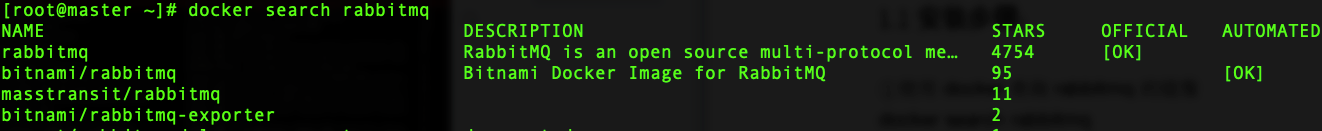
## 基于docker安装

### 1.1安装步骤

视频教程：<https://www.bilibili.com/video/BV15k4y1k7Ep>

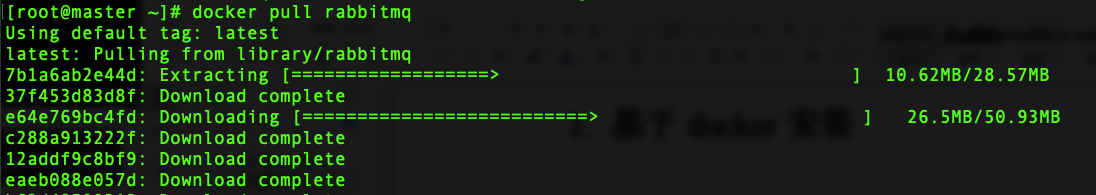
1. 使用docker查询rabbitmq的镜像

docker search rabbitmq



1. 拉取name为rabbitmq的镜像，这是最新的。如果需要安装其他版本在rabbitmq后面跟上版本号即可。

docker pull rabbitmq



1. 运行mq

docker run -d -p 15672:15672 -p 5672:5672 --name rabbitmq rabbitmq

(如果端口被占用先kill掉 netstat -antp | grep :15672 )

可选：

-e RABBITMQ\_DEFAULT\_VHOST=my\_vhost \

-e RABBITMQ\_DEFAULT\_USER=admin \

-e RABBITMQ\_DEFAULT\_PASS=admin \

--hostname myRabbit \

参数说明：

-d：表示在后台运行容器；

-p：将容器的端口 5672（应用访问端口）和 15672 （控制台Web端口号）映射到主机中；

-e：指定环境变量：

RABBITMQ\_DEFAULT\_VHOST：默认虚拟机名；

RABBITMQ\_DEFAULT\_USER：默认的用户名；

RABBITMQ\_DEFAULT\_PASS：默认的用户密码；

--hostname：指定主机名（RabbitMQ 的一个重要注意事项是它根据所谓的 节点名称 存储数据，默认为主机名）；

--name rabbitmq：设置容器名称；

rabbitmq：容器使用的镜像名称；

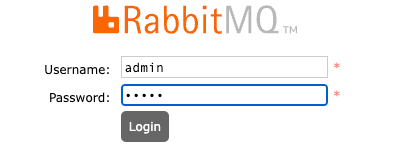
1. 启动rabbitmq\_management

docker exec -it rabbitmq rabbitmq-plugins enable rabbitmq\_management

5.访问rabbitmq后台

浏览器输入地址：http://ip:15672 即可访问后台管理页面，这里的 ip 为运行 RabbitMQ 所在的服务器的 IP 地址；

默认用户名和密码都是 guest（若容器创建时指定了用户名密码，则填写相应用户名密码）；



若点击Channels提示"Stats in management UI are disabled on this node"，

则进入容器内部

docker exec -it rabbitmq /bin/bash

进入目录

cd /etc/rabbitmq/conf.d/

执行

echo management\_agent.disable\_metrics\_collector = false > management\_agent.disable\_metrics\_collector.conf

推出容器

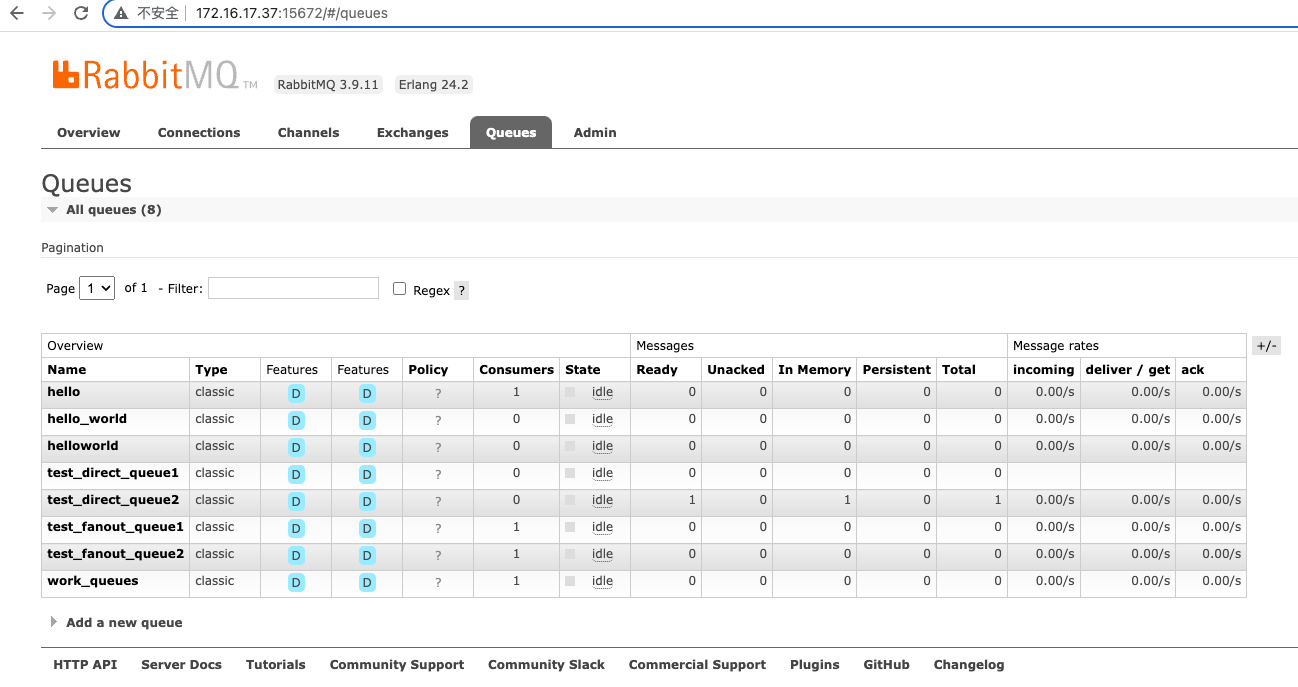
exit;

