

RabbitMQ-消息中间件

目标

1. 理解消息中间件
2. 了解常用消息中间件
3. 能够安装rabbitMQ
4. 能够使用rabbitMQ发送消息
5. 能够使用邮件工具类
6. 能够使用rabbit发送注册邮件

第1章 消息中间件

1.1 为什么使用消息中间件

在生活中，我们经常会在网上碰到一些比较奇怪的案例，比如，我们经常在注册用户的时候，网站常常会给我们发送一封激活邮件，但是我们很久才能够收到这个邮件，我们在平常生活中进行一些支付的时候，经常会发生的情况是我们支付成功以后，一直到晚上或者第二天的时候才能够收到这个短信。为什么我们的信息不同步呢？没有实时的得到我们想要的结果。其实就是有消息中间件的存在。

如果没有消息中间件，可能出现的问题是：在互联网中有延迟或者网络原因的情况，这种情况我们如果无法避免的话，意味着需要一直等待。当我们进行消费后，我们需要等到对方收到钱以后才能够离开，可是可能已经到了第二天，这样的结果不是我们希望看到的。那么怎么解决这样的情况呢？

那么，在生活中如果遇到这样的情况，我们会找一个中间人，有对方做一个保障，那么所有的问题将会完美的解决。而消息中间件正好在这充当这么一个角色。

1.2 什么是消息中间件

消息中间件利用高效可靠的消息传递机制进行平台无关的数据交流，并基于数据通信来进行 分布式系统的集成。通过提供消息传递和消息排队模型，它可以在分布式环境下扩展进程间的通信。对于消息中间件，常见的角色 大致也就有 Producer（生产者）、Consumer（消费者）。消息队列中间件是分布式系统中重要的组件，主要解决应用解耦，异步消息，流量削峰等问题，实现高性能，高可用，可伸缩和最终一致性架构。

1.3 常见的消息中间件

应用场景 以下介绍消息队列在实际应用中常用的使用场景。异步处理，应用解耦，流量削峰和消息通讯四个 场景。

JMS1.1 和 J2EE 1.4 规范的 JMS Provider 实现。

2 RabbitMQ

AMQP 协议的领导实现，支持多种场景。淘宝的 MySQL 集群内部有使用它进行通讯，openStack 开源云平台的通信组件，最先在金融行业得到运用。

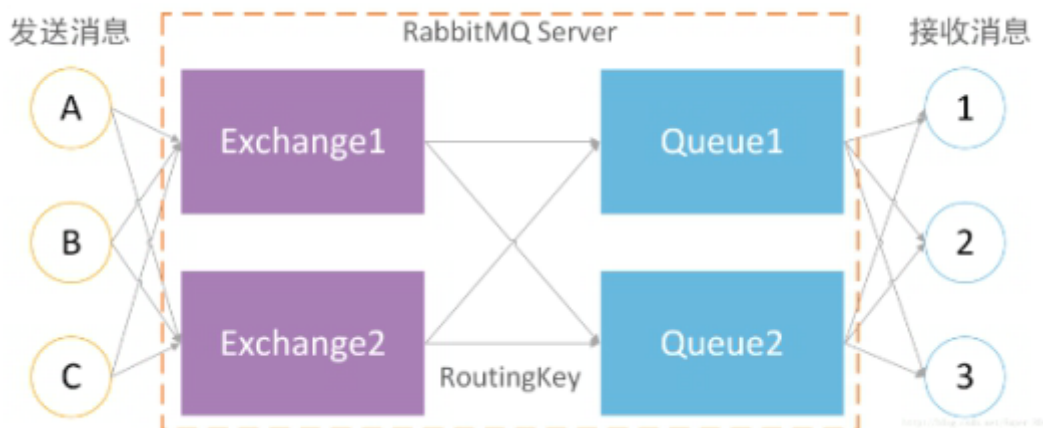
3 ZeroMQ

史上最快的消息队列系统。

4 Kafka

Apache 下的一个子项目。特点：高吞吐，在一台普通的服务器上既可以达到 10w/s 的吞吐速率；完全的分布式系统。适合处理海量数据。

1.4 架构图与主要概念



RabbitMQ Server：也叫broker server，它是一种传输服务。他的角色就是维护一条从Producer到Consumer的路线，保证数据能够按照指定的方式进行传输。

Producer：消息生产者，如图A、B、C，数据的发送方。消息生产者连接RabbitMQ服务器然后将消息投递到Exchange。

Consumer：消息消费者，如图1、2、3，数据的接收方。消息消费者订阅队列，RabbitMQ将Queue中的消息发送到消息消费者。

Exchange：生产者将消息发送到Exchange（交换器），由Exchange将消息路由到一个或多个Queue中（或者丢弃）。Exchange并不存储消息。RabbitMQ中的Exchange有 direct、fanout、topic、headers四种类型，每种类型对应不同的路由规则。

Queue：（队列）是RabbitMQ的内部对象，用于存储消息。消息消费者就是通过订阅

队列来获取消息的，RabbitMQ中的消息都只能存储在Queue中，生产者生产消息并最终投递到Queue中，消费者可以从Queue中获取消息并消费。多个消费者可以订阅同一个Queue，这时Queue中的消息会被平均分摊给多个消费者进行处理，而不是每个消费者

都收到所有的消息并处理。



Exchange Type与binding key固定的情况下（在正常使用时一般这些内容都是固定配置好的），我们的生产者就可以在发送消息给Exchange时，通过指定routing key来决定消息流向哪里。RabbitMQ为routing key设定的长度限制为255 bytes。

Connection：（连接）：Producer和Consumer都是通过TCP连接到RabbitMQ Server 的。以后我们可以看到，程序的起始处就是建立这个TCP连接。

Channels：（信道）：它建立在上述的TCP连接中。数据流动都是在Channel中进行的。也就是说，一般情况是程序起始建立TCP连接，第二步就是建立这个Channel。

VirtualHost：权限控制的基本单位，一个VirtualHost里面有若干Exchange和 MessageQueue，以及指定被哪些user使用

第2章 安装

rabbitMQ 需要现安装Erlang环境

rabbitmq

编辑

讨论

收藏 | 251 | 14

本词条由“科普中国”科学百科词条编写与应用工作项目 审核。

RabbitMQ是实现了高级消息队列协议（AMQP）的开源消息代理软件（亦称面向消息的中间件）。RabbitMQ服务器是用Erlang语言编写的，而集群和故障转移是构建在开放电信平台框架上的。所有主要的编程语言均有与代理接口通讯的客户端库。

中文名	消息队列	简称	MQ
外文名	Message Queue	释义	一种程序对程序的通信方法

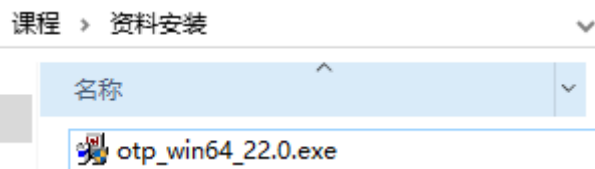
目录	1 简介	3 基本概念	5 安装
	2 历史	4 主要特性	6 参见

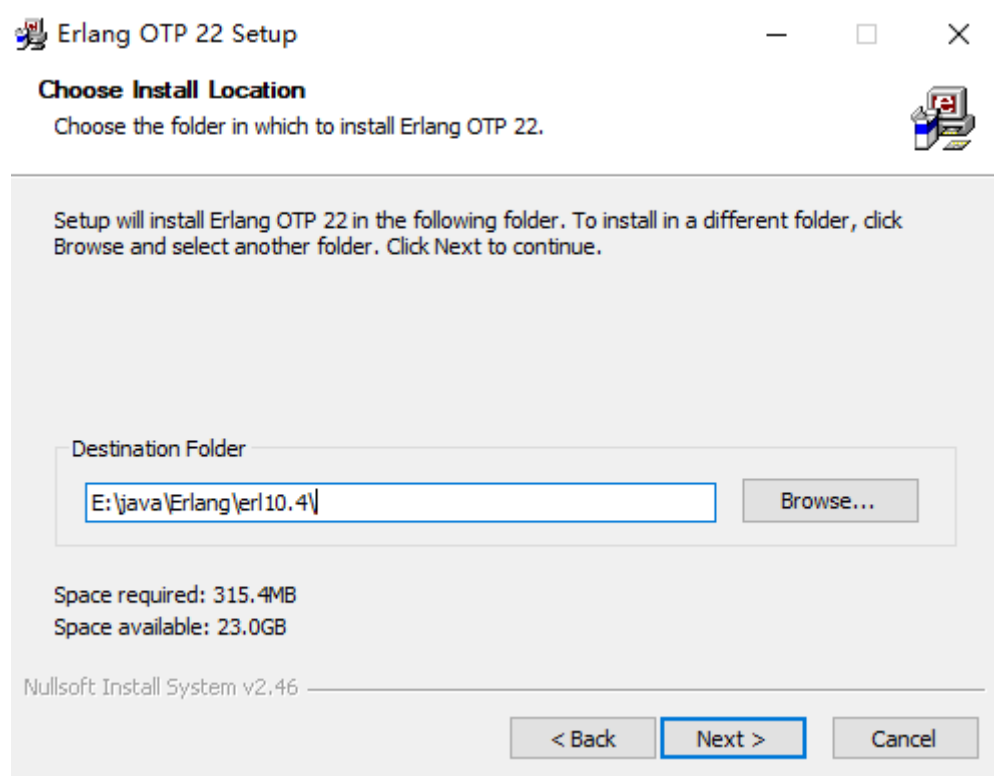
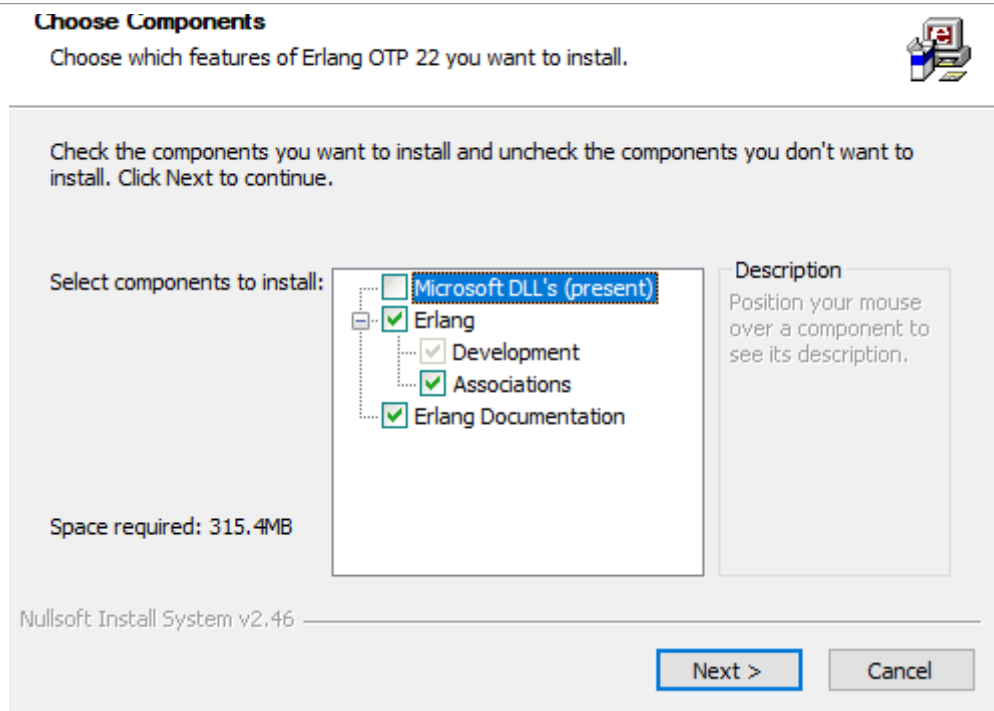
简介

