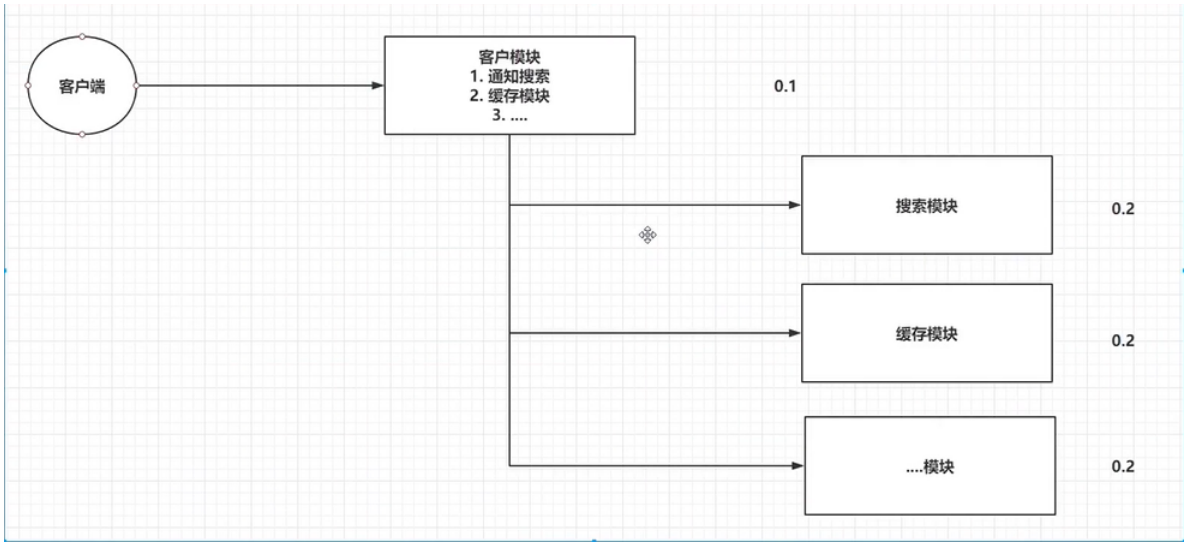
RabbitMQ

一、RabbitMQ介绍

1.1 引言

> 1、模块之间的耦合度过高，导致一个模块宕机后，全部功能不能用了。
>
> 2、同步通讯的成本问题。



1.2RabbitMQ的介绍

> 市面上比较火爆的几款MQ：
>
> ActiveMQ、RocketMQ、Kafka、RabbitMQ
>
> 1、语言的支持：ActiveMQ、RocketMQ只支持Java语言，Kafka可以支持多门语言，RabbitMQ支持多门语言
>
> 2、效率方面：ActiveMQ、RocketMQ、Kafka效率是毫秒级别的，RabbitMQ是微秒级别的。
>
> 3、消息丢失，消息重复问题：RabbitMQ针对消息的持久化，和重复问题都有比较成熟的解决方案。
>
> 4、学习成本：RabbitMQ是非常简单的中间件。
>
> RabbitMQ是由Rabbit公司去研发和维护的，最终在Pivotal。
>
> Rabbit严格的遵守AMOP协议，高级消息队列协议，帮助我们在进程之间传递异步消息。

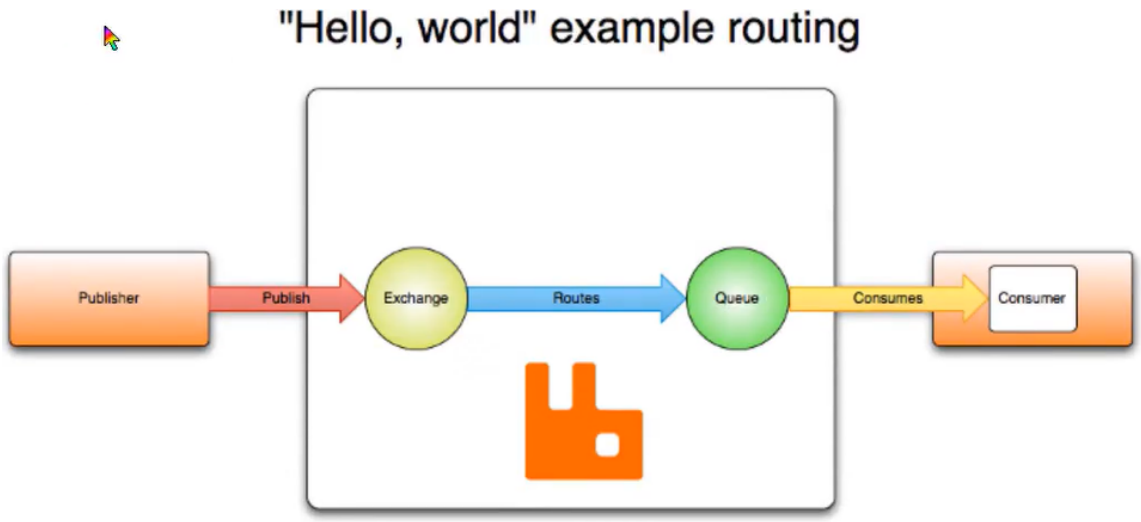
二、RabbitMQ安装

```
version:"3.1"
services:
  rabbitmq:
    image: daocloud.io/library/rabbitmq:management
    restart: always
    container_name: rabbitmq
    ports:
      - 5672:5672
      - 15672:15672
    volumes:
      - ./data:/var/lib/rabbitmq
```

