**声明：此文档为个人翻译官方文档，未经允许请勿外传。**

**翻译者：杨德军**

Publish/Subscribe(发布/订阅)

**(using the spring-amqp client)**

**前置条件**

本教程假定RabbitMQ在默认端口（5672）上的本地主机上安装并运行。 如果您使用不同的host, port or credentials，则connections设置需要进行调整。

**从哪里获得帮助**

如果您在阅读本教程时遇到困难，可以通过邮件列表与我们联系[contact us](https://groups.google.com/forum/#!forum/rabbitmq-users)。

在 [first tutorial](http://www.rabbitmq.com/tutorials/tutorial-one-spring-amqp.html)中，我们展示了如何使用start.spring.io来利用Spring Initializr创建一个具有RabbitMQ启动器依赖项的项目，以创建spring-amqp应用程序。

在[previous tutorial](http://www.rabbitmq.com/tutorials/tutorial-two-spring-amqp.html)中，我们创建了一个新的package（tut2）来放置我们的config, sender and receiver，并创建了一个包含两个consumers的work queue。 work queue之后的假设是，每个任务只被delivered给一个worker。

在这一部分，我们将实现fanout pattern向多个consumers deliver message。 这种模式被称为"publish/subscribe"，并通过在我们的Tut3Config文件中配置一些bean来实现。

基本上，published messages将被broadcast给所有的receivers。

Exchanges

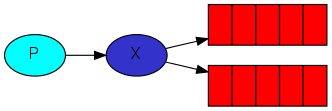
在本教程的前几部分中，我们sent messages并从queue中received messages。 现在是时候介绍在Rabbit中的full messaging model。

让我们快速回顾一下前面教程中的内容：

* *producer* 是sends messages的用户应用程序
* *queue* 是存储消息的缓冲区
* *consumer* 是receives messages 的用户应用程序

RabbitMQ中的消息传递模型的核心思想是producer永远不会将任何messages直接发送到queue中。 实际上，生产者通常甚至不知道message是否会被delivered到任何queue中。

相反，producer只能send messages给*exchange*。 *exchange*是一件非常简单的事情。 一方面它receives来自producers的messages，另一方则pushes它们到queues。 exchange必须知道如何处理收到的消息。 是否应该appended到指定queue？ 它应该appended到许多queues中吗？ 或者它应该被discarded。 这些规则由*exchange type*定义。



有几种可用的exchange类型：direct, topic, headers and fanout。 我们将关注最后一个 - fanout。 让我们配置一个bean来描述这种类型的exchange，并将其称为tut.fanout:

import org.springframework.amqp.core.\*;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Profile;

@Profile({"tut3", "pub-sub", "publish-subscribe"})

@Configuration

public class Tut3Config {

@Bean

public FanoutExchange fanout() {

return new FanoutExchange("tut.fanout");

}

@Profile("receiver")

private static class ReceiverConfig {

@Bean

public Queue autoDeleteQueue1() {

return new AnonymousQueue();

}

@Bean

public Queue autoDeleteQueue2() {

return new AnonymousQueue();

}

@Bean

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();

}

}

@Profile("sender")

@Bean

public Tut3Sender sender() {

return new Tut3Sender();

}

}

我们采用与前两篇教程相同的方法。 我们创建了三个配置文件，教程（“tut3”，“pub-sub”或“publish-subscribe”）。 它们都是运行fanout profile tutorial.的同义词。 接下来，我们将FanoutExchange配置为一个bean。 在“receiver”（Tut3Receiver）文件中，我们定义了“四个bean; 1）两个autoDeleteQueues或AnonymousQueues和两个bindings来将这些queues bind到exchange。

fanout exchange非常简单。 正如你可能从名字中猜出的那样，它只是将收到的所有消息广播到它所知道的所有queues中。 这正是我们需要fanning out our messages。

**Listing exchanges**

要列出服务器上的exchanges，您可以运行永远有用的rabbitmqctl命令：

sudo rabbitmqctl list\_exchanges

在这个列表中将会有一些amq.\* exchanges和default (unnamed) exchange。 这些是默认创建的，但目前不太可能需要使用它们。

**Nameless exchange**

在本教程的以前部分，我们对exchanges一无所知，但仍能够将消息发送到queues。 这是可能的，因为我们使用了一个default exchange，我们用空字符串("")来标识。

回想一下我们之前如何published message：

template.convertAndSend(fanout.getName(), "", message);

第一个参数是autowired sender的exchange的name。 空字符串表示缺省或*nameless* exchange：如果messages存在，则将messages routed到具有由routingKey指定的name的queue。

现在，我们可以publish到我们的named exchange：

@Autowired

private RabbitTemplate template;

