1、消息队列与功能

消息是为了异步通信能力和解耦能力而产生的，就是将前台客户发送的请求作为消息存入一个消息队列里面，然后由消息的消费者再慢慢处理，实现了很好的异步处理（一件事情可以不必全部做完才做另一件事），除此之外对于流量削峰，应用解耦也是很有好处

1异步处理





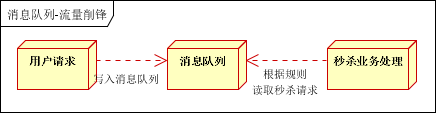
第一个图为同步处理，他们需要将所有的事情做完才能告诉用户注册完成，耗时150，而图二引入消息后，只需要将数据写入数据库和消息队列后就可以返回用户注册完成，至于后续的事情，系统会从消息队列里面找到数据再处理，这样极大的加强了体验

2应用解耦



对于两个分布式的系统，他们不必再紧密耦合，订单系统只需将订单加入消息队列，就算完成，而库存系统只需从消息队列里面取出数据然后处理即可

3流量削峰（秒杀）



比如十万个人抢一万个商品，只需将消息队列设置为一万，那么能进入队列则成功，其他不能进入队列的马上告诉秒杀失败，后续的任务可以等业务处理系统慢慢完成

消息的基本概念

1消息代理与目的地

消息代理解释市面上的一些软件如（AtiveMQ），目的地就是消息进入哪个队列，当消息发送者发送消息以后，将由消息代理接管，消息

理保证消息传递到指定目的地。

2点-点式

对于消息只有一个发送者和一个接收者，也就是说消息发送者将消息存入队列后，当有一个消息消费者得到消息后，消息动队列删除，其他消费者不能再得到消息

3发布订阅模式

发送者会将消息发给所有订阅他的人

4JMS  
基于JVM的消息队列服务api，跨平台能力较弱（ActiveMQ）

5AMQP(rabbitMQ)

网络线级协议,跨平台跨语言能利好，常用他的实现rabbitMQ，他只支持byte通信，支持五种消息交换

（1） direct exchange

（2）、fanout exchange

（3）、topic change

（4）、headers exchange

（5）、system exchange

6 Spring支持

spring-jms提供了对JMS的支持

spring-rabbit提供了对AMQP的支持

需要ConnectionFactory的实现来连接消息代理

提供JmsTemplate、RabbitTemplate来发送消息

@JmsListener（JMS）、@RabbitListener（AMQP）注解在方法上监听消息代理发布的消息

@EnableJms、@EnableRabbit开启支持

7 Springboot自动配置

JmsAutoConfiguration

RabbitAutoConfiguration

8 RabitMq

RabbitMq是基于AMQP实现的开源消息实现

9一些核心概念

Message

消息，消息是不具名的，它由消息头和消息体组成。消息体是不透明的，而消息头则由一系列的可选属性组成，这些属性包括routing-key（**路由键**）、priority（相对于其他消息的优先权）、delivery-mode（指出该消息可能需要持久性存储）等。

消息主要包含properties和body。Propertity可以对这个消息进行修饰，如优先级，延迟，而body则是这个消息的消息体

Publisher

消息的生产者，也是一个向交换器发布消息的客户端应用程序。

Exchange

交换器，用来接收生产者发送的消息并将这些消息路由给服务器中的队列。

Exchange有4种类型：direct(默认)，fanout, topic, 和headers，不同类型的Exchange转发消息的策略有所区别

queue

消息队列，用来保存消息直到发送给消费者。它是消息的容器，也是消息的终点。一个消息可投入一个或多个队列。消息一直在队列里面，等待消费者连接到这个队列将其取走。

Queue的一些设置

x-max-length 队列的最大长度

x-overflow：队列满了之后的策略，

1从队列的头部丢弃然后将这个消息收入队列

2拒绝投递

3丢弃在队列中最长时间的消息

Binding

绑定，用于消息队列和交换器之间的关联。一个绑定就是基于路由键将交换器和消息队列连接起来的路由规则，所以可以将交换器理解成一个由绑定构成的路由表。

Exchange 和Queue的绑定可以是多对多的关系。

Connection

网络连接，比如一个TCP连接，就是我们的应用和broker建立联系的连接，rabbitmq使用信道复用的技术将一个tcp连接复用为多个channel。

Channel

信道，多路复用连接中的一条独立的双向数据流通道。信道是建立在真实的TCP连接内的虚拟连接，AMQP 命令都是通过信道发出去的，不管是发布消息、订阅队列还是接收消息，这些动作都是通过信道完成。因为对于操作系统来说建立和销毁 TCP 都是非常昂贵的开销，所以引入了信道的概念，以复用一条 TCP 连接。