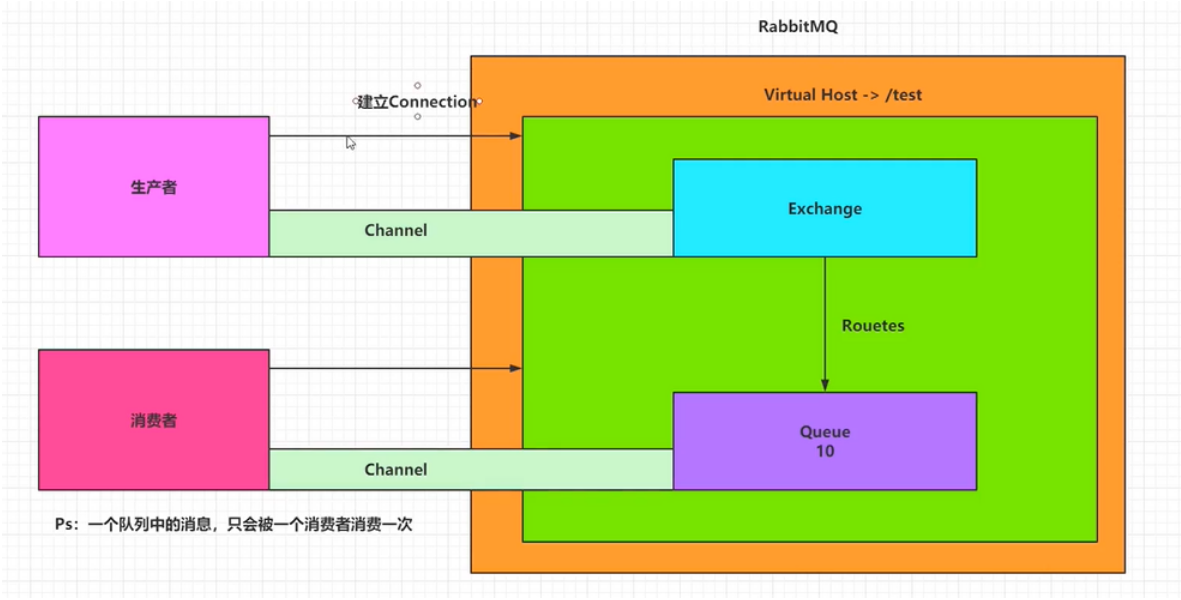
三、RabbitMQ架构

3.1 官方的简单架构图

> 1、publisher - 生产者：发布消息到rabbitMQ中的Exchange
>
> 2、consumer - 消费者：监听RabbitMQ中的Qunue中的消息
>
> 3、Exchange - 交换机：和生产者建立连接并接收生产者的消息
>
> 4、Qunue - 队列：Exchange会将消息分发到指定的Queue，Qunue和消费者进行交互
>
> 5、Routes - 路由：交换机以什么样的策略将消息发布到Qunue

