

# 0. 学习目标

---

- 能够说出什么是消息队列
- 能够安装RabbitMQ
- 能够编写RabbitMQ的入门程序
- 能够说出RabbitMQ的5种模式特征
- 能够使用SpringBoot整合RabbitMQ

## 1. 消息队列概述

---

### 1.1. 消息队列MQ

---

MQ全称为Message Queue，消息队列是应用程序和应用程序之间的通信方法。

- 为什么使用MQ

在项目中，可将一些无需即时返回且耗时的操作提取出来，进行**异步处理**，而这种异步处理的方式大大的节省了服务器的请求响应时间，从而**提高了系统的吞吐量**。

- 开发中消息队列通常有如下应用场景：

#### 1、任务异步处理

将不需要同步处理的并且耗时长操作由消息队列通知消息接收方进行异步处理。提高了应用程序的响应时间。

#### 2、应用程序解耦合

MQ相当于一个中介，生产方通过MQ与消费方交互，它将应用程序进行解耦合。

### 1.2. AMQP 和 JMS

---

MQ是消息通信的模型；实现MQ的大致有两种主流方式：AMQP、JMS。

#### 1.2.1. AMQP

AMQP是一种协议，更准确的说是一种binary wire-level protocol（链接协议）。这是其与JMS的本质差别，AMQP不从API层进行限定，而是直接定义网络交换的数据格式。

#### 1.2.2. JMS

JMS即Java消息服务（JavaMessage Service）应用程序接口，是一个Java平台中关于面向消息中间件（MOM）的API，用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。

### 1.2.3. AMQP 与 JMS 区别

- JMS是定义了统一的接口，来对消息操作进行统一；AMQP是通过规定协议来统一数据交互的格式
- JMS限定了必须使用Java语言；AMQP只是协议，不规定实现方式，因此是跨语言的。
- JMS规定了两种消息模式；而AMQP的消息模式更加丰富

## 1.3. 消息队列产品

---

市场上常见的消息队列有如下：

- ActiveMQ：基于JMS
- ZeroMQ：基于C语言开发
- RabbitMQ：基于AMQP协议，erlang语言开发，稳定性好
- RocketMQ：基于JMS，阿里巴巴产品
- Kafka：类似MQ的产品；分布式消息系统，高吞吐量

## 1.4. RabbitMQ

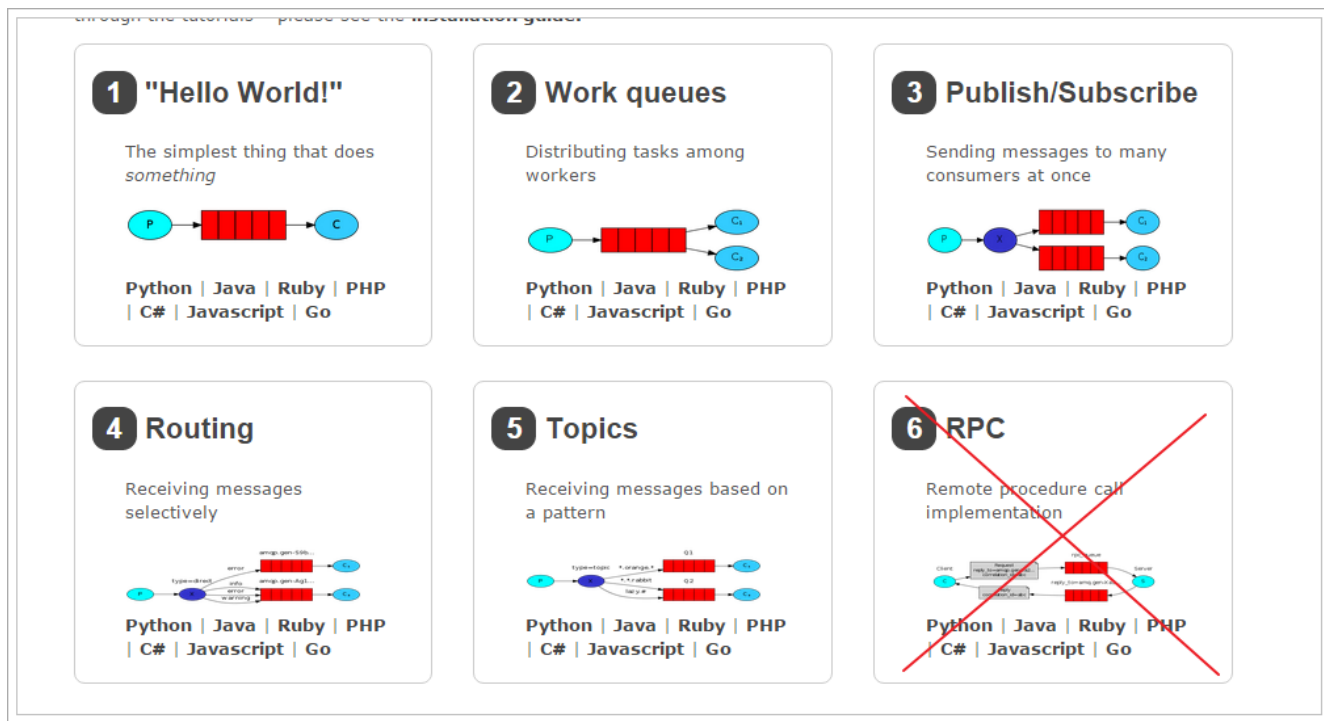
---

RabbitMQ是由erlang语言开发，基于AMQP（Advanced Message Queue 高级消息队列协议）协议实现的消息队列，它是一种应用程序之间的通信方法，消息队列在分布式系统开发中应用非常广泛。

RabbitMQ官方地址：<http://www.rabbitmq.com/>

RabbitMQ提供了6种模式：简单模式，work模式，Publish/Subscribe发布与订阅模式，Routing路由模式，Topics主题模式，RPC远程调用模式（远程调用，不太算MQ；不作介绍）；

官网对应模式介绍：<https://www.rabbitmq.com/getstarted.html>



## 2. 安装及配置RabbitMQ

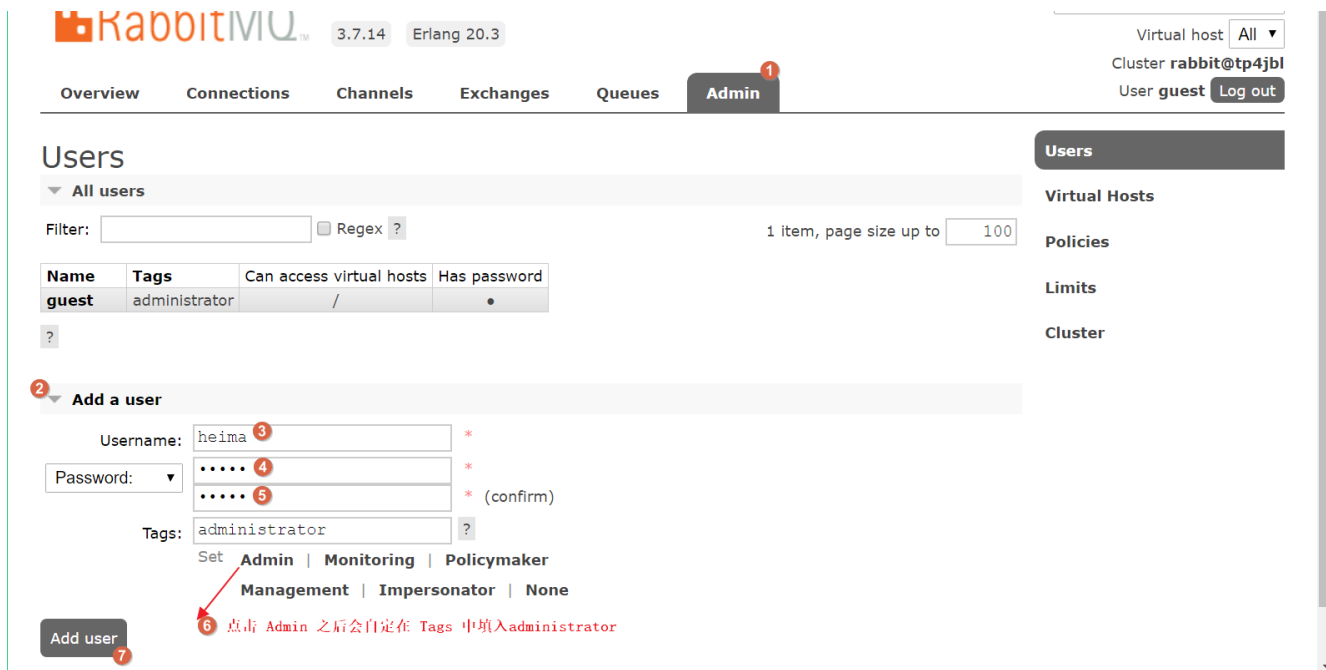
### 2.1. 安装说明

详细查看 [资料/软件/安装windows RabbitMQ.pdf](#) 文档。

### 2.2. 用户以及Virtual Hosts配置

#### 2.2.1. 用户角色

RabbitMQ在安装好后，可以访问<http://localhost:15672>；其自带了guest/guest的用户名和密码；如果需要创建自定义用户；那么也可以登录管理界面后，如下操作：



### 角色说明：

#### 1、超级管理员(administrator)

可登陆管理控制台，可查看所有的信息，并且可以对用户，策略(policy)进行操作。

#### 2、监控者(monitoring)

可登陆管理控制台，同时可以查看rabbitmq节点的相关信息(进程数，内存使用情况，磁盘使用情况等)

#### 3、策略制定者(polycymaker)

可登陆管理控制台，同时可以对policy进行管理。但无法查看节点的相关信息(上图红框标识的部分)。

#### 4、普通管理者(management)

仅可登陆管理控制台，无法看到节点信息，也无法对策略进行管理。

#### 5、其他

无法登陆管理控制台，通常就是普通的生产者和消费者。

## 2.2.2. Virtual Hosts配置

像mysql拥有数据库的概念并且可以指定用户对库和表等操作的权限。RabbitMQ也有类似的权限管理；在RabbitMQ中可以虚拟消息服务器Virtual Host，每个Virtual Hosts相当于一个相对独立的RabbitMQ服务器，每个Virtual Host之间是相互隔离的。exchange、queue、message不能互通。相当于mysql的db。Virtual Name一般以/开头。

### 1. 创建Virtual Hosts

RabbitMQ

3.7.14Erlang 20.3

Virtual hostAllCluster rabbit@tp4jblUser guestLog out

OverviewConnectionsChannelsExchangesQueuesAdmin

Virtual Hosts

All virtual hosts

Filter:  ☐ Regex ?1 item, page size up to 100

| Overview |         |         | Messages |         |       | Network     |           | Message rates |               |
|----------|---------|---------|----------|---------|-------|-------------|-----------|---------------|---------------|
| Name     | Users ? | State   | Ready    | Unacked | Total | From client | To client | publish       | deliver / get |
| /        | guest   | running | 0        | 0       | 0     | 0B/s        | 0B/s      | 0.00/s        | 0.00/s        |

2

Add a new virtual host

Name: /itcast

Add virtual host

HTTP APIServer DocsTutorialsCommunity SupportCommunity SlackCommercial SupportPluginsGitHubChangelog