几乎所有的操作都是在channel上进行，channel是消息读写的通道，是单个会话。

Consumer

消息的消费者，表示一个从消息队列中取得消息的客户端应用程序。

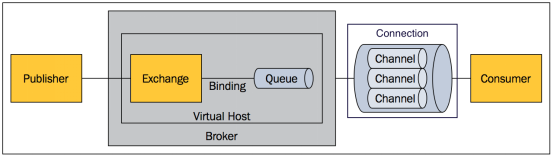
Broker

表示消息队列服务器实体，如rabbitmq（里面可包含多个vHost）

Virtual Host

虚拟主机，表示一批交换器、消息队列和相关对象。虚拟主机是共享相同的身份认证和加密环境的独立服务器域。每个 vhost 本质上就是一个 mini 版的 RabbitMQ 服务器，拥有自己的队列、交换器、绑定和权限机制。vhost 是 AMQP 概念的基础，必须在连接时指定，RabbitMQ 默认的 vhost 是 / 。

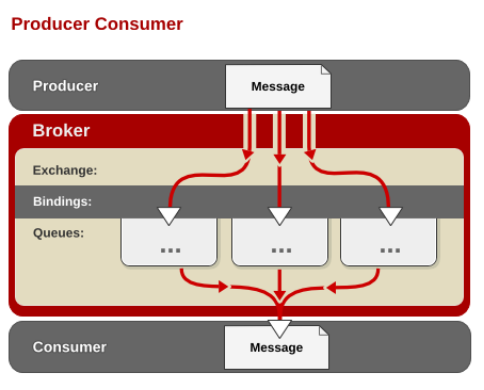
一个border下面可以有多个virtalhost，每个virtal host之间是相互隔离不相干扰的。



生产者与Broker只建立一条tcp连接，多路复用，当消息到达broker，brokker将数据给某一个exchange，然后exchange根据路由键和交换策略（Binding：绑定关系）将消息发送到合适的队列，消费者从队列里面拿数据也是通过一条tcp连接然后多路复用

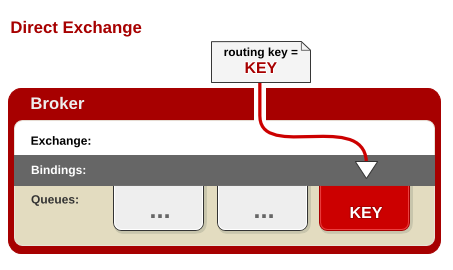
10 rabbitMq的运行机制

AMQP 中增加了 Exchange 和 Binding 的角色。生产者把消息发布到 Exchange 上，消息最终到达队列并被消费者接收，而 Binding 决定交换器的消息应该发送到那个队列。



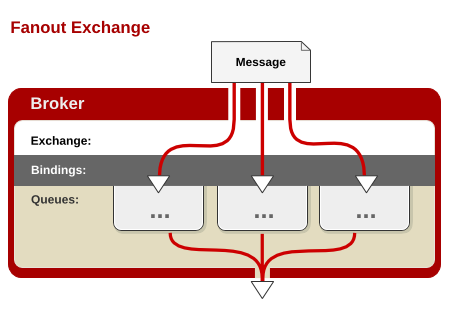
11 rabbitmq的Exchange

1direct模式

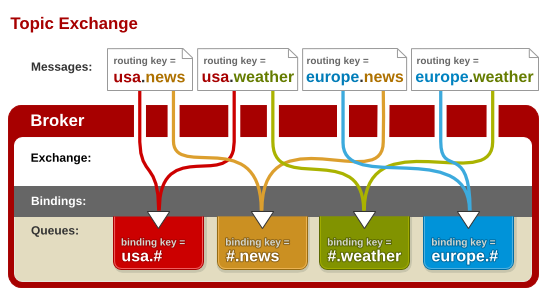


消息中的路由键（routing key）如果和 Binding 中的 binding key 一致， 交换器就将消息发到对应的队列中。路由键与队列名完全匹配，如果一个队列绑定到交换机要求路由键为“dog”，则只转发 routing key 标记为“dog”的消息，不会转发“dog.puppy”，也不会转发“dog.guard”等等。它是完全匹配、单播的模式。