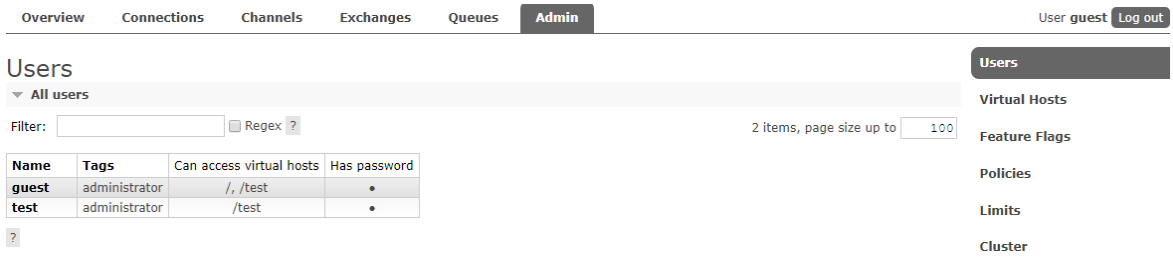


3.2 RabbitMQ完整架构图



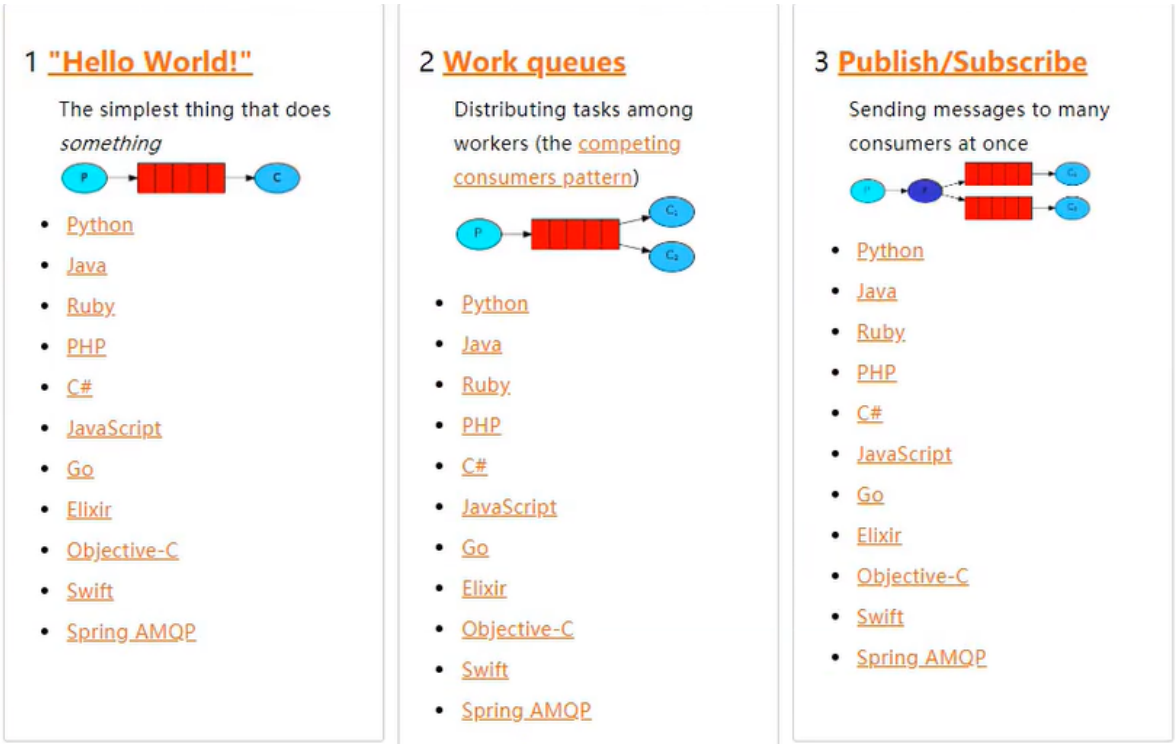
3.3 查看图形化界面并创建一个Virtual Host

> 创建一个全新的用户和全新的Virtual Host，并且将test用户设置上可以/test的Virtual Host权限



四、RabbitMQ的使用

4.1RabbitMQ的通讯方式（总共7种，RPC不讲）



4.2 Java连接RabbitMQ

> 1、创建maven项目

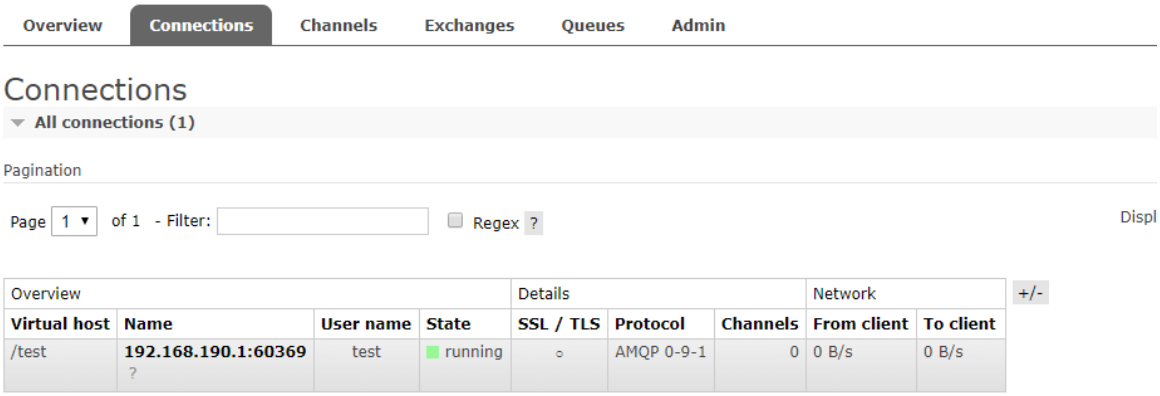
> 2、导入依赖

```
<dependencies>
        <!-- https://mvnrepository.com/artifact/com.rabbitmq/amqp-client -->
        <dependency>
            <groupId>com.rabbitmq</groupId>
            <artifactId>amqp-client</artifactId>
            <version>5.6.0</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>

    </dependencies>
```

> 3、创建工具类连接RabbitMQ

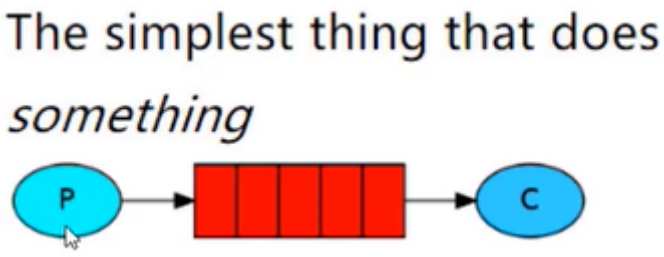
```
package com.lj.config;

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

import java.io.IOException;
import java.util.concurrent.TimeoutException;

public class RabbitMQClient {
    public static Connection getConnection(){

        //创建Connection工厂
```

```
        ConnectionFactory factory=new ConnectionFactory();
        factory.setHost("192.168.190.130");
        factory.setPort(5672);
        factory.setUsername("test");
        factory.setPassword("test");
        factory.setVirtualHost("/test");

        //创建Connection
        Connection conn= null;
        try {
            conn = factory.newConnection();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (TimeoutException e) {
            e.printStackTrace();
        }

        //返回
        return conn;
    }
}
```



