# 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);

}

}

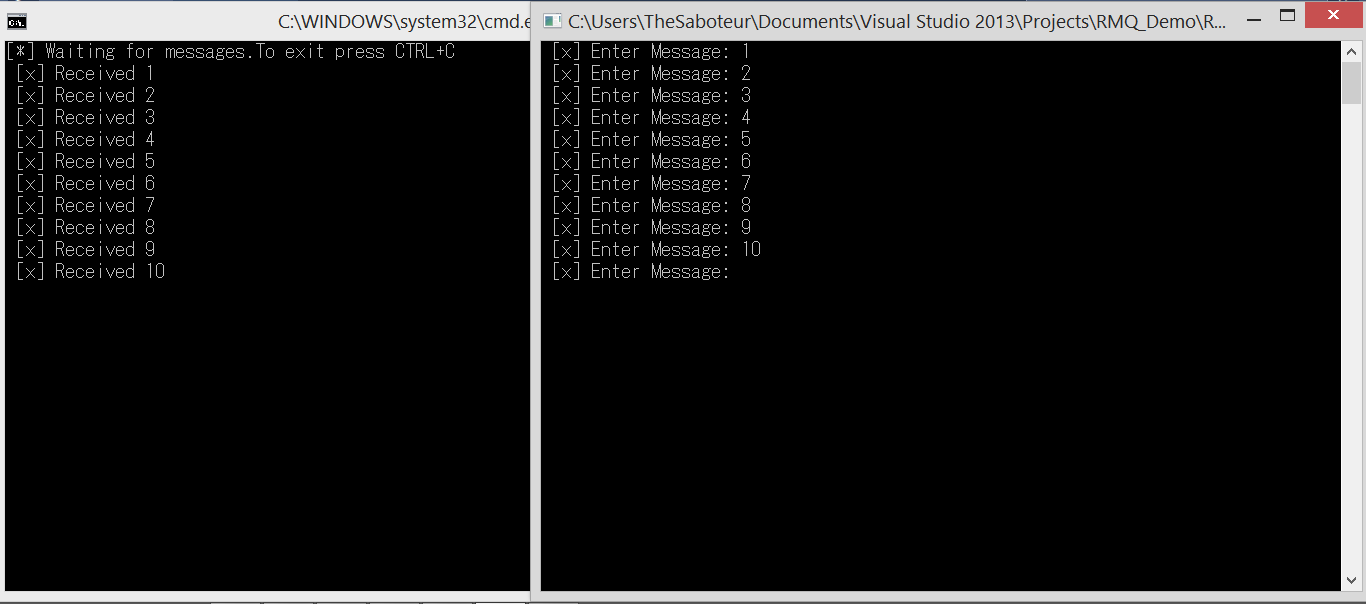
}

}

}

}

**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:**