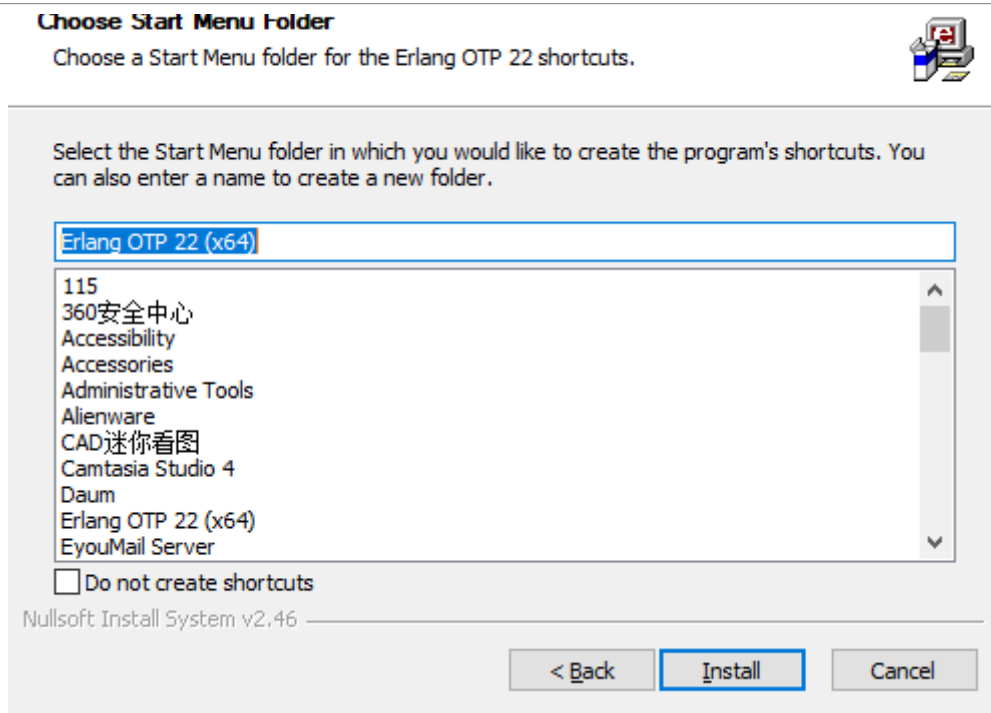
编辑

RabbitMQ是实现了高级消息队列协议（AMQP）的开源消息代理软件（亦称面向消息的中间件）。RabbitMQ服务器是用Erlang语言编写的，而集群和故障转移是构建在开放电信平台框架上的。所有主要的编程语言均有与代理接口通讯的客户端库。

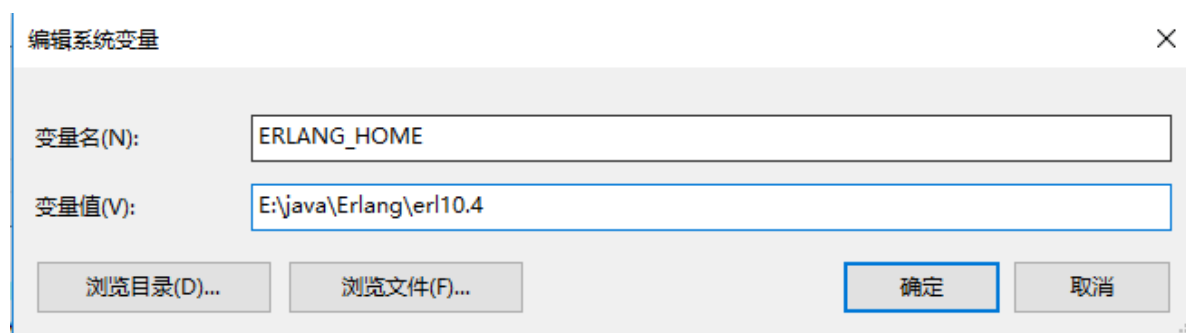
2.1 环境安装Erlang

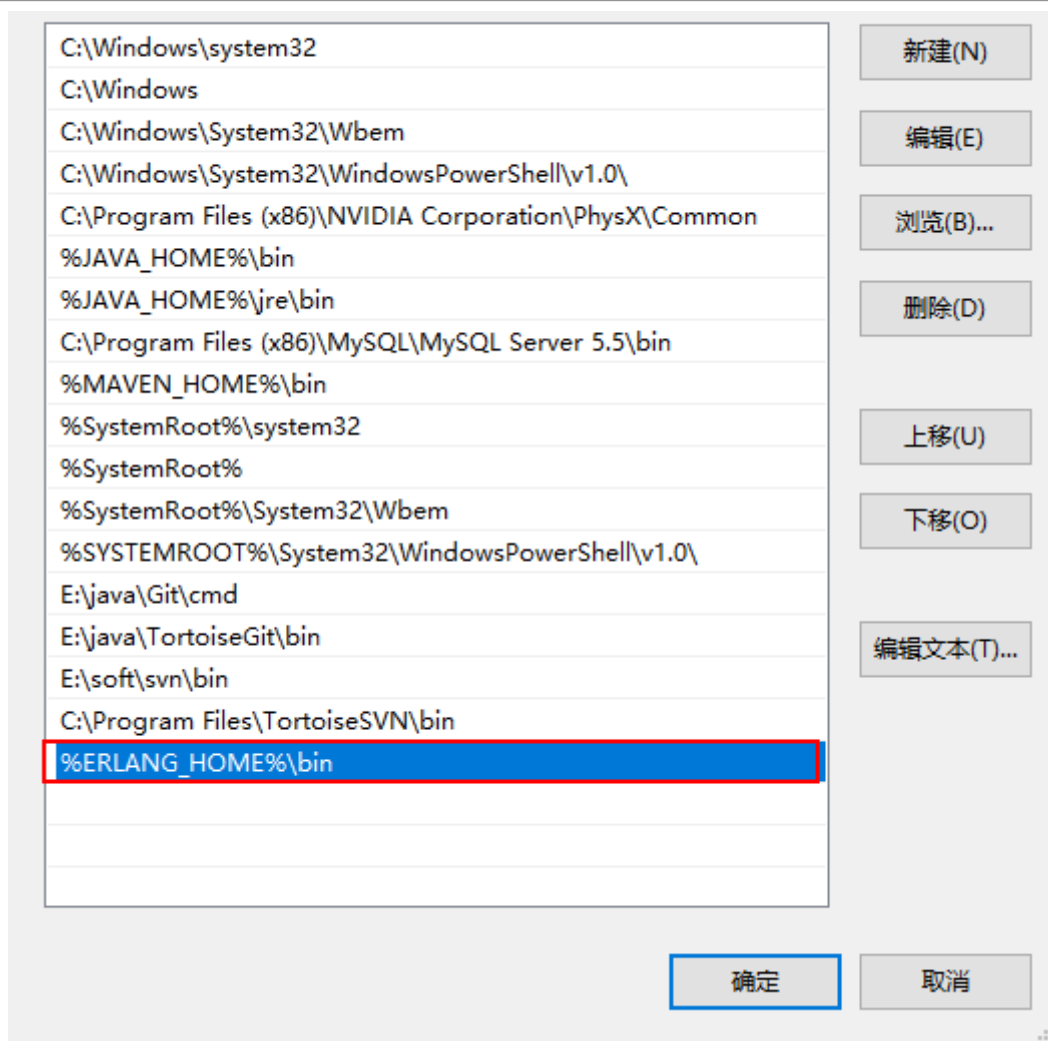






配置环境变量





测试安装成功

```
C:\Users\zy>erl -version
Erlang (SMP,ASYNC_THREADS) (BEAM) emulator version 10.4

C:\Users\zy>_
```

2.2 安装rabbitMQ服务器

Downloading and Installing RabbitMQ

The latest release of RabbitMQ is **3.7.17**. See [changelog](#) for release notes. See [RabbitMQ support timeline](#) to find out what release series are supported.

RabbitMQ Server

Installation Guides

- Linux, BSD, UNIX: [Debian, Ubuntu](#) | [RHEL, CentOS, Fedora](#) | [Generic binary build](#) | [Solaris](#)
- Windows: [Installer \(recommended\)](#) | [Binary build](#)
- MacOS: [Homebrew](#) | [Generic binary build](#) | [Standalone](#)
- [Erlang/OTP for RabbitMQ](#)

Installing on Windows



Download the Server

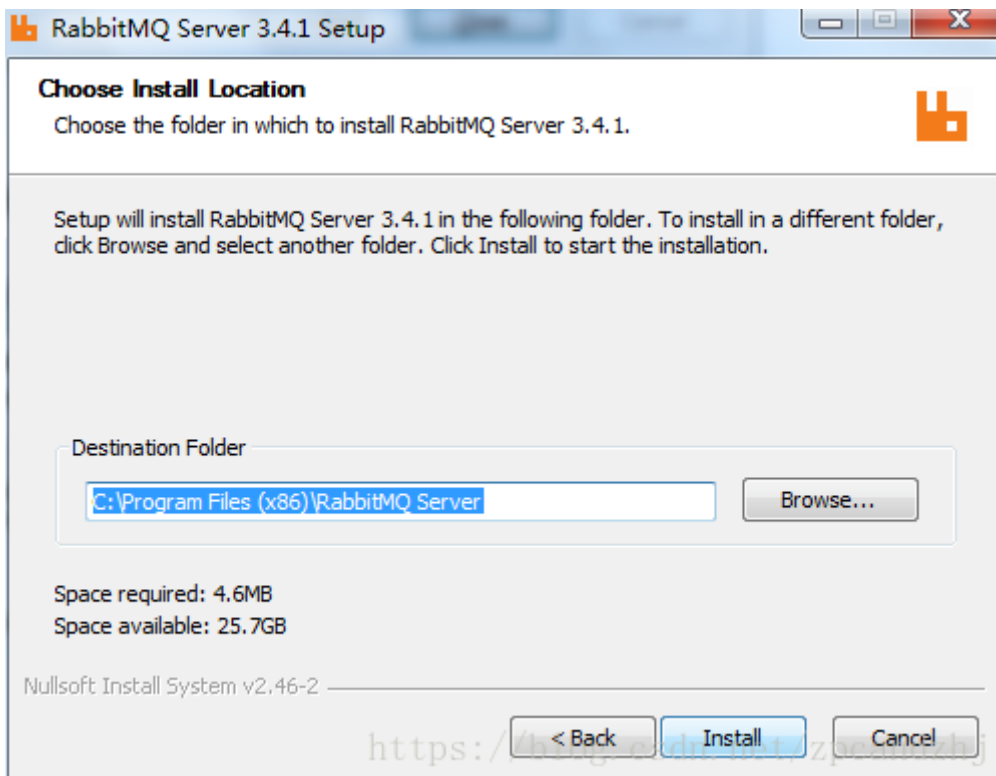
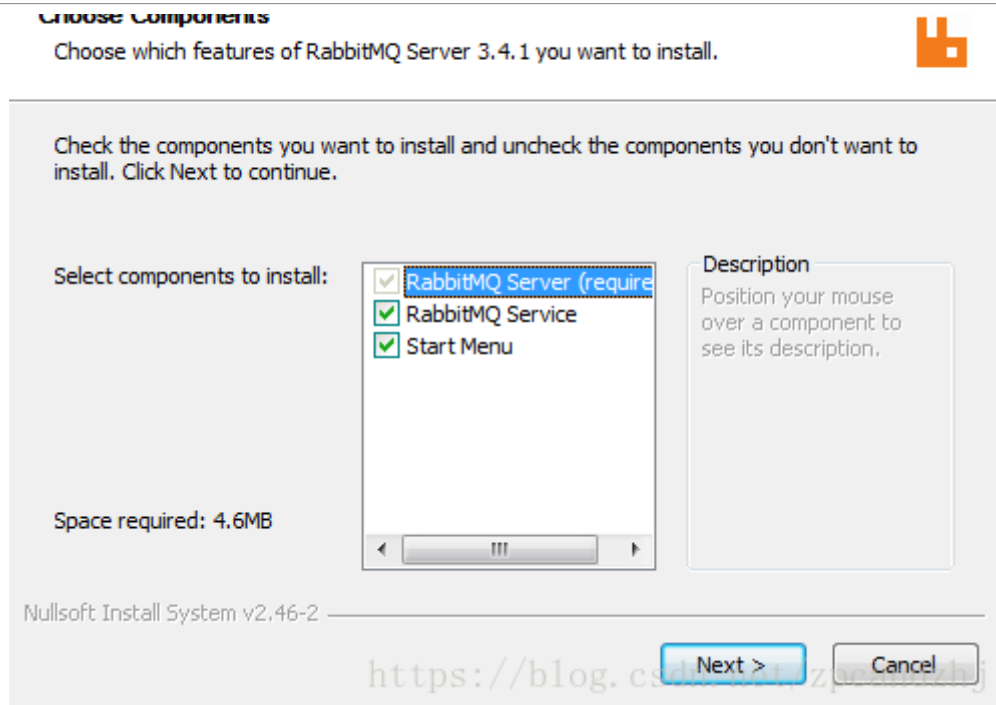
Description	Download	
Installer for Windows systems (from GitHub , recommended)	rabbitmq-server-3.7.17.exe	(Signature)
Alternative download location (from Bintray)	rabbitmq-server-3.7.17.exe	(Signature)

