# RabbitMQ实战教程

# 什么是MQ

消息队列（Message Queue，简称MQ），从字面意思上看，本质是个队列，FIFO先入先出，只不过队列中存放的内容是message而已。

其主要用途：不同进程Process/线程Thread之间通信。为什么会产生消息队列？有几个原因：

不同进程（process）之间传递消息时，两个进程之间耦合程度过高，改动一个进程，引发必须修改另一个进程，为了隔离这两个进程，在两进程间抽离出一层（一个模块），所有两进程之间传递的消息，都必须通过消息队列来传递，单独修改某一个进程，不会影响另一个；

不同进程（process）之间传递消息时，为了实现标准化，将消息的格式规范化了，并且，某一个进程接受的消息太多，一下子无法处理完，并且也有先后顺序，必须对收到的消息进行排队，因此诞生了事实上的消息队列；

　　关于消息队列的详细介绍请参阅：

[《Java帝国之消息队列》](https://mp.weixin.qq.com/s?__biz=MzAxOTc0NzExNg==&mid=2665513507&idx=1&sn=d6db79c1ae03ba9260fb0fb77727bb54&chksm=80d67a60b7a1f376e7ad1e2c3276e8b565f045b1c7e21ef90926f69d99f969557737eb5d8128&mpshare=1&scene=1&srcid=1019awkBx8kaLyFohcuW4Ee7" \l "rd)

[《一个故事告诉你什么是消息队列》](https://github.com/jasonGeng88/blog/blob/master/201705/MQ.md)

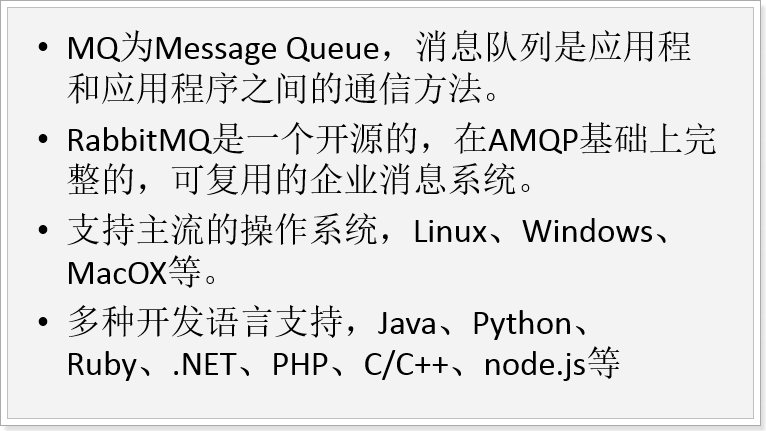
[《到底什么时候该使用MQ》](https://mp.weixin.qq.com/s?__biz=MjM5ODYxMDA5OQ==&mid=2651960012&idx=1&sn=c6af5c79ecead98daa4d742e5ad20ce5&chksm=bd2d07108a5a8e0624ae6ad95001c4efe09d7ba695f2ddb672064805d771f3f84bee8123b8a6&mpshare=1&scene=1&srcid=04054h4e90lz5Qc2YKnLNuvY" \l "rd)

MQ框架非常之多，比较流行的有RabbitMq、ActiveMq、ZeroMq、kafka，以及阿里开源的[RocketMQ](http://www.baidu.com/link?url=zOKzcPj5SK9o69mlCOl2wZ558CvgfY_73pWhTfl0HephpijE8pBR8ToHmbdtwLYp&ck=3632.7.1532847711083.0.0.229.144.0&shh=www.baidu.com&sht=baiduhome_pg" \t "https://www.baidu.com/_blank)。本文主要介绍RabbitMq。

本教程代码下载地址：<https://download.csdn.net/download/zpcandzhj/10585077>

# RabbitMQ

## RabbitMQ的简介



开发语言：Erlang – 面向并发的编程语言。

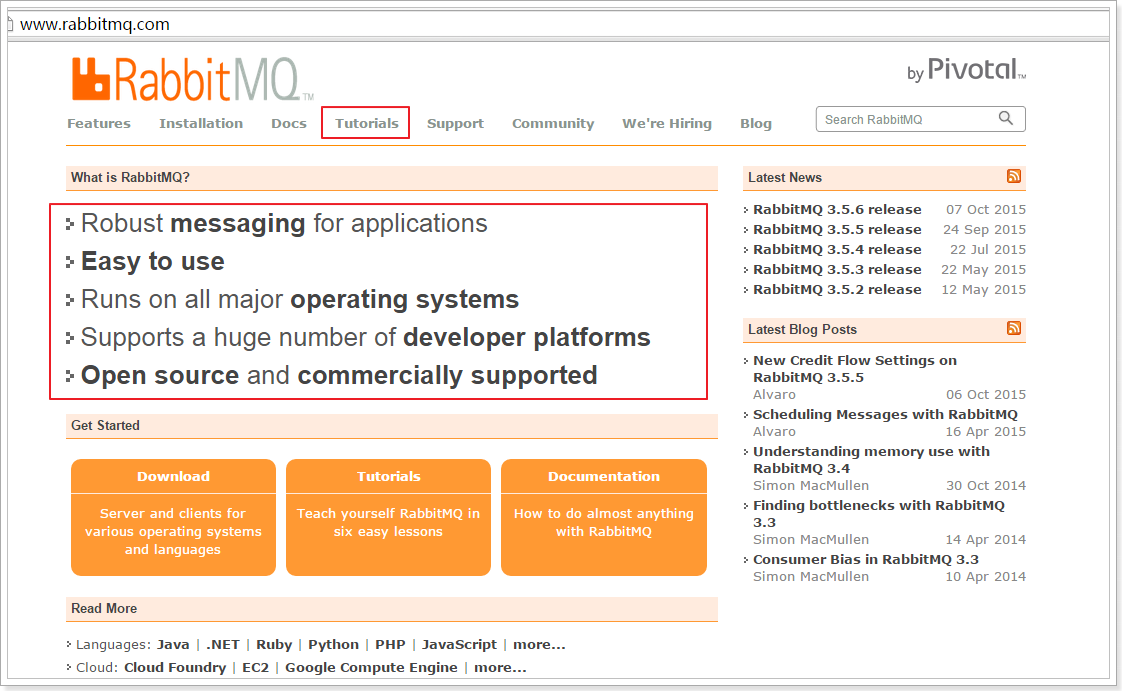


### AMQP

AMQP是消息队列的一个协议。



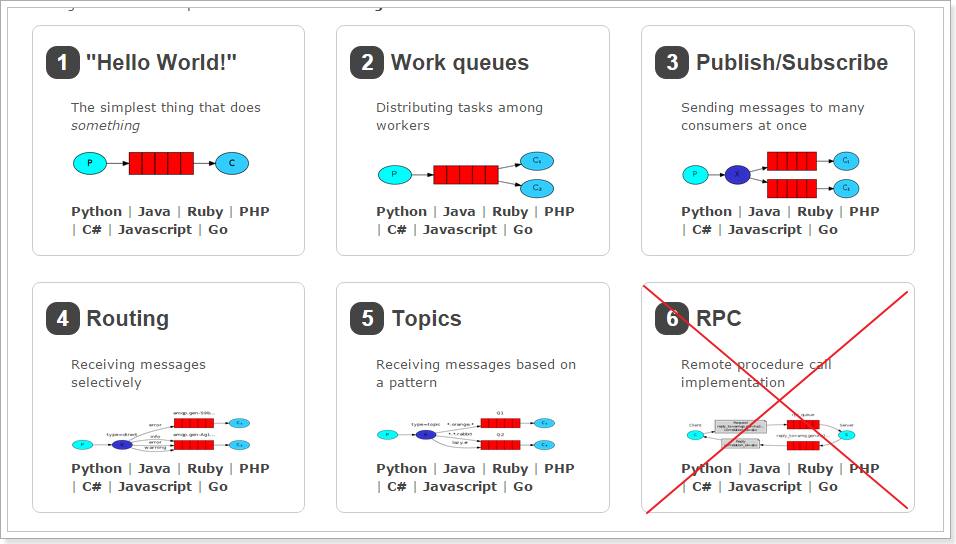
## 官网



## MQ的其他产品



## 学习5种队列



## 安装文档



# 搭建RabbitMQ环境

## 下载

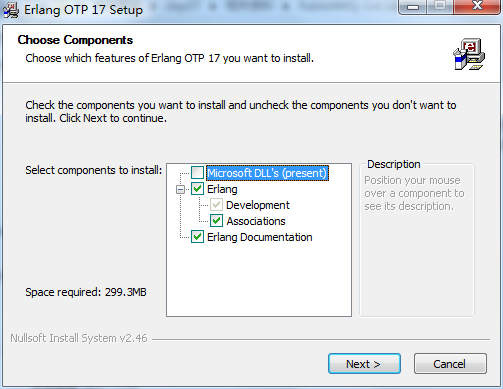
下载地址：<http://www.rabbitmq.com/download.html>

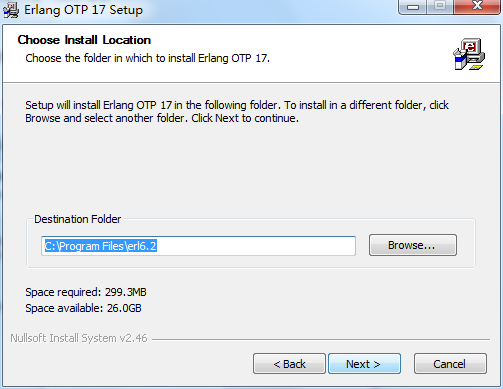
## windows下安装

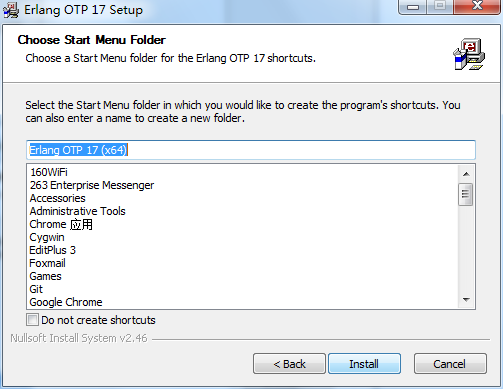
### 安装Erlang

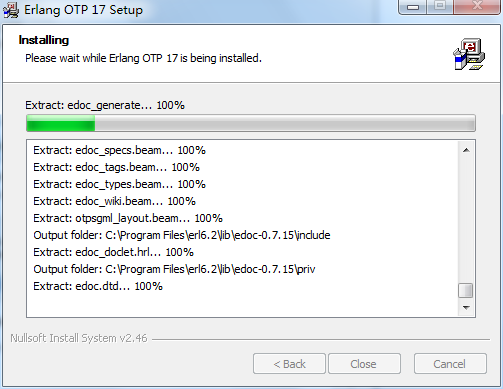
下载：<http://www.erlang.org/download/otp_win64_17.3.exe>

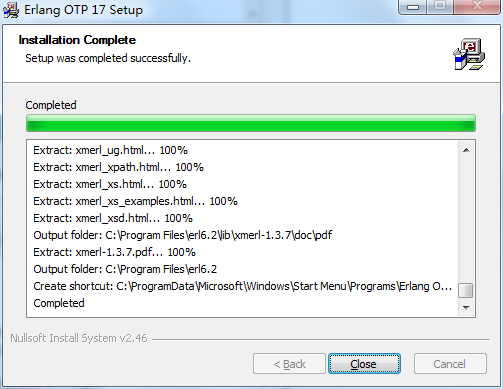
安装：





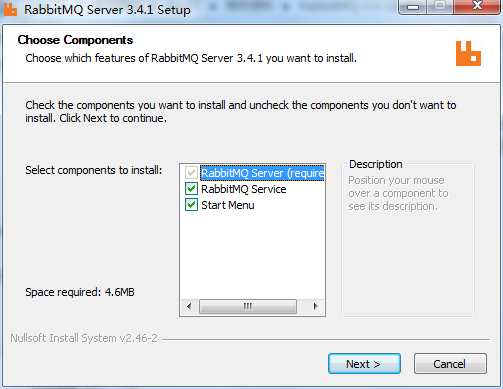


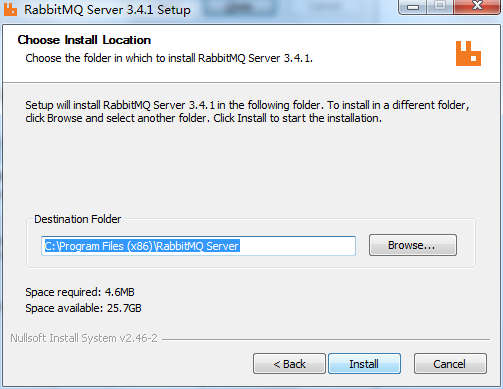


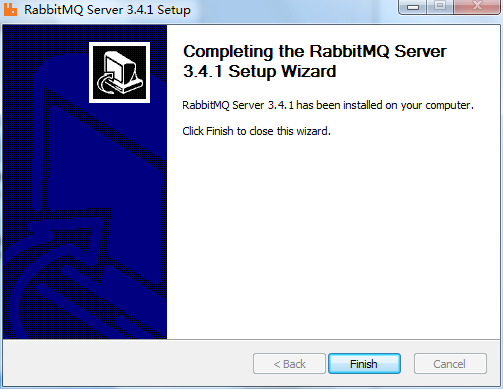


安装完成。

### 安装RabbitMQ

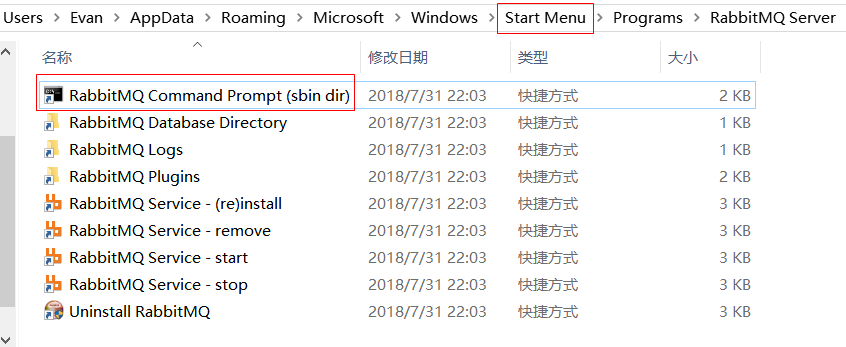






安装完成。

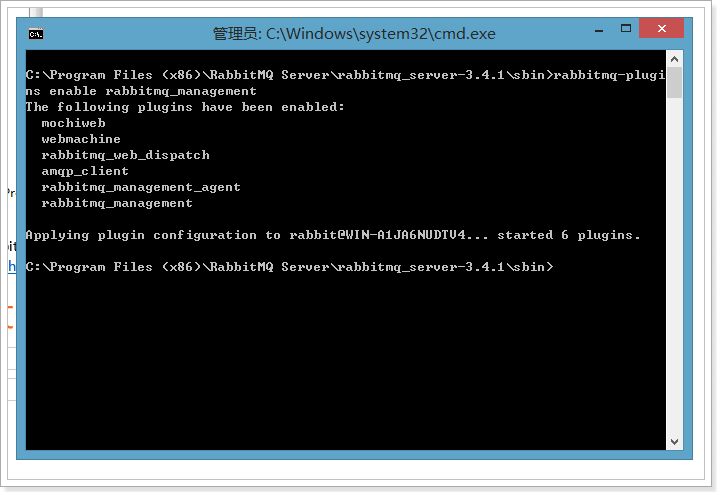
开始菜单里出现如下选项：



启动、停止、重新安装等。

### 启用管理工具

1. 双击
2. 进入C:\Program Files (x86)\RabbitMQ Server\rabbitmq\_server-3.4.1\sbin输入命令：

