RoutingKeyExchange";
private string _routingKeyQueueOne = "RoutingKeyQueueOne";
private string _routingKeyQueueTwo = "RoutingKeyQueueTwo";
```

If you'd like to bind queue 1 and the routing exchange with multiple routing keys then you can call the QueueBind multiple times:

```
model.QueueBind(_routingKeyQueueTwo, _routingKeyExchange, "trucks");
model.QueueBind(_routingKeyQueueTwo, _routingKeyExchange, "donkeys");
model.QueueBind(_routingKeyQueueTwo, _routingKeyExchange, "mules");
```

You'll recognise this code from earlier posts on RabbitMQ: we set up an exchange of type Direct, two queues and bind them using the routing keys of cars and trucks.

Insert a new Console app, call it RoutingSender. Add the usual references: RabbitMQ NuGet, RabbitMqService. Insert the following code to Main:

```
AmqpMessagingService messagingService = new AmqpMessagingService();
IConnection connection = messagingService.GetRabbitMqConnection();
IModel model = connection.CreateModel();
messagingService.SetUpExchangeAndQueuesForRoutingDemo(model);
```

Set RoutingSender as the start up project and run the application. Check in the RabbitMQ console that the exchange and queues have been set up correctly. Comment out the call to messagingService.SetUpExchangeAndQueuesForRoutingDemo.

Insert the following method to Program.cs which will extract the routing key and the message from the console entry:

```
private static void RunRoutingDemo(IModel model, AmqpMessagingService messagingService)
{
    Console.WriteLine("Enter your message as follows: the routing key, followed by a semicolon, and t
    while (true)
    {
        string fullEntry = Console.ReadLine();
        string[] parts = fullEntry.Split(new char[] { ';' }, StringSplitOptions.RemoveEmptyEntries);
        string key = parts[0];
        string message = parts[1];
        if (message.ToLower() == "q") break;
        messagingService.SendRoutingMessage(message, key, model);
    }
}
```

Add a call to this method from Main:

```
RunRoutingDemo(model, messagingService);
```

…where SendRoutingMessage in AmqpMessagingService looks as follows:

```
public void SendRoutingMessage(string message, string routingKey, IModel model)
{
    IBasicProperties basicProperties = model.CreateBasicProperties();
    basicProperties.SetPersistent(_durable);
    byte[] messageBytes = Encoding.UTF8.GetBytes(message);
    model.BasicPublish(_routingKeyExchange, routingKey, basicProperties, messageBytes);
}
```

As you see we follow the same pattern as before: we publish to an exchange and provide the routing key, the basic properties and the message body as the arguments.

In preparation for the two receivers add the following methods to AmqpMessagingService:

```
public void ReceiveRoutingMessageReceiverOne(IModel model)
{
    model.BasicQos(0, 1, false);
    Subscription subscription = new Subscription(model, _routingKeyQueueOne, false);
    while (true)
    {
        BasicDeliverEventArgs deliveryArguments = subscription.Next();
        String message = Encoding.UTF8.GetString(deliveryArguments.Body);
        Console.WriteLine("Message from queue: {0}", message);
        subscription.Ack(deliveryArguments);
    }
}

public void ReceiveRoutingMessageReceiverTwo(IModel model)
{
    model.BasicQos(0, 1, false);
    Subscription subscription = new Subscription(model, _routingKeyQueueTwo, false);
    while (true)
    {
        BasicDeliverEventArgs deliveryArguments = subscription.Next();
        String message = Encoding.UTF8.GetString(deliveryArguments.Body);
        Console.WriteLine("Message from queue: {0}", message);
        subscription.Ack(deliveryArguments);
    }
}
```

Look through the Publish/Subscribe MEP in the third part of this series (https://dotnetcodr.com/2014/05/05/messaging-with-rabbitmq-and-net-c-part-3-message-exchange-patterns/) if you're not sure what this code means.

Next add two new Console applications to the solution: RoutingReceiverOne and RoutingReceiverTwo. Add the usual references to both: RabbitMQ NuGet, RabbitMqService. Add the following code to RoutingReceiverOne.Main:

```
AmqpMessagingService messagingService = new AmqpMessagingService();
IConnection connection = messagingService.GetRabbitMqConnection();
IModel model = connection.CreateModel();
messagingService.ReceiveRoutingMessageReceiverOne(model);
```

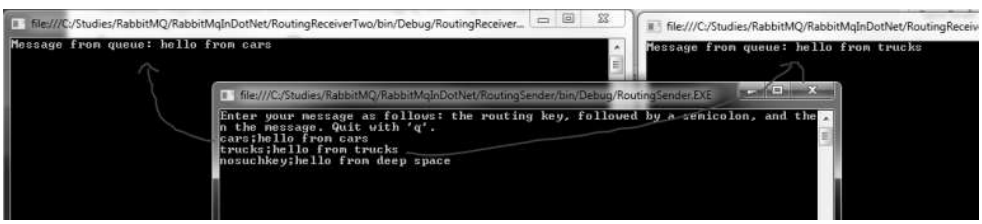
…and the following to RoutingReceiverTwo.Main:

```
AmqpMessagingService messagingService = new AmqpMessagingService();
IConnection connection = messagingService.GetRabbitMqConnection();
IModel model = connection.CreateModel();
messagingService.ReceiveRoutingMessageReceiverTwo(model);
```

Follow these steps to run the demo:

1. Make sure RoutingSender is the start up project and then start the application
2. Start RoutingReceiverOne by right-clicking it in VS, Debug, Start new instance
3. Start RoutingReceiverTwo the same way
4. Now you should have 3 console screens up and running

Start sending messages from the sender. Make sure you use the ';' delimiter to indicate the routing key and the message. The messages should be routed correctly:



(https://dotnetcodr.files.wordpress.com/2014/03/routingmepconsolewindows.png)

This wasn't too difficult, right? Messages with no matching routing key will be discarded by RabbitMQ.

Topics

The Topic MEP is similar to Routing. The sender sends a message to an exchange with a routing key attached. The message will be forwarded to queues with a matching **expression**. The routing key can include special characters:

- '*' to replace one word
- '#' to replace 0 or more words

The purpose of this pattern is that the receiver can specify a pattern, sort of like a regular expression, as the routing key it is interested in: #world#, cars* etc. Then the sender sends a message with a routing key "world news" and then another one with a routing key "the end of the world" and the queue will receive both messages. If there are no queues with a matching routing key pattern then the message is discarded.

Let's set up the exchange and the queues. In this demo we'll have three queues listening on 3 different routing key patterns. Add the following 4 private fields to AmqpMessagingService.cs:

```
1  private string _topicsExchange = "TopicsExchange";
2  private string _topicsQueueOne = "TopicsQueueOne";
3  private string _topicsQueueTwo = "TopicsQueueTwo";
4  private string _topicsQueueThree = "TopicsQueueThree";
```

Insert the following method that will set up the exchange and the queues:

```
 1  public void SetUpExchangeAndQueuesForTopicsDemo(IModel model)
 2  {
 3      model.ExchangeDeclare(_topicsExchange, ExchangeType.Topic, true);
 4      model.QueueDeclare(_topicsQueueOne, true, false, false, null);
 5      model.QueueDeclare(_topicsQueueTwo, true, false, false, null);
 6      model.QueueDeclare(_topicsQueueThree, true, false, false, null);
 7      model.QueueBind(_topicsQueueOne, _topicsExchange, "*.world.*");
 8      model.QueueBind(_topicsQueueTwo, _topicsExchange, "#.world.#");
 9      model.QueueBind(_topicsQueueThree, _topicsExchange, "#.world");
10  }
```

You can set up multiple bindings with different keywords as I showed above. This technique allows for some very refined searches among the routing keys.

We'll investigate how those different wildcard characters behave differently.

Insert a new Console application called TopicsSender. Add references to RabbitMQ NuGet and RabbitMqService. The following code in Main will call SetUpExchangeAndQueuesForTopicsDemo:

```
1  AmqpMessagingService messagingService = new AmqpMessagingService();
2  IConnection connection = messagingService.GetRabbitMqConnection();
3  IModel model = connection.CreateModel();
4  messagingService.SetUpExchangeAndQueuesForTopicsDemo(model);
```

Set TopicsSender as the start up project and run the application. Check in the RabbitMQ management UI that all queues, the exchange and the bindings have been set up properly. Comment out the call to messagingService.SetUpExchangeAndQueuesForTopicsDemo. Instead add a call to the following private method:

```
 1  private static void RunTopicsDemo(IModel model, AmqpMessagingService messagingService)
 2  {
 3      Console.WriteLine("Enter your message as follows: the routing key, followed by a semicolon, and t
 4      while (true)
 5      {
 6          string fullEntry = Console.ReadLine();
 7          string[] parts = fullEntry.Split(new char[] { ';' }, StringSplitOptions.RemoveEmptyEntries);
 8          string key = parts[0];
 9          string message = parts[1];
10          if (message.ToLower() == "q") break;
11          messagingService.SendTopicsMessage(message, key, model);
12      }
13  }
```

…where SendTopicsMessage looks like this in AmqpMessagingService.cs:

```
public void SendTopicsMessage(string message, string routingKey, IModel model)
{
    IBasicProperties basicProperties = model.CreateBasicProperties();
    basicProperties.SetPersistent(_durable);
    byte[] messageBytes = Encoding.UTF8.GetBytes(message);
    model.BasicPublish(_topicsExchange, routingKey, basicProperties, messageBytes);
}
```

Let's set up the missing pieces. We're now so knowledgeable on RabbitMQ in .NET that this part almost feels boring, right? Insert 3 new Console apps: TopicsReceiverOne, TopicsReceiverTwo, TopicsReceiverThree. Add references to the RabbitMQ NuGet package and the RabbitMqService library to all three. Add the following methods to AmqpMessagingService.cs which will handle the reception of the messages for each receiver:

```
public void ReceiveTopicMessageReceiverOne(IModel model)
{
    model.BasicQos(0, 1, false);
    Subscription subscription = new Subscription(model, _topicsQueueOne, false);
    while (true)
    {
        BasicDeliverEventArgs deliveryArguments = subscription.Next();
        String message = Encoding.UTF8.GetString(deliveryArguments.Body);
        Console.WriteLine("Message from queue: {0}", message);
        subscription.Ack(deliveryArguments);
    }
}

public void ReceiveTopicMessageReceiverTwo(IModel model)
{
    model.BasicQos(0, 1, false);
    Subscription subscription = new Subscription(model, _topicsQueueTwo, false);
    while (true)
    {
        BasicDeliverEventArgs deliveryArguments = subscription.Next();
        String message = Encoding.UTF8.GetString(deliveryArguments.Body);
        Console.WriteLine("Message from queue: {0}", message);
        subscription.Ack(deliveryArguments);
    }
}

public void ReceiveTopicMessageReceiverThree(IModel model)
{
    model.BasicQos(0, 1, false);
    Subscription subscription = new Subscription(model, _topicsQueueThree, false);
    while (true)
    {
        BasicDeliverEventArgs deliveryArguments = subscription.Next();
        String message = Encoding.UTF8.GetString(deliveryArguments.Body);
        Console.WriteLine("Message from queue: {0}", message);
        subscription.Ack(deliveryArguments);
    }
}
```

All that should look familiar by now, so I won't go into any details. In TopicsReceiverOne.Main add the following:

```
AmqpMessagingService messagingService = new AmqpMessagingService();
IConnection connection = messagingService.GetRabbitMqConnection();
IModel model = connection.CreateModel();
messagingService.ReceiveTopicMessageReceiverOne(model);
```

…in TopicsReceiverTwo.Main…:

```
AmqpMessagingService messagingService = new AmqpMessagingService();
IConnection connection = messagingService.GetRabbitMqConnection();
IModel model = connection.CreateModel();
messagingService.ReceiveTopicMessageReceiverTwo(model);
```

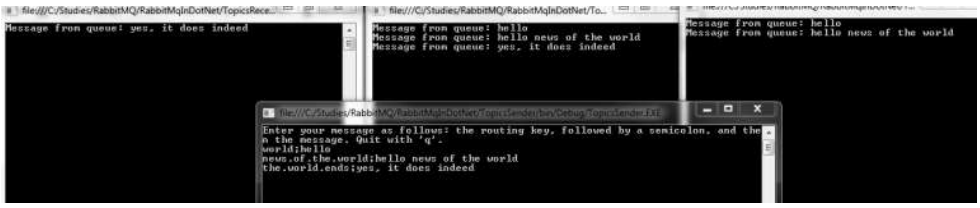
…and in TopicsReceiverThree.Main…:

```
AmqpMessagingService messagingService = new AmqpMessagingService();
IConnection connection = messagingService.GetRabbitMqConnection();
IModel model = connection.CreateModel();
messagingService.ReceiveTopicMessageReceiverThree(model);
```

To run the demo:

1. Make sure that TopicsSender is the start up project and start the application
2. Run the 3 topic receivers following the same technique as above (Debug, Run new instance)
3. You should have 4 console windows up and running on your screen

Start sending messages to RabbitMQ. Take care when typing the routing key and the message. Delimit the routing key sections with a '.':



(https://dotnetcodr.files.wordpress.com/2014/03/topicsmepconsole.png)

Explanation:

- 'world': received by receiver 2 and 3 as the topic routing keys #.world and #.world.# match it. Topic key *.world.* is no match as the '*' replaces one word
- 'news.of.the.world': same as above
-
- 'the.world.ends': matches receiver 1 and 2, but not 3 as there's a word after 'world.' in the routing key

It can be a bit confusing with the topic keys and matches at first but the Topics pattern is not much different from the routing one.

Read the next part of this series here (https://dotnetcodr.com/2014/05/12/messaging-with-rabbitmq-and-net-c-part-5-headers-and-scattergather/).

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

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

One Response to *Messaging with RabbitMQ and .NET C# part 4: routing and topics*

Skr says:
June 23, 2020 at 7:51 pm
hi,
Greeting

My requirement need to get missed messages, used the "x-recent-history" plugin with Key
but the sub not working with routing KEY , receiving all the messages within the "pos.hist".

Dictionary args = new Dictionary();
args.Add("x-recent-history-length", 60);
channel.ExchangeDeclare("pos.hist", "x-recent-history",false,true, args);

sorry for my english.

how to get "x-recent-history" with routing key.

Reply