4.3 Hello-World

> 一个生产者，一个默认的交换机，一个队列，一个消费者



> 1、创建生产者，创建一个channel， 发布消息到Exchange，指定路由规则

```
public void publish() throws Exception {
        //1.获取Connection
        Connection connection = RabbitMQClient.getConnection();

        //2.创建Channel
        Channel channel = connection.createChannel();
```

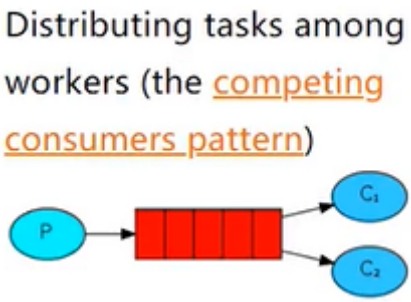
```
        //3. 发布消息到exchange，同时指定路由的规则
        String msg="Hello-World";
        //参数1：指定exchange，使用""。
        //参数2：指定路由的规则，使用具体的队列名称。
        //参数3：指定传递的消息所携带的properties,使用null
        //参数4：指定发布的具体消息，byte[]类型
        channel.basicPublish("","HelloWorld",null,msg.getBytes());
        //Ps: exchange是不会帮你将消息持久化到本地的，Queue才会帮你持久化消息
        System.out.println("生产者发布消息成功！！！");
        //4.释放资源
        channel.close();
        connection.close();

    }
```

> 2、创建消费者，创建一个channel,创建一个队列，并且去消费当前队列

```
public void Consume() throws Exception {
        //1. 获取连接对象
        Connection connection = RabbitMQClient.getConnection();

        //2. 创建channel
        Channel channel = connection.createChannel();


        //3. 声明队列-HelloWorld
        //参数1：queue - 指定队列的名称
        //参数2：durable - 当前队列是否持久化(true)
        //参数3：exclusive - 是否排外（conn.close() - 当前队列会被自动删除，当前队列只能被一个消费者消费）
        //参数4：autoDelete - 如果这个队列没有消费者在消费，队列自动删除
        //参数5：arguments - 指定当前队列的其他信息
        channel.queueDeclare("HelloWorld",true,false,false,null);

        //4. 开启监听Queue
        DefaultConsumer consumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {
                System.out.println("接收到消息："+new String(body,"UTF-8"));
            }
        };

        //参数1：queue - 指定消费哪个队列
        //参数2：autoAck - 指定是否自动ACK(true,接收到消息后，会立即告诉RabbitMQ)
        //参数3：consumer - 指定消费回调
        channel.basicConsume("HelloWorld",true,consumer);
        System.out.println("消费者开始监听队列！！！");
        //System.in.read()
        System.in.read();
        //5. 释放资源
        channel.close();
        connection.close();
    }
```

4.4 Work

> 一个生产者，一个默认的交换机，一个队列，两个消费者

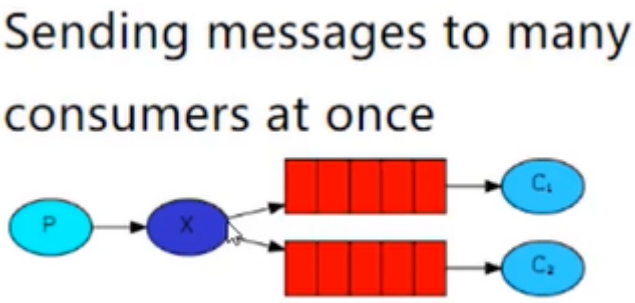


> 只需要在消费者端，添加Qos能力以及更改为手动ack即可让消费者，根据自己的能力去消费指定的消息，而不是默认情况下由RabbiitMQ平均分配了。

```
//1 指定当前消费者，一次消费多少个消息
        channel.basicQos(1);
//2. 开启监听Queue
        DefaultConsumer consumer=new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("消费者1号接收到消息："+new String(body,"UTF-
8"));

                //手动ack
                channel.basicAck(envelope.getDeliveryTag(),false);
            }
        };


//3. 指定手动ack
channel.basicConsume("Work",false,consumer);
```

