



RabbitMQ

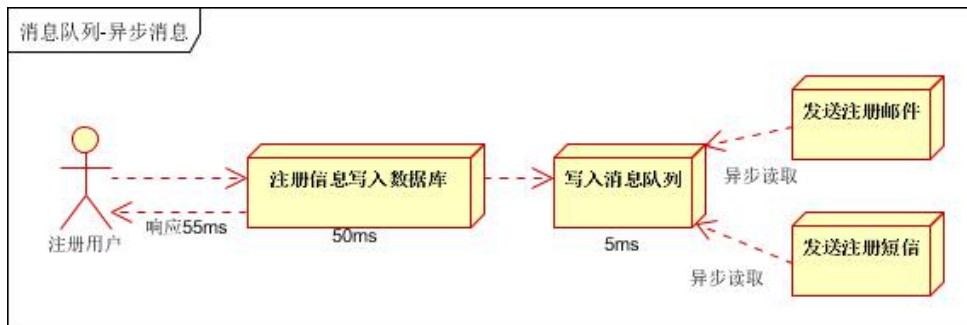
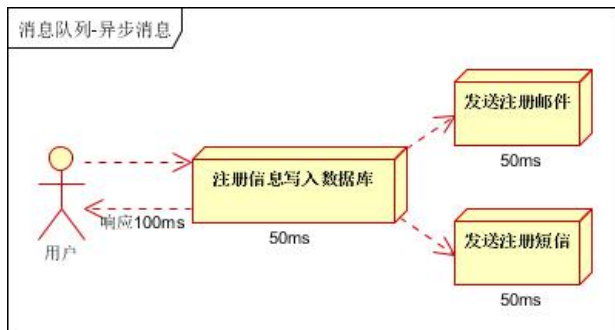
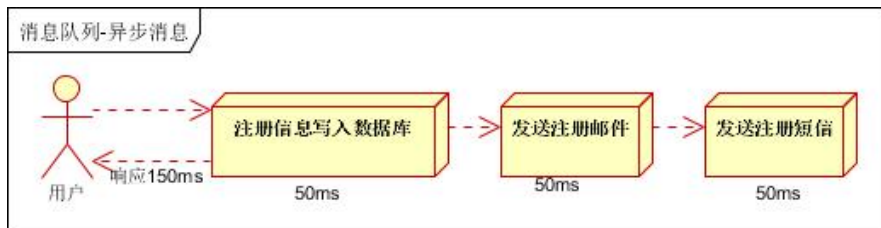
Message Queue消息队列



消息中间件

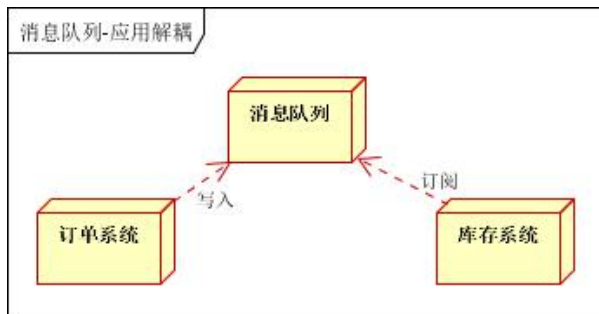
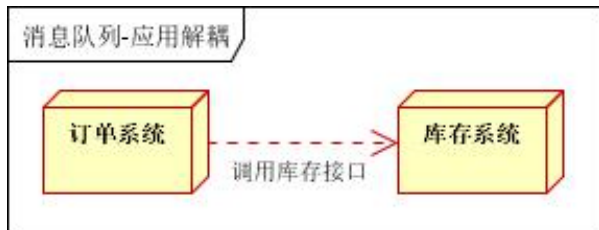


