RabbitMQ for Windows: Topic Exchanges

18 May, 2012. It was a Friday.

Posts In This Series

This is the seventh installment to the series: RabbitMQ for Windows. In the last installment, we walked through creating a fanout exchange example. In this installment, we'll be walking through a topic exchange example.

Topic exchanges are similar to direct exchanges in that they use a routing key to determine which queue a message should be delivered to, but they differ in that they provide the ability to match on portions of a routing key. When publishing to a topic exchange, a routing key consisting of multiple words separated by periods (e.g. "word1.word2.word3") will be matched against a pattern supplied by the binding queue. Patterns may contain an asterisk ("*") to match a word in a specific segment or a hash ("#") to match zero or more words. As discussed earlier in the series, the topic exchange type can be useful for directing messages based on multiple categories or for routing messages originating from multiple sources.

To demonstrate topic exchanges, we'll return to our logging example, but this time we'll subscribe to a subset of the messages being published to demonstrate the flexibility of how routing keys are used by topic exchanges. For this example, we'll be modeling a scenario where a company may have multiple client installations, each of which may be used to service different sectors of a company's business model (e.g. Business or Personal sectors). We'll use a routing key that specifies the sector and subscribe to messages published for the Personal sector only.

As with our previous examples, we'll keep things simple by creating console applications for a Producer and a Consumer. Let's start by creating the Producer app and establishing a connection using the default settings:

```
using RabbitMQ.Client;

namespace Producer
{
  class Program
  {
    const long ClientId = 10843;

    static void Main(string[] args)
    {
      var connectionFactory = new ConnectionFactory();
```

```
      IConnection connection = connectionFactory.CreateConnection();
    }
  }
}
```

Rather than just publishing messages directly from the Main() method as with our first logging example, let's create a separate logger object this time. Here the logger interface and implementation we'll be using:

```
interface ILogger
  {
    void Write(Sector sector, string entry, TraceEventType traceEventType);
  }

  class RabbitLogger : ILogger, IDisposable
  {
    readonly long _clientId;
    readonly IModel _channel;
    bool _disposed;

    public RabbitLogger(IConnection connection, long clientId)
    {
      _clientId = clientId;
      _channel = connection.CreateModel();
      _channel.ExchangeDeclare("direct-exchange-example", ExchangeType.Topic, false, true, null);
    }

    public void Dispose()
    {
```

```
      if (!_disposed)
      {
        if (_channel != null && _channel.IsOpen)
        {
          _channel.Close();
        }
      }
      GC.SuppressFinalize(this);
    }

    public void Write(Sector sector, string entry, TraceEventType traceEventType)
    {
      byte[] message = Encoding.UTF8.GetBytes(entry);
      string routingKey = string.Format("{0}.{1}.{2}", _clientId, sector.ToString(), traceEventType.ToS
      _channel.BasicPublish("topic-exchange-example", routingKey, null, message);
    }

    ~RabbitLogger()
    {
      Dispose();
    }
  }
```

In addition to an open IConnection, our RabbitLogger class is instantiated with a client Id. We use this as part of the routing key. Since each log can vary by sector, we pass a Sector enum as part of the Write() method. Here's our Sector enum:

```
public enum Sector
  {
    Personal,
```

```
    Business
  }
```

Returning to our Main() method, we now need to instantiate our RabbitLogger and log messages with differing sectors. As as way to ensure our client has an opportunity to subscribe to our messages and to help emulate a continual stream of log messages being published, let's use the logger to publish a series of log messages every second for 10 seconds:

```
TimeSpan time = TimeSpan.FromSeconds(10);
      var stopwatch = new Stopwatch();
      Console.WriteLine("Running for {0} seconds", time.ToString("ss"));
      stopwatch.Start();

      while (stopwatch.Elapsed < time)
      {
        using (var logger = new RabbitLogger(connection, ClientId))
        {
          Console.Write("Time to complete: {0} seconds\r", (time - stopwatch.Elapsed).ToString("ss"));
          logger.Write(Sector.Personal, "This is an information message", TraceEventType.Information);
          logger.Write(Sector.Business, "This is an warning message", TraceEventType.Warning);
          logger.Write(Sector.Business, "This is an error message", TraceEventType.Error);
          Thread.Sleep(1000);
        }
      }
```

This code prints out the time remaining just to give us a little feedback on the publishing progress. Finally, we'll close our our connection and prompt the user to exit the console application:

```
connection.Close();
      Console.Write("                              \r");
```

```
      Console.WriteLine("Press any key to exit");
      Console.ReadKey();
```

