# File cần thiết.

* <http://www.rabbitmq.com/releases/rabbitmq-dotnet-client/v3.4.4/rabbitmq-dotnet-client-3.4.4-dotnet-3.5.zip>
* Sau khi tải file trên về có hai file cần thiết là **RabbitMQ.Client.dll** và **RabbitMQ.ServiceModel.dll** được đặt trong thư mục Client.

# Ký hiệu.



P: Producing, đóng vai trò gửi.

C: Consumer đóng vai trò nhận phần lần consumer chỉ đợi để nhận tin nhắn.

Q: Queue

X: Exchange

# VD1:



Ở ví dụ đầu tiên, ta sẻ viết ứng dụng để P gửi trực tiếp tin nhắn lên queue và trữ tại đó, khi C kết nối với Queue thì sẻ lần lượt lấy tin nhắn và in lên console.

**Code của P:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RabbitMQ.Client;

namespace RMQ\_Demo\_01

{

class Send

{

static void Main(string[] args)

{

// Tiến hành kết nối đến server, ở đây là localhost

var factory = new ConnectionFactory() { HostName = "localhost" };

using (var connection = factory.CreateConnection())

{

// Tiến hành tạo một channel, phần lớn các lệnh trong API sẻ dùng trên channel

using (var channel = connection.CreateModel())

{

// Để có thể gửi tin nhắn thì ta sẻ tạo một queue để gửi, tên queue ở đây là SomeQueue

/\* [x] 1: tên của queue

\* [x] 2: "durable": Nghĩa là cái queue này sẻ không bị mất nếu server

\* bị tắt hoặc khởi động lại, và khôi phục lại tin nhắn, tuy nhiên, chỉ tin nhắn nào có

\* gắn nhãn là presistent mới được khôi phục.

\* [x] 3: "exclusive": Chỉ được dùng bởi một connection và sẻ bị xóa nếu kết

\* nối đó đóng.

\* [x] 4: autoDelete: Queue này sẻ tự động được delete khi mà không còn

\* consumer nào kết nối đến nữa

\* [x] 5: ;Argument some brokers use it to implement additional features like message TTL;

\*/

channel.QueueDeclare("SomeQueue", false, false, false, null);

while (true)

{

Console.Write(" [x] Enter Message: ");

string message = Console.ReadLine();

// Chuyển tin nhánh thành mãng byte

var body = Encoding.UTF8.GetBytes(message);

/\* Đưa tin nhắn nào vào queue có tên là SomeQueue

\* [x]1: Exchange tạm thời chưa cần đến.

\* [x]2: tên của queue, ta đã khai báo bên trên là SomeQueue

\* [x]3: Các thược tính của queue này tạm thời tin nhắn không có gì

\* đặc biệt nên đễ là null.

\* [x]4: là dữ liệu sẻ được gửi lên queue.

\*/

channel.BasicPublish("", "SomeQueue", null, body);

}

}

}

}

}

}

**Code của C:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

using System.Threading;

namespace Receiveer

{

class Receiver

{

static void Main(string[] args)

{

var factory = new ConnectionFactory() { HostName = "localhost" };

using (var connection = factory.CreateConnection())

{

using (var channel = connection.CreateModel())

{

// Tạo queue vì có thể C sẻ chạy trước P, nếu không có queue sẻ báo lỗi.

channel.QueueDeclare("SomeQueue", false, false, false, null);

/\* [x] Ta sẻ kêu server chuyển tin nhắn từ trên server xuống. Tin nhắn được chuyển

\* từ server xuống sẻ có dạng bất động bộ, nên ta sẻ cung cấp một callback dùng làm

\* bộ đệm để trử tin nhắn cho đến khi cần dùng đến, có là tác dụng của QueueingBasicConsumer\*/

var consumer = new QueueingBasicConsumer(channel);

/\* [1]: Tên của queue mà consumer sẻ nhận tin nhắn từ server.

\* [2]: noAck: true nghĩa là server sẻ xóa tin nhắn ngay khi gửi cho C mà không

\* cần biết C có nhận được tin nhắn đó hay không, Ack = Acknowledged.

\* [3]:

\*/

channel.BasicConsume("SomeQueue", true, consumer);

Console.WriteLine("[\*] Waiting for messages." +

"To exit press CTRL+C");

while (true)

{

/\* Code sẻ đứng tại đây cho đến khi nhận được tin nhắn nào đó từ server\*/

var ea = (BasicDeliverEventArgs)consumer.Queue.Dequeue();

/\*Tin nhắn nhận được từ server gồm các thông số khác như thuộc tính, header,..

tuy nhiên khi sender gửi ta không có khao báo thuộc tính, chỉ đơn thuần là gửi tin

nhắn nên chỉ cần lấy tin nhắn (body)\*/

var body = ea.Body;

/\*Chuyển từ byte thành string và tiến hành in ra màn hình console.\*/

var message = Encoding.UTF8.GetString(body);

Console.WriteLine(" [x] Received {0}", message);