异步处理

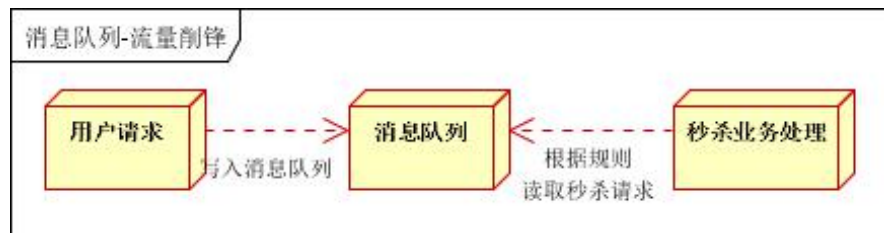




应用解耦



流量控制





1. 大多应用中，可通过消息服务中间件来提升系统异步通信、扩展解耦能力
2. 消息服务中两个重要概念：

消息代理 (message broker) 和目的地 (destination)

当消息发送者发送消息以后，将由消息代理接管，消息代理保证消息传递到指定目的地。

3. 消息队列主要有两种形式的目的地
 1. 队列 (queue)：点对点消息通信 (point-to-point)
 2. 主题 (topic)：发布 (publish) / 订阅 (subscribe) 消息通信



4. 点对点式:

- 消息发送者发送消息, 消息代理将其放入一个队列中, 消息接收者从队列中获取消息内容, 消息读取后被移出队列
- 消息只有唯一的发送者和接受者, 但并不是说只能有一个接收者

5. 发布订阅式:

- 发送者 (发布者) 发送消息到主题, 多个接收者 (订阅者) 监听 (订阅) 这个主题, 那么就会在消息到达时同时收到消息

6. JMS (Java Message Service) JAVA消息服务:

- 基于JVM消息代理的规范。ActiveMQ、HornetMQ是JMS实现

7. AMQP (Advanced Message Queuing Protocol)

- 高级消息队列协议, 也是一个消息代理的规范, 兼容JMS
- RabbitMQ是AMQP的实现



	JMS (Java Message Service)	AMQP (Advanced Message Queuing Protocol)
定义	Java api	网络线级协议
跨语言	否	是
跨平台	否	是
Model	提供两种消息模型： (1)、Peer-2-Peer (2)、Pub/sub	提供了五种消息模型： (1)、direct exchange (2)、fanout exchange (3)、topic change (4)、headers exchange (5)、system exchange 本质来讲，后四种和JMS的pub/sub模型没有太大差别， 仅是在路由机制上做了更详细的划分；
支持消息类型	多种消息类型： TextMessage MapMessage BytesMessage StreamMessage ObjectMessage Message（只有消息头和属性）	byte[] 当实际应用时，有复杂的消息，可以将消息序列化后发送。
综合评价	JMS 定义了JAVA API层面的标准；在java体系中，多个client均可以通过JMS进行交互，不需要应用修改代码，但是其对跨平台的支持较差；	AMQP定义了wire-level层的协议标准；天然具有跨平台、跨语言特性。



8. 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**

9. Spring Boot自动配置

- **JmsAutoConfiguration**
- **RabbitAutoConfiguration**
- **10、市面的MQ产品**
 - **ActiveMQ、RabbitMQ、RocketMQ、Kafka**



RabbitMQ简介:

RabbitMQ是一个由erlang开发的AMQP(Advanced Message Queue Protocol)的开源实现。

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。

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。

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。

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。



Consumer

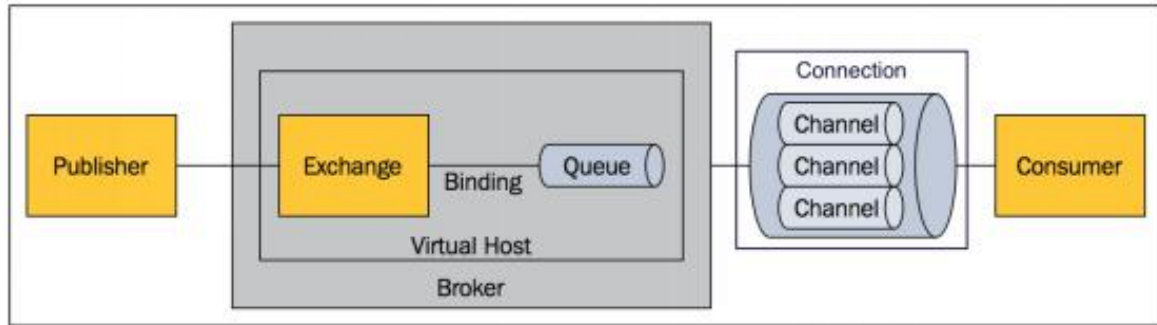
消息的消费者，表示一个从消息队列中取得消息的客户端应用程序。

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 /。

Broker

表示消息队列服务器实体





```
docker run -d --name rabbitmq -p 5671:5671 -p 5672:5672 -p 4369:4369 -p  
25672:25672 -p 15671:15671 -p 15672:15672 rabbitmq:management
```

4369, 25672 (Erlang发现&集群端口)

5672, 5671 (AMQP端口)

15672 (web管理后台端口)

61613, 61614 (STOMP协议端口)

1883, 8883 (MQTT协议端口)

<https://www.rabbitmq.com/networking.html>



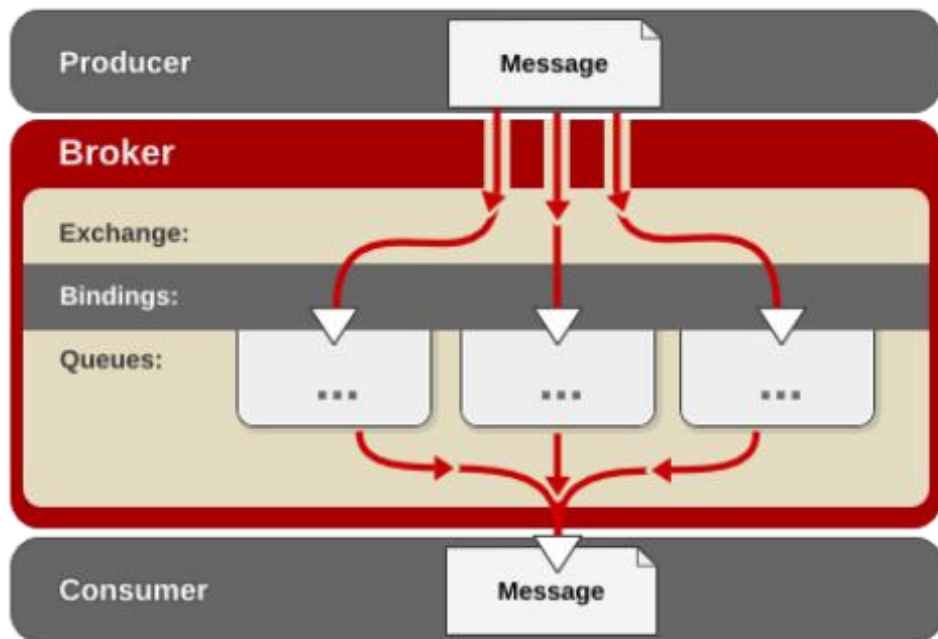
- 4369: [epmd](#), a peer discovery service used by RabbitMQ nodes and CLI tools
- 5672, 5671: used by AMQP 0-9-1 and 1.0 clients without and with TLS
- 25672: used for inter-node and CLI tools communication (Erlang distribution server port) and is allocated from a dynamic range (limited to a single port by default, computed as AMQP port + 20000). Unless external connections on these ports are really necessary (e.g. the cluster uses [federation](#) or CLI tools are used on machines outside the subnet), these ports should not be publicly exposed. See [networking guide](#) for details.
- 35672-35682: used by CLI tools (Erlang distribution client ports) for communication with nodes and is allocated from a dynamic range (computed as server distribution port + 10000 through server distribution port + 10010). See [networking guide](#) for details.
- 15672: [HTTP API](#) clients, [management UI](#) and [rabbitmqadmin](#) (only if the [management plugin](#) is enabled)
- 61613, 61614: [STOMP clients](#) without and with TLS (only if the [STOMP plugin](#) is enabled)
- 1883, 8883: ([MQTT clients](#) without and with TLS, if the [MQTT plugin](#) is enabled)
- 15674: STOMP-over-WebSockets clients (only if the [Web STOMP plugin](#) is enabled)
- 15675: MQTT-over-WebSockets clients (only if the [Web MQTT plugin](#) is enabled)
- 15692: Prometheus metrics (only if the [Prometheus plugin](#) is enabled)