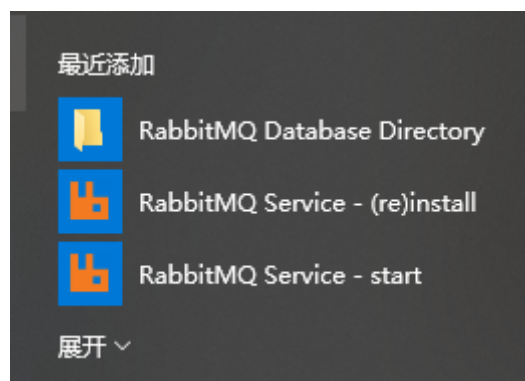
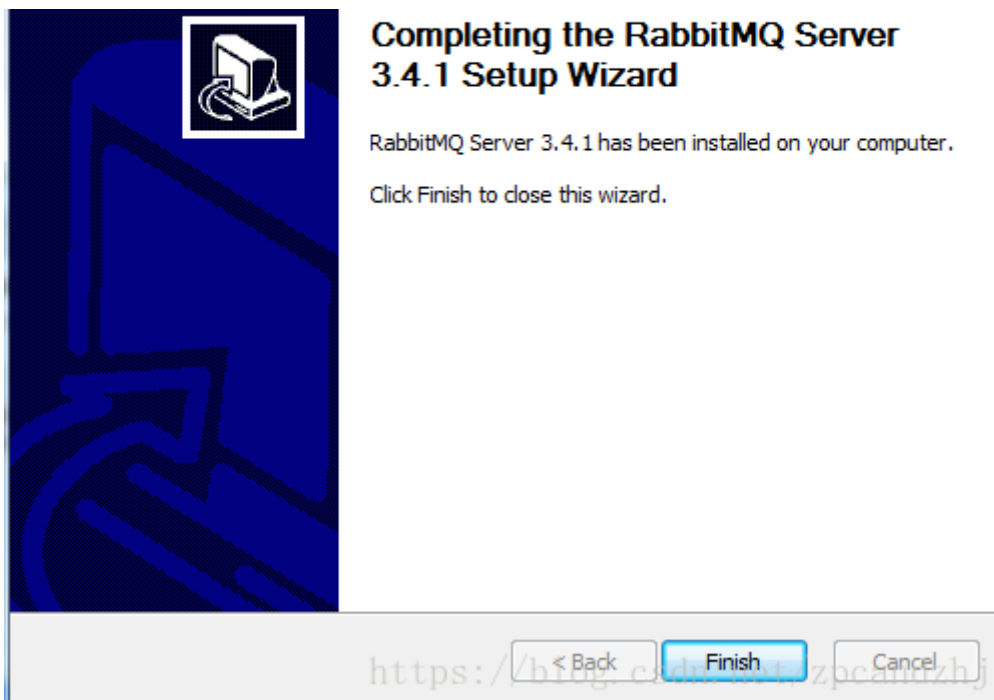
Install the Server

RabbitMQ requires a 64-bit [supported version of Erlang](#) for Windows to be installed. Erlang releases include a [Windows installer](#). [Erlang Solutions](#) provide binary 64-bit builds of Erlang as well.

课程 > 资料安装

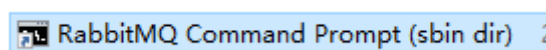
名称	修改日期	类型	大小
 otp_win64_22.0.exe	2019/8/15 20:17	应用程序	91,890 KB
 rabbitmq-server-3.7.17.exe	2019/8/14 16:42	应用程序	9,876 KB





本地磁盘 (C:) > Users > zy > AppData > Roaming > Microsoft > Windows > Start Menu > Programs > RabbitMQ Server				
	名称	修改日期	类型	大小
e	RabbitMQ Command Prompt (sbin d...	2019/8/15 20:34	快捷方式	2 KB
	RabbitMQ Database Directory	2019/8/15 20:34	快捷方式	1 KB
	RabbitMQ Logs	2019/8/15 20:34	快捷方式	1 KB
	RabbitMQ Plugins	2019/8/15 20:34	快捷方式	1 KB
	RabbitMQ Service - (re)install	2019/8/15 20:34	快捷方式	3 KB
	RabbitMQ Service - remove	2019/8/15 20:34	快捷方式	3 KB
	RabbitMQ Service - start	2019/8/15 20:34	快捷方式	3 KB
	RabbitMQ Service - stop	2019/8/15 20:34	快捷方式	3 KB
	Uninstall RabbitMQ	2019/8/15 20:41	快捷方式	1 KB

启动管理工具



在cmd中进入MQ安装路径: E:\java\rabbitMQ\rabbitmq_server-3.7.17\sbin

输入指令 : rabbitmq-plugins enable rabbitmq_management 启动插件

名称	修改日期	类型	大小
cuttlefish	2019/7/29 11:28	文件	458 KB
rabbitmqctl.bat	2019/7/29 11:28	Windows 批处理...	3 KB
rabbitmq-defaults.bat	2019/7/29 11:28	Windows 批处理...	2 KB
rabbitmq-diagnostics.bat	2019/7/29 11:28	Windows 批处理...	3 KB
rabbitmq-echopid.bat	2019/7/29 11:28	Windows 批处理...	2 KB
rabbitmq-env.bat	2019/7/29 11:28	Windows 批处理...	18 KB
rabbitmq-plugins.bat	2019/7/29 11:28	Windows 批处理...	3 KB
rabbitmq-server.bat	2019/7/29 11:28	Windows 批处理...	11 KB
rabbitmq-service.bat	2019/7/29 11:28	Windows 批处理...	14 KB

```
管理员: C:\Windows\system32\cmd.exe

C:\Program Files (x86)\RabbitMQ Server\rabbitmq_server-3.4.1\sbin>rabbitmq-plugins
ns enable rabbitmq_management
The following plugins have been enabled:
  mochiweb
  webmachine
  rabbitmq_web_dispatch
  amqp_client
  rabbitmq_management_agent
  rabbitmq_management

Applying plugin configuration to rabbit@WIN-A1JA6NUDTU4... started 6 plugins.
C:\Program Files (x86)\RabbitMQ Server\rabbitmq_server-3.4.1\sbin>
```

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



Username: *

Password: *

Login

使用默认账号登录：guest/ guest

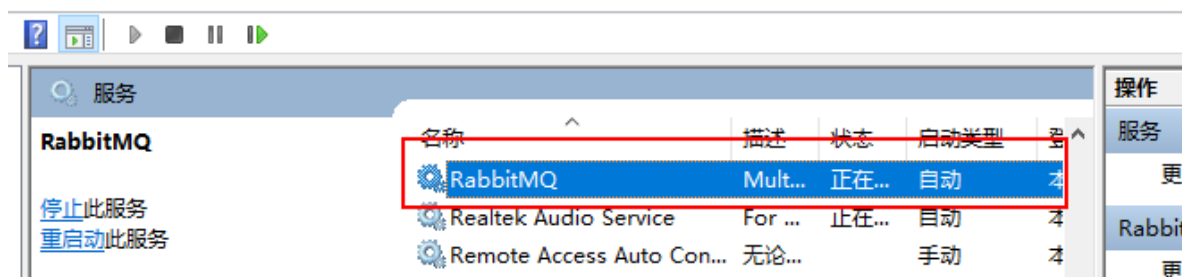
注意:rabbit在window中使用的时候，机器名称不能是中文否则安装可能 出问题

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 (推荐)

使用公用的RabbitMQ服务，在192.168.50.22

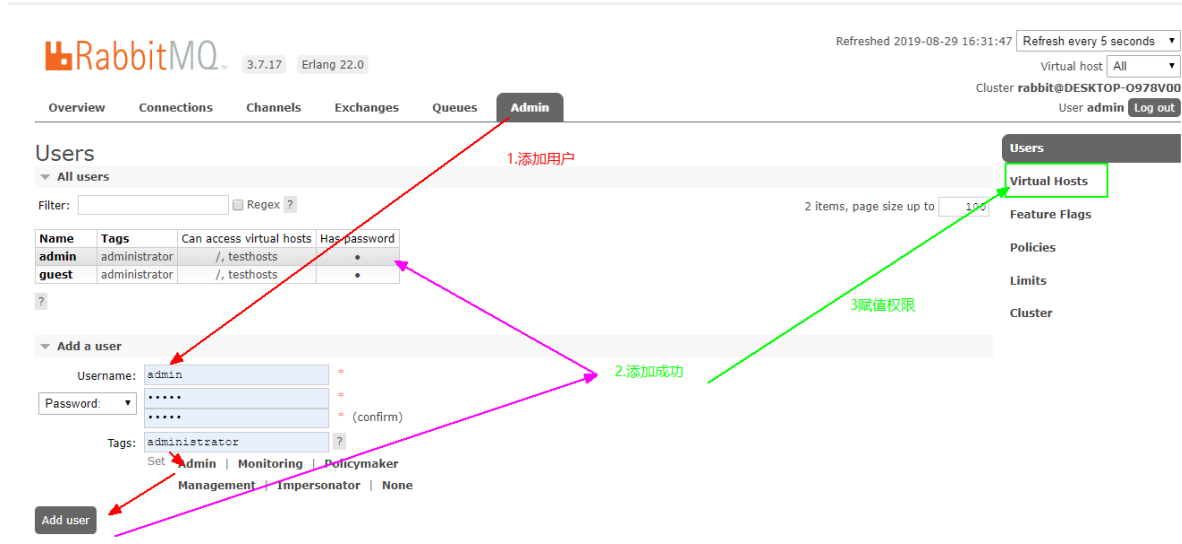
以后使用



2.3 管理界面



2.4 添加用户



2.5 创建Virtual Hosts

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	
/	admin, guest	running	NaN	NaN	NaN					
testhosts	admin, guest	running	NaN	NaN	NaN					

▼ Add a new virtual host

Name:

Add virtual host

创建权限组

[HTTP API](#) [Server Docs](#) [Tutorials](#) [Community Support](#) [Community Slack](#) [Commercial Support](#) [Plugins](#) [GitHub](#) [Changelog](#)

Users

Virtual Hosts

Feature Flags

Policies

Limits

Cluster

Virtual Hosts

▼ All virtual hosts

Filter: ☐ Regex ?

3 items, page size up to 100

Overview			Messages			Network		Message rates		+/-
Name	Users ?	State	Ready	Unacked	Total	From client	To client	publish	deliver / get	
/	admin, guest	running	NaN	NaN	NaN					
test	admin	running	NaN	NaN	NaN					
testhosts	admin, guest	running	NaN	NaN	NaN					

▼ Add a new virtual host

Name:

Add virtual host

点击进入

Users

Virtual Hosts

Feature Flags

Policies

Limits

Cluster

Overview Connections Channels Exchanges Queues Admin

Cluster rabbit@DESKTOP-0978V00

User admin Log out

Data rates last minute

Waiting for data...

Details

Tracing enabled: ☐
State: rabbit@DESKTOP-0978V00 : running

▼ Permissions

Current permissions

User	Configure regexp	Write regexp	Read regexp	
admin	<input type="text" value=".*"/>	<input type="text" value=".*"/>	<input type="text" value=".*"/>	<input type="button" value="Clear"/>

Set permission

权限

User:

Configure regexp:

Write regexp:

Read regexp:

Set permission

添加权限

▼ Topic permissions

Overview Connections Channels Exchanges Queues Admin

Cluster rabbit@DESKTOP-0978V00

User admin Log out

Users

▼ All users

Filter: ☐ Regex ?