Users

Virtual Hosts 1

Policies

Limits

Cluster

## 2. 设置Virtual Hosts权限

RabbitMQ

3.7.14Erlang 20.3

Virtual hostAllCluster rabbit@tp4jblUser guestLog out

OverviewConnectionsChannelsExchangesQueuesAdmin

Virtual Hosts

All virtual hosts

Filter:  ☐ Regex ?2 Items, page size up to 100

| Overview |         |         | Messages |         |       | Network     |           | Message rates |               |
|----------|---------|---------|----------|---------|-------|-------------|-----------|---------------|---------------|
| Name     | Users ? | State   | Ready    | Unacked | Total | From client | To client | publish       | deliver / get |
| /        | guest   | running | 0        | 0       | 0     | 0B/s        | 0B/s      | 0.00/s        | 0.00/s        |
| /itcast  | guest   | running | NaN      | NaN     | NaN   |             |           |               |               |

点

2

Add a new virtual host

Name:

Add virtual host

HTTP APIServer DocsTutorialsCommunity SupportCommunity SlackCommercial SupportPluginsGitHubChangelog

Users

Virtual Hosts

Policies

Limits

Cluster

RabbitMQ

3.7.14Erlang 20.3

Virtual hostAllCluster rabbit@tp4jblUser guestLog out

OverviewConnectionsChannelsExchangesQueuesAdmin

Current permissions

| User  | Configure regexp | Write regexp | Read regexp |
|-------|------------------|--------------|-------------|
| guest | .*               | .*           | .*          |

Set permission

User heima

Configure regexp: .\*

Write regexp: .\*

Read regexp: .\*

Set permission

Users

Virtual Hosts

Policies

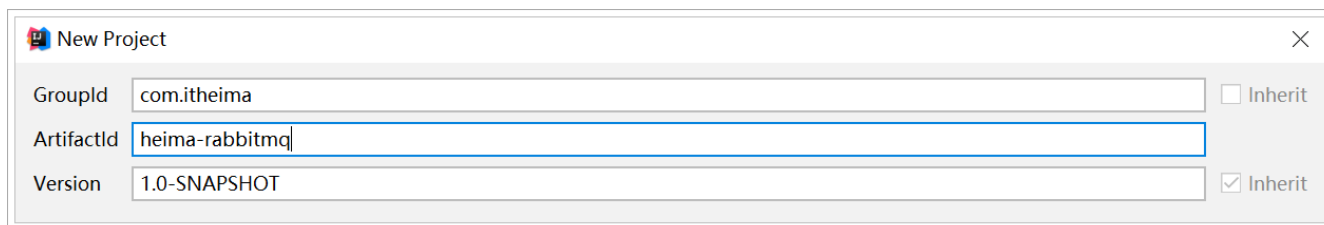
Limits

Cluster

## 3. RabbitMQ入门

### 3.1. 搭建示例工程

#### 3.1.1. 创建工程

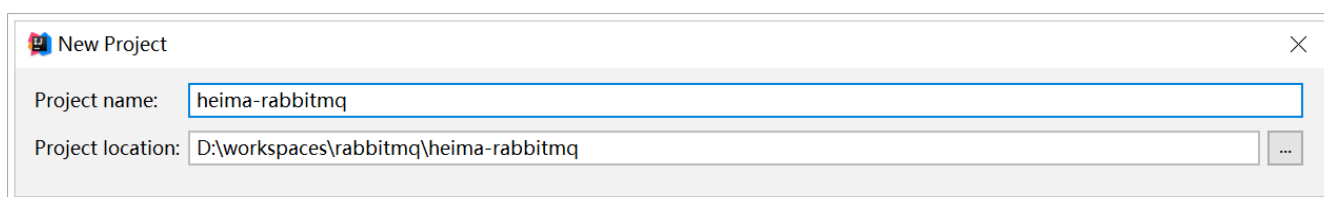


New Project

GroupId: com.itheima ☐ Inherit

ArtifactId: heima-rabbitmq

Version: 1.0-SNAPSHOT ☒ Inherit



New Project

Project name: heima-rabbitmq

Project location: D:\workspaces\rabbitmq\heima-rabbitmq

#### 3.1.2. 添加依赖

往zs-rabbitmq的pom.xml文件中添加如下依赖:

```
1      <dependency>
2          <groupId>com.rabbitmq</groupId>
3          <artifactId>amqp-client</artifactId>
4          <version>5.6.0</version>
5      </dependency>
```

### 3.2. 编写生产者

编写消息生产者com.ithouse.rabbitmq.simple.Producer

```
1 package com.ithouse.rabbitmq.simple;
2
3 import com.rabbitmq.client.Channel;
4 import com.rabbitmq.client.Connection;
5 import com.rabbitmq.client.ConnectionFactory;
6
7 public class Producer {
8
9     static final String QUEUE_NAME = "simple_queue";
10
11     public static void main(String[] args) throws Exception {
12         //创建连接工厂
13         ConnectionFactory connectionFactory = new ConnectionFactory();
```

```

14 //主机地址;默认为 localhost
15 connectionFactory.setHost("localhost");
16 //连接端口;默认为 5672
17 connectionFactory.setPort(5672);
18 //虚拟主机名称;默认为 /
19 connectionFactory.setVirtualHost("/ithouse");
20 //连接用户名;默认为guest
21 connectionFactory.setUsername("zs");
22 //连接密码;默认为guest
23 connectionFactory.setPassword("123");
24
25 //创建连接
26 Connection connection = connectionFactory.newConnection();
27
28 // 创建频道
29 Channel channel = connection.createChannel();
30
31 // 声明 (创建) 队列
32 /**
33  * 参数1: 队列名称
34  * 参数2: 是否定义持久化队列
35  * 参数3: 是否独占本次连接
36  * 参数4: 是否在不使用的时候自动删除队列
37  * 参数5: 队列其它参数
38  */
39 channel.queueDeclare(QUEUE_NAME, true, false, false, null);
40
41 // 要发送的信息
42 String message = "你好; 小兔子! ";
43 /**
44  * 参数1: 交换机名称, 如果没有指定则使用默认Default Exchage
45  * 参数2: 路由key,简单模式可以传递队列名称
46  * 参数3: 消息其它属性
47  * 参数4: 消息内容
48  */
49 channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
50 System.out.println("已发送消息: " + message);
51
52 // 关闭资源
53 channel.close();
54 connection.close();
55 }
56 }
57

```

在执行上述的消息发送之后; 可以登录rabbitMQ的管理控制台, 可以发现队列和其消息:

 3.7.14 Erlang 20.3

OverviewConnectionsChannelsExchanges**Queues**Admin

## Queues

▼ All queues (1)

Pagination

Page 1 of 1 - Filter:  ☐ Regex ?

| Overview     |                     |          |       | Messages |         |       | Message rates |               |     | +/- |
|--------------|---------------------|----------|-------|----------|---------|-------|---------------|---------------|-----|-----|
| Virtual host | Name                | Features | State | Ready    | Unacked | Total | incoming      | deliver / get | ack |     |
| /itcast      | <b>simple_queue</b> | D        | idle  | 1        | 0       | 1     | 0.00/s        |               |     |     |

 3.7.14 Erlang 20.3

OverviewConnectionsChannelsExchanges**Queues**Admin

Warning: getting messages from a queue is a destructive action. ?

Ack Mode:

Encoding:  ?

Messages:

**Get Message(s)**

Message 1

The server reported 0 messages remaining.

|                  |                |
|------------------|----------------|
| Exchange         | (AMQP default) |
| Routing Key      | simple_queue   |
| Redelivered      | o              |
| Properties       |                |
| Payload          | 你好; 小兔子!       |
| 21 bytes         |                |
| Encoding: string |                |

### 3.3. 编写消费者

抽取创建connection的工具类com.ithouse.rabbitmq.util.ConnectionUtil;

```

1 package com.ithouse.rabbitmq.util;
2
3 import com.rabbitmq.client.Connection;
4 import com.rabbitmq.client.ConnectionFactory;
5
6 public class ConnectionUtil {
7
8     public static Connection getConnection() throws Exception {
9         //创建连接工厂
10        ConnectionFactory connectionFactory = new ConnectionFactory();
11        //主机地址;默认为 localhost
12        connectionFactory.setHost("localhost");
13        //连接端口;默认为 5672
14        connectionFactory.setPort(5672);
15        //虚拟主机名称;默认为 /
16        connectionFactory.setVirtualHost("/ithouse");
17        //连接用户名;默认为guest
18        connectionFactory.setUsername("zs");
19        //连接密码;默认为guest
20        connectionFactory.setPassword("zs");
21
22        //创建连接
23        return connectionFactory.newConnection();
24    }
25
26 }
27

```

编写消息的消费者com.ithouse.rabbitmq.simple.Consumer

```

1 package com.ithouse.rabbitmq.simple;
2
3 import com.ithouse.rabbitmq.util.ConnectionUtil;
4 import com.rabbitmq.client.*;
5
6 import java.io.IOException;
7
8 public class Consumer {
9
10    public static void main(String[] args) throws Exception {
11        Connection connection = ConnectionUtil.getConnection();
12
13        // 创建频道
14        Channel channel = connection.createChannel();
15
16        // 声明 (创建) 队列
17        /**
18         * 参数1: 队列名称
19         * 参数2: 是否定义持久化队列
20         * 参数3: 是否独占本次连接
21         * 参数4: 是否在不使用的时候自动删除队列

```

```

22      * 参数5: 队列其它参数
23      */
24      channel.queueDeclare(Producer.QUEUE_NAME, true, false, false, null);
25
26      //创建消费者; 并设置消息处理
27      DefaultConsumer consumer = new DefaultConsumer(channel){
28          @Override
29          /**
30           * consumerTag 消息者标签, 在channel.basicConsume时候可以指定
31           * envelope 消息包的内容, 可从中获取消息id, 消息routingkey, 交换机, 消息和重传标志
(收到消息失败后是否需要重新发送)
32           * properties 属性信息
33           * body 消息
34           */
35           public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
36               //路由key
37               System.out.println("路由key为: " + envelope.getRoutingKey());
38               //交换机
39               System.out.println("交换机为: " + envelope.getExchange());
40               //消息id
41               System.out.println("消息id为: " + envelope.getDeliveryTag());
42               //收到的消息
43               System.out.println("接收到的消息为: " + new String(body, "utf-8"));
44           }
45       };
46      //监听消息
47      /**
48       * 参数1: 队列名称
49       * 参数2: 是否自动确认, 设置为true为表示消息接收到自动向mq回复接收到了, mq接收到回复会删除消
息, 设置为false则需要手动确认
50       * 参数3: 消息接收到后回调
51       */
52      channel.basicConsume(Producer.QUEUE_NAME, true, consumer);
53
54      //不关闭资源, 应该一直监听消息
55      //channel.close();
56      //connection.close();
57  }
58 }

```

### 3.4. 小结

上述的入门案例中其实使用的是如下的简单模式:



在上图的模型中，有以下概念：

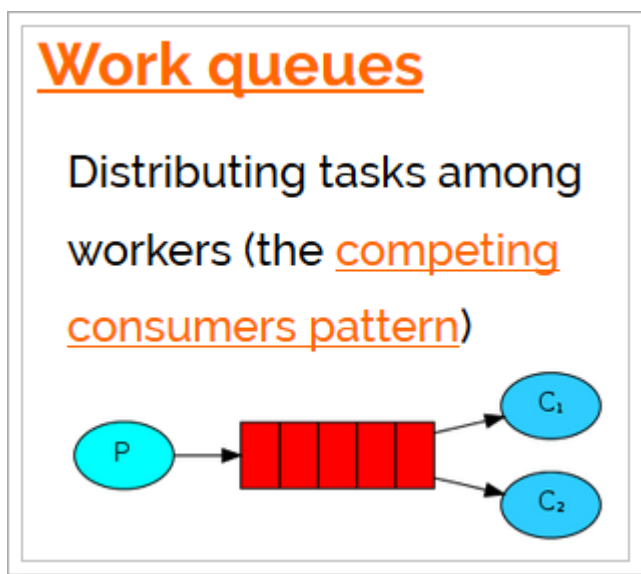
- P：生产者，也就是要发送消息的程序
- C：消费者：消息的接受者，会一直等待消息到来。
- queue：消息队列，图中红色部分。类似一个邮箱，可以缓存消息；生产者向其中投递消息，消费者从其中取出消息。

在rabbitMQ中消息者是一定要某个消息队列中去获取消息的

## 4. RabbitMQ工作模式

### 4.1. Work queues工作队列模式

#### 4.1.1. 模式说明



work queues 与入门程序的简单模式相比，多了一个或一些消费端，多个消费端共同消费同一个队列中的消息。

**应用场景：**对于任务过重或任务较多情况使用工作队列可以提高任务处理的速度。

#### 4.1.2. 代码

work queues 与入门程序的简单模式的代码是几乎一样的；可以完全复制，并复制多一个消费者进行多个消费者同时消费消息的测试。

##### 1) 生产者

```
1 package com.ithouse.rabbitmq.work;
2
3 import com.ithouse.rabbitmq.util.ConnectionUtil;
4 import com.rabbitmq.client.Channel;
5 import com.rabbitmq.client.Connection;
6 import com.rabbitmq.client.ConnectionFactory;
7
```

```

8 public class Producer {
9
10     static final String QUEUE_NAME = "work_queue";
11
12     public static void main(String[] args) throws Exception {
13
14         //创建连接
15         Connection connection = ConnectionUtil.getConnection();
16
17         // 创建频道
18         Channel channel = connection.createChannel();
19
20         // 声明 (创建) 队列
21         /**
22          * 参数1: 队列名称
23          * 参数2: 是否定义持久化队列
24          * 参数3: 是否独占本次连接
25          * 参数4: 是否在不使用的时候自动删除队列
26          * 参数5: 队列其它参数
27          */
28         channel.queueDeclare(QUEUE_NAME, true, false, false, null);
29
30         for (int i = 1; i <= 30; i++) {
31             // 发送信息
32             String message = "你好; 小兔子! work模式--" + i;
33             /**
34              * 参数1: 交换机名称, 如果没有指定则使用默认Default Exchage
35              * 参数2: 路由key, 简单模式可以传递队列名称
36              * 参数3: 消息其它属性
37              * 参数4: 消息内容
38              */
39             channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
40             System.out.println("已发送消息: " + message);
41         }
42
43         // 关闭资源
44         channel.close();
45         connection.close();
46     }
47 }
48

```

## 2) 消费者1

```

1 package com.ithouse.rabbitmq.work;
2
3 import com.ithouse.rabbitmq.util.ConnectionUtil;
4 import com.rabbitmq.client.*;
5
6 import java.io.IOException;
7
8 public class Consumer1 {
9

```

```

10     public static void main(String[] args) throws Exception {
11         Connection connection = ConnectionUtil.getConnection();
12
13         // 创建频道
14         Channel channel = connection.createChannel();
15
16         // 声明 (创建) 队列
17         /**
18          * 参数1: 队列名称
19          * 参数2: 是否定义持久化队列
20          * 参数3: 是否独占本次连接
21          * 参数4: 是否在不使用的时候自动删除队列
22          * 参数5: 队列其它参数
23          */
24         channel.queueDeclare(Producer.QUEUE_NAME, true, false, false, null);
25
26         //一次只能接收并处理一个消息
27         channel.basicQos(1);
28
29         //创建消费者; 并设置消息处理
30         DefaultConsumer consumer = new DefaultConsumer(channel){
31             @Override
32             /**
33              * consumerTag 消息者标签, 在channel.basicConsume时候可以指定
34              * envelope 消息包的内容, 可从中获取消息id, 消息routingkey, 交换机, 消息和重传标志
35              * (收到消息失败后是否需要重新发送)
36              * properties 属性信息
37              * body 消息
38              */
39             public void handleDelivery(String consumerTag, Envelope envelope,
40             AMQP.BasicProperties properties, byte[] body) throws IOException {
41                 try {
42                     //路由key
43                     System.out.println("路由key为: " + envelope.getRoutingKey());
44                     //交换机
45                     System.out.println("交换机为: " + envelope.getExchange());
46                     //消息id
47                     System.out.println("消息id为: " + envelope.getDeliveryTag());
48                     //收到的消息
49                     System.out.println("消费者1-接收到的消息为: " + new String(body, "utf-
50             8"));
51
52                     Thread.sleep(1000);
53
54                     //确认消息
55                     channel.basicAck(envelope.getDeliveryTag(), false);
56                 } catch (InterruptedException e) {
57                     e.printStackTrace();
58                 }
59             }
60         };
61
62         //监听消息
63         /**
64          * 参数1: 队列名称

```

```

60      * 参数2: 是否自动确认, 设置为true为表示消息接收到自动向mq回复接收到了, mq接收到回复会删除消
      息, 设置为false则需要手动确认
61      * 参数3: 消息接收到后回调
62      */
63      channel.basicConsume(Producer.QUEUE_NAME, false, consumer);
64  }
65 }

```

### 3) 消费者2

```

1  package com.ithouse.rabbitmq.work;
2
3  import com.ithouse.rabbitmq.util.ConnectionUtil;
4  import com.rabbitmq.client.*;
5
6  import java.io.IOException;
7
8  public class Consumer2 {
9
10     public static void main(String[] args) throws Exception {
11         Connection connection = ConnectionUtil.getConnection();
12
13         // 创建频道
14         Channel channel = connection.createChannel();
15
16         // 声明 (创建) 队列
17         /**
18          * 参数1: 队列名称
19          * 参数2: 是否定义持久化队列
20          * 参数3: 是否独占本次连接
21          * 参数4: 是否在不使用的时候自动删除队列
22          * 参数5: 队列其它参数
23          */
24         channel.queueDeclare(Producer.QUEUE_NAME, true, false, false, null);
25
26         //一次只能接收并处理一个消息
27         channel.basicQos(1);
28
29         //创建消费者; 并设置消息处理
30         DefaultConsumer consumer = new DefaultConsumer(channel){
31             @Override
32             /**
33              * consumerTag 消息者标签, 在channel.basicConsume时候可以指定
34              * envelope 消息包的内容, 可从中获取消息id, 消息routingkey, 交换机, 消息和重传标志
              (收到消息失败后是否需要重新发送)
35              * properties 属性信息
36              * body 消息
37              */
38             public void handleDelivery(String consumerTag, Envelope envelope,
              AMQP.BasicProperties properties, byte[] body) throws IOException {
39                 try {
40                     //路由key
41                     System.out.println("路由key为: " + envelope.getRoutingKey());

```

```

42         //交换机
43         System.out.println("交换机为: " + envelope.getExchange());
44         //消息id
45         System.out.println("消息id为: " + envelope.getDeliveryTag());
46         //收到的消息
47         System.out.println("消费者2-接收到的消息为: " + new String(body, "utf-
8"));
48
49         Thread.sleep(1000);
50
51         //确认消息
52         channel.basicAck(envelope.getDeliveryTag(), false);
53     } catch (InterruptedException e) {
54         e.printStackTrace();
55     }
56 };
57 //监听消息
58 /**
59     * 参数1: 队列名称
60     * 参数2: 是否自动确认, 设置为true为表示消息接收到自动向mq回复接收到了, mq接收到回复会删除消
61     息, 设置为false则需要手动确认
62     * 参数3: 消息接收到后回调
63     */
64 channel.basicConsume(Producer.QUEUE_NAME, false, consumer);
65 }
66 }

```

### 4.1.3. 测试

启动两个消费者, 然后再启动生产者发送消息; 到IDEA的两个消费者对应的控制台查看是否竞争性的接收到消息。

```
Consumer1 x Consumer2 x Producer (1) x
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder
路由key为: work_queue
交换机为:
消息id为: 1
消费者1-接收到的消息为: 你好; 小兔子! work模式--1
路由key为: work_queue
交换机为:
消息id为: 2
消费者1-接收到的消息为: 你好; 小兔子! work模式--3
路由key为: work_queue
交换机为:
消息id为: 3
消费者1-接收到的消息为: 你好; 小兔子! work模式--5
路由key为: work_queue
交换机为:
消息id为: 4
```

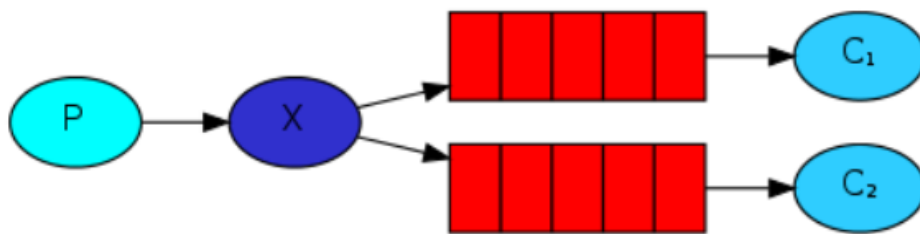
```
Consumer1 x Consumer2 x Producer (1) x
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder
路由key为: work_queue
交换机为:
消息id为: 1
消费者2-接收到的消息为: 你好; 小兔子! work模式--2
路由key为: work_queue
交换机为:
消息id为: 2
消费者2-接收到的消息为: 你好; 小兔子! work模式--4
路由key为: work_queue
交换机为:
消息id为: 3
消费者2-接收到的消息为: 你好; 小兔子! work模式--6
路由key为: work_queue
```

