**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**

Topics

**(using the spring-amqp)**

**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，connections设置将需要调整。

**从哪里获得帮助**

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

在 [previous tutorial](http://www.rabbitmq.com/tutorials/tutorial-four-spring-amqp.html)中，我们改进了信息的灵活性。 我们没有使用仅支持dummy broadcasting的fanoutexchange，而是使用了direct的广播，并且有可能根据routing key选择性接收消息。

尽管使用direct exchange改进了我们的系统，但它仍然有局限性 - 它不能根据多个标准进行routing。

在我们的messaging system中，我们可能希望不仅subscribe based on the routing key的队列，而且还based on the source which produced the message。 您可能从[syslog](http://en.wikipedia.org/wiki/Syslog) unix工具知道这个概念，该工具根据严重性(info/warn/crit...)和工具(auth/cron/kern...)来routes logs。 我们的例子比这个简单一点。

这个例子会给我们很大的灵活性 - 我们可能想要listen来自'cron'的严重错误，而且还要听取来自'kern'的所有日志。

为了在我们的日志系统中实现这种灵活性，我们需要了解更复杂的topic exchange。

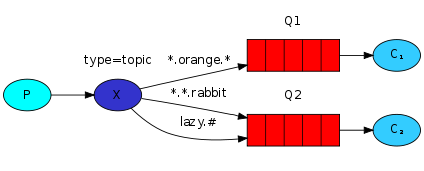
Topic exchange

发送到topic exchange的消息不能有任意的routing\_key - 它必须是由点(dots)分隔的单词列表。 单词可以是任意的，但通常它们指定了与该消息相关的一些功能。 一些有效的routing key例子: "stock.usd.nyse", "nyse.vmw", "quick.orange.rabbit". 只要您愿意，routing key中可以有多个字，最多255个字节。

binding key也必须是相同的形式。 topic exchange背后的逻辑类似于direct  - 使用特定routing key发送的消息将被传递到与匹配binding key的所有queues。 但是binding keys有两个重要的特殊情况：

* \* (星号) 可以替代一个word.
* # (hash) 可以替代0个或多个words.

在一个简单的例子中可以解释:



在这个例子中，我们将发送所有描述动物的消息。 messages将使用由三个words（两个dots）组成的routing key发送。 routing key中的第一个word将描述speed，第二个colour和第三个种species: "<speed>.<colour>.<species>".

我们创建了三个bindings：Q1绑定了binding key "\*.orange.\*"，Q2绑定了"\*.\*.rabbit" and "lazy.#".

这些bindings 可以概括为:

* Q1 对所有orange动物感兴趣.
* Q2 希望听到关于rabbits的一切，以及关于lazy animals的一切.

将routing key设置为"quick.orange.rabbit"的message将delivered到两个queues。 消息"lazy.orange.elephant"也会去他们两个。 另一方面，"quick.orange.fox"只会进入第一个queue，而"lazy.brown.fox"只会进入第二个queue。"lazy.pink.rabbit"只会传递到第二个queue一次，即使它匹配了两个bindings。"quick.brown.fox"不匹配任何binding，因此将被discarded。

如果我们违反我们的contract并发送带有一个或四个单词的消息，如"orange" or "quick.orange.male.rabbit"，会发生什么情况？ 那么，这些消息将不匹配任何bindings，并会discarded。

另一方面，"lazy.orange.male.rabbit"即使有四个单词，也会匹配最后一个绑定，并将传递到第二个queue。

**Topic exchange**

Topic exchange 功能强大，可以具有其他exchanges一样的行为。

当使用"#" (hash) binding key绑定queue时，它将接收所有消息，而不管routing key如何 - 就像在 fanout exchange中一样。

当在bindings中没有使用特殊字符"\*" (star) and "#" (hash)时，topic exchange将像direct exchange一样。

Putting it all together

我们将在我们的messaging system中使用topic exchange。 我们首先假定routing keys将利用通配符(wildcards)和hash tag。

代码几乎与 [previous tutorial](http://www.rabbitmq.com/tutorials/tutorial-four-spring-amqp.html)中的代码相同。

首先让我们在tut5包的Tut5Config.java中配置一些配置文件和bean：

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Profile;

@Profile({"tut5","topics"})

@Configuration

public class Tut5Config {

@Bean

public TopicExchange topic() {

return new TopicExchange("tut.topic");

}

@Profile("receiver")

private static class ReceiverConfig {

@Bean

public Tut5Receiver receiver() {

return new Tut5Receiver();

}

@Bean

public Queue autoDeleteQueue1() {

return new AnonymousQueue();

}

@Bean

public Queue autoDeleteQueue2() {

return new AnonymousQueue();

}

@Bean

public Binding binding1a(TopicExchange topic,

Queue autoDeleteQueue1) {

return BindingBuilder.bind(autoDeleteQueue1)

.to(topic)

.with("\*.orange.\*");