3 items, page size up to 100

Name	Tags	Can access virtual hosts	Has password
admin	administrator	/, test, testhosts	•
admin2	administrator	No access	•
guest	administrator	/, testhosts	•

?

▼ Add a user

Username:

Password: (confirm)

Tags:

Set Admin | Monitoring | Policymaker
Management | Impersonator | None

Add user

找到新添加的用户 点击进入

Users

Virtual Hosts

Feature Flags

Policies

Limits

Cluster

User: admin2

This user does not have permission to access any virtual hosts.
Use "Set Permission" below to grant permission to access virtual hosts.

Overview

Tags administrator

Can log in with password

Permissions

Current permissions

... no permissions ...

Set permission

Virtual Host: /

Configure regexp: .*

Write regexp: .*

Read regexp: .*

Set permission

选择权限组 给用户添加相应的权限

Overview Connections Channels Exchanges Queues Admin

Cluster rabbit@DESKTOP-0978V00

User admin Log out

Users

All users

Filter: Regex ?

3 items, page size up to 100

Name	Tags	Can access virtual hosts	Has password
admin	administrator	/, test, testhosts	•
admin2	administrator	/, test, testhosts	•
guest	administrator	/, testhosts	•

添加成功

Add a user

Username: admin

Password:

(confirm)

Tags ?

Set Admin | Monitoring | Policymaker
Management | Impersonator | None

Add user

Overview Connections Channels Exchanges Queues Admin

/	(AMQP default)	direct	D		
/	amq.direct	direct	D		
/	amq.fanout	fanout	D		
/	amq.headers	headers	D		
/	amq.match	headers	D		
/	amq.rabbitmq.trace	topic	D I		
/	amq.topic	topic	D		
test	(AMQP default)	direct	D		
test	amq.direct	direct	D		
test	amq.fanout	fanout	D		
test	amq.headers	headers	D		
test	amq.match	headers	D		
test	amq.rabbitmq.trace	topic	D I		
test	amq.topic	topic	D		
testhosts	(AMQP default)	direct	D		
testhosts	amq.direct	direct	D		
testhosts	amq.fanout	fanout	D		

交换机中多了信息

第3章 rabbitMQ消息类型

rabbitMQ提供了六种消息模型，但第六种是RPC调用，并不是MQ，所以我们只需学习前五种即可。

1, 2两种属于队列方式

3, 4, 5属于订阅方式

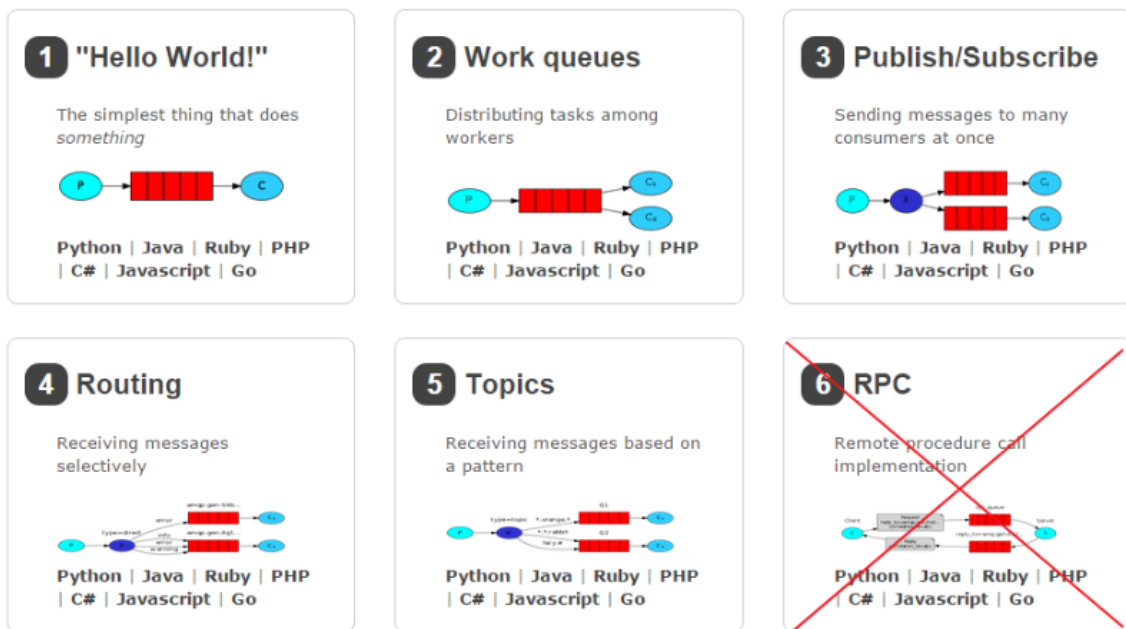
这里简单介绍下六种工作模式的主要特点。

work模式：一个生产者，多个消费者，每个消费者获取到的消息唯一。

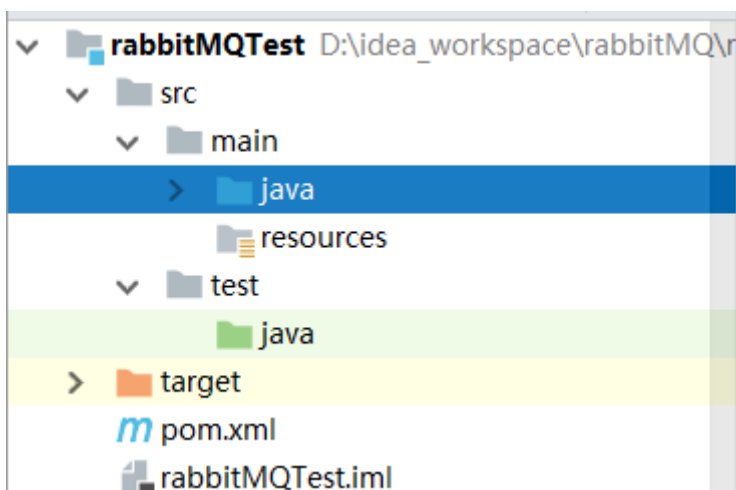
订阅模式：一个生产者发送的消息会被多个消费者获取。

路由模式：发送消息到交换机并且要指定路由key，消费者将队列绑定到交换机时需要指定路由key

topic模式：将路由键和某模式进行匹配，此时队列需要绑定在一个模式上，“#”匹配一个词或多个词，“*”只匹配一个词。



3.1 导入项目

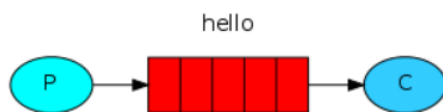


3.2 队列模式

3.2.1 简单模式-Hello-World

MQ说明: 我们可以认为所谓的MQ其实本身是一个邮局系统，需要有写信人的投递，才会后续的收件人收取

Our overall design will look like:



Producer sends messages to the "hello" queue. The consumer receives messages from that queue.

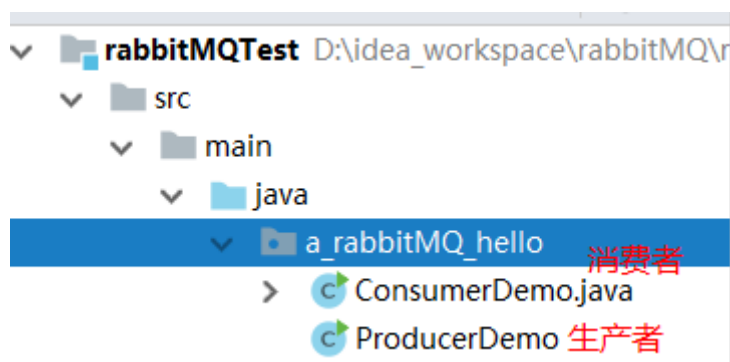
<https://blog.csdn.net/zpcandzhj>

P: 消息的生产者，消息的投递方

C: 消息的消费者，消息的接收方

红色：队列，存储消息的地方

测试过程：我们需要提前准备rabbitMQ服务器，生产者必须往消息队列中发送消息，消费者从队列中拿到消息并消费



3.2.1.1 生产者

```
package a_rabbitMQ_hello;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

public class ProducerDemo {

    private final static String QUEUE_NAME = "test_demo_1"; // 定义消息队列名称
    public static void main(String[] args) 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("testhosts");
        factory.setUsername("admin");
        factory.setPassword("admin");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
```



```
// 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();
}
```

执行生产者效果如下

Overview Connections Channels Exchanges **Queues** Admin

Queues

All queues (1)

Pagination

Page 1 of 1 - Filter: ☐ Regexp ?

Displaying 1 item, page size up to: 100

Overview	Virtual host	Name	Features	State	Ready	Unacked	Total	Message rates	Incoming	deliver / get	ack
testhosts	test_demo_1			idle	1	0	1	0.00/s			

▶ Add a new queue

HTTP API Server Docs Tutorials Community Support Community Slack Commercial Support Plugins GitHub Changelog

3.7.17 Erlang 22.0

Virtual host All

Cluster rabbit@DESKTOP-0978V00

User admin Log out

Overview Connections Channels Exchanges **Queues** Admin

▶ Publish message

▼ Get messages

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

Ack Mode:

Encoding:

Messages:

Get Message(s)

▶ Move messages

▶ Delete

▶ Purge

Get Message(s)

Message 1

The server reported 0 messages remaining.

Exchange	(AMQP default)
Routing Key	test_demo_1
Redelivered	0
Properties	
Payload	Hello World!
12 bytes	
Encoding: string	

队列中已经有消息



创建消费者消费消息

```
package a_rabbitMQ_hello;

import com.rabbitmq.client.*;
import java.io.IOException;

public class ConsumerDemo {
    private final static String QUEUE_NAME = "test_demo_1"; // 队列名称
    public static void main(String[] args) 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("testhosts");
        factory.setUsername("admin");
        factory.setPassword("admin");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
        // 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);
    }
}

// 消费者消费消息
class QueueingConsumer extends DefaultConsumer {
    // 构造接收通道
    public QueueingConsumer(Channel channel) {
        super(channel);
    }

    // 获取消息的方法，当监听到队列中有消息的时候 默认执行
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
        AMQP.BasicProperties properties, byte[] body) throws IOException {
        // super.handleDelivery(consumerTag, envelope, properties, body);
        String msg = new String(body);
        System.out.println("获得数据:" + msg);
    }
}
```

执行效果

OverviewConnectionsChannelsExchangesQueuesAdmin

User adminLog out

Queues

All queues (1)

Pagination

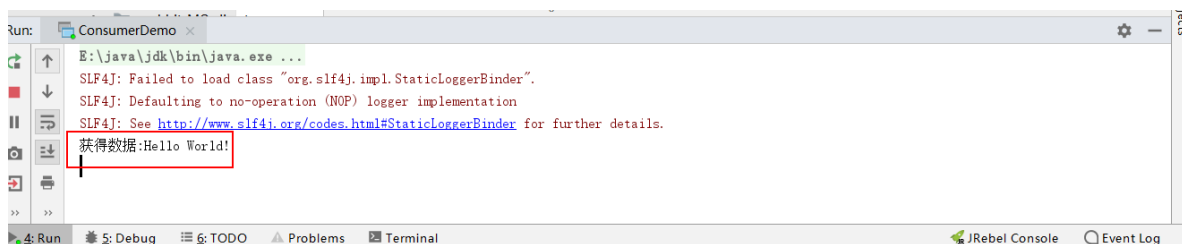
Page 1 of 1 - Filter: ☐ Regex ?