重启容器

docker restart rabbitmq

队列消息查看方法：

点击Queues选项卡-->点击队列名,进入队列详情页面-->点击Get messages伸缩栏-->点击Get Message(s)按钮。

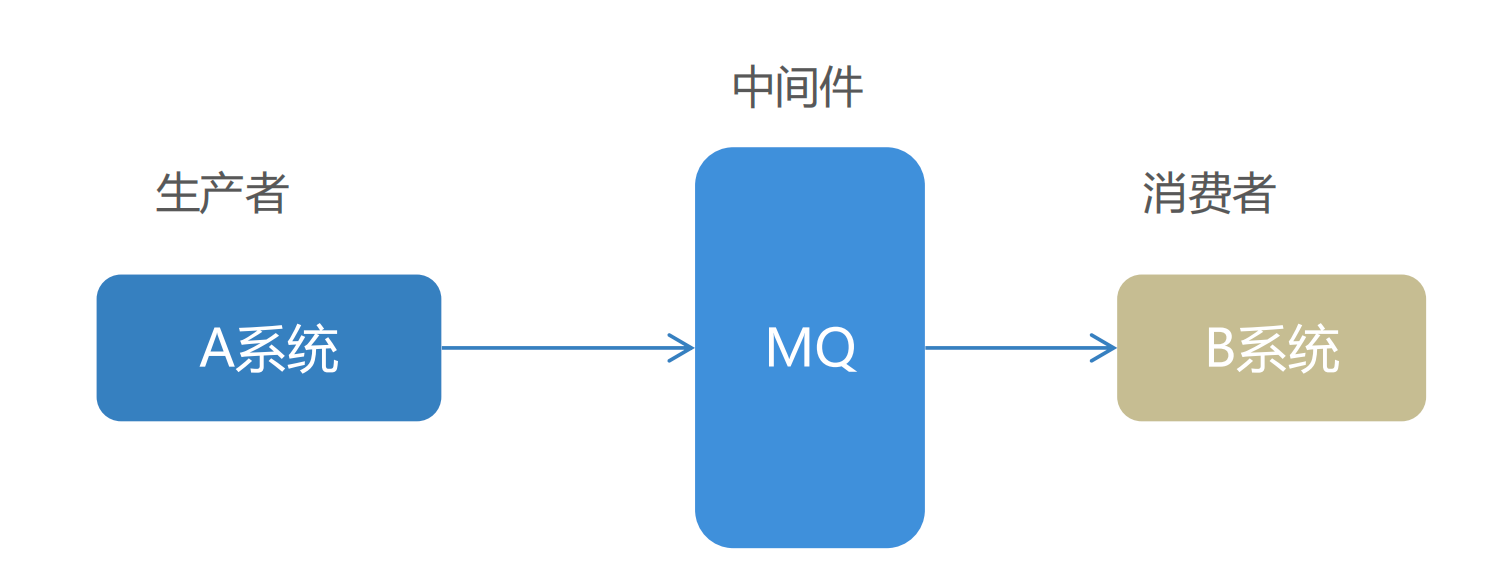
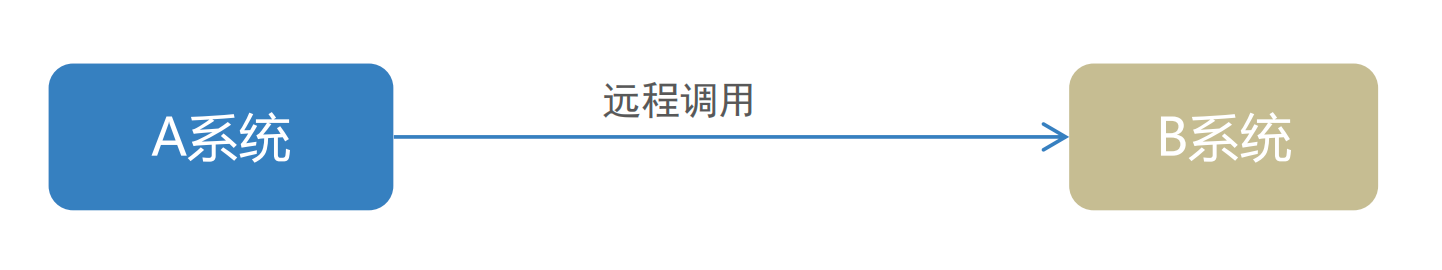




