

RabbitMQ 文档

一 简介

MQ全称为Message Queue, 消息队列（MQ）是一种应用程序对应用程序的通信方法。应用程序通过读写出入队列的消息（针对应用程序的数据）来通信，而无需专用连接来链接它们。消息传递指的是程序之间通过在消息中发送数据进行通信，而不是通过直接调用彼此来通信，直接调用通常是用于诸如远程过程调用的技术。排队指的是应用程序通过 队列来通信。队列的使用除去了接收和发送应用程序同时执行的要求。其中较为成熟的MQ产品有IBM WEBSPPHERE MQ等等。

RabbitMQ是一个在AMQP基础上完成的，可复用的企业消息系统。他遵循Mozilla Public License开源协议。

AMQP，即Advanced Message Queuing Protocol,一个提供统一消息服务的应用层标准高级消息队列协议,是应用层协议的一个开放标准,为面向消息的中间件设计。基于此协议的客户端与消息中间件可传递消息，并不受客户端/中间件不同产品，不同的开发语言等条件的限制。Erlang中的实现有RabbitMQ等。

二 安装 RabbitMQ

CentOS 7 安装 RabbitMQ 3.7

安装Erlang

安装依赖

```
sudo yum install -y gcc gcc-c++ glibc-devel make ncurses-devel openssl-devel  
autoconf java-1.8.0-openjdk-devel git
```

创建yum源

参考：(<https://github.com/rabbitmq/erlang-rpm>)

```
sudo vi /etc/yum.repos.d/rabbitmq-erlang.repo
```

#添加内容

```
[rabbitmq-erlang]
name=rabbitmq-erlang
baseurl=https://dl.bintray.com/rabbitmq/rpm/erlang/20/el/7
gpgcheck=1
gpgkey=https://dl.bintray.com/rabbitmq/Keys/rabbitmq-release-signing-key.asc
repo_gpgcheck=0
enabled=1
```

安装

```
sudo yum install -y erlang
```

进入erlang命令行表示成功

```
erl
```

安装 socat

```
yum install -y socat
```

RabbitMQ 安装

官网下载地址: <https://www.rabbitmq.com/install-rpm.html>

```
sudo rpm -Uvh https://github.com/rabbitmq/rabbitmq-
server/releases/download/v3.7.4/rabbitmq-server-3.7.4-1.el7.noarch.rpm
```

如果遇到erlang已安装且版本正确, 但是RabbitMQ检测失败的情况 可以追加参数 `--nodeps` (不验证软件包依赖)

systemctl 操作 RabbitMQ服务

```
systemctl start rabbitmq-server
systemctl status rabbitmq-server
systemctl restart rabbitmq-server
```

设置为开机启动

```
systemctl enable rabbitmq-server
```

开启 允许远程访问(非必须)

```
vi /etc/rabbitmq/rabbitmq.config  
###添加一下内容  
[{rabbit, [{loopback_users, []}]}].
```

开启 web 端管理访问(非必须,如果要开启,需要先开启允许远程访问)

```
rabbitmq-plugins enable rabbitmq_management
```

安装消息延迟插件

```
cd /usr/lib/rabbitmq/lib/rabbitmq_server-3.6.6/plugins  
  
wget https://dl.bintray.com/rabbitmq/community-  
plugins/rabbitmq_delayed_message_exchange-0.0.1.ez  
  
rabbitmq-plugins enable rabbitmq_delayed_message_exchange
```

放行端口(可以直接关闭防火墙)

```
firewall-cmd --add-port=15672/tcp --permanent  
  
firewall-cmd --add-port=5672/tcp --permanent
```

三 添加用户

3.1 账号级别

1. 超级管理员administrator,可以登录控制台,查看所有信息,可以对用户和策略进行操作
2. 监控者monitoring,可以登录控制台,可以查看节点的相关信息,比如进程数,内存磁盘使用情况
3. 策略制定者policymaker ,可以登录控制台,制定策略,但是无法查看节点信息
4. 普通管理员 management 仅能登录控制台
5. 其他, 无法登录控制台,一般指的是提供者和消费者

3.2 添加账号

3.2.1 命令模式

```
rabbitmqctl add_user luke luke #添加账号 luke 密码是 luke  
  
rabbitmqctl set_user_tags luke administrator #设置 luke 为administrator级别
```

