# Topics

在前面的教程中，我们改进了我们的日志系统. 而不是使用一个只能虚拟广播的扇出交换(fanout exchange).使用了直接交换 (direct exchange)，并且增加了有选择地接收日志消息的可能性。虽然使用直接交换(direct exchange)改善了我们的系统，它仍然有局限性-不能够基于多个标准路由。

在我们的日志系统中，我们可能希望订阅不仅仅基于严重性的日志，而且能够基于发出的日志的来源。你可能从syslog UNIX工具知道这个概念，路由日志基于严重程度(severity[info][warning][critical]) 和 设备(facility[auth][ cron][kern])

我们也许想要监听不仅仅关于来自[cron]的关键错误，也要监听来自[kern]的所有的日志消息。这会给我们很大的伸缩性.

要实现在我们的日志系统中，我们需要了解一个更复杂的主题交换(topic exchange)

## 1、主题交换（Topic exchange）

消息发送到一个主题交换(topic exchange),不能够用随意的 routing\_key  它必须是单词列表，由点 . 分隔，但是单词是可以任意的，通常我们都是指定一些特性与消息相关联。这里举出几个有效的 routing key 的例子：

 "stock.usd.nyse"

"nyse.vmw"

"quick.orange.rabbit"

routing key 的上限是 255(bytes) 个字节.

binding key 也必须是以相同的形式构成。

主题交换(topic exchange) 的逻辑与 直接交换(direct exchange)是很相似的：

附带了特定的routing key 的消息会被发送到所有绑定了相匹配的 binding key 的队列(queue)中。

然而，还有两种重要而特殊的 binding keys

* \* 可以代替一个词
* # 可以代替零个或多个词



在这个例子中，我们将发送所有描述动物的信息。消息将会被发送并附带由三个词(两个点 . )组成的 routing key。routing key 中 第一个词描述的是 速度,第二个描述的颜色，第三个描述的种类。

我们创建了三个绑定(binding)：

队列Q1 绑定了 “\*.orange.\* ” 也就是中间匹配颜色 orange (橙色)

队列 Q2 绑定了 “\*.\*.rabbit” 也就是最后一个词匹配种类 rabbit (兔子)

还绑定了 “lazy.#” 即匹配形如 lazy. 开头的 routing key “lazy.abc” “lazy.abc.def”

一个附带有 “quick.orange.rabbit” routing key 的消息

匹配 队列 Q1 的 “\*.orange.\*” 从而转发到队列 Q1

匹配 队列Q2 的 “\*.\*.rabbit” 从而转发到 队列Q2

一个附带有 “lazy.orange.elephant” routing key 的消息

匹配 队列 Q1 的 “\*.orange.\*” 从而转发到队列 Q1

匹配 队列Q2 的 “lazy.#” 从而转发到 队列Q2

如果一个消息附带有 "quick.brown.fox" 不匹配任何一个 binding key 就会被丢弃。

主题交换(topic exchange)是非常强大的，它可以表现得和其它交换(exchange)一样。

如果队列仅仅绑定 “#” ，它将会接收所有的消息。

如果不使用 “\*” “#”，它就表现得和 直接交换(direct exchange)一样。

## 2、代码实现

### EmitLogTopic

**package** com.symbol.rabbitmq.topic;

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

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

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

**public** **class** EmitLogTopic {

**private** **static** **final** String ***EXCHANGE\_NAME*** = "topic\_logs";

**private** **static** **final** String[] ***ROUTING\_KEYS*** = { "Model.auth", "Model.kern", "Model.critical",

"View.auth","View.kern", "View.critical",

"Ctl.auth", "Ctl.kern", "Ctl.critical",

"Ctl.auth.Model", "Ctl.kern.View", "Ctl.critical.Ctl",

"Model.auth.Model", "Model.kern.View", "Model.critical.Ctl",

"View.auth.Model.haha.hehe", "View.kern.View.haha.hehe", "View.critical.Ctl.haha.hehe"};

**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");

**for** (**int** i= 0; i< ***ROUTING\_KEYS***.length; i++) {

String message = *getLogMessage*(***ROUTING\_KEYS***[i],i);

channel.basicPublish(***EXCHANGE\_NAME***, ***ROUTING\_KEYS***[i], **null**, message.getBytes());

System.***out***.println(" [x] Sent '" + ***ROUTING\_KEYS***[i] + "':'" + message + "'");

}

connection.close();

}

// ...

**private** **static** String getLogMessage(String rk,**int** idx) {

**return** "facility source: ["+rk+"]"+"----- " + idx;

}

**public** **static** String getMessage(String[] strings) {

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

**return** "Hello World!";

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

}

**public** **static** 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();

}

}

### ReceiveLogsTopic

package com.symbol.rabbitmq.topic;

import com.rabbitmq.client.\*;

import java.io.IOException;

public class ReceiveLogsTopic {

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

private static final String[] ROUTING\_KEYS = { "\*.auth", "kern.#", "\*.critical.\*",

"View.\*", "\*.kern.#", "View.critical.Ctl"};

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 (ROUTING\_KEYS.length < 1) {

System.err.println("Usage: ReceiveLogsTopic [binding\_key]...");

System.exit(1);

}

for (String bindingKey : ROUTING\_KEYS) {

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 + "'");

}

};

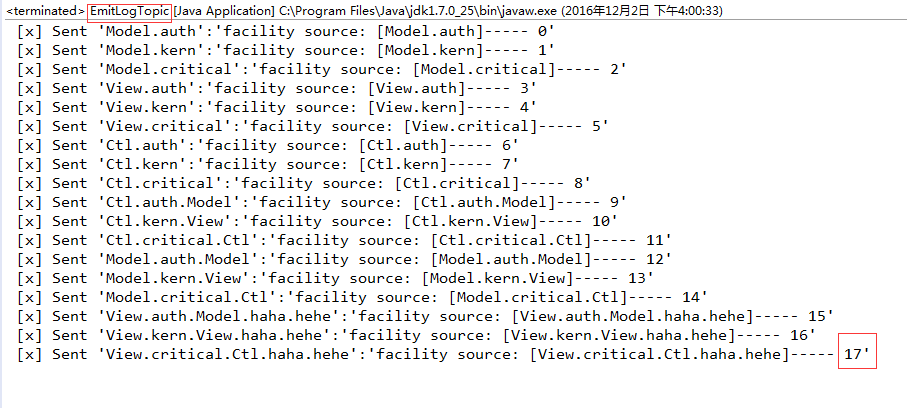
channel.basicConsume(queueName, true, consumer);

}

}

3、测试结果

发送端: 发送了 18 条数据



接收端：只接收 消息的 routing key 匹配 binding key 的消息

发送端序号为 2,8,9,12,15 的消息没有被接收，因为它们的 routing key 分别为

2 Model.critical

8 Ctl.critical

9 Ctl.auth.Model

12 Model.auth.Model

15 View.auth.Model.haha.hehe

并没有与之匹配的规则 （  
"\*.auth",

"kern.#",

"\*.critical.\*",

"View.\*",

"\*.kern.#",

"View.critical.Ctl"

）