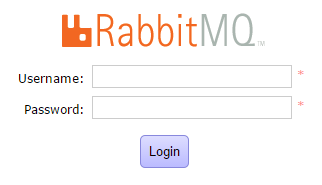
rabbitmq-plugins enable rabbitmq\_management  
  


这样就启动了管理工具，可以试一下命令：

停止：net stop RabbitMQ

启动：net start RabbitMQ

1. 在浏览器中输入地址查看：<http://127.0.0.1:15672/>



1. 使用默认账号登录：guest/ guest

## Linux下安装

### 安装Erlang

### 添加yum支持

cd /usr/local/src/

mkdir rabbitmq

cd rabbitmq

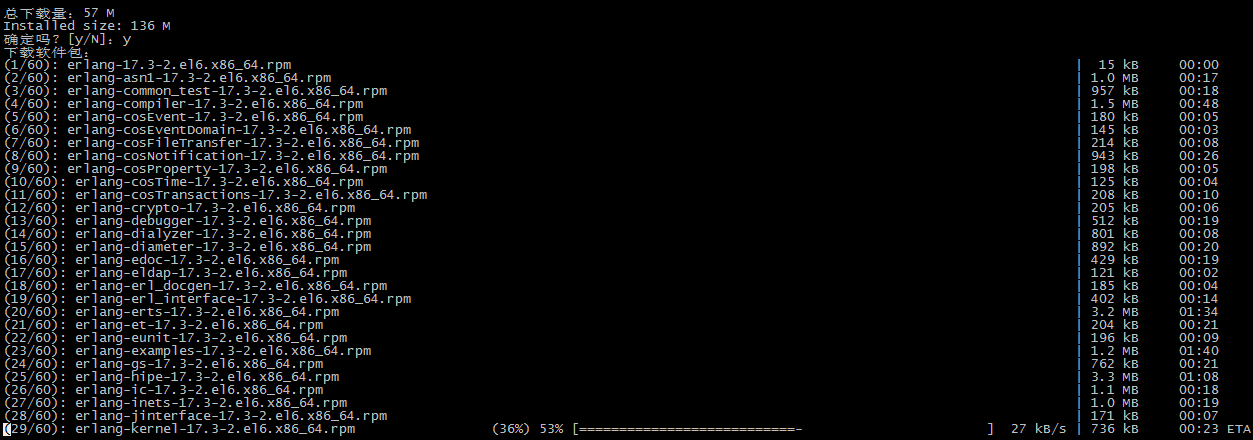
wget <http://packages.erlang-solutions.com/erlang-solutions-1.0-1.noarch.rpm>

rpm -Uvh erlang-solutions-1.0-1.noarch.rpm

rpm --import <http://packages.erlang-solutions.com/rpm/erlang_solutions.asc>

使用yum安装：

sudo yum install erlang



### 安装RabbitMQ

上传rabbitmq-server-3.4.1-1.noarch.rpm文件到/usr/local/src/rabbitmq/

安装：

rpm -ivh rabbitmq-server-3.4.1-1.noarch.rpm

### 启动、停止

service rabbitmq-server start

service rabbitmq-server stop

service rabbitmq-server restart

### 设置开机启动

chkconfig rabbitmq-server on

### 设置配置文件

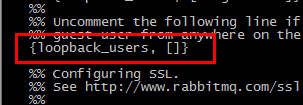
cd /etc/rabbitmq

cp /usr/share/doc/rabbitmq-server-3.4.1/rabbitmq.config.example /etc/rabbitmq/

mv rabbitmq.config.example rabbitmq.config

### 开启用户远程访问

vi /etc/rabbitmq/rabbitmq.config



**注意要去掉后面的逗号。**

### 开启web界面管理工具

rabbitmq-plugins enable rabbitmq\_management

service rabbitmq-server restart

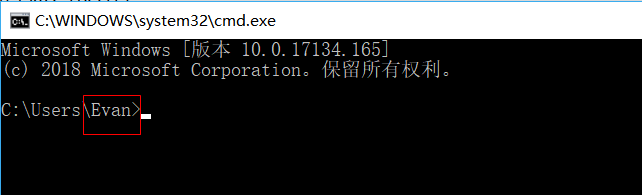
### 防火墙开放15672端口

/sbin/iptables -I INPUT -p tcp --dport 15672 -j ACCEPT

/etc/rc.d/init.d/iptables save

## 安装的注意事项

1. 推荐使用默认的安装路径
2. 系统用户名必须是英文

Win10改名字非常麻烦，具体方法百度  
 

1. 计算机名必须是英文

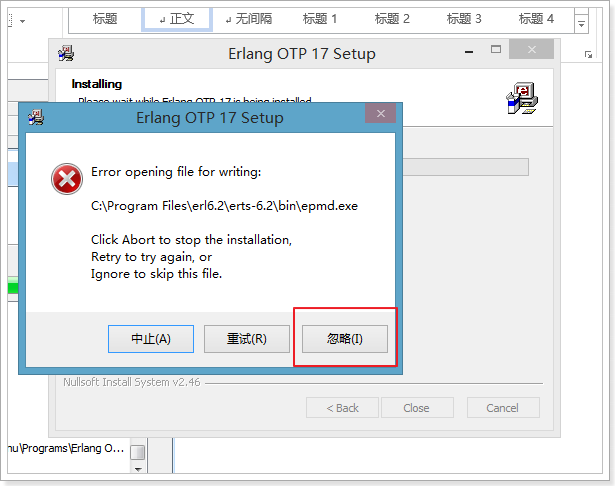


1. 系统的用户必须是管理员

如果安装失败应该如何解决：

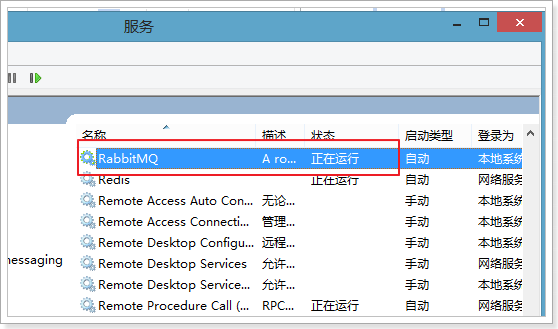
1. 重装系统 -- 不推荐
2. 将RabbitMQ安装到linux虚拟机中
   1. 推荐
3. 使用别人安装好的RabbitMQ服务
   1. 只要给你开通一个账户即可。
   2. 使用公用的RabbitMQ服务，在192.168.50.22
   3. 推荐

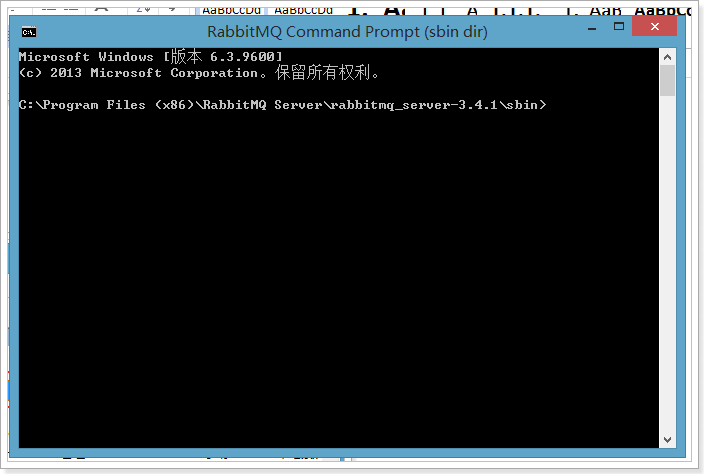
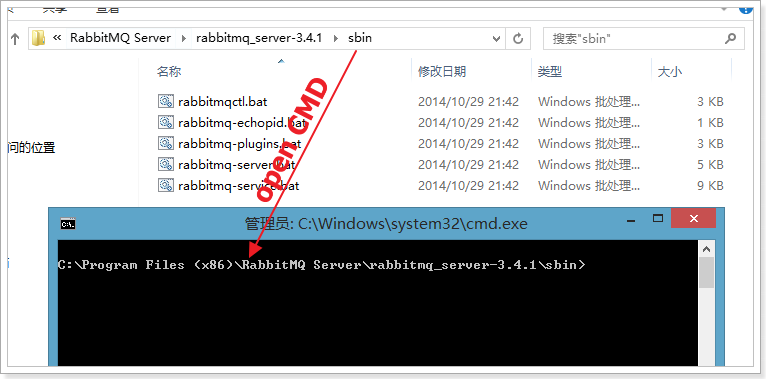
常见错误：

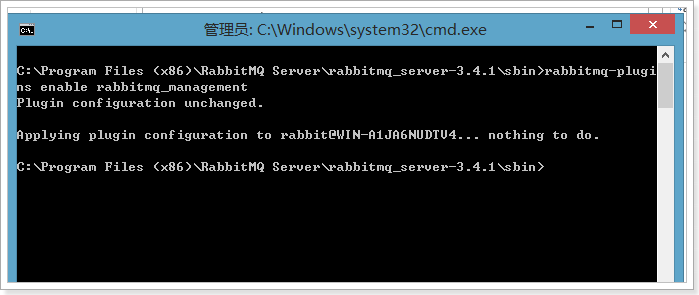


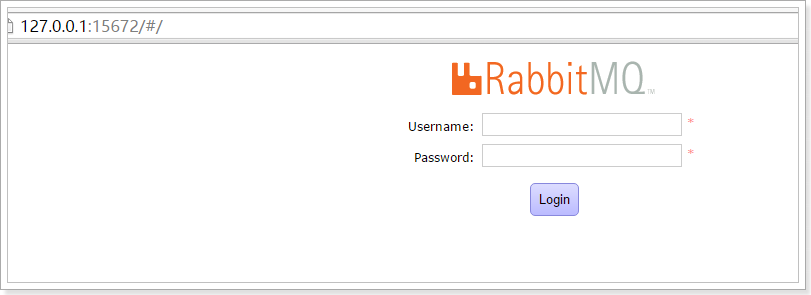
## 安装完成后操作

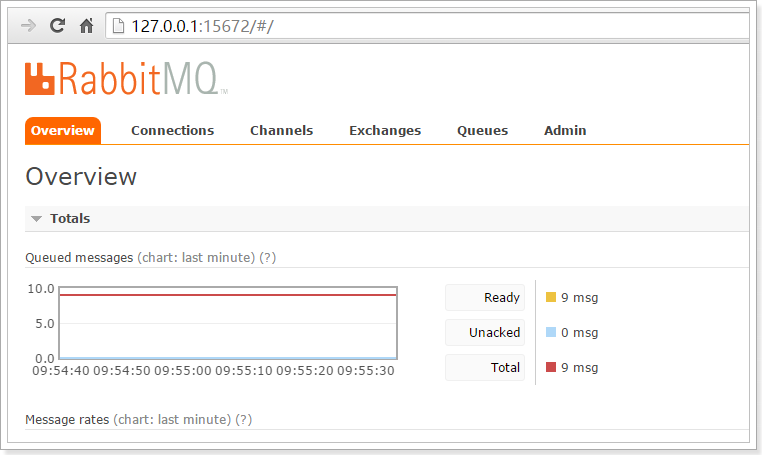
1、系统服务中有RabbitMQ服务，停止、启动、重启



2、打开命令行工具  
  
如果找不到命令行工具：  


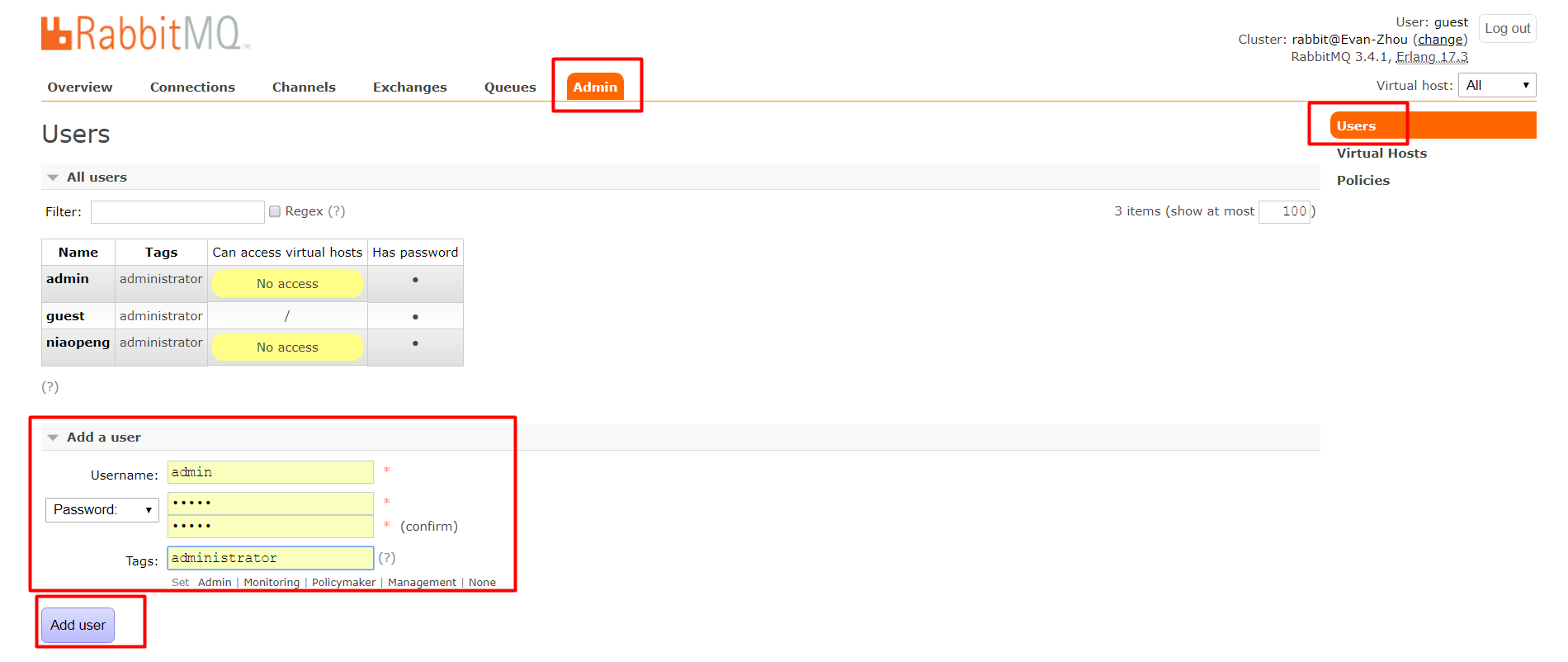
输入命令rabbitmq-plugins enable rabbitmq\_management启用管理插件  