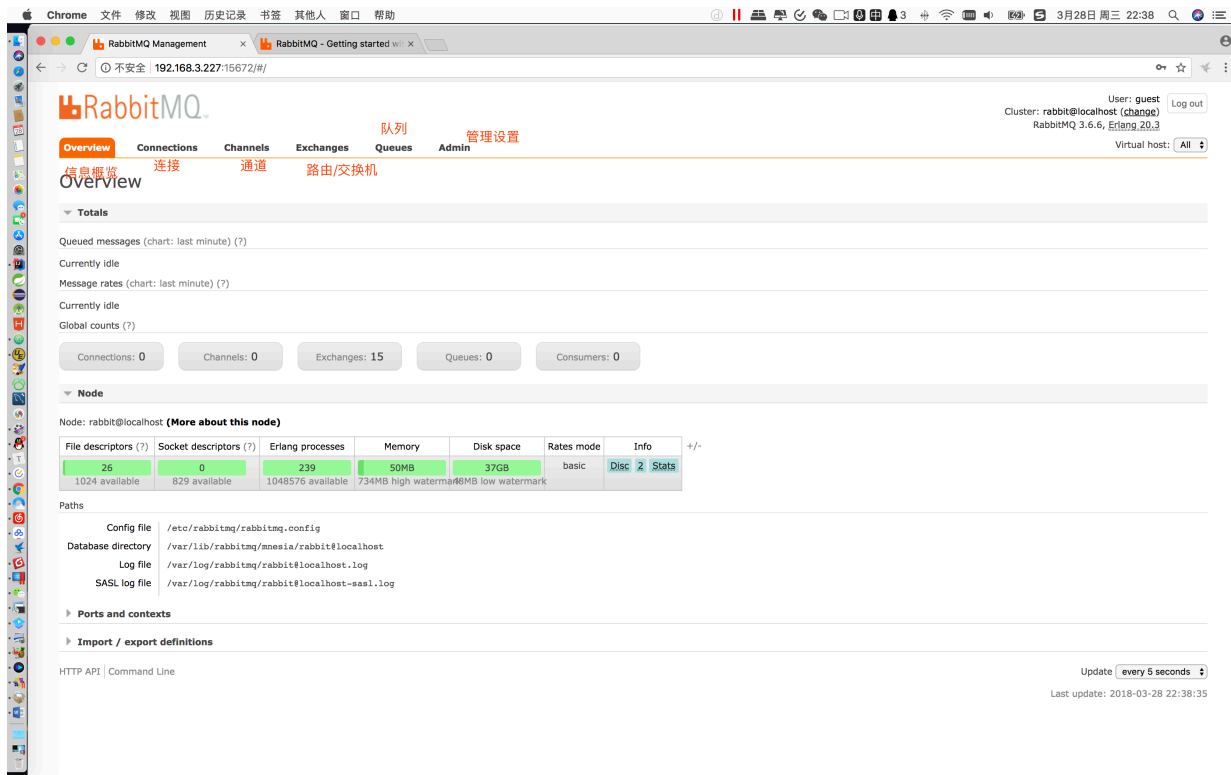
3.2.2 web 方式

可以通过 web 页面添加账号,此方式需要开启 web 访问

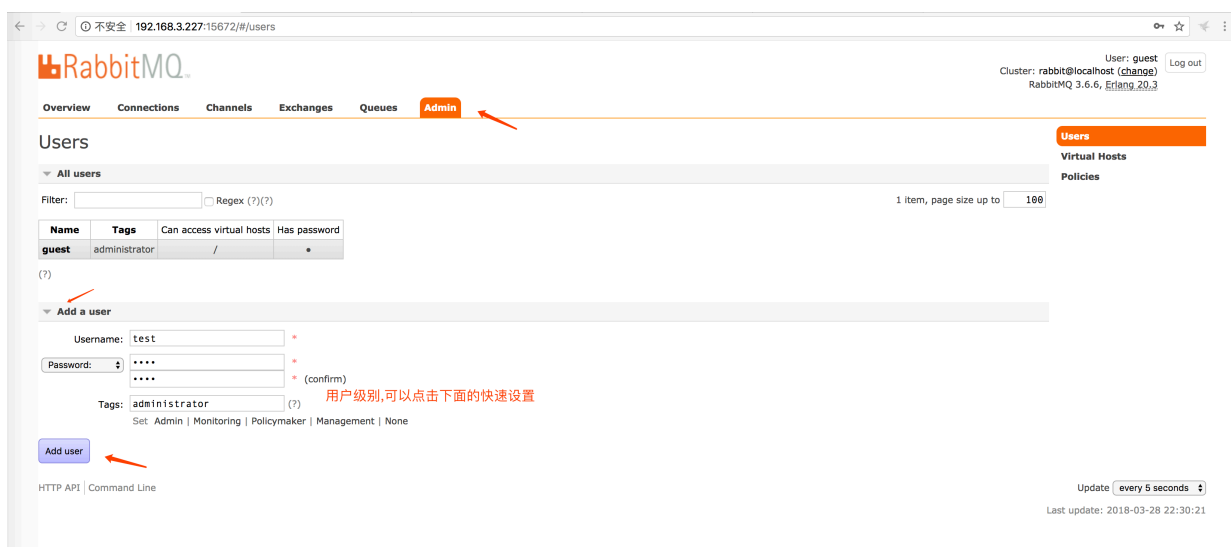
3.2.2.1 访问web

<http://192.168.3.227:15672/> 此处 ip 是本人 ip, 实际中请以实际 ip 为准

使用 guest guest 登录 guest 具有最高权限



3.2.2.2 添加用户



3.2.2.3 分配可以访问的虚拟主机

默认情况下没有任何可以访问的,我们可以添加一个主机(相当于添加一个数据库),然后分配权限

← → ↻ 不安全 192.168.3.227:15672/#/users

RabbitMQ User: guest Cluster: rabbit@localhost (change) RabbitMQ 3.6.6, Erlang 20.3 Log out

Overview Connections Channels Exchanges Queues **Admin**

Users

▼ All users Filter: ☐ Regex (?) (?) 2 items, page size up to 100

Name	Tags	Can access virtual hosts	Has password
guest	administrator	/	*
test	administrator	No access	*

(?)

没有可以访问的虚拟主机

▼ Add a user

Username: *

Password: * (confirm) *

Tags: (?)

Set: Admin | Monitoring | Policymaker | Management | None

Add user

HTTP API | Command Line

Update every 5 seconds Last update: 2018-03-28 22:31:50

Users
Virtual Hosts 管理虚拟主机
Policies

3.2.2.4 创建虚拟主机

RabbitMQ User: guest Cluster: rabbit@localhost (change) RabbitMQ 3.6.6, Erlang 20.3

Overview Connections Channels Exchanges Queues **Admin**

Virtual Hosts

▼ All virtual hosts Filter: ☐ Regex (?) (?) 1 item, page size up to 100

Overview		Messages			Network		Message rates		
Name	Users (?)	Ready	Unacked	Total	From client	To client	publish	deliver / get	+/-
/	guest	NaN	NaN	NaN					

▼ Add a new virtual host

Name: /test *

Add virtual host

HTTP API | Command Line

Update every 5 seconds Last update: 2018-03-28

Users
Virtual Hosts
Policies

3.2.2.5 给虚拟主机分配权限

RabbitMQ

Overview

Connections

Channels

Exchanges

Queues

Admin

Virtual Hosts

All virtual hosts

Filter: ☐ Regexp (?)

Overview		Messages			Network		Message rates	
Name	Users (?)	Ready	Unacked	Total	From client	To client	publish	deliver / get
/	guest	NaN	NaN	NaN				
/test	No users	NaN	NaN	NaN				

Add a new virtual host

Name:

Add virtual host

HTTP API | Command Line

3.2.2.6 给指定用户分配权限

RabbitMQ

Overview

Connections

Channels

Exchanges

Queues

Admin

Virtual Host: /test

Users

Virtual Hosts

Policies

Overview

Permissions

Queued messages (chart: last minute) (?)

Waiting for data...

Message rates (chart: last minute) (?)

Currently idle

Data rates (chart: last minute)

Waiting for data...

Details

Tracing enabled: ☐

Current permissions

User	Configure regexp	Write regexp	Read regexp	
test	.*	.*	.*	Clear

Set permission

User:

Configure regexp:

Write regexp:

Read regexp:

Set permission

四 消息

<http://www.rabbitmq.com/getstarted.html>

消息测试都在一个项目中,不同包下做测试

4.1消息模式种类

1 "Hello World!"

The simplest thing that does something



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

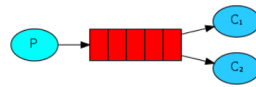
[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

2 Work queues

Distributing tasks among workers (the [competing consumers pattern](#))



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

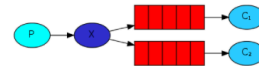
[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

3 Publish/Subscribe

Sending messages to many consumers at once



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

4 Routing

Receiving messages selectively



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

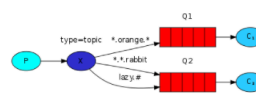
[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

5 Topics

Receiving messages based on a pattern (topics)



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

[Objective-C](#)

[Swift](#)

[Spring AMQP](#)

6 RPC

[Request/reply pattern](#)
example



[Python](#)

[Java](#)

[Ruby](#)

[PHP](#)

[C#](#)

[JavaScript](#)

[Go](#)

[Elixir](#)

[Spring AMQP](#)

4.2 pom&log4j.properties

```
<dependencies>
  <!-- https://mvnrepository.com/artifact/com.rabbitmq/amqp-client -->
  <!-- https://mvnrepository.com/artifact/com.rabbitmq/amqp-client -->
  <dependency>
    <groupId>com.rabbitmq</groupId>
```

```

        <artifactId>amqp-client</artifactId>
        <version>4.5.0</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-log4j12 -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.25</version>
    </dependency>

    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.3.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework.amqp/spring-
rabbit
整合 spring 时使用,amqp 只对 rabbitmq 做了支持
-->
    <dependency>
        <groupId>org.springframework.amqp</groupId>
        <artifactId>spring-rabbit</artifactId>
        <version>1.7.6.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>4.3.7.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>
</dependencies>

