# 今日大纲

1. 问题，如何实现商品数据的同步？
2. 学习MQ（消息队列）
   1. 搭建RabbitMQ的环境
   2. 学习RabbitMQ的队列
   3. 学习Spring-Rabbit
3. 使用RabbitMQ完成商品数据的同步

# 如何实现商品数据同步

之前的数据同步的实现：

1. 在前台系统开放接口
   1. 该接口中完成将Redis中的数据删除
2. 后台系统在商品编辑、删除时调用该接口，即可实现商品数据的同步

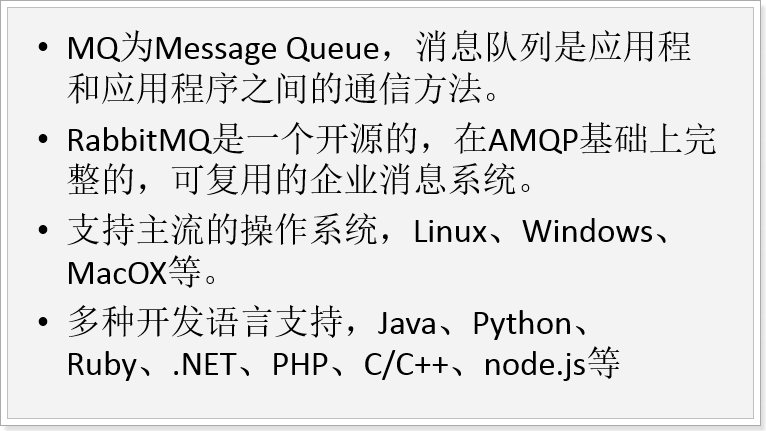
该方案存在的问题： 系统间的耦合度太高了。

耦合度是指：其他团队的版本升级会影响到后端团队的版本升级。

商品的数据已经实现了和前台系统的同步，但是，搜索系统中索引数据没有和后台系统的数据同步，导致，后台系统的商品数据进行了更新，在搜索系统中搜索到的数据是旧的数据，也可以采用之前的方案解决，但是，因为系统间的耦合度太高了，所以不推荐使用该方案，所以需要使用MQ来解决该问题。

# RabbitMQ

## RabbitMQ的简介



开发语言：Erlang – 面向并发的编程语言。

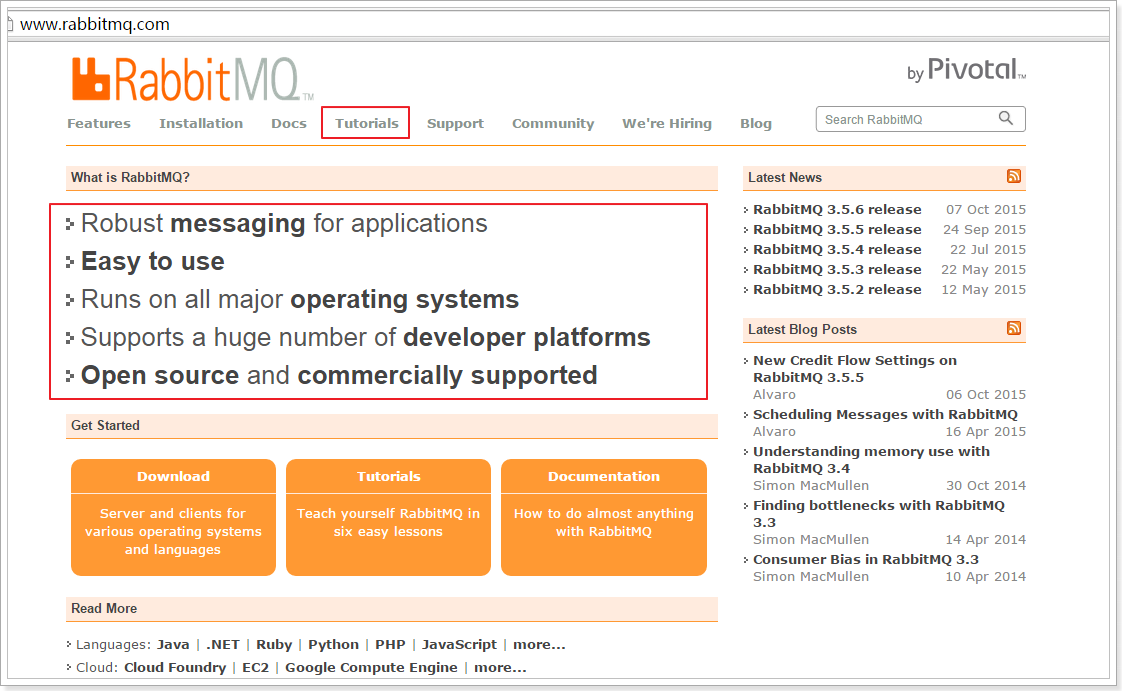


### AMQP

AMQP是消息队列的一个协议。



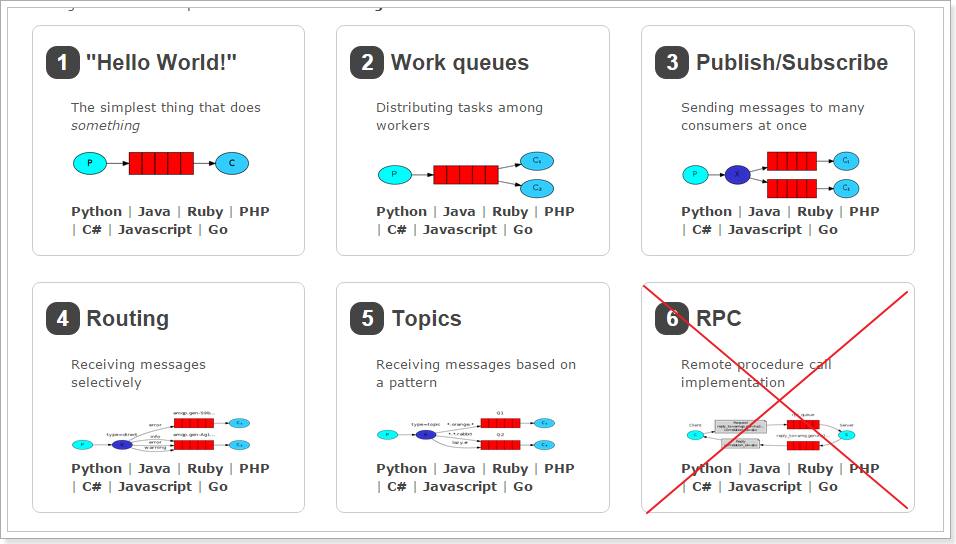
## 官网



## MQ的其他产品



## 学习5种队列



## 安装文档



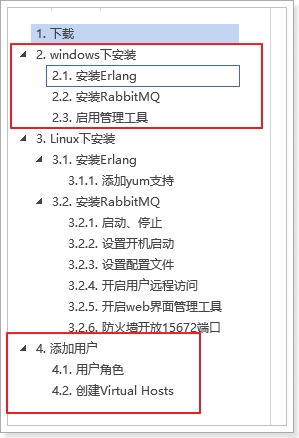
# 搭建RabbitMQ环境

具体参考《RabbitMQ-3.4.1安装手册.docx》

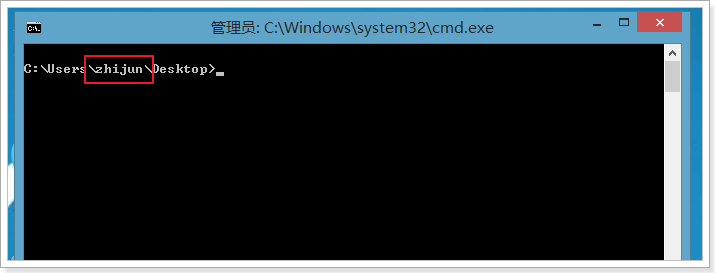
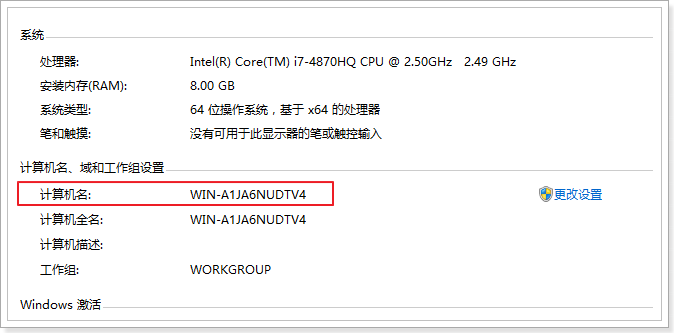
## 版本