2 fanout（广播式，所有订阅的都能收到）



3 topic（高级模式，模糊匹配模式）



topic 交换器通过模式匹配分配消息的路由键属性，将路由键和某个模式进行匹配，此时队列需要绑定到一个模式上。它将路由键和绑定键的字符串切分成单词，这些单词之间用点隔开。它同样也会识别两个通配符：符号“#”和符号“\*”。#匹配0个或多个单词，\*匹配一个单词。

12 docker上安装rabbitMq，记得安装带management（如rabbitmq：3-management）的这样会有自带的web管理界面

13启动rabitmq

他需要两个端口，分别给rabbitmq和他的监控页面

# docker run -d -p 5672:5672 -p 15672:15672 --name myrabbitmq df80af9ca0c9

启动后可访问他的管理界面：ip：15762

如（192.268.0.101:15672），

默认账号密码为

Username：guest

Password：guest

可以在管理界面里面添加删除交换器和队列，也可以做队列与交换器之间的绑定

springboot整合rabbitmq原理

1、自动配置类为RabbitAutoConfiguration，他为我们配置了

rabbitConnectionFactory：连接rabbitMq：而属性来自于配置文件，故要在配置文件里书写rabbitmq的地址

rabbitTemplate：用于操作rabit的添加消息和接收消息

amqpAdmin：用于对rabbit本身的添加删除交换器和队列，也可以做队列与交换器之间的绑定

2、默认读取的配置文件为RabbitProperties

16、使用rabbitmq

1、在创建项目的时候勾选rabbitmq

2、书写配置文件

**spring.rabbitmq.host**=**192.168.0.101  
spring.rabbitmq.username**=**guest  
spring.rabbitmq.password**=**guest**

1. 使用rabbitTemplaterabbitmq来发送和接收数据  
   @RunWith(SpringRunner.**class**)  
   @SpringBootTest  
   **public class** RabbitmqApplicationTests {  
    @Autowired  
    RabbitTemplate **rabbitTemplate**;  
    @Test  
    **public void** contextLoads() {  
    }  
    */\*  
    \* 单播*

**rabbitTemplate**.convertAndSend传三个参数，交换器名字、路由键、数据对象（Object型，可多样话） *\* \*/* @Test  
 **public void** addmessage2(){  
 **rabbitTemplate**.convertAndSend(**"exchange.direct"**,**"atguigu"**,**new** book(**"红落寞"**,**"曹雪芹"**));  
 }  
 */\*  
 \* 广播  
 \* \*/* @Test  
 **public void** addmessage3(){  
 **rabbitTemplate**.convertAndSend(**"exchange.fanout"**,**"atguigu"**,**new** book(**"红落寞"**,**"曹雪芹"**));  
 }

//接收消息  
 @Test  
 **public void** recivemesage(){  
 Object o = **rabbitTemplate**.receiveAndConvert(**"atguigu"**);//接收**atguigu队列上的消息**  
 System.***out***.println(o.getClass());  
 System.***out***.println(o);  
 }

}

5、rabbitmq默认使用的是jdk的序列化器和转化器，想要转为json可自己网站容器里注入一个json的转换器

@Configuration  
**public class** rabbitmq\_json {  
 @Bean  
 **public** MessageConverter messageConverter(){  
 **return new** Jackson2JsonMessageConverter();  
 }  
}

17、实际开发中如何使用rabbitmq（ @EnableRabbit+ @RabbitListener）

1、在主程序开启基于注解的rabbitmq

@EnableRabbit

2、

如一个销售系统，有下订单，和库存系统，那么可以这样

下订单系统，用 **rabbitTemplate**.convertAndSend向队列存入一个订单

@Test  
**public void** addbook(){

//向**atguigu队列发消息**

**rabbitTemplate**.convertAndSend(**"exchange.direct"**,**"atguigu"**,**new** book(**"红落寞"**,**"曹雪芹"**));  
}

库存系统可以监听相应的队列，进行处理

@RabbitListener：标注在方法上的时候，就会监听相应的队列，当监听到队列有消息的时候会进行消息获取并自动启动执行方法完成业务逻辑

@Service  
**public class** bookservice {  
 @RabbitListener(queues = **"atguigu"**) //监听**atguigu队列**  
 **public void** getmessageandbuy(book book){  
 System.***out***.println(**"收到书本消息"**);  
 System.***out***.println(book);  
 System.***out***.println(**"购买书本。。。。"**);  
 }  
  