查看管理页面  


通过默认账户 guest/guest 登录  
如果能够登录，说明安装成功。  


# 添加用户

## 添加admin用户



## 用户角色

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。

1. 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)

1. 策略制定者(policymaker)

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)。

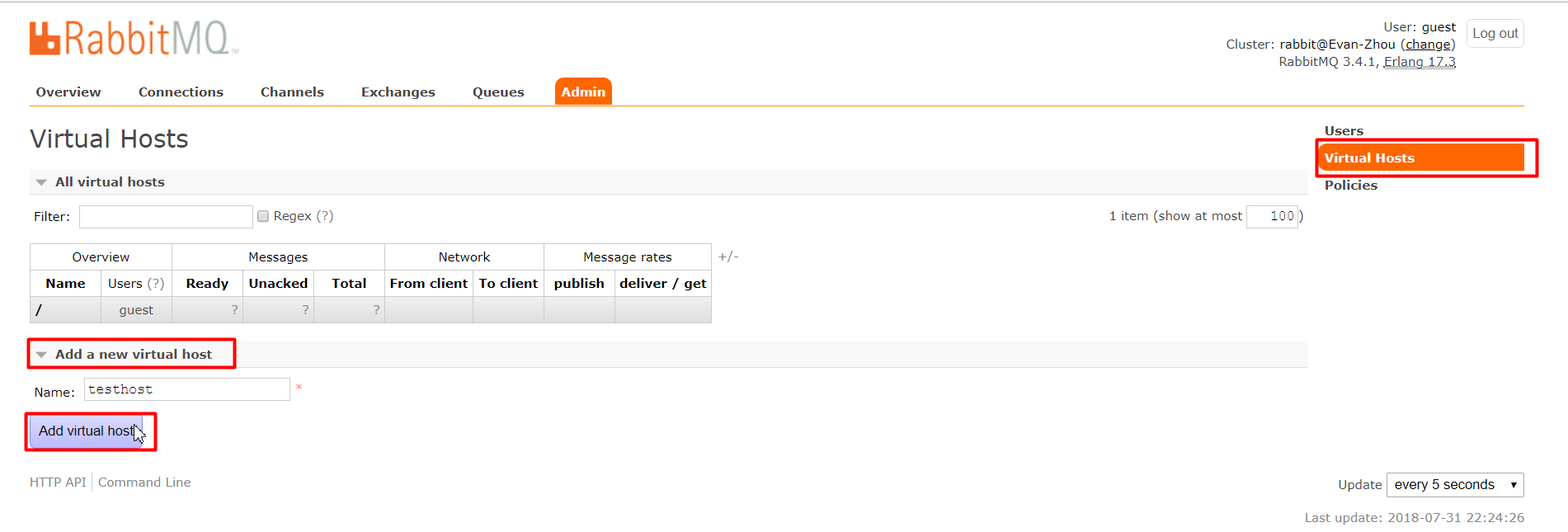
1. 普通管理者(management)

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。

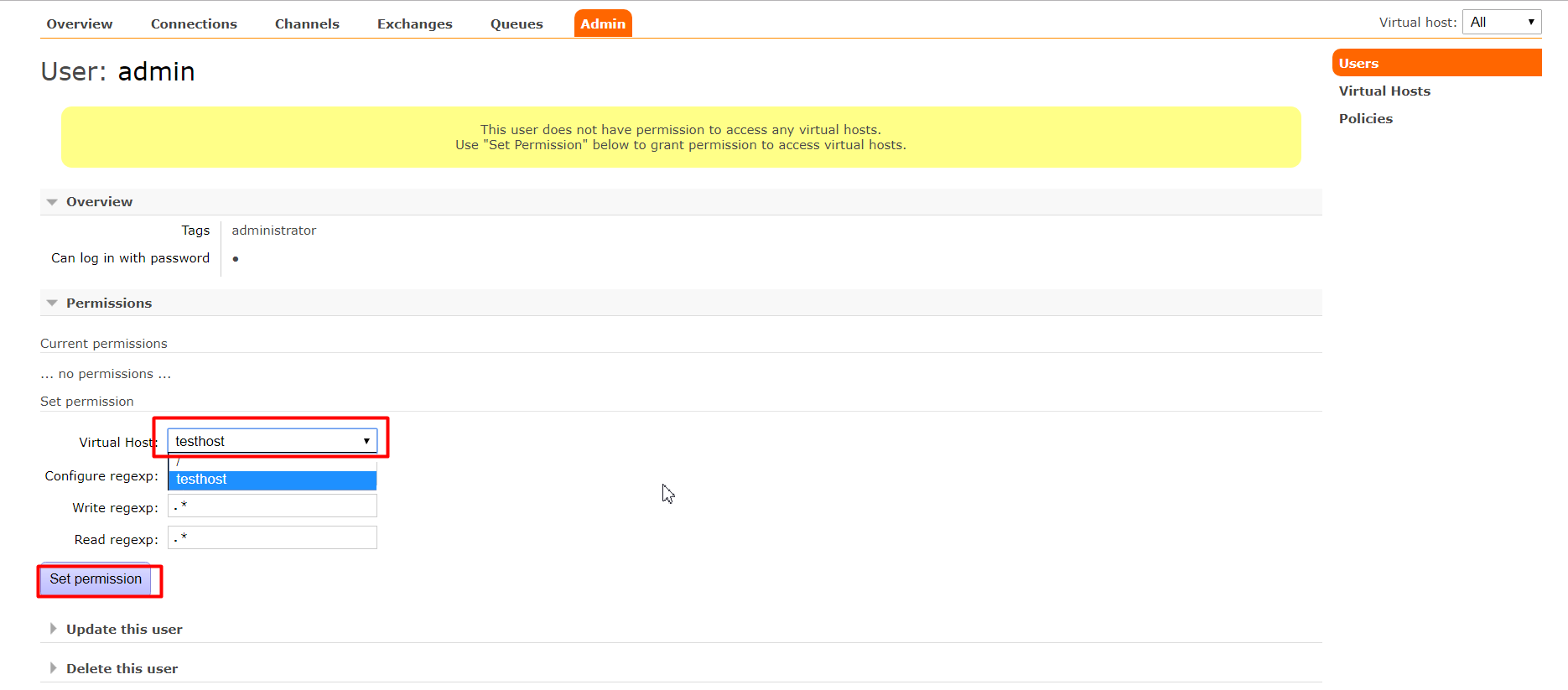
1. 其他

无法登陆管理控制台，通常就是普通的生产者和消费者。

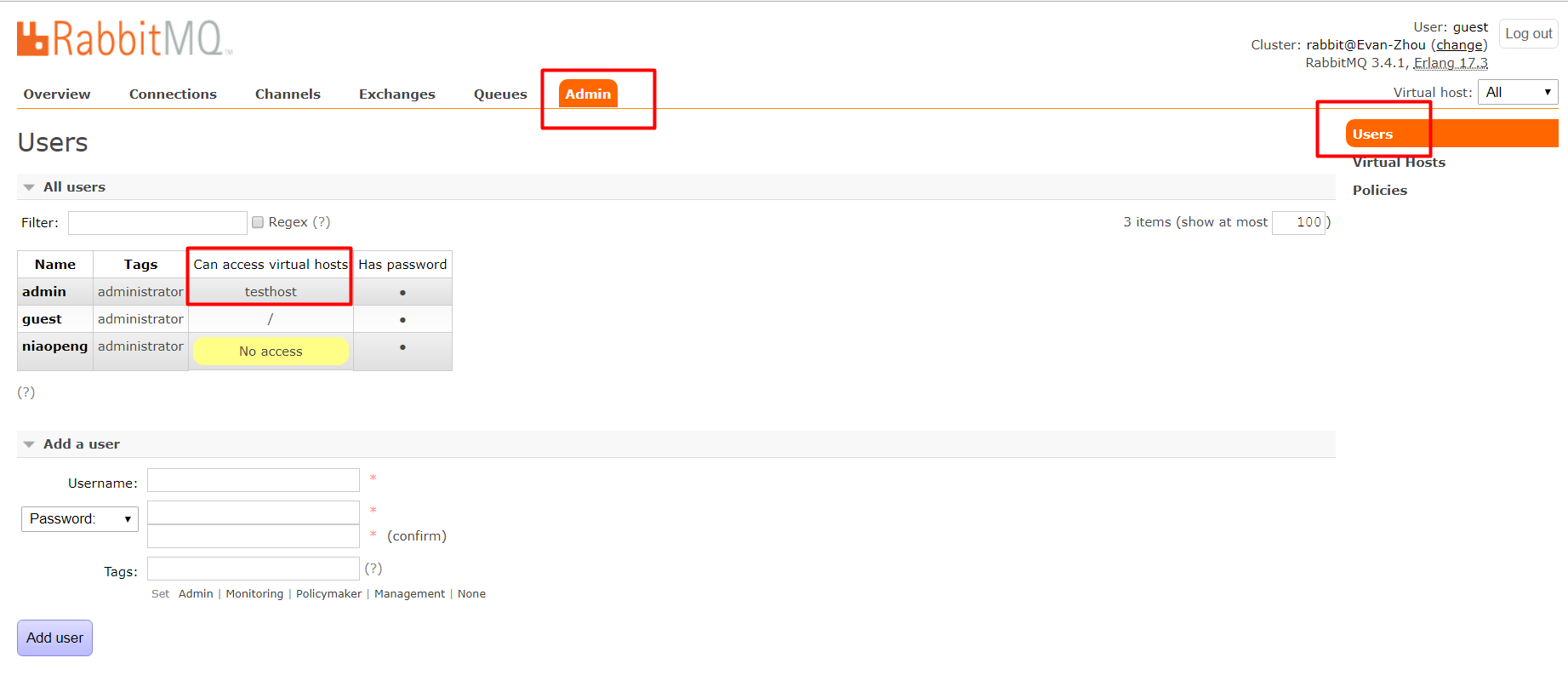
## 创建Virtual Hosts



选中Admin用户，设置权限：

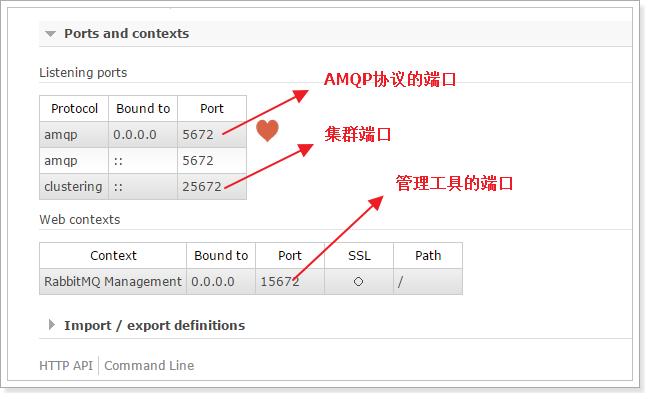


看到权限已加：

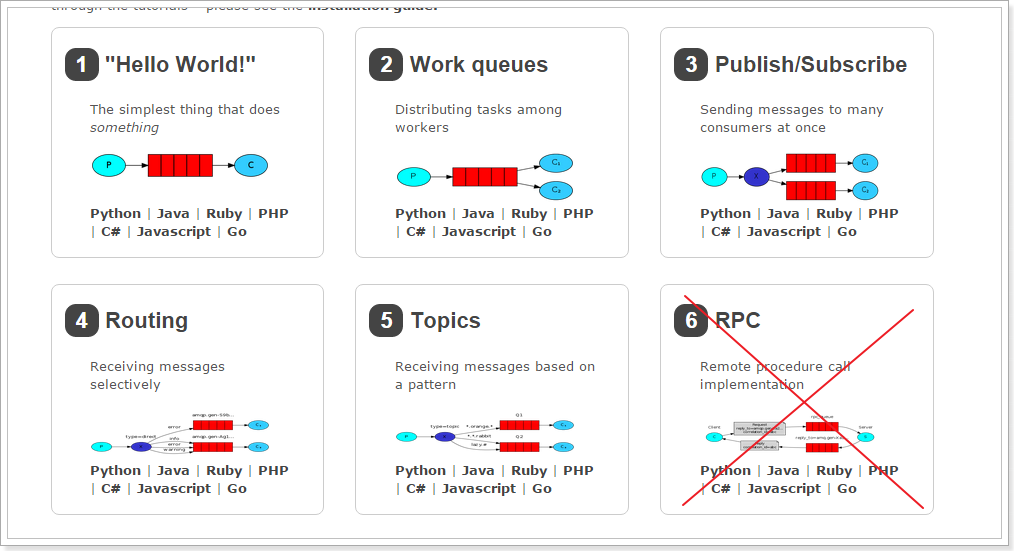


## 管理界面中的功能



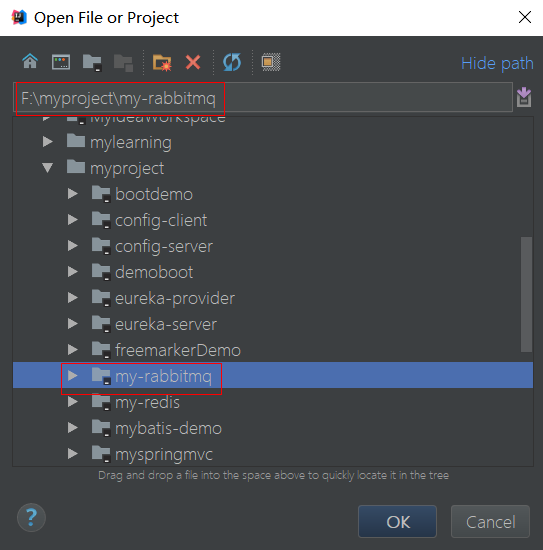


# 五种队列



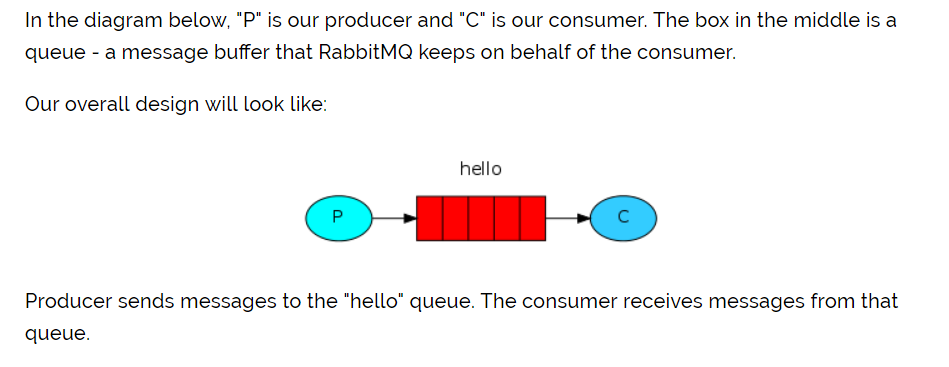
## 导入my-rabbitmq

项目下载地址：



## 简单队列

### 图示



P：消息的生产者

C：消息的消费者

红色：队列

生产者将消息发送到队列，消费者从队列中获取消息。

### 导入RabbitMQ的客户端依赖

<dependency>  
 <groupId>com.rabbitmq</groupId>  
 <artifactId>amqp-client</artifactId>  
 <version>3.4.1</version>  
</dependency>