4.5 Publish/Subscribe

> 一个生产者，一个交换机，两个队列，两个消费者



> 声明一个Fanout类型的exchange,并且将exchange和queue绑定在一起，绑定的方式就是直接绑定

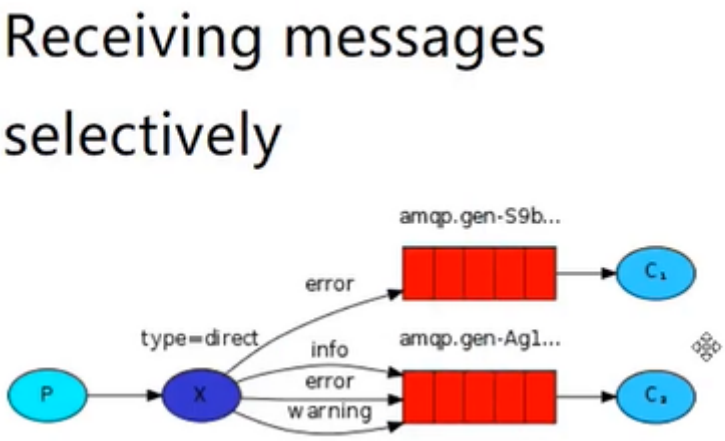
> 1、让生产者创建一个exhange并且指定类型，和一个或多个队列绑定到一起。

```
//3. 创建exchange - 绑定某一个队列
        //参数1：exchange的名称
        //参数2：指定exchange的类型 FANOUT - pubsub , DIRECT - Routing , TOPIC - 
Topics
        channel.exchangeDeclare("pubsub-exchange", BuiltinExchangeType.FANOUT);
        channel.queueBind("pubsub-queue1","pubsub-exchange","");
        channel.queueBind("pubsub-queue2","pubsub-exchange","");
```

> 2、消费者还是正常的监听某一个队列即可。

4.6 Rounting

> 一个生产者，一个交换机，两个队列，两个消费者



> 创建一个DIRECT类型的exchange,并且去根据RoutingKey去绑定指定的队列

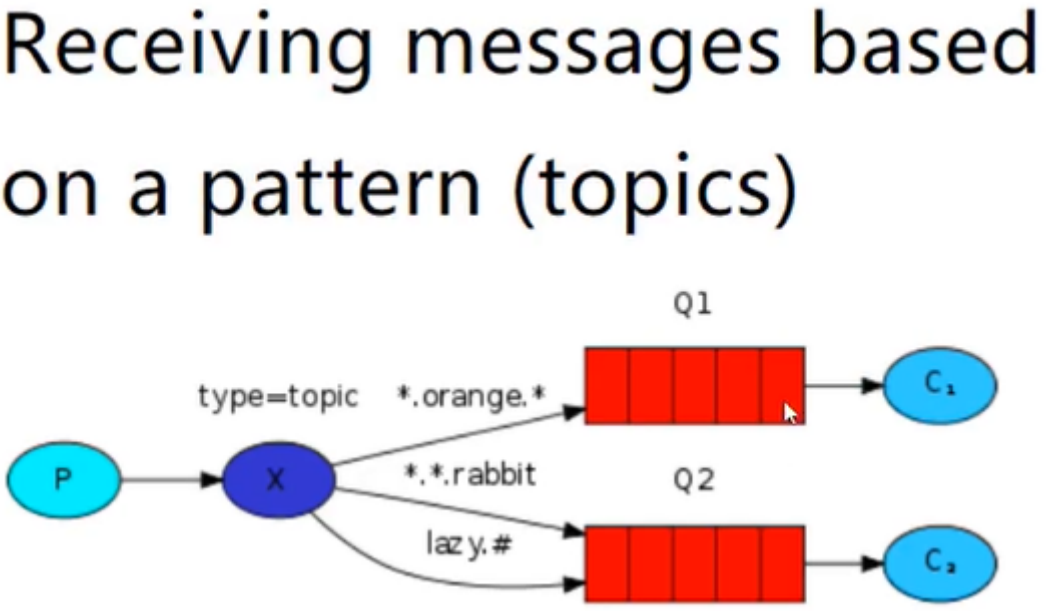
> 1、生产者在创建DIRECT类型的exchange后，去绑定响应的队列，并且在发送消息时，指定消息的具体RoutingKey即可。

```
//3. 创建exchange,routing-queue-error,routing-queue-info
        channel.exchangeDeclare("routing-exchange", BuiltinExchangeType.DIRECT);
        channel.queueBind("routing-queue-error","routing-exchange","ERROR");
        channel.queueBind("routing-queue-info","routing-exchange","INFO");

        //3. 发布消息到exchange，同时指定路由的规则
        channel.basicPublish("routing-
exchange","ERROR",null,"ERROR".getBytes());
        channel.basicPublish("routing-exchange","INFO",null,"INFO1".getBytes());
        channel.basicPublish("routing-exchange","INFO",null,"INFO2".getBytes());
        channel.basicPublish("routing-exchange","INFO",null,"INFO3".getBytes());
```

2、消费者基本没有变化

4.7 Topic

> 一个生产者，一个交换机，两个队列，两个消费者



> 1、生产者创建Topic的exchange并且绑定到队列中，这次绑定可以通过*和#关键字，对指定RoutingKey内容，编写时注意格式xxx.xxx.xxx去编写， * 代表 一个xxx ,而#代表多个xxx.xxx，在发送信息时，指定具体的RoutingKey到底是什么。

```
//3. 创建exchange绑定队列    topic-queue-1 topic-queue-2
//举例   动物的信息 <speed> <color> <what>
// *.red.*           -> *占位符
// fast.#            -> #通配符
//*.*.rabbit
channel.exchangeDeclare("topic-exchange", BuiltinExchangeType.TOPIC);
channel.queueBind("topic-queue-1","topic-exchange","*.red.*");
channel.queueBind("topic-queue-2","topic-exchange","fast.#");
channel.queueBind("topic-queue-2","topic-exchange","*.*.rabbit");
```

```
//3. 发布消息到exchange，同时指定路由的规则
channel.basicPublish("topic-exchange","fast.red.monkey",null,"红快猴
子".getBytes());
channel.basicPublish("topic-exchange","slow.black.monkey",null,"黑慢
狗".getBytes());
channel.basicPublish("topic-exchange","fast.white.cat",null,"快白猫".getBytes());
```