```

log4j.properties


```
log4j.rootLogger=DEBUG,A1
log4j.logger.com.taotao = DEBUG
log4j.logger.org.mybatis = DEBUG

log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss,SSS} [%t] [%c]-
[%p] %m%n
```

4.3 工具类ConnectionUtil

```
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("192.168.3.227");
        //端口
        factory.setPort(5672);
        //设置账号信息, 用户名、密码、vhost
        factory.setVirtualHost("/test");
        factory.setUsername("test");
        factory.setPassword("test");
        // 通过工程获取连接
        Connection connection = factory.newConnection();
        return connection;
    }

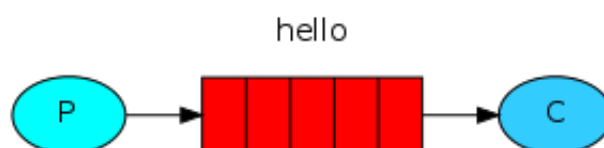
}
```

4.2 简单模式

<http://www.rabbitmq.com/tutorials/tutorial-one-java.html>

简单模式就是我们的生产者将消息发到队列,消费者从队列中取消息

一条消息对应一个消费者



4.2.1 生产者

```
public class Send {

    private final static String QUEUE_NAME = "test_queue";

    public static void main(String[] argv) throws Exception {
        // 获取到连接以及mq通道
        Connection connection = ConnectionUtil.getConnection();//相当于数据库中的创建连接
        // 从连接中创建通道
        Channel channel = connection.createChannel();//相当于数据库中的 statement

        // 声明（创建）队列,如果存在就不创建,不存在就创建
        //参数1 队列名,
        //参数2 durable: 是否持久化, 队列的声明默认是存放内存中的, 如果rabbitmq重启会丢失, 如果想重启之后还存在就要使队列持久化, 保存到Erlang自带的Mnesia数据库中, 当rabbitmq重启之后会读取该数据库
        //exclusive: 是否排外的, 有两个作用, 一: 当连接关闭时connection.close()该队列是否会自动删除; 二: 该队列是否是私有的private, 如果不是排外的, 可以使用两个消费者都访问同一个队列, 没有任何问题, 如果是排外的, 会对当前队列加锁, 其他通道channel是不能访问的, 如果强制访问会报异常: com.rabbitmq.client.ShutdownSignalException: channel error;
        protocol method: #method<channel.close>(reply-code=405, reply-text=RESOURCE_LOCKED - cannot obtain exclusive access to locked queue 'queue_name' in vhost '/', class-id=50, method-id=20)一般等于true的话用于一个队列只能有一个消费者来消费的场景
        //autoDelete: 是否自动删除, 当最后一个消费者断开连接之后队列是否自动被删除, 可以通过RabbitMQ Management, 查看某个队列的消费者数量, 当consumers = 0时队列就会自动删除
        //arguments: 参数

        channel.queueDeclare(QUEUE_NAME, false, false, false, null);//

        // 消息内容
        String message = "Hello World!";
        //参数1 交换机,此处无 参数2 发送到哪个队列 ,参数3 属性 参数4 内容
        channel.basicPublish("", QUEUE_NAME, null, message.getBytes());//将消息发送到数据库
        System.out.println(" 发送数据 '" + message + "'");

        //关闭通道和连接
        channel.close();
        connection.close();
    }
}
```

4.2.2 消费者

```
public class Recv {
```

```

private final static String QUEUE_NAME = "test_queue";

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,相当于我要声明我发送给谁,也就是手机号,如果队列不存在,则创建队列,如果存在则
    //不做事情

    //参数1 队列的名字
    //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会
    //丢失,持久化后会存放到数据库中,当服务器重启后会重新读取数据
    //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否
    //为私有的,如果私有了,只能由一个消费者能访问该队列

    channel.queueDeclare(QUEUE_NAME,false,false ,false,null);
    QueueingConsumer consumer = new QueueingConsumer(channel);
    //参数2 自动应答,当收到消息的时候自动告诉服务器我已经收到消息,这样的话 服务器就
    //不会一直给我们推送消息
    channel.basicConsume(QUEUE_NAME,true,consumer);//使用当前消费者处理指定队列
    //的消息, 相当于我们声明我的手机号是多少

    while (true) {

        QueueingConsumer.Delivery delivery = consumer.nextDelivery();//获取消
        //息的封装对象

        byte[] body = delivery.getBody();//传递的内容

        System.err.println("消费者收到了:====>  "+new String(body));
    }
}

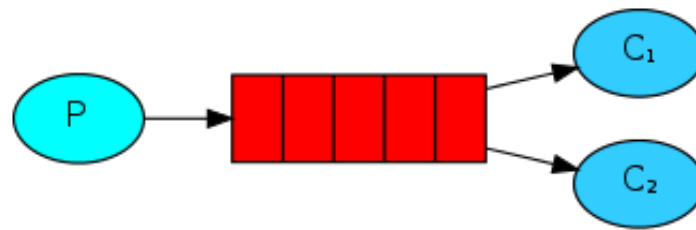
```

4.2.3 测试

运行 send, 和 recv 测试可以收发消息

4.3 work 模式

一条消息可以被多个消费者尝试接收,但是最终只能有一个消费者能获取



4.3.1 发送者

```
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++) { // 循环发送消息, 但是每条消息的时间间隔越来越长  
            // 消息内容  
            String message = "" + i;  
            channel.basicPublish("", QUEUE_NAME, null, message.getBytes());  
            System.out.println(" 发送消息 '" + message + "'");  
  
            Thread.sleep(i * 10); // 休眠  
        }  
  
        channel.close();  
        connection.close();  
    }  
}
```

4.3.2 消费者1

```
public class Recv {  
    private static final String QUEUENAME = "workqueue";  
  
    public static void main(String[] args) throws Exception {  
        Connection connection = ConnectionUtils.getConnection(); // 获取连接  
        Channel channel = connection.createChannel(); // 相当于我们连接数据库的时候的  
        statement\  
        // 声明队列, 相当于我要声明我发送给谁, 也就是手机号, 如果队列不存在, 则创建队列, 如果  
        存在则不做事情
```

```

        //参数1 队列的名字
        //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会丢失,持久化后会存放到数据库中,当服务器重启后会重新读取数据
        //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否为私有的,如果私有了,只能由一个消费者能访问该队列

        channel.queueDeclare(QUEUENAME,false,false ,false,null);
        //注释掉后可以获取多条消息,但是会一条一条处理
        channel.basicQos(1);//声明每次处理一个消息,只有处理完成并应答服务器之后才会处理下一条消息

        //创建消费者,并重写处理消息的方法,此方法为阻塞的,不需要后面 while 循环
        DefaultConsumer defaultConsumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
            AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者1收到的内容是:"+new String(body));

                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                //应答,告诉服务器我收到消息了
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        };
        // 监听队列, 手动返回完成,参数2手动确认模式
        channel.basicConsume(QUEUENAME, false, defaultConsumer);
    }
}

```

4.3.3 消费者2

```

public class Recv2 {
    private static final String QUEUENAME = "workqueue";

    public static void main(String[] args) throws Exception {

        Connection connection = ConnectionUtils.getConnection();//获取连接
        Channel channel = connection.createChannel();//相当于我们连接数据库的时候的statement\
        //声明队列,相当于我要声明我发送给谁,也就是手机号,如果队列不存在,则创建队列,如果存在则不做事情

        //参数1 队列的名字