### 2.2基本概念

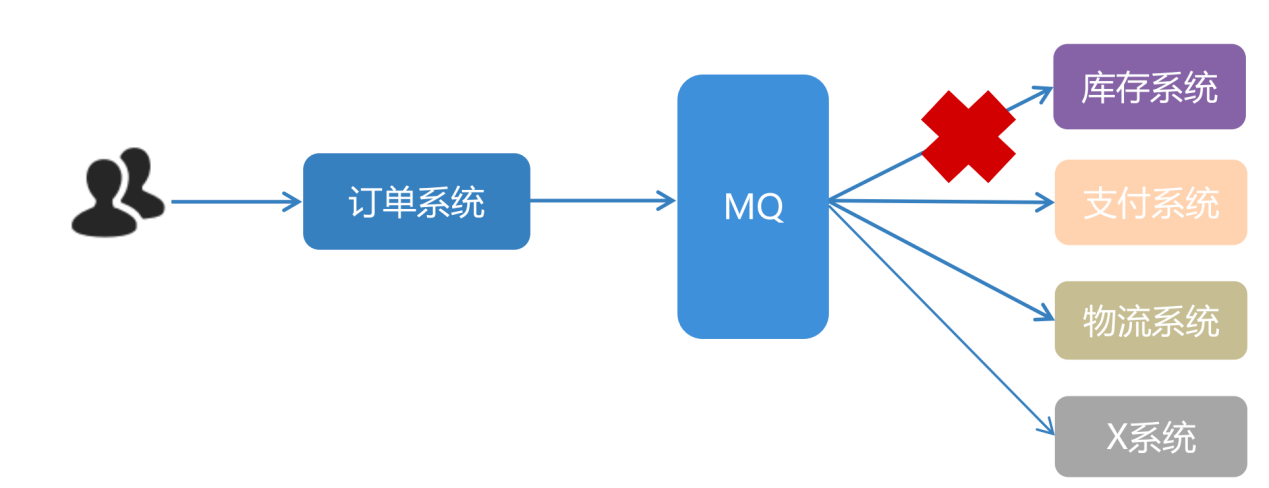
#### （1）MQ概述

MQ全称Message Queue（消息队列），是在消息的传输过程中保存消息的容器。多用于分布式系统之间进行通信。



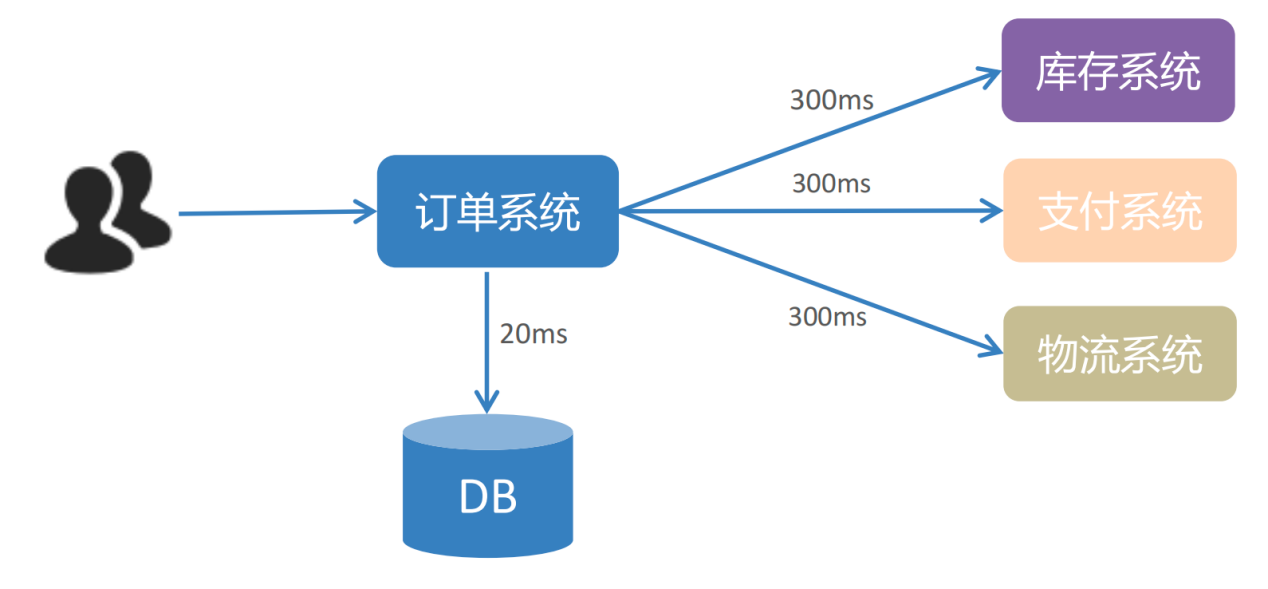
#### （2）MQ的优势

1. **应用解耦**

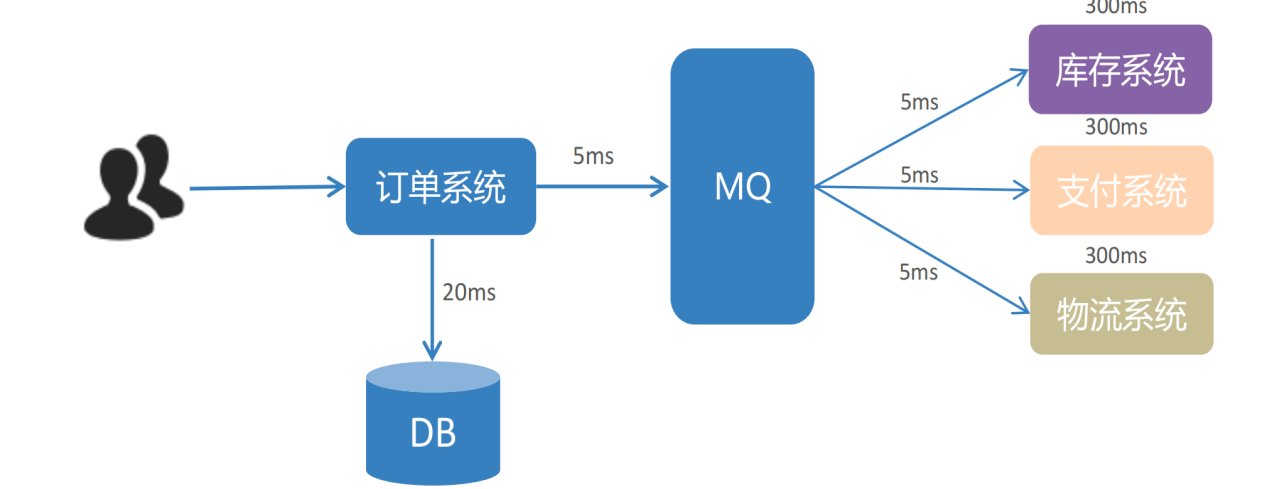


引入MQ后，使得订单系统与后面的系统解耦。当库存系统短暂产生异常时，也不会影响到订单系统。所以使用MQ后使得应用间解耦，提升了容错性和可维护性。

1. **异步提速**

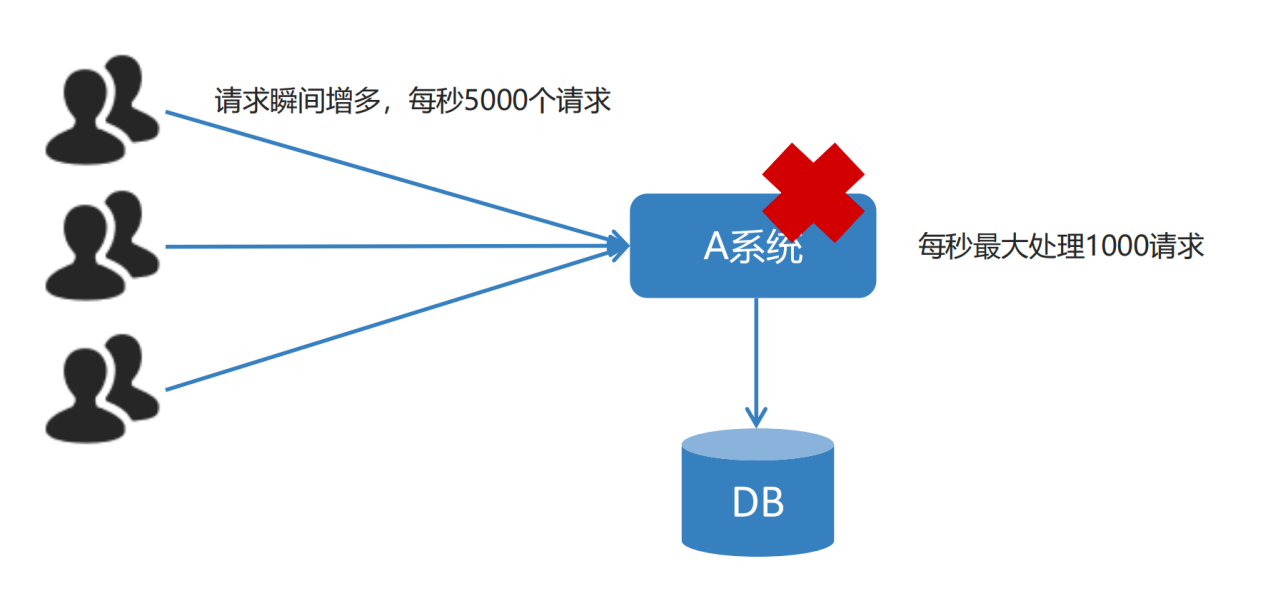


一个下单操作耗时：20 + 300 + 300 + 300 = 920ms

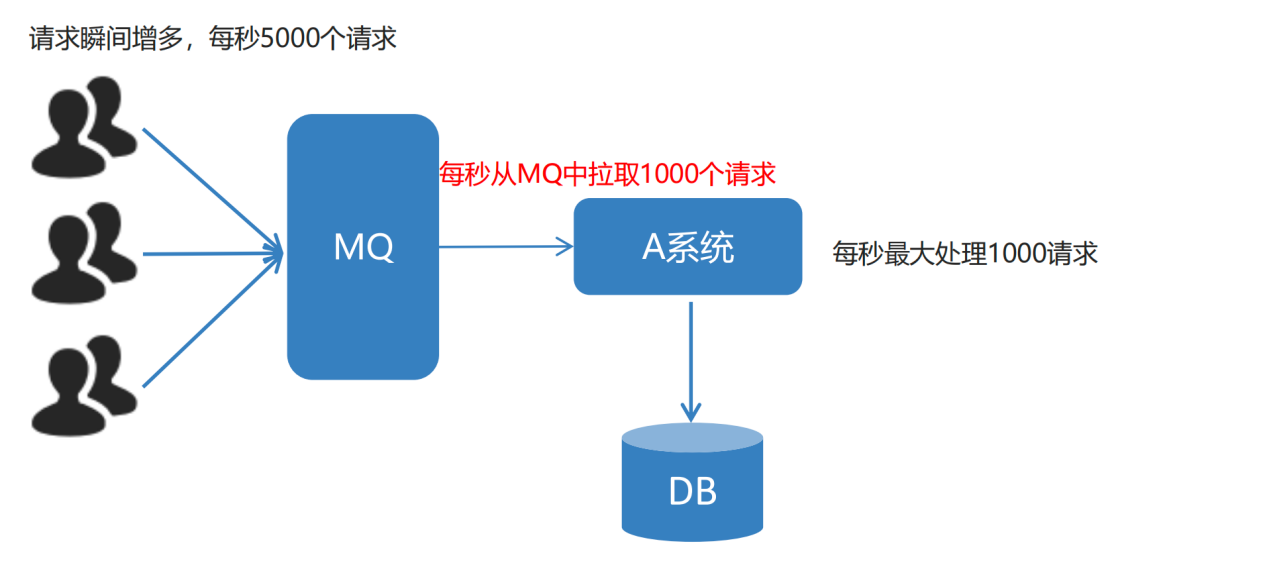
用户点击完下单按钮后，需要等待920ms才能得到下单响应，太慢。

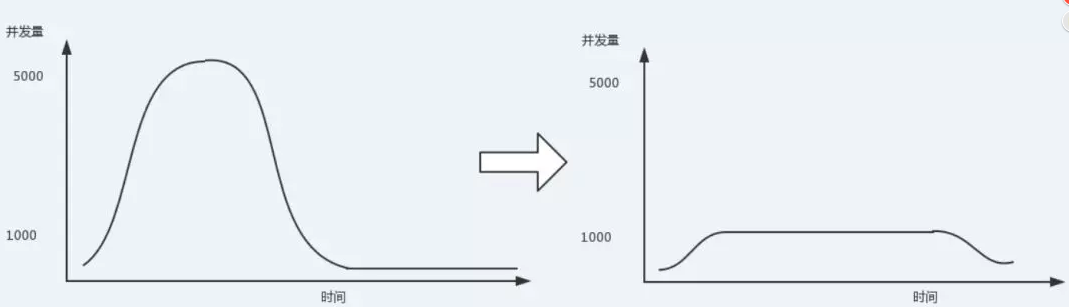
用户点击完下单按钮后，只需等待25ms就能得到下单响应(20 + 5 = 25ms)。提升用户体验和系统吞吐量（单位时间内处理请求的数目）。