推荐使用3.4.1版本。

## 安装



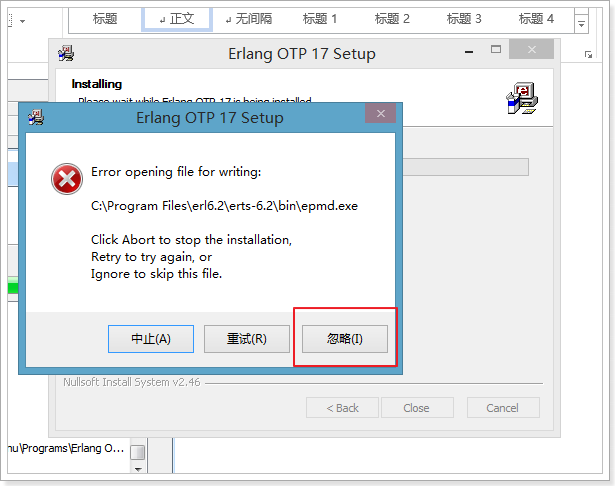
## 安装的注意事项

1. 推荐使用默认的安装路径
2. 系统用户名必须是英文
   1. Win8改名字非常麻烦，具体方法百度  
      
3. 计算机名必须是英文
   1. 
4. 系统的用户必须是管理员

如果安装失败的同学应该如何解决：

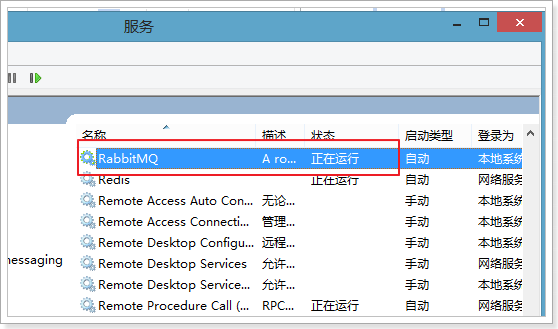
1. 重装系统 -- 不推荐
2. 将RabbitMQ安装到linux虚拟机中
   1. 推荐
3. 使用别人安装好的RabbitMQ服务
   1. 只要给你开通一个账户即可。
   2. 使用公用的RabbitMQ服务，在192.168.50.22
   3. 推荐

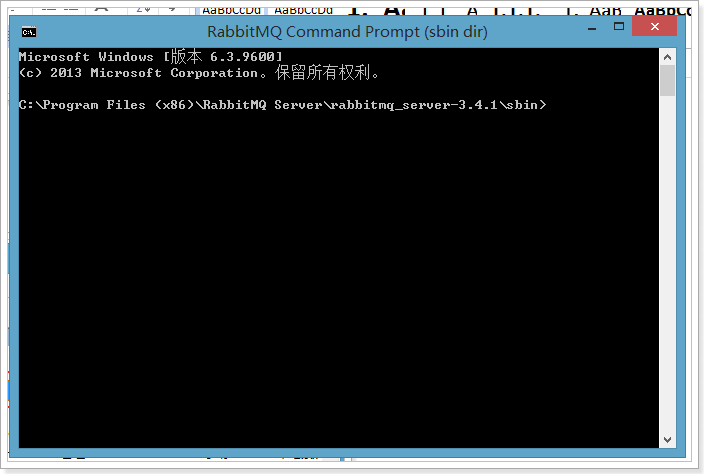
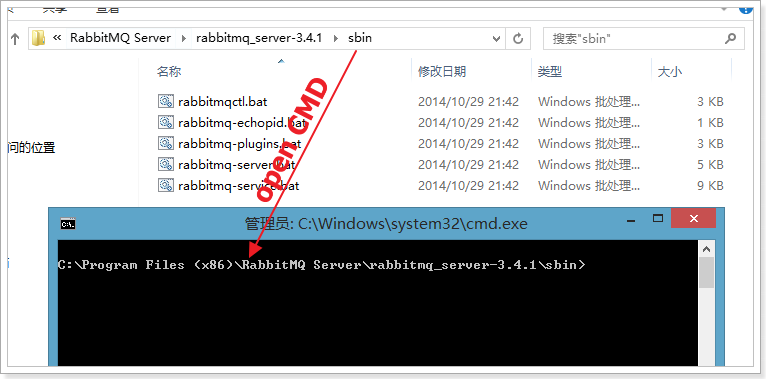
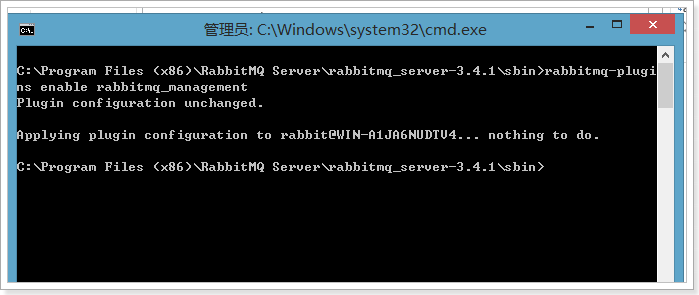
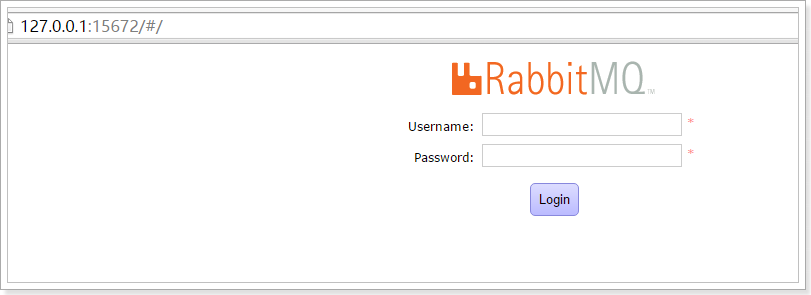
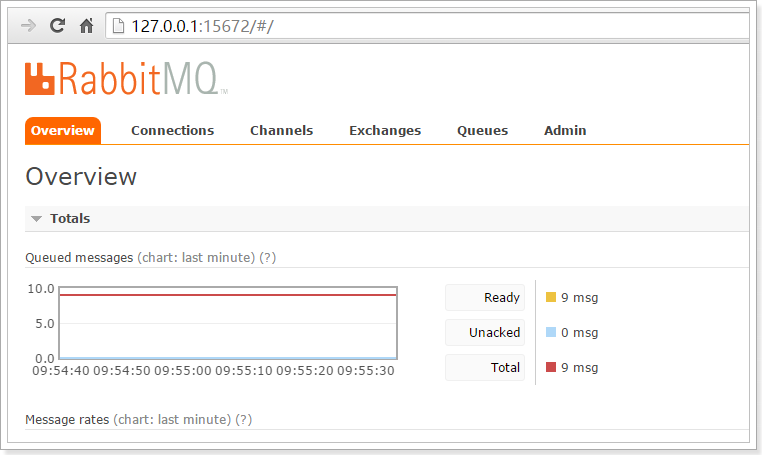
错误：