```

//参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会丢失,持久化后会存放到数据库中,当服务器重启后会重新读取数据

//参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否为私有的,如果私有了,只能由一个消费者能访问该队列

```
channel.queueDeclare(QUEUENAME, false, false, false, null);
channel.basicQos(1); //声明每次处理一个消息
DefaultConsumer defaultConsumer = new DefaultConsumer(channel) {
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
        AMQP.BasicProperties properties, byte[] body) throws IOException {
        System.out.println("消费者2222收到的内容是:" + new String(body));

        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //应答
        channel.basicAck(envelope.getDeliveryTag(), false);
    }
};

channel.basicConsume(QUEUENAME, false, defaultConsumer);
}
```

4.3.4 测试

启动消费者1,消费者2, 发送者

在channel.basicQos(1);代码注释掉的情况下, 我们发现两个消费者获取到的消息数量是一致的, 会轮流从队列取消息

channel.basicQos(1);代码打开后,发现消费者1可以获取到更多数据,因为消费者的处理时间端, 处理快,所以可以获取到更多的消息

4.4 消息的确认模式

当我们发送消息后,服务端如何知道消息已经被消费

模式1:自动模式,不管消费者获取到消息后是否是成功处理消息,服务端都认为是成功的

模式2:手动模式,消费者获取到消息后,服务器会将消息标记为不可用,等待消费者反馈,如果不反馈,则一直标记为不可用

4.5 订阅模式

<http://www.rabbitmq.com/tutorials/tutorial-three-python.html>

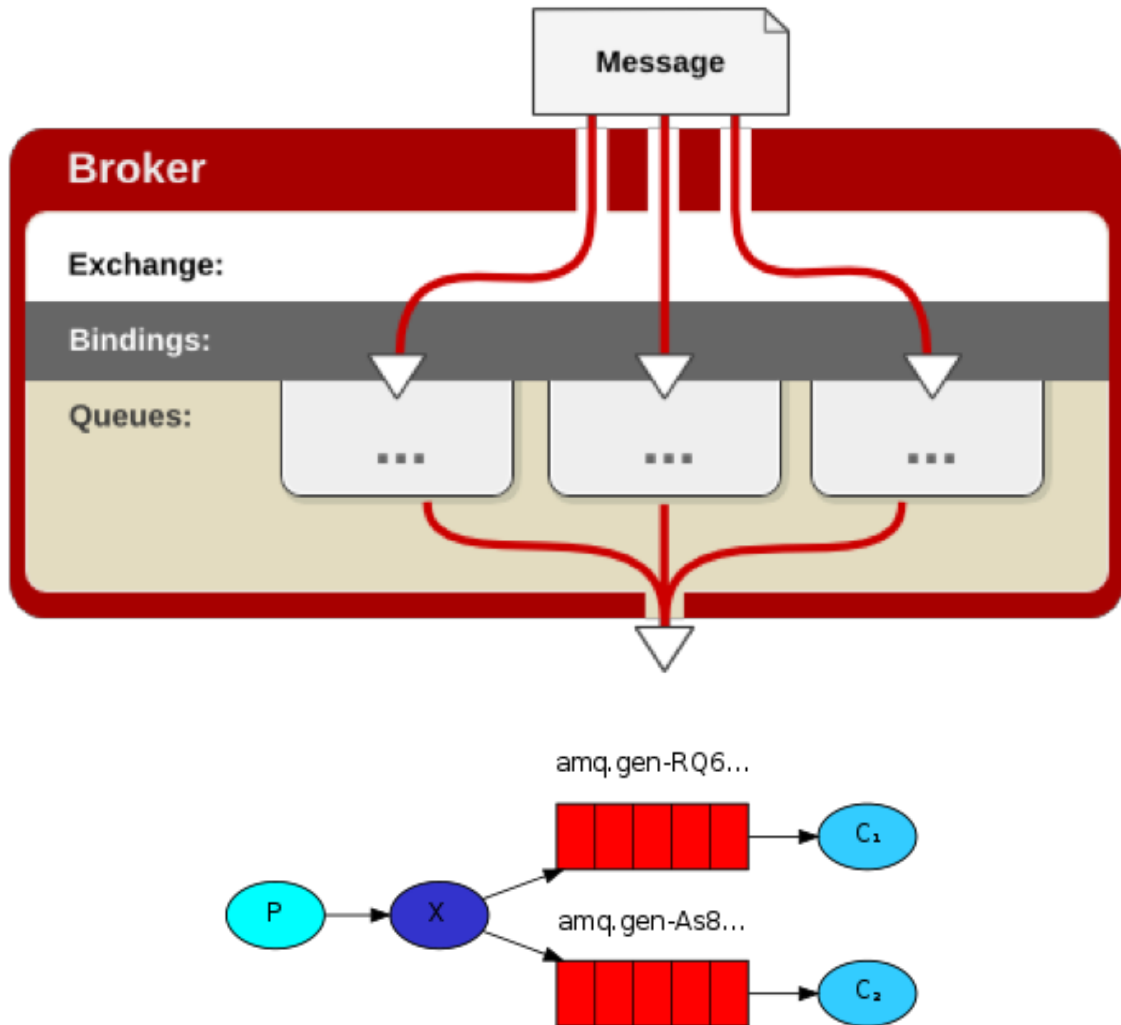
一条消息可以被多个消费者同时获取

生产者将消息发送到交换机

消费者将自己对应的队列注册到交换机

当发送消息后 所有注册的队列的消费者都可以收到消息

Fanout Exchange



4.5.1 生产者

```
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!";
//将消息发送到交换机,如果此时没有队列绑定,则消息会丢失,因为交换机没有存储消息的能力

channel.basicPublish(EXCHANGE_NAME, "", null, message.getBytes());
System.out.println(" 发送消息 '" + message + "'");

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

4.5.2 消费者1

```
public class Recv {

    private static final String EXCHANGENAME = "fanoutexchange222222";
    private static final String QUEUENAME = "subqueue1";

    public static void main(String[] args) throws Exception {
        Connection connection = ConnectionUtils.getConnection();//获取连接
        Channel channel = connection.createChannel();//相当于我们连接数据库的时候的statement\
        //声明队列,相当于我要声明我发送给谁,也就是手机号,如果队列不存在,则创建队列,如果存在则不做事情

        //参数1 队列的名字
        //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会丢失,持久化后会存放到数据库中,当服务器重启后会重新读取数据
        //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否为私有的,如果私有了,只能由一个消费者能访问该队列
        channel.exchangeDeclare(EXCHANGENAME, "fanout");//交换机必须先声明后使用,不管是谁声明的,队列必须要先存在

        channel.queueDeclare(QUEUENAME, false, false , false, null);

        channel.queueBind(QUEUENAME, EXCHANGENAME, "");//将制定的队列绑定到指定的交换机上面

        channel.basicQos(1);//声明每次处理一个消息,只有处理完成并应答服务器之后才会处理下一条消息

        DefaultConsumer defaultConsumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
            AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者1收到的内容是:"+new String(body));
            }
        }
    }
}
```



```

        try {
            Thread.sleep(10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //应答,告诉服务器我收到消息了
        channel.basicAck(envelope.getDeliveryTag(), false);
    }
};

channel.basicConsume(QUEUENAME, false, defaultConsumer);