> 消费者只是监听队列，没变化

五、RabbitMQ整合SpringBoot

5.1 Springboot整合RabbitMQ

> 1、 SpringBoot整合RabbitMQ

> 2、导入依赖

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

> 3、编写配置文件

```
spring:
  rabbitmq:
    host: 192.168.190.130
    port: 5672
    username: test
    password: test
    virtual-host: /test
```

> 4、编写配置类，声明exchange和queue，并且绑定到一起

```
@Configuration
public class RabbitMQConfig {


    //1. 创建exchange - topic
    @Bean
    public TopicExchange getTopicExchange(){
        return new TopicExchange("boot-topic-exchange",true,false);
    }

    //2. 创建queue
    @Bean
    public Queue getQueue(){
        return new Queue("boot-queue",true,false,false,null);
    }

    //3. 绑定在一起
    @Bean
    public Binding getBinding(TopicExchange topicExchange,Queue queue){
        return BindingBuilder.bind(queue).to(topicExchange).with("*.red.*");
    }
```

```
}
```

> 5、发布消息到RabbitMQ

```
    @Autowired
    private RabbitTemplate rabbitTemplate;

    @Test
    void contextLoads() {
        rabbitTemplate.convertAndSend("boot-topic-exchange","slow.red.dog","红色大狼狗");
    }
```

> 6、创建消费者监听消息

```
@Component
public class Consumer {

    @RabbitListener(queues = "boot-queue")
    public void getMessage(Object message){
        System.out.println("接收到消息："+message);
    }
}
```

5.2 实现手动Ack

> 1、添加配置文件

```
spring:
  rabbitmq:
    listener:
      simple:
        acknowledge-mode: manual
```

> 2、在消费消息的位置，修改方法，再手动ack

```
    @RabbitListener(queues = "boot-queue")
    public void getMessage(String msg, Channel channel, Message message) throws IOException {
        System.out.println("接收到消息："+msg);
        //手动Ack
        channel.basicAck(message.getMessageProperties().getDeliveryTag(),false);
    }
```

六、RabbitMQ的其他操作

6.1 消息的可靠性

> RabbitMQ的事务：事务可以保证消息100%传递，可以通过事务的回滚去记录日志，后面定时再次发送当前消息。事务的操作，效率太低，加了事务操作后，比平时的操作效率的至少要慢100倍以上。

6.1.1 Confirm机制

> RabbitMQ除了事务，还提供Confirm的确认机制，这个效率比事务高很多。
>
> 普通Confirm方式

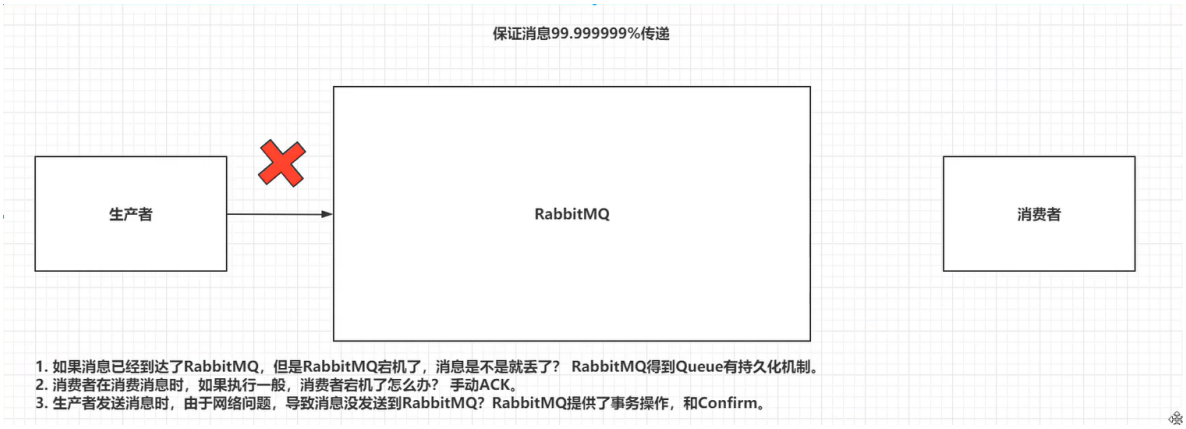
```
//3.1 开启confirm
channel.confirmSelect();
//3.2 发送消息
String msg="Hello-World";
channel.basicPublish("","HelloWorld",null,msg.getBytes());
//3.3 判断消息是否发送成功
if(channel.waitForConfirms()){
System.out.println("消息发送成功");
}
else {
System.out.println("消息发送失败");
}
```