Displaying 1 item, page size up to: 100

Overview						Messages			Message rates			+/-
Virtual host	Name	Features	State		Ready	Unacked	Total	incoming	deliver	get	ack	
testhosts	test_demo_1		idle		0	0	0	0.00/s	0.00/s	0.00/s	0.00/s	

► Add a new queue

HTTP APIServer DocsTutorialsCommunity SupportCommunity SlackCommercial SupportPluginsGitHubChangelog



3.2.1.3 问题-消息回执

当消费者从队列中获得数据以后，MQ中就会将消息移除，但是，MQ是如何知道数据已经被获取了呢？

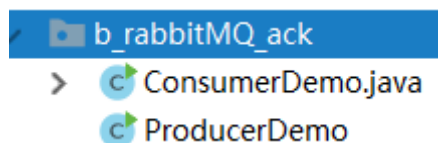
这就要取决于消息回执的机制（ACK，acknowledge），例如在生活中，快递员将快递交到我们手上的时候，需要签字确认是一个道理。当消费者拿到消息以后，会发送一个ACK给MQ。当MQ收到ACK以后，就会从MQ中移除消息，而ACK共分为两种形式：自动ACK和手动ACK

自动ACK：简而言之，自动ACK就是消费者拿到消息以后自动的发送ACK回执

手动ACK：我们需要在消费消息之后，手动的通知MQ消息已经被处理

应用场景：当我们处理比较重要的消息不允许丢失的情况，那么需要使用手动ACK，而类似于日志消息的时候，我们使用自动ACK即可

消费者代码-手动ACK



修改配置autoAck为false

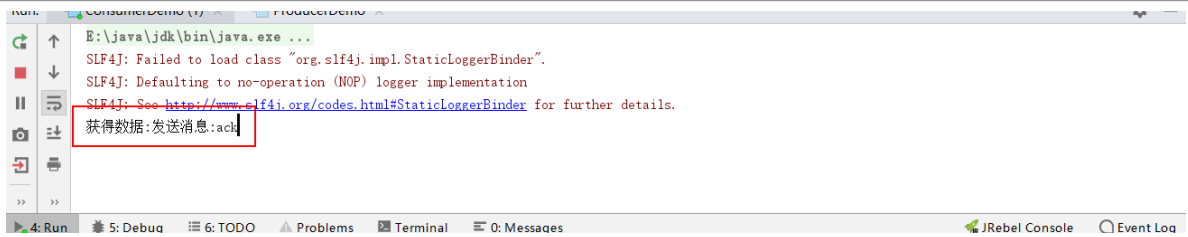
```

Channel channel = connection.createChannel();
// 声明队列
channel.queueDeclare(QUEUE_NAME, durable: false, exclusive: false,
autoDelete: false, arguments: null);

// 定义队列的消费者
QueueingConsumer consumer = new QueueingConsumer(channel);

// 监听队列
//参数2 表示为是否需要自动回执
channel.basicConsume(QUEUE_NAME, autoAck: false, consumer);
    
```

是否需要自动回执
true表示消费消息自动回执
false 不需要自动回执 需要手动回执



MQ的消息由就绪状态转换为没有收到ACK状态

► All queues (2)

Overview				Messages			Message rates			+/-
Virtual host	Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
testhosts	test_demo_1		idle	0	0	0	0.00/s	0.00/s	0.00/s	
testhosts	test_demo_2		idle	0	1	1	0.00/s	0.00/s	0.00/s	

一段时间后，如果还没收到回执，将回滚到就绪状态

Overview Connections Channels Exchanges Queues Admin

Queues

► All queues (2)

Overview				Messages		Message rates				+/-
Virtual host	Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
testhosts	test_demo_1		idle	0	0	0	0.00/s	0.00/s	0.00/s	
testhosts	test_demo_2		idle	1	0	1	0.00/s	0.00/s	0.00/s	

手动回执

在消费消息后确认消息收到回执

```
//获取消息的方法,当监听到队列中有消息的时候 默认执行
@Override
public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties,
byte[] body) throws IOException {
    //System.out.println(1/0);
    String msg = new String(body);
    System.out.println("获得数据:" + msg);
    /*try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    } */
    //确认收到一个或多个消息
    channel.basicAck(envelope.getDeliveryTag(), multiple: false);
}
```

MQ移除消息

Queues

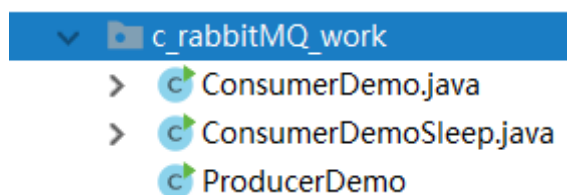
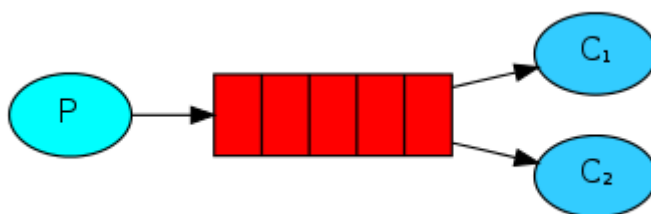
► All queues (2)

Overview				Messages			Message rates			+/-
Virtual host	Name	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
testhosts	test_demo_1		idle	0	0	0	0.00/s	0.00/s	0.00/s	
testhosts	test_demo_2		idle	0	0	0	0.00/s	0.00/s	0.00/s	

▼ Add a new queue

如图所示，在基本模型中，增加消费者，也就是一个队列有多个消费者处理。

场景来源：生活中，我们如果只有一个快递员负责运送所有的货物，那么，必然造成运输太慢的问题。而开发中同理，如果生产者生产的消息足够多，而消费者不能够及时的消费，会产生消息堆积的问题。如何解决呢？也就是再招聘几个人即可，所以，一个队列，有多个消费者进行消费，保证我们的执行效率问题。



3.2.2.1 生产者

```
public class ProducerDemo {

    private final static String QUEUE_NAME = "test_demo_3";
    public static void main(String[] args) 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("testhosts");
        factory.setUsername("admin");
        factory.setPassword("admin");
        // 通过工程获取连接
        Connection connection = factory.newConnection();

        // 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);

        for(int i = 0 ; i < 100 ; i ++){
            // 消息内容
            String message = "发送消息:" + i;
            channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
        }

        //关闭通道和连接
    }
}
```



```
}
```

3.2.2.2 消费者

```
public class ConsumerDemo {
    private final static String QUEUE_NAME = "test_demo_3";
    public static void main(String[] args) 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("testhosts");
        factory.setUsername("admin");
        factory.setPassword("admin");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
        // 从连接中创建通道
        Channel channel = connection.createChannel();

        // 声明队列
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        //设置每次取的数量
        //channel.basicQos(1);

        // 定义队列的消费者
        QueueingConsumer consumer = new QueueingConsumer(channel);

        // 监听队列
        //参数2表示为是否需要自动回执
        channel.basicConsume(QUEUE_NAME, false, consumer);
    }
}

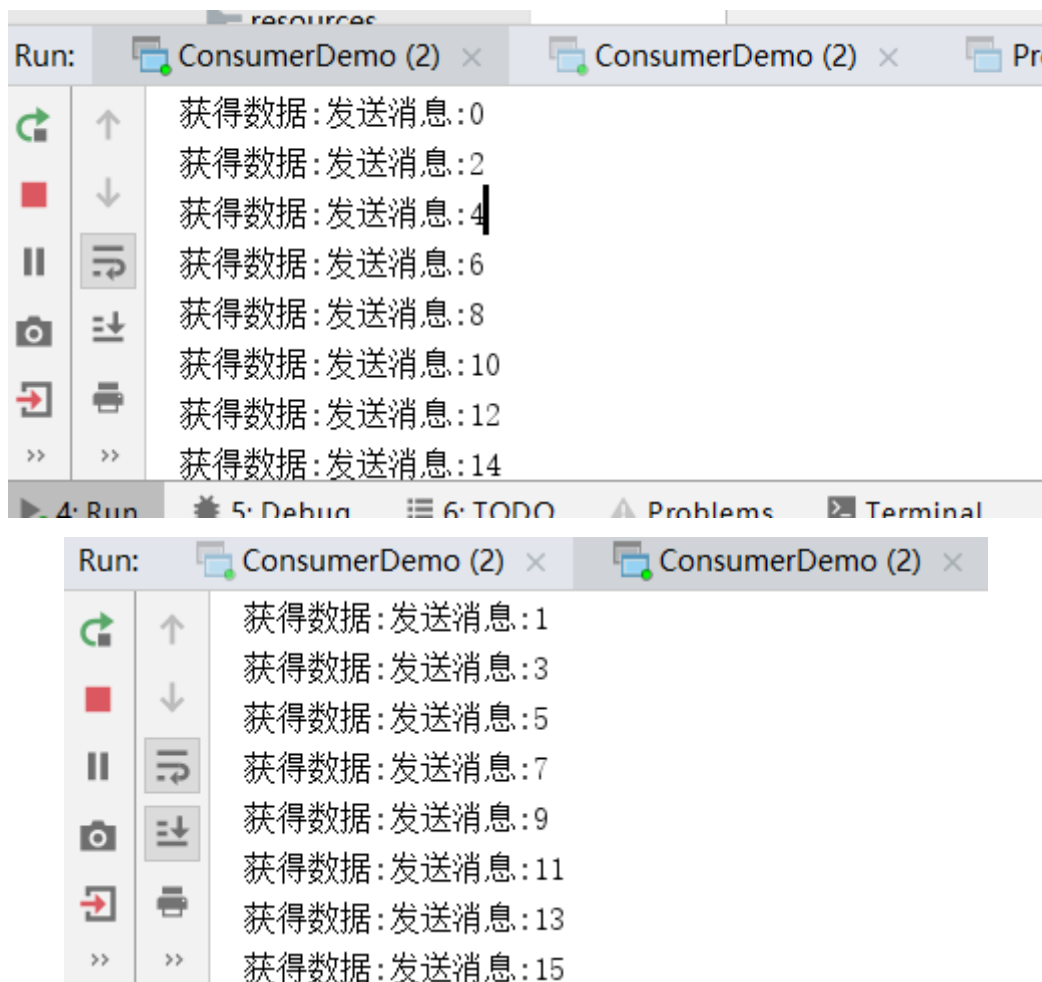
class QueueingConsumer extends DefaultConsumer{
    private final static String QUEUE_NAME = "test_demo_3";
    private Channel channel;
    //构造接收通道
    public QueueingConsumer(Channel channel) {
        super(channel);
        this.channel=channel;
    }

    //获取消息的方法,当监听到队列中有消息的时候 默认执行
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
        AMQP.BasicProperties properties, byte[] body) throws IOException {
        String msg = new String(body);
        System.out.println("获得数据:"+msg);
        //确认收到一个或多个消息
    }
}
```

3.2.2.3 执行过程

1. 先启动消费者两次
2. 然后启动生产者代码发送消息
3. 查看结果

3.2.2.4 执行效果



3.2.2.5 问题

我们发现两个消费者比较和谐，一人一次获得消息执行。但是在现实生活中好像是不太现实的，如果有一个消费者处理速度比较慢，那么消息必然造成等待的情况，因此我们采用能者多劳的形式，谁处理的快，谁就处理的多

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。



首先，我们在一个消费者中处理消息的时候加上睡眠1秒，一个已经处理完毕，另一个依旧在执行

```
E:\java\jdk\bin\java.exe ...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
获得数据xx:发送消息:1
获得数据xx:发送消息:3
获得数据xx:发送消息:5
获得数据xx:发送消息:7

发送消息:85
发送消息:86
发送消息:87
发送消息:88
发送消息:89
发送消息:90
发送消息:91
发送消息:92
发送消息:93
发送消息:94
发送消息:95
发送消息:96
发送消息:97
发送消息:98
发送消息:99

Process finished with exit code 0
```