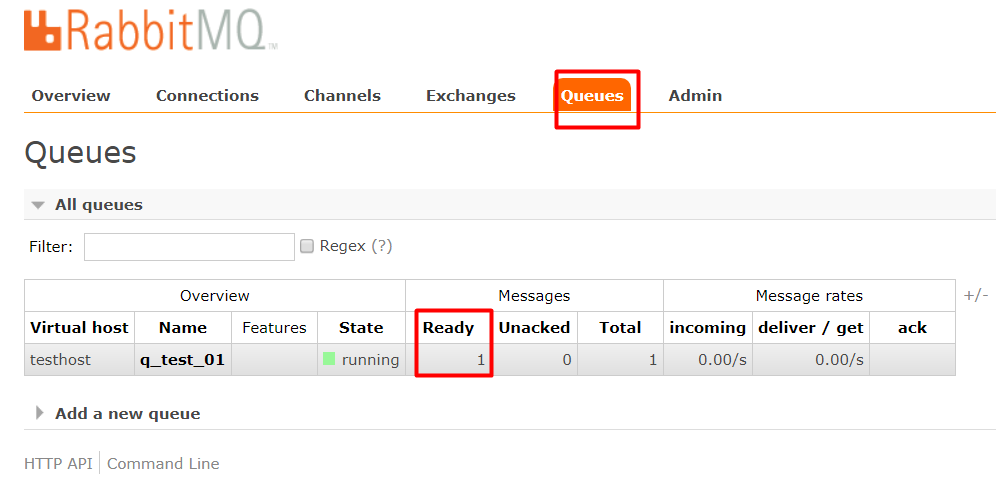
### 获取MQ的连接

package com.zpc.rabbitmq.util;  
import com.rabbitmq.client.ConnectionFactory;  
import com.rabbitmq.client.Connection;  
  
public class ConnectionUtil {  
  
 public static Connection getConnection() throws Exception {  
 //定义连接工厂  
 ConnectionFactory factory = new ConnectionFactory();  
 //设置服务地址  
 factory.setHost("localhost");  
 //端口  
 factory.setPort(5672);  
 //设置账号信息，用户名、密码、vhost  
 factory.setVirtualHost("testhost");  
 factory.setUsername("admin");  
 factory.setPassword("admin");  
 // 通过工程获取连接  
 Connection connection = factory.newConnection();  
 return connection;  
 }  
}

### 生产者发送消息到队列

package com.zpc.rabbitmq.simple;  
  
import com.zpc.rabbitmq.util.ConnectionUtil;  
  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.Connection;  
  
public class Send {  
  
 private final static String ***QUEUE\_NAME*** = "q\_test\_01";  
  
 public static void main(String[] argv) throws Exception {  
 // 获取到连接以及mq通道  
 Connection connection = ConnectionUtil.getConnection();  
 // 从连接中创建通道  
 Channel channel = connection.createChannel();  
  
 // 声明（创建）队列  
 channel.queueDeclare(***QUEUE\_NAME***, false, false, false, null);  
  
 // 消息内容  
 String message = "Hello World!";  
 channel.basicPublish("", ***QUEUE\_NAME***, null, message.getBytes());  
 System.***out***.println(" [x] Sent '" + message + "'");  
 //关闭通道和连接  
 channel.close();  
 connection.close();  
 }  
}

### 管理工具中查看消息



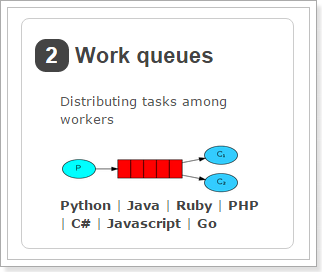
点击上面的队列名称，查询具体的队列中的信息：



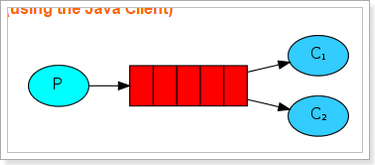
### 消费者从队列中获取消息

package com.zpc.rabbitmq.simple;  
  
import com.zpc.rabbitmq.util.ConnectionUtil;  
  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.Connection;  
import com.rabbitmq.client.QueueingConsumer;  
  
public class Recv {  
  
 private final static String ***QUEUE\_NAME*** = "q\_test\_01";  
  
 public static void main(String[] argv) throws Exception {  
  
 // 获取到连接以及mq通道  
 Connection connection = ConnectionUtil.getConnection();  
 // 从连接中创建通道  
 Channel channel = connection.createChannel();  
 // 声明队列  
 channel.queueDeclare(***QUEUE\_NAME***, false, false, false, null);  
  
 // 定义队列的消费者  
 QueueingConsumer consumer = new QueueingConsumer(channel);  
  
 // 监听队列  
 channel.basicConsume(***QUEUE\_NAME***, true, consumer);  
  
 // 获取消息  
 while (true) {  
 QueueingConsumer.Delivery delivery = consumer.nextDelivery();  
 String message = new String(delivery.getBody());  
 System.***out***.println(" [x] Received '" + message + "'");  
 }  
 }  
}

## Work模式



### 图示



一个生产者、2个消费者。

一个消息只能被一个消费者获取。

### 消费者1

package com.zpc.rabbitmq.work;  
  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.Connection;  
import com.rabbitmq.client.QueueingConsumer;  
import com.zpc.rabbitmq.util.ConnectionUtil;  
  
public class Recv {  
  
 private final static String ***QUEUE\_NAME*** = "test\_queue\_work";  
  
 public static void main(String[] argv) throws Exception {  
  
 // 获取到连接以及mq通道  
 Connection connection = ConnectionUtil.getConnection();  
 Channel channel = connection.createChannel();  
  
 // 声明队列  
 channel.queueDeclare(***QUEUE\_NAME***, false, false, false, null);  
  
 // 同一时刻服务器只会发一条消息给消费者  
 //channel.basicQos(1);  
  
 // 定义队列的消费者  
 QueueingConsumer consumer = new QueueingConsumer(channel);  
 // 监听队列，false表示手动返回完成状态，true表示自动  
 channel.basicConsume(***QUEUE\_NAME***, true, consumer);  
  
 // 获取消息  
 while (true) {  
 QueueingConsumer.Delivery delivery = consumer.nextDelivery();  
 String message = new String(delivery.getBody());  
 System.***out***.println(" [y] Received '" + message + "'");  
 //休眠  
 Thread.sleep(10);  
 // 返回确认状态，注释掉表示使用自动确认模式  
 //channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);  
 }  
 }  
}

### 消费者2

package com.zpc.rabbitmq.work;  
  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.Connection;  
import com.rabbitmq.client.QueueingConsumer;  
import com.zpc.rabbitmq.util.ConnectionUtil;  
  
public class Recv2 {  
  
 private final static String ***QUEUE\_NAME*** = "test\_queue\_work";  
  
 public static void main(String[] argv) throws Exception {  
  
 // 获取到连接以及mq通道  
 Connection connection = ConnectionUtil.getConnection();  
 Channel channel = connection.createChannel();  
  
 // 声明队列  
 channel.queueDeclare(***QUEUE\_NAME***, false, false, false, null);  
  
 // 同一时刻服务器只会发一条消息给消费者  
 //channel.basicQos(1);  
  
 // 定义队列的消费者  
 QueueingConsumer consumer = new QueueingConsumer(channel);  
 // 监听队列，false表示手动返回完成状态，true表示自动  
 channel.basicConsume(***QUEUE\_NAME***, true, consumer);  
  
 // 获取消息  
 while (true) {  
 QueueingConsumer.Delivery delivery = consumer.nextDelivery();  
 String message = new String(delivery.getBody());  
 System.***out***.println(" [x] Received '" + message + "'");  
 // 休眠1秒  
 Thread.sleep(1000);  
 //下面这行注释掉表示使用自动确认模式  
 //channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);  
 }  
 }  
}

### 生产者

向队列中发送100条消息。

package com.zpc.rabbitmq.work;  
  
import com.zpc.rabbitmq.util.ConnectionUtil;  
  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.Connection;  
  
public class Send {  
  
 private final static String ***QUEUE\_NAME*** = "test\_queue\_work";  
  
 public static void main(String[] argv) throws Exception {  
 // 获取到连接以及mq通道  
 Connection connection = ConnectionUtil.getConnection();  
 Channel channel = connection.createChannel();  
  
 // 声明队列  
 channel.queueDeclare(***QUEUE\_NAME***, false, false, false, null);  
  
 for (int i = 0; i < 100; i++) {  
 // 消息内容  
 String message = "" + i;  
 channel.basicPublish("", ***QUEUE\_NAME***, null, message.getBytes());  
 System.***out***.println(" [x] Sent '" + message + "'");  
  
 Thread.sleep(i \* 10);  
 }  
  
 channel.close();  
 connection.close();  
 }  
}

### 测试

测试结果：

1. 消费者1和消费者2获取到的消息内容是不同的，同一个消息只能被一个消费者获取。
2. 消费者1和消费者2获取到的消息的数量是相同的，一个是消费奇数号消息，一个是偶数。

其实，这样是不合理的，因为消费者1线程停顿的时间短。应该是消费者1要比消费者2获取到的消息多才对。

RabbitMQ 默认将消息顺序发送给下一个消费者，这样，每个消费者会得到相同数量的消息。即轮询（round-robin）分发消息。

怎样才能做到按照每个消费者的能力分配消息呢？联合使用 Qos 和 Acknowledge 就可以做到。

basicQos 方法设置了当前信道最大预获取（prefetch）消息数量为1。消息从队列异步推送给消费者，消费者的 ack 也是异步发送给队列，从队列的视角去看，总是会有一批消息已推送但尚未获得 ack 确认，Qos 的 prefetchCount 参数就是用来限制这批未确认消息数量的。设为1时，队列只有在收到消费者发回的上一条消息 ack 确认后，才会向该消费者发送下一条消息。prefetchCount 的默认值为0，即没有限制，队列会将所有消息尽快发给消费者。

2个概念

**轮询分发** ：使用任务队列的优点之一就是可以轻易的并行工作。如果我们积压了好多工作，我们可以通过增加工作者（消费者）来解决这一问题，使得系统的伸缩性更加容易。在默认情况下，RabbitMQ将逐个发送消息到在序列中的下一个消费者(而不考虑每个任务的时长等等，且是提前一次性分配，并非一个一个分配)。平均每个消费者获得相同数量的消息。这种方式分发消息机制称为Round-Robin（轮询）。

**公平分发** ：虽然上面的分配法方式也还行，但是有个问题就是：比如：现在有2个消费者，所有的奇数的消息都是繁忙的，而偶数则是轻松的。按照轮询的方式，奇数的任务交给了第一个消费者，所以一直在忙个不停。偶数的任务交给另一个消费者，则立即完成任务，然后闲得不行。而RabbitMQ则是不了解这些的。这是因为当消息进入队列，RabbitMQ就会分派消息。它不看消费者为应答的数目，只是盲目的将消息发给轮询指定的消费者。

为了解决这个问题，我们使用basicQos( prefetchCount = 1)方法，来限制RabbitMQ只发不超过1条的消息给同一个消费者。当消息处理完毕后，有了反馈，才会进行第二次发送。

还有一点需要注意，使用公平分发，必须关闭自动应答，改为手动应答。

## Work模式的“能者多劳”

打开上述代码的注释：

// 同一时刻服务器只会发一条消息给消费者  
channel.basicQos(1);

//开启这行 表示使用手动确认模式  
channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);

同时改为手动确认：

// 监听队列，false表示手动返回完成状态，true表示自动  
channel.basicConsume(***QUEUE\_NAME***, false, consumer);