}
}

```

4.5.3 消费者2

```

public class Recv2 {

    private static final String EXCHANGENAME = "fanoutexchange222222";
    private static final String QUEUENAME = "subqueue2";

    public static void main(String[] args) throws Exception {

        Connection connection = ConnectionUtils.getConnection();//获取连接
        Channel channel = connection.createChannel();//相当于我们连接数据库的时候的
        statement\
        //声明队列,相当于我要声明我发送给谁,也就是手机号,如果队列不存在,则创建队列,如果
        存在则不做事情

        //参数1 队列的名字
        //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会
        丢失,持久化后会存放到数据库中,当服务器重启后会重新读取数据
        //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否
        为私有的,如果私有了,只能由一个消费者能访问该队列
        channel.exchangeDeclare(EXCHANGENAME, "fanout");//交换机必须先声明后使用,不
        管是谁声明的,队列必须要先存在

        channel.queueDeclare(QUEUENAME, false, false , false, null);

        channel.queueBind(QUEUENAME, EXCHANGENAME, "");//将制定的队列绑定到指定的交
        换机上面
        channel.basicQos(1);//声明每次处理一个消息

        DefaultConsumer defaultConsumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
            AMQP.BasicProperties properties, byte[] body) throws IOException {

```

```

        System.out.println("消费者2222收到的内容是:"+new String(body));

        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //应答
        channel.basicAck(envelope.getDeliveryTag(), false);
    }
};

channel.basicConsume(QUEUENAME, false, defaultConsumer);
}
}

```

4.5.4 测试

启动消费者1,2 生产者测试

4.6 路由模式

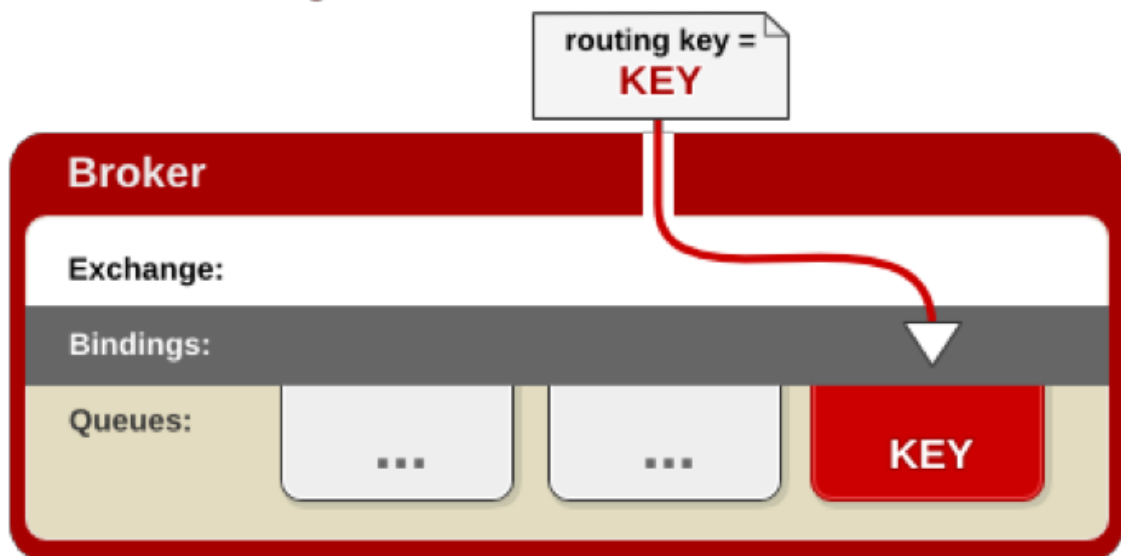
<http://www.rabbitmq.com/tutorials/tutorial-four-python.html>

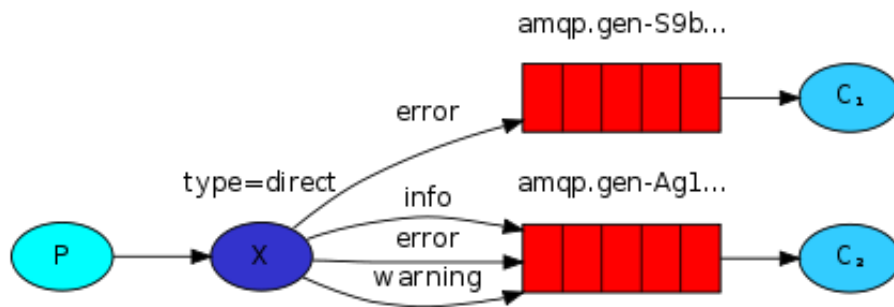
生产者将消息发送到了 type 为 direct 模式的交换机

消费者的队列在将自己绑定到路由的时候会给自己绑定一个 key

只有消费者发送对应 key 格式的消息时候 队列才会收到消息

Direct Exchange





4.6.1 生产者

```

public class Send {

    private final static String EXCHANGE_NAME = "test_exchange_direct";//路由名字

    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, type 是direct
        channel.exchangeDeclare(EXCHANGE_NAME, "direct");

        // 消息内容
        String message = "Hello World!";
        channel.basicPublish(EXCHANGE_NAME, "key2", null, message.getBytes()); //
        // 发送 key 为 key2的消息
        System.out.println(" 发送消息'" + message + "'");

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

```

4.6.2 消费者1

```

public class Recv {
    private static final String EXCHANGENAME = "directexchange";
    private static final String QUEUENAME = "directqueue1";

    public static void main(String[] args) throws Exception {
        Connection connection = ConnectionUtils.getConnection();//获取连接
        Channel channel = connection.createChannel();//相当于我们连接数据库的时候的
        //statement\
        //声明队列,相当于我要声明我发送给谁,也就是手机号,如果队列不存在,则创建队列,如果
        //存在则不做事情

        //参数1 队列的名字
    }
}

```

//参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会丢失,持久化后会存放到数据库中,当服务器重启后会重新读取数据

//参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否为私有的,如果私有了,只能由一个消费者能访问该队列

channel.exchangeDeclare(EXCHANGENAME, "direct");//交换机必须先声明后使用,不管是谁声明的,队列必须要先存在

```
channel.queueDeclare(QUEUENAME,false,false ,false,null);
```

// 绑定队列到交换机,绑定自己的关键字 key 为key,注意在绑定到指定路由(交换机)的时候,该路由必须存在,也就是我们必须先由发送者创建一个路由才可以

```
channel.queueBind(QUEUENAME, EXCHANGENAME, "abc");
```

//如果要绑定多个 key 多次执行即可

channel.queueBind(QUEUENAME, EXCHANGENAME, "abcd");//将制定的队列绑定到指定的交换机上面

channel.queueBind(QUEUENAME, EXCHANGENAME, "asdf");//将制定的队列绑定到指定的交换机上面

channel.basicQos(1);//声明每次处理一个消息,只有处理完成并应答服务器之后才会处理下一条消息

```
DefaultConsumer defaultConsumer=new DefaultConsumer(channel){
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
        System.out.println("消费者1收到的内容是:"+new String(body));

        try {
            Thread.sleep(10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //应答,告诉服务器我收到消息了
        channel.basicAck(envelope.getDeliveryTag(), false);
    }
};
// 监听队列, 手动返回完成
channel.basicConsume(QUEUENAME, false, defaultConsumer);
}
}
```

4.6.3 消费者2