AMQP 中的消息路由

- AMQP 中消息的路由过程和 Java 开发者熟悉的 JMS 存在一些差别，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。

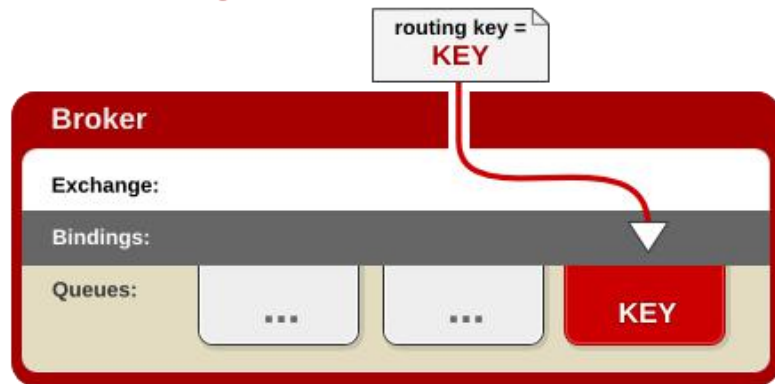
Producer Consumer





- **Exchange**分发消息时根据类型的不同分发策略有区别，目前共四种类型：**direct**、**fanout**、**topic**、**headers**。headers 匹配 AMQP 消息的 header 而不是路由键，headers 交换器和 direct 交换器完全一致，但性能差很多，目前几乎用不到了，所以直接看另外三种类型：