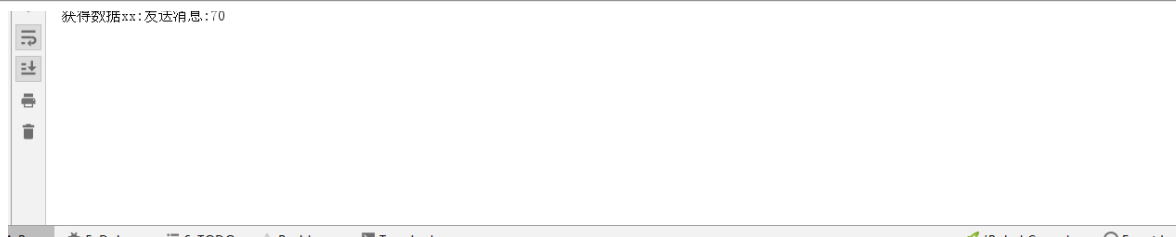
修改，限定每次获得一个，而每次消费完需要回执，保证了消费者处理的时候，如果有回执，才可以处理下一个消息

```
// 声明队列
channel.queueDeclare(QUEUE_NAME, durable: false, exclusive: false, autoDelete: false, arguments: null);

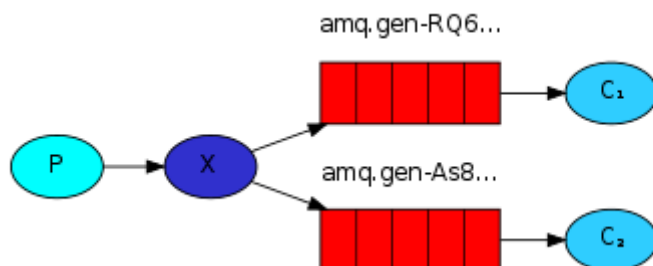
//设置每次取的数量
channel.basicQos( prefetchCount: 1);

// 定义队列的消费者
QueueingConsumer consumer = new QueueingConsumer(channel);
```

```
获得数据:发送消息:89
获得数据:发送消息:90
获得数据:发送消息:91
获得数据:发送消息:92
获得数据:发送消息:93
获得数据:发送消息:94
获得数据:发送消息:95
获得数据:发送消息:96
获得数据:发送消息:97
获得数据:发送消息:98
获得数据:发送消息:99
```



3.3 订阅模式-发布订阅-publish/subscribe



一个生产者发送消息在队列模式上，只有一个消费者能获得并消费，而主题模式就是一个生产者生产的消息，同时可以被多个消费者拿到消费，而队列和订阅模式有点像我们平常微信或者QQ中的私聊和频道聊天，在队列模式基础上加入了交换机（exchange），交换机的用途是帮助我们进行发送消息

P：表示producer 生产者，用于消息发布的一方

X：表示exchange交换机，用于传递消息，但不保存消息

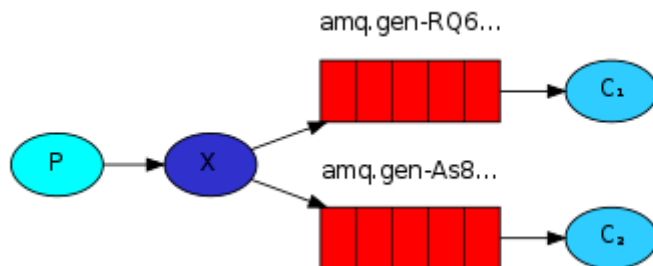
红色部分：队列，一个交换机上允许存在多个队列，当交换机产生数据发送的时候，会发送到多个队列上

C：表示consumer 消费者，消费者拿到消息并且消费

订阅模式解释

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，通过交换机来发送给不同的队列 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

3.3.1 订阅模式-fanout



3.3.1.1 步骤

1. 先将消费者绑定到同一个交换机上（但需要注意的是，交换机必须提前存在）
2. 然后由消息生成者发送消息
3. 查看结果


```
public class ProducerDemo {
    private final static String EXCHANGE_NAME = "test_exchange_fanout";

    public static void main(String[] argv) 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("testhosts");
        factory.setUsername("admin");
        factory.setPassword("admin");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
        //创建通道
        Channel channel = connection.createChannel();

        // 声明exchange
        //参数1: 交换机名称
        //参数2: 交换机类型
        channel.exchangeDeclare(EXCHANGE_NAME, "fanout");

        // 消息内容
        String message = "双十二折上折";
        channel.basicPublish(EXCHANGE_NAME, "", null, message.getBytes());
        System.out.println(" 大家好 " + message + "");

        channel.close();
        connection.close();
    }
}
```

3.3.1.3 消费者

```
public class ConsumerDemo {
    private final static String QUEUE_NAME = "test_queue_work1";
    private final static String EXCHANGE_NAME = "test_exchange_fanout";

    public static void main(String[] args) 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("testhosts");
        factory.setUsername("admin");
        factory.setPassword("admin");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
        // 从连接中创建通道
```



```
// 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, "");

// 定义队列的消费者
QueueingConsumer consumer = new QueueingConsumer(channel);

// 监听队列
//参数2表示为是否需要自动回执
channel.basicConsume(Queue_NAME, false, consumer);

}
}

class QueueingConsumer extends DefaultConsumer{

    private Channel channel;
    //构造接收通道
    public QueueingConsumer(Channel channel) {
        super(channel);
        this.channel=channel;
    }

    //获取消息的方法,当监听到队列中有消息的时候 默认执行
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
        AMQP.BasicProperties properties, byte[] body) throws IOException {
        String msg = new String(body);
        System.out.println("消费者张三获得数据:"+msg);
        //确认收到一个或多个消息
        channel.basicAck(envelope.getDeliveryTag() , false);
    }
}
```

3.3.1.4 执行效果

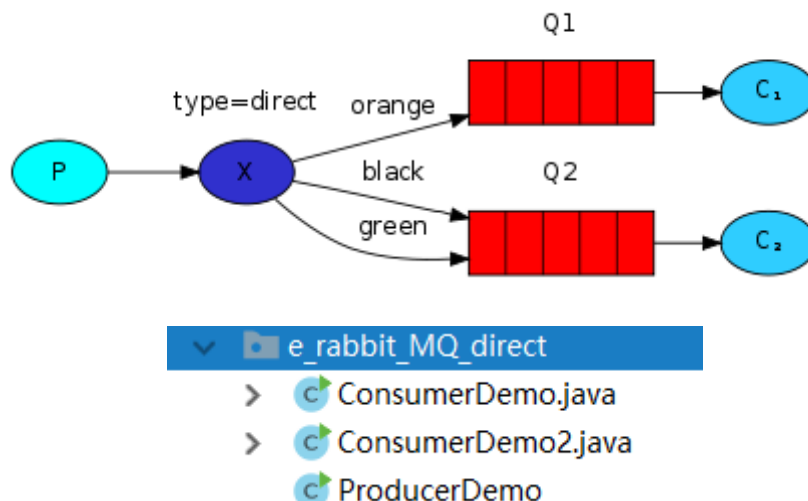
```
E:\java\jdk\bin\java.exe ...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
大家好 '双十二折上折'
Process finished with exit code 0
```

SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See <http://www.slf4j.org/codes.html#StaticLoggerBinder> for further details.
消费者李四获得数据:双十二折上折

多个队列同时绑定到一个交换机上，有生产者发送消息的时候，所有消费者都会接收到同一个消息

3.3.2 订阅模式-路由模式-direct

在fanout广播模式下，当生产者发送消息的时候，所有的消费者都会收到消息，但是，在某些场景会发现这种情况是有一定问题的，例如，我们在看电视的时候，常常说只关注我们自己喜欢的台就够了，但是其他人不一定喜欢，那我们就只告诉这几个人即可。



3.3.2.1 步骤

1. 需要多个消费者同时绑定到同一个交换机上，并通知交换机需要接收什么消息
2. 交换机会给某一部分的消费者发送消息，而这取决于消费者绑定的时候设定的routing key
3. 生产者发送消息

3.3.2.2 生产者

```
public class ProducerDemo {
    private final static String EXCHANGE_NAME = "test_exchange_direct";

    public static void main(String[] argv) 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("testhosts");
        factory.setUsername("admin");
        factory.setPassword("admin");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
        //创建通道
        Channel channel = connection.createChannel();

        // 声明exchange
```



```
// 消息内容
String message = "双十二折上折";
channel.basicPublish(EXCHANGE_NAME, "back", null, message.getBytes());
System.out.println(" 退货 '" + message + "'");

channel.close();
connection.close();
}
}
```

3.3.2.3 消费者1

```
public class ConsumerDemo {
    private final static String QUEUE_NAME = "test_queue_work1";
    private final static String EXCHANGE_NAME = "test_exchange_direct";

    public static void main(String[] args) 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("testhosts");
        factory.setUsername("admin");
        factory.setPassword("admin");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
        // 从连接中创建通道
        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, "buy");
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "back");
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "ask");

        // 定义队列的消费者
        QueueingConsumer consumer = new QueueingConsumer(channel);

        // 监听队列
        //参数2表示为是否需要自动回执
        channel.basicConsume(QUEUE_NAME, false, consumer);
    }
}

class QueueingConsumer extends DefaultConsumer{

    private Channel channel;
    // 消息队列
}
```



```
        this.channel=channel;
    }

    //获取消息的方法,当监听到队列中有消息的时候 默认执行
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
        AMQP.BasicProperties properties, byte[] body) throws IOException {
        //super.handleDelivery(consumerTag, envelope, properties, body);
        //System.out.println(1/0);
        String msg = new String(body);
        System.out.println("消费者张三获得数据:"+msg);
        //确认收到一个或多个消息
        channel.basicAck(envelope.getDeliveryTag() , false);
    }
}
```

3.3.2.4 消费者2

```
public class ConsumerDemo2 {
    private final static String QUEUE_NAME = "test_queue_work2";
    private final static String EXCHANGE_NAME = "test_exchange_direct";

    public static void main(String[] args) 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("testhosts");
        factory.setUsername("admin");
        factory.setPassword("admin");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
        // 从连接中创建通道
        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, "buy");

        // 定义队列的消费者
        QueueingConsumer2 consumer = new QueueingConsumer2(channel);

        // 监听队列
        //参数2表示为是否需要自动回执
        channel.basicConsume(QUEUE_NAME, false, consumer);
    }
}
```

```
//构造接收通道
public QueueingConsumer2(Channel channel) {
    super(channel);
    this.channel=channel;
}

//获取消息的方法,当监听到队列中有消息的时候 默认执行
@Override
public void handleDelivery(String consumerTag, Envelope envelope,
    AMQP.BasicProperties properties, byte[] body) throws IOException {
    //super.handleDelivery(consumerTag, envelope, properties, body);
    //System.out.println(1/0);
    String msg = new String(body);
    System.out.println("消费者李四获得数据:"+msg);
    //确认收到一个或多个消息
    channel.basicAck(envelope.getDeliveryTag() , false);
}
}
```

消费者1和消费者2订阅的routing key不一致

消费者1

```
channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "buy");
channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "back");
channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "ask");
```

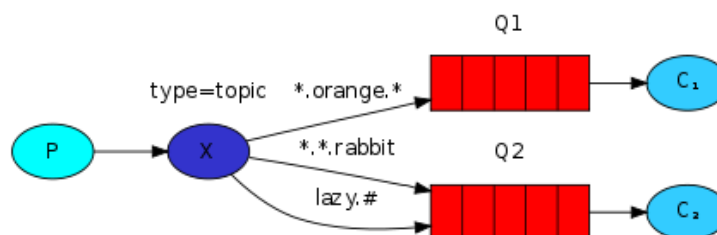
消费者2

```
channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "buy");
```

生产者发送消息的时候 需要额外指定发送到哪个频道，关注该频道的消费者都会接收到消息

```
channel.basicPublish(EXCHANGE_NAME, "back", null, message.getBytes());
```

3.3.3 订阅模式-主题模式-topics



路由模式下我们需要绑定一个routing key，而这个key每次我们只能绑定一个，那么有一个问题就是，如果我需要订阅频道更多的情况下，很明显，我们不适合使用这种情况，而MQ提供了第三种模式，叫主题模式，我们可以绑定某一类频道。主题模式跟路由模式基本类似，只不过可以使用通配符。

通配符#：匹配一个或多个词 例如 temp.# 可以匹配temp.buy.say 或者 temp.buy

通配符* 举例temp.* 可以匹配 temp.say或者 temp.buy

3.3.3.1 步骤

1. 需要有多多个消费者同时绑定到同一个交换机上，并通知交换机需要接收什么消息

3. 生产者发送消息

3.3.3.2 生产者

```
public class ProducerDemo {
    private final static String EXCHANGE_NAME = "test_exchange_topic";

    public static void main(String[] argv) 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("testhosts");
        factory.setUsername("admin");
        factory.setPassword("admin");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
        //创建通道
        Channel channel = connection.createChannel();

        // 声明exchange
        //参数1：交换机名称
        //参数2：交换机类型
        channel.exchangeDeclare(EXCHANGE_NAME, "topic");

        // 消息内容
        String message = "双十二折上折";
        //channel.basicPublish(EXCHANGE_NAME, "buy.phone", null,
        message.getBytes());
        channel.basicPublish(EXCHANGE_NAME, "ask.phone", null,
        message.getBytes());
        System.out.println(" 大家好 '" + message + "'");

        channel.close();
        connection.close();
    }
}
```

