# 06-rabbitmq-发布订阅-spring

## 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

在第一个教程中，我们展示了如何使用start.spring.io来创建一个Spring Initializr项目。这是一个spring-amqp应用程序具有RabbitMQ启动器依赖项。

在上一个教程中，我们创建了一个新的包（tut2）来放置我们的配置，发送者和接收者，并创建了一个包含两个消费者的工作队列。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，我们将实现扇出模式，向多个消费者传递消息。这种模式被称为“发布/订阅”，通过在Tut3Config文件中配置一些bean来实现。

基本上，已发布的消息将被广播到所有接收者。

## 交换器

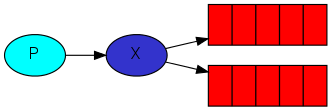
在本教程的前面部分，我们发送和接收到队列中的消息。现在是时候在Rabbit中引入完整的消息传递模式了。

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。我们来配置一个bean来描述这种类型的交换，并将其称为tut.fanout：

|  |
| --- |
| **package** com.example.rabbitmq.tut3;  **import** org.springframework.amqp.core.\*; **import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.Configuration; **import** org.springframework.context.annotation.Profile;  */\*\*  \* 配置文件  \* Author: 王俊超  \* Date: 2017-06-17 11:24  \* All Rights Reserved !!!  \*/* @Profile({**"tut3"**, **"pub-sub"**, **"publish-subscribe"**}) @Configuration **public class** Tut3Config {  */\*\*  \* 扇出交换器  \*  \** ***@return*** *\*/* @Bean  **public** FanoutExchange fanout() {  **return new** FanoutExchange(**"tut.fanout"**);  }   */\*\*  \* 消息接收者配置类  \*/* @Profile(**"receiver"**)  **private static class** ReceiverConfig {  */\*\*  \* 匿名队列  \*  \** ***@return*** *\*/* @Bean  **public** Queue autoDeleteQueue1() {  **return new** AnonymousQueue();  }   @Bean  **public** Queue autoDeleteQueue2() {  **return new** AnonymousQueue();  }   */\*\*  \* 将匿名队列绑定到扇出交换器上  \*  \** ***@param fanout*** *\** ***@param autoDeleteQueue1*** *\** ***@return*** *\*/* **public** Binding binding1(FanoutExchange fanout, Queue autoDeleteQueue1) {  **return** BindingBuilder.*bind*(autoDeleteQueue1).to(fanout);  }   @Bean  **public** Binding binding2(FanoutExchange fanout, Queue autoDeleteQueue2) {  **return** BindingBuilder.*bind*(autoDeleteQueue2).to(fanout);  }   @Bean  **public** Tut3Receiver receiver() {  **return new** Tut3Receiver();  }  }   */\*\*  \* 消息接收者  \** ***@return*** *\*/* @Profile(**"sender"**)  @Bean  **public** Tut3Sender sender() {  **return new** Tut3Sender();  } } |

我们采用与前两个教程相同的方法。我们创建了三个配置文件，教程（“tut3”，“pub-sub”或“publish-subscribe”）。他们都是运行fanout配置文件教程的同义词。接下来我们将FanoutExchange配置为一个bean。在“接收器”（Tut3Receiver）文件中，我们定义“四个bean; 1）两个autoDeleteQueues或者AnonymousQueues和两个绑定，以将这些队列绑定到交换机。

扇出交换非常简单。正如您可以从名称中猜出的，它只是将所有收到的消息广播到所有已知的队列。这就是我们所需要的，以消除我们的信息。

## 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：

|  |
| --- |
| @Autowired  private RabbitTemplate template;  @Autowired  private FanoutExchange fanout; // configured in Tut3Config above  template.convertAndSend(fanout.getName(), "", message); |

从现在起，扇出交换将消息附加到我们的队列中。

## 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）。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。

但是我们的扇出例子不是这样。我们希望记录到所有的消息，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。