1. **削峰填谷**



请求瞬间增多时，A系统无法处理过多请求，发生宕机，系统变得不可用。

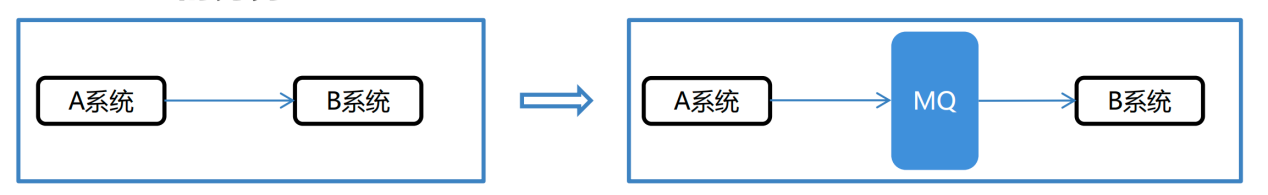




使用了MQ 之后，限制A系统消费消息的速度为1000，这样一来，高峰期产生的数据势必会被积压在MQ 中，高峰就被“削”掉了，但是因为消息积压，在高峰期过后的一段时间内，A系统消费消息的速度还是会维持在1000，直到消费完积压的消息，这就叫做“填谷”。

使用MQ后，可以提高系统稳定性。

#### （3）MQ的劣势



1. **系统可用性降低**

系统引入的外部依赖越多，系统稳定性越差。一旦MQ 宕机，就会对业务造成影响。如何保证MQ的高可用？

1. **系统复杂度提高**

MQ 的加入大大增加了系统的复杂度，以前系统间是同步的远程调用，现在是通过MQ 进行异步调用。如何保证消息没有被重复消费？怎么处理消息丢失情况？那么保证消息传递的顺序性？

1. **一致性问题**

A 系统处理完业务，通过MQ 给B、C、D三个系统发消息数据，如果B 系统、C 系统处理成功，D 系统处理失败。如何保证消息数据处理的一致性？

#### （4）小结

既然MQ 有优势也有劣势，那么使用MQ 需要满足什么条件呢？

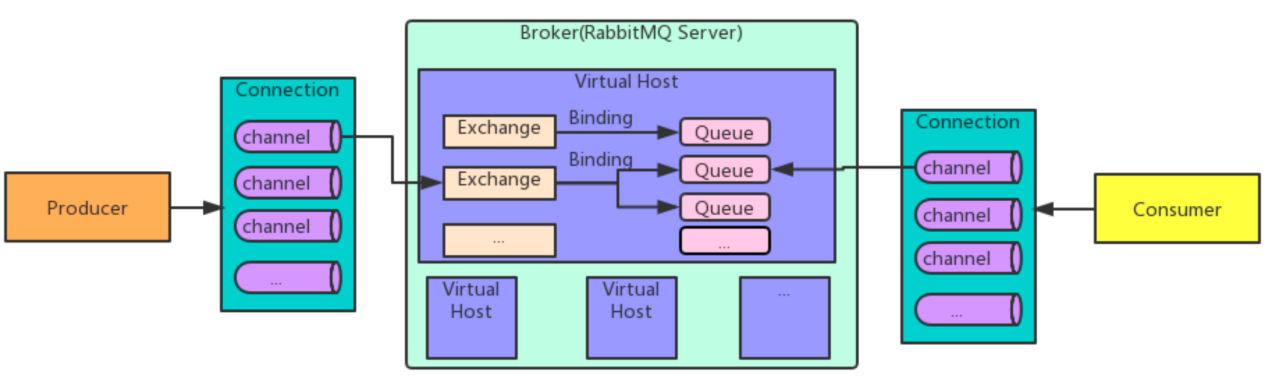
①生产者不需要从消费者处获得反馈。引入消息队列之前的直接调用，其接口的返回值应该为空，这才让明明下层的动作还没做，上层却当成动作做完了继续往后走，即所谓异步成为了可能。

②容许短暂的不一致性。

③确实是用了有效果。即解耦、提速、削峰这些方面的收益，超过加入MQ，管理MQ这些成本。

#### （5）RabbitMQ相关概念

RabbitMQ基础架构如下图所示：



⚫Broker：接收和分发消息的应用，RabbitMQ Server就是Message Broker

⚫Virtual host：出于多租户和安全因素设计的，把AMQP 的基本组件划分到一个虚拟的分组中，类似于网络中的namespace 概念。当多个不同的用户使用同一个RabbitMQ server 提供的服务时，可以划分出多

个vhost，每个用户在自己的vhost 创建exchange／queue 等

⚫Connection：publisher／consumer 和broker 之间的TCP 连接

⚫Channel：如果每一次访问RabbitMQ 都建立一个Connection，在消息量大的时候建立TCP Connection的开销将是巨大的，效率也较低。Channel 是在connection 内部建立的逻辑连接，如果应用程序支持多线程，通常每个thread创建单独的channel 进行通讯，AMQP method 包含了channel id 帮助客户端和

message broker 识别channel，所以channel 之间是完全隔离的。Channel 作为轻量级的Connection 极大减少了操作系统建立TCP connection 的开销

⚫Exchange：message 到达broker 的第一站，根据分发规则，匹配查询表中的routing key，分发消息到queue 中去。常用的类型有：direct (point-to-point), topic (publish-subscribe) and fanout (multicast)

⚫Queue：消息最终被送到这里等待consumer 取走

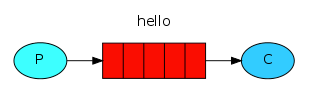
⚫Binding：exchange 和queue 之间的虚拟连接，binding 中可以包含routing key。Binding 信息被保存到exchange 中的查询表中，用于message 的分发依据

#### （6）工作模式

RabbitMQ提供了7种工作模式：简单模式、工作队列模式、Publish/Subscribe发布与订阅模式、Routing 路由模式、Topics主题模式、RPC远程调用模式、发布者确认模式。

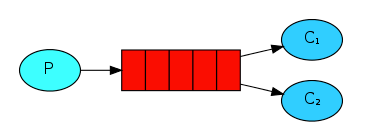
官网对应模式介绍：<https://www.rabbitmq.com/getstarted.html>

⚫简单模式



一个生产者对应一个消费者，生产者生成消息，消费者监听消息，若是消费者监听到它所需要的消息，就会消费该消息，这种消息是次性的，被消费了就没有了。

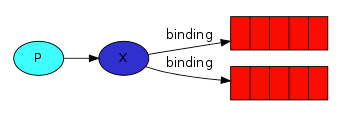
⚫工作队列模式



工作队列也就是简单模式的强化版，一个队列可以有多个生产者，也可以有多个消费者来竞争消费消息，但是我们仍需保证队列的[幂等性](https://so.csdn.net/so/search?q=%E5%B9%82%E7%AD%89%E6%80%A7&spm=1001.2101.3001.7020" \t "/Users/yongchaoxing/Documents\\x/_blank)，队列存在就不能再创建同名队列

应用场景：对于任务过重或任务较多的情况，使用工作队列可以提高任务处理速度。

⚫Publish/Subscribe模式



在订阅模型中，多了一个Exchange 角色，而且过程略有变化：

P：生产者，也就是要发送消息的程序，但是不再发送到队列中，而是发给X（交换机）

C：消费者，消息的接收者，会一直等待消息到来

Queue：消息队列，接收消息、缓存消息

Exchange：交换机(X)。一方面，接收生产者发送的消息。另一方面，知道如何处理消息，例如递交给某个特别队列、递交给所有队列、或是将消息丢弃。到底如何操作，取决于Exchange的类型。Exchange有常见以下3种类型：

➢Fanout：广播，将消息交给所有绑定到交换机的队列

➢Direct：定向，把消息交给符合指定routing key 的队列

➢Topic：通配符，把消息交给符合routing pattern（路由模式）的队列

Exchange（交换机）只负责转发消息，不具备存储消息的能力，因此如果没有任何队列与Exchange 绑定，或者没有符合路由规则的队列，那么消息会丢失。

代码实现:

包括3个类：Producer\_PubSub、Consumer\_PubSub1、Consumer\_PubSub2

1. 编写一个生产者类Producer\_PubSub

|  |
| --- |
| import com.rabbitmq.client.BuiltinExchangeType;  import com.rabbitmq.client.Channel;  import com.rabbitmq.client.Connection;  import com.rabbitmq.client.ConnectionFactory;  import java.io.IOException;  import java.util.concurrent.TimeoutException;  /\*\*  \* 发送消息  \*/  public class Producer\_PubSub {  public static void main(String[] args) throws IOException, TimeoutException {  //1.创建连接工厂  ConnectionFactory factory = new ConnectionFactory();  //2. 设置参数  factory.setHost("172.16.17.37");//默认值 localhost，rabbitmq所运行的服务器ip  factory.setPort(5672); //端口 默认值 5672  // factory.setVirtualHost("/itcast");//虚拟机 默认值/  factory.setUsername("guest");//用户名 默认 guest  factory.setPassword("guest");//密码 默认值 guest  //3. 创建连接 Connection  Connection connection = factory.newConnection();  //4. 创建Channel  Channel channel = connection.createChannel();  /\*  exchangeDeclare(String exchange, BuiltinExchangeType type, boolean durable, boolean autoDelete, boolean internal, Map<String, Object> arguments)  参数：  1. exchange:交换机名称  2. type:交换机类型  DIRECT("direct"),：定向  FANOUT("fanout"),：扇形（广播），发送消息到每一个与之绑定队列。  TOPIC("topic"),通配符的方式  HEADERS("headers");参数匹配  3. durable:是否持久化  4. autoDelete:自动删除  5. internal：内部使用。 一般false  6. arguments：参数  \*/  String exchangeName = "test\_fanout";  //5. 创建交换机  channel.exchangeDeclare(exchangeName, BuiltinExchangeType.FANOUT,true,false,false,null);  //6. 创建队列  String queue1Name = "test\_fanout\_queue1";  String queue2Name = "test\_fanout\_queue2";  channel.queueDeclare(queue1Name,true,false,false,null);  channel.queueDeclare(queue2Name,true,false,false,null);  //7. 绑定队列和交换机  /\*  queueBind(String queue, String exchange, String routingKey)  参数：  1. queue：队列名称  2. exchange：交换机名称  3. routingKey：路由键，绑定规则  如果交换机的类型为fanout ，routingKey设置为""  \*/  channel.queueBind(queue1Name,exchangeName,"");  channel.queueBind(queue2Name,exchangeName,"");  String body = "日志信息：张三调用了findAll方法...日志级别：info...";  //8. 发送消息  channel.basicPublish(exchangeName,"",null,body.getBytes());  //9. 释放资源  channel.close();  connection.close();  }  } |

1. 编写消费者类Consumer\_PubSub1

|  |
| --- |
| import com.rabbitmq.client.\*;  import java.io.IOException;  import java.util.concurrent.TimeoutException;  public class Consumer\_PubSub1 {  public static void main(String[] args) throws IOException, TimeoutException {  //1.创建连接工厂  ConnectionFactory factory = new ConnectionFactory();  //2. 设置参数  factory.setHost("172.16.17.37");//ip 默认值 localhost  factory.setPort(5672); //端口 默认值 5672  // factory.setVirtualHost("/itcast");//虚拟机 默认值/  factory.setUsername("guest");//用户名 默认 guest  factory.setPassword("guest");//密码 默认值 guest  //3. 创建连接 Connection  Connection connection = factory.newConnection();  //4. 创建Channel  Channel channel = connection.createChannel();  String queue1Name = "test\_fanout\_queue1";  String queue2Name = "test\_fanout\_queue2";  /\*  basicConsume(String queue, boolean autoAck, Consumer callback)  参数：  1. queue：队列名称  2. autoAck：是否自动确认  3. callback：回调对象  \*/  // 接收消息  Consumer consumer = new DefaultConsumer(channel){  /\*  回调方法，当收到消息后，会自动执行该方法  1. consumerTag：标识  2. envelope：获取一些信息，交换机，路由key...  3. properties:配置信息  4. body：数据  \*/  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {  /\* System.out.println("consumerTag："+consumerTag);  System.out.println("Exchange："+envelope.getExchange());  System.out.println("RoutingKey："+envelope.getRoutingKey());  System.out.println("properties："+properties);\*/  System.out.println("body："+new String(body));  System.out.println("将日志信息打印到控制台.....");  }  };  channel.basicConsume(queue1Name,true,consumer);  //关闭资源？不要  }  } |

1. 编写消费者类Consumer\_PubSub2

|  |
| --- |
| import com.rabbitmq.client.\*;  import java.io.IOException;  import java.util.concurrent.TimeoutException;  public class Consumer\_PubSub2 {  public static void main(String[] args) throws IOException, TimeoutException {  //1.创建连接工厂  ConnectionFactory factory = new ConnectionFactory();  //2. 设置参数  factory.setHost("172.16.17.37");//ip 默认值 localhost  factory.setPort(5672); //端口 默认值 5672  // factory.setVirtualHost("/itcast");//虚拟机 默认值/  factory.setUsername("guest");//用户名 默认 guest  factory.setPassword("guest");//密码 默认值 guest  //3. 创建连接 Connection  Connection connection = factory.newConnection();  //4. 创建Channel  Channel channel = connection.createChannel();  String queue1Name = "test\_fanout\_queue1";  String queue2Name = "test\_fanout\_queue2";  /\*  basicConsume(String queue, boolean autoAck, Consumer callback)  参数：  1. queue：队列名称  2. autoAck：是否自动确认  3. callback：回调对象  \*/  // 接收消息  Consumer consumer = new DefaultConsumer(channel){  /\*  回调方法，当收到消息后，会自动执行该方法  1. consumerTag：标识  2. envelope：获取一些信息，交换机，路由key...  3. properties:配置信息  4. body：数据  \*/  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {  /\* System.out.println("consumerTag："+consumerTag);  System.out.println("Exchange："+envelope.getExchange());  System.out.println("RoutingKey："+envelope.getRoutingKey());  System.out.println("properties："+properties);\*/  System.out.println("body："+new String(body));  System.out.println("将日志信息保存数据库.....");  }  };  channel.basicConsume(queue2Name,true,consumer);  //关闭资源？不要  }  } |

**执行过程：**

先执行Producer\_PubSub的main方法，创建交换机test\_fanout，并将2个队列test\_fanout\_queue1和test\_fanout\_queue2绑定到test\_fanout，向交换机发送消息，消息会被转发给两个队列进行存储。

再分别执行Consumer\_PubSub1和Consumer\_PubSub2的main方法，消费者Consumer\_PubSub1取出test\_fanout\_queue1中的消息进行消费，消费者Consumer\_PubSub2取出test\_fanout\_queue2中的消息进行消费。

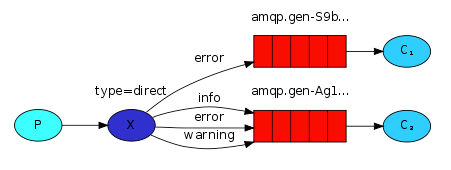
Consumer\_PubSub1对应的控制台输出



Consumer\_PubSub2对应的控制台输出



⚫Routing模式



说明：

⭕队列与交换机的绑定，不能是任意绑定了，而是要指定一个RoutingKey（路由key）

⭕消息的发送方在向Exchange 发送消息时，也必须指定消息的RoutingKey

Exchange 不再把消息交给每一个绑定的队列，而是根据消息的Routing Key 进行判断，只有队列的Routingkey 与消息的Routing key 完全一致，才会接收到消息