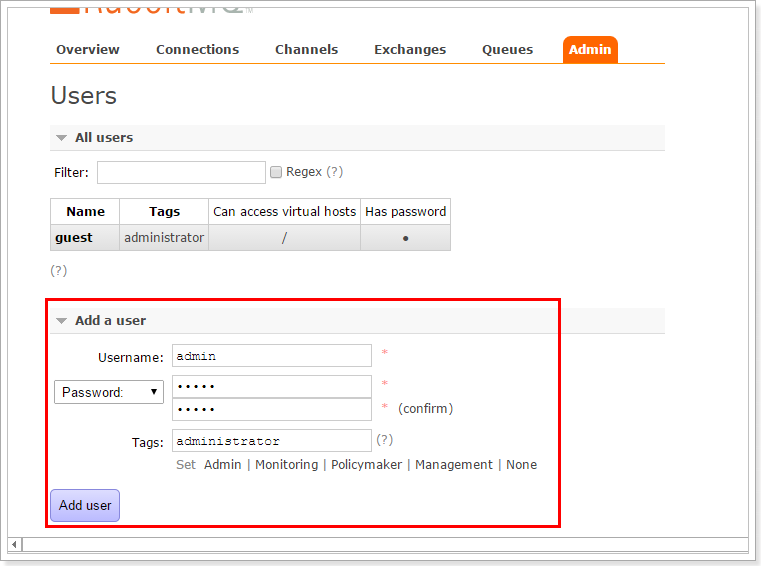
## 安装完成后操作

1、系统服务中有RabbitMQ服务，停止、启动、重启



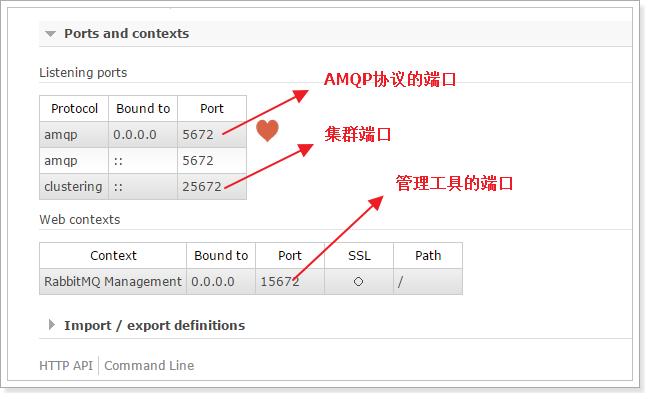
1. 打开命令行工具  
     
   如果找不到命令行工具：  
   
2. 启用管理插件  
   
3. 查看管理页面  
   
4. 通过默认账户 guest/guest 登录  
   如果能够登录，说明安装成功。  
   

## 添加taotao用户

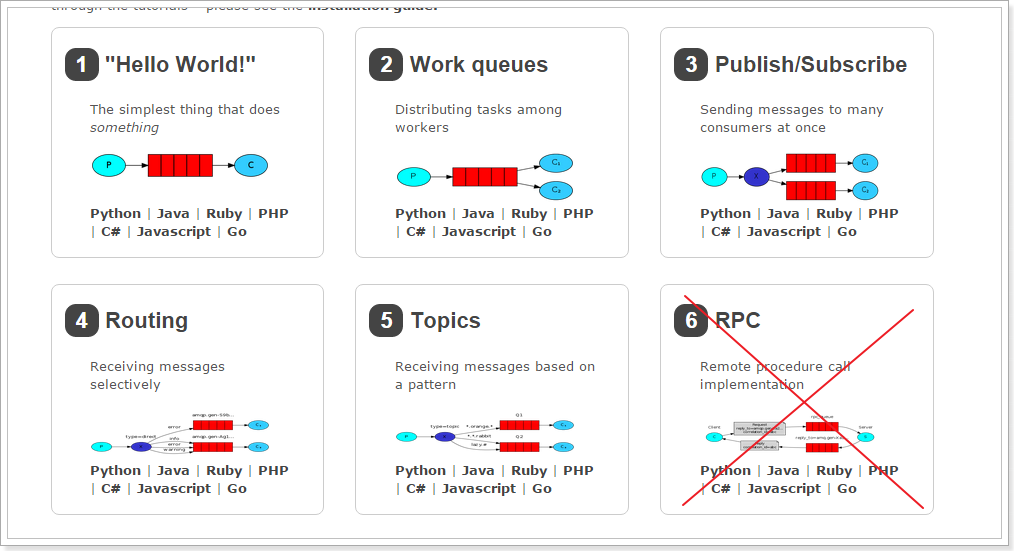


## 管理界面中的功能

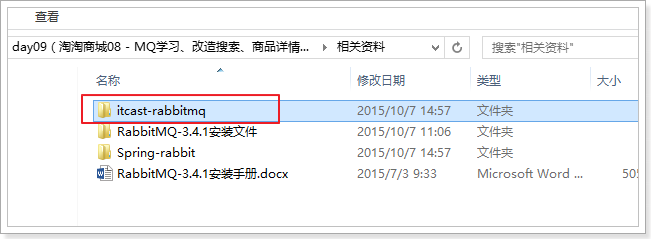




# 5种队列

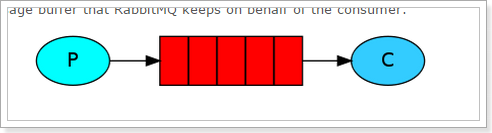


## 导入itcast-rabbitmq



## 简单队列

### 图示



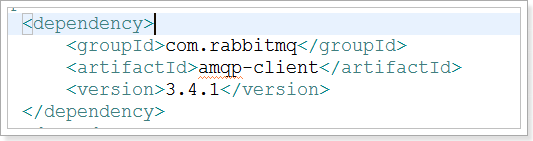
P：消息的生产者

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



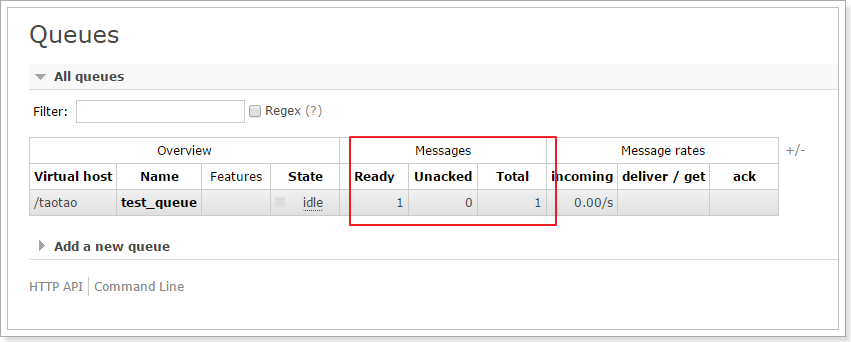
### 获取MQ的连接



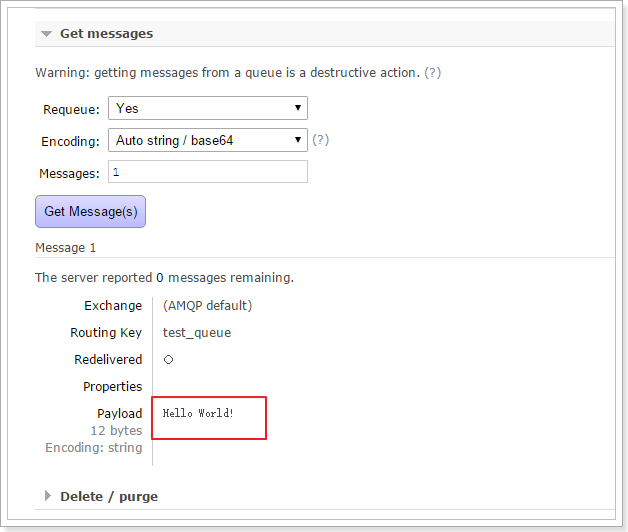
### 生产者发送消息到队列



### 管理工具中查看消息



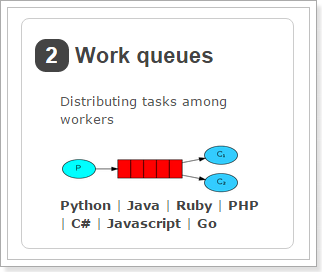
点击上面的队列名称，查询具体的队列中的信息：



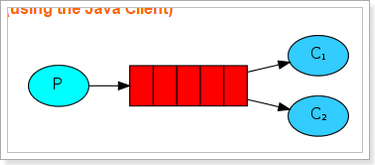
### 消费者从队列中获取消息



## 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

**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，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(" [x] Received '" + message + "'");