Here's the full Producer listing:

```
using System;
using System.Diagnostics;
using System.Text;
using System.Threading;
using RabbitMQ.Client;


namespace Producer
{
  public enum Sector
  {
    Personal,
    Business
  }

  interface ILogger
  {
    void Write(Sector sector, string entry, TraceEventType traceEventType);
  }

  class RabbitLogger : ILogger, IDisposable
  {
    readonly long _clientId;
    readonly IModel _channel;
```

```
bool _disposed;

public RabbitLogger(IConnection connection, long clientId)
{
  _clientId = clientId;
  _channel = connection.CreateModel();
  _channel.ExchangeDeclare("direct-exchange-example", ExchangeType.Topic, false, true, null);
}

public void Dispose()
{
  if (!_disposed)

  {
    if (_channel != null && _channel.IsOpen)
    {
      _channel.Close();
    }
  }
  GC.SuppressFinalize(this);
}

public void Write(Sector sector, string entry, TraceEventType traceEventType)
{
  byte[] message = Encoding.UTF8.GetBytes(entry);
  string routingKey = string.Format("{0}.{1}.{2}", _clientId, sector.ToString(), traceEventType.ToS
  _channel.BasicPublish("topic-exchange-example", routingKey, null, message);
}

~RabbitLogger()
{
```

```
      Dispose();
    }
  }

  class Program
  {
    const long ClientId = 10843;

    static void Main(string[] args)
    {
      var connectionFactory = new ConnectionFactory();
      IConnection connection = connectionFactory.CreateConnection();


      TimeSpan time = TimeSpan.FromSeconds(10);
      var stopwatch = new Stopwatch();
      Console.WriteLine("Running for {0} seconds", time.ToString("ss"));
      stopwatch.Start();

      while (stopwatch.Elapsed < time)
      {
        using (var logger = new RabbitLogger(connection, ClientId))
        {
          Console.Write("Time to complete: {0} seconds\r", (time - stopwatch.Elapsed).ToString("ss"));
          logger.Write(Sector.Personal, "This is an information message", TraceEventType.Information);
          logger.Write(Sector.Business, "This is an warning message", TraceEventType.Warning);
          logger.Write(Sector.Business, "This is an error message", TraceEventType.Error);
          Thread.Sleep(1000);
        }
      }
```

```
      connection.Close();
      Console.Write("                              \r");
      Console.WriteLine("Press any key to exit");
      Console.ReadKey();
    }
  }
}
```

For our Consumer app, we'll pretty much be using the same code as with our fanout exchange example, but we'll need to change the exchange type along with the exchange and queue names. Additionally, we also need to provide a routing key that registers for logs in the Personal sector only. The messages published by the Producer will be in the form: [client Id].[sector].[log severity], so we can use a routing key of "*.Personal.*" (or alternately "*.Personal.#"). Here's the full Consumer listing:

```
using System;
using System.IO;
using System.Text;
using RabbitMQ.Client;
using RabbitMQ.Client.Events;

namespace Consumer
{
  class Program
  {
    static void Main(string[] args)
    {
      var connectionFactory = new ConnectionFactory();
      IConnection connection = connectionFactory.CreateConnection();
```

```
      IModel channel = connection.CreateModel();

      channel.ExchangeDeclare("topic-exchange-example", ExchangeType.Topic, false, true, null);
      channel.QueueDeclare("log", false, false, true, null);
      channel.QueueBind("log", "topic-exchange-example", "*.Personal.*");

      var consumer = new QueueingBasicConsumer(channel);
      channel.BasicConsume("log", true, consumer);

      while (true)
      {
        try
        {
          var eventArgs = (BasicDeliverEventArgs) consumer.Queue.Dequeue();
          string message = Encoding.UTF8.GetString(eventArgs.Body);
          Console.WriteLine(string.Format("{0} - {1}", eventArgs.RoutingKey, message));
        }
        catch (EndOfStreamException)
        {
          // The consumer was cancelled, the model closed, or the connection went away.
          break;
        }
      }

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

Setting the solution to run both the Producer and Consumer on startup, we should see similar output to the following listings:

Producer

```
Running for 10 seconds
Time to complete: 06 seconds
```

Consumer

```
10843.Personal.Information - This is an information message
10843.Personal.Information - This is an information message
10843.Personal.Information - This is an information message
10843.Personal.Information - This is an information message
10843.Personal.Information - This is an information message
10843.Personal.Information - This is an information message
10843.Personal.Information - This is an information message
```

This concludes our topic exchange example. Next time, we'll walk through an example using the final exchange type: Header Exchanges.

← RabbitMQ for Windows: Fanout Exchanges

RabbitMQ for Windows: Headers Exchanges →

Kevin Reilly • 9 years ago

Hi Derek - Greate series on RabbitMQ. One question regarding the Topic Exchange code. Can you describe why two different exchanges need be used when creating the Producer?

derekgreer Mod Kevin Reilly • 9 years ago

That's a cut-n-paste error. All references to the exchange in this example should read "topic-exchange-example". I didn't notice the error because, while the producer declares the incorrect exchange name, it does publish to the correct exchange name which the client also declares.

Recent Author Posts

- Hello, React! - A Beginner's Setup Tutorial
- Exploring TypeScript
- Git on Windows: Whence Cometh Configuration
- Separation of Concerns: Application Builds & Continuous Integration
- Survey of Entity Framework Unit of Work Patterns
- Introducing NUnit.Specifications
- Being Agile
- Expected Objects Custom Comparisons
- RabbitBus: An Example
- Adventures in Debugging: The NHibernate 'don't flush the Session' Error

Recent Site Posts

- Domain-Driven Refactoring: Defactoring and Pushing Behavior Down
- Domain-Driven Refactoring: Extracting Domain Services
- Domain-Driven Refactoring: Long Methods
- Domain-Driven Refactoring: Procedural Beginnings
- Domain-Driven Refactoring: Intro
- Local Development with Azure Service Bus
- Taming the WSL 2 Resource Monster
- Crossing the Generics Divide
- OpenTelemetry 1.0 Extensions Released
- Choosing a ServiceLifetime

Authors

- Andrew Siemer
- Chad Myers
- Chris Missal
- Chris Patterson
- Derek Greer
- Derik Whittaker
- Eric Anderson
- Eric Hexter
- Gabriel Schenker
- Gregory Long
- Hugo Bonacci
- James Gregory

- Jimmy Bogard
- John Teague
- Josh Arnold
- Joshua Flanagan
- Joshua Lockwood
- Keith Dahlby
- Matt Hinze
- Patrick Lioi
- Rod Paddock
- Ryan Rauh
- Ryan Svihla
- Scott Densmore
- Sean Biefeld
- Sean Chambers
- Sharon Cichelli
- Steve Donie