> 批量Confirm方式

```
//3.1 开启confirm
channel.confirmSelect();
//3.2 批量发送消息
for(int i=0;i<1000;i++){
    String msg="Hello-World!!! "+i;
    channel.basicPublish("","HelloWorld",null,msg.getBytes());
}
//3.3 确定批量操作是否成功
channel.waitForConfirmsOrDie();//当你发送的全部信息，有一个失败的时候，就直接全部失败，抛出异常IOException
```

> 异步Confirm方式

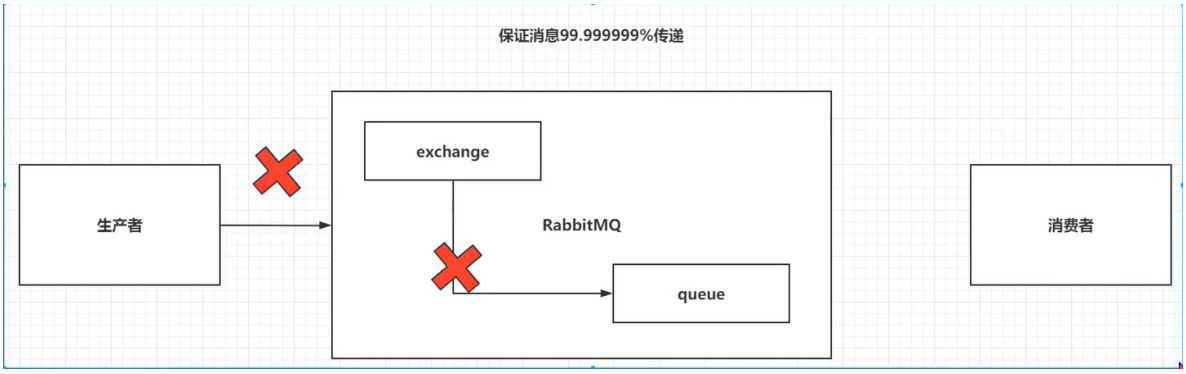
```
//3.1 开启confirm
channel.confirmSelect();
//3.2 批量发送消息
for(int i=0;i<1000;i++){
    String msg="Hello-World!!! "+i;
    channel.basicPublish("","HelloWorld",null,msg.getBytes());
}

//3.3 开启异步回调
channel.addConfirmListener(new ConfirmListener() {
    public void handleAck(long l, boolean b) throws IOException {
        System.out.println("消息发送成功，标识："+l+"，是否是批量"+b);
    }

    public void handleNack(long l, boolean b) throws IOException {
        System.out.println("消息发送失败，标识："+l+"，是否是批量"+b);
    }
});
```



ps:图中的一般是指一半（打错了字）

6.1.2 Return机制

> Confirm只能保证消息到达exchange，无法保证消息可以被exchange分发到指定queue。
>
> 而且exchange是不能持久化消息的，queue是可以持久化消息的。
>
> 采用Return机制来监听消息是否从exchange送到了指定的queue中。



> 1、开启Return机制

```
//开启Return机制
channel.addReturnListener(new ReturnListener() {
    public void handleReturn(int i, String s, String s1, String s2,
AMQP.BasicProperties basicProperties, byte[] bytes) throws IOException {
        //当消息没有到达queue时，才会执信。
        System.out.println(new String(bytes,"UTF-8")+"没有送达到Queue中！！");
    }
});

// 在发送消息时，指定var3参数为true
void basicPublish(String var1, String var2, boolean var3, BasicProperties var4,
byte[] var5) throws IOException;
```

6.1.3 SpringBoot实现Confirm以及Return机制

> 1、编写配置文件，开启Confirm以及Return机制

```
spring:
  rabbitmq:
    publisher-confirm-type: simple
    publisher-returns: true
```

> 2、指定RabbitTemplate对象，开启Confirm和Return机制，并且编写了回调方法

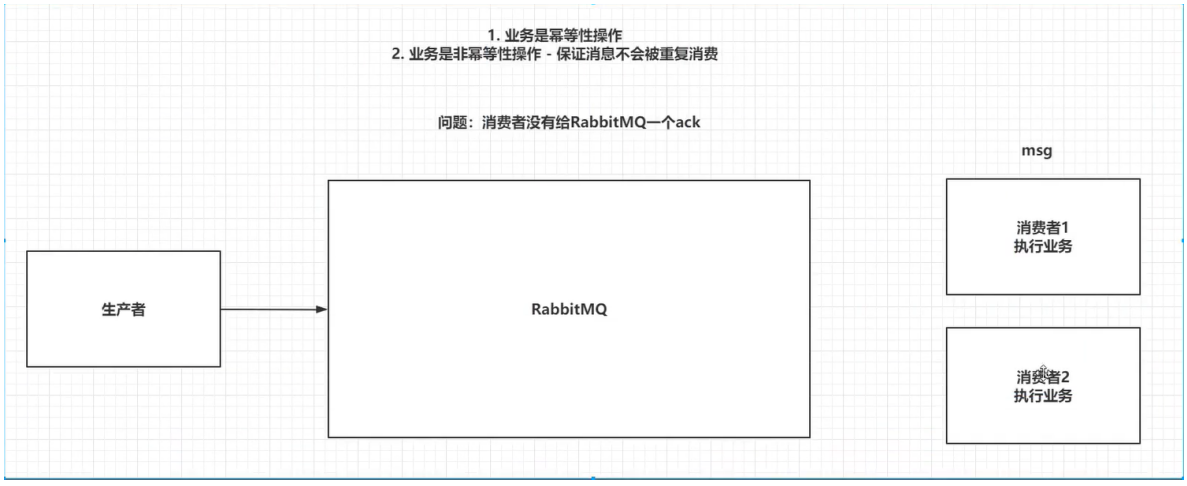
```
@Component
public class PublisherConfirmAndReturnConfig implements
 RabbitTemplate.ConfirmCallback,RabbitTemplate.ReturnsCallback {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    @PostConstruct //init-method
    public void initMethod(){
        rabbitTemplate.setConfirmCallback(this);
        rabbitTemplate.setReturnsCallback(this);
    }


    @Override
    public void confirm(CorrelationData correlationData, boolean b, String s) {
        if(b){
            System.out.println("消息已经送达到了Exchange");
        }
        else {
            System.out.println("消息没有送达到Exchange");
        }
    }


    @Override
    public void returnedMessage(ReturnedMessage returnedMessage) {
        System.out.println("消息没有送达到Queue");
    }
}
```