测试：

消费者1比消费者2获取的消息更多。

## 消息的确认模式

消费者从队列中获取消息，服务端如何知道消息已经被消费呢？

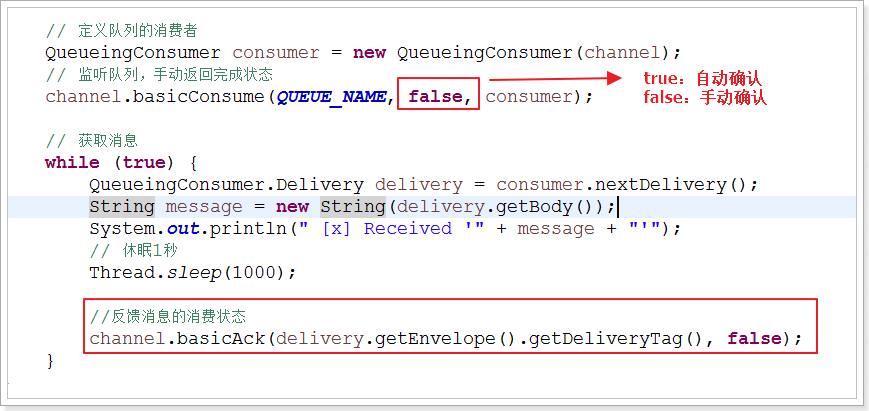
模式1：自动确认

只要消息从队列中获取，无论消费者获取到消息后是否成功消息，都认为是消息已经成功消费。

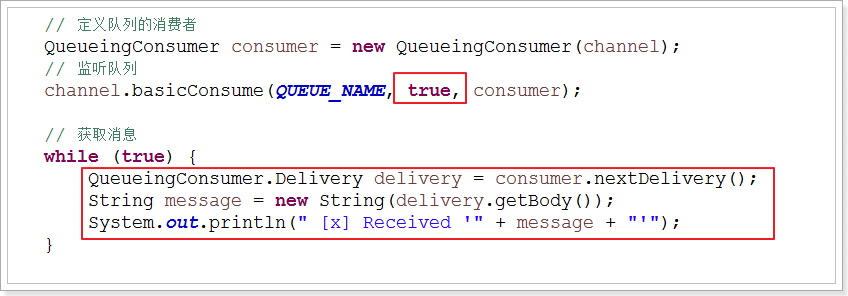
模式2：手动确认

消费者从队列中获取消息后，服务器会将该消息标记为不可用状态，等待消费者的反馈，如果消费者一直没有反馈，那么该消息将一直处于不可用状态。

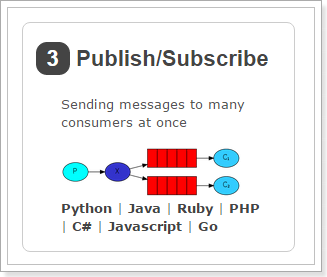
手动模式：



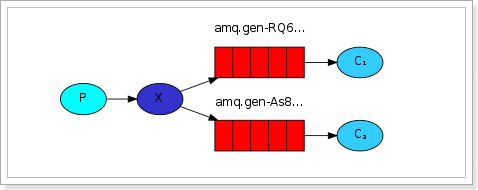
自动模式：



## 订阅模式



### 图示



解读：

1、1个生产者，多个消费者

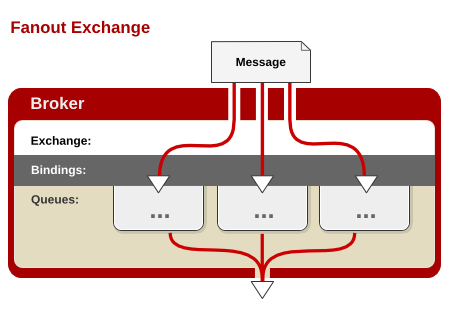
2、每一个消费者都有自己的一个队列

3、生产者没有将消息直接发送到队列，而是发送到了交换机

4、每个队列都要绑定到交换机

5、生产者发送的消息，经过交换机，到达队列，实现，一个消息被多个消费者获取的目的

注意：一个消费者队列可以有多个消费者实例，只有其中一个消费者实例会消费



### 消息的生产者（看作是后台系统）

向交换机中发送消息。

package com.zpc.rabbitmq.subscribe;  
  
import com.zpc.rabbitmq.util.ConnectionUtil;  
  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.Connection;  
  
public class Send {  
  
 private final static String ***EXCHANGE\_NAME*** = "test\_exchange\_fanout";  
  
 public static void main(String[] argv) throws Exception {  
 // 获取到连接以及mq通道  
 Connection connection = ConnectionUtil.getConnection();  
 Channel channel = connection.createChannel();  
  
 // 声明exchange  
 channel.exchangeDeclare(***EXCHANGE\_NAME***, "fanout");  
  
 // 消息内容  
 String message = "Hello World!";  
 channel.basicPublish(***EXCHANGE\_NAME***, "", null, message.getBytes());  
 System.***out***.println(" [x] Sent '" + message + "'");  
  
 channel.close();  
 connection.close();  
 }  
}

注意：消息发送到没有队列绑定的交换机时，消息将丢失，因为，交换机没有存储消息的能力，消息只能存在在队列中。

### 消费者1（看作是前台系统）

package com.zpc.rabbitmq.subscribe;  
  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.Connection;  
import com.rabbitmq.client.QueueingConsumer;  
  
import com.zpc.rabbitmq.util.ConnectionUtil;  
  
public class Recv {  
  
 private final static String ***QUEUE\_NAME*** = "test\_queue\_work1";  
  
 private final static String ***EXCHANGE\_NAME*** = "test\_exchange\_fanout";  
  
 public static void main(String[] argv) throws Exception {  
  
 // 获取到连接以及mq通道  
 Connection connection = ConnectionUtil.getConnection();  
 Channel channel = connection.createChannel();  
  
 // 声明队列  
 channel.queueDeclare(***QUEUE\_NAME***, false, false, false, null);  
  
 // 绑定队列到交换机  
 channel.queueBind(***QUEUE\_NAME***, ***EXCHANGE\_NAME***, "");  
  
 // 同一时刻服务器只会发一条消息给消费者  
 channel.basicQos(1);  
  
 // 定义队列的消费者  
 QueueingConsumer consumer = new QueueingConsumer(channel);  
 // 监听队列，手动返回完成  
 channel.basicConsume(***QUEUE\_NAME***, false, consumer);  
  
 // 获取消息  
 while (true) {  
 QueueingConsumer.Delivery delivery = consumer.nextDelivery();  
 String message = new String(delivery.getBody());  
 System.***out***.println(" [Recv] Received '" + message + "'");  
 Thread.sleep(10);  
  
 channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);  
 }  
 }  
}

### 消费者2（看作是搜索系统）

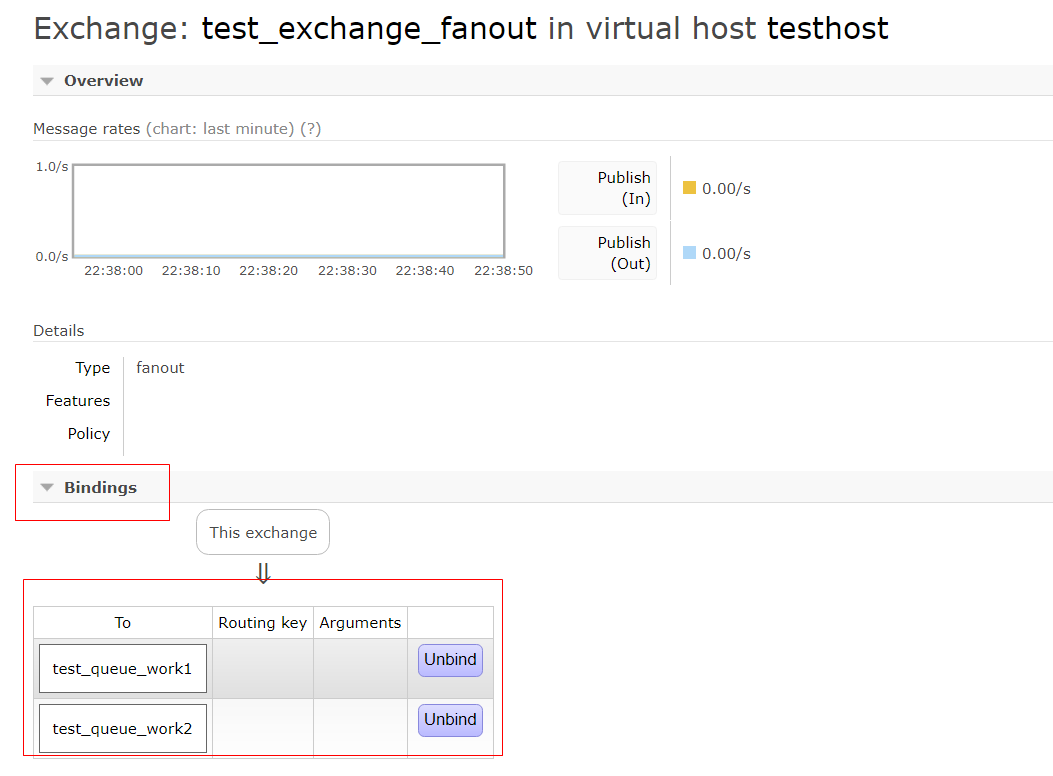
package com.zpc.rabbitmq.subscribe;  
  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.Connection;  
import com.rabbitmq.client.QueueingConsumer;  
  
import com.zpc.rabbitmq.util.ConnectionUtil;  
  
public class Recv2 {  
  
 private final static String ***QUEUE\_NAME*** = "test\_queue\_work2";  
  
 private final static String ***EXCHANGE\_NAME*** = "test\_exchange\_fanout";  
  
 public static void main(String[] argv) throws Exception {  
  
 // 获取到连接以及mq通道  
 Connection connection = ConnectionUtil.getConnection();  
 Channel channel = connection.createChannel();  
  
 // 声明队列  
 channel.queueDeclare(***QUEUE\_NAME***, false, false, false, null);  
  
 // 绑定队列到交换机  
 channel.queueBind(***QUEUE\_NAME***, ***EXCHANGE\_NAME***, "");  
  
 // 同一时刻服务器只会发一条消息给消费者  
 channel.basicQos(1);  
  
 // 定义队列的消费者  
 QueueingConsumer consumer = new QueueingConsumer(channel);  
 // 监听队列，手动返回完成  
 channel.basicConsume(***QUEUE\_NAME***, false, consumer);  
  
 // 获取消息  
 while (true) {  
 QueueingConsumer.Delivery delivery = consumer.nextDelivery();  
 String message = new String(delivery.getBody());  
 System.***out***.println(" [Recv2] Received '" + message + "'");  
 Thread.sleep(10);  
  
 channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);  
 }  
 }  
}

### 测试

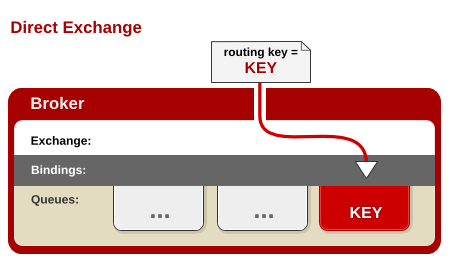
测试结果：

同一个消息被多个消费者获取。一个消费者队列可以有多个消费者实例，只有其中一个消费者实例会消费到消息。

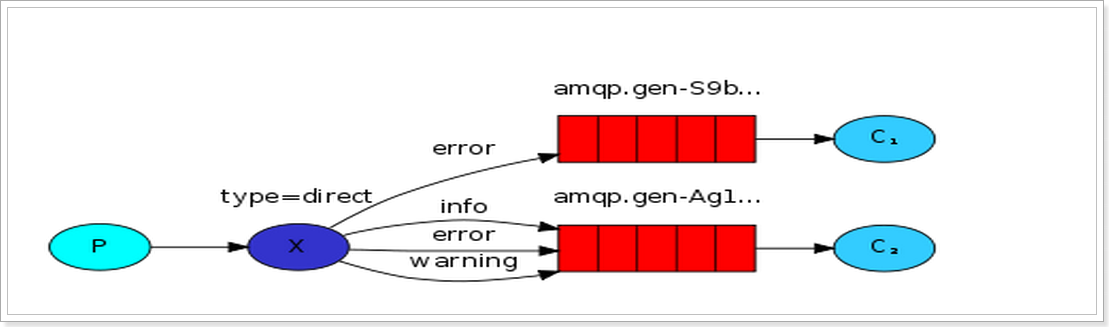
在管理工具中查看队列和交换机的绑定关系：



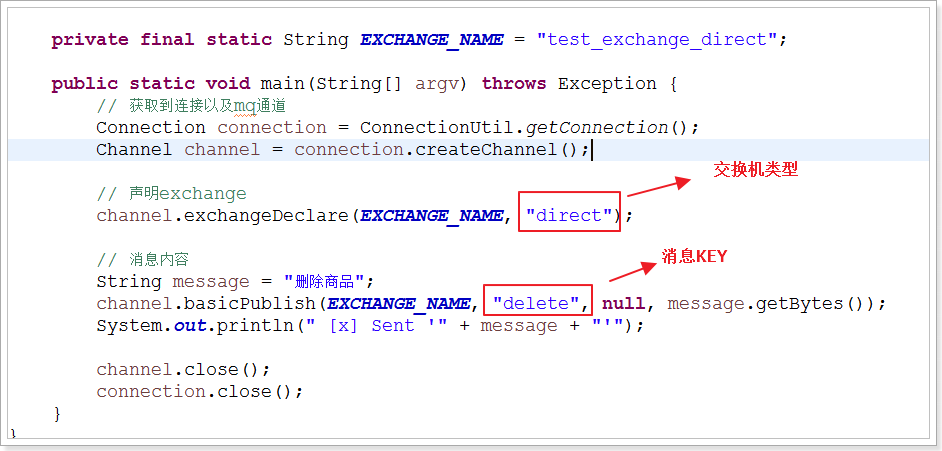
## 路由模式



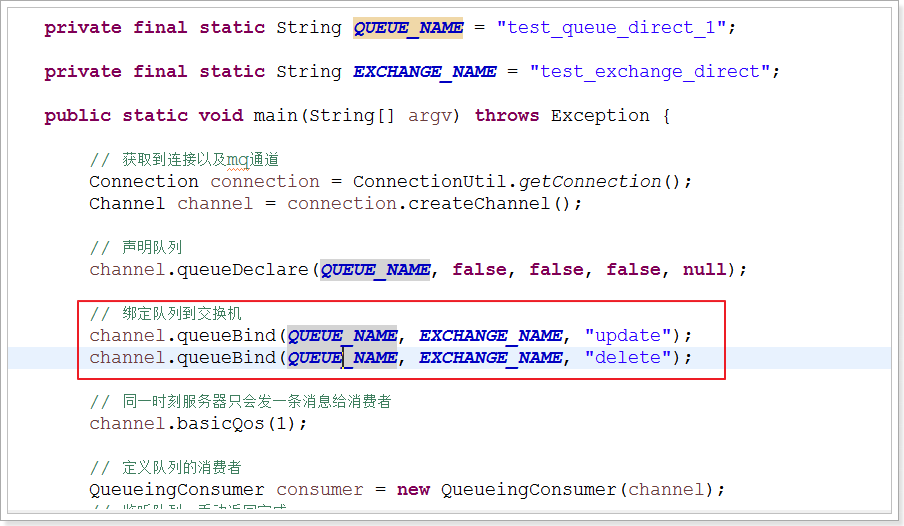
### 图示



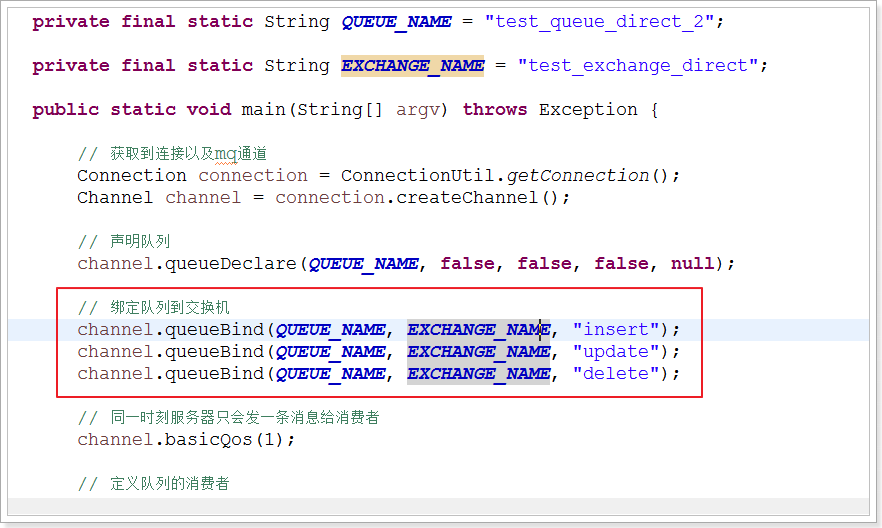
### 生产者



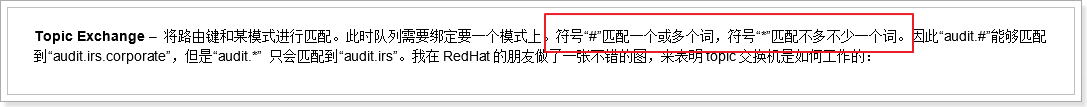
### 消费者1(前台系统)