}

当addbook向队列发消息时bookservice 就会执行得到如下结果

收到书本消息

book{name='红落寞', author='曹雪芹'}

购买书本。。。。

使用 创建交换器，队列、和绑定关系

**public void** create(){

//创建交换器  
  *amqpAdmin.declareExchange(new DirectExchange("zzx.exchange"));*

*//创建队列  
 amqpAdmin.declareQueue(new Queue("zzx.queue"));*

*//将二者绑定* **amqpAdmin**.declareBinding(**new** Binding(**"zzx.queue"**,Binding.DestinationType.***QUEUE***,**"zzx.queue"**,**"zzx"**,**null**));  
}

Rabbitmq的一些基本的配置

spring.rabbitmq.cache.channel.checkout-timeout 当缓存已满时，获取Channel的等待时间，单位为毫秒

spring.rabbitmq.cache.channel.size 缓存中保持的Channel数量

spring.rabbitmq.listener.acknowledge-mode 容器的acknowledge模式，有自动，手动，不设置spring.rabbitmq.listener.concurrency 同时消费者的最小数量

spring.rabbitmq.listener.default-requeue-rejected 投递失败时是否重新排队 true

spring.rabbitmq.listener.max-concurrency 同时消费者的最大数量

spring.rabbitmq.listener.prefetch 在单个请求中处理的消息个数，他应该大于等于事务数量

spring.rabbitmq.listener.retry.enabled 不论是不是重试的发布 false

spring.rabbitmq.listener.retry.initial-interval 第一次与第二次投递尝试的时间间隔 1000

spring.rabbitmq.listener.retry.max-attempts 尝试投递消息的最大数量 3

spring.rabbitmq.listener.retry.max-interval 两次尝试的最大时间间隔 10000

spring.rabbitmq.listener.retry.multiplier 上一次尝试时间间隔的乘数 1.0

spring.rabbitmq.listener.retry.stateless 不论重试是有状态的还是无状态的 true

spring.rabbitmq.listener.transaction-size 在一个事务中处理的消息数量。为了获得最佳效果，该值应设置为小于等于每个请求中处理的消息个数，即spring.rabbitmq.listener.prefetch的值

spring.rabbitmq.password 登录到RabbitMQ的密码

spring.rabbitmq.port RabbitMQ的端口号 5672

spring.rabbitmq.publisher-confirms 开启Publisher Confirm机制 false

spring.rabbitmq.publisher-returns 开启publisher Return机制 false

spring.rabbitmq.requested-heartbeat 请求心跳超时时间，单位为秒

spring.rabbitmq.template.receive-timeout receive()方法的超时时间 0

spring.rabbitmq.template.reply-timeout sendAndReceive()方法的超时时间 5000

spring.rabbitmq.template.retry.enabled 设置为true的时候RabbitTemplate能够实现重试 false

spring.rabbitmq.template.retry.initial-interval 第一次与第二次发布消息的时间间隔 1000

spring.rabbitmq.template.retry.max-attempts 尝试发布消息的最大数量 3

spring.rabbitmq.template.retry.max-interval 尝试发布消息的最大时间间隔 10000

spring.rabbitmq.template.retry.multiplier 上一次尝试时间间隔的乘数 1.0

如何实现消息的可靠性投递（如何防止消息丢失）

就是说生产者如何保证消息可以百分之百的投递出去

生产者消息发送部分（将消息发送和业务放到一个事务）

业务逻辑数据库

Mq

1 2

定时任务，查询未能收到回调的消息

生产者消息确认

消息数据库

1

3

4

重发

6 5

5

1. 在发送消息前先将生产者端任务的业务逻辑写进数据库，并且将待发送的消息的内容写到消息数据库，在消息上加一个字段，

消息状态，有0未确认，1已确认，2失败。在写入的时候这个状态为0,未确认

超时时间、在插入消息的时候写入超时的时间为当前时间的一分钟之后

重试次数、消息重新投递的次数

1. 将消息投递到mq
2. Mq返回消息确认，
3. 然后生产者段调用回调函数将消息状态改为1已确认
4. 定时任务定时找出消息状态为0并且已经到达超时时间并且重新投递次数小于3的消息，并且然后重新投递，更新超时时间和投递次数，如果投递次数达到3还是失败，将消息状态设置为2（失败），等待后续人工处理
5. 重新发送消息