}

}

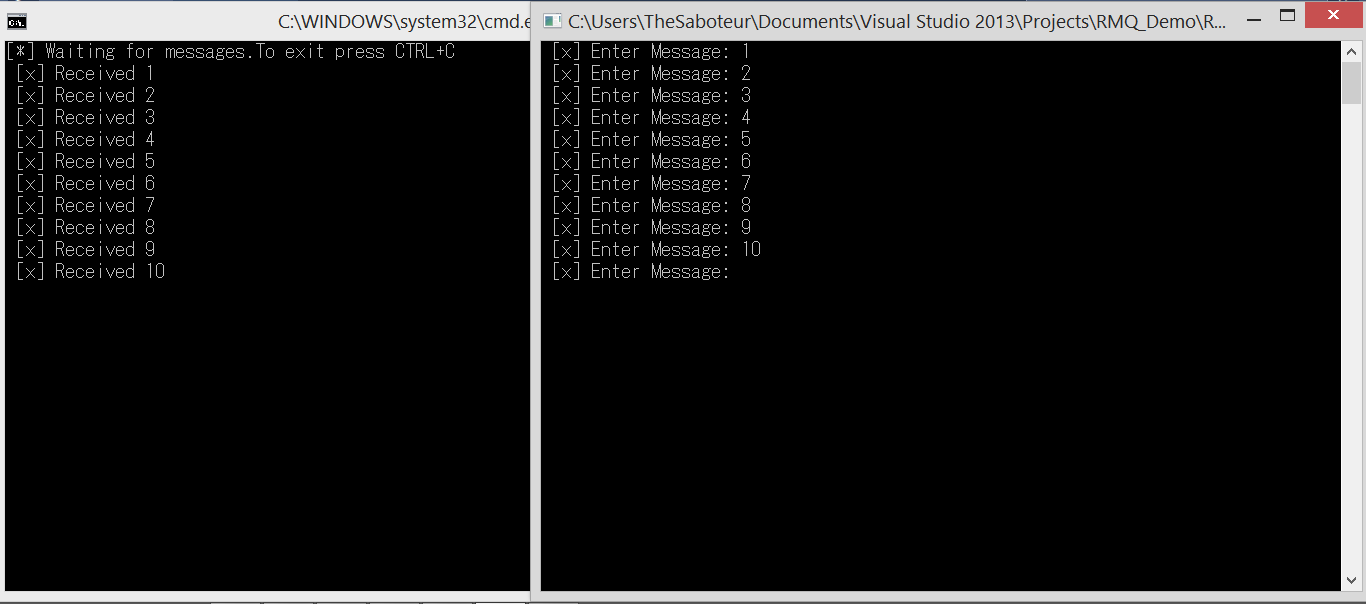
}

}

}

}

**Kết quả:**

****

# VD2:



* Ở ví dụ này cũng một queue nhưng ta sẻ có 2 consumer, và hai consummer này nhận tin nhắn dạng round-robin và tìm hiểu thêm làm sao để queue và tin nhắn của queue đó không bị mất khi RabbitMQ server được tắt và các cơ chế để đảm bảo rằng tin nhắn sẻ không bị mất.

\***Code của P (Newtask.cs)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RabbitMQ.Client;

namespace NewTask

{

class NewTask

{

public static void Main(string[] args)

{

var factory = new ConnectionFactory() { HostName = "localhost" };

using (var connection = factory.CreateConnection())

{

using (var channel = connection.CreateModel())

{

/\* Bên consumer đã dùng Ack nên tin nhắn sẻ không bị mất nếu consumer tắt giữa chừng,

\* nhưng nếu server của RabbitMQ bị mất thì tin nhắn vẫn sẻ mất, vì thế bên dưới ở phần

\* [x] QueueDeclare ta cho durable = true ~ task\_queue sẻ không bị mất nếu RabbitMQ server tắt

\* [x] Tiếp theo ta phải SetPersistent n = true.

\*/

channel.QueueDeclare("task\_queue",true, false, false, null); // durable = true

// Tạo ra thuộc tính cơ bản của channel

var properties = channel.CreateBasicProperties();

/\* Chỉnh DeliveryMode thành Persistent

[x] Khi tạo queue nếu queue có durable = true thì queue sẻ không bị mất nếu server

tắt, tuy nhiên tin nhắn sẻ vẫn bị mất đối với những tin nhắn thường, cho nên ta dùng

hàm bên dưới để thay đổi cách channel sẻ gửi tin nhắn lên queue\*/

properties.SetPersistent(true);

while (true)

{

string message = Console.ReadLine();

var body = Encoding.UTF8.GetBytes(message);

/\*Khi gửi tin nhắn, ta gửi kèm properties lên\*/

channel.BasicPublish("", "task\_queue", properties, body);

}

}

}

}

}

}

\***Code của C (Worker.cs)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

using System.Threading;

namespace Worker

{

class Worker

{

static void Main(string[] args)

{

var factory = new ConnectionFactory() { HostName = "localhost" };

using (var connection = factory.CreateConnection())

{

using (var channel = connection.CreateModel())

{

channel.QueueDeclare("task\_queue", true, false, false, null);

/\* Fair: Dispatch

\* Trường hợp nếu có tin nhắn tồn đọng trong queue, nếu ta mở 2 worker lên thì workder nào

\* được mở trước sẻ được gửi tất cả các queue tồn đọng và cố gắng nhận trong khi worker thứ 2

\* không làm gì cả, vì RabbitMQ chỉ dispatch tin nhắn khi mới vào queue

\* [x] Để tránh trường hợp này ta chỉnh cho channel BasicQos có prefetchCount = 1, nghĩa là

\* không gửi quá 1 tin nhắn cho worker này cho đến khi nó xong tin cũ và ack lên server.