//休眠

Thread.*sleep*(10);

// 返回确认状态

channel.basicAck(delivery.getEnvelope().getDeliveryTag(), **false**);

}

}

}

### 消费者2

**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，手动返回完成状态

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(" [x] Received '" + message + "'");

// 休眠1秒

Thread.*sleep*(1000);

channel.basicAck(delivery.getEnvelope().getDeliveryTag(), **false**);

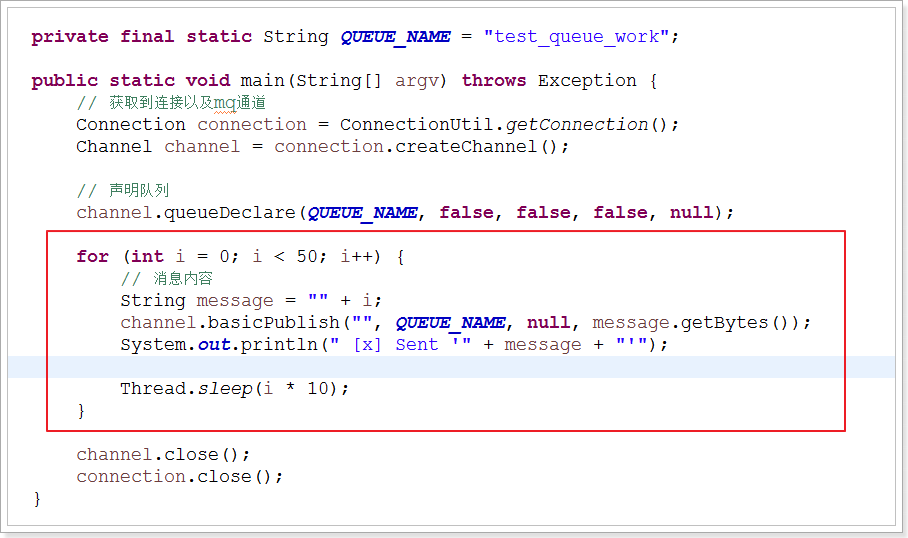
}

}

}

### 生产者

向队列中发送50条消息。



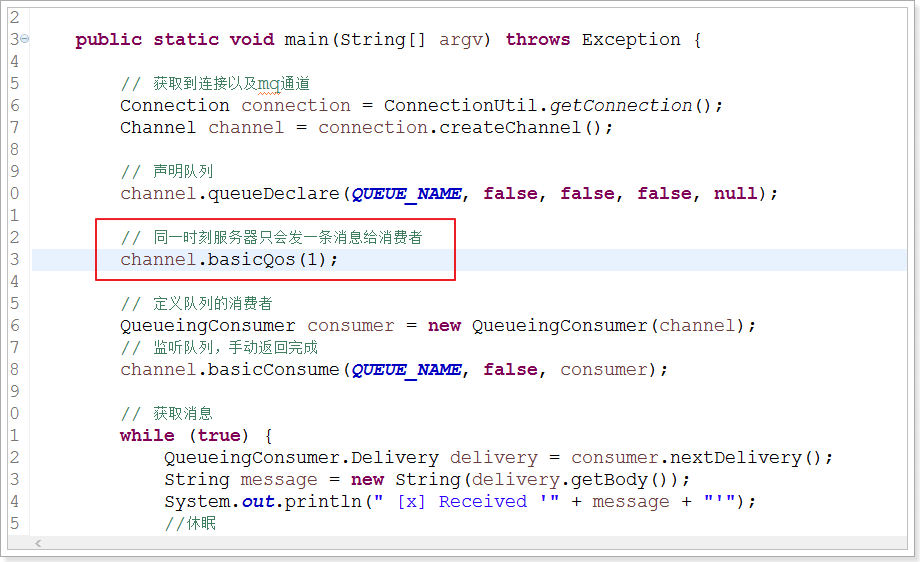
### 测试

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，一个是奇数一个是偶数。

其实，这样是不合理的，应该是消费者1要比消费者2获取到的消息多才对。

