RabbitMQ for Windows: Fanout Exchanges

16 May, 2012. It was a Wednesday.

Posts In This Series

- RabbitMQ for Windows: Introduction
- RabbitMQ for Windows: Building Your First Application
- RabbitMQ for Windows: Hello World Review
- RabbitMQ for Windows: Exchange Types
- RabbitMQ for Windows: Direct Exchanges
- RabbitMQ for Windows: Fanout Exchanges
- RabbitMQ for Windows: Topic Exchanges
- RabbitMQ for Windows: Headers Exchanges

This is the sixth installment to the series: RabbitMQ for Windows. In the last installment, we walked through creating a direct exchange example and introduced the push API. In this installment, we'll walk through a fanout exchange example.

As discussed earlier in the series, the fanout exchange type is useful for facilitating the publish-subscribe pattern. When we publish a message to a fanout exchange, the message is delivered indiscriminately to all bound queues. With the Direct, Topic, and Headers exchange types, a criteria is used by a routing algorithm taking the form of a routing key or a collection of message headers depending on the exchange type in question. A routing key or a collection of message headers may also be specified with the fanout exchange which will be delivered as part of the message's metadata, but they will not be used as a filter in determining which queue receives a published message.

To demonstrate the fanout exchange, we'll use a stock ticker example. In the previous example, logs were routed to queues based upon a matching routing key (an empty string in the logging example's case). In this example, we'd like our messages to be delivered to all bound queues regardless of qualification.

Similar to the previous example, we'll create a Producer console application which periodically publishes stock quote messages and a Consumer console application which displays the message to the console.

We'll start our Producer app as before by establishing a connection using the default settings, creating the connection, and creating a channel:

```
namespace Producer
{
  class Program
  {
    static volatile bool _cancelling;

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

Next, we need to declare an exchange of type “fanout”. We’ll name our new exchange “fanout-exchange-example”:

```
channel.ExchangeDeclare("fanout-exchange-example", ExchangeType.Fanout, false, true, null);
```

To publish the stock messages periodically, we’ll call a PublishQuotes() method with the provided channel and run it on a background thread:

```
var thread = new Thread(() => PublishQuotes(channel));
thread.Start();
```

Next, we’ll provide a way to exit the application by prompting the user to enter ‘x’ and use a simple Boolean to signal the background thread when to exit:

```
Console.WriteLine("Press 'x' to exit");
var input = (char) Console.Read();
_cancelling = true;
```

Lastly, we need to close the channel and connection:

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

For our PublishQuotes() method, well iterate over a set of stock symbols, retrieve the stock information for each symbol, and publish a simple string-based message in the form [symbol]:[price]:

```
static void PublishQuotes(IModel channel)
{
  while (true)
  {
    if (_cancelling) return;
    IEnumerable<string> quotes = FetchStockQuotes(new[] {"GOOG", "HD", "MCD"});
    foreach (string quote in quotes)
    {
      byte[] message = Encoding.UTF8.GetBytes(quote);
      channel.BasicPublish("fanout-exchange-example", "", null, message);
    }
    Thread.Sleep(5000);
  }
}
```

To implement the FetchStockQuotes() method, we'll use the Yahoo Finance API which entails retrieving an XML-based list of stock quotes and parsing out the bit of information we're interested in for our example:

```
static IEnumerable<string> FetchStockQuotes(string[] symbols)
{
  var quotes = new List<string>();

  string url = string.Format("http://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20yahoo.fina
      String.Join("%2C", symbols.Select(s => "%22" + s + "%22")));
  var wc = new WebClient {Proxy = WebRequest.DefaultWebProxy};
  var ms = new MemoryStream(wc.DownloadData(url));
  var reader = new XmlTextReader(ms);
  XDocument doc = XDocument.Load(reader);
  XElement results = doc.Root.Element("results");

  foreach (string symbol in symbols)
  {
    XElement q = results.Elements("quote").First(w => w.Attribute("symbol").Value == symbol);
    quotes.Add(symbol + ":" + q.Element("AskRealtime").Value);
  }