```
public class Recv2 {

    private static final String EXCHANGENAME = "directexchange";
    private static final String QUEUENAME = "directqueue2";
```

```

public static void main(String[] args) throws Exception {
    Connection connection = ConnectionUtils.getConnection();//获取连接
    Channel channel = connection.createChannel();//相当于我们连接数据库的时候的
    statement\
        //声明队列,相当于我要声明我发送给谁,也就是手机号,如果队列不存在,则创建队列,如果
    存在则不做事情

    //参数1 队列的名字
    //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会
    丢失,持久化后会存放到数据库中,当服务器重启后会重新读取数据
    //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否
    为私有的,如果私有了,只能由一个消费者能访问该队列
    channel.exchangeDeclare(EXCHANGENAME, "direct");//交换机必须先声明后使用,不
    管是谁声明的,队列必须先存在

    channel.queueDeclare(QUEUENAME,false,false ,false,null);
    // 绑定队列到交换机,绑定自己的关键字 key 为key,注意在绑定到指定路由(交换机)的时
    候,该路由必须存在,也就是我们必须先由发送者创建一个路由才可以
    channel.queueBind(QUEUENAME, EXCHANGENAME, "abc");
    channel.basicQos(1);//声明每次处理一个消息,只有处理完成并应答服务器之后才会处理
    下一条消息

    DefaultConsumer defaultConsumer=new DefaultConsumer(channel){
        @Override
        public void handleDelivery(String consumerTag, Envelope envelope,
        AMQP.BasicProperties properties, byte[] body) throws IOException {
            System.out.println("消费者2222收到的内容是:"+new String(body));

            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            //应答,告诉服务器我收到消息了
            channel.basicAck(envelope.getDeliveryTag(), false);
        }
    };
    // 监听队列, 手动返回完成
    channel.basicConsume(QUEUENAME, false, defaultConsumer);
}
}

```

4.6.4 测试

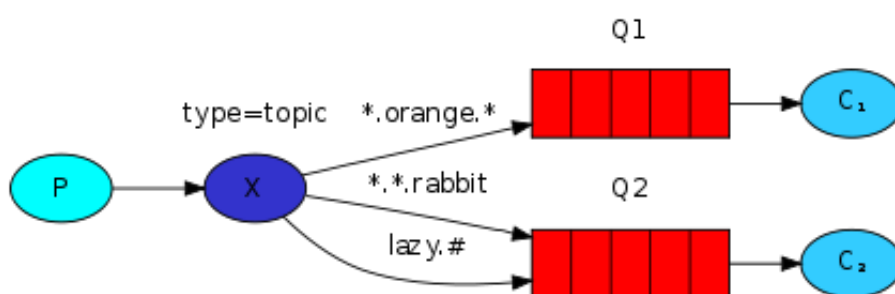
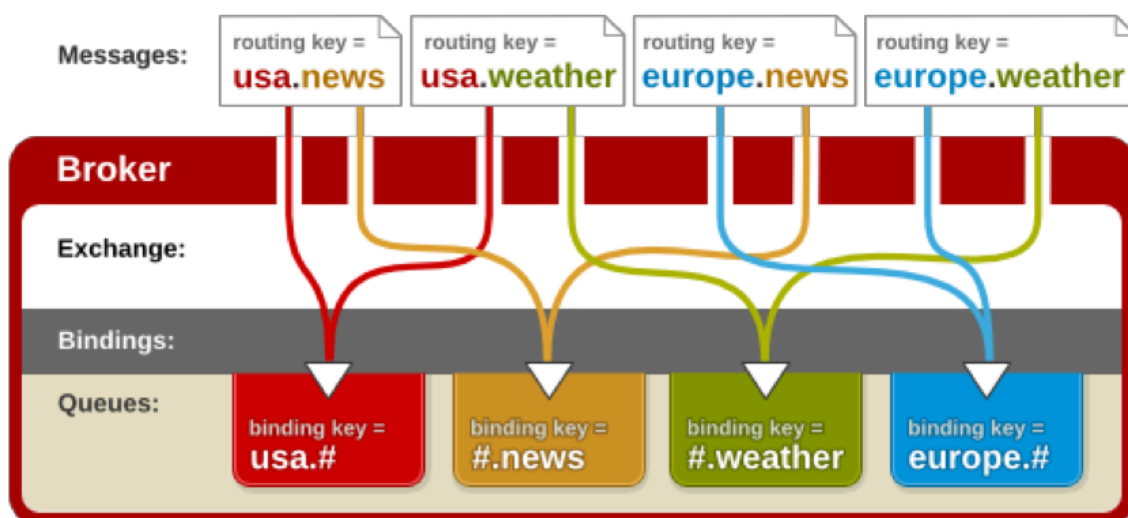
启动发送者创建路由,启动消费者1,消费者2

通过修改发送者代码中的 key 来多次执行测试,发现可以分别收到不同消息,如果监听了相同的 key 可以一起收到消息

4.7 通配符模式

<http://www.rabbitmq.com/tutorials/tutorial-five-python.html>

Topic Exchange



- *(star) can substitute for exactly one word.
- # (hash) can substitute for zero or more words.
- 将路由键和某模式进行匹配。此时队列需要绑定到一个模式上。符号“#”匹配一个或多个词，符号“*”匹配不多不少一个词。因此“audit.#”能够匹配到“audit.irs.corporate”，但是“audit.”只会匹配到“audit.irs”

4.7.1 生产者

```
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,声明为 topic 也就是通配符类型  
        channel.exchangeDeclare(EXCHANGE_NAME, "topic");  
    }  
}
```

```

// 消息内容
String message = "Hello World!";
//发送 key.1 数据,凡是能匹配到这个关键词的都会收到
channel.basicPublish(EXCHANGE_NAME, "key.1", null, message.getBytes());
System.out.println(" 发送消息 '" + message + "'");

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

```

4.7.2消费者1

```

public class Recv {
    private static final String EXCHANGENAME = "topicexchange";
    private static final String QUEUENAME = "topicqueue1";

    public static void main(String[] args) throws Exception {
        Connection connection = ConnectionUtils.getConnection();//获取连接
        Channel channel = connection.createChannel();//相当于我们连接数据库的时候的
        statement\
        //声明队列,相当于我要声明我发送给谁,也就是手机号,如果队列不存在,则创建队列,如果
        存在则不做事情

        //参数1 队列的名字
        //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会
        丢失,持久化后会存放数据库,当服务器重启后会重新读取数据
        //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否
        为私有的,如果私有了,只能由一个消费者能访问该队列
        channel.exchangeDeclare(EXCHANGENAME, "topic");//交换机必须先声明后使用,不管
        是谁声明的,队列必须先存在

        channel.queueDeclare(QUEUENAME, false, false , false, null);
        channel.queueBind(QUEUENAME, EXCHANGENAME, "key.*");
        channel.queueBind(QUEUENAME, EXCHANGENAME, "abc.#");//将制定的队列绑定到指定
        的交换机上面
        channel.queueBind(QUEUENAME, EXCHANGENAME, "*.aaa.#");//将制定的队列绑定到
        指定的交换机上面
        channel.basicQos(1);//声明每次处理一个消息,只有处理完成并应答服务器之后才会处理
        下一条消息

        DefaultConsumer defaultConsumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
            AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者1收到的内容是:"+new String(body));
            }
        }
    }
}

```