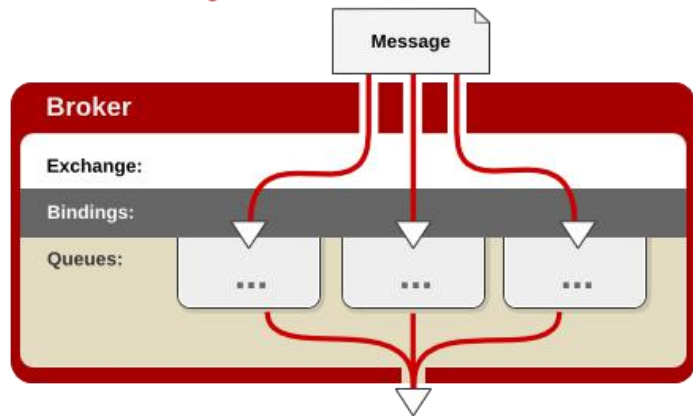
Direct Exchange



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。

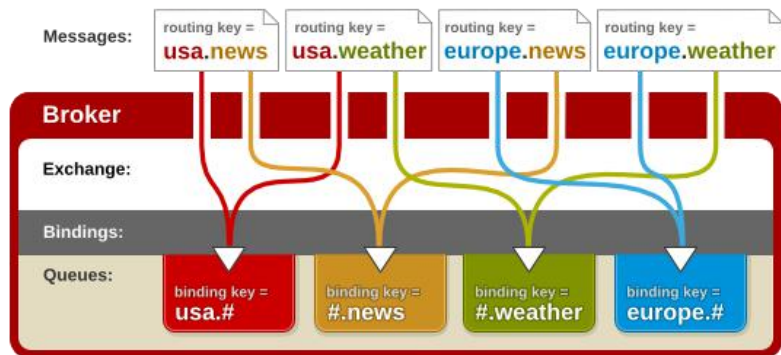


Fanout Exchange

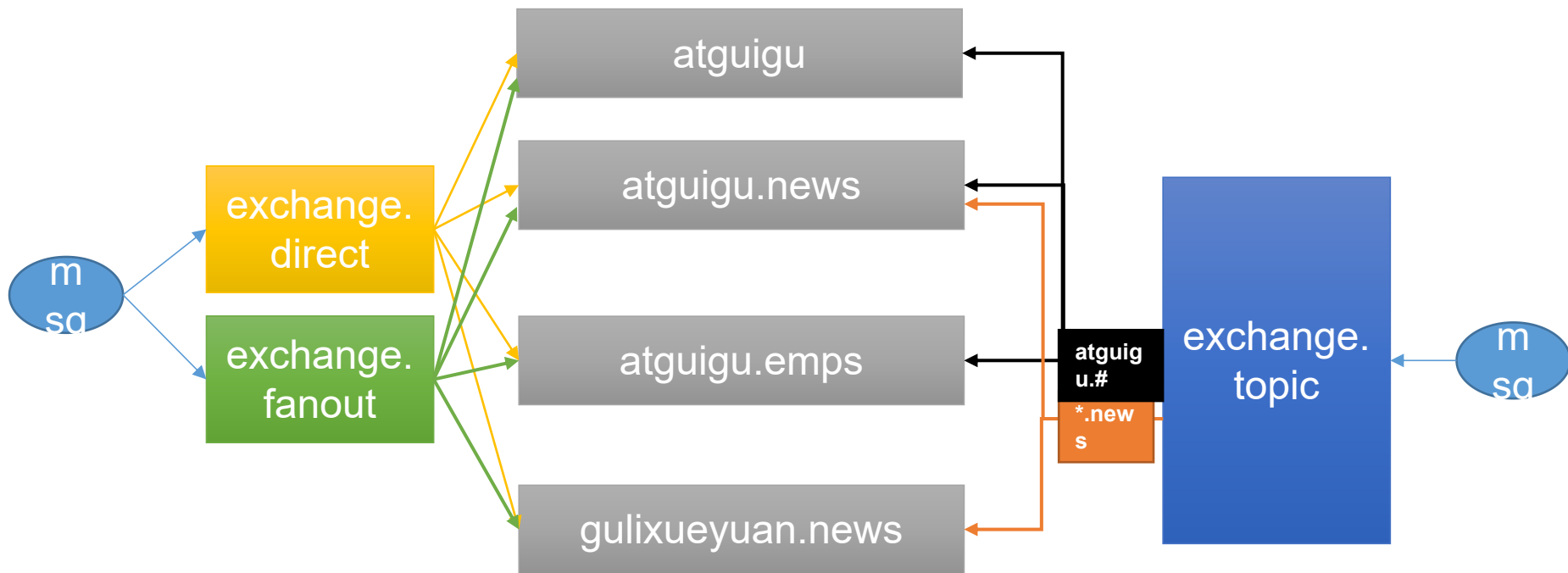


每个发到 fanout 类型交换器的消息都会分到所有绑定的队列上去。fanout 交换器不处理路由键，只是简单的将队列绑定到交换器上，每个发送到交换器的消息都会被转发到与该交换器绑定的所有队列上。很像子网广播，每台子网内的主机都获得了一份复制的消息。fanout 类型转发消息是最快的。