6.2 消息重复消费

6.2.1 避免消息重复消费实现

> 重复消费消息，会对费幂等性操作造成问题。

> 重复消费消息的原因是，消费者没有给RabbitMQ一个ack



> 为了解决消息重复的问题，可以采用Redis，在消费者消费消息之前，现将消息的id放到Redis中

> id-0（正在执行业务）
>
> id-1（执行业务成功）
>
> 如果ack失败，在RabbitMQ将消息交给其他的消费者时，先执行setnx，如果key已经存在，。获取他的值，如果是0，当前消费者就什么杜不做，如果是1，直接ack。
>
> 极端情况：第一个消费者在执行业务时，出现了死锁，在setnx的基础上，再给key设置一个生存时间。

> 1、生产者，在发送消息时，指定messageId

```
AMQP.BasicProperties properties= new AMQP.BasicProperties().builder()
        .deliveryMode(1)  //指定消息是否需要持久化  1- 需要持久化  2- 不需要持久化
        .messageId(UUID.randomUUID().toString())
        .build();
String msg="Hello-World!!! ";
channel.basicPublish("","HelloWorld",true,properties,msg.getBytes());
```

> 2、消费者，在消费消息时，根据具体业务逻辑去操作redis

```
DefaultConsumer consumer=new DefaultConsumer(channel){
    @Override
    public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
        Jedis jedis=new Jedis("39.100.38.230",6379);
        String messageId = properties.getMessageId();
        //1. setnx到Redis中，默认指定value-0
        String result = jedis.set(messageId, "0", "NX", "EX", 10);
        if(result != null && result.equalsIgnoreCase("OK")){
            System.out.println("接收到消息："+new String(body,"UTF-8"));
            //2. 消费成功，set messageId 1
            jedis.set(messageId,"1");
            channel.basicAck(envelope.getDeliveryTag(),false);
        }

        else {
            //3. 如果1中的setnx失败，获取key对应的value，如果是0 ，return，如果是1
            String s=jedis.get(messageId);
            if("1".equalsIgnoreCase(s)){
                channel.basicAck(envelope.getDeliveryTag(),false);
            }
        }
    }
};
```

6.2.2 SpringBoot如何实现

> 1、导入依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

> 2、编写配置文件

```
  redis:
    host: 39.100.38.230
    port: 6379
```

> 3、修改生产者

```
@Test
void contextLoads() throws IOException {
    CorrelationData messageId=new CorrelationData(UUID.randomUUID().toString());
    rabbitTemplate.convertAndSend("boot-topic-exchange","slow.red.dog","红色大狼
狗",messageId);
    System.in.read();
}
```

> 修改消费者

```
@Autowired
private StringRedisTemplate redisTemplate;

@RabbitListener(queues = "boot-queue")
public void getMessage(String msg, Channel channel, Message message) throws
IOException {
    System.out.println(redisTemplate.toString());
    //0. 获取MessageId
    String
messageId=message.getMessageProperties().getHeader("spring_returned_message_corr
elation");
    //1. 设置key到Redis
    if(redisTemplate.opsForValue().setIfAbsent(messageId,"0",10,
TimeUnit.SECONDS)){

 System.out.println(redisTemplate.opsForValue().setIfAbsent(messageId,"0",10,
TimeUnit.SECONDS));
        //2. 消费消息
        System.out.println("接收到消息：" + msg);

        //3. 设置key的value为1
        redisTemplate.opsForValue().set(messageId,"1",10,TimeUnit.SECONDS);
        //4. 手动ack
        channel.basicAck(message.getMessageProperties().getDeliveryTag(),false);
    }
    else {
        //5. 获取Redis中的value即可，如果是1，手动ack
        if("1".equalsIgnoreCase(redisTemplate.opsForValue().get(messageId))){
            //手动Ack

 channel.basicAck(message.getMessageProperties().getDeliveryTag(),false);
        }
    }
}
```

七、RabbitMQ应用