**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)**

**Prerequisites**

This tutorial assumes RabbitMQ is [installed](http://www.rabbitmq.com/download.html) and running on localhost on standard port (5672). In case you use a different host, port or credentials, connections settings would require adjusting.

**Where to get help**

If you're having trouble going through this tutorial you can [contact us](https://groups.google.com/forum/#!forum/rabbitmq-users) through the mailing list.

In the [previous tutorial](http://www.rabbitmq.com/tutorials/tutorial-four-spring-amqp.html) we improved our messaging flexibility. Instead of using a fanoutexchange only capable of dummy broadcasting, we used a direct one, and gained a possibility of selectively receiving the message based on the routing key.

Although using the direct exchange improved our system, it still has limitations - it can't do routing based on multiple criteria.

In our messaging system we might want to subscribe to not only queues based on the routing key, but also based on the source which produced the message. You might know this concept from the [syslog](http://en.wikipedia.org/wiki/Syslog) unix tool, which routes logs based on both severity (info/warn/crit...) and facility (auth/cron/kern...). Our example is a little simpler than this.

That example would give us a lot of flexibility - we may want to listen to just critical errors coming from 'cron' but also all logs from 'kern'.

To implement that flexibility in our logging system we need to learn about a more complex topic exchange.

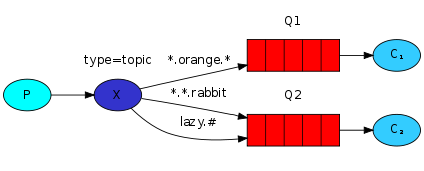
Topic exchange

Messages sent to a topic exchange can't have an arbitrary routing\_key - it must be a list of words, delimited by dots. The words can be anything, but usually they specify some features connected to the message. A few valid routing key examples: "stock.usd.nyse", "nyse.vmw", "quick.orange.rabbit". There can be as many words in the routing key as you like, up to the limit of 255 bytes.

The binding key must also be in the same form. The logic behind the topic exchange is similar to a direct one - a message sent with a particular routing key will be delivered to all the queues that are bound with a matching binding key. However there are two important special cases for binding keys:

* \* (star) can substitute for exactly one word.
* # (hash) can substitute for zero or more words.

It's easiest to explain this in an example:



In this example, we're going to send messages which all describe animals. The messages will be sent with a routing key that consists of three words (two dots). The first word in the routing key will describe speed, second a colour and third a species: "<speed>.<colour>.<species>".

We created three bindings: Q1 is bound with binding key "\*.orange.\*" and Q2 with "\*.\*.rabbit" and "lazy.#".

These bindings can be summarised as:

* Q1 is interested in all the orange animals.
* Q2 wants to hear everything about rabbits, and everything about lazy animals.

A message with a routing key set to "quick.orange.rabbit" will be delivered to both queues. Message "lazy.orange.elephant" also will go to both of them. On the other hand "quick.orange.fox" will only go to the first queue, and "lazy.brown.fox" only to the second. "lazy.pink.rabbit" will be delivered to the second queue only once, even though it matches two bindings. "quick.brown.fox" doesn't match any binding so it will be discarded.

What happens if we break our contract and send a message with one or four words, like "orange" or "quick.orange.male.rabbit"? Well, these messages won't match any bindings and will be lost.

On the other hand "lazy.orange.male.rabbit", even though it has four words, will match the last binding and will be delivered to the second queue.

**Topic exchange**

Topic exchange is powerful and can behave like other exchanges.

When a queue is bound with "#" (hash) binding key - it will receive all the messages, regardless of the routing key - like in fanout exchange.

When special characters "\*" (star) and "#" (hash) aren't used in bindings, the topic exchange will behave just like a direct one.

Putting it all together

We're going to use a topic exchange in our messaging system. We'll start off with a working assumption that the routing keys will take advantage of both wildcards and a hash tag.

The code is almost the same as in the [previous tutorial](http://www.rabbitmq.com/tutorials/tutorial-four-spring-amqp.html).

First lets configure some profiles and beans in the Tut5Config.java of the tut5 package:

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();

}

}

We setup our profiles for executing the topics as the choice of "tut5" or "topics". We then created the bean for our TopicExchange. The "receiver" profile is the ReceiverConfig defining our receiver, two AnonymousQueues as in the previous tutorial and the bindings for the topics utilizing the topic syntax. We also create the "sender" profile as the creation of the Tut5Sender class.

The Tut5Sender again uses the @RabbitListener to receive messages from the respective topics.

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);

}

}

}

}

The code for Tut5Sender.java:

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 + "'");

}

}

Compile and run the examples as described in [Tutorial 1](http://www.rabbitmq.com/tutorials/tutorial-one-spring-amqp.html). Or if you have been following along through the tutorials you only need to do the following:

To build the project:

mvn clean package

To execute the sender and receiver with the correct profiles execute the jar with the correct parameters:

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

The output from the sender will look something like:

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'

And the receiver will respond with the following output:

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

Have fun playing with these programs. Note that the code doesn't make any assumption about the routing or binding keys, you may want to play with more than two routing key parameters.

(Full source code for [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). The configuration is in [Tut5Config.java source](https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/spring-amqp/src/main/java/org/springframework/amqp/tutorials/tut5/Tut5Config.java). )

Next, find out how to do a round trip message as a remote procedure call in [tutorial 6](http://www.rabbitmq.com/tutorials/tutorial-six-spring-amqp.html)