Topic Exchange



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。



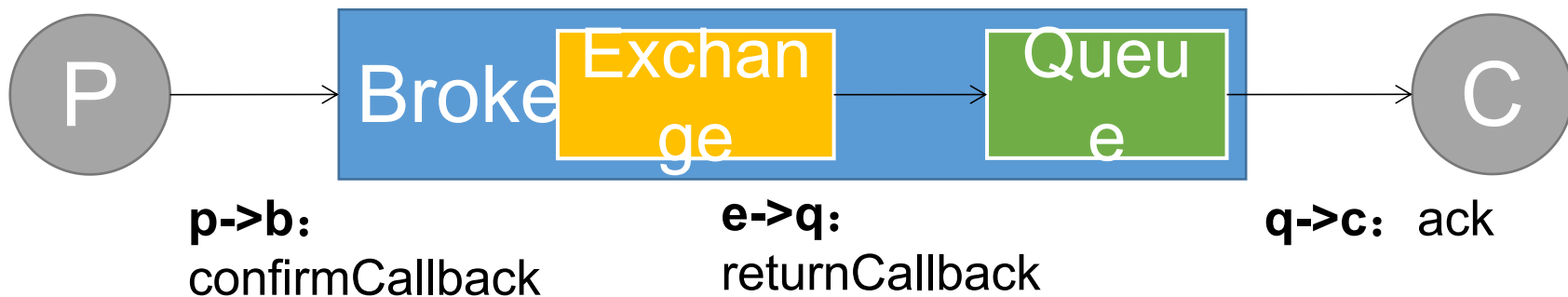


1. **引入** spring-boot-starter-amqp
2. **application.yml配置**
3. **测试RabbitMQ**
 1. **AmqpAdmin: 管理组件**
 2. **RabbitTemplate: 消息发送处理组件**
 3. **@RabbitListener** 监听消息的方法可以有三种参数（不分数量，顺序）
 - Object content, Message message, Channel channel



六、RabbitMQ消息确认机制-可靠抵达

- 保证消息不丢失，可靠抵达，可以使用事务消息，性能下降250倍，为此引入确认机制
- **publisher** `confirmCallback` 确认模式
- **publisher** `returnCallback` 未投递到 queue 退回模式
- **consumer** `ack`机制





- `spring.rabbitmq.publisher-confirms=true`
 - 在创建 `connectionFactory` 的时候设置 `PublisherConfirms(true)` 选项，开启 `confirmcallback`。
 - `CorrelationData`：用来表示当前消息唯一性。
 - 消息只要被 broker 接收到就会执行 `confirmCallback`，如果是 cluster 模式，需要所有 broker 接收到才会调用 `confirmCallback`。
 - 被 broker 接收到只能表示 message 已经到达服务器，并不能保证消息一定会被投递到目标 queue 里。所以需要用到接下来的 `returnCallback`。



- `spring.rabbitmq.publisher-returns=true`
- `spring.rabbitmq.template.mandatory=true`
 - `confirm` 模式只能保证消息到达 broker, 不能保证消息准确投递到目标 queue 里。在有些业务场景下, 我们需要保证消息一定要投递到目标 queue 里, 此时就需要用到 `return` 退回模式。
 - 这样如果未能投递到目标 queue 里将调用 `returnCallback`, 可以记录下详细到投递数据, 定期的巡检或者自动纠错都需要这些数据。



- 消费者获取到消息，成功处理，可以回复Ack给Broker
 - `basic.ack`用于肯定确认；broker将移除此消息
 - `basic.nack`用于否定确认；可以指定broker是否丢弃此消息，可以批量
 - `basic.reject`用于否定确认；同上，但不能批量
- 默认自动ack，消息被消费者收到，就会从broker的queue中移除
- queue无消费者，消息依然会被存储，直到消费者消费
- 消费者收到消息，默认会自动ack。但是如果无法确定此消息是否被处理完成，或者成功处理。我们可以开启手动ack模式
 - 消息处理成功，`ack()`，接受下一个消息，此消息broker就会移除
 - 消息处理失败，`nack()/reject()`，重新发送给其他人进行处理，或者容错处理后ack
 - 消息一直没有调用ack/nack方法，broker认为此消息正在被处理，不会投递给别人，此时客户端断开，消息不会被broker移除，会投递给别人