如何防止消息被重复消费

对于生产者端可能会出现重复投递的消息，也就是说可能会出现消息同时投递两次，那么消费者如何防止消息被消费两次

对于这个问题其实是一个幂等性的问题，解决这种问题采用的有

可以给等待消费的数据加上唯一的key，当要消费这条数据的时候先去redis里面或者去数据库里面查找这条数据是否存在，如果存在则不再消费或者更新一下，不存在则消费并且将消息的信息更新进入redis或者数据库

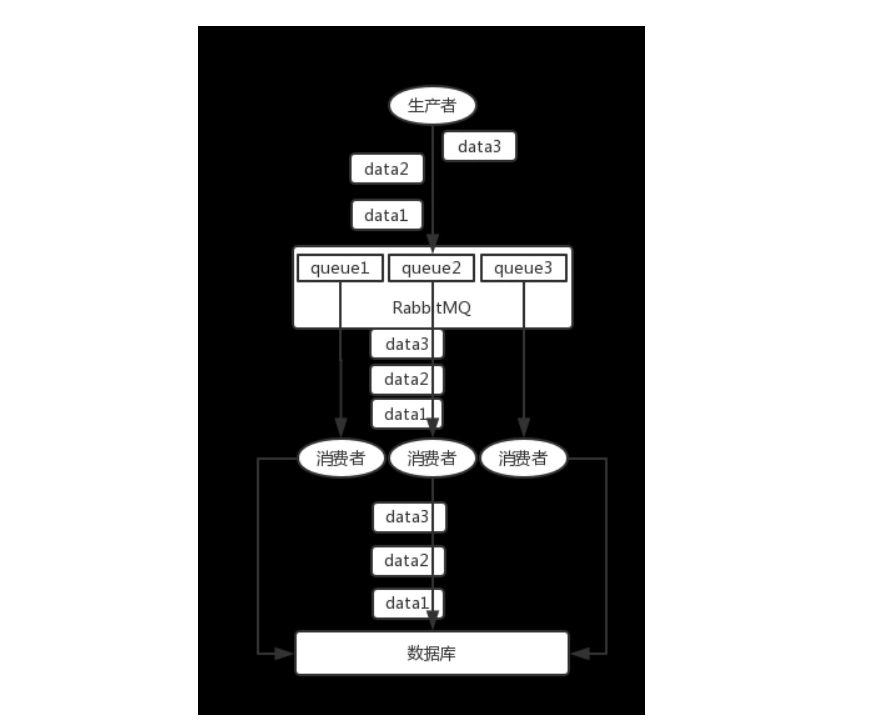
如何保证消息的顺序性

有些时候消息需要保证顺序，则解决方法有

1就一个队列一个消费者

2一个队列一个消费者，消费者拿到消息后按照一定的规则进行排序，然后进行消费

3多个队列多个消费者，但是一个队列对应一个消费者，生产者将消息顺序放入一个队列，然后消费者消费消息



在消费者集群的时候消费者采用均摊消费的原则

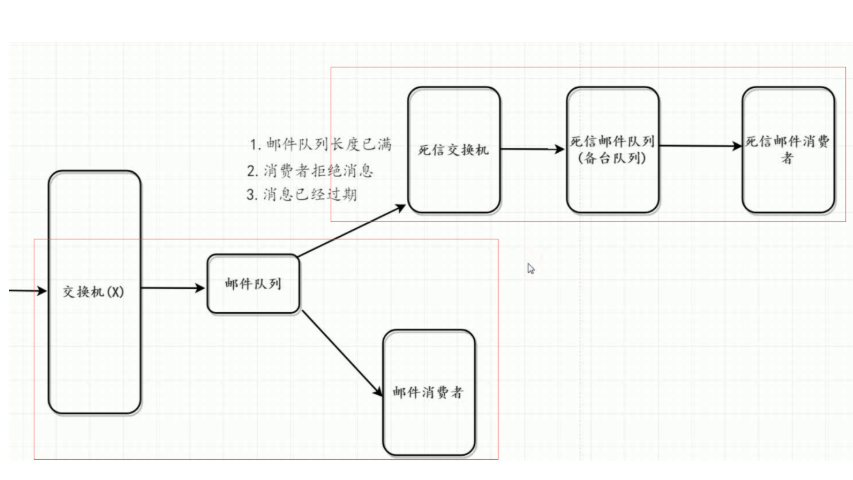
即如果有两个消费者他们消费的消息的数目是相同的，一个消费13579一个243810。当然也可以设置其他消费的模式。

