

1 ["Hello World!"](#)

The simplest thing that does *something*

[Python](#) [Java](#) [Ruby](#) [PHP](#) [C#](#) [JavaScript](#) [Go](#) [Elixir](#) [Objective-C](#) [Swift](#)
[Spring AMQP](#)

2 [Work queues](#)

Distributing tasks among workers (the [competing consumers pattern](#))

[Python](#) [Java](#) [Ruby](#) [PHP](#) [C#](#) [JavaScript](#) [Go](#) [Elixir](#) [Objective-C](#) [Swift](#)
[Spring AMQP](#)

3 [Publish/Subscribe](#)

Sending messages to many consumers at once

[Python](#) [Java](#) [Ruby](#) [PHP](#) [C#](#) [JavaScript](#) [Go](#) [Elixir](#) [Objective-C](#) [Swift](#)
[Spring AMQP](#)

4 [Routing](#)

Receiving messages selectively

[Python](#) [Java](#) [Ruby](#) [PHP](#) [C#](#) [JavaScript](#) [Go](#) [Elixir](#) [Objective-C](#) [Swift](#)
[Spring AMQP](#)

5 [Topics](#)

Receiving messages based on a pattern (topics)

[Python](#) [Java](#) [Ruby](#) [PHP](#) [C#](#) [JavaScript](#) [Go](#) [Elixir](#) [Objective-C](#) [Swift](#)
[Spring AMQP](#)

6 [RPC](#)

[Request/reply pattern](#) example

[Python](#) [Java](#) [Ruby](#) [PHP](#) [C#](#) [JavaScript](#) [Go](#) [Elixir](#) [Spring AMQP](#)

Topics

(using the Java client)

In the [previous tutorial](#) we improved our logging system. Instead of using a `fanout` exchange only capable of dummy broadcasting, we used a `direct` one, and gained a possibility of selectively receiving the logs.

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

In our logging system we might want to subscribe to not only logs based on severity, but also based on the source which emitted the log. You might know this concept from the [syslog](#) unix tool, which routes logs based on both severity (`info/warn/crit...`) and facility (`auth/cron/kern...`).

That 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 in our logging system we need to learn about a more complex `topic` exchange.

Prerequisites

This tutorial assumes RabbitMQ is [installed](#) 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](#) through the mailing list.

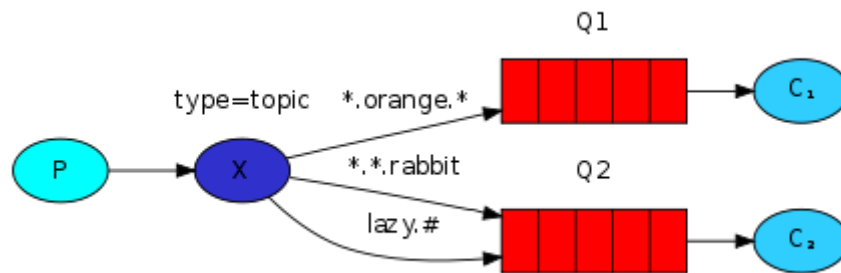
Topic exchange

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 logging system. We'll start off with a working assumption that the routing keys of logs will have two words: "<facility>.<severity>".

The code is almost the same as in the [previous tutorial](#).

The code for `EmitLogTopic.java`:

```
import com.rabbitmq.client.*;

import java.io.IOException;

public class EmitLogTopic {

    private static final String EXCHANGE_NAME = "topic_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, "topic");

    String routingKey = getRouting(argv);
    String message = getMessage(argv);

    channel.basicPublish(EXCHANGE_NAME, routingKey, null, message.getBytes());
    System.out.println(" [x] Sent '" + routingKey + "':" + message + "'");

    connection.close();
}
//...
}
```

The code for ReceiveLogsTopic.java:

```
import com.rabbitmq.client.*;

import java.io.IOException;

public class ReceiveLogsTopic {
    private static final String EXCHANGE_NAME = "topic_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, "topic");
        String queueName = channel.queueDeclare().getQueue();
```

```

if (argv.length < 1) {
    System.err.println("Usage: ReceiveLogsTopic [binding_key]...");
    System.exit(1);
}

for (String bindingKey : argv) {
    channel.queueBind(queueName, EXCHANGE_NAME, bindingKey);
}

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 '" + envelope.getRoutingKey() + "':'" +
message + "'");
    }
};
channel.basicConsume(queueName, true, consumer);
}
}

```

Compile and run the examples, including the classpath as in [Tutorial 1](#) - on Windows, use %CP%.

To compile:

```
javac -cp $CP ReceiveLogsTopic.java EmitLogTopic.java
```

To receive all the logs:

```
java -cp $CP ReceiveLogsTopic "#"
```

To receive all logs from the facility "kern":

```
java -cp $CP ReceiveLogsTopic "kern.*"
```

Or if you want to hear only about "critical" logs:

```
java -cp $CP ReceiveLogsTopic "*.critical"
```

You can create multiple bindings:

```
java -cp $CP ReceiveLogsTopic "kern.*" "*.critical"
```

And to emit a log with a routing key "kern.critical" type:

```
java -cp $CP EmitLogTopic "kern.critical" "A critical kernel error"
```

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 [EmitLogTopic.java](#) and [ReceiveLogsTopic.java](#))

Next, find out how to do a round trip message as a remote procedure call in [tutorial 6](#)

Build software people love at **SpringOne Platform**

REGISTER NOW

Copyright © 2007-Present Pivotal Software, Inc. All rights reserved. Terms of Use, Privacy and Trademark Guidelines

