# 路由（Routing）

在前面的教程中，我们建立了一个简单的日志记录系统。我们能够向许多接收器广播日志消息。在本教程中，我们将添加一个功能，我们将让它能够只订阅一个消息的子集。例如，我们能够只引导关键错误的消息到**日志文件**中(为了节省磁盘空间)。但是仍然能够输出所有的消息到控制台。

## 1、Bindings

在前面的例子中，我们已经创建绑定。

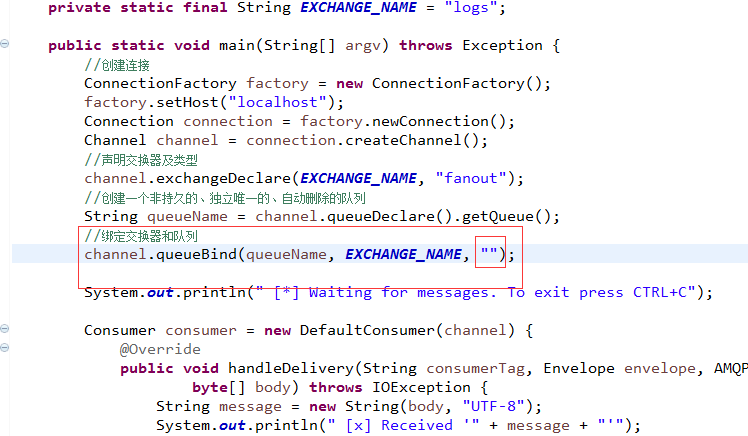
channel**.**queueBind**(**queueName**,**EXCHANGE\_NAME**,**""**);**

绑定是一个交换器和队列之间的关系。可以简单的理解为：这个队列对交换器(exchange)的消息很感兴趣。

绑定可以携带额外的routingkey参数。为了避免与basic\_publish的参数混淆，我们称之为binding key.

channel**.**queueBind**(**queueName**,**EXCHANGE\_NAME**,**"black"**);**

在之前的教程中的绑定：



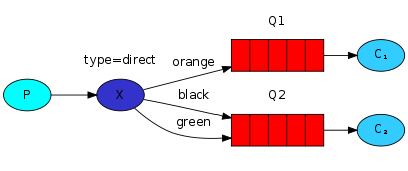
binding key的含义取决于交换器(exchange)的类型。之前使用的fanout exchange 基本上完全忽略了binding key 的值。

## 2、直接交换（Direct exchange）

在之前的教程当中，我们的日志系统是广播所有的消息给所有的消费者。我们希望扩展成允许基于消息的严重程度来过滤消息。例如，我们希望能有个程序，将仅仅接受了关键错误的日志消息写到磁盘中，而不要因为警告或普通信息日志消息而浪费磁盘空间。我们使用过的fanout exchange 并没有给我们太多的灵活性，它只是盲目地广播消息。

我们将使用直接交换（Direct exchange）取而代之，直接交换（Direct exchange）的路由算法是很简单的：

一个消息（message）到队列（queue），该队列(queue)的binding key 刚好与消息(message)的 routing key 相匹配。

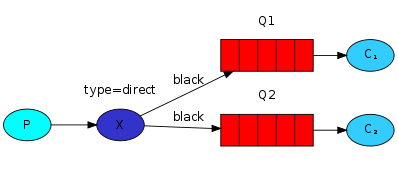


由上图可以看到，直接交换(Direct exchange)与两个队列(queue)绑定，第一个队列绑定了一个orange的binding key。第二个队列有两个绑定，一个是black,一个是green.

由此可以知道，当一个消息（message）的routing key是orange 将会被发布到队列Q1,

消息的routing key 是black 或者green 的，将会被转发到队列Q2。

## 3、多重绑定（Multiple bindings）



此前，最多看到是一个队列有两个binding key与之对应。现在是一个binding key 对应多个队列（queue）。

用相同的binding key绑定多个队列是完全合法的。在我们的例子中，我们还可以增加一个binding key 为black 的绑定直接交换（Direct exchange）X和队列Q1.