\*/

/\*

[1]: prefectSize: Nghĩa là tin nhắn sẻ gửi trước, khi mà client đã xử lý xong tin nhắn trước

đó thì tin nhắn này đã được gửi trước, nên nó sẻ nằm trong máy thay vì gửi xuống channel

[2]: prefect-count: Nói với rabbitMQ rằng cũng một lúc không gửi quá 1 tin nhắn cho worker

này, mà phải đợi cho worker đó xong trước tin nhắn trước đó, chỉ hoạt động khi noAck = false;

\*/

channel.BasicQos(0,1, false);

var consumer = new QueueingBasicConsumer(channel);

/\*ack = acknowledgments = false, nghĩa là khi gửi tin nhắn từ server thì queue đó trên

server chưa mất cho ack = false, nếu đã chỉnh ack = false thì sau khi xử lý xong tin nhắn

phải ack lên server.\*/

channel.BasicConsume("task\_queue", false, consumer);

Console.WriteLine(" [\*] Waiting for messages. " +

"To exit press CTRL+C");

while (true)

{

/\*Dequeue\*/

var ea = (BasicDeliverEventArgs)consumer.Queue.Dequeue();

/\*Lấy nội dung tin nhắn\*/

var body = ea.Body;

/\*Chuyển sang string\*/

var message = Encoding.UTF8.GetString(body);

Console.WriteLine(" [x] Received {0}", message);

/\*1 dấu . trong tin nhắn sẻ delay 1 giây, dùng để test cho nhận thấy được tin nhắn

được dispatch như thế nào\*/

int dots = message.Split('.').Length - 1;

Thread.Sleep(dots \* 1000);

Console.WriteLine(" [x] Done");

/\*Nếu đã chỉnh Ack = false thì sau khi tải xong phải gọi hàm này để RMQ xóa và gửi tiếp nếu

còn file trong queue.\*/

channel.BasicAck(ea.DeliveryTag, false);