P：生产者，向Exchange 发送消息，发送消息时，会指定一个routing key

X：Exchange（交换机），接收生产者的消息，然后把消息递交给与routing key 完全匹配的队列

C1：消费者，其所在队列指定了需要routing key 为error 的消息

C2：消费者，其所在队列指定了需要routing key 为info、error、warning 的消息

代码实现：

包括3个类：Producer\_Routing、Consumer\_Routing1、Consumer\_Routing2

1. 编写生产者类Producer\_Routing

|  |
| --- |
| import com.rabbitmq.client.BuiltinExchangeType;  import com.rabbitmq.client.Channel;  import com.rabbitmq.client.Connection;  import com.rabbitmq.client.ConnectionFactory;  import java.io.IOException;  import java.util.concurrent.TimeoutException;  /\*\*  \* 发送消息  \*/  public class Producer\_Routing {  public static void main(String[] args) throws IOException, TimeoutException {  //1.创建连接工厂  ConnectionFactory factory = new ConnectionFactory();  //2. 设置参数  factory.setHost("172.16.17.37");//ip 默认值 localhost  factory.setPort(5672); //端口 默认值 5672  // factory.setVirtualHost("/itcast");//虚拟机 默认值/  factory.setUsername("guest");//用户名 默认 guest  factory.setPassword("guest");//密码 默认值 guest  //3. 创建连接 Connection  Connection connection = factory.newConnection();  //4. 创建Channel  Channel channel = connection.createChannel();  /\*  exchangeDeclare(String exchange, BuiltinExchangeType type, boolean durable, boolean autoDelete, boolean internal, Map<String, Object> arguments)  参数：  1. exchange:交换机名称  2. type:交换机类型  DIRECT("direct"),：定向  FANOUT("fanout"),：扇形（广播），发送消息到每一个与之绑定队列。  TOPIC("topic"),通配符的方式  HEADERS("headers");参数匹配  3. durable:是否持久化  4. autoDelete:自动删除  5. internal：内部使用。 一般false  6. arguments：参数  \*/  String exchangeName = "test\_direct";  //5. 创建交换机  channel.exchangeDeclare(exchangeName, BuiltinExchangeType.DIRECT,true,false,false,null);  //6. 创建队列  String queue1Name = "test\_direct\_queue1";  String queue2Name = "test\_direct\_queue2";  channel.queueDeclare(queue1Name,true,false,false,null);  channel.queueDeclare(queue2Name,true,false,false,null);  //7. 绑定队列和交换机  /\*  queueBind(String queue, String exchange, String routingKey)  参数：  1. queue：队列名称  2. exchange：交换机名称  3. routingKey：路由键，绑定规则  如果交换机的类型为fanout ，routingKey设置为""  \*/  //队列1绑定 error  channel.queueBind(queue1Name,exchangeName,"error");  //队列2绑定 info error warning  channel.queueBind(queue2Name,exchangeName,"info");  channel.queueBind(queue2Name,exchangeName,"error");  channel.queueBind(queue2Name,exchangeName,"warning");  String body = "日志信息：张三调用了delete方法...出错误了。。。日志级别：error...";  //8. 发送消息  channel.basicPublish(exchangeName,"error",null,body.getBytes());  //9. 释放资源  channel.close();  connection.close();  }  } |

1. 编写消费者类Consumer\_Routing1

|  |
| --- |
| import com.rabbitmq.client.\*;  import java.io.IOException;  import java.util.concurrent.TimeoutException;  public class Consumer\_Routing1 {  public static void main(String[] args) throws IOException, TimeoutException {  //1.创建连接工厂  ConnectionFactory factory = new ConnectionFactory();  //2. 设置参数  factory.setHost("172.16.17.37");//ip 默认值 localhost  factory.setPort(5672); //端口 默认值 5672  // factory.setVirtualHost("/itcast");//虚拟机 默认值/  factory.setUsername("guest");//用户名 默认 guest  factory.setPassword("guest");//密码 默认值 guest  //3. 创建连接 Connection  Connection connection = factory.newConnection();  //4. 创建Channel  Channel channel = connection.createChannel();  String queue1Name = "test\_direct\_queue1";  String queue2Name = "test\_direct\_queue2";  /\*  basicConsume(String queue, boolean autoAck, Consumer callback)  参数：  1. queue：队列名称  2. autoAck：是否自动确认  3. callback：回调对象  \*/  // 接收消息  Consumer consumer = new DefaultConsumer(channel){  /\*  回调方法，当收到消息后，会自动执行该方法  1. consumerTag：标识  2. envelope：获取一些信息，交换机，路由key...  3. properties:配置信息  4. body：数据  \*/  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {  /\* System.out.println("consumerTag："+consumerTag);  System.out.println("Exchange："+envelope.getExchange());  System.out.println("RoutingKey："+envelope.getRoutingKey());  System.out.println("properties："+properties);\*/  System.out.println("body："+new String(body));  System.out.println("将日志信息打印到控制台.....");  }  };  channel.basicConsume(queue1Name,true,consumer);  //关闭资源？不要  }  } |

1. 编写消费者类Consumer\_Routing2

|  |
| --- |
| import com.rabbitmq.client.\*;  import java.io.IOException;  import java.util.concurrent.TimeoutException;  public class Consumer\_Routing2 {  public static void main(String[] args) throws IOException, TimeoutException {  //1.创建连接工厂  ConnectionFactory factory = new ConnectionFactory();  //2. 设置参数  factory.setHost("172.16.17.37");//ip 默认值 localhost  factory.setPort(5672); //端口 默认值 5672  // factory.setVirtualHost("/itcast");//虚拟机 默认值/  factory.setUsername("guest");//用户名 默认 guest  factory.setPassword("guest");//密码 默认值 guest  //3. 创建连接 Connection  Connection connection = factory.newConnection();  //4. 创建Channel  Channel channel = connection.createChannel();  String queue1Name = "test\_direct\_queue1";  String queue2Name = "test\_direct\_queue2";  /\*  basicConsume(String queue, boolean autoAck, Consumer callback)  参数：  1. queue：队列名称  2. autoAck：是否自动确认  3. callback：回调对象  \*/  // 接收消息  Consumer consumer = new DefaultConsumer(channel){  /\*  回调方法，当收到消息后，会自动执行该方法  1. consumerTag：标识  2. envelope：获取一些信息，交换机，路由key...  3. properties:配置信息  4. body：数据  \*/  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {  /\* System.out.println("consumerTag："+consumerTag);  System.out.println("Exchange："+envelope.getExchange());  System.out.println("RoutingKey："+envelope.getRoutingKey());  System.out.println("properties："+properties);\*/  System.out.println("body："+new String(body));  System.out.println("将日志信息存储到数据库.....");  }  };  channel.basicConsume(queue2Name,true,consumer);  //关闭资源？不要  }  } |

**执行过程：**

先执行Producer\_Routing的main方法，创建交换机test\_direct，创建2个队列test\_direct\_queue1和test\_direct\_queue2，队列1绑定到test\_direct并设置routing key为error，队列2绑定到test\_direct并设置routing key为 info、error和warning。向交换机发送routing key为error的消息，消息会被转发给两个队列进行存储。

再分别执行Consumer\_Routing1和Consumer\_Routing2的main方法，消费者Consumer\_Routing1取出test\_direct\_queue1中的消息进行消费，消费者Consumer\_Routing2取出test\_direct\_queue2中的消息进行消费。

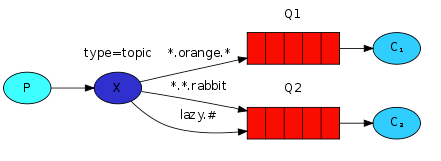
Consumer\_Routing1对应的控制台输出



Consumer\_Routing2对应的控制台输出



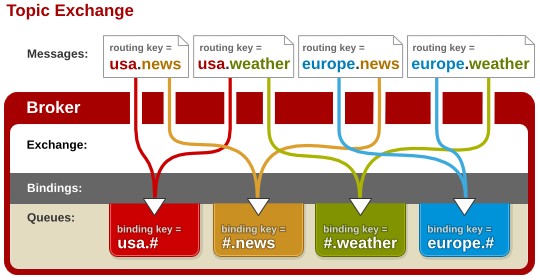
⚫Topics模式



⭕Topic 类型与Direct 相比，都是可以根据RoutingKey 把消息路由到不同的队列。只不过Topic 类型Exchange 可以让队列在绑定Routing key 的时候使用通配符。

