Messaging with RabbitMQ and .NET review part 9: headers

AUGUST 29, 2016 LEAVE A COMMENT (HTTPS://DOTNETCODR.COM/2016/08/29/MESSAGING-WITH-RABBITMQ-AND-NET-REVIEW-PART-9-HEADERS/#RESPOND)

Introduction

In the previous post (https://dotnetcodr.com/2016/08/25/messaging-with-rabbitmq-and-net-review-part-8-routing-and-topics/) we looked at two ways to filter messages from an exchange to one or more queues bound to it. Routing keys provide a simple mechanism where the routing key of the message, such as "asia" is forwarded to all queues that also have a routing key "asia". In other words the filtering is based on a direct string comparison. The Topic message exchange pattern is a more sophisticated variant where the '*' and '#' placeholders let you fine-tune the binding rule between an exchange and a queue. We can also bind multiple queues to the same exchange with varying routing keys.

In this post we'll look at one more message filtering technique called headers which is in fact very similar to the topic MEP. The headers MEP offers a very fine-grained way to set up the binding rules between the exchange and the queue(s).

Headers

The Headers exchange pattern is very similar to Topics we saw in the previous part of this series. The sender sends a message to RabbitMq. The message is routed based on the header values. All queues with a matching key will receive the message. We dedicate an exchange to deliver the messages but the routing key will be ignored as it is the headers that will be the basis for the match. We can specify more than one header and a rule that says if all headers or just one of them must match using the "x-match" property which can have 2 values: "any" or "all". The default value of this property is "all" so all headers must match for a queue to receive a message.

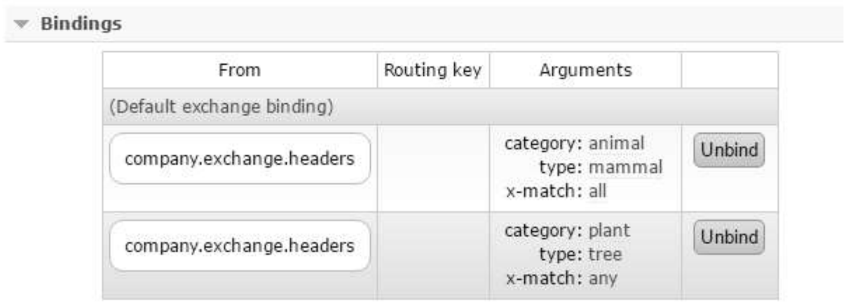
We'll be using the same demo console application as throughout this course to demonstrate the concept. Here's the code that sets up the exchange, the queue and the bindings and also sends several messages:

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

namespace RabbitMqNetTests
{
    class Program
    {
        static void Main(string[] args)
        {
            SetUpHeadersExchange();
        }
        private static void SetUpHeadersExchange()
        {
            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.ExchangeDeclare("company.exchange.headers", ExchangeType.Headers, true, false, n
            channel.QueueDeclare("company.queue.headers", true, false, false, null);
            Dictionary<string, object> headerOptionsWithAll = new Dictionary<string, object>();
            headerOptionsWithAll.Add("x-match", "all");
            headerOptionsWithAll.Add("category", "animal");
            headerOptionsWithAll.Add("type", "mammal");

            channel.QueueBind("company.queue.headers", "company.exchange.headers", "", headerOptions

            Dictionary<string, object> headerOptionsWithAny = new Dictionary<string, object>();
            headerOptionsWithAny.Add("x-match", "any");
            headerOptionsWithAny.Add("category", "plant");
            headerOptionsWithAny.Add("type", "tree");

            channel.QueueBind("company.queue.headers", "company.exchange.headers", "", headerOptions

            IBasicProperties properties = channel.CreateBasicProperties();
            Dictionary<string, object> messageHeaders = new Dictionary<string, object>();
            messageHeaders.Add("category", "animal");
            messageHeaders.Add("type", "insect");
            properties.Headers = messageHeaders;
            PublicationAddress address = new PublicationAddress(ExchangeType.Headers, "company.excha
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("Hello from the world o

            properties = channel.CreateBasicProperties();
            messageHeaders = new Dictionary<string, object>();
            messageHeaders.Add("category", "animal");
            messageHeaders.Add("type", "mammal");
            messageHeaders.Add("mood", "awesome");
            properties.Headers = messageHeaders;
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("Hello from the world o

            properties = channel.CreateBasicProperties();
            messageHeaders = new Dictionary<string, object>();
            messageHeaders.Add("category", "animal");
            messageHeaders.Add("type", "mammal");
            properties.Headers = messageHeaders;
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("Hello from the world o

            properties = channel.CreateBasicProperties();
```

```
                messageHeaders = new Dictionary<string, object>();
                messageHeaders.Add("category", "animal");
                properties.Headers = messageHeaders;
                channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("Hello from the world o

                properties = channel.CreateBasicProperties();
                messageHeaders = new Dictionary<string, object>();
                messageHeaders.Add("category", "fungi");
                messageHeaders.Add("type", "champignon");
                properties.Headers = messageHeaders;
                channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("Hello from the world o