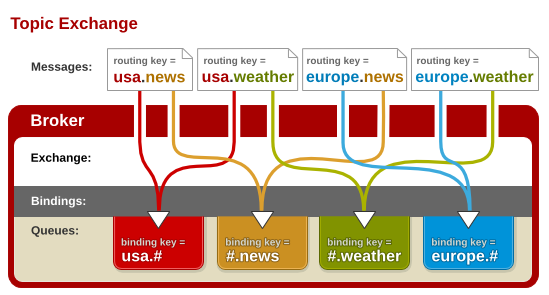


### 消费2（搜索系统）

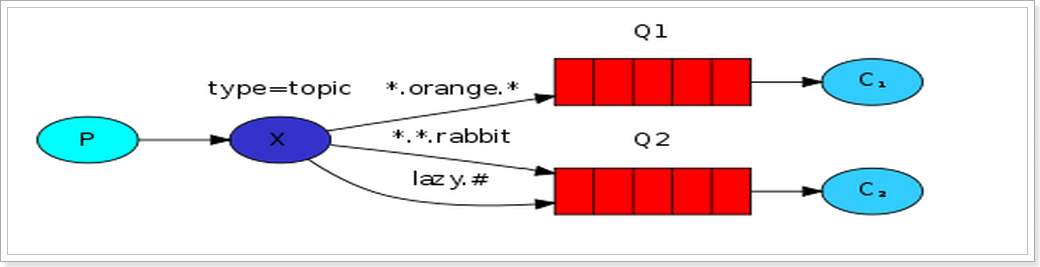


## 主题模式（通配符模式）





### 图示



同一个消息被多个消费者获取。一个消费者队列可以有多个消费者实例，只有其中一个消费者实例会消费到消息。

### 生产者

package com.zpc.rabbitmq.topic;  
  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.Connection;  
  
import com.zpc.rabbitmq.util.ConnectionUtil;  
  
public class Send {  
  
 private final static String ***EXCHANGE\_NAME*** = "test\_exchange\_topic";  
  
 public static void main(String[] argv) throws Exception {  
 // 获取到连接以及mq通道  
 Connection connection = ConnectionUtil.getConnection();  
 Channel channel = connection.createChannel();  
  
 // 声明exchange  
 channel.exchangeDeclare(***EXCHANGE\_NAME***, "topic");  
  
 // 消息内容  
 String message = "Hello World!!";  
 channel.basicPublish(***EXCHANGE\_NAME***, "routekey.1", null, message.getBytes());  
 System.***out***.println(" [x] Sent '" + message + "'");  
  
 channel.close();  
 connection.close();  
 }  
}

### 消费者1（前台系统）

package com.zpc.rabbitmq.topic;  
  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.Connection;  
import com.rabbitmq.client.QueueingConsumer;  
  
import com.zpc.rabbitmq.util.ConnectionUtil;  
  
public class Recv {  
  
 private final static String ***QUEUE\_NAME*** = "test\_queue\_topic\_work\_1";  
  
 private final static String ***EXCHANGE\_NAME*** = "test\_exchange\_topic";  
  
 public static void main(String[] argv) throws Exception {  
  
 // 获取到连接以及mq通道  
 Connection connection = ConnectionUtil.getConnection();  
 Channel channel = connection.createChannel();  
  
 // 声明队列  
 channel.queueDeclare(***QUEUE\_NAME***, false, false, false, null);  
  
 // 绑定队列到交换机  
 channel.queueBind(***QUEUE\_NAME***, ***EXCHANGE\_NAME***, "routekey.\*");  
  
 // 同一时刻服务器只会发一条消息给消费者  
 channel.basicQos(1);  
  
 // 定义队列的消费者  
 QueueingConsumer consumer = new QueueingConsumer(channel);  
 // 监听队列，手动返回完成  
 channel.basicConsume(***QUEUE\_NAME***, false, consumer);  
  
 // 获取消息  
 while (true) {  
 QueueingConsumer.Delivery delivery = consumer.nextDelivery();  
 String message = new String(delivery.getBody());  
 System.***out***.println(" [Recv\_x] Received '" + message + "'");  
 Thread.sleep(10);  
  
 channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);  
 }  
 }  
}

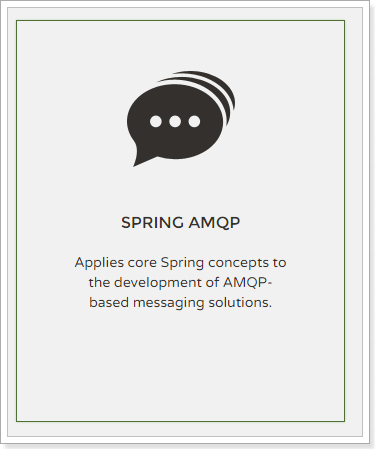
### 消费者2（搜索系统）

package com.zpc.rabbitmq.topic;  
  
import com.zpc.rabbitmq.util.ConnectionUtil;  
  
import com.rabbitmq.client.Channel;  
import com.rabbitmq.client.Connection;  
import com.rabbitmq.client.QueueingConsumer;  
  
public class Recv2 {  
  
 private final static String ***QUEUE\_NAME*** = "test\_queue\_topic\_work\_2";  
  
 private final static String ***EXCHANGE\_NAME*** = "test\_exchange\_topic";  
  
 public static void main(String[] argv) throws Exception {  
  
 // 获取到连接以及mq通道  
 Connection connection = ConnectionUtil.getConnection();  
 Channel channel = connection.createChannel();  
  
 // 声明队列  
 channel.queueDeclare(***QUEUE\_NAME***, false, false, false, null);  
  
 // 绑定队列到交换机  
 channel.queueBind(***QUEUE\_NAME***, ***EXCHANGE\_NAME***, "\*.\*");  
  
 // 同一时刻服务器只会发一条消息给消费者  
 channel.basicQos(1);  
  
 // 定义队列的消费者  
 QueueingConsumer consumer = new QueueingConsumer(channel);  
 // 监听队列，手动返回完成  
 channel.basicConsume(***QUEUE\_NAME***, false, consumer);  
  
 // 获取消息  
 while (true) {  
 QueueingConsumer.Delivery delivery = consumer.nextDelivery();  
 String message = new String(delivery.getBody());  
 System.***out***.println(" [Recv2\_x] Received '" + message + "'");  
 Thread.sleep(10);  
  
 channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);  
 }  
 }  
}

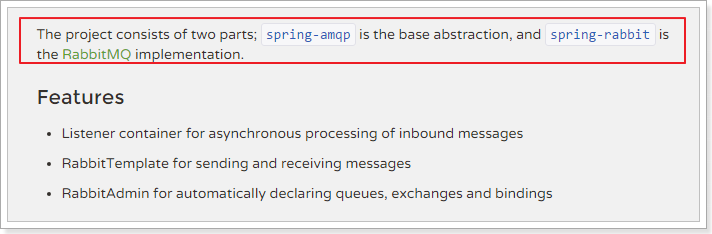
# Spring-Rabbit

## Spring项目

<http://spring.io/projects>



## 简介





## 使用

### 消费者

package com.zpc.rabbitmq.spring;  
  
/\*\*  
 \* 消费者  
 \*  
 \* @author Evan  
 \*/  
public class Foo {  
  
 //具体执行业务的方法  
 public void listen(String foo) {  
 System.***out***.println("\n消费者： " + foo + "\n");  
 }  
}

### 生产者

package com.zpc.rabbitmq.spring;  
  
import org.springframework.amqp.rabbit.core.RabbitTemplate;  
import org.springframework.context.support.AbstractApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class SpringMain {  
 public static void main(final String... args) throws Exception {  
 AbstractApplicationContext ctx = new ClassPathXmlApplicationContext(  
 "classpath:spring/rabbitmq-context.xml");  
 //RabbitMQ模板  
 RabbitTemplate template = ctx.getBean(RabbitTemplate.class);  
 //发送消息  
 template.convertAndSend("Hello, 鸟鹏!");  
 Thread.sleep(1000);// 休眠1秒  
 ctx.destroy(); //容器销毁  
 }  
}

### 配置文件

1. 定义连接工厂

<!-- 定义RabbitMQ的连接工厂 -->  
<rabbit:connection-factory id="connectionFactory"  
 host="127.0.0.1" port="5672" username="admin" password="admin"  
 virtual-host="testhost" />

1. 定义模板（可以指定交换机或队列）

<rabbit:template id="amqpTemplate" connection-factory="connectionFactory" exchange="fanoutExchange" />

1. 定义队列、交换机、以及完成队列和交换机的绑定

<!-- 定义队列，自动声明 -->  
<rabbit:queue name="zpcQueue" auto-declare="true"/>  
  
<!-- 定义交换器，把Q绑定到交换机，自动声明 -->  
<rabbit:fanout-exchange name="fanoutExchange" auto-declare="true">  
 <rabbit:bindings>  
 <rabbit:binding queue="zpcQueue"/>  
 </rabbit:bindings>  
</rabbit:fanout-exchange>

1. 定义监听

<rabbit:listener-container connection-factory="connectionFactory">  
 <rabbit:listener ref="foo" method="listen" queue-names="zpcQueue" />  
</rabbit:listener-container>  
  
<bean id="foo" class="com.zpc.rabbitmq.spring.Foo" />

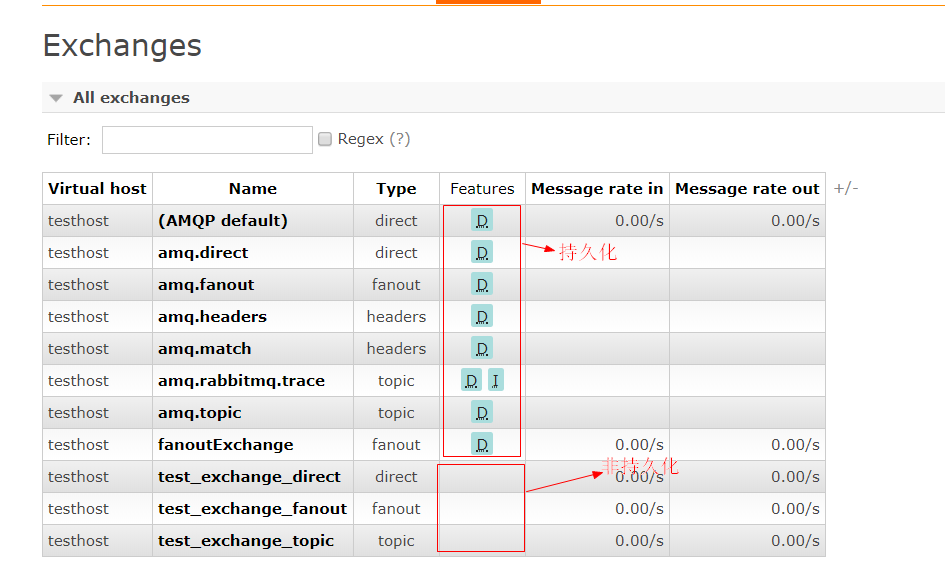
1. 定义管理，用于管理队列、交换机等：

<!-- MQ的管理，包括队列、交换器等 -->  
<rabbit:admin connection-factory="connectionFactory" />

完整配置文件rabbitmq-context.xml

<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:rabbit="http://www.springframework.org/schema/rabbit"  
 xsi:schemaLocation="http://www.springframework.org/schema/rabbit  
 http://www.springframework.org/schema/rabbit/spring-rabbit-1.4.xsd  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-4.1.xsd">  
  
 <!-- 定义RabbitMQ的连接工厂 -->  
 <rabbit:connection-factory id="connectionFactory"  
 host="127.0.0.1" port="5672" username="admin" password="admin"  
 virtual-host="testhost" />  
  
 <!-- 定义Rabbit模板，指定连接工厂以及定义exchange -->  
 <rabbit:template id="amqpTemplate" connection-factory="connectionFactory" exchange="fanoutExchange" />  
 <!-- <rabbit:template id="amqpTemplate" connection-factory="connectionFactory"  
 exchange="fanoutExchange" routing-key="foo.bar" /> -->  
  
 <!-- MQ的管理，包括队列、交换器等 -->  
 <rabbit:admin connection-factory="connectionFactory" />  
  
 <!-- 定义队列，自动声明 -->  
 <rabbit:queue name="zpcQueue" auto-declare="true"/>  
   
 <!-- 定义交换器，把Q绑定到交换机，自动声明 -->  
 <rabbit:fanout-exchange name="fanoutExchange" auto-declare="true">  
 <rabbit:bindings>  
 <rabbit:binding queue="zpcQueue"/>  
 </rabbit:bindings>  
 </rabbit:fanout-exchange>  
   
<!-- <rabbit:topic-exchange name="myExchange">  
 <rabbit:bindings>  
 <rabbit:binding queue="myQueue" pattern="foo.\*" />  
 </rabbit:bindings>  
 </rabbit:topic-exchange> -->  
  
 <!-- 队列监听 -->  
 <rabbit:listener-container connection-factory="connectionFactory">  
 <rabbit:listener ref="foo" method="listen" queue-names="zpcQueue" />  
 </rabbit:listener-container>  
  