在这种情况下，直接交换 (Direct exchange )将会表现的和 fanout exchange 一样，将广播消息到所有匹配的队列中。

routing key为black 的消息，将会被发送到队列Q1 和队列Q2。

## 4、Emitting logs

我们将为我们的日志系统使用这个模型。将fanout exchange 换成直接交换(Direct exchange)，将消息发送到直接交换（Direct exchange）中。

我们将提供日志严重性作为路由键(routing key).这样接收程序就能够选择它希望接收的严重性。

一如既往，创建一个交换：类型为direct

channel**.**exchangeDeclare**(**EXCHANGE\_NAME**,**"direct"**);**

准备发送消息：

channel**.**basicPublish**(**EXCHANGE\_NAME**,**severity**,null,**message**.**getBytes**());**

为了简化，我们假设严重性分为“info”,”warning”,”error”。

## 5、订阅 Subscribing

接收消息将和之前的教程中一样，但是有一个例外是，我们将为每个我们感兴趣的严重性(“info”,”warning”,”error”)创建新的绑定。

StringqueueName**=**channel**.**queueDeclare**().**getQueue**();**

**for(**Stringseverity**:**argv**){**

channel**.**queueBind**(**queueName**,**EXCHANGE\_NAME**,**severity**);**

**}**

## 6、代码实现



### 6.1 EmitLogDirect

**package** com.symbol.rabbitmq.routing;

**import**java.util.ArrayList;

**import**java.util.List;

**import** java.util.concurrent.TimeoutException;

**import** com.rabbitmq.client.Channel;

**import** com.rabbitmq.client.Connection;

**import** com.rabbitmq.client.ConnectionFactory;

**publicclass** EmitLogDirect {

**privatestaticfinal** String ***EXCHANGE\_NAME*** = "direct\_logs";

// severity list

**privatestaticfinal** String[] ***LOG\_SEVERITY*** = { "info", "warning", "error" };

**publicstaticvoid** main(String[] argv) **throws** java.io.IOException, TimeoutException {

ConnectionFactory factory = **new** ConnectionFactory();

factory.setHost("localhost");

Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

//创建并指定交换类型

channel.exchangeDeclare(***EXCHANGE\_NAME***, "direct");

// 发布消息

**for** (**int**i = 0; i< 10; i++) {

String severity = ***LOG\_SEVERITY***[(i%***LOG\_SEVERITY***.length)];

String[] ms = { "Ja0ck5-Log-..." +severity+"-"+ i };

String message = *getMessage*(ms);

channel.basicPublish(***EXCHANGE\_NAME***, severity, **null**, message.getBytes());

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

}

channel.close();

connection.close();

}

**publicstatic** String getMessage(String[] strings) {

**if** (strings.length< 1)

**return**"Hello World!";

**return***joinStrings*(strings, " ");

}

**publicstatic** String joinStrings(String[] strings, String delimiter) {

**int**length = strings.length;

**if** (length == 0)

**return**"";

StringBuilder words = **new** StringBuilder(strings[0]);

**for** (**int**i = 1; i<length; i++) {

words.append(delimiter).append(strings[i]);

}

**return**words.toString();

}

}

### 6.2 ReceiveLogsDirect

package com.symbol.rabbitmq.routing;

import com.rabbitmq.client.\*;

import java.io.IOException;

public class ReceiveLogsDirect {

private static final String EXCHANGE\_NAME = "direct\_logs";

// severity list

private static final String[] LOG\_SEVERITY = { "info", "warning", "error" };

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, "direct");

String queueName = channel.queueDeclare().getQueue();

if (LOG\_SEVERITY.length < 1) {

System.err.println("Usage: ReceiveLogsDirect [info] [warning] [error]");

System.exit(1);

}

for (String severity : LOG\_SEVERITY) {

channel.queueBind(queueName, EXCHANGE\_NAME, severity);

}

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);

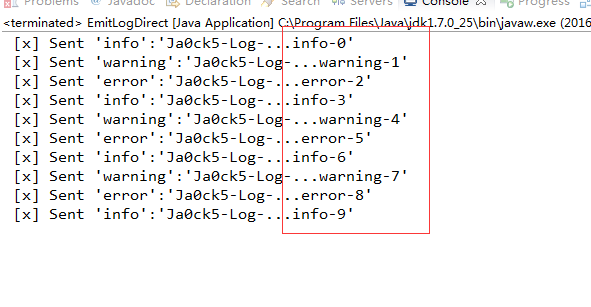
}

}

### 6.3 测试代码

开启接收端。

发送端发送消息



接收消息：根据不同的严重性severity .接收感兴趣的消息。