@Autowired

private FanoutExchange fanout; *// configured in Tut3Config above*

template.convertAndSend(fanout.getName(), "", message);

从现在开始，fanout exchange会将messages append到我们的queue中。

Temporary queues

正如你以前可能记得我们使用的是具有指定名称的queues (hello)。 能够命名queues对我们至关重要 - 我们需要将workers指向同一队列。 当你想在producers and consumers之间分享queue时，给队列一个名字是很重要的。

但我们的fanout deme并非如此。 我们希望了解所有messages，而不仅仅是其中的一部分。 我们也只对目前flowing messages感兴趣，而不是旧消息。 要解决这个问题，我们需要两件事。

首先，每当我们连接到Rabbit，我们需要一个fresh, empty queue。要做到这一点，我们可以创建一个random name的queue，或者甚至更好 - 让server为我们选择一个随机队列名称。

其次，一旦我们断开consumer，queue应该被automatically deleted。 为了用spring-amqp客户端做到这一点，我们定义了AnonymousQueue，它创建了一个non-durable, exclusive, autodelete queue，其中包含一个生成的名称：

@Bean

public Queue autoDeleteQueue1() {

return new AnonymousQueue();

}

@Bean

public Queue autoDeleteQueue2() {

return new AnonymousQueue();

}

此时我们的queue names包含一个random queue names。 例如，它可能看起来像amq.gen-JzTY20BRgKO-HjmUJj0wLg。

Bindings



我们已经创建了一个fanout exchange和一个queue。 现在我们需要告诉exchange将消息send到我们的queue。 exchange和queue之间的关系称为*binding*。 在上面的Tut3Config中，您可以看到我们有两个bindings，每个AnonymousQueue都有一个绑定。

@Bean

public Binding binding1(FanoutExchange fanout,

Queue autoDeleteQueue1) {

return BindingBuilder.bind(autoDeleteQueue1).to(fanout);

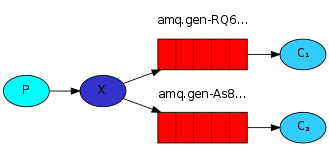
}

**Listing bindings**

你猜对了,你可以使用命令查处存在的bindings,

rabbitmqctl list\_bindings

Putting it all together



发出消息的producer program与以前的教程没有多大区别。 最重要的变化是，我们现在要发布消息给我们的fanout exchange而不是无名的fanout。 发送时我们需要提供一个routingKey ，但是对于fanout exchanges，它的值将被忽略。 这里是tut3.Sender.java程序的代码：

import org.springframework.amqp.core.FanoutExchange;

import org.springframework.amqp.rabbit.core.RabbitTemplate;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.scheduling.annotation.Scheduled;

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 + "'");

}

}

[Tut3Sender.java source](https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/spring-amqp/src/main/java/org/springframework/amqp/tutorials/tut3/Tut3Sender.java)

正如您所看到的，我们利用Tut3Config文件中的bean以及RabbitTemplate中的autowire我们配置的FanoutExchange，这一步是必需的，因为禁止发布到non-existing exchange。

如果没有queue bound到exchange上，这些message将会discard，但这对我们来说没问题; 如果没有消费者正在listening，我们可以放心地discard message。

Tut3Receiver.java:

import org.springframework.amqp.rabbit.annotation.RabbitListener;

import org.springframework.util.StopWatch;

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);

}

}

}

}

[Tut3Receiver.java source](https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/spring-amqp/src/main/java/org/springframework/amqp/tutorials/tut3/Tut3Receiver.java)

像以前一样编译，我们准备好执行fanout sender and receiver。

mvn clean package

当然，要执行本教程，请执行以下操作：

java -jar target/rabbit-tutorials-1.7.1.RELEASE.jar --spring.profiles.active=pub-sub,receiver

--tutorial.client.duration=60000

java -jar target/rabbit-tutorials-1.7.1.RELEASE.jar --spring.profiles.active=pub-sub,sender

--tutorial.client.duration=60000

使用rabbitmqctl list\_bindings，你可以验证代码是否真正创建了bindings and queues。 在运行两个ReceiveLogs.java程序时，您应该看到如下所示的内容：

sudo rabbitmqctl list\_bindings

tut.fanout exchange 8b289c9c-a1eb-4a3a-b6a9-163c4fdcb6c2 queue []

tut.fanout exchange d7e7d193-65b1-4128-a532-466a5256fd31 queue []

结果的解释很简单：来自exchange logs的数据转到两个带有server-assigned names的queues。 这正是我们的目的。

To find out how to listen for a subset of messages, [tutorial 4](http://www.rabbitmq.com/tutorials/tutorial-four-spring-amqp.html)