场景：

比如未付款订单，超过一定时间后，系统自动取消订单并释放占有物品。

常用解决方案：

spring的 schedule 定时任务轮询数据库

缺点：

消耗系统内存、增加了数据库的压力、存在较大的时间误差

解决： rabbitmq的消息TTL和死信Exchange结合



- 消息的TTL就是消息的存活时间。
- RabbitMQ可以对队列和消息分别设置TTL。
 - 对队列设置就是队列没有消费者连着的保留时间，也可以对每一个单独的消息做单独的设置。超过了这个时间，我们认为这个消息就死了，称之为死信。
 - 如果队列设置了，消息也设置了，那么会取小的。所以一个消息如果被路由到不同的队列中，这个消息死亡的时间有可能不一样（不同的队列设置）。这里单讲单个消息的TTL，因为它才是实现延迟任务的关键。可以通过设置消息的expiration字段或者x-message-ttl属性来设置时间，两者是一样的效果。



- 一个消息在满足如下条件下，会进死信路由，记住这里是路由而不是队列，一个路由可以对应很多队列。（什么是死信）
 - 一个消息被Consumer拒收了，并且reject方法的参数里requeue是false。也就是说不会被再次放在队列里，被其他消费者使用。 (`basic.reject/ basic.nack`) `requeue=false`
 - 上面的消息的TTL到了，消息过期了。
 - 队列的长度限制满了。排在前面的消息会被丢弃或者扔到死信路由上
- Dead Letter Exchange其实就是一种普通的exchange，和创建其他exchange没有两样。只是在某一个设置Dead Letter Exchange的队列中有消息过期了，会自动触发消息的转发，发送到Dead Letter Exchange中去。
- 我们既可以控制消息在一段时间后变成死信，又可以控制变成死信的消息被路由到某一个指定的交换机，结合二者，其实就可以实现一个延时队列