#### 4.1.4. 小结

在一个队列中如果有多个消费者，那么消费者之间对于同一个消息的关系是**竞争**的关系。

## 4.2. 订阅模式类型

订阅模式示例图：



前面2个案例中，只有3个角色：

- P：生产者，也就是要发送消息的程序
- C：消费者：消息的接受者，会一直等待消息到来。
- queue：消息队列，图中红色部分

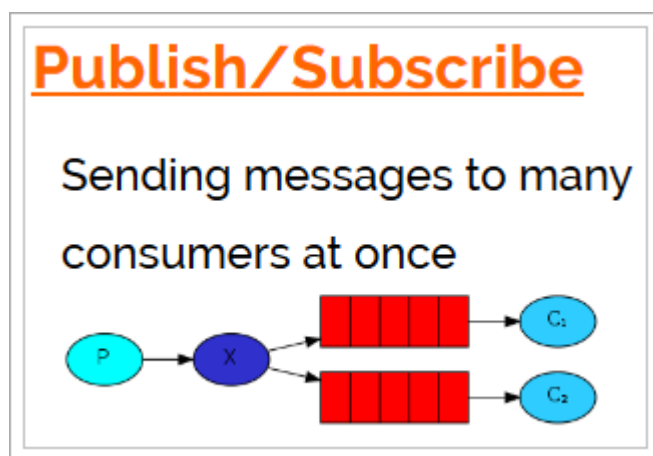
而在订阅模型中，多了一个exchange角色，而且过程略有变化：

- P：生产者，也就是要发送消息的程序，但是不再发送到队列中，而是发给X（交换机）
- C：消费者，消息的接受者，会一直等待消息到来。
- Queue：消息队列，接收消息、缓存消息。
- Exchange：交换机，图中的X。一方面，接收生产者发送的消息。另一方面，知道如何处理消息，例如递交给某个特别队列、递交给所有队列、或是将消息丢弃。到底如何操作，取决于Exchange的类型。Exchange有常见以下3种类型：
  - Fanout：广播，将消息交给所有绑定到交换机的队列
  - Direct：定向，把消息交给符合指定routing key 的队列
  - Topic：通配符，把消息交给符合routing pattern（路由模式）的队列

**Exchange（交换机）只负责转发消息，不具备存储消息的能力**，因此如果没有任何队列与Exchange绑定，或者没有符合路由规则的队列，那么消息会丢失！

## 4.3. Publish/Subscribe发布与订阅模式

### 4.3.1. 模式说明



发布订阅模式：1、每个消费者监听自己的队列。2、生产者将消息发给broker，由交换机将消息转发到绑定此交换机的每个队列，每个绑定交换机的队列都将接收到消息

## 4.3.2. 代码

### 1) 生产者

```
1 package com.ithouse.rabbitmq.ps;
2
3 import com.ithouse.rabbitmq.util.ConnectionUtil;
4 import com.rabbitmq.client.BuiltinExchangeType;
5 import com.rabbitmq.client.Channel;
6 import com.rabbitmq.client.Connection;
7
8 /**
9  * 发布与订阅使用的交换机类型为: fanout
10  */
11 public class Producer {
12
13     //交换机名称
14     static final String FANOUT_EXCHANGE = "fanout_exchange";
15     //队列名称
16     static final String FANOUT_QUEUE_1 = "fanout_queue_1";
17     //队列名称
18     static final String FANOUT_QUEUE_2 = "fanout_queue_2";
19
20     public static void main(String[] args) throws Exception {
21
22         //创建连接
23         Connection connection = ConnectionUtil.getConnection();
24
25         // 创建频道
26         Channel channel = connection.createChannel();
27
28         /**
29          * 声明交换机
30          * 参数1: 交换机名称
31          * 参数2: 交换机类型, fanout、topic、direct、headers
32          */
33         channel.exchangeDeclare(FANOUT_EXCHANGE, BuiltinExchangeType.FANOUT);
34
35         // 声明 (创建) 队列
36         /**
37          * 参数1: 队列名称
38          * 参数2: 是否定义持久化队列
39          * 参数3: 是否独占本次连接
40          * 参数4: 是否在不使用的时候自动删除队列
41          * 参数5: 队列其它参数
42          */
43         channel.queueDeclare(FANOUT_QUEUE_1, true, false, false, null);
44         channel.queueDeclare(FANOUT_QUEUE_2, true, false, false, null);
```

```

45
46 //队列绑定交换机
47 channel.queueBind(FANOUT_QUEUE_1, FANOUT_EXCHANGE, "");
48 channel.queueBind(FANOUT_QUEUE_2, FANOUT_EXCHANGE, "");
49
50 for (int i = 1; i <= 10; i++) {
51     // 发送信息
52     String message = "你好; 小兔子! 发布订阅模式--" + i;
53     /**
54      * 参数1: 交换机名称, 如果没有指定则使用默认Default Exchange
55      * 参数2: 路由key, 简单模式可以传递队列名称
56      * 参数3: 消息其它属性
57      * 参数4: 消息内容
58      */
59     channel.basicPublish(FANOUT_EXCHANGE, "", null, message.getBytes());
60     System.out.println("已发送消息: " + message);
61 }
62
63 // 关闭资源
64 channel.close();
65 connection.close();
66 }
67 }

```

## 2) 消费者1

```

1 package com.ithouse.rabbitmq.ps;
2
3 import com.ithouse.rabbitmq.util.ConnectionUtil;
4 import com.rabbitmq.client.*;
5
6 import java.io.IOException;
7
8 public class Consumer1 {
9
10     public static void main(String[] args) throws Exception {
11         Connection connection = ConnectionUtil.getConnection();
12
13         // 创建频道
14         Channel channel = connection.createChannel();
15
16         //声明交换机
17         channel.exchangeDeclare(Producer.FANOUT_EXCHANGE, BuiltinExchangeType.FANOUT);
18
19         // 声明 (创建) 队列
20         /**
21          * 参数1: 队列名称
22          * 参数2: 是否定义持久化队列
23          * 参数3: 是否独占本次连接
24          * 参数4: 是否在不使用的时候自动删除队列
25          * 参数5: 队列其它参数

```

```

26         */
27         channel.queueDeclare(Producer.FANOUT_QUEUE_1, true, false, false, null);
28
29         //队列绑定交换机
30         channel.queueBind(Producer.FANOUT_QUEUE_1, Producer.FANOUT_EXCHANGE, "");
31
32         //创建消费者; 并设置消息处理
33         DefaultConsumer consumer = new DefaultConsumer(channel){
34             @Override
35             /**
36              * consumerTag 消息者标签, 在channel.basicConsume时候可以指定
37              * envelope 消息包的内容, 可从中获取消息id, 消息routingkey, 交换机, 消息和重传标志
38              * properties 属性信息
39              * body 消息
40              */
41             public void handleDelivery(String consumerTag, Envelope envelope,
42             AMQP.BasicProperties properties, byte[] body) throws IOException {
43                 //路由key
44                 System.out.println("路由key为: " + envelope.getRoutingKey());
45                 //交换机
46                 System.out.println("交换机为: " + envelope.getExchange());
47                 //消息id
48                 System.out.println("消息id为: " + envelope.getDeliveryTag());
49                 //收到的消息
50                 System.out.println("消费者1-接收到的消息为: " + new String(body, "utf-
51                 8"));
52             }
53         };
54         //监听消息
55         /**
56          * 参数1: 队列名称
57          * 参数2: 是否自动确认, 设置为true为表示消息接收到自动向mq回复接收到了, mq接收到回复会删除消
58          * 息, 设置为false则需要手动确认
59          * 参数3: 消息接收到后回调
60          */
61         channel.basicConsume(Producer.FANOUT_QUEUE_1, true, consumer);

```

### 3) 消费者2

```

1 package com.ithouse.rabbitmq.ps;
2
3 import com.ithouse.rabbitmq.util.ConnectionUtil;
4 import com.rabbitmq.client.*;
5
6 import java.io.IOException;
7
8 public class Consumer2 {
9
10     public static void main(String[] args) throws Exception {

```

```

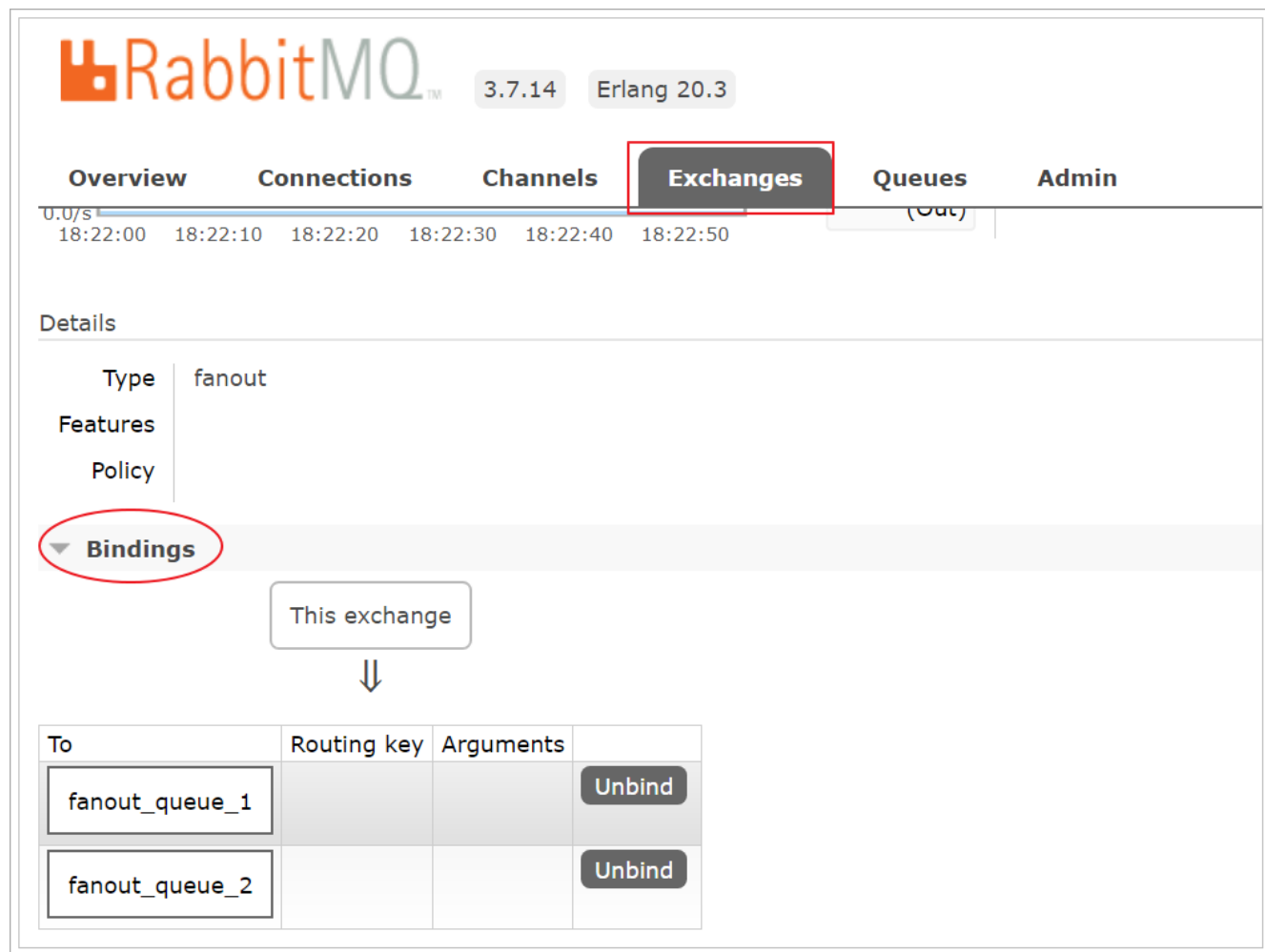
11      Connection connection = ConnectionUtil.getConnection();
12
13      // 创建频道
14      Channel channel = connection.createChannel();
15
16      //声明交换机
17      channel.exchangeDeclare(Producer.FANOUT_EXCHANGE, BuiltinExchangeType.FANOUT);
18
19      // 声明 (创建) 队列
20      /**
21       * 参数1: 队列名称
22       * 参数2: 是否定义持久化队列
23       * 参数3: 是否独占本次连接
24       * 参数4: 是否在不使用的时候自动删除队列
25       * 参数5: 队列其它参数
26       */
27      channel.queueDeclare(Producer.FANOUT_QUEUE_2, true, false, false, null);
28
29      //队列绑定交换机
30      channel.queueBind(Producer.FANOUT_QUEUE_2, Producer.FANOUT_EXCHANGE, "");
31
32      //创建消费者; 并设置消息处理
33      DefaultConsumer consumer = new DefaultConsumer(channel){
34          @Override
35          /**
36           * consumerTag 消息者标签, 在channel.basicConsume时候可以指定
37           * envelope 消息包的内容, 可从中获取消息id, 消息routingkey, 交换机, 消息和重传标志
(收到消息失败后是否需要重新发送)
38           * properties 属性信息
39           * body 消息
40           */
41          public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
42              //路由key
43              System.out.println("路由key为: " + envelope.getRoutingKey());
44              //交换机
45              System.out.println("交换机为: " + envelope.getExchange());
46              //消息id
47              System.out.println("消息id为: " + envelope.getDeliveryTag());
48              //收到的消息
49              System.out.println("消费者2-接收到的消息为: " + new String(body, "utf-
8"));
50          }
51      };
52      //监听消息
53      /**
54       * 参数1: 队列名称
55       * 参数2: 是否自动确认, 设置为true为表示消息接收到自动向mq回复接收到了, mq接收到回复会删除消
息, 设置为false则需要手动确认
56       * 参数3: 消息接收到后回调
57       */
58      channel.basicConsume(Producer.FANOUT_QUEUE_2, true, consumer);
59  }