}

}

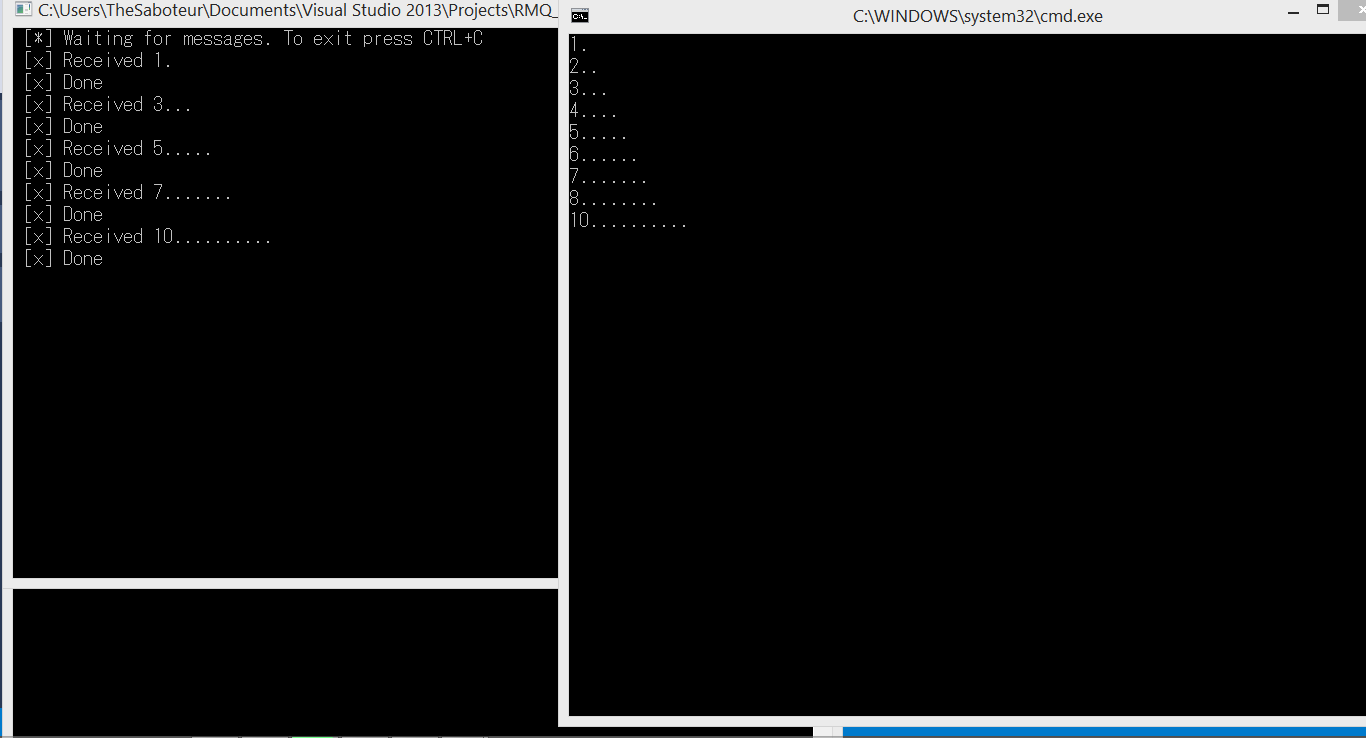
}

}

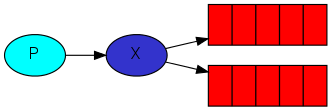
}

}

\***Kết quả:**



# VD3:



* Ờ các VD trước, ta chỉ gửi tin nhắn cho một consumer ở VD này ta sẻ gửi một tin nhắn cho nhiều consumer, và tin nhắn không được gửi trực tiếp từ producer vào queue mà sẻ thông qua exchange (*Ký hiệu là X*).
* Exchange làm công việc là nhận tin nhắn từ một phía và đẩy nó vào queue. Exchange phải biết được rằng nó phải làm gì với tin nhắn mà nó nhận được, nên đẩy vào queue nào đó, đầy vào nhiều queue hay xóa tin nhắn đó đi…
* Có một số exchange có sẵn là direct, topic, header và fanout, ở VD này sẻ dùng fanout, nghĩa là tin nhắn sẻ được đẫy cho tất cả các queue đang kết nối với nó

\***Code của P(Emitlog.cs)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RabbitMQ.Client;

namespace EmitLog

{

class Emitlog

{

static void Main(string[] args)

{

var factory = new ConnectionFactory() { HostName = "localhost" };

using (var connection = factory.CreateConnection())

{

using (var channel = connection.CreateModel())

{

/\*

\* [x] fanout: Gửi tin nhắn nó nhận được cho tất cả các queue nó biết.

\*/

channel.ExchangeDeclare("logs", "fanout");

while(true)

{

var message = Console.ReadLine();

var body = Encoding.UTF8.GetBytes(message);

/\*VD trước ta gửi tin nhắn vào thẳng queue, ở đây là sẻ gửi vào exchange.\*/

channel.BasicPublish("logs", "", null, body);

}

}

}

}

}

}

\***Code của C(ReceiveLog.cs)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

namespace ReceiveLogs

{

class ReceiveLog

{

static void Main(string[] args)

{

var factory = new ConnectionFactory() { HostName = "localhost" };

using (var connection = factory.CreateConnection())

{

using (var channel = connection.CreateModel())

{

/\*Tạo ra 1 exchange kiểu fanout: fanout là tin nhắn sẻ được gửi cho tất cả các queue đang

kết nối vối exchange này \*/

channel.ExchangeDeclare("logs", "fanout");

/\*Các VD trước khi tạo queue thì ta gán cho nó một tên nhưng trong trường hợp này khi

kết nối với exchange ta cần một tên queue khác mà không trùng với các tên queue đã có,

hàm bên dưới sẻ yeu cầu server tạo một queue không trùng tên, queue bên dưới durable = false

Một random queue thường có dạng tương tự amq.gen-JzTY20BRgKO-HjmUJj0wLg.\*/

var queueName = channel.QueueDeclare().QueueName;

/\*Bind queue vừa mới tạo với Exchange\*/

channel.QueueBind(queueName, "logs", "");

var consumer = new QueueingBasicConsumer(channel);

channel.BasicConsume(queueName, true, consumer);

Console.WriteLine("[\*] Waiting for Message");

while (true)

{

var ea = (BasicDeliverEventArgs)consumer.Queue.Dequeue();

var body = ea.Body;

var message = Encoding.UTF8.GetString(body);

Console.WriteLine(" [x] {0}", message);