## Work模式的“能者多劳”



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？

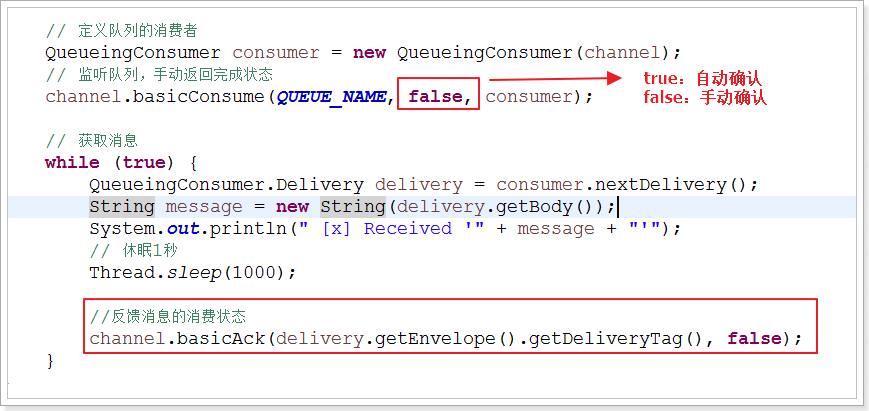
模式1：自动确认

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。

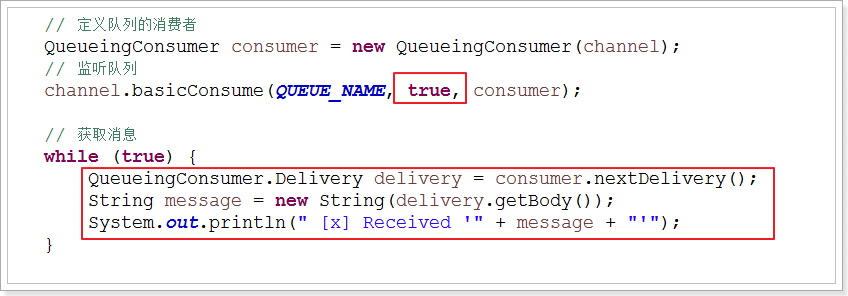
模式2：手动确认

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。

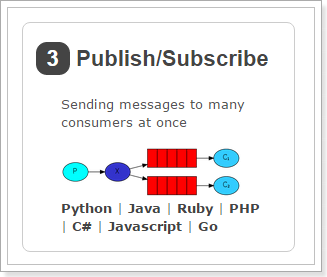
手动模式：



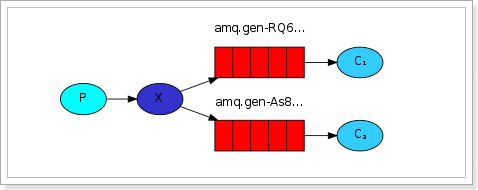
自动模式：



## 订阅模式



### 图示



解读：

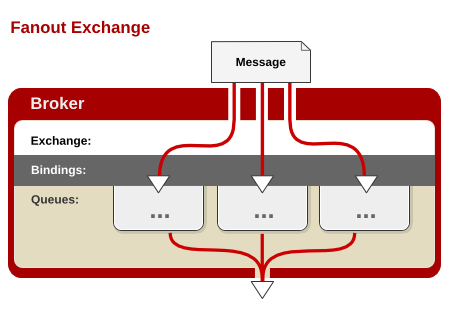
1、1个生产者，多个消费者

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



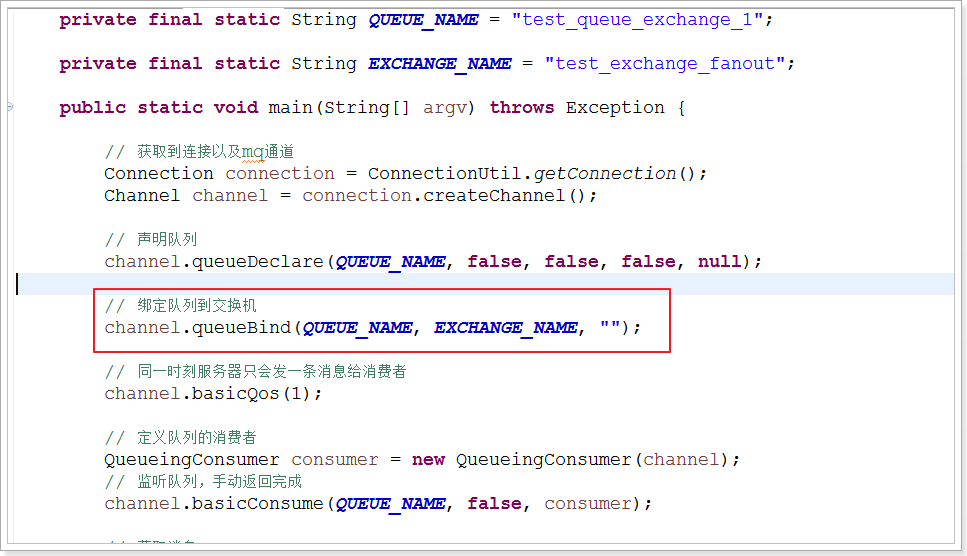
### 消息的生产者（看作是后台系统）

向交换机中发送消息。

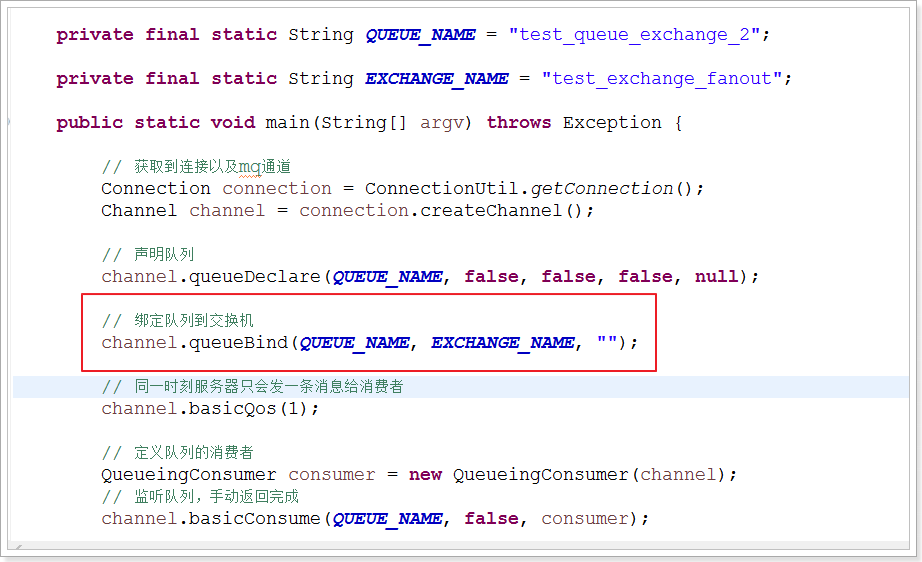


注意：消息发送到没有队列绑定的交换机时，消息将丢失，因为，交换机没有存储消息的能力，消息只能存在在队列中。