 <bean id="foo" class="com.zpc.rabbitmq.spring.Foo" />  
  
</beans>

## 持久化交换机和队列



持久化：将交换机或队列的数据保存到磁盘，服务器宕机或重启之后依然存在。

非持久化：将交换机或队列的数据保存到内存，服务器宕机或重启之后将不存在。

非持久化的性能高于持久化。

如何选择持久化？非持久化？ -- 看需求。

# Spring集成RabbitMQ一个完整的例子

创建三个系统A,B,C

A作为生产者，B、C作为消费者

项目下载地址：

<https://download.csdn.net/download/zpcandzhj/10585077>

## 在A系统中发送消息到交换机

### 导入依赖

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.zpc</groupId>  
 <artifactId>myrabbitA</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
 <name>myrabbit</name>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.amqp</groupId>  
 <artifactId>spring-rabbit</artifactId>  
 <version>1.4.0.RELEASE</version>  
 </dependency>  
  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 <version>1.2.47</version>  
 </dependency>  
 </dependencies>  
</project>

### 队列和交换机的绑定关系

实现：

1. 在配置文件中将队列和交换机完成绑定
2. 可以在管理界面中完成绑定
   1. 绑定关系如果发生变化，需要修改配置文件，并且服务需要重启
   2. 管理更加灵活
   3. 更容易对绑定关系的权限管理，流程管理

本例选择第2种方式

### 配置

rabbitmq-context.xml

<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:rabbit="http://www.springframework.org/schema/rabbit"  
 xsi:schemaLocation="http://www.springframework.org/schema/rabbit  
 http://www.springframework.org/schema/rabbit/spring-rabbit-1.4.xsd  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-4.1.xsd">  
  
 <!-- 定义RabbitMQ的连接工厂 -->  
 <rabbit:connection-factory id="connectionFactory"  
 host="127.0.0.1" port="5672" username="admin" password="admin"  
 virtual-host="testhost" />  
  
 <!-- MQ的管理，包括队列、交换器等 -->  
 <rabbit:admin connection-factory="connectionFactory" />  
  
 <!-- 定义交换器，暂时不把Q绑定到交换机，在管理界面去绑定 -->  
 <!--<rabbit:topic-exchange name="topicExchange" auto-declare="true" ></rabbit:topic-exchange>-->  
 <rabbit:direct-exchange name="directExchange" auto-declare="true" ></rabbit:direct-exchange>  
 <!--<rabbit:fanout-exchange name="fanoutExchange" auto-declare="true" ></rabbit:fanout-exchange>-->  
  
 <!-- 定义Rabbit模板，指定连接工厂以及定义exchange(exchange要和上面的一致) -->  
 <!--<rabbit:template id="amqpTemplate" connection-factory="connectionFactory" exchange="topicExchange" />-->  
 <rabbit:template id="amqpTemplate" connection-factory="connectionFactory" exchange="directExchange" />  
 <!--<rabbit:template id="amqpTemplate" connection-factory="connectionFactory" exchange="fanoutExchange" />-->  
</beans>

### 消息内容

方案：

1. 消息内容使用对象做json序列化发送
   1. 数据大
   2. 有些数据其他人是可能用不到的
2. 发送特定的业务字段，如id、操作类型

### 实现

生产者MsgSender.java：

package com.zpc.myrabbit;  
  
import com.alibaba.fastjson.JSON;  
import org.springframework.amqp.rabbit.core.RabbitTemplate;  
import org.springframework.context.support.AbstractApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.HashMap;  
import java.util.Map;  
  
/\*\*  
 \* 消息生产者  
 \*/  
public class MsgSender {  
 public static void main(String[] args) throws Exception {  
 AbstractApplicationContext ctx = new ClassPathXmlApplicationContext(  
 "classpath:spring/rabbitmq-context.xml");  
 //RabbitMQ模板  
 RabbitTemplate template = ctx.getBean(RabbitTemplate.class);  
  
 String date = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());//24小时制  
 //发送消息  
 Map<String, Object> msg = new HashMap<String, Object>();  
 msg.put("type", "1");  
 msg.put("date", date);  
 template.convertAndSend("type2", JSON.toJSONString(msg));  
 Thread.sleep(1000);// 休眠1秒  
 ctx.destroy(); //容器销毁  
 }  
}

## 在B系统接收消息

### 导入依赖

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.zpc</groupId>  
 <artifactId>myrabbitB</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>war</packaging>  
  
 <name>myrabbit</name>  
 <properties>  
 <spring.version>4.1.3.RELEASE</spring.version>  
 <fastjson.version>1.2.46</fastjson.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>com.rabbitmq</groupId>  
 <artifactId>amqp-client</artifactId>  
 <version>3.4.1</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.amqp</groupId>  
 <artifactId>spring-rabbit</artifactId>  
 <version>1.4.0.RELEASE</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>fastjson</artifactId>  
 <version>1.2.47</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <finalName>${project.artifactId}</finalName>  
 <plugins>  
 <!-- web层需要配置Tomcat插件 -->  
 <plugin>  
 <groupId>org.apache.tomcat.maven</groupId>  
 <artifactId>tomcat7-maven-plugin</artifactId>  
 <configuration>  
 <path>/testRabbit</path>  
 <uriEncoding>UTF-8</uriEncoding>  
 <port>8081</port>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

### 配置

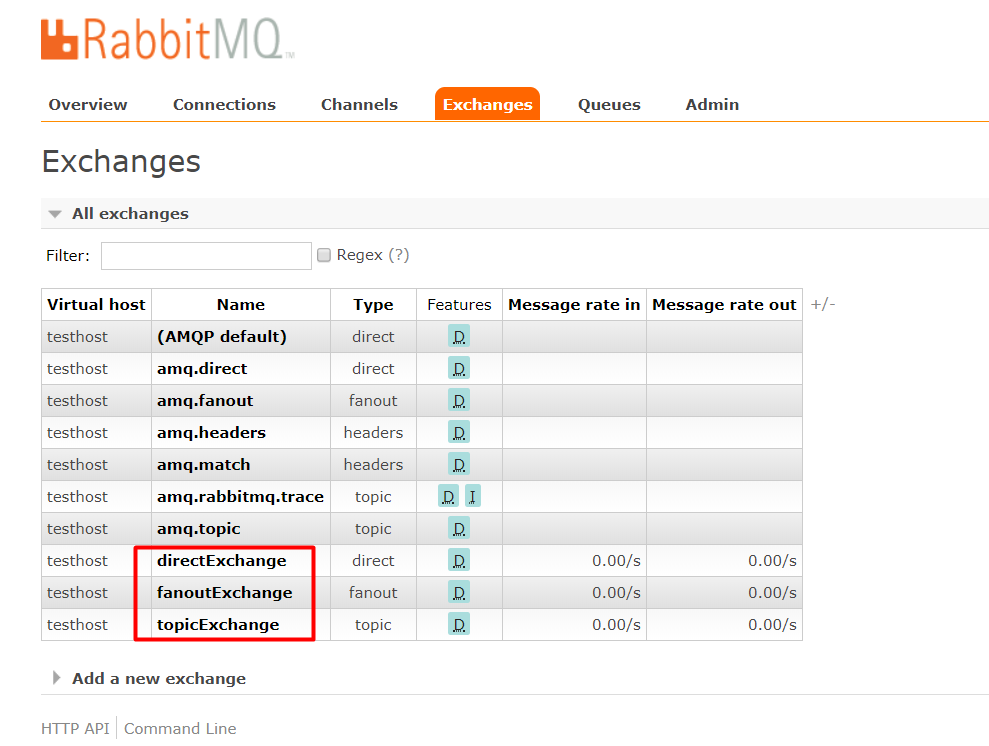
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:rabbit="http://www.springframework.org/schema/rabbit"  
 xsi:schemaLocation="http://www.springframework.org/schema/rabbit  
 http://www.springframework.org/schema/rabbit/spring-rabbit-1.4.xsd  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-4.1.xsd">  
  
 <!-- 定义RabbitMQ的连接工厂 -->  
 <rabbit:connection-factory id="connectionFactory"  
 host="127.0.0.1" port="5672" username="admin" password="admin"  
 virtual-host="testhost" />  
  
 <!-- MQ的管理，包括队列、交换器等 -->  
 <rabbit:admin connection-factory="connectionFactory" />  
  
 <!-- 定义B系统需要监听的队列，自动声明 -->  
 <rabbit:queue name="q\_topic\_testB" auto-declare="true"/>  
  
 <!-- 队列监听 -->  
 <rabbit:listener-container connection-factory="connectionFactory">  
 <rabbit:listener ref="myMQlistener" method="listen" queue-names="q\_topic\_testB" />  
 </rabbit:listener-container>  
  
 <bean id="myMQlistener" class="com.zpc.myrabbit.listener.Listener" />  
</beans>

### 具体处理逻辑

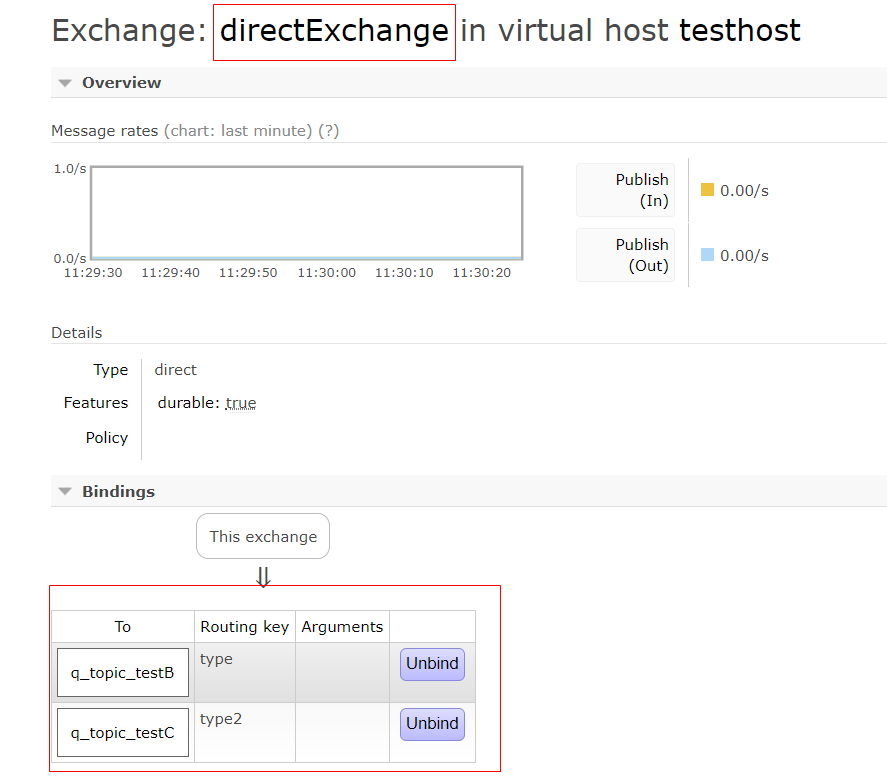
public class Listener {  
 //具体执行业务的方法  
 public void listen(String msg) {  
 System.***out***.println("\n消费者B开始处理消息： " + msg + "\n");  
 }  
}

### 在界面管理工具中完成绑定关系

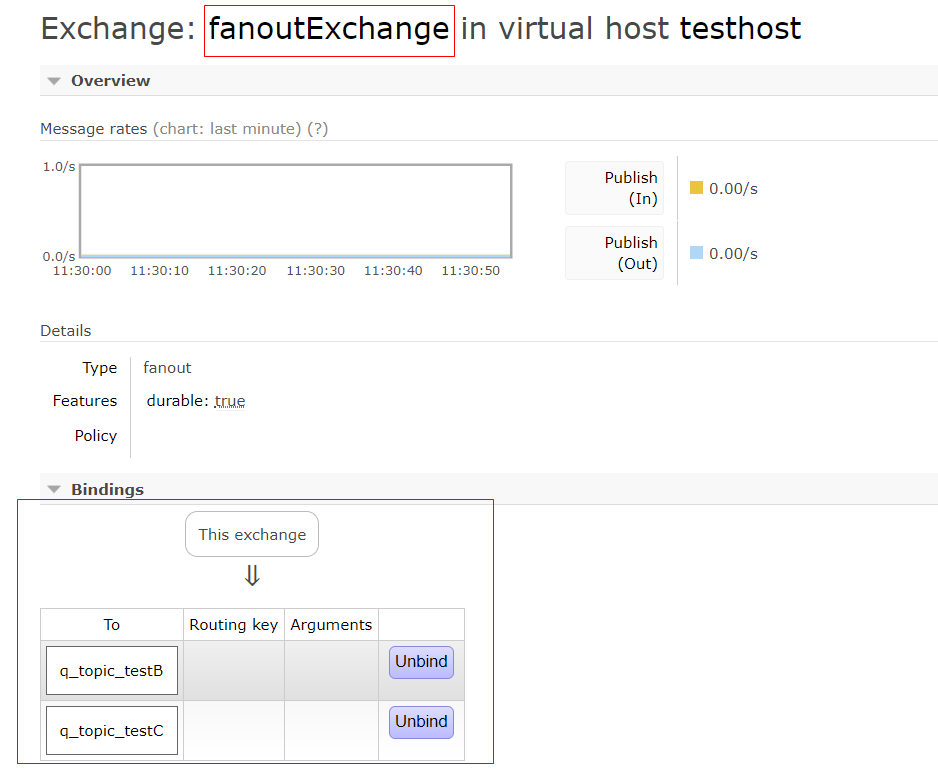
选中定义好的交换机(exchange)