```

### 4.3.3. 测试

启动所有消费者，然后使用生产者发送消息；在每个消费者对应的控制台可以查看到生产者发送的所有消息；到达广播的效果。

在执行完测试代码后，其实到RabbitMQ的管理后台找到 Exchanges 选项卡，点击 `fanout_exchange` 的交换机，可以查看到如下的绑定：



### 4.3.4. 小结

交换机需要与队列进行绑定，绑定之后；一个消息可以被多个消费者都收到。

#### 发布订阅模式与工作队列模式的区别

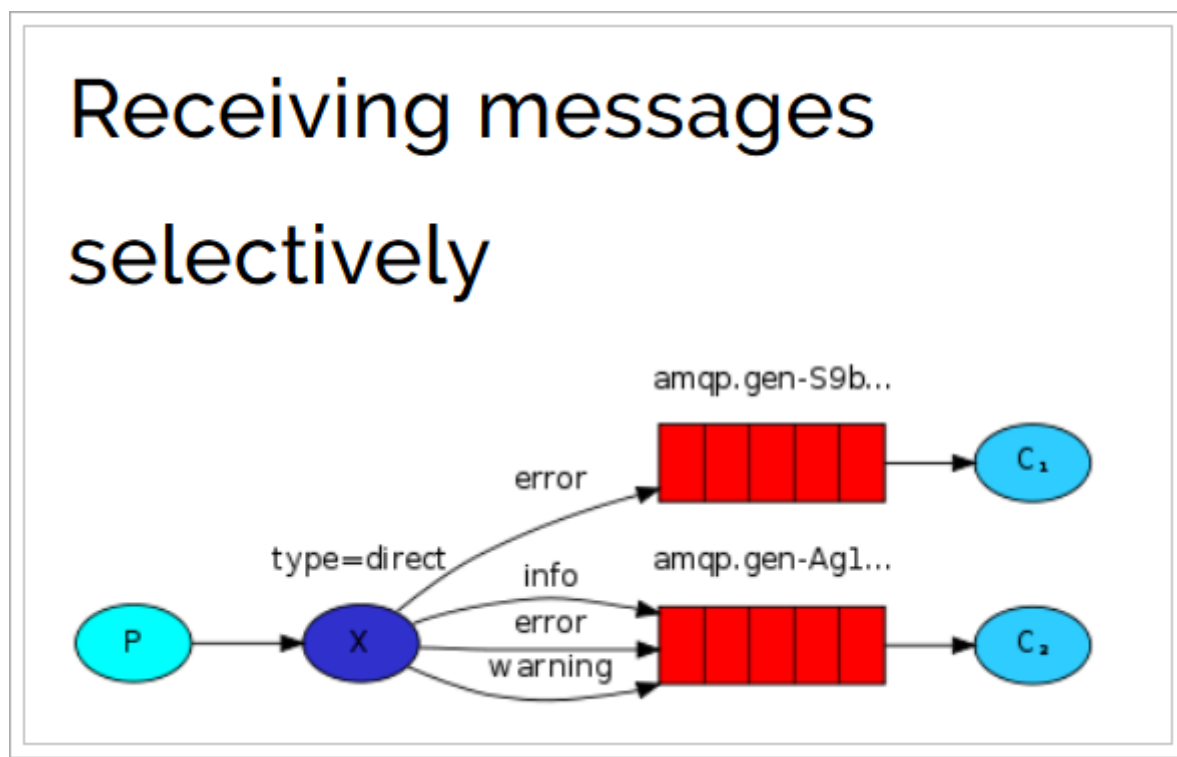
- 1、工作队列模式不用定义交换机，而发布/订阅模式需要定义交换机。
- 2、发布/订阅模式的生产方是面向交换机发送消息，工作队列模式的生产方是面向队列发送消息(底层使用默认交换机)。
- 3、发布/订阅模式需要设置队列和交换机的绑定，工作队列模式不需要设置，实际上工作队列模式会将队列绑定到默认的交换机。

## 4.4. Routing路由模式

### 4.4.1. 模式说明

路由模式特点：

- 队列与交换机的绑定，不能是任意绑定了，而是要指定一个 `RoutingKey`（路由key）
- 消息的发送方在向 `Exchange` 发送消息时，也必须指定消息的 `RoutingKey`。
- `Exchange` 不再把消息交给每一个绑定的队列，而是根据消息的 `Routing Key` 进行判断，只有队列的 `Routingkey` 与消息的 `Routing key` 完全一致，才会接收到消息



图解：

- P：生产者，向Exchange发送消息，发送消息时，会指定一个routing key。
- X：Exchange（交换机），接收生产者的消息，然后把消息递交给与routing key完全匹配的队列
- C1：消费者，其所在队列指定了需要routing key 为 error 的消息
- C2：消费者，其所在队列指定了需要routing key 为 info、error、warning 的消息

### 4.4.2. 代码

在编码上与 `Publish/Subscribe` 发布与订阅模式 的区别是交换机的类型为：Direct，还有队列绑定交换机的时候需要指定routing key。

#### 1) 生产者

```
1 package com.ithouse.rabbitmq.routing;
2
3 import com.ithouse.rabbitmq.util.ConnectionUtil;
4 import com.rabbitmq.client.BuiltinExchangeType;
```

```

5  import com.rabbitmq.client.Channel;
6  import com.rabbitmq.client.Connection;
7
8  /**
9   * 路由模式的交换机类型为: direct
10  */
11  public class Producer {
12
13      //交换机名称
14      static final String DIRECT_EXCHAGE = "direct_exchange";
15      //队列名称
16      static final String DIRECT_QUEUE_INSERT = "direct_queue_insert";
17      //队列名称
18      static final String DIRECT_QUEUE_UPDATE = "direct_queue_update";
19
20      public static void main(String[] args) throws Exception {
21
22          //创建连接
23          Connection connection = ConnectionUtil.getConnection();
24
25          // 创建频道
26          Channel channel = connection.createChannel();
27
28          /**
29           * 声明交换机
30           * 参数1: 交换机名称
31           * 参数2: 交换机类型, fanout、topic、direct、headers
32           */
33          channel.exchangeDeclare(DIRECT_EXCHAGE, BuiltinExchangeType.DIRECT);
34
35          // 声明 (创建) 队列
36          /**
37           * 参数1: 队列名称
38           * 参数2: 是否定义持久化队列
39           * 参数3: 是否独占本次连接
40           * 参数4: 是否在不使用的时候自动删除队列
41           * 参数5: 队列其它参数
42           */
43          channel.queueDeclare(DIRECT_QUEUE_INSERT, true, false, false, null);
44          channel.queueDeclare(DIRECT_QUEUE_UPDATE, true, false, false, null);
45
46          //队列绑定交换机
47          channel.queueBind(DIRECT_QUEUE_INSERT, DIRECT_EXCHAGE, "insert");
48          channel.queueBind(DIRECT_QUEUE_UPDATE, DIRECT_EXCHAGE, "update");
49
50          // 发送信息
51          String message = "新增了商品。路由模式; routing key 为 insert ";
52          /**
53           * 参数1: 交换机名称, 如果没有指定则使用默认Default Exchange
54           * 参数2: 路由key, 简单模式可以传递队列名称
55           * 参数3: 消息其它属性
56           * 参数4: 消息内容
57           */

```

```

58     channel.basicPublish(DIRECT_EXCHANGE, "insert", null, message.getBytes());
59     System.out.println("已发送消息: " + message);
60
61     // 发送信息
62     message = "修改了商品。路由模式; routing key 为 update" ;
63     /**
64      * 参数1: 交换机名称, 如果没有指定则使用默认Default Exchange
65      * 参数2: 路由key,简单模式可以传递队列名称
66      * 参数3: 消息其它属性
67      * 参数4: 消息内容
68      */
69     channel.basicPublish(DIRECT_EXCHANGE, "update", null, message.getBytes());
70     System.out.println("已发送消息: " + message);
71
72     // 关闭资源
73     channel.close();
74     connection.close();
75 }
76 }
77

```

## 2) 消费者1

```

1  package com.ithouse.rabbitmq.routing;
2
3  import com.ithouse.rabbitmq.util.ConnectionUtil;
4  import com.rabbitmq.client.*;
5
6  import java.io.IOException;
7
8  public class Consumer1 {
9
10     public static void main(String[] args) throws Exception {
11         Connection connection = ConnectionUtil.getConnection();
12
13         // 创建频道
14         Channel channel = connection.createChannel();
15
16         //声明交换机
17         channel.exchangeDeclare(Producer.DIRECT_EXCHANGE, BuiltinExchangeType.DIRECT);
18
19         // 声明 (创建) 队列
20         /**
21          * 参数1: 队列名称
22          * 参数2: 是否定义持久化队列
23          * 参数3: 是否独占本次连接
24          * 参数4: 是否在不使用的时候自动删除队列
25          * 参数5: 队列其它参数
26          */
27         channel.queueDeclare(Producer.DIRECT_QUEUE_INSERT, true, false, false, null);
28
29         //队列绑定交换机