均摊消费是有问题的，比如两个消费者的消息处理速度不一样，这时候就需要公平消费。

公平消费:就是根据消费者的消费速度进消费消息

实现方式：采用手动ack的模式，就是消费者消费完消息返回ack确认才分配下一条消息，这样就能消费者的速度进行消费

死信队列

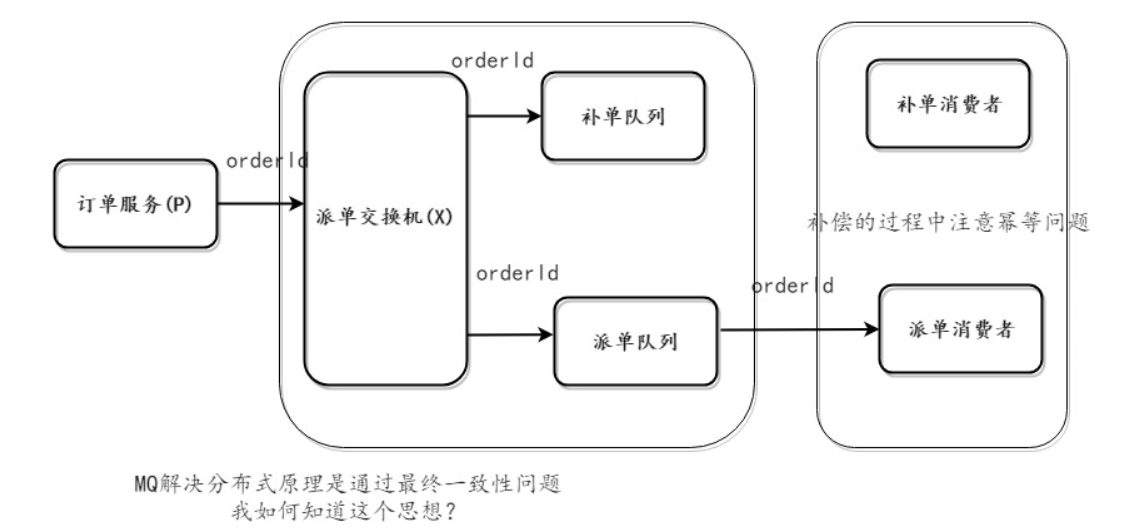


死信队列类似于备胎，当出现队列已经满，消费者拒绝消费消息，消息过期等情况的时候，将消息投入到死信队列去处理，死信队列也有对应的交换机和消费者去处理里面的消息

可以查看文章<https://www.cnblogs.com/xinruyi/p/11220525.html>

Rabbitmq简单的分布式事务

Rabbitmq的分布式事务采取最终一致性来进行



1. 确认生产者一定要将数据投递到MQ服务器中（采用MQ消息确认机制）

采用上面的消息投递机制保证消息一定投递到mq中

1. MQ消费者消息能够正确消费消息，采用手动ACK模式（注意重试幂等性问题）

3、如何保证第一个事务先执行，采用补偿机制，在创建一个补单队列和补单消费者进行监听，如果订单没有创建成功，进行补单。

处理方式如下

1订单服务创建订单并且采用conform机制保证将消息投递到mq

2mq将消息同时投递到派单队列和补单队列

3派单队列将消息派发给消费者消费，消费失败则重试，不行则人工解决

4补单队列查看订单是否创建成功，即看一下订单事务会不会回滚，如果创建成功则直接ack，不成功则补充创建订单，然后ack，从而达到两个服务的最终一致性

Cvte的消息推动平台采用的就是上面的原理