}

}

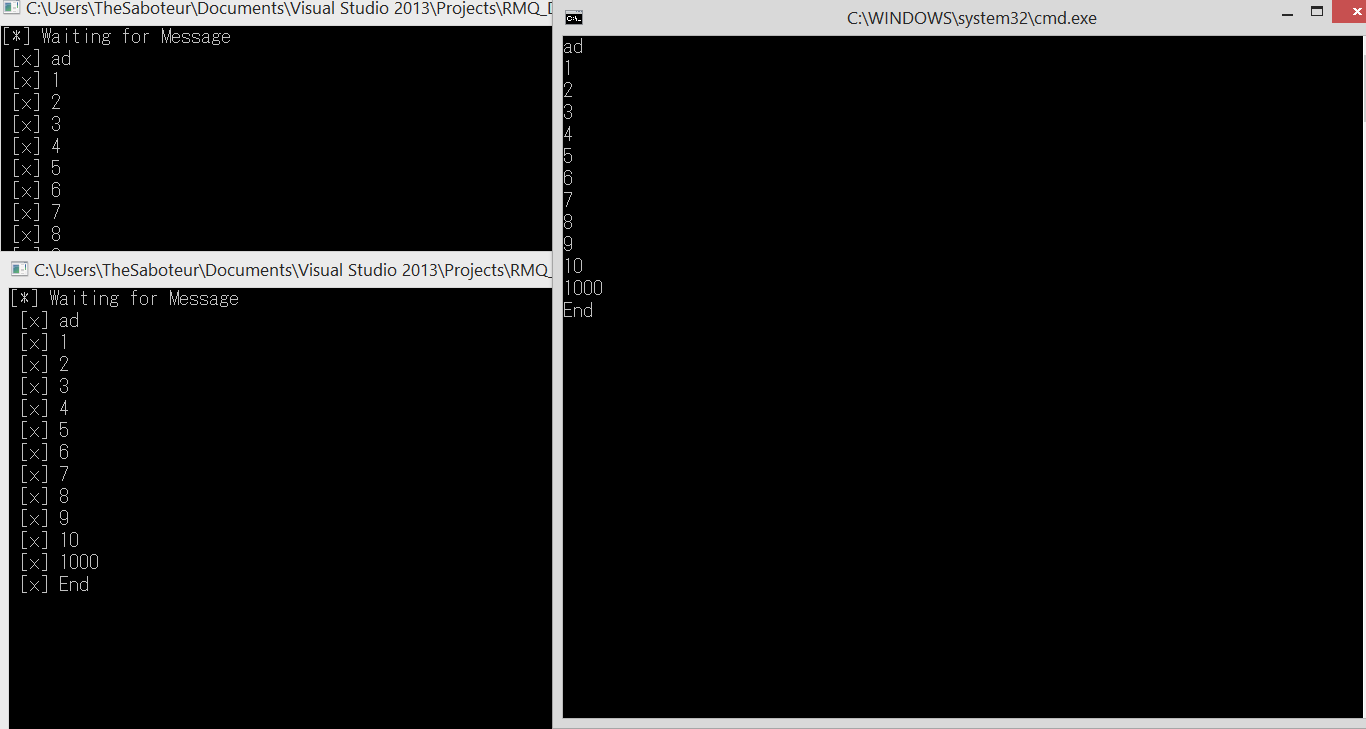
}

}

}

}

\***Kết quả**



# VD4:



* Ờ các VD này ta sẻ dùng direct exchange, trong trường hợp này, mỗi tin nhắn từ P gửi đi sẻ kèm theo routing key đưa lên cho exchange, và direct exchange này sẻ dựa vào routing key và gửi cho queue thích hợp.
* Giả sử rằng có phần mềm bảo lỗi, ta muốn những lỗi nào là error và warning thì gửi cho một C, còn info thì gửi cho một C còn những tin nhắn khác thì hũy.

**\*Code của C (ReceiveLog.cs):**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

namespace ReceiveLogs

{

class ReceiveLog

{

static void Main(string[] args)

{

var factory = new ConnectionFactory() { HostName = "localhost" };

using (var connection = factory.CreateConnection())

{

using (var channel = connection.CreateModel())

{

/\*\*/

channel.ExchangeDeclare("direct\_logs", "direct");

var queueName = channel.QueueDeclare().QueueName;

Console.WriteLine("Enter severity (severity,severity,...)");

string severity = Console.ReadLine();

/\*Tách chuỗi theo dấu , nghĩa là consumer này sẻ nói với exchange là chỉ nhận những tin nhắn

có routing key là những chuỗi nhập vào, cách nhau bởi dấu ,\*/

foreach (string s in severity.Split(','))

{

channel.QueueBind(queueName, "direct\_logs", s.Trim());

}

var consumer = new QueueingBasicConsumer(channel);

channel.BasicConsume(queueName, true, consumer);

Console.WriteLine("[\*] Waiting for Message");

while (true)

{

var ea = (BasicDeliverEventArgs)consumer.Queue.Dequeue();

var body = ea.Body;

var message = Encoding.UTF8.GetString(body);

Console.WriteLine(" [x] {0}", message);

}

}

}

}

}

}

**\*Code của P (EmitLog.cs):**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RabbitMQ.Client;

namespace EmitLog

{

class Emitlog

{

static void Main(string[] args)

{

var factory = new ConnectionFactory() { HostName = "localhost" };

using (var connection = factory.CreateConnection())

{

using (var channel = connection.CreateModel())

{

/\*Direct\_logs chỉ gửi tin nhắn cho những queue nào mà có routing key phù hợp\*/

channel.ExchangeDeclare("direct\_logs", "direct");

Console.WriteLine("Severity: info - warning - error");

Console.WriteLine("Enter Message (Message:Severity)");

while(true)

{

/\*Tin nhắn có dạng <Tin nhắn>:<Routing key>\*/

var message = Console.ReadLine();

if (!message.Contains(':'))

continue;

string[] arr = message.Split(':');

var severity = arr[1];

var body = Encoding.UTF8.GetBytes(arr[0]);

/\*routingKey = severity, tùy vào severity mà tin nhắn này sẻ đến đúng comsumer cần đến.\*/

channel.BasicPublish("direct\_logs", severity, null, body);

