# 05-rabbitmq-发布订阅

## 先决条件

本教程假定RabbitMQ已在标准端口（5672）上的localhost上安装并运行。如果使用不同的主机，端口或凭据，连接设置将需要调整。

## 发布/订阅

在上一个教程中，我们创建了一个工作队列。工作队列背后的假设是每个任务都交付给一个工作人员。在这部分中，我们会做一些完全不同的事情 - 我们会向多个消费者传递信息。这种模式被称为“发布/订阅”。

为了说明这个模式，我们要建立一个简单的日志记录系统。它将包括两个程序 - 第一个将发出日志消息，第二个将接收并打印它们。

在我们的记录系统中，接收器程序的每个运行副本都会收到消息。这样我们就可以运行一个接收器并将日志指向磁盘; 同时我们可以运行另一个接收器并查看屏幕上的日志。

基本上，已发布的日志消息将被广播到所有接收者

## 交换器

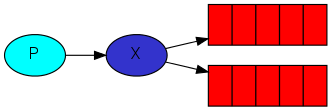
在本教程的前面部分，我们发送和接收到队列中的消息。现在是时候在Rabbit中引入完整的消息传递模式了。

让我们快速了解我们在以前的教程中介绍的内容：

* **生产者**是发送消息的用户应用程序。
* **队列**是存储消息的缓冲器。
* **消费者**是接收消息的用户的应用程序。

RabbitMQ中的消息传递模型的核心思想是，生产者从不将任何消息直接发送到队列。实际上，生产者通常甚至不知道是否将消息传递到任何队列。

相反，生产者只能将信息发送到交换器。交换是一件非常简单的事情。一方面，它收到来自生产者的消息，另一方将它们推送到队列。交换器必须准确知道接收到的消息如何处理。消息应该附加到特定队列吗？消息应该附加到很多队列吗？消息或者应该丢弃。其规则由**交换类型**定义 。



有几种交换类型可用：**直接**，**主题**，**标题** 和**扇出**。我们将重点关注最后一个——扇出。让我们创建一个这种类型的交换，并将其称为日志：

|  |
| --- |
| channel.exchangeDeclare("logs", "fanout"); |

扇出交换非常简单。正如您可以从名称猜测的，它只是将所有收到的消息广播到所有知道的队列。这正是我们需要的记录器。

## 列出交换器

要列出服务器上的交换机，您可以运行有用的rabbitmqctl：

|  |
| --- |
| sudo rabbitmqctl list\_exchanges |

在这个列表中会有一些amq.\*交换和默认（未命名）交换。这些是默认创建的，但是不太可能需要使用它们。

## 无名交换器

在本教程的前面部分，我们对交换没有任何意见，但仍然能够将消息发送到队列。这是可能的，因为我们使用默认交换，我们通过空字符串（“”）标识。

回想一下我们之前发布的消息：

|  |
| --- |
| channel.basicPublish("", "hello", null, message.getBytes()); |

第一个参数是交换的名称。空字符串表示默认或无名交换器：如果routingKey存在，routingKey将消息路由到指定队列。

现在，我们可以发布消息到我们命名的交换器：

|  |
| --- |
| channel.basicPublish( "logs", "", null, message.getBytes()); |

## 临时队列

你以前记得我们使用的是具有指定名称的队列（记得hello和task\_queue？）。能够命名队列对我们而言至关重要 - 我们需要将工作进程指向同一个队列。当您想要在生产者和消费者之间共享队列时，给队列一个名字很重要。

但是我们的日志记录器不是这样。我们希望记录到所有的日志消息，而不仅仅是它们的一部分。我们也只对当前的消息感兴趣对旧的消息不感兴趣。要解决这个问题我们需要两件东西。