### 消费者1（看作是前台系统）



### 消费者2（看作是搜索系统）

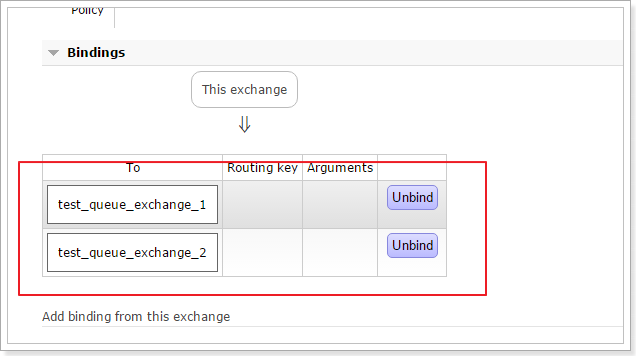


### 测试

测试结果：

同一个消息被多个消费者获取。

在管理工具中查看队列和交换机的绑定关系：



## 使用订阅模式能否实现商品数据的同步？

答案：可以的。

后台系统就是消息的生产者。

前台系统和搜索系统是消息的消费者。

后台系统将消息发送到交换机中，前台系统和搜索系统都创建自己的队列，然后将队列绑定到交换机，即可实现。

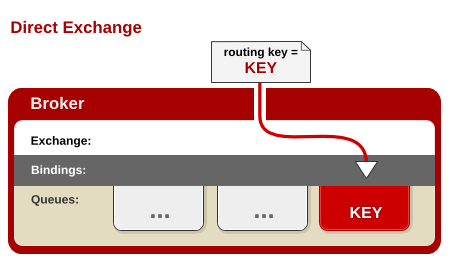
消息，新增商品、修改商品、删除商品。

前台系统：修改商品、删除商品。

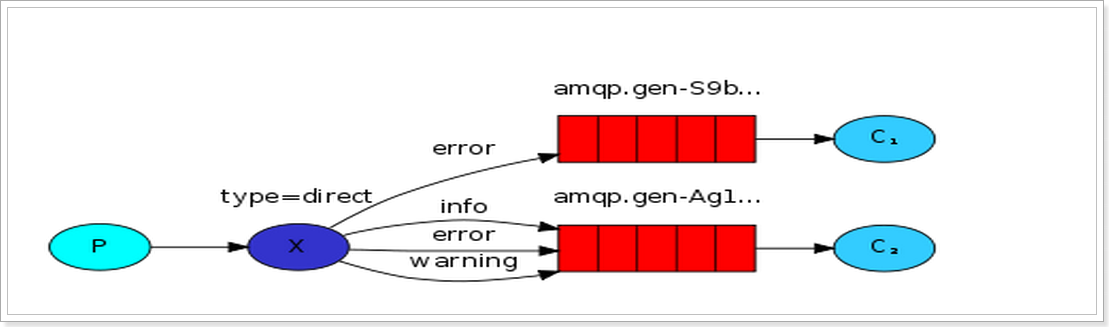
搜索系统：新增商品、修改商品、删除商品。

所以使用订阅模式实现商品数据的同步并不合理。

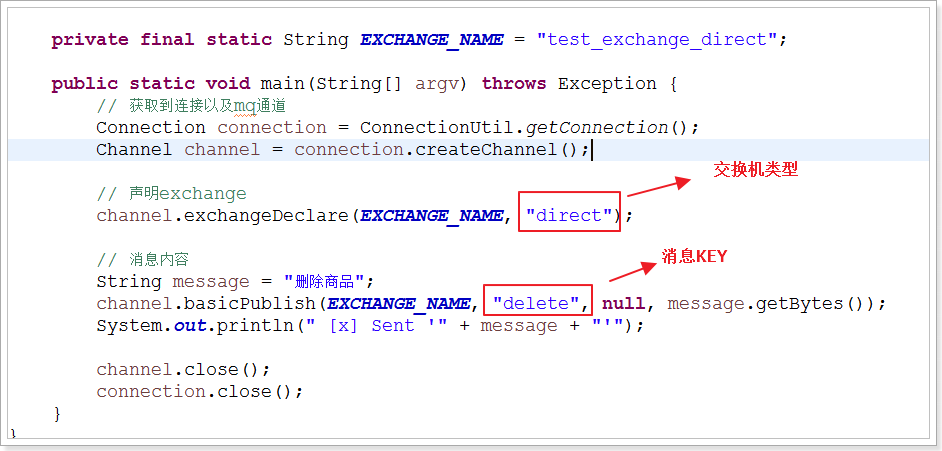
## 路由模式



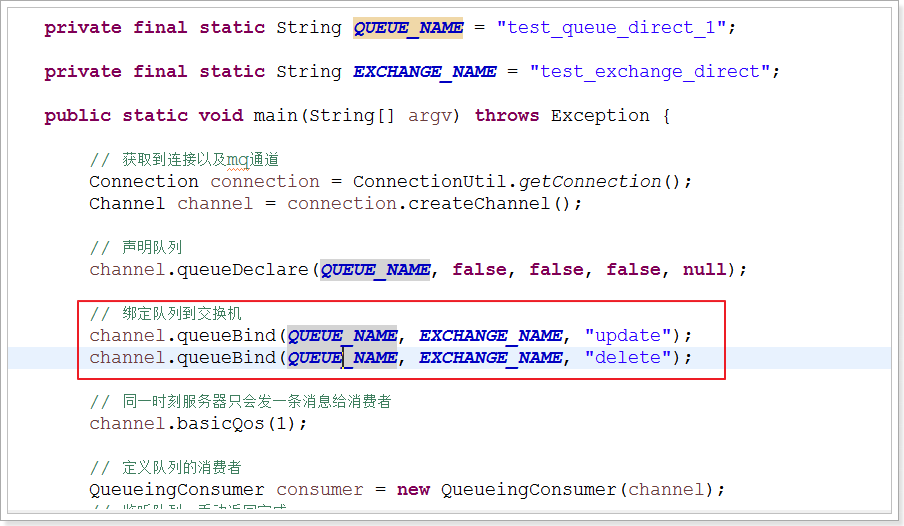
### 图示



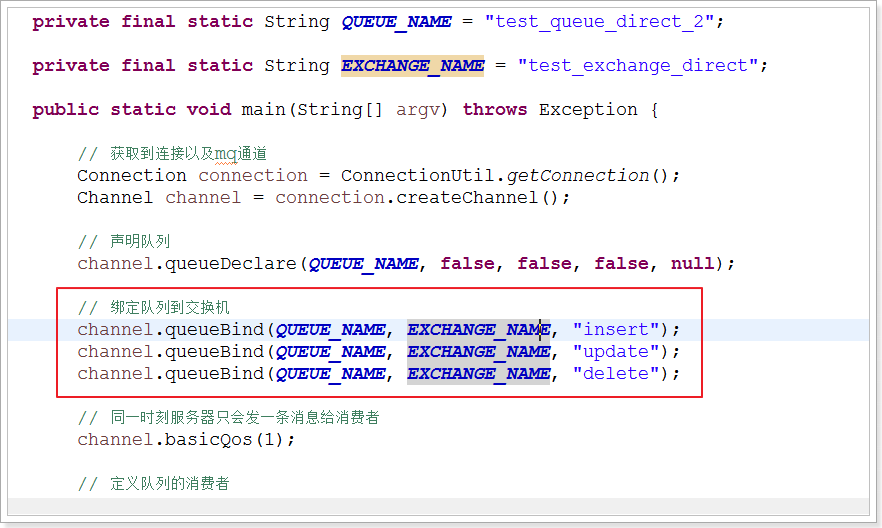
### 生产者



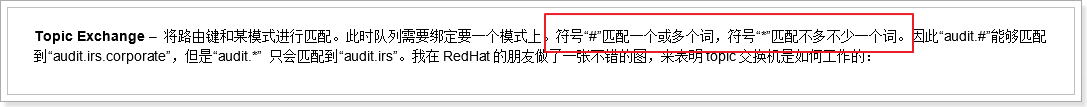
### 消费者1(前台系统)