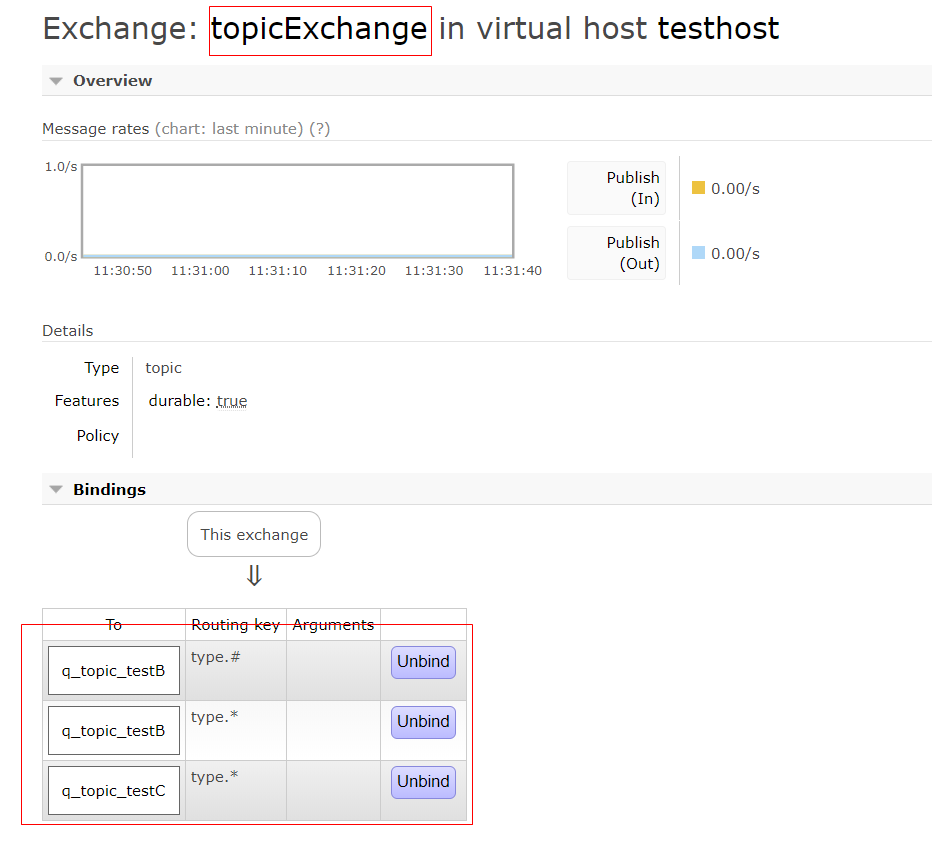
1）direct



2）fanout



3）topic



## 在C系统中接收消息

（和B系统配置差不多，无非是Q名和Q对应的处理逻辑变了）

### 配置

<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:rabbit="http://www.springframework.org/schema/rabbit"  
 xsi:schemaLocation="http://www.springframework.org/schema/rabbit  
 http://www.springframework.org/schema/rabbit/spring-rabbit-1.4.xsd  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-4.1.xsd">  
  
 <!-- 定义RabbitMQ的连接工厂 -->  
 <rabbit:connection-factory id="connectionFactory"  
 host="127.0.0.1" port="5672" username="admin" password="admin"  
 virtual-host="testhost" />  
  
 <!-- MQ的管理，包括队列、交换器等 -->  
 <rabbit:admin connection-factory="connectionFactory" />  
  
 <!-- 定义C系统需要监听的队列，自动声明 -->  
 <rabbit:queue name="q\_topic\_testC" auto-declare="true"/>  
  
 <!-- 队列监听 -->  
 <rabbit:listener-container connection-factory="connectionFactory">  
 <rabbit:listener ref="myMQlistener" method="listen" queue-names="q\_topic\_testC" />  
 </rabbit:listener-container>  
  
 <bean id="myMQlistener" class="com.zpc.myrabbit.listener.Listener" />  
</beans>

### 处理业务逻辑

public class Listener {  
  
 //具体执行业务的方法  
 public void listen(String msg) {  
 System.***out***.println("\n消费者C开始处理消息： " + msg + "\n");  
 }  
}

### 在管理工具中绑定队列和交换机

见7.2.4

### 测试

分别启动B,C两个web应用，然后运行A的MsgSender主方法发送消息，分别测试fanout、direct、topic三种类型

# Springboot集成RabbitMQ

springboot集成RabbitMQ非常简单，如果只是简单的使用配置非常少，springboot提供了spring-boot-starter-amqp对消息各种支持。

代码下载地址：<https://download.csdn.net/download/zpcandzhj/10585077>

## 简单队列

1、配置pom文件，主要是添加spring-boot-starter-amqp的支持

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-amqp</artifactId>  
</dependency>

2、配置application.properties文件

配置rabbitmq的安装地址、端口以及账户信息

spring.application.name=spirng-boot-rabbitmq  
spring.rabbitmq.host=127.0.0.1  
spring.rabbitmq.port=5672  
spring.rabbitmq.username=admin  
spring.rabbitmq.password=admin

3、配置队列

package com.zpc.rabbitmq;  
  
import org.springframework.amqp.core.Queue;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class RabbitConfig {  
 @Bean  
 public Queue queue() {  
 return new Queue("q\_hello");  
 }  
}

4、发送者

package com.zpc.rabbitmq;  
  
import org.springframework.amqp.core.AmqpTemplate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
@Component  
public class HelloSender {  
 @Autowired  
 private AmqpTemplate rabbitTemplate;  
  
 public void send() {  
 String date = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());//24小时制  
 String context = "hello " + date;  
 System.***out***.println("Sender : " + context);  
 //简单对列的情况下routingKey即为Q名  
 this.rabbitTemplate.convertAndSend("q\_hello", context);  
 }  
}

5、接收者

package com.zpc.rabbitmq;  
  
import org.springframework.amqp.rabbit.annotation.RabbitHandler;  
import org.springframework.amqp.rabbit.annotation.RabbitListener;  
import org.springframework.stereotype.Component;  
  
@Component  
@RabbitListener(queues = "q\_hello")  
public class HelloReceiver {  
  
 @RabbitHandler  
 public void process(String hello) {  
 System.***out***.println("Receiver : " + hello);  
 }  
}

6、测试

package com.zpc.rabbitmq;  
  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.test.context.junit4.SpringRunner;  
  
@RunWith(SpringRunner.class)  
@SpringBootTest  
public class RabbitMqHelloTest {  
  
 @Autowired  
 private HelloSender helloSender;  
  
 @Test  
 public void hello() throws Exception {  
 helloSender.send();  
 }  
}

## 多对多使用（Work模式）

注册两个Receiver:

package com.zpc.rabbitmq;  
  
import org.springframework.amqp.rabbit.annotation.RabbitHandler;  
import org.springframework.amqp.rabbit.annotation.RabbitListener;  
import org.springframework.stereotype.Component;  
  
@Component  
@RabbitListener(queues = "q\_hello")  
public class HelloReceiver2 {  
  
 @RabbitHandler  
 public void process(String hello) {  
 System.***out***.println("Receiver2 : " + hello);  
 }  
  
}

@Test  
public void oneToMany() throws Exception {  
 for (int i=0;i<100;i++){  
 helloSender.send(i);  
 Thread.sleep(300);  
 }  
}

public void send(int i) {  
 String date = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new Date());//24小时制  
 String context = "hello " + i + " " + date;  
 System.***out***.println("Sender : " + context);  
 //简单对列的情况下routingKey即为Q名  
 this.rabbitTemplate.convertAndSend("q\_hello", context);  
}

## Topic Exchange（主题模式）

topic 是RabbitMQ中最灵活的一种方式，可以根据routing\_key自由的绑定不同的队列

首先对topic规则配置，这里使用两个队列(消费者)来演示。

1)配置队列，绑定交换机

package com.zpc.rabbitmq.topic;  
  
import org.springframework.amqp.core.Binding;  
import org.springframework.amqp.core.BindingBuilder;  
import org.springframework.amqp.core.Queue;  
import org.springframework.amqp.core.TopicExchange;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class TopicRabbitConfig {  
  
 final static String ***message*** = "q\_topic\_message";  
 final static String ***messages*** = "q\_topic\_messages";  
  
 @Bean  
 public Queue queueMessage() {  
 return new Queue(TopicRabbitConfig.***message***);  
 }  
  
 @Bean  
 public Queue queueMessages() {  
 return new Queue(TopicRabbitConfig.***messages***);  
 }  
  
 /\*\*  
 \* 声明一个Topic类型的交换机  
 \* @return  
 \*/  
 @Bean  
 TopicExchange exchange() {  
 return new TopicExchange("mybootexchange");  
 }  
  
 /\*\*  
 \* 绑定Q到交换机,并且指定routingKey  
 \* @param queueMessage  
 \* @param exchange  
 \* @return  
 \*/  
 @Bean  
 Binding bindingExchangeMessage(Queue queueMessage, TopicExchange exchange) {  
 return BindingBuilder.bind(queueMessage).to(exchange).with("topic.message");  
 }  
  
 @Bean  
 Binding bindingExchangeMessages(Queue queueMessages, TopicExchange exchange) {  
 return BindingBuilder.bind(queueMessages).to(exchange).with("topic.#");  
 }  
}

2)创建2个消费者

q\_topic\_message 和q\_topic\_messages

package com.zpc.rabbitmq.topic;  
  
import org.springframework.amqp.rabbit.annotation.RabbitHandler;  
import org.springframework.amqp.rabbit.annotation.RabbitListener;  
import org.springframework.stereotype.Component;  
  
@Component  
@RabbitListener(queues = "q\_topic\_message")  
public class Receiver1 {  
  
 @RabbitHandler  
 public void process(String hello) {  
 System.***out***.println("Receiver1 : " + hello);  
 }  
}

package com.zpc.rabbitmq.topic;  
  
import org.springframework.amqp.rabbit.annotation.RabbitHandler;  
import org.springframework.amqp.rabbit.annotation.RabbitListener;  
import org.springframework.stereotype.Component;  
  
@Component  
@RabbitListener(queues = "q\_topic\_messages")  
public class Receiver2 {  
  
 @RabbitHandler  
 public void process(String hello) {  
 System.***out***.println("Receiver2 : " + hello);  
 }  
}

3)消息发送者（生产者）

package com.zpc.rabbitmq.topic;  
  
import org.springframework.amqp.core.AmqpTemplate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
@Component  
public class MsgSender {  
  
 @Autowired  
 private AmqpTemplate rabbitTemplate;  
  
 public void send1() {  
 String context = "hi, i am message 1";  
 System.***out***.println("Sender : " + context);  
 this.rabbitTemplate.convertAndSend("mybootexchange", "topic.message", context);  
 }

public void send2() {  
 String context = "hi, i am messages 2";  
 System.***out***.println("Sender : " + context);  
 this.rabbitTemplate.convertAndSend("mybootexchange", "topic.messages", context);  
 }  
}

send1方法会匹配到topic.#和topic.message，两个Receiver都可以收到消息，发送send2只有topic.#可以匹配所有只有Receiver2监听到消息。

4)测试

package com.zpc.rabbitmq.topic;  
  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.test.context.junit4.SpringRunner;  
  
@RunWith(SpringRunner.class)  
@SpringBootTest  
public class RabbitTopicTest {  
  
 @Autowired  
 private MsgSender msgSender;  
  
 @Test  
 public void send1() throws Exception {  
 msgSender.send1();  
 }  
  
 @Test  
 public void send2() throws Exception {  
 msgSender.send2();  
 }  
}

## Fanout Exchange（订阅模式）

Fanout 就是我们熟悉的广播模式或者订阅模式，给Fanout交换机发送消息，绑定了这个交换机的所有队列都收到这个消息。

1)配置队列，绑定交换机

package com.zpc.rabbitmq.fanout;  
  
import org.springframework.amqp.core.Binding;  
import org.springframework.amqp.core.BindingBuilder;  
import org.springframework.amqp.core.FanoutExchange;  
import org.springframework.amqp.core.Queue;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class FanoutRabbitConfig {  
  
 @Bean  
 public Queue aMessage() {  
 return new Queue("q\_fanout\_A");  
 }  
  
 @Bean  
 public Queue bMessage() {  
 return new Queue("q\_fanout\_B");  
 }  
  
 @Bean  
 public Queue cMessage() {  
 return new Queue("q\_fanout\_C");  
 }  
  
 @Bean  
 FanoutExchange fanoutExchange() {  
 return new FanoutExchange("mybootfanoutExchange");  
 }  
  
 @Bean  
 Binding bindingExchangeA(Queue aMessage, FanoutExchange fanoutExchange) {  
 return BindingBuilder.bind(aMessage).to(fanoutExchange);  
 }  
  
 @Bean  
 Binding bindingExchangeB(Queue bMessage, FanoutExchange fanoutExchange) {  
 return BindingBuilder.bind(bMessage).to(fanoutExchange);  
 }  
  
 @Bean  
 Binding bindingExchangeC(Queue cMessage, FanoutExchange fanoutExchange) {  
 return BindingBuilder.bind(cMessage).to(fanoutExchange);  
 }  
}

1. 创建3个消费者

package com.zpc.rabbitmq.fanout;  
  
import org.springframework.amqp.rabbit.annotation.RabbitHandler;  
import org.springframework.amqp.rabbit.annotation.RabbitListener;  
import org.springframework.stereotype.Component;  
  
@Component  
@RabbitListener(queues = "q\_fanout\_A")  
public class ReceiverA {  
  
 @RabbitHandler  
 public void process(String hello) {  
 System.***out***.println("AReceiver : " + hello + "/n");  
 }  
}

package com.zpc.rabbitmq.fanout;  
  
import org.springframework.amqp.rabbit.annotation.RabbitHandler;  
import org.springframework.amqp.rabbit.annotation.RabbitListener;  
import org.springframework.stereotype.Component;  
  
@Component  
@RabbitListener(queues = "q\_fanout\_B")  
public class ReceiverB {  
  
 @RabbitHandler  
 public void process(String hello) {  
 System.***out***.println("BReceiver : " + hello + "/n");  
 }  
}

package com.zpc.rabbitmq.fanout;  
  
import org.springframework.amqp.rabbit.annotation.RabbitHandler;  
import org.springframework.amqp.rabbit.annotation.RabbitListener;  
import org.springframework.stereotype.Component;  
  
@Component  
@RabbitListener(queues = "q\_fanout\_C")  
public class ReceiverC {  
  
 @RabbitHandler  
 public void process(String hello) {  
 System.***out***.println("CReceiver : " + hello + "/n");  
 }  
}

1. 生产者

package com.zpc.rabbitmq.fanout;  
  
import org.springframework.amqp.core.AmqpTemplate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
@Component  
public class MsgSenderFanout {  
  
 @Autowired  
 private AmqpTemplate rabbitTemplate;  
  
 public void send() {  
 String context = "hi, fanout msg ";  
 System.***out***.println("Sender : " + context);  
 this.rabbitTemplate.convertAndSend("mybootfanoutExchange","", context);  
 }  
}

1. 测试

package com.zpc.rabbitmq.fanout;  
  
import org.junit.Test;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.test.context.junit4.SpringRunner;  
  
@RunWith(SpringRunner.class)  
@SpringBootTest  
public class RabbitFanoutTest {  
  
 @Autowired  
 private MsgSenderFanout msgSender;  
  
 @Test  
 public void send1() throws Exception {  
 msgSender.send();  
 }  
}

结果如下，三个消费者都收到消息：

AReceiver : hi, fanout msg

CReceiver : hi, fanout msg

BReceiver : hi, fanout msg

# 总结

使用MQ实现商品数据的同步优势：

1. 降低系统间耦合度
2. 便于管理数据的同步（数据一致性）

推荐阅读

《[RabbitMQ详解](http://www.ityouknow.com/springboot/2016/11/30/spring-boot-rabbitMQ.html)》

[《大型网站技术架构：核心原理与案例分析》](https://download.csdn.net/download/zpcandzhj/10584276)