3.3.3.3 消费者1

```
public class ConsumerDemo {
    private final static String QUEUE_NAME = "test_queue_work1";
    private final static String EXCHANGE_NAME = "test_exchange_topic";

    public static void main(String[] args) 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);
```



```
factory.setUsername("admin");
factory.setPassword("admin");
// 通过工程获取连接
Connection connection = factory.newConnection();
// 从连接中创建通道
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, "buy");
channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "back");
channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "ask");*/
channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "buy.*");

// 定义队列的消费者
QueueingConsumer consumer = new QueueingConsumer(channel);

// 监听队列
//参数2表示为是否需要自动回执
channel.basicConsume(QUEUE_NAME, false, consumer);

}
}

class QueueingConsumer extends DefaultConsumer{

    private Channel channel;
    //构造接收通道
    public QueueingConsumer(Channel channel) {
        super(channel);
        this.channel=channel;
    }

    //获取消息的方法,当监听到队列中有消息的时候 默认执行
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
        AMQP.BasicProperties properties, byte[] body) throws IOException {
        //super.handleDelivery(consumerTag, envelope, properties, body);
        //System.out.println(1/0);
        String msg = new String(body);
        System.out.println("消费者张三获得数据:"+msg);
        //确认收到一个或多个消息
        channel.basicAck(envelope.getDeliveryTag() , false);
    }
}
```

3.3.3.4 消费者2

```
public class ConsumerDemo2 {
    private final static String QUEUE_NAME = "test_queue_work2";
    private final static String EXCHANGE_NAME = "test_exchange_topic";
```



```
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("testhosts");
factory.setUsername("admin");
factory.setPassword("admin");
// 通过工程获取连接
Connection connection = factory.newConnection();
// 从连接中创建通道
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, ".*.*");

// 定义队列的消费者
QueueingConsumer2 consumer = new QueueingConsumer2(channel);

// 监听队列
//参数2表示为是否需要自动回执
channel.basicConsume(Queue.NAME, false, consumer);

}
}

class QueueingConsumer2 extends DefaultConsumer{

    private Channel channel;
    //构造接收通道
    public QueueingConsumer2(Channel channel) {
        super(channel);
        this.channel=channel;
    }

    //获取消息的方法,当监听到队列中有消息的时候 默认执行
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
        AMQP.BasicProperties properties, byte[] body) throws IOException {
        //super.handleDelivery(consumerTag, envelope, properties, body);
        //System.out.println(1/0);
        String msg = new String(body);
        System.out.println("消费者李四获得数据:"+msg);
        //确认收到一个或多个消息
        channel.basicAck(envelope.getDeliveryTag() , false);
    }
}
```

3.3.3.5 执行效果

第4章 Spring整合RabbitMQ

4.1 目标

由于原生的rabbitMQ编写起来太过麻烦，所以，我们需要整合spring进行操作会更简单。

4.2 配置准备

创建两个web工程，导入同样的配置

4.3 POM

```
<dependencies>
  <dependency>
    <groupId>com.rabbitmq</groupId>
    <artifactId>amqp-client</artifactId>
    <version>5.7.3</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.amqp</groupId>
    <artifactId>spring-rabbit</artifactId>
    <version>2.1.7.RELEASE</version>
  </dependency>

  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.9.5</version>
  </dependency>
</dependencies>
```

4.4 生产者

4.4.1 配置文件

```
<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"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/rabbit
http://www.springframework.org/schema/rabbit/spring-rabbit-2.1.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.1.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context-4.1.xsd">

  <!--包扫描-->
  <context:component-scan base-package="com.itheima"></context:component-scan>
```



```

class="org.springframework.amqp.rabbit.connection.CachingConnectionFactory">
    <property name="virtualHost" value="testhosts"></property>
    <!-- username,访问RabbitMQ服务器的账户,默认是guest -->
    <property name="username" value="admin" />
    <!-- username,访问RabbitMQ服务器的密码,默认是guest -->
    <property name="password" value="admin" />
    <!-- host,RabbitMQ服务器地址,默认值"localhost" -->
    <property name="host" value="127.0.0.1" />
    <!-- port, RabbitMQ服务端口,默认值为5672 -->
    <property name="port" value="5672" />
</bean>

<rabbit:admin connection-factory="connectionFactory"/>

<!--解析json , MQ会自动将数据转换成json字符串-->
<bean id="jackson2JsonMessageConverter"
class="org.springframework.amqp.support.converter.Jackson2JsonMessageConverter"/
>

<!--模版对象 用于发送消息-->
<rabbit:template id="amqpTemplate" exchange="test-mq-exchange" connection-
factory="connectionFactory" message-converter="jackson2JsonMessageConverter"
/>

<!--配置队列 durable="true" 表示消息需要持久化 即服务器崩溃会数据不会丢失 auto-
delete="false" 表示服务器停止后数据是否自动删除 exclusive:表示该消息队列是否只在当前
connection生效。默认false。 -->
<rabbit:queue id="test_queue_key2" name="mail.send" durable="true" auto-
delete="false" exclusive="false" />

<!--交换机
    定义消息队列,durable:是否持久化,如果想在RabbitMQ退出或崩溃的时候,不会失去所有的
queue和消息,需要同时标志队列(queue)和交换机(exchange)是持久化的,即rabbit:queue标签和
rabbit:direct-exchange中的durable=true,而消息(message)默认是持久化的可以看类
org.springframework.amqp.core.MessageProperties中的属性public static final
MessageDeliveryMode DEFAULT_DELIVERY_MODE =
MessageDeliveryMode.PERSISTENT;exclusive: 仅创建者可以使用的私有队列,断开后自动删除;
auto_delete: 当所有消费客户端连接断开后,是否自动删除队列

    绑定队列,rabbitmq的exchangeType常用的三种模式: direct, fanout, topic三种,我们用
direct模式,即rabbit:direct-exchange标签,Direct交换器很简单,如果是Direct类型,就会将消
息中的RoutingKey与该Exchange关联的所有Binding中的BindingKey进行比较,如果相等,则发送到该
Binding对应的Queue中。有一个需要注意的地方: 如果找不到指定的exchange,就会报错。但routing
key找不到的话,不会报错,这条消息会直接丢失,所以此处要小心,auto-delete:自动删除,如果为Yes,
则该交换机所有队列queue删除后,自动删除交换机,默认为false
-->
<rabbit:direct-exchange name="test-mq-exchange" durable="true" auto-
delete="false" id="test-mq-exchange">
    <rabbit:bindings>
        <!--绑定队列 队列名称为mail.send-->
        <rabbit:binding queue="test_queue_key2" key="mail.send"/>
    </rabbit:bindings>
</rabbit:direct-exchange>
</beans>

```

4.4.2 接口

```
    * 发送消息到指定队列
    * @param queueKey
    * @param object
    */
    public void sendDataToQueue(String queueKey, Object object);
}
```

4.4.3 实现类

```
@Service
public class MQProducerImpl implements MQProducer {
    //注入模版对象
    @Autowired
    private AmqpTemplate amqpTemplate;

    public void sendDataToQueue(String queueKey, Object object) {
        //发送消息
        //convertAndSend: 将Java对象转换为消息发送到匹配Key的交换机中Exchange，由于配置了JSON转换，这里是将Java对象转换成JSON字符串的形式。
        amqpTemplate.convertAndSend(queueKey, object);
    }
}
```

4.4.4 测试代码

```
public class TestDemo {
    public static void main(String[] args) {

        ClassPathXmlApplicationContext ac = new
        ClassPathXmlApplicationContext("applicationContext.xml");
        MQProducer mqProducer = ac.getBean(MQProducer.class);
        //构建数据map数据
        HashMap<String, String> stringStringHashMap = new HashMap<>();
        stringStringHashMap.put("key1", "value1");
        stringStringHashMap.put("key2", "value1");
        stringStringHashMap.put("key3", "value1");
        mqProducer.sendDataToQueue("mail.send", stringStringHashMap);

        //构建对象数据发送
        //mqProducer.sendDataToQueue("mail.send", new
        MailObject("jack", "1234"));
        ac.close();
    }
}
```

4.4.5 执行效果

消息队列中已经存储了json的数据，接着就是我们需要获得数据的部分

Message 1

The server reported 0 messages remaining.

Exchange test-mq-exchange
Routing Key mail.send
Redelivered 0
Properties
 priority: 0
 delivery_mode: 2
 headers: __ContentTypeId__: java.lang.Object
 __KeyTypeId__: java.lang.Object
 __TypeId__: java.util.HashMap
content_encoding: UTF-8
content_type: application/json

Payload
49 bytes
Encoding: string
{"key1": "value1", "key2": "value1", "key3": "value1"}

4.5 消费者

4.5.1 配置文件

```
<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"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/rabbit/spring-rabbit-2.1.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.1.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context-4.1.xsd">

    <!--连接工厂-->
    <bean id="connectionFactory"
        class="org.springframework.amqp.rabbit.connection.CachingConnectionFactory">
        <!--<constructor-arg value="${mq.vhost}" />-->
        <property name="virtualHost" value="testhosts"/></property>
        <!-- username,访问RabbitMQ服务器的账户,默认是guest -->
        <property name="username" value="admin" />
        <!-- username,访问RabbitMQ服务器的密码,默认是guest -->
        <property name="password" value="admin" />
        <!-- host,RabbitMQ服务器地址,默认值"localhost" -->
        <property name="host" value="127.0.0.1" />
        <!-- port, RabbitMQ服务端口,默认值为5672 -->
        <property name="port" value="5672" />
    </bean>
    <rabbit:admin connection-factory="connectionFactory" />

    <rabbit:queue name="mail.send" auto-declare="true" durable="true" />
    <!-- 消费者部分 -->
    <!-- 自定义接口类 -->
    <bean id="testHandler" class="com.itheima.listener.MvListener"></bean>
```



```
<bean id="jackson2JsonMessageConverter"
class="org.springframework.amqp.support.converter.Jackson2JsonMessageConverter"/
>
```

<!-- 配置监听acknowledge="manual"设置手动应答，它能够保证即使在一个worker处理消息的时候用CTRL+C来杀掉这个worker，或者一个consumer挂了(channel关闭了、connection关闭了或者TCP连接断了)，也不会丢失消息。因为RabbitMQ知道没发送ack确认消息导致这个消息没有被完全处理，将会对这条消息做re-queue处理。如果此时有另一个consumer连接，消息会被重新发送至另一个consumer会一直重发,直到消息处理成功,监听容器acknowledge="auto" concurrency="30"设置发送次数,最多发送30次 -->

```
<rabbit:listener-container connection-factory="connectionFactory"
acknowledge="auto" concurrency="20">
    <rabbit:listener queues="mail.send" ref="testHandler" />
</rabbit:listener-container>
</beans>
```

4.5.2 监听代码

```
package com.itheima.listener;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.core.MessageListener;

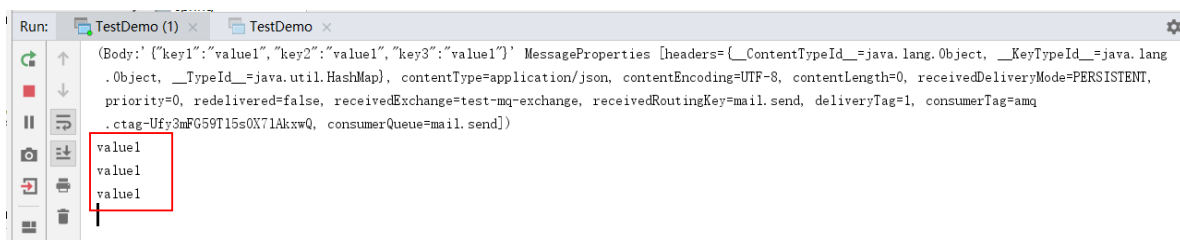
//实现监听器接口
public class MyListener implements MessageListener{
    //定义转换json的接口
    private static final ObjectMapper MAPPER = new ObjectMapper();
    public void onMessage(Message message) {
        try {
            //msg就是rabbitmq传来的消息，需要的同学自己打印看一眼
            // 使用jackson解析
            System.out.println(message.toString());
            JsonNode jsonData = MAPPER.readTree(message.getBody());
            System.out.println(jsonData.get("key1").asText());
            System.out.println(jsonData.get("key2").asText());
            System.out.println(jsonData.get("key3").asText());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4.5.2 测试代码