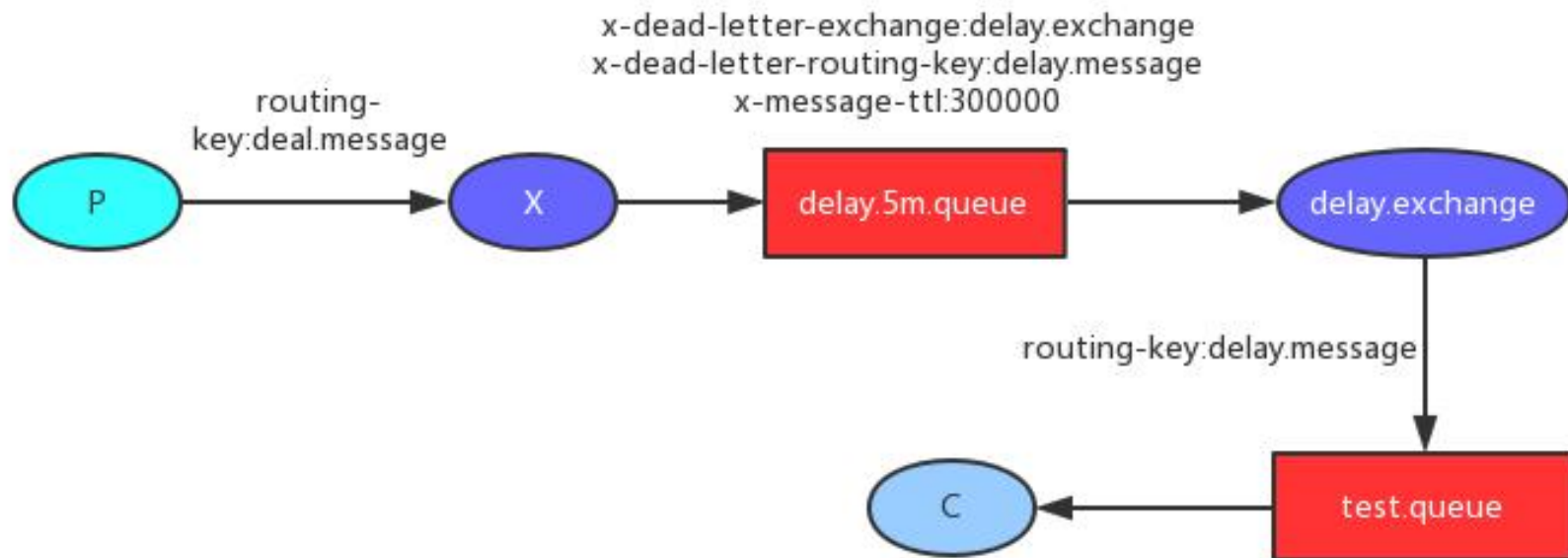


- 手动ack&异常消息统一放在一个队列处理建议的两种方式
 - catch异常后，**手动发送到指定队列**，然后使用channel给rabbitmq确认消息已消费
 - 给Queue绑定死信队列，使用nack (requeue为false) 确认消息消费失败



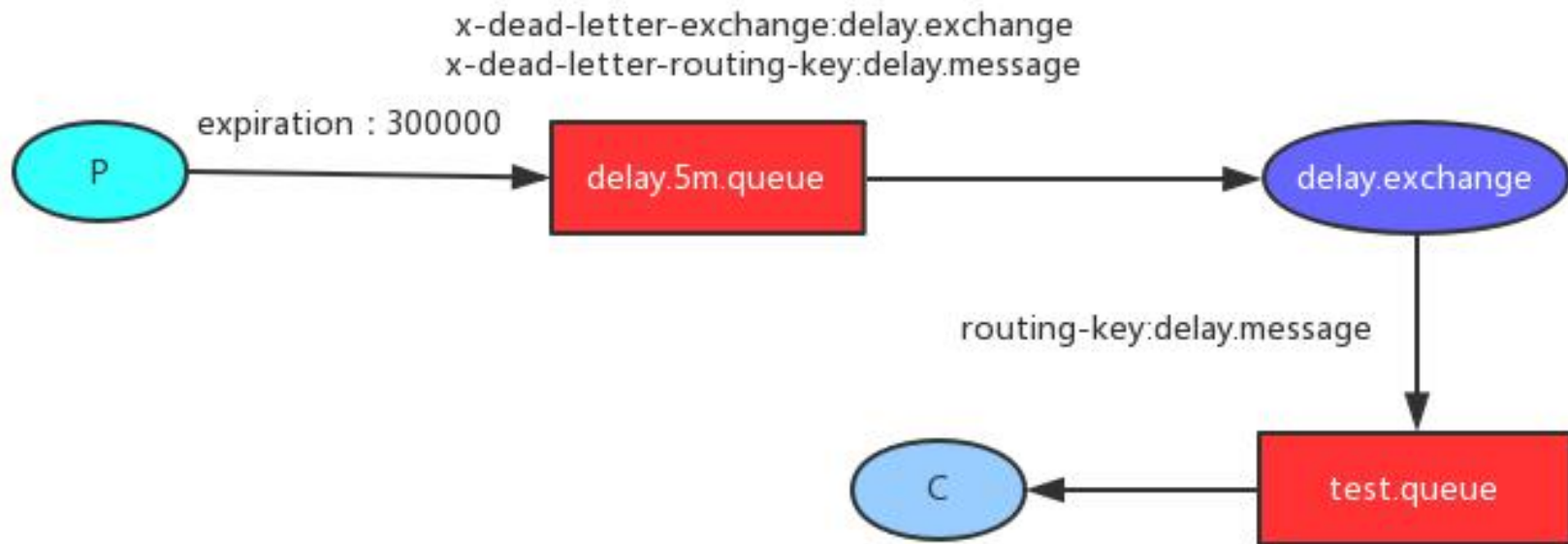