其次，一旦我们断开消费者，队列应该被自动删除。为了使用spring-amqp客户端，我们定义了AnonymousQueue，它创建了一个非持久的，独占的，自动删除的队列，其生成名称为：

|  |
| --- |
| @Bean  public Queue autoDeleteQueue1() {  return new AnonymousQueue();  }  @Bean  public Queue autoDeleteQueue2() {  return new AnonymousQueue();  } |

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器和队列。现在我们需要告诉交换器发送消息到我们的队列。交换器和队列之间的关系称为**绑定**。在上面的Tut3Config中，您可以看到我们有两个绑定，每个AnonymousQueue一个绑定。

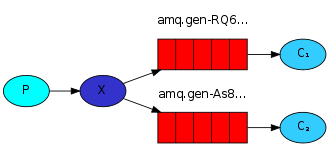
|  |
| --- |
| @Bean  public Binding binding1(FanoutExchange fanout,  Queue autoDeleteQueue1) {  return BindingBuilder.bind(autoDeleteQueue1).to(fanout);  } |

## 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扇出交换器上，而不是无名的交换器。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。这里是tut3.Sender.java程序的代码：

|  |
| --- |
| **package** com.example.rabbitmq.tut3;  **import** org.springframework.amqp.core.FanoutExchange; **import** org.springframework.amqp.rabbit.annotation.RabbitListener; **import** org.springframework.amqp.rabbit.core.RabbitTemplate; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.scheduling.annotation.Scheduled; **import** org.springframework.util.StopWatch;  */\*\*  \* Author: 王俊超  \* Date: 2017-06-17 11:29  \* All Rights Reserved !!!  \*/* **public class** Tut3Sender {    @Autowired  **private** RabbitTemplate **template**;   @Autowired  **private** FanoutExchange **fanout**;   **int dots** = **0**;   **int count** = **0**;   @Scheduled(fixedDelay = **1000**, initialDelay = **500**)  **public void** send() {  StringBuilder builder = **new** StringBuilder(**"Hello"**);  **if** (**dots**++ == **3**) {  **dots** = **1**;  }  **for** (**int** i = **0**; i < **dots**; i++) {  builder.append(**'.'**);  }  builder.append(Integer.*toString*(++**count**));  String message = builder.toString();  **template**.convertAndSend(**fanout**.getName(), **""**, message);  System.***out***.println(**" [x] Sent '"** + message + **"'"**);  } } |

如您所见，我们利用Tut3Config文件中的bean和RabbitTemplate中的自动连接以及我们配置的FanoutExchange。此步骤是必需的，因为禁止发布到不存在的交换机。

如果没有任何队列绑定到交换机，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。

Tut3Receiver.java的代码：

|  |
| --- |
| **package** com.example.rabbitmq.tut3;  **import** org.springframework.amqp.rabbit.annotation.RabbitListener; **import** org.springframework.util.StopWatch;  */\*\*  \* Author: 王俊超  \* Date: 2017-06-17 11:29  \* All Rights Reserved !!!  \*/* **public class** Tut3Receiver {  @RabbitListener(queues = **"#{autoDeleteQueue1.name}"**)  **public void** receive1(String in) **throws** InterruptedException {  receive(in, **1**);  }   @RabbitListener(queues = **"#{autoDeleteQueue2.name}"**)  **public void** receive2(String in) **throws** InterruptedException {  receive(in, **2**);  }   **public void** receive(String in, **int** receiver) **throws** InterruptedException {  StopWatch watch = **new** StopWatch();  watch.start();  System.***out***.println(**"instance "** + receiver + **" [x] Received '"** + in + **"'"**);  doWork(in);  watch.stop();  System.***out***.println(**"instance "** + receiver + **" [x] Done in "** + watch.getTotalTimeSeconds() + **"s"**);  }   **private void** doWork(String in) **throws** InterruptedException {  **for** (**char** ch : in.toCharArray()) {  **if** (ch == **'.'**) {  Thread.*sleep*(**1000**);  }  }  } } |

## 运行

先运行接收者，需要添加运行参数

--spring.profiles.active = pub-sub,receiver --tutorial.client.duration = 60000

再运行发送者，需要添加运行参数

--spring.profiles.active = pub-sub,sender --tutorial.client.duration = 60000