⭕Routingkey 一般都是有一个或多个单词组成，多个单词之间以”.”分割，例如：item.insert

⭕通配符规则：# 匹配一个或多个词，\* 匹配不多不少恰好1个词，例如：item.# 能够匹配item.insert.abc 或者item.insert，item.\* 只能匹配item.insert



图解：

⭕红色Queue：绑定的是usa.# ，因此凡是以usa. 开头的routing key 都会被匹配到

⭕黄色Queue：绑定的是#.news ，因此凡是以.news 结尾的routing key 都会被匹配

代码实现：

包括3个类，Producer\_Topics、Consumer\_Topic1和Consumer\_Topic2

1. 编写生产者类Producer\_Topics

|  |
| --- |
| import com.rabbitmq.client.BuiltinExchangeType;  import com.rabbitmq.client.Channel;  import com.rabbitmq.client.Connection;  import com.rabbitmq.client.ConnectionFactory;  import java.io.IOException;  import java.util.concurrent.TimeoutException;  /\*\*  \* 发送消息  \*/  public class Producer\_Topics {  public static void main(String[] args) throws IOException, TimeoutException {  //1.创建连接工厂  ConnectionFactory factory = new ConnectionFactory();  //2. 设置参数  factory.setHost("172.16.17.37");//ip 默认值 localhost  factory.setPort(5672); //端口 默认值 5672  // factory.setVirtualHost("/itcast");//虚拟机 默认值/  factory.setUsername("guest");//用户名 默认 guest  factory.setPassword("guest");//密码 默认值 guest  //3. 创建连接 Connection  Connection connection = factory.newConnection();  //4. 创建Channel  Channel channel = connection.createChannel();  /\*  exchangeDeclare(String exchange, BuiltinExchangeType type, boolean durable, boolean autoDelete, boolean internal, Map<String, Object> arguments)  参数：  1. exchange:交换机名称  2. type:交换机类型  DIRECT("direct"),：定向  FANOUT("fanout"),：扇形（广播），发送消息到每一个与之绑定队列。  TOPIC("topic"),通配符的方式  HEADERS("headers");参数匹配  3. durable:是否持久化  4. autoDelete:自动删除  5. internal：内部使用。 一般false  6. arguments：参数  \*/  String exchangeName = "test\_topic";  //5. 创建交换机  channel.exchangeDeclare(exchangeName, BuiltinExchangeType.TOPIC,true,false,false,null);  //6. 创建队列  String queue1Name = "test\_topic\_queue1";  String queue2Name = "test\_topic\_queue2";  channel.queueDeclare(queue1Name,true,false,false,null);  channel.queueDeclare(queue2Name,true,false,false,null);  //7. 绑定队列和交换机  /\*  queueBind(String queue, String exchange, String routingKey)  参数：  1. queue：队列名称  2. exchange：交换机名称  3. routingKey：路由键，绑定规则  如果交换机的类型为fanout ，routingKey设置为""  \*/  // routing key 系统的名称.日志的级别。  //=需求： 所有error级别的日志存入数据库，所有order系统的日志存入数据库  channel.queueBind(queue1Name,exchangeName,"#.error");  channel.queueBind(queue1Name,exchangeName,"order.\*");  channel.queueBind(queue2Name,exchangeName,"\*.\*");  String body = "日志信息：张三调用了findAll方法...日志级别：info...";  //8. 发送消息  channel.basicPublish(exchangeName,"order.info",null,body.getBytes());  //9. 释放资源  channel.close();  connection.close();  }  } |

1. 编写消费者类Consumer\_Topic1

|  |
| --- |
| import com.rabbitmq.client.\*;  import java.io.IOException;  import java.util.concurrent.TimeoutException;  public class Consumer\_Topic1 {  public static void main(String[] args) throws IOException, TimeoutException {  //1.创建连接工厂  ConnectionFactory factory = new ConnectionFactory();  //2. 设置参数  factory.setHost("172.16.17.37");//ip 默认值 localhost  factory.setPort(5672); //端口 默认值 5672  // factory.setVirtualHost("/itcast");//虚拟机 默认值/  factory.setUsername("guest");//用户名 默认 guest  factory.setPassword("guest");//密码 默认值 guest  //3. 创建连接 Connection  Connection connection = factory.newConnection();  //4. 创建Channel  Channel channel = connection.createChannel();  String queue1Name = "test\_topic\_queue1";  String queue2Name = "test\_topic\_queue2";  /\*  basicConsume(String queue, boolean autoAck, Consumer callback)  参数：  1. queue：队列名称  2. autoAck：是否自动确认  3. callback：回调对象  \*/  // 接收消息  Consumer consumer = new DefaultConsumer(channel){  /\*  回调方法，当收到消息后，会自动执行该方法  1. consumerTag：标识  2. envelope：获取一些信息，交换机，路由key...  3. properties:配置信息  4. body：数据  \*/  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {  /\* System.out.println("consumerTag："+consumerTag);  System.out.println("Exchange："+envelope.getExchange());  System.out.println("RoutingKey："+envelope.getRoutingKey());  System.out.println("properties："+properties);\*/  System.out.println("body："+new String(body));  System.out.println("将日志信息存入数据库.......");  }  };  channel.basicConsume(queue1Name,true,consumer);  //关闭资源？不要  }  } |

1. 编写消费者类Consumer\_Topic2

|  |
| --- |
| import com.rabbitmq.client.\*;  import java.io.IOException;  import java.util.concurrent.TimeoutException;  public class Consumer\_Topic2 {  public static void main(String[] args) throws IOException, TimeoutException {  //1.创建连接工厂  ConnectionFactory factory = new ConnectionFactory();  //2. 设置参数  factory.setHost("172.16.17.37");//ip 默认值 localhost  factory.setPort(5672); //端口 默认值 5672  // factory.setVirtualHost("/itcast");//虚拟机 默认值/  factory.setUsername("guest");//用户名 默认 guest  factory.setPassword("guest");//密码 默认值 guest  //3. 创建连接 Connection  Connection connection = factory.newConnection();  //4. 创建Channel  Channel channel = connection.createChannel();  String queue1Name = "test\_topic\_queue1";  String queue2Name = "test\_topic\_queue2";  /\*  basicConsume(String queue, boolean autoAck, Consumer callback)  参数：  1. queue：队列名称  2. autoAck：是否自动确认  3. callback：回调对象  \*/  // 接收消息  Consumer consumer = new DefaultConsumer(channel){  /\*  回调方法，当收到消息后，会自动执行该方法  1. consumerTag：标识  2. envelope：获取一些信息，交换机，路由key...  3. properties:配置信息  4. body：数据  \*/  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {  /\* System.out.println("consumerTag："+consumerTag);  System.out.println("Exchange："+envelope.getExchange());  System.out.println("RoutingKey："+envelope.getRoutingKey());  System.out.println("properties："+properties);\*/  System.out.println("body："+new String(body));  System.out.println("将日志信息打印控制台.......");  }  };  channel.basicConsume(queue2Name,true,consumer);  //关闭资源？不要  }  } |

**执行过程：**

先执行Producer\_Topics的main方法，创建类型为TOPIC的交换机test\_topic，创建2个队列test\_topic\_queue1和test\_topic\_queue2，队列1绑定到test\_topic并设置routing key为#.error和order.\*，队列2绑定到test\_direct并设置routing key为 \*.\*。向交换机发送routing key为order.info的消息，消息会匹配到队列1的order.\*，转发给队列1，也会匹配到队列2的\*.\*，转发给队列2。

再分别执行Consumer\_Topic1和Consumer\_Topic2的main方法，消费者Consumer\_Topic1取出test\_topic\_queue1中的消息进行消费，消费者Consumer\_Topic2取出test\_topic\_queue2中的消息进行消费。

Consumer\_Topic1对应的控制台输出



Consumer\_Topic2对应的控制台输出



⚫工作模式总结

1. 简单模式HelloWorld

一个生产者、一个消费者，不需要设置交换机（使用默认的交换机）。

1. 工作队列模式Work Queue

一个生产者、多个消费者（竞争关系），不需要设置交换机（使用默认的交换机）。

1. 发布订阅模式Publish/subscribe

需要设置类型为fanout 的交换机，并且交换机和队列进行绑定，当发送消息到交换机后，交换机会将消息发送到绑定的队列。

1. 路由模式Routing

需要设置类型为direct 的交换机，交换机和队列进行绑定，并且指定routing key，当发送消息到交换机后，交换机会根据routing key 将消息发送到对应的队列。