首先，每当我们连接到RabbitMQ，我们需要一个新的空的队列。为此，我们可以创建一个具有随机名称的队列，或者甚至更好 - 让服务器为我们选择一个随机队列名称。

其次，一旦我们断开消费者，队列应该被自动删除。

在Java客户端中，当我们没有为queueDeclare()提供参数时， 我们创建了一个具有生成名称的非持久，排他，自动删除队列：

|  |
| --- |
| String queueName = channel.queueDeclare().getQueue(); |

此时，queueName包含一个随机队列名称。例如，它可能看起来像amq.gen-JzTY20BRgKO-HjmUJj0wLg。

## 绑定



我们已经创建了一个扇出交换和队列。现在我们需要告诉交换器发送消息到我们的队列。交换和队列之间的关系称为**绑定**。

|  |
| --- |
| channel.queueBind(queueName, "logs", ""); |

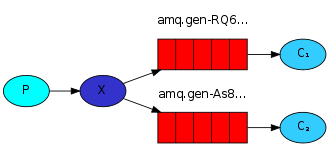
从现在开始，日志交换将附加消息到我们的队列。

## 列出绑定

你可以列出现有的绑定

|  |
| --- |
| rabbitmqctl list\_bindings |

## 把它们放在一起



发出日志消息的生产者程序与上一个教程并没有太大的区别。最重要的变化是我们现在想将消息发布到我们的日志交换，而不是无名的。发送时需要提供一个routingKey，但是对于扇出交换来说，它的值被忽略。以下是EmitLog.java程序的代码 ：

|  |
| --- |
| **package** com.example.rabbitmq;  **import** com.rabbitmq.client.BuiltinExchangeType; **import** com.rabbitmq.client.Channel; **import** com.rabbitmq.client.Connection; **import** com.rabbitmq.client.ConnectionFactory;  */\*\*  \* Author: 王俊超  \* Date: 2017-06-17 11:08  \* All Rights Reserved !!!  \*/* **public class** EmitLog {  **private static final** String ***EXCHANGE\_NAME*** = **"logs"**;   **public static void** main(String[] args) **throws** Exception {  ConnectionFactory factory = **new** ConnectionFactory();  factory.setHost(**"localhost"**);  Connection connection = factory.newConnection();  Channel channel = connection.createChannel();   *// 声明一个消息场所* channel.exchangeDeclare(***EXCHANGE\_NAME***, BuiltinExchangeType.***FANOUT***);   String message = *getMessage*(args);   *// 发布消息* channel.basicPublish(***EXCHANGE\_NAME***, **""**, **null**, message.getBytes(**"UTF-8"**));  System.***out***.println(**" [x] Sent '"** + message + **"'"**);   channel.close();  connection.close();  }   **private static** String getMessage(String[] strings) {  **if** (strings.**length** < **1**) {  **return "info: Hello World!"**;  }  **return** *joinStrings*(strings, **" "**);  }   **private static** String joinStrings(String[] strings, String delimiter) {  **int** length = strings.**length**;   **if** (length == **0**) {  **return ""**;  }  StringBuilder words = **new** StringBuilder(strings[**0**]);   **for** (**int** i = **1**; i < length; i++) {  words.append(delimiter).append(strings[i]);  }  **return** words.toString();  } } |

如你所见，建立连接后，我们宣布交换。此步骤是必须的，因为禁止发布到不存在的交换器。

如果没有任何队列绑定到交换器，消息将丢失，但是对我们来说没关系; 如果没有消费者正在收听，我们可以放心地放弃信息。

ReceiveLogs.java的代码：

|  |
| --- |
| **package** com.example.rabbitmq;  **import** com.rabbitmq.client.\*;  **import** java.io.IOException;  */\*\*  \* Author: 王俊超  \* Date: 2017-06-17 11:15  \* All Rights Reserved !!!  \*/* **public class** ReceiveLogs {  **private static final** String ***EXCHANGE\_NAME*** = **"logs"**;   **public static void** main(String[] argv) **throws** Exception {  ConnectionFactory factory = **new** ConnectionFactory();  factory.setHost(**"localhost"**);  Connection connection = factory.newConnection();  Channel channel = connection.createChannel();   channel.exchangeDeclare(***EXCHANGE\_NAME***, BuiltinExchangeType.***FANOUT***);  String queueName = channel.queueDeclare().getQueue();  channel.queueBind(queueName, ***EXCHANGE\_NAME***, **""**);   System.***out***.println(**" [\*] Waiting for messages. To exit press CTRL+C"**);   Consumer consumer = **new** DefaultConsumer(channel) {  @Override  **public void** handleDelivery(String consumerTag, Envelope envelope,  AMQP.BasicProperties properties, **byte**[] body) **throws** IOException {  String message = **new** String(body, **"UTF-8"**);  System.***out***.println(**" [x] Received '"** + message + **"'"**);  }  };  channel.basicConsume(queueName, **true**, consumer);  } } |

## 运行

先运行消费者

再运行生产者