}

}

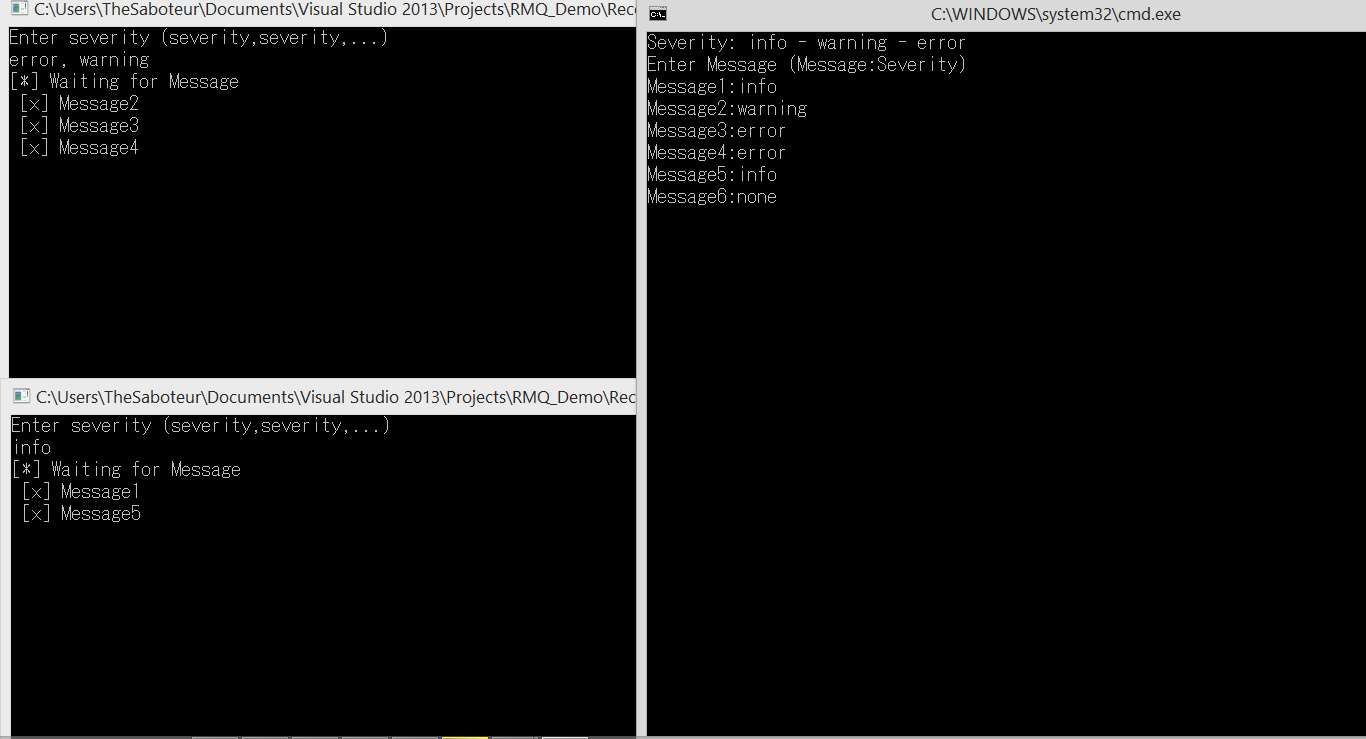
}

}

}

}

**\*Kết quả**



# VD5:



* VD trước ta gửi tin nhắn vào direct exchange, và tùy theo độ nghiêm trọng tin nhắn đó sẻ được gửi cho đúng queue, đặt ra thêm trường xem phải làm thể nào nếu phải xét luôn cả người gửi là ai kèm với độ nghiêm trọng? Ta sẻ dùng topic exchange.
* Topic exchange củng tương tự như direct exchange nhưng khác ở routing key, routing key của topic exchange sẻ được cách nhau bởi dấu “.”, miễn sao không vượt quá 255 ký tự, ngoài ra có các ký hiệu đặc biệt:
  + “\*”: Thay thế cho một chữ.
  + “#”: Thay thế cho một câu hoặc không.
* VD này ta sẻ viết ứng dụng có 3 C:
  + C1: chỉ nhận những tin nhắn báo lỗi là error và warning (*Không quan tâm người gửi*).
  + C2: nhận các tin nhắn có báo lỗi là info (*Không quan tâm người gửi*).
  + C3: Nhận tất cả các thông báo lỗi từ người dùng có tên admin.

**\*Code của C (ReceiveLog.cs)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

namespace ReceiveLogs

{

class ReceiveLog

{

static void Main(string[] args)

{

var factory = new ConnectionFactory() { HostName = "localhost" };

using (var connection = factory.CreateConnection())

{

using (var channel = connection.CreateModel())

{

channel.ExchangeDeclare("topic\_logs", "topic");

var queueName = channel.QueueDeclare().QueueName;

Console.WriteLine("Enter binding key: ");

string bindingKey = Console.ReadLine();

foreach (string s in bindingKey.Split(','))

{

channel.QueueBind(queueName, "topic\_logs", s.Trim());

}

var consumer = new QueueingBasicConsumer(channel);

channel.BasicConsume(queueName, true, consumer);

Console.WriteLine("[\*] Waiting for Message");

while (true)

{

var ea = (BasicDeliverEventArgs)consumer.Queue.Dequeue();

var body = ea.Body;

var message = Encoding.UTF8.GetString(body);

Console.WriteLine(" [x] {0}", message);

}

}

}

}

}

}

**\*Code của P (Emitlog.cs)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RabbitMQ.Client;

namespace EmitLog

{

class Emitlog

{

static void Main(string[] args)

{

var factory = new ConnectionFactory() { HostName = "localhost" };

using (var connection = factory.CreateConnection())

{

using (var channel = connection.CreateModel())

{

/\*

\* Topic exchange là lọc tin nhắn theo nhiều điều kiện khác nhau.