1. 通配符模式Topic

需要设置类型为topic 的交换机，交换机和队列进行绑定，并且指定通配符方式的routing key，当发送消息到交换机后，交换机会根据routing key 将消息发送到对应的队列。

#### （7）Spring整合RabbitMQ

需求：使用Spring整合RabbitMQ

步骤：

|  |  |
| --- | --- |
| 生产者   1. 创建生产者工程 2. 添加依赖 3. 配置整合 4. 编写代码发送消息 | 消费者  ①创建消费者工程  ②添加依赖  ③配置整合  ④编写消息监听器 |

代码详情参考rabbit-mq项目中的spring-rabbit-consumers模块和spring-rabbit-producers模块。其中，resources文件夹中的rabbitmq.properties配置文件中存储rabbitmq的ip、port、用户名和密码（修改为自己的rabbitmq相关信息）；spring-rabbitmq-xxx.xml文件中定义了交换机Exchange、队列Queue、rabbitTemplate和队列监听器等bean对象。

**执行步骤：**

①依次执行spring-rabbitmq-producers/test/java/org/example/ProducerTest类中的@Test注解的方法，观察Rabbitmq管理页面出现的相应变化。

②执行spring-rabbitmq-consumers/test/java/org/example/ConsumerTest类中的test1方法，观察控制台中是否打印监听器监听到的spring\_queue队列的消息。

③重新执行spring-rabbitmq-producers/test/java/org/example/ProducerTest类中的testHelloWorld方法，观察ConsumerTest对应的控制台是否打印出监听器监听到的spring\_queue队列的实时消息。

#### （8）Springboot整合RabbitMQ

步骤：

|  |  |
| --- | --- |
| **生产端**  1. 创建生产者SpringBoot工程  2. 引入start，依赖坐标  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-amqp</artifactId>  </dependency>  3. 编写yml配置，基本信息配置  4. 定义交换机，队列以及绑定关系的配置类  5. 注入RabbitTemplate，调用方法，完成消息发送 | **消费端**  1. 创建消费者SpringBoot工程  2. 引入start，依赖坐标  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-amqp</artifactId>  </dependency>  3. 编写yml配置，基本信息配置  4. 定义监听类，使用@RabbitListener注解完成队列监听。  5.启动项目，监听器持续监听。 |

代码详情参考producer-springboot和consumer-springboot项目。

**执行步骤：**

①先执行producer-springboot/src/test/java/com/example/producerspringboot/ProducerSpringbootApplicationTests类中的testSend方法，向交换机boot\_topic\_exchange

发送routing key为”boot.haha”的消息。观察rabbitmq的管理页面中的队列bootQueue中是否出现该消息。

②运行consumer-springboot项目的启动类，观察控制台是否打印出监听器从bootQueue中监听到的消息。

#### 2.9高级特性

**⚫消息的可靠性投递**

在使用RabbitMQ 的时候，作为消息发送方希望杜绝任何消息丢失或者投递失败场景。RabbitMQ 为我们提供了两种方式用来控制消息的投递可靠性模式。

① confirm 确认模式

② return 退回模式

rabbitmq 整个消息投递的路径为：

producer--->rabbitmq broker--->exchange--->queue--->consumer

消息从producer 到exchange则会返回一个confirmCallback。

消息从exchange-->queue 投递失败则会返回一个returnCallback 。我们将利用这两个callback 控制消息的可靠性投递。

**⚫Consumer Ack**

ack指Acknowledge，确认。表示消费端收到消息后的确认方式。有三种确认方式：

•自动确认：acknowledge="none"

•手动确认：acknowledge="manual"

•根据异常情况确认：acknowledge="auto"，（这种方式使用麻烦，不作讲解）

其中自动确认是指，当消息一旦被Consumer接收到，则自动确认收到，并将相应message 从RabbitMQ 的消息缓存中移除。但是在实际业务处理中，很可能消息接收到，业务处理出现异常，那么该消息就会丢失。如果设置了手动确认方式，则需要在业务处理成功后，调用channel.basicAck()，手动签收，如果出现异常，则调用channel.basicNack()方法，让其自动重新发送消息。

**⚫消费端限流**

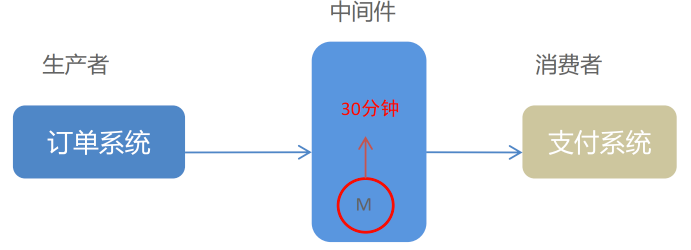
在<rabbit:listener-container> 中配置 prefetch属性设置消费端一次拉取多少消息

**⚫TTL**

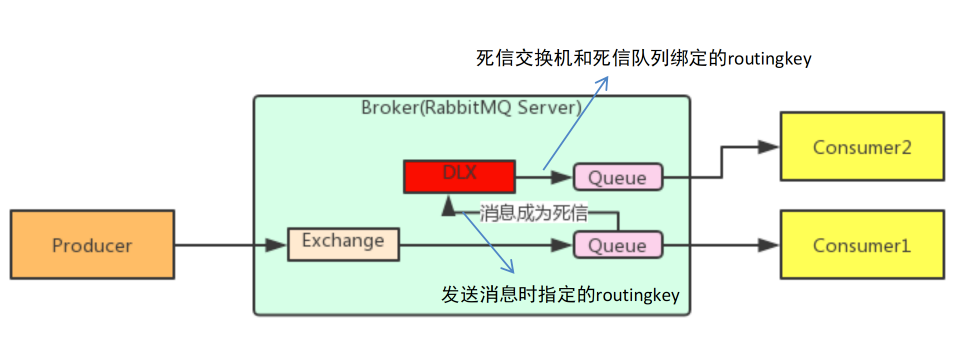
➢ TTL 全称 Time To Live（存活时间/过期时间）。

➢ 当消息到达存活时间后，还没有被消费，会被自动清除。

➢ RabbitMQ可以对消息设置过期时间，也可以对整个队列（Queue）设置过期时间。



**⚫死信队列**



1. 死信交换机和死信队列和普通的没有区别

2. 当消息成为死信后，如果该队列绑定了死信交换机，则消息会被死信交换机重新路由到死信队列

3. 消息成为死信的三种情况：

（1） 队列消息长度到达限制；

（2） 消费者拒接消费消息，并且不重回队列；

（3） 原队列存在消息过期设置，消息到达超时时间未被消费；

**⚫延迟队列**

延迟队列，即消息进入队列后不会立即被消费，只有到达指定时间后，才会被消费。

需求：

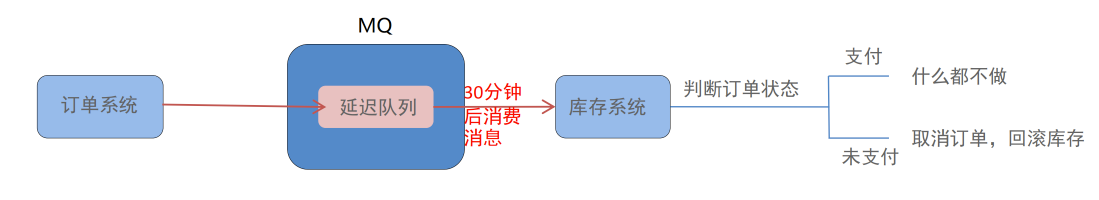
1. 下单后，30分钟未支付，取消订单，回滚库存。

2. 新用户注册成功7天后，发送短信问候。

实现方式：

1. 定时器

2. 延迟队列



很可惜，在RabbitMQ中并未提供延迟队列功能。

但是可以使用：**TTL+死信队列** 组合实现延迟队列的效果。