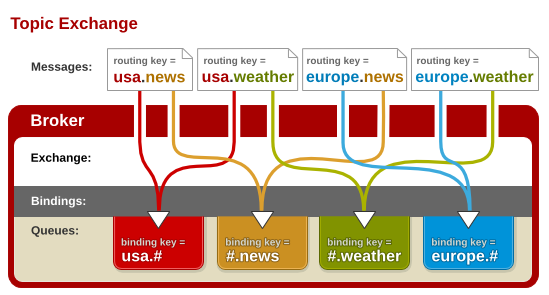


### 消费2（搜索系统）

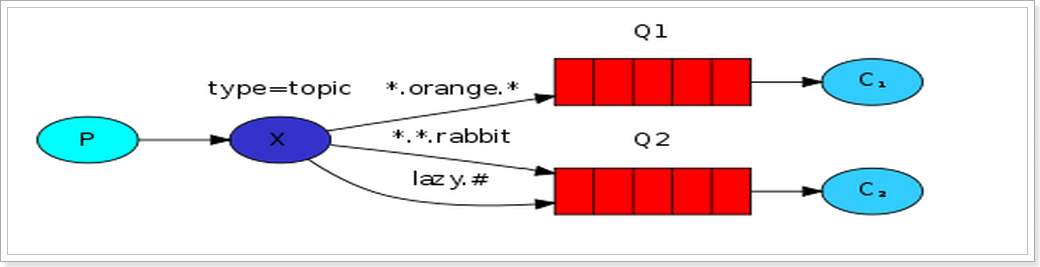


## 通配符模式

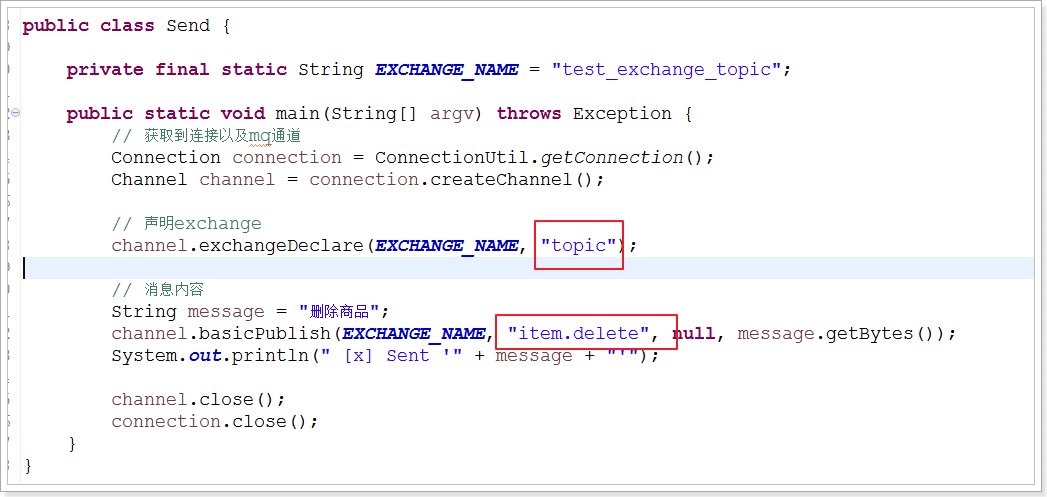




### 图示



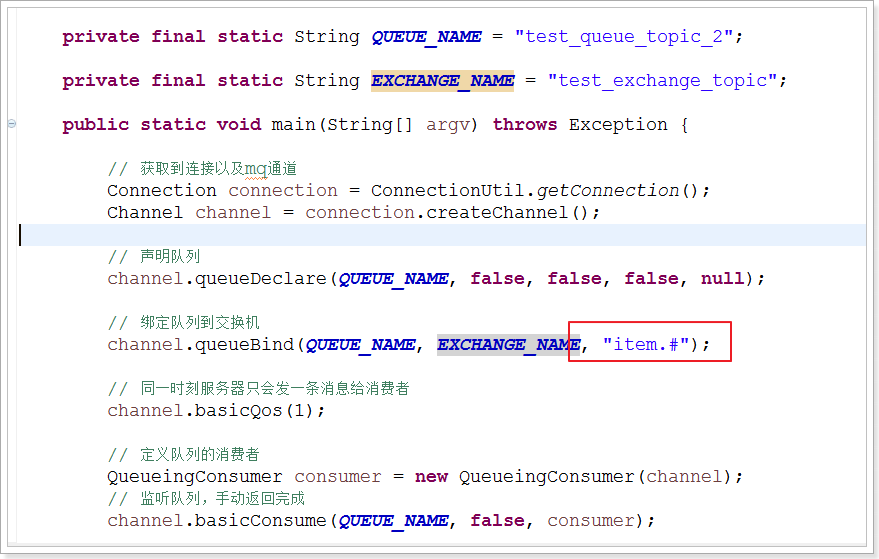
### 生产者



### 消费者1（前台系统）



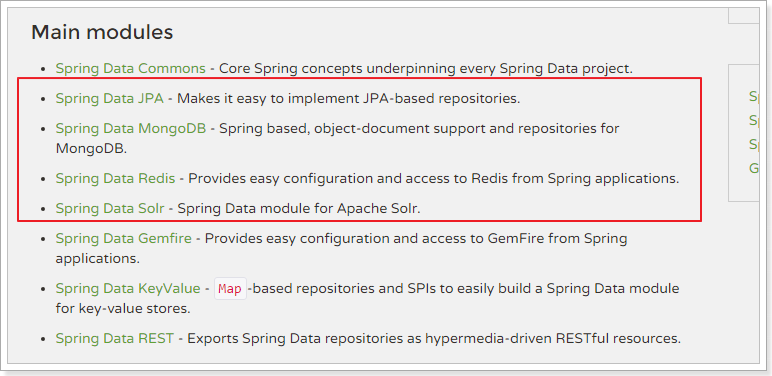
### 消费者2（搜索系统）

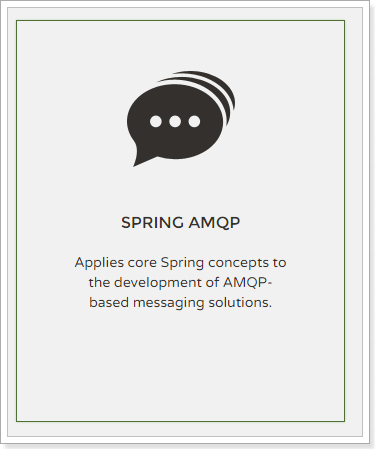


# Spring-Rabbit

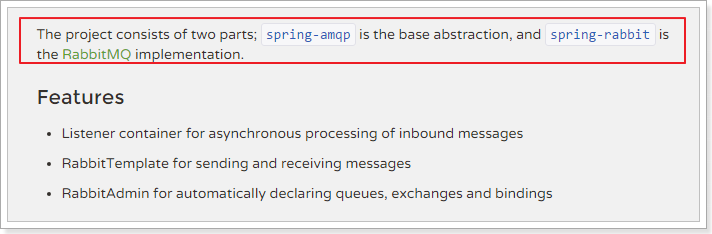
## Spring项目

<http://spring.io/projects>





## 简介



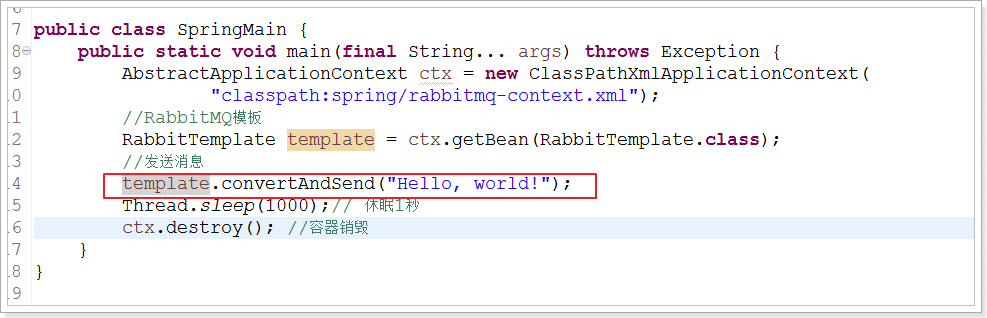


## 使用

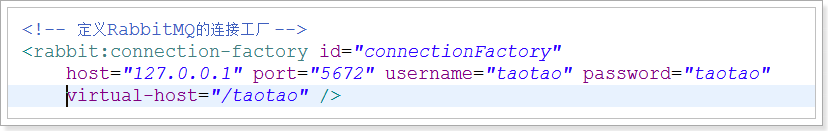
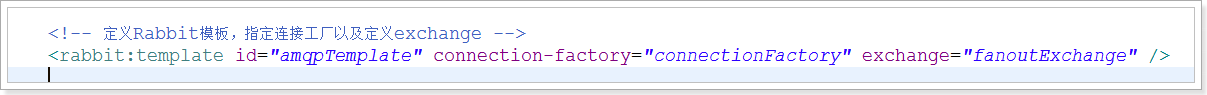
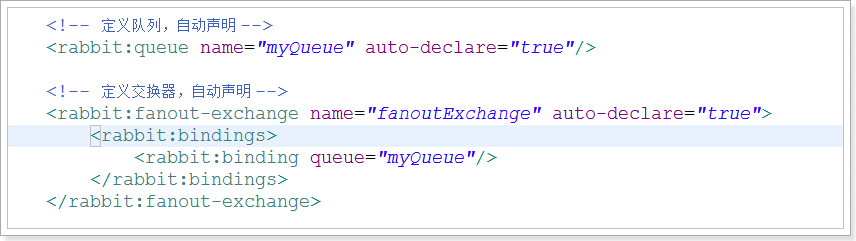
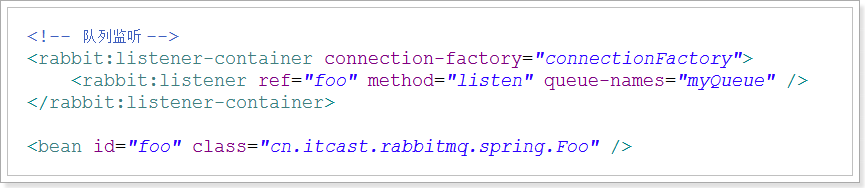
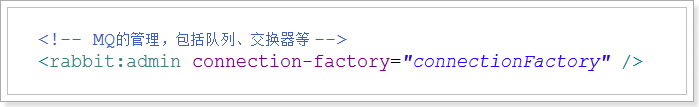
### 消费者