}

@Bean

public Binding binding1b(TopicExchange topic,

Queue autoDeleteQueue1) {

return BindingBuilder.bind(autoDeleteQueue1)

.to(topic)

.with("\*.\*.rabbit");

}

@Bean

public Binding binding2a(TopicExchange topic,

Queue autoDeleteQueue2) {

return BindingBuilder.bind(autoDeleteQueue2)

.to(topic)

.with("lazy.#");

}

}

@Profile("sender")

@Bean

public Tut5Sender sender() {

return new Tut5Sender();

}

}

我们设置我们的配置文件来执行topics作为"tut5" or "topics"的选择。 然后我们为TopicExchange创建了这个bean。 "receiver"配置文件是定义我们的receiver的ReceiverConfig，如前一个教程中的两个AnonymousQueues以及使用topics语法的主题的绑定。 我们还创建了"sender"”配置文件，以创建Tut5Sender类。

Tut5Sender再次使用@RabbitListener接收来自相应topics的messages。

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

import org.springframework.util.StopWatch;

public class Tut5Receiver {

@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);

}

}

}

}

Tut5Sender.java 代码如下:

import org.springframework.amqp.core.TopicExchange;

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

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

import org.springframework.scheduling.annotation.Scheduled;

public class Tut5Sender {

@Autowired

private RabbitTemplate template;

@Autowired

private TopicExchange topic;

private int index;

private int count;

private final String[] keys = {"quick.orange.rabbit",

"lazy.orange.elephant", "quick.orange.fox",

"lazy.brown.fox", "lazy.pink.rabbit", "quick.brown.fox"};

@Scheduled(fixedDelay = 1000, initialDelay = 500)

public void send() {

StringBuilder builder = new StringBuilder("Hello to ");

if (++this.index == keys.length) {

this.index = 0;

}

String key = keys[this.index];

builder.append(key).append(' ');

builder.append(Integer.toString(++this.count));

String message = builder.toString();

template.convertAndSend(topic.getName(), key, message);

System.out.println(" [x] Sent '" + message + "'");

}

}

按照 [Tutorial 1](http://www.rabbitmq.com/tutorials/tutorial-one-spring-amqp.html)中的描述编译并运行示例。或者，如果您一直遵循教程，则只需执行以下操作：

To build the project:

mvn clean package

用正确的配置文件执行sender and receiver，使用正确的参数执行jar：

java -jar target/rabbit-tutorials-1.7.1.RELEASE.jar

--spring.profiles.active=topics,receiver

--tutorial.client.duration=60000

java -jar target/rabbit-tutorials-1.7.1.RELEASE.jar

--spring.profiles.active=topics,sender

--tutorial.client.duration=60000

sender的输出将如下所示：

Ready ... running for 60000ms

[x] Sent 'Hello to lazy.orange.elephant 1'

[x] Sent 'Hello to quick.orange.fox 2'

[x] Sent 'Hello to lazy.brown.fox 3'

[x] Sent 'Hello to lazy.pink.rabbit 4'

[x] Sent 'Hello to quick.brown.fox 5'

[x] Sent 'Hello to quick.orange.rabbit 6'

[x] Sent 'Hello to lazy.orange.elephant 7'

[x] Sent 'Hello to quick.orange.fox 8'

[x] Sent 'Hello to lazy.brown.fox 9'

[x] Sent 'Hello to lazy.pink.rabbit 10'

并且receiver将以下面的输出作出响应：

instance 1 [x] Received 'Hello to lazy.orange.elephant 1'

instance 2 [x] Received 'Hello to lazy.orange.elephant 1'

instance 2 [x] Done in 2.005s

instance 1 [x] Done in 2.005s

instance 1 [x] Received 'Hello to quick.orange.fox 2'

instance 2 [x] Received 'Hello to lazy.brown.fox 3'

instance 1 [x] Done in 2.003s

instance 2 [x] Done in 2.003s

instance 1 [x] Received 'Hello to lazy.pink.rabbit 4'

instance 2 [x] Received 'Hello to lazy.pink.rabbit 4'

instance 1 [x] Done in 2.006s

instance 2 [x] Done in 2.006s

玩这些程序玩得开心。 请注意，代码没有对routing or binding keys,作任何假设，您可能需要使用两个以上的routing key parameters。

（[Tut5Receiver.java source](https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/spring-amqp/src/main/java/org/springframework/amqp/tutorials/tut5/Tut5Receiver.java) and [Tut5Sender.java source](https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/spring-amqp/src/main/java/org/springframework/amqp/tutorials/tut5/Tut5Sender.java)的完整源代码，配置位于 [Tut5Config.java source](https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/spring-amqp/src/main/java/org/springframework/amqp/tutorials/tut5/Tut5Config.java)源代码中。）

接下来，在[tutorial 6](http://www.rabbitmq.com/tutorials/tutorial-six-spring-amqp.html)中，了解如何远程过程调用并返回消息