                properties = channel.CreateBasicProperties();
                messageHeaders = new Dictionary<string, object>();
                messageHeaders.Add("category", "plant");
                properties.Headers = messageHeaders;
                channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("Hello from the world o

                properties = channel.CreateBasicProperties();
                messageHeaders = new Dictionary<string, object>();
                messageHeaders.Add("category", "plant");
                messageHeaders.Add("type", "tree");
                properties.Headers = messageHeaders;
                channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("Hello from the world o

                properties = channel.CreateBasicProperties();
                messageHeaders = new Dictionary<string, object>();
                messageHeaders.Add("mood", "sad");
                messageHeaders.Add("type", "tree");
                properties.Headers = messageHeaders;
                channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("Hello from the world o

                channel.Close();
                connection.Close();
            }
        }
    }
```

We set up the exchange called "company.exchange.headers" and declare it as type Headers. We then create a queue called "company.queue.headers". We supply the QueueBind with a dictionary of headers. In fact we have two bindings: one where all headers must match between the message and the queue and one where the match is based on any of the headers. Then we publish 8 messages in total. Each message has its own set of headers declared by the Headers property of IBasicProperties. Note that the code binds a single queue with different header settings to the same exchange. It's equally feasible to create 2 queues where each queue will have its own listener and set of headers.

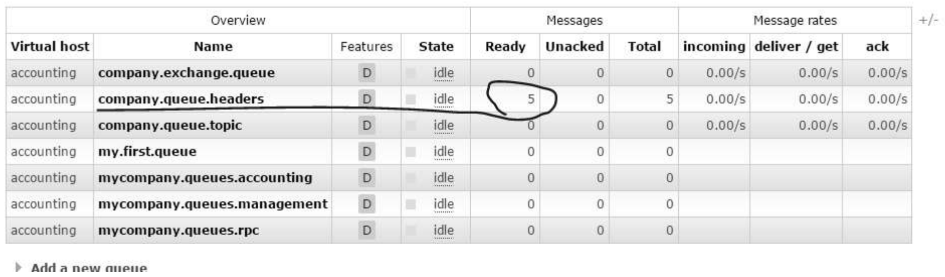
If you run the code then you'll see the bindings in the RabbitMq console:



From	Routing key	Arguments	
(Default exchange binding)			
company.exchange.headers		category: animal type: mammal x-match: all	Unbind
company.exchange.headers		category: plant type: tree x-match: any	Unbind

(https://dotnetcodr.files.wordpress.com/2016/08/queue-bound-to-rabbitmq-exchange-through-various-headers.png)

The management UI is also suggesting that 5 of the 8 messages were routed to the queue:



Overview				Messages			Message rates		
Virtual host	Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
accounting	company.exchange.queue	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
accounting	company.queue.headers	D	idle	5	0	5	0.00/s	0.00/s	0.00/s
accounting	company.queue.topic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
accounting	my.first.queue	D	idle	0	0	0			
accounting	mycompany.queues.accounting	D	idle	0	0	0			
accounting	mycompany.queues.management	D	idle	0	0	0			
accounting	mycompany.queues.rpc	D	idle	0	0	0			

(https://dotnetcodr.files.wordpress.com/2016/08/five-messages-were-routed-to-rabbitmq-headers-queue.png)

The consumer's code is the same as what we saw before. The only difference is the queue name and that we also print each header key and value. Note that each header value is transmitted as type object which must be cast to a byte array. The byte array can be then converted into a string:

```
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
    {
        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();
            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 wind
                Debug.WriteLine(string.Concat("Message received from the exchange ", basicDeliveryEve
                Debug.WriteLine(string.Concat("Message: ", Encoding.UTF8.GetString(basicDeliveryEvent
                StringBuilder headersBuilder = new StringBuilder();
                headersBuilder.Append("Headers: ").Append(Environment.NewLine);
                foreach (var kvp in basicProperties.Headers)
                {
                    headersBuilder.Append(kvp.Key).Append(": ").Append(Encoding.UTF8.GetString(kvp.Va
                }
                Debug.WriteLine(headersBuilder.ToString());
                channel.BasicAck(basicDeliveryEventArgs.DeliveryTag, false);
            };

            channel.BasicConsume("company.queue.headers", false, eventingBasicConsumer);
        }
    }
}
```

…and here are the 5 messages routed to the queue:

```
Message received from the exchange company.exchange.headers
Message: Hello from the world of awesome mammals
Headers:
category: animal
type: mammal
mood: awesome

Message received from the exchange company.exchange.headers
Message: Hello from the world of mammals
Headers:
```

```
category: animal
type: mammal

Message received from the exchange company.exchange.headers
Message: Hello from the world of plants
Headers:
category: plant

Message received from the exchange company.exchange.headers
Message: Hello from the world of trees
Headers:
category: plant
type: tree

Message received from the exchange company.exchange.headers
Message: Hello from the world of sad trees
Headers:
mood: sad
type: tree
```

The messages from “insects”, “fungi” and “animals” were discarded as their routing patterns didn’t match any of the bindings we set up above.

There’s one more message exchange pattern we haven’t looked at so far: scatter/gather. We’ll do that in the next post (https://dotnetcodr.com/2016/09/01/messaging-with-rabbitmq-and-net-review-part-10-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.