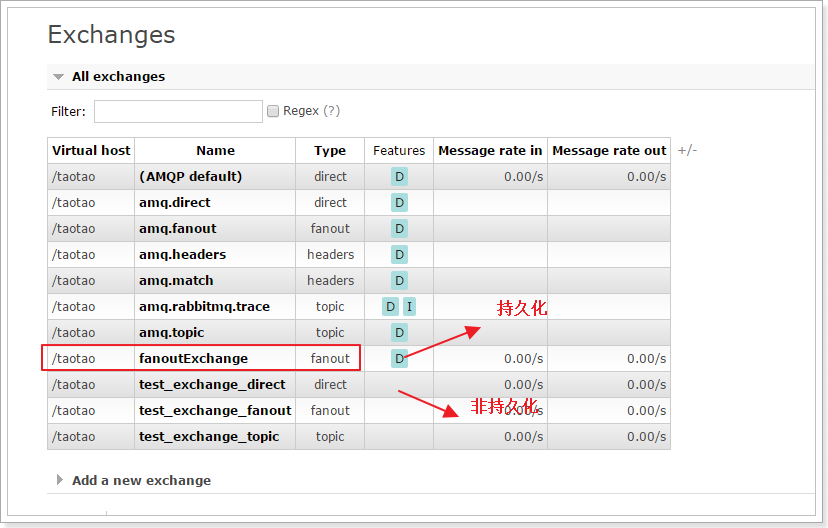
### 生产者



### 配置文件

1. 定义连接工厂  
   
2. 定义模板（可以指定交换机或队列）  
   
3. 定义队列、交换机、以及完成队列和交换机的绑定  
   
4. 定义监听  
   
5. 定义管理，用于管理队列、交换机等：  
   

## 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。

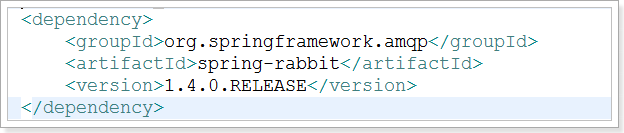
# 实现商品数据的同步

## 使用什么队列模式?

使用通配符模式。

## 在后台系统中发送消息到交换机

### 导入依赖



### 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

### 配置

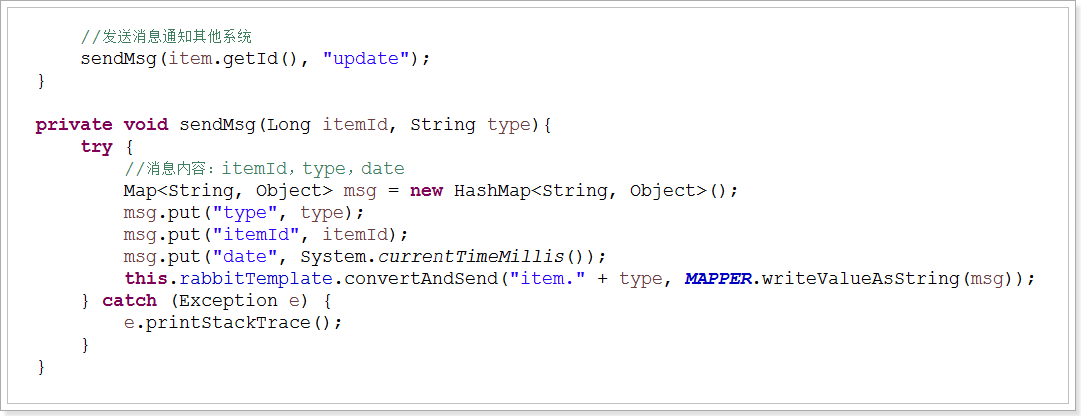


### 消息内容

方案：

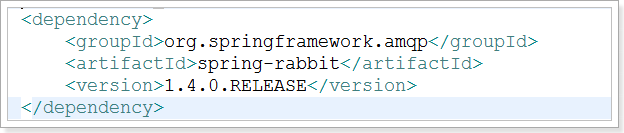
1. 将Item对象做json序列化发送
   1. 数据大
   2. 有些数据其他人是可能用不到的
2. 发送商品的id、操作类型

### 实现



## 前台系统接收消息

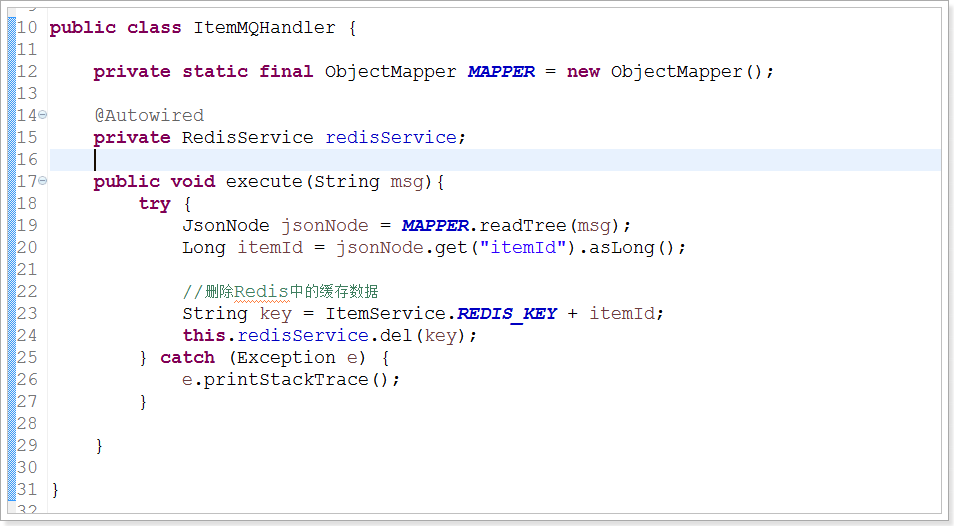
### 导入依赖



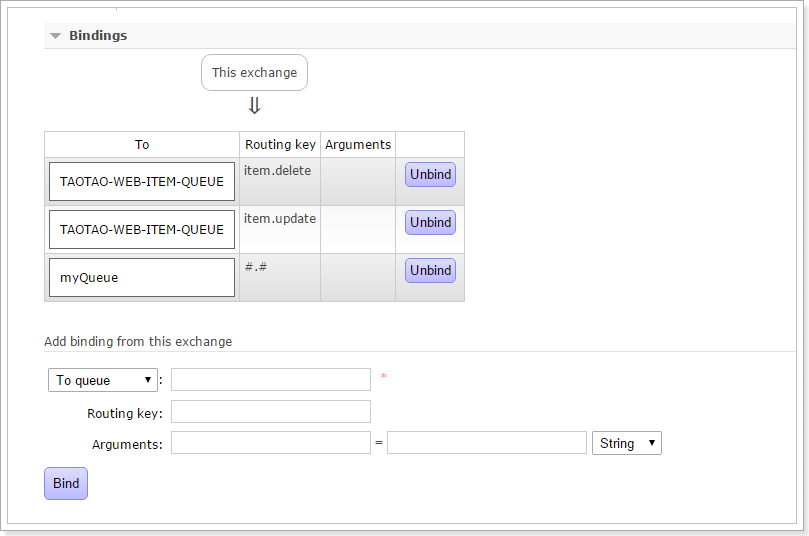
### 配置



### 具体处理逻辑



### 在界面管理工具中完成绑定关系



## 搜索系统中接收消息

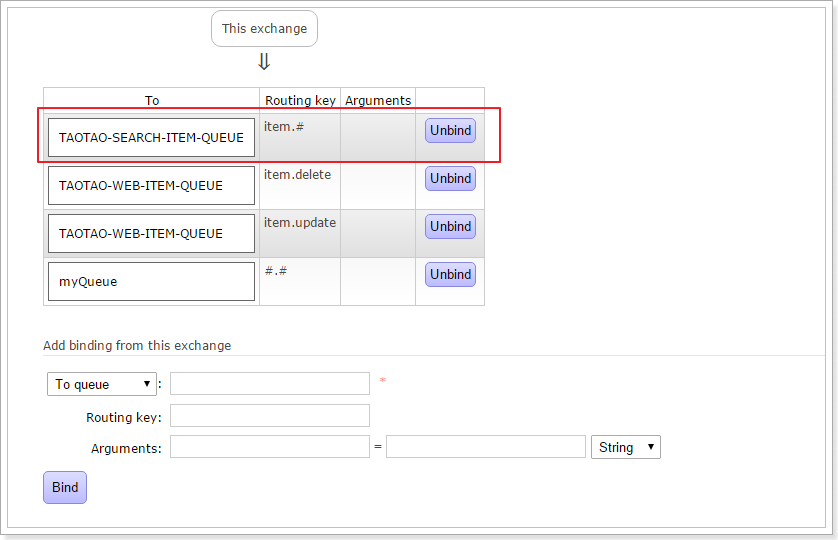
### 配置



### 处理业务逻辑



### 在管理工具中绑定队列和交换机



### 测试

测试结果：商品数据已经完成和索引数据的同步。

## 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