```

        try {
            Thread.sleep(10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //应答,告诉服务器我收到消息了
        channel.basicAck(envelope.getDeliveryTag(), false);
    }
};

channel.basicConsume(QUEUENAME, false, defaultConsumer);

}
}

```

4.7.3 消费者2

```

public class Recv2 {

    Connection connection = ConnectionUtils.getConnection();//获取连接
    Channel channel = connection.createChannel();//相当于我们连接数据库的时候的
    statement\
        //声明队列,相当于我要声明我发送给谁,也就是手机号,如果队列不存在,则创建队列,如果
    存在则不做事情

        //参数1 队列的名字
        //参数2 是否持久化,队列的消息默认是放在内存中, 如果rabbitmq 重启或者退出,消息会
    丢失,持久化后会存放到数据库中,当服务器重启后会重新读取数据
        //参数3 是否排外,两个作用: 一 当连接关闭的时候 是否会自动删除 ,二 声明该队列是否
    为私有的,如果私有了,只能由一个消费者能访问该队列
        channel.exchangeDeclare(EXCHANGENAME, "topic");//交换机必须先声明后使用,不管
    是谁声明的,队列必须要先存在

        channel.queueDeclare(QUEUENAME, false, false , false, null);

        channel.queueBind(QUEUENAME, EXCHANGENAME, "key.*");//将制定的队列绑定到指定
    的交换机上面
        channel.basicQos(1);//声明每次处理一个消息

        DefaultConsumer defaultConsumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
            AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("消费者2222收到的内容是:"+new String(body));

                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {

```



```

        e.printStackTrace();
    }
    //应答
    channel.basicAck(envelope.getDeliveryTag(), false);
}
};

channel.basicConsume(QUEUENAME, false, defaultConsumer);
}
}

```

4.7.4 测试

通过发送不同的 key 的数据测试发现消费者可能会在不同情况下分别收到数据

五 整合 spring

spring 对 amqp 做了支持,但是当前只实现了 rabbitmq

5.1 spring 自动模式

5.1.1 spring 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"
    xsi:schemaLocation="http://www.springframework.org/schema/rabbit
        http://www.springframework.org/schema/rabbit/spring-rabbit-1.7.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd">

    <!-- 定义RabbitMQ的连接工厂 -->
    <rabbit:connection-factory id="connectionFactory"
        host="192.168.3.227" port="5672" username="test" password="test"
        virtual-host="/test" />

    <!-- 定义Rabbit模板, 指定连接工厂以及定义exchange
        如果要消息发送到队列而不是交换机,则声明queue="" 而不是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="myQueue" auto-declare="true"/>

```

```

<!-- 定义交换器，自动声明 -->
<rabbit:fanout-exchange name="fanoutExchange" auto-declare="true" >
    <rabbit:bindings>
        <!--将下列队列绑定到当前交换机-->
        <rabbit:binding queue="myQueue"/>
    </rabbit:bindings>
</rabbit:fanout-exchange>
<!--
通配符模式
-->
<!--

<rabbit:topic-exchange name="myExchange">
    <rabbit:bindings>
        <rabbit:binding queue="myQueue" pattern="foo.*" />
    </rabbit:bindings>
</rabbit:topic-exchange>

-->

<!--路由设置 将队列绑定，属于direct类型
<rabbit:direct-exchange id="directExchange"
                        name="directExchange" durable="true" auto-
delete="false">
    <rabbit:bindings>
        <rabbit:binding queue="myQueue"
key="{rabbitmq.system.out.log.error.mail}" />
    </rabbit:bindings>
</rabbit:direct-exchange>
-->

<!-- 队列监听
acknowledged = "manual" 属性为手动应答
-->
<rabbit:listener-container connection-factory="connectionFactory">
    <!--指定对应队列myQueue的监听为 foo 中的 listen 方法-->
    <rabbit:listener ref="foo" method="listen" queue-names="myQueue" />
</rabbit:listener-container>

<bean id="foo" class="com.qianfeng.rabbitmq.spring.Foo" />

</beans>

```

5.1.2 接收者

```

/**
 * 消费者类,任意类都可以
 */
public class Foo {

    //具体执行业务的方法
    public void listen(String foo) {
        System.out.println("消费者: " + foo);
    }
}

```

5.1.3 测试类

```

public class TestMain {
    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, world!");
        Thread.sleep(1000); // 休眠1秒
        ctx.destroy(); //容器销毁
    }
}

```

5.1.4 启动测试

5.2 spring 手动模式

5.2.1 配置文件

主要介绍template 和 listener 的不同,其他同上

```

<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-1.7.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.3.xsd

```

```

http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd">

<context:component-scan base-package="com.qianfeng.rabbitmq"/>
    <bean id="jsonMessageConverter"
class="org.springframework.amqp.support.converter.Jackson2JsonMessageConverter" />

    <!-- 定义RabbitMQ的连接工厂 -->
    <rabbit:connection-factory id="connectionFactory"
                                host="192.168.3.227" port="5672" username="test"
password="test"
                                virtual-host="/test" />

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

    <!-- 给模板指定转换器 --><!-- mandatory必须设置true,return callback才生效 -->
    <rabbit:template id="amqpTemplate"    connection-factory="connectionFactory"
                                confirm-callback="confirmCallBackListener"
                                return-callback="returnCallBackListener"
                                mandatory="true"
    />

    <rabbit:queue name="CONFIRM_TEST" />

    <rabbit:direct-exchange name="DIRECT_EX" id="DIRECT_EX" >
        <rabbit:bindings>
            <rabbit:binding queue="CONFIRM_TEST" />
        </rabbit:bindings>
    </rabbit:direct-exchange>

    <!-- 配置consumer, 监听的类和queue的对应关系 -->
    <rabbit:listener-container
        connection-factory="connectionFactory" acknowledge="manual" >
        <rabbit:listener queues="CONFIRM_TEST" ref="receiveConfirmTestListener" />
    </rabbit:listener-container>
</beans>

```

5.2.2 消费者

```

/**
 * Created by jackiechan on 2018/3/29/上午12:51
 */
@Service("receiveConfirmTestListener")
public class ReceiveConfirmTestListener implements ChannelAwareMessageListener {
    @Override
    public void onMessage(Message message, Channel channel) throws Exception {

```

```

        try{
            System.err.println("消费者收到消息--
:"+message.getMessageProperties()+" "+new String(message.getBody()));
            channel.basicAck(message.getMessageProperties().getDeliveryTag(),
false); //手动应答
        }catch(Exception e){
            e.printStackTrace();//TODO 业务处理
            channel.basicNack(message.getMessageProperties().getDeliveryTag(),
false,false);
        }
    }
}

```

5.2.3 确认后回调

```

/**
 * Created by jackiechan on 2018/3/29/上午12:51
 */
@Service("confirmCallbackListener")
public class ConfirmCallbackListener implements RabbitTemplate.ConfirmCallback {
    @Override
    public void confirm(CorrelationData correlationData, boolean ack, String
cause) {
        System.err.println("确认--
:correlationData:"+correlationData+",ack:"+ack+",cause:"+cause);
    }
}