\* [x] VD <Người dùng>.<Lỗi>

\* admin.\*: Mọi lỗi mà admin bị

\* \*.high : Lỗi mà bất cứ người nào bị có độ ưu tiên cao.

\*/

channel.ExchangeDeclare("topic\_logs", "topic");

Console.WriteLine("Severity: info - warning - error");

Console.WriteLine("Enter Message (Message:Sender.Severity)");

while(true)

{

var message = Console.ReadLine();

if (!message.Contains(':'))

continue;

string[] arr = message.Split(':');

var severity = arr[1];

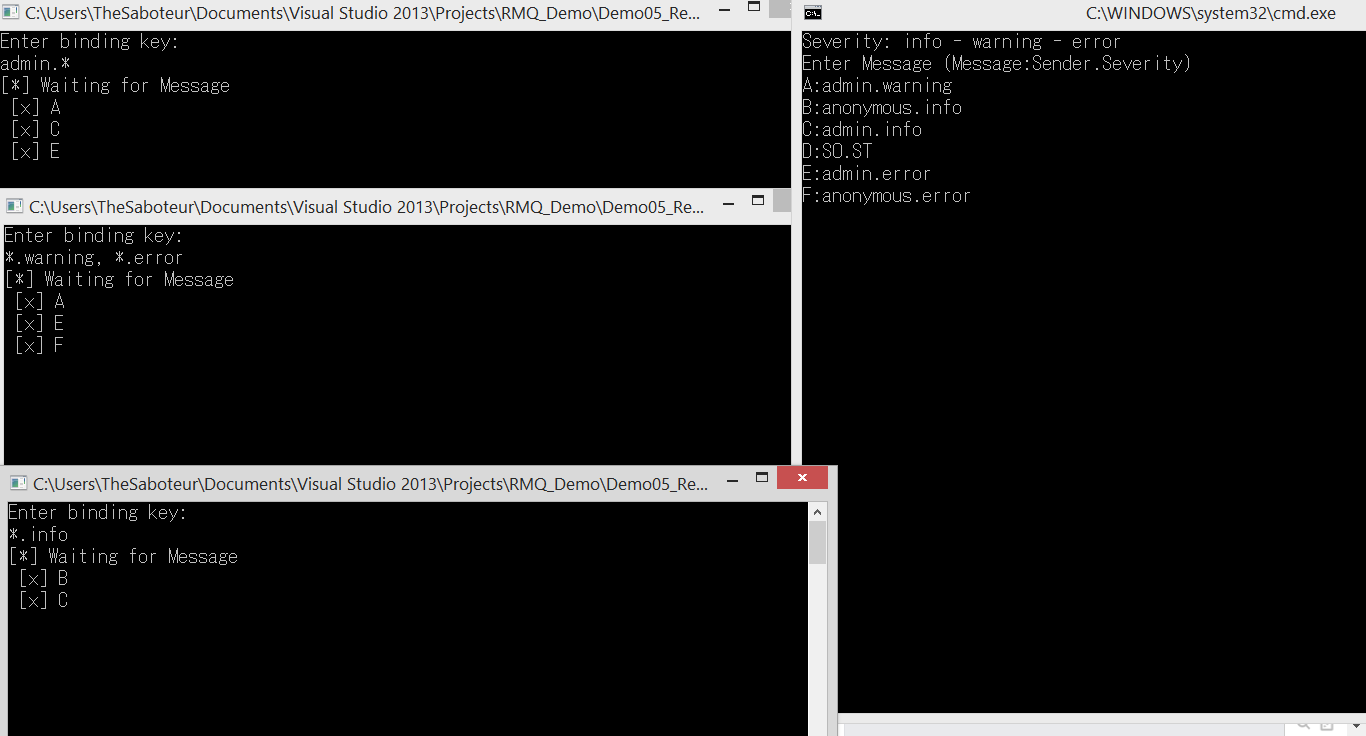
var body = Encoding.UTF8.GetBytes(arr[0]);

// routingKey = severity, tùy vào severity mà tin nhắn này sẻ đến đúng comsumer cần đến.