```

```

30     channel.queueBind(Producer.DIRECT_QUEUE_INSERT, Producer.DIRECT_EXCHANGE,
    "insert");
31
32     //创建消费者; 并设置消息处理
33     DefaultConsumer consumer = new DefaultConsumer(channel){
34         @Override
35         /**
36          * consumerTag 消息者标签, 在channel.basicConsume时候可以指定
37          * envelope 消息包的内容, 可从中获取消息id, 消息routingkey, 交换机, 消息和重传标志
    (收到消息失败后是否需要重新发送)
38          * properties 属性信息
39          * body 消息
40          */
41         public void handleDelivery(String consumerTag, Envelope envelope,
    AMQP.BasicProperties properties, byte[] body) throws IOException {
42             //路由key
43             System.out.println("路由key为: " + envelope.getRoutingKey());
44             //交换机
45             System.out.println("交换机为: " + envelope.getExchange());
46             //消息id
47             System.out.println("消息id为: " + envelope.getDeliveryTag());
48             //收到的消息
49             System.out.println("消费者1-接收到的消息为: " + new String(body, "utf-
    8"));
50         }
51     };
52     //监听消息
53     /**
54      * 参数1: 队列名称
55      * 参数2: 是否自动确认, 设置为true为表示消息接收到自动向mq回复接收到了, mq接收到回复会删除消
    息, 设置为false则需要手动确认
56      * 参数3: 消息接收到后回调
57      */
58     channel.basicConsume(Producer.DIRECT_QUEUE_INSERT, true, consumer);
59 }
60 }
61

```

### 3) 消费者2

```

1  package com.ithouse.rabbitmq.routing;
2
3  import com.ithouse.rabbitmq.util.ConnectionUtil;
4  import com.rabbitmq.client.*;
5
6  import java.io.IOException;
7
8  public class Consumer2 {
9
10     public static void main(String[] args) throws Exception {
11         Connection connection = ConnectionUtil.getConnection();
12
13         // 创建频道

```

```

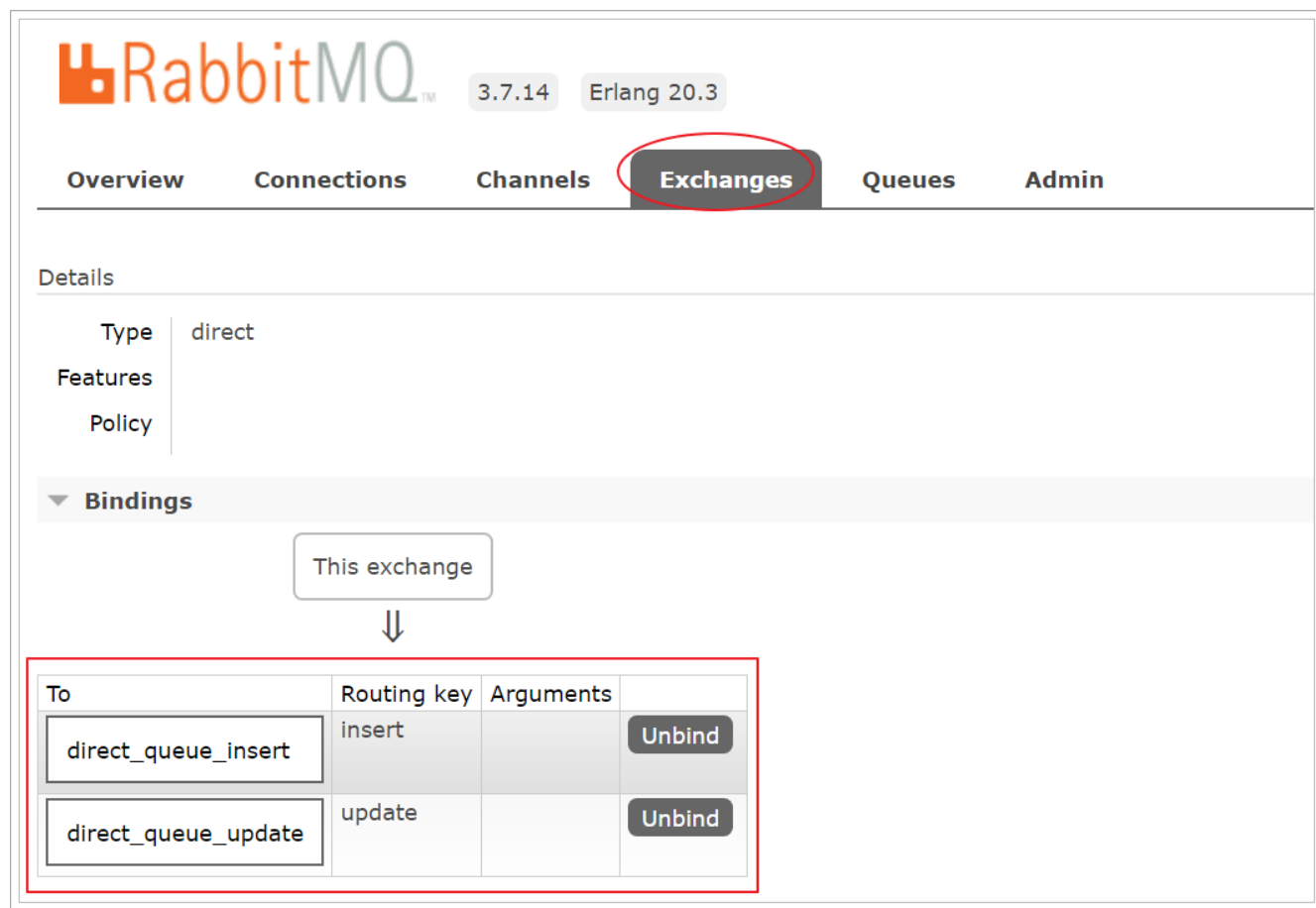
14     Channel channel = connection.createChannel();
15
16     //声明交换机
17     channel.exchangeDeclare(Producer.DIRECT_EXCHANGE, BuiltinExchangeType.DIRECT);
18
19     // 声明（创建）队列
20     /**
21      * 参数1: 队列名称
22      * 参数2: 是否定义持久化队列
23      * 参数3: 是否独占本次连接
24      * 参数4: 是否在不使用的时候自动删除队列
25      * 参数5: 队列其它参数
26      */
27     channel.queueDeclare(Producer.DIRECT_QUEUE_UPDATE, true, false, false, null);
28
29     //队列绑定交换机
30     channel.queueBind(Producer.DIRECT_QUEUE_UPDATE, Producer.DIRECT_EXCHANGE,
31 "update");
32
33     //创建消费者; 并设置消息处理
34     DefaultConsumer consumer = new DefaultConsumer(channel){
35         @Override
36         /**
37          * consumerTag 消息者标签, 在channel.basicConsume时候可以指定
38          * envelope 消息包的内容, 可从中获取消息id, 消息routingkey, 交换机, 消息和重传标志
39          (收到消息失败后是否需要重新发送)
40          * properties 属性信息
41          * body 消息
42          */
43         public void handleDelivery(String consumerTag, Envelope envelope,
44 AMQP.BasicProperties properties, byte[] body) throws IOException {
45             //路由key
46             System.out.println("路由key为: " + envelope.getRoutingKey());
47             //交换机
48             System.out.println("交换机为: " + envelope.getExchange());
49             //消息id
50             System.out.println("消息id为: " + envelope.getDeliveryTag());
51             //收到的消息
52             System.out.println("消费者2-接收到的消息为: " + new String(body, "utf-
53 8"));
54         }
55     };
56     //监听消息
57     /**
58      * 参数1: 队列名称
59      * 参数2: 是否自动确认, 设置为true为表示消息接收到自动向mq回复接收到了, mq接收到回复会删除消
60 息, 设置为false则需要手动确认
61      * 参数3: 消息接收到后回调
62      */
63     channel.basicConsume(Producer.DIRECT_QUEUE_UPDATE, true, consumer);
64 }
65 }

```

### 4.4.3. 测试

启动所有消费者，然后使用生产者发送消息；在消费者对应的控制台可以查看到生产者发送对应routing key对应队列的消息；到达**按照需要接收**的效果。

在执行完测试代码后，其实到RabbitMQ的管理后台找到 Exchanges 选项卡，点击 `direct_exchange` 的交换机，可以查看到如下的绑定：



The screenshot shows the RabbitMQ management interface. The 'Exchanges' tab is selected. Under 'Details', the 'Type' is 'direct'. The 'Bindings' section shows a table with two entries:

| To                               | Routing key         | Arguments |                         |
|----------------------------------|---------------------|-----------|-------------------------|
| <code>direct_queue_insert</code> | <code>insert</code> |           | <button>Unbind</button> |
| <code>direct_queue_update</code> | <code>update</code> |           | <button>Unbind</button> |

### 4.4.4. 小结

Routing模式要求队列在绑定交换机时要指定routing key，消息会转发到符合routing key的队列。

## 4.5. Topics通配符模式

### 4.5.1. 模式说明

Topic 类型与 Direct 相比，都是可以根据 RoutingKey 把消息路由到不同的队列。只不过 Topic 类型 Exchange 可以让队列在绑定 Routing key 的时候**使用通配符**！

Routingkey 一般都是有一个或多个单词组成，多个单词之间以“.”分割，例如：`item.insert`

通配符规则：

#：匹配一个或多个词

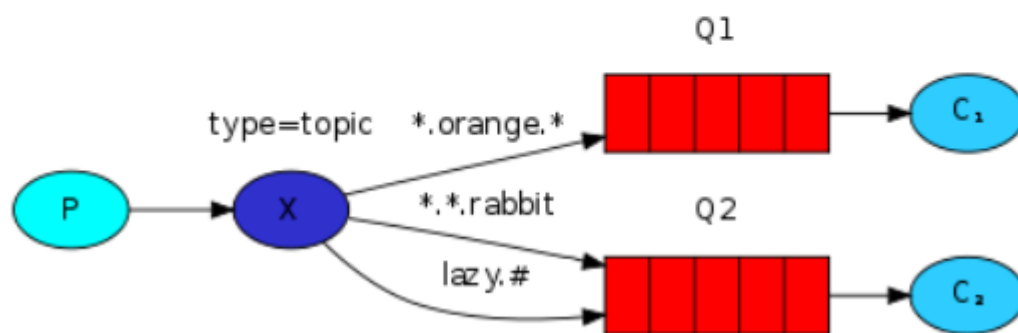
\*: 匹配不多不少恰好1个词

举例:

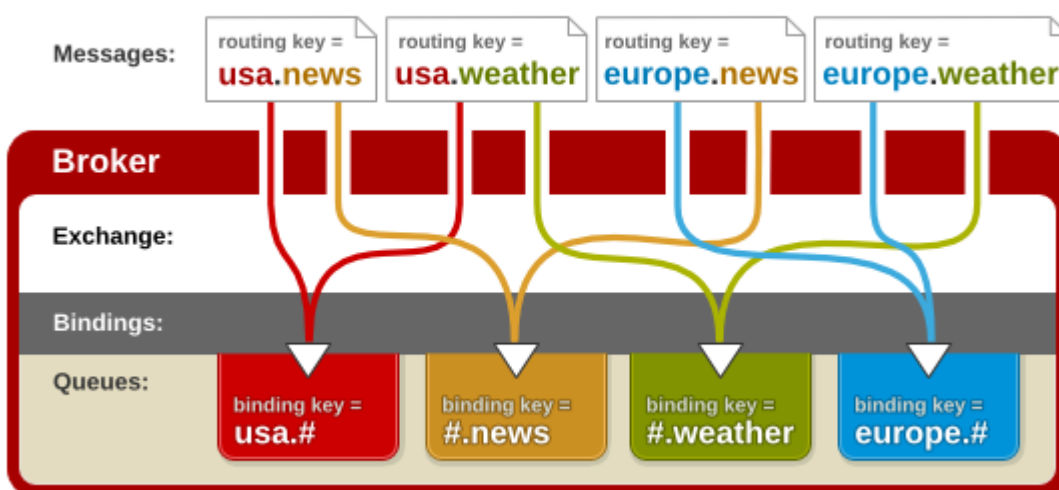
item.#: 能够匹配 item.insert.abc 或者 item.insert

item.\*: 只能匹配 item.insert

## Receiving messages based on a pattern (topics)



### Topic Exchange



图解:

- 红色Queue: 绑定的是 `usa.#`, 因此凡是以 `usa.` 开头的 routing key 都会被匹配到
- 黄色Queue: 绑定的是 `#.news`, 因此凡是以 `.news` 结尾的 routing key 都会被匹配

## 4.5.2. 代码

### 1) 生产者

使用topic类型的Exchange，发送消息的routing key有3种： `item.insert`、`item.update`、`item.delete`：

```
1 package com.ithouse.rabbitmq.topic;
2
3 import com.ithouse.rabbitmq.util.ConnectionUtil;
4 import com.rabbitmq.client.BuiltinExchangeType;
5 import com.rabbitmq.client.Channel;
6 import com.rabbitmq.client.Connection;
7
8 /**
9  * 通配符Topic的交换机类型为: topic
10  */
11 public class Producer {
12
13     //交换机名称
14     static final String TOPIC_EXCHANGE = "topic_exchange";
15     //队列名称
16     static final String TOPIC_QUEUE_1 = "topic_queue_1";
17     //队列名称
18     static final String TOPIC_QUEUE_2 = "topic_queue_2";
19
20     public static void main(String[] args) throws Exception {
21
22         //创建连接
23         Connection connection = ConnectionUtil.getConnection();
24
25         // 创建频道
26         Channel channel = connection.createChannel();
27
28         /**
29          * 声明交换机
30          * 参数1: 交换机名称
31          * 参数2: 交换机类型, fanout、topic、topic、headers
32          */
33         channel.exchangeDeclare(TOPIC_EXCHANGE, BuiltinExchangeType.TOPIC);
34
35
36         // 发送信息
37         String message = "新增了商品。Topic模式; routing key 为 item.insert ";
38         channel.basicPublish(TOPIC_EXCHANGE, "item.insert", null, message.getBytes());
39         System.out.println("已发送消息: " + message);
40
41         // 发送信息
42         message = "修改了商品。Topic模式; routing key 为 item.update ";
43         channel.basicPublish(TOPIC_EXCHANGE, "item.update", null, message.getBytes());
44         System.out.println("已发送消息: " + message);
45     }
46 }
```

```

46      // 发送信息
47      message = "删除了商品。Topic模式; routing key 为 item.delete" ;
48      channel.basicPublish(TOPIC_EXCHANGE, "item.delete", null, message.getBytes());
49      System.out.println("已发送消息: " + message);
50
51      // 关闭资源
52      channel.close();
53      connection.close();
54  }
55 }
56

```

## 2) 消费者1

接收两种类型的消息：更新商品和删除商品

```

1  package com.ithouse.rabbitmq.topic;
2
3  import com.ithouse.rabbitmq.util.ConnectionUtil;
4  import com.rabbitmq.client.*;
5
6  import java.io.IOException;
7
8  public class Consumer1 {
9
10     public static void main(String[] args) throws Exception {
11         Connection connection = ConnectionUtil.getConnection();
12
13         // 创建频道
14         Channel channel = connection.createChannel();
15
16         //声明交换机
17         channel.exchangeDeclare(Producer.TOPIC_EXCHANGE, BuiltinExchangeType.TOPIC);
18
19         // 声明（创建）队列
20         /**
21          * 参数1: 队列名称
22          * 参数2: 是否定义持久化队列
23          * 参数3: 是否独占本次连接
24          * 参数4: 是否在不使用的时候自动删除队列
25          * 参数5: 队列其它参数
26          */
27         channel.queueDeclare(Producer.TOPIC_QUEUE_1, true, false, false, null);
28
29         //队列绑定交换机
30         channel.queueBind(Producer.TOPIC_QUEUE_1, Producer.TOPIC_EXCHANGE,
31             "item.update");
32         channel.queueBind(Producer.TOPIC_QUEUE_1, Producer.TOPIC_EXCHANGE,
33             "item.delete");
34
35         //创建消费者; 并设置消息处理
36         DefaultConsumer consumer = new DefaultConsumer(channel){
37             @Override
38

```

```

36         /**
37         * consumerTag 消息者标签, 在channel.basicConsume时候可以指定
38         * envelope 消息包的内容, 可从中获取消息id, 消息routingkey, 交换机, 消息和重传标志
(收到消息失败后是否需要重新发送)
39         * properties 属性信息
40         * body 消息
41         */
42         public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
43             //路由key
44             System.out.println("路由key为: " + envelope.getRoutingKey());
45             //交换机
46             System.out.println("交换机为: " + envelope.getExchange());
47             //消息id
48             System.out.println("消息id为: " + envelope.getDeliveryTag());
49             //收到的消息
50             System.out.println("消费者1-接收到的消息为: " + new String(body, "utf-
51             8"));
52         }
53     };
54     /**
55     * 参数1: 队列名称
56     * 参数2: 是否自动确认, 设置为true为表示消息接收到自动向mq回复接收到了, mq接收到回复会删除消
息, 设置为false则需要手动确认
57     * 参数3: 消息接收到后回调
58     */
59     channel.basicConsume(Producer.TOPIC_QUEUE_1, true, consumer);
60 }
61 }
62

```

### 3) 消费者2

接收所有类型的消息：新增商品，更新商品和删除商品。

```

1 package com.ithouse.rabbitmq.topic;
2
3 import com.ithouse.rabbitmq.util.ConnectionUtil;
4 import com.rabbitmq.client.*;
5
6 import java.io.IOException;
7
8 public class Consumer2 {
9
10     public static void main(String[] args) throws Exception {
11         Connection connection = ConnectionUtil.getConnection();
12
13         // 创建频道
14         Channel channel = connection.createChannel();
15
16         //声明交换机
17         channel.exchangeDeclare(Producer.TOPIC_EXCHANGE, BuiltinExchangeType.TOPIC);

```

```

18
19 // 声明 (创建) 队列
20 /**
21  * 参数1: 队列名称
22  * 参数2: 是否定义持久化队列
23  * 参数3: 是否独占本次连接
24  * 参数4: 是否在不使用的时候自动删除队列
25  * 参数5: 队列其它参数
26  */
27 channel.queueDeclare(Producer.TOPIC_QUEUE_2, true, false, false, null);
28
29 //队列绑定交换机
30 channel.queueBind(Producer.TOPIC_QUEUE_2, Producer.TOPIC_EXCHANGE, "item.*");
31
32 //创建消费者; 并设置消息处理
33 DefaultConsumer consumer = new DefaultConsumer(channel){
34     @Override
35     /**
36      * consumerTag 消息者标签, 在channel.basicConsume时候可以指定
37      * envelope 消息包的内容, 可从中获取消息id, 消息routingkey, 交换机, 消息和重传标志
38      (收到消息失败后是否需要重新发送)
39      * properties 属性信息
40      * body 消息
41      */
42     public void handleDelivery(String consumerTag, Envelope envelope,
43 AMQP.BasicProperties properties, byte[] body) throws IOException {
44         //路由key
45         System.out.println("路由key为: " + envelope.getRoutingKey());
46         //交换机
47         System.out.println("交换机为: " + envelope.getExchange());
48         //消息id
49         System.out.println("消息id为: " + envelope.getDeliveryTag());
50         //收到的消息
51         System.out.println("消费者2-接收到的消息为: " + new String(body, "utf-
52 8"));
53     }
54 };
55 //监听消息
56 /**
57  * 参数1: 队列名称
58  * 参数2: 是否自动确认, 设置为true为表示消息接收到自动向mq回复接收到了, mq接收到回复会删除消
59 息, 设置为false则需要手动确认
60  * 参数3: 消息接收到后回调
61  */
62 channel.basicConsume(Producer.TOPIC_QUEUE_2, true, consumer);
63 }
64 }
65

```

### 4.5.3. 测试

启动所有消费者, 然后使用生产者发送消息; 在消费者对应的控制台可以查看到生产者发送对应routing key对应队列的消息; 到达按照需要接收的效果; 并且这些routing key可以使用通配符。

在执行完测试代码后，其实到RabbitMQ的管理后台找到 Exchanges 选项卡，点击 `topic_exchange` 的交换机，可以查看到如下的绑定：



#### 4.5.4. 小结

Topic主题模式可以实现 `Publish/subscribe`发布与订阅模式 和 `Routing`路由模式 的功能；只是Topic在配置routing key 的时候可以使用通配符，显得更加灵活。

## 4.6. 模式总结

RabbitMQ工作模式： **1、简单模式 HelloWorld** 一个生产者、一个消费者，不需要设置交换机（使用默认的交换机）

**2、工作队列模式 Work Queue** 一个生产者、多个消费者（竞争关系），不需要设置交换机（使用默认的交换机）

**3、发布订阅模式 Publish/subscribe** 需要设置类型为fanout的交换机，并且交换机和队列进行绑定，当发送消息到交换机后，交换机会将消息发送到绑定的队列

**4、路由模式 Routing** 需要设置类型为direct的交换机，交换机和队列进行绑定，并且指定routing key，当发送消息到交换机后，交换机会根据routing key将消息发送到对应的队列

**5、通配符模式 Topic** 需要设置类型为topic的交换机，交换机和队列进行绑定，并且指定通配符方式的routing key，当发送消息到交换机后，交换机会根据routing key将消息发送到对应的队列

## 5. Spring Boot整合RabbitMQ

## 5.1. 简介

在Spring项目中，可以使用Spring-Rabbit去操作RabbitMQ <https://github.com/spring-projects/spring-amqp>

尤其是在spring boot项目中只需要引入对应的amqp启动器依赖即可，方便的使用RabbitTemplate发送消息，使用注解接收消息。

一般在开发过程中:

### 生产者工程:

1. application.yml文件配置RabbitMQ相关信息;
2. 在生产者工程中编写配置类，用于创建交换机和队列，并进行绑定
3. 注入RabbitTemplate对象，通过RabbitTemplate对象发送消息到交换机