```

5.2.3 失败回滚

```

/**
 * Created by jackiechan on 2018/3/29/上午12:51
 */
@Service("returnCallbackListener")
public class ReturnCallbackListener implements RabbitTemplate.ReturnCallback {
    @Override
    public void returnedMessage(Message message, int replyCode, String replyText,
String exchange, String routingKey) {
        System.err.println("失败--message:"+new
String(message.getBody())+",replyCode:"+replyCode+",replyText:"+replyText+",exchan
ge:"+exchange+",routingKey:"+routingKey);
    }
}

```

5.2.3 生产者

```

@Service("publishService")
public class PublishService {
    @Autowired
    private AmqpTemplate amqpTemplate;

    public void send(String exchange, String routingKey, Object message) {
        amqpTemplate.convertAndSend(exchange, routingKey, message);
    }
}

```

5.2.4 消费者

```

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath:spring/application-context.xml"})

public class TestConfirm {
    @Autowired
    private PublishService publishService;

    private static String exChange = "DIRECT_EX";

    @Test
    public void test1() throws InterruptedException{
        String message = "currentTime:"+System.currentTimeMillis();
        System.out.println("test1---message:"+message);
        //exchange,queue 都正确,confirm被回调, ack=true
        publishService.send(exChange,"CONFIRM_TEST",message);
        Thread.sleep(1000);
    }

    @Test
    public void test2() throws InterruptedException{
        String message = "currentTime:"+System.currentTimeMillis();
        System.out.println("test2---message:"+message);
        //exchange 错误,queue 正确,confirm被回调, ack=false
        publishService.send(exChange+"NO","CONFIRM_TEST",message);
        Thread.sleep(1000);
    }

    @Test
    public void test3() throws InterruptedException{
        String message = "currentTime:"+System.currentTimeMillis();
        System.out.println("test3---message:"+message);
        //exchange 正确,queue 错误 ,confirm被回调, ack=true; return被回调
        replyText:NO_ROUTE
        publishService.send(exChange,"",message);
        //        Thread.sleep(1000);
    }
}

```

```

    }

    @Test
    public void test4() throws InterruptedException{
        String message = "currentTime:"+System.currentTimeMillis();
        System.out.println("test4---message:"+message);
        //exchange 错误,queue 错误,confirm被回调, ack=false
        publishService.send(exChange+"NO", "CONFIRM_TEST",message);
        Thread.sleep(1000);
    }
}

```

5.2.5 启动测试

六 整合 springboot

6.1 pom.xml

```

<?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.qianfeng</groupId>
    <artifactId>rabbitmqdemo</artifactId>
    <version>1.0</version>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.6.RELEASE</version>
    </parent>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-
boot-starter-amqp -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-amqp</artifactId>
        </dependency>

    </dependencies>

```

```
</project>
```

6.2 yml 配置

```
spring:
  rabbitmq:
    host: rabbitmq1.qfjava.cn
    port: 5672
    username: gp02
    password: gp02
    virtual-host: /gp02
```

6.3 sender 发送者

```
@Component
public class Sender {
    @Autowired
    private AmqpTemplate amqpTemplate;

    public void send(String message) {
        amqpTemplate.convertAndSend("dongwei", message);
    }

    public void sendtoExchange(String message) {
        amqpTemplate.convertAndSend("fanoutgp02", "", message);
    }

    public void sendtoDirectExchange(String message) {
        amqpTemplate.convertAndSend("directgp02", "abc", message);
    }
}
```

6.4 消费者的监听器

springboot 通过 listener 来监听消息,不需要单独写连接等


```

@Component
@RabbitListener(queues = "wxpayqueue")//标记当前类是用来处理来自于wxpayqueue这个队列
的消息的
public class Messagelistener {
    @RabbitHandler//标记当前方法是用来处理消息的

    public void aaaaa(String message) {
        System.out.println("俊伟收到了文龙通过学智在东伟这个交换机上面发送的内容：
=>"+message);
    }
}

```

6.5 starter&config

```

@SpringBootApplication
public class StartApp {

    @Bean//创建了一个队列,队列在rabbitmq中的名字是dongwei,在spring中的名字叫xuezhi
    public Queue xuezhi() {
        Queue queue = new Queue("wxpayqueue");
        return queue;
    }

    @Bean
    public ConnectionFactory connectionFactory() {
        CachingConnectionFactory cachingConnectionFactory = new
CachingConnectionFactory();
        cachingConnectionFactory.setHost("rabbitmq1.qfjava.cn");
        cachingConnectionFactory.setPort(5672);
        cachingConnectionFactory.setUsername("gp02");
        cachingConnectionFactory.setPassword("gp02");
        cachingConnectionFactory.setVirtualHost("/gp02");
        return cachingConnectionFactory;
    }

    /**
     * 声明交换机,fanout 类型
     * @return
     */
    @Bean
    public FanoutExchange fanoutExchange() {
        FanoutExchange fanoutExchange = new FanoutExchange("wxpayfanout");
        return fanoutExchange;
    }
}

```

```

    /**
     * 将队列和交换机绑定
     * 此处使用的是一个队列,如果有多个队列,请注意方法队列的参数名字区分
     * @return
     */
    @Bean
    public Binding bindingFanoutExchange(Queue queue, FanoutExchange
fanoutExchange) {
        return BindingBuilder.bind(queue).to(fanoutExchange);
    }

    /**
     * 声明交换机,direct 类型
     * @return
     */
    @Bean
    public DirectExchange directExchange() {
        DirectExchange directExchange = new DirectExchange("directgp02");
        return directExchange;
    }

    /**
     * 将队列和交换机绑定
     * 此处使用的是一个队列,如果有多个队列,请注意方法队列的参数名字区分
     * @return
     */
    @Bean
    public Binding bindingDirectExchange(Queue queue, DirectExchange
directExchange) {
        return BindingBuilder.bind(queue).to(directExchange).with("abc");
    }
}

```

6.6 测试

```

/**
 * Created by jackiechan on 18-12-26/下午2:44
 *
 * @Author jackiechan
 */
public class TestMain {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext configApplicationContext = new
AnnotationConfigApplicationContext(StartApp.class);
        Sender sender = configApplicationContext.getBean(Sender.class);
    }
}

```

```

        sender.send("文龙对俊伟说:在有生的瞬间能遇到你,竟花光所有运气");//哎哟喂
    }

    /**
     * 测试消费者
     * @throws InterruptedException
     */
    @Test
    public void test1() throws InterruptedException {
        //因为消费者是springboot创建的,所有只需要加载spring的容器就可以了,会自动创建对象,自动去链接服务器
        AnnotationConfigApplicationContext configApplicationContext = new
AnnotationConfigApplicationContext(StartApp.class);
        Thread.sleep(1002000);
    }

    @Test
    public void test2() throws InterruptedException {
        AnnotationConfigApplicationContext configApplicationContext = new
AnnotationConfigApplicationContext(StartApp.class);
        Sender sender = configApplicationContext.getBean(Sender.class);
        sender.sendtoExchange("dfgkjldhjkghsghjkl dsfghjdshjkl");//哎哟喂
        Thread.sleep(1002000);
    }

    @Test
    public void test3() throws InterruptedException {
        AnnotationConfigApplicationContext configApplicationContext = new
AnnotationConfigApplicationContext(StartApp.class);
        Sender sender = configApplicationContext.getBean(Sender.class);
        sender.sendtoDirectExchange("1111111111111111");//哎哟喂
        Thread.sleep(1002000);
    }
}

```