  return quotes;
}
```

Here is the complete Producer listing:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading;
using System.Xml;
using System.Xml.Linq;
using RabbitMQ.Client;

namespace Producer
{
  class Program
  {
    static volatile bool _cancelling;

    static void Main(string[] args)
    {
```

```
    var connectionFactory = new ConnectionFactory();
    IConnection connection = connectionFactory.CreateConnection();
    IModel channel = connection.CreateModel();
    channel.ExchangeDeclare("fanout-exchange-example", ExchangeType.Fanout, false, true, null);

    var thread = new Thread(() => PublishQuotes(channel));
    thread.Start();

    Console.WriteLine("Press 'x' to exit");
    var input = (char) Console.Read();
    _cancelling = true;

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

  static void PublishQuotes(IModel channel)
  {
    while (true)
    {
      if (_cancelling) return;
      IEnumerable<string> quotes = FetchStockQuotes(new[] {"GOOG", "HD", "MCD"});
      foreach (string quote in quotes)
      {
        byte[] message = Encoding.UTF8.GetBytes(quote);
        channel.BasicPublish("fanout-exchange-example", "", null, message);
      }
      Thread.Sleep(5000);
    }
  }
```

```
    static IEnumerable<string> FetchStockQuotes(string[] symbols)
    {
      var quotes = new List<string>();

      string url = string.Format("http://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20yahoo.
          String.Join("%2C", symbols.Select(s => "%22" + s + "%22")));
      var wc = new WebClient {Proxy = WebRequest.DefaultWebProxy};
      var ms = new MemoryStream(wc.DownloadData(url));
      var reader = new XmlTextReader(ms);
      XDocument doc = XDocument.Load(reader);
      XElement results = doc.Root.Element("results");

      foreach (string symbol in symbols)
      {
        XElement q = results.Elements("quote").First(w => w.Attribute("symbol").Value == symbol);
        quotes.Add(symbol + ":" + q.Element("AskRealtime").Value);
      }

      return quotes;
    }
  }
}
```

Our Consumer application will be similar to the one used in our logging example, but we'll change the exchange name, queue name, and exchange type and put the processing of the messages within a while loop to continually display our any updates to our stock prices. Here's the full listing for our Consumer app:

```
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("fanout-exchange-example", ExchangeType.Fanout, false, true, null);
      channel.QueueDeclare("quotes", false, false, true, null);
      channel.QueueBind("quotes", "fanout-exchange-example", "");

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

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

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

Setting our solution startup projects to run both the Producer and Consumer apps together, we should see messages similar to the following for the Consumer output:

```
GOOG:611.62
HD:48.66
MCD:91.06
GOOG:611.58
HD:48.66
MCD:91.06
```

To show our queue would receive messages published to the fanout exchange regardless of the routing key value, we can change the value of the routing key to “anything”:

```
channel.QueueBind("quotes", "fanout-exchange-example", "anything");
```

Running the application again shows the same values:

```
GOOG:611.62
HD:48.66
MCD:91.06
GOOG:611.58
HD:48.66
MCD:91.06
```

That concludes our fanout exchange example. Next time, we'll take a look at the topic exchange type.

← RabbitMQ for Windows: Direct Exchanges

RabbitMQ for Windows: Topic Exchanges →

Comments for this thread are now closed ×

3 Comments Los Techies Disqus' Privacy Policy 1 Login

Recommend Tweet Share Sort by Best

Patty • 8 years ago

This is really helpful! You do have one typo above that you might want to correct:

ExchangeDeclare("direct-exchange-example", ExchangeType.Fanout,...

should be:

ExchangeDeclare("fanout-exchange-example", ExchangeType.Fanout,

• Share ›

derekgreer Mod Patty • 8 years ago

Thanks, Patty. This has been updated.

• Share ›

Mike Hadlow • 9 years ago

Great series Derek. Can I make a shameless plug for my simplified and opinionated RabbitMQ API for .NET, EasyNetQ ;)

https://github.com/mikehadl...

• Share ›

Subscribe Add Disqus to your siteAdd DisqusAdd Do Not Sell My Data

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