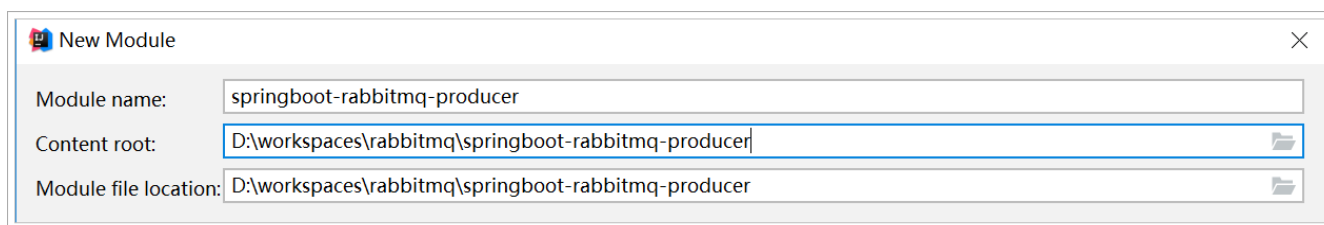
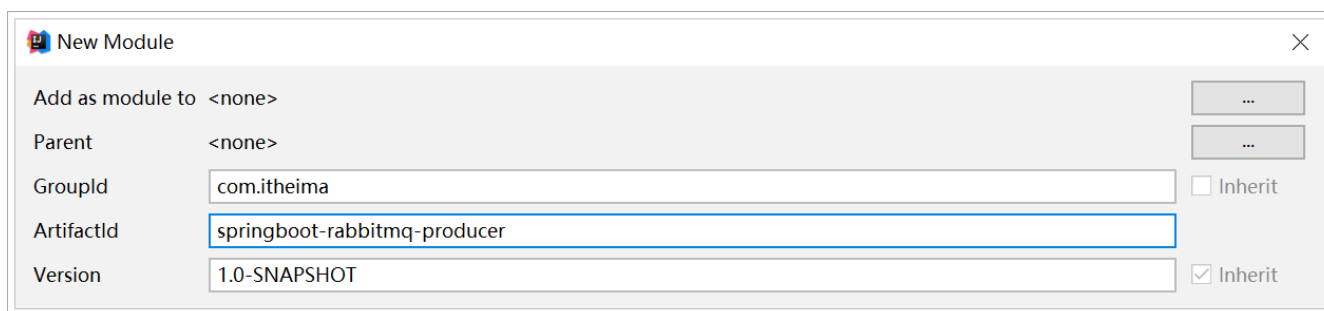
### 消费者工程:

1. application.yml文件配置RabbitMQ相关信息
2. 创建消息处理类，用于接收队列中的消息并进行处理

## 5.2. 搭建生产者工程

### 5.2.1. 创建工程

创建生产者工程springboot-rabbitmq-producer



### 5.2.2. 添加依赖

修改pom.xml文件内容为如下:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6     <parent>
7         <groupId>org.springframework.boot</groupId>
8         <artifactId>spring-boot-starter-parent</artifactId>
9         <version>2.1.4.RELEASE</version>
10    </parent>
11    <groupId>com.ithouse</groupId>
12    <artifactId>springboot-rabbitmq-producer</artifactId>
13    <version>1.0-SNAPSHOT</version>
14
15    <dependencies>
16        <dependency>
17            <groupId>org.springframework.boot</groupId>
18            <artifactId>spring-boot-starter-amqp</artifactId>
19        </dependency>
20        <dependency>
21            <groupId>org.springframework.boot</groupId>
22            <artifactId>spring-boot-starter-test</artifactId>
23        </dependency>
24    </dependencies>
25 </project>

```

### 5.2.3. 启动类

```

1 package com.ithouse.rabbitmq;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class ProducerApplication {
8     public static void main(String[] args) {
9         SpringApplication.run(ProducerApplication.class);
10    }
11 }
12

```

### 5.2.4. 配置RabbitMQ

#### 1) 配置文件

创建application.yml, 内容如下:

```
1 spring:
2   rabbitmq:
3     host: localhost
4     port: 5672
5     virtual-host:
6       /ithouseusername:
7     zspassword: zs
```

## 2) 绑定交换机和队列

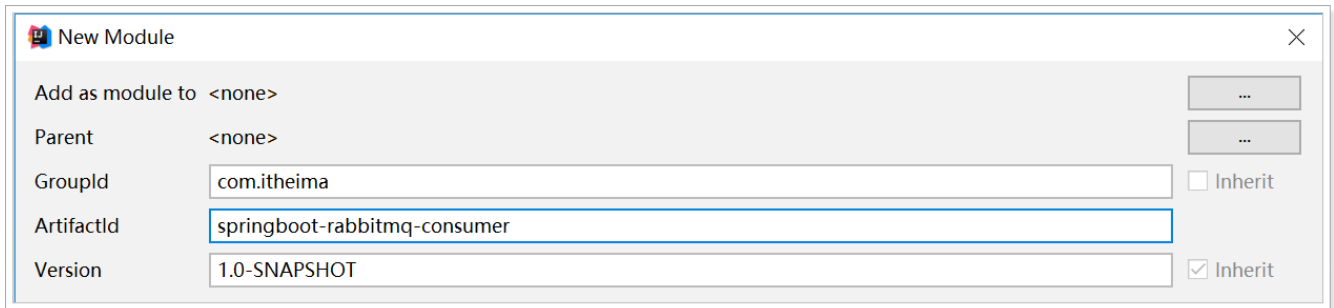
创建RabbitMQ队列与交换机绑定的配置类com.ithouse.rabbitmq.config.RabbitMQConfig

```
1 package com.ithouse.rabbitmq.config;
2
3 import org.springframework.amqp.core.*;
4 import org.springframework.beans.factory.annotation.Qualifier;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7
8 @Configuration
9 public class RabbitMQConfig {
10   //交换机名称
11   public static final String ITEM_TOPIC_EXCHANGE = "item_topic_exchange";
12   //队列名称
13   public static final String ITEM_QUEUE = "item_queue";
14
15   //声明交换机
16   @Bean("itemTopicExchange")
17   public Exchange topicExchange(){
18     return
19     ExchangeBuilder.topicExchange(ITEM_TOPIC_EXCHANGE).durable(true).build();
20   }
21
22   //声明队列
23   @Bean("itemQueue")
24   public Queue itemQueue(){
25     return QueueBuilder.durable(ITEM_QUEUE).build();
26   }
27
28   //绑定队列和交换机
29   @Bean
30   public Binding itemQueueExchange(@Qualifier("itemQueue") Queue queue,
31                                     @Qualifier("itemTopicExchange") Exchange
32                                     exchange){
33     return BindingBuilder.bind(queue).to(exchange).with("item.#").noargs();
34   }
35 }
```

## 5.3. 搭建消费者工程

### 5.3.1. 创建工程

创建消费者工程springboot-rabbitmq-consumer



New Module

Add as module to <none> ...

Parent <none> ...

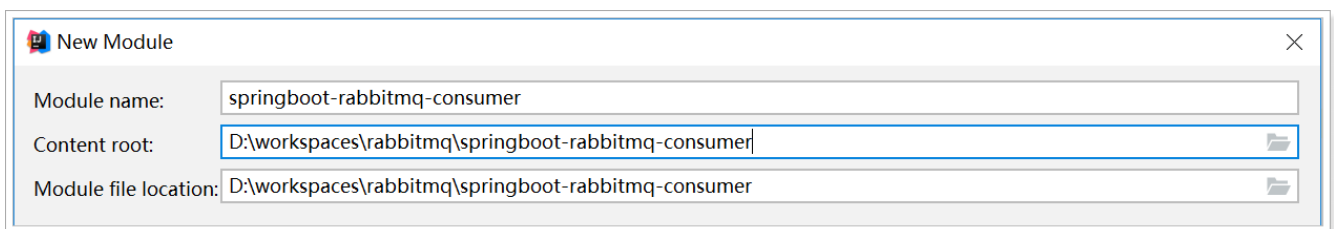
GroupId com.itheima

ArtifactId springboot-rabbitmq-consumer

Version 1.0-SNAPSHOT

☐ Inherit

☒ Inherit



New Module

Module name: springboot-rabbitmq-consumer

Content root: D:\workspaces\rabbitmq\springboot-rabbitmq-consumer

Module file location: D:\workspaces\rabbitmq\springboot-rabbitmq-consumer

### 5.3.2. 添加依赖

修改pom.xml文件内容为如下：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7     <parent>
8         <groupId>org.springframework.boot</groupId>
9         <artifactId>spring-boot-starter-parent</artifactId>
10        <version>2.1.4.RELEASE</version>
11    </parent>
12    <groupId>com.itheima</groupId>
13    <artifactId>springboot-rabbitmq-consumer</artifactId>
14    <version>1.0-SNAPSHOT</version>
15    <dependencies>
16        <dependency>
17            <groupId>org.springframework.boot</groupId>
18            <artifactId>spring-boot-starter-amqp</artifactId>
19        </dependency>
20    </dependencies>
21
22 </project>
```

### 5.3.3. 启动类

```
1 package com.ithouse.rabbitmq;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class ConsumerApplication {
8     public static void main(String[] args) {
9         SpringApplication.run(ConsumerApplication.class);
10    }
11 }
```

### 5.3.4. 配置RabbitMQ

创建application.yml，内容如下：

```
1 spring:
2   rabbitmq:
3     host: localhost
4     port: 5672
5     virtual-host:
6     /ithouseusername:
7     zspassword: zs
```

### 5.3.5. 消息监听处理类

编写消息监听器com.ithouse.rabbitmq.listener.MyListener

```
1 package com.ithouse.rabbitmq.listener;
2
3 import org.springframework.amqp.rabbit.annotation.RabbitListener;
4 import org.springframework.stereotype.Component;
5
6 @Component
7 public class MyListener {
8
9     /**
10      * 监听某个队列的消息
11      * @param message 接收到的消息
12      */
13     @RabbitListener(queues = "item_queue")
14     public void myListener1(String message){
15         System.out.println("消费者接收到的消息为: " + message);
16     }
17 }
```

## 5.4. 测试

在生产者工程springboot-rabbitmq-producer中创建测试类，发送消息：

```
1 package com.ithouse.rabbitmq;
2
3 import com.ithouse.rabbitmq.config.RabbitMQConfig;
4 import org.junit.Test;
5 import org.junit.runner.RunWith;
6 import org.springframework.amqp.rabbit.core.RabbitTemplate;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.boot.test.context.SpringBootTest;
9 import org.springframework.test.context.junit4.SpringRunner;
10
11 @RunWith(SpringRunner.class)
12 @SpringBootTest
13 public class RabbitMQTest {
14
15     @Autowired
16     private RabbitTemplate rabbitTemplate;
17
18     @Test
19     public void test(){
20         rabbitTemplate.convertAndSend(RabbitMQConfig.ITEM_TOPIC_EXCHANGE,
21             "item.insert", "商品新增, routing key 为item.insert");
22         rabbitTemplate.convertAndSend(RabbitMQConfig.ITEM_TOPIC_EXCHANGE,
23             "item.update", "商品修改, routing key 为item.update");
24         rabbitTemplate.convertAndSend(RabbitMQConfig.ITEM_TOPIC_EXCHANGE,
25             "item.delete", "商品删除, routing key 为item.delete");
26     }
27 }
```

先运行上述测试程序（交换机和队列才能先被声明和绑定），然后启动消费者；在消费者工程springboot-rabbitmq-consumer中控制台查看是否接收到对应消息。

另外；也可以在RabbitMQ的管理控制台中查看到交换机与队列的绑定：

## Details

|          |                      |
|----------|----------------------|
| Type     | topic                |
| Features | durable: <u>true</u> |
| Policy   |                      |

## ▼ Bindings

This exchange



| To         | Routing key | Arguments |        |
|------------|-------------|-----------|--------|
| item_queue | item.#      |           | Unbind |