```
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestDemo {
    public static void main(String[] args) {
        //解析配置文件 监听器启动就可以从队列获得我们想要的数
        ClassPathXmlApplicationContext ac = new
        ClassPathXmlApplicationContext("applicationContext.xml");
    }
}
```

4.5.3 执行效果



第5章 发送邮件

5.1 需求

添加用户成功，向用户的邮箱中发送一封邮件。

5.2 环境搭建

5.2.1 POM文件

```
<dependencies>
    <dependency>
        <groupId>com.rabbitmq</groupId>
        <artifactId>amqp-client</artifactId>
        <version>5.7.3</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.amqp</groupId>
        <artifactId>spring-rabbit</artifactId>
        <version>2.1.7.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.5</version>
    </dependency>
</dependencies>
```

5.2.2 发送邮件工具类



```
import javax.mail.Address;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.util.Properties;

public class MailUtil {

    /**
     *
     * @param to 收件人
     * @param subject 主题
     * @param content 内容
     * @throws Exception
     */
    //实现邮件发送的方法
    public static void sendMsg(String to ,String subject ,String content) throws
Exception{
        Properties props = new Properties();
        props.setProperty("mail.smtp.host", "smtp.163.com"); //设置主机地址
        smtp.qq.com      smtp.sina.com

        props.setProperty("mail.smtp.auth", "true");//授权认证 代码客户端访问 必须设置
        为true 需要手机验证

        //2.产生一个用于邮件发送的Session对象
        Session session = Session.getInstance(props);

        //3.产生一个邮件的消息对象
        MimeMessage message = new MimeMessage(session);

        //4.设置消息的发送者
        Address fromAddr = new InternetAddress("itcast_hzb163@163.com");
        message.setFrom(fromAddr);

        //5.设置消息的接收者
        Address toAddr = new InternetAddress(to);
        //TO 直接发送 CC抄送 BCC密送
        message.setRecipient(MimeMessage.RecipientType.TO, toAddr);

        //6.设置主题
        message.setSubject(subject);
        //7.设置正文
        message.setText(content);

        //8.准备发送，得到火箭
        Transport transport = session.getTransport("smtp");
        //9.设置火箭的发射目标
        transport.connect("smtp.163.com", "itcast_hzb163@163.com", "true123"); //
        密码 授权密码!=登陆密码
        //10.发送
        transport.sendMessage(message, message.getAllRecipients());
    }
}
```



```
public static void main(String[] args) throws Exception {  
    MailUtil.sendMsg("itcast_hzb126@126.com" , "我们去玩把", "你好");  
}  
  
}
```

5.3 传统发送邮件代码

```
/**  
 * 添加或修改  
 * @return  
 */  
@RequestMapping(value = "/edit" ,name = "添加或者修改")  
public String edit(User user ) throws Exception {  
    //手动赋值数据 公司的id和公司的名称 根据登陆用户得到  
    user.setCompanyName(super.companyName);  
    user.setCompanyId(super.companyId);  
    if(StringUtils.isBlank(user.getId())){  
        String password = user.getPassword();  
        //对密码进行加密处理  
        String md5Pwd = Encrypt.md5(user.getPassword(), user.getEmail());  
        user.setPassword(md5Pwd);  
        userService.save(user); //1.已经往数据库添加了数据  
        //2.发送邮件  
        MailUtil.sendMsg(user.getEmail() , "恭喜您注册成功" , "欢迎加入saas-export大  
平台,我们将赠送您一百块的优惠券,您的密码是:"+ password);  
    }else{  
        userService.update(user);  
    }  
    return "redirect:/system/user/list.do";  
}
```

5.3.1 依赖

```
<!-- Javamail -->  
<dependency>  
    <groupId>javax.mail</groupId>  
    <artifactId>mail</artifactId>  
    <version>1.4.4</version>  
</dependency>
```

5.4 整合MQ发送邮件

5.4.1 POM

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

```
<groupId>org.springframework.amqp</groupId>
<artifactId>spring-rabbit</artifactId>
<version>2.1.7.RELEASE</version>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.5</version>
</dependency>
```

5.4.2 生产者配置文件

```
<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"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/rabbit/spring-rabbit-2.1.xsd
  http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-4.1.xsd
  http://www.springframework.org/schema/context
  https://www.springframework.org/schema/context/spring-context-4.1.xsd">

  <!--包扫描-->
  <!-- <context:component-scan base-package="com.itheima"></context:component-
  scan-->

  <!-- 公共部分 -->
  <!-- 创建连接类 连接安装好的 rabbitmq -->
  <bean id="connectionFactory"
  class="org.springframework.amqp.rabbit.connection.CachingConnectionFactory">
    <property name="virtualHost" value="testhosts"></property>
    <!-- username,访问RabbitMQ服务器的账户,默认是guest -->
    <property name="username" value="admin" />
    <!-- username,访问RabbitMQ服务器的密码,默认是guest -->
    <property name="password" value="admin" />
    <!-- host,RabbitMQ服务器地址,默认值"localhost" -->
    <property name="host" value="127.0.0.1" />
    <!-- port, RabbitMQ服务端口,默认值为5672 -->
    <property name="port" value="5672" />
  </bean>

  <rabbit:admin connection-factory="connectionFactory"/>

  <!--解析json , MQ会自动将数据转换成json字符串-->
  <bean id="jackson2JsonMessageConverter"
  class="org.springframework.amqp.support.converter.Jackson2JsonMessageConverter"/>

  <!--模版对象 用于发送消息-->
  <rabbit:template id="amqpTemplate" exchange="test-mq-exchange" connection-
  factory="connectionFactory" message-converter="jackson2JsonMessageConverter"
  />
```



connection生效。默认false。 -->

```
<rabbit:queue id="test_queue_key2" name="mail.send" durable="true" auto-
delete="false" exclusive="false" />
```

<!--交换机

定义消息队列,durable:是否持久化,如果想在RabbitMQ退出或崩溃的时候,不会失去所有的queue和消息,需要同时标志队列(queue)和交换机(exchange)是持久化的,即rabbit:queue标签和rabbit:direct-exchange中的durable=true,而消息(message)默认是持久化的可以看类org.springframework.amqp.core.MessageProperties中的属性public static final MessageDeliveryMode DEFAULT_DELIVERY_MODE = MessageDeliveryMode.PERSISTENT;exclusive: 仅创建者可以使用的私有队列,断开后自动删除;auto_delete: 当所有消费客户端连接断开后,是否自动删除队列

绑定队列,rabbitmq的exchangeType常用的三种模式: direct, fanout, topic三种,我们用direct模式,即rabbit:direct-exchange标签,Direct交换器很简单,如果是Direct类型,就会将消息中的RoutingKey与该Exchange关联的所有Binding中的BindingKey进行比较,如果相等,则发送到该Binding对应的Queue中。有一个需要注意的地方: 如果找不到指定的exchange,就会报错。但routing key找不到的话,不会报错,这条消息会直接丢失,所以此处要小心,auto-delete:自动删除,如果为Yes,则该交换机所有队列queue删除后,自动删除交换机,默认为false

-->

```
<rabbit:direct-exchange name="test-mq-exchange" durable="true" auto-
delete="false" id="test-mq-exchange">
  <rabbit:bindings>
    <!--绑定队列 队列名称为mail.send-->
    <rabbit:binding queue="test_queue_key2" key="mail.send"/>
  </rabbit:bindings>
</rabbit:direct-exchange>
</beans>
```

5.4.3 生产者代码

```
@Service
public class MQProducerImpl implements MQProducer {
    //注入模版对象
    @Autowired
    private AmqpTemplate amqpTemplate;

    public void sendDataToQueue(String queueKey, Object object) {
        //发送消息
        //convertAndSend: 将Java对象转换为消息发送到匹配Key的交换机中Exchange, 由于配置了JSON转换, 这里是将Java对象转换成JSON字符串的形式。
        amqpTemplate.convertAndSend(queueKey,object);
    }
}
```

5.4.4 改造注册用户

```
/**
 * 添加或修改
 * @return
 */
@RequestMapping(value = "/edit" ,name = "添加或者修改")
public String edit(User user ) throws Exception {
    //手动赋值数据 公司的id和公司的名称 根据登陆用户得到
```



```
String password = user.getPassword();
//对密码进行加密处理
String md5Pwd = Encrypt.md5(user.getPassword(), user.getEmail());
user.setPassword(md5Pwd);
//userService.save(user); //1.已经往数据库添加了数据
//2.发送邮件
//MailUtil.sendMsg(user.getEmail(), "恭喜您注册成功", "欢迎加入saas-export
大平台,我们将赠送您一百块的优惠券,您的密码是:"+ password);
//MQ发送邮件
Map<String, String> map = new HashMap<>();
map.put("to", user.getEmail());
map.put("subject", "恭喜您注册成功");
map.put("content", "欢迎加入saas-export大平台,我们将赠送您一百块的优惠券,您的密
码是:"+ password);
mqProducer.sendDataToQueue("mail.send", map);
}else{
    userService.update(user);
}
return "redirect:/system/user/list.do";
}
```

5.4.5 消费者配置

```
<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"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:schemaLocation="http://www.springframework.org/schema/rabbit
http://www.springframework.org/schema/rabbit/spring-rabbit-2.1.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.1.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context-4.1.xsd">

<!--连接工厂-->
<bean id="connectionFactory"
class="org.springframework.amqp.rabbit.connection.CachingConnectionFactory">
<!--<constructor-arg value="${mq.vhost}" />-->
<property name="virtualHost" value="testhosts"></property>
<!-- username,访问RabbitMQ服务器的账户,默认是guest -->
<property name="username" value="admin" />
<!-- username,访问RabbitMQ服务器的密码,默认是guest -->
<property name="password" value="admin" />
<!-- host,RabbitMQ服务器地址,默认值"localhost" -->
<property name="host" value="127.0.0.1" />
<!-- port, RabbitMQ服务端口,默认值为5672 -->
<property name="port" value="5672" />
</bean>
<rabbit:admin connection-factory="connectionFactory" />

<rabbit:queue name="mail.send" auto-declare="true" durable="true" />
<!-- 消费者部分 -->
<!-- 自定义接口类 -->
```



```
<bean id="jackson2JsonMessageConverter"
class="org.springframework.amqp.support.converter.Jackson2JsonMessageConverter"/
>

<!-- 配置监听acknowledge="manual"设置手动应答，它能够保证即使在一个worker处理消息的时
候用CTRL+C来杀掉这个worker，或者一个consumer挂了(channel关闭了、connection关闭了或者TCP
连接断了)，也不会丢失消息。因为RabbitMQ知道没发送ack确认消息导致这个消息没有被完全处理，将会对
这条消息做re-queue处理。如果此时有另一个consumer连接，消息会被重新发送至另一个consumer会一直
重发,直到消息处理成功,监听容器acknowledge="auto" concurrency="30"设置发送次数,最多发送30
次 -->

<rabbit:listener-container connection-factory="connectionFactory"
acknowledge="auto" concurrency="20">
    <rabbit:listener queues="mail.send" ref="testHandler" />
</rabbit:listener-container>
</beans>
```

5.4.5 消费者代码

```
//实现监听器接口
public class MyListener implements MessageListener{
    //定义转换json的接口
    private static final ObjectMapper MAPPER = new ObjectMapper();
    public void onMessage(Message message) {
        try {
            //msg就是rabbitmq传来的消息，需要的同学自己打印看一眼
            // 使用jackson解析
            JsonNode jsonData = MAPPER.readTree(message.getBody());
            String to = jsonData.get("to").asText();
            String subject = jsonData.get("subject").asText();
            String content = jsonData.get("content").asText();
            MailUtil.sendMsg(to , subject , content);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```