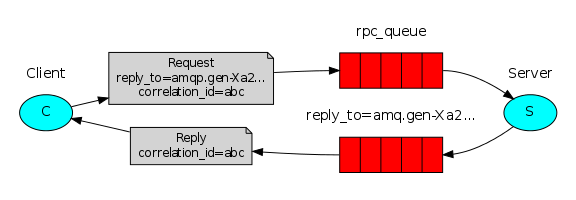
channel.BasicPublish("topic\_logs", severity, null, body);

}}}}}

**\*Kết quả:**



# VD6:



* Ở các VD trước, ta chỉ làm một bên gửi, một bên nhận ở VD này ta sẻ cho C có cả chức năng nhận và gửi.
  + Client tạo một queue riêng của mình.
  + Client gửi tin nhắn lên rpc\_queue kèm các thông tin như ID
  + Server kiểm tra, nếu có tin nhắn từ queue thì nhận tin nhắn,xử lý trả đúng thông tin đó về queue mà trên tin nhắn đã kèm theo.
  + Client nhận tin nhắn, kiểm tra ID, chuyển và xuât tin nhắn.

**\*Code của Client (RPCClient.cs)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

namespace RPCClient

{

class RPCClient

{

IConnection connection;

IModel channel;

string replyQueueName;

QueueingBasicConsumer consumer;

public RPCClient()

{

// Tạo connection

var factory = new ConnectionFactory() { HostName = "localhost" };

connection = factory.CreateConnection();

channel = connection.CreateModel();

replyQueueName = channel.QueueDeclare();

consumer = new QueueingBasicConsumer(channel);

channel.BasicConsume(replyQueueName, true, consumer);

}

public string Call(string message)

{

/\*Tạo ra một corrId không trùng\*/

var corrId = Guid.NewGuid().ToString();

var props = channel.CreateBasicProperties();

props.ReplyTo = replyQueueName;

/\*Gửi thông tin này theo để lát server gửi lại sẻ kèm Id này, sau đó ta tiền hành kiểm tra,

nếu trùng nhau nghĩa là tin nhắn trả đúng\*/

props.CorrelationId = corrId;

var messageBytes = Encoding.UTF8.GetBytes(message);

/\*Gửi lên cho rpc\_queue\*/

channel.BasicPublish("", "rpc\_queue", props, messageBytes);

while (true)

{

// Đứng nhận tin nhắn

var ea = (BasicDeliverEventArgs)consumer.Queue.Dequeue();

/\* Kiểm tra xem cái Id nhận lại có trùng với Id ta gửi đi hay không? \*/

if (ea.BasicProperties.CorrelationId == corrId)

{

return Encoding.UTF8.GetString(ea.Body);

}

}

}

public void Close()

{

connection.Close();

}

}

class MainS

{

public static void Main()

{

var rpcClient = new RPCClient();

Console.WriteLine(" [x] Enter number: ");

string requestNumber = Console.ReadLine();

var response = rpcClient.Call(requestNumber); // Gửi tin nhắn 30 đi.

Console.WriteLine(" [.] {0}", response);

Console.ReadLine();

rpcClient.Close();

}

}

}

**\*Code của Server (RPCServer.cs)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

namespace RPCServer

{

class RPCServer

{

private static int fib(int n)

{

if (n == 0 || n == 1) return n;

return fib(n - 1) + fib(n - 2);

}

static void Main(string[] args)

{

var factory = new ConnectionFactory() { HostName = "localhost" };

using (var connection = factory.CreateConnection())

{

using (var channel = connection.CreateModel())

{

/\*Tạo queue, tất cả request sẻ gửi lên queue này\*/

channel.QueueDeclare("rpc\_queue",false, false, false, null);

/\*Không gửi quá một tin nhắn cho Worker này\*/

channel.BasicQos(0, 1, false);

var consumer = new QueueingBasicConsumer(channel);

channel.BasicConsume("rpc\_queue", false, consumer);

Console.WriteLine(" [x] Awaiting RPC request");

while (true)

{

string response = null;

/\* Đợi bắt tin nhắn từ queue \*/

var ea = (BasicDeliverEventArgs)consumer.Queue.Dequeue();

var body = ea.Body; // Lấy nội dung tin nhắn

var props = ea.BasicProperties; // Lấy thiết lập của tin nhắn

var replyProps = channel.CreateBasicProperties(); // Tạo ra thiết lập mới của tin nhắn

// Gán corrId của tin gửi lên với replyProps(sẻ gửi về) để bên nhận kiểm tra Id

replyProps.CorrelationId = props.CorrelationId;

try

{

var message = Encoding.UTF8.GetString(body);

int n = int.Parse(message);

Console.WriteLine(" [.] fib({0})", message);

response = fib(n).ToString();

}

catch (Exception e)

{

Console.WriteLine("Ex: " + e.Message);

response = "";

}

finally

{

//Console.WriteLine("props.Replyto: " + props.ReplyTo.ToString());

var responseBytes = Encoding.UTF8.GetBytes(response);

// props.ReplyTo là tên của queue đã tạo bên client, nói chung là để gửi cho đúng queue.

channel.BasicPublish("", props.ReplyTo, replyProps, responseBytes);

/\*Do bên trên đã thiết lập Qos nên khi nhận và xử lý xongt in nhắn phải ack cho

server biết để nếu còn tin nhắn tồn đọng trong queue thì server sẻ tiếp tục gửi\*/

channel.BasicAck(ea.DeliveryTag, false);

}}}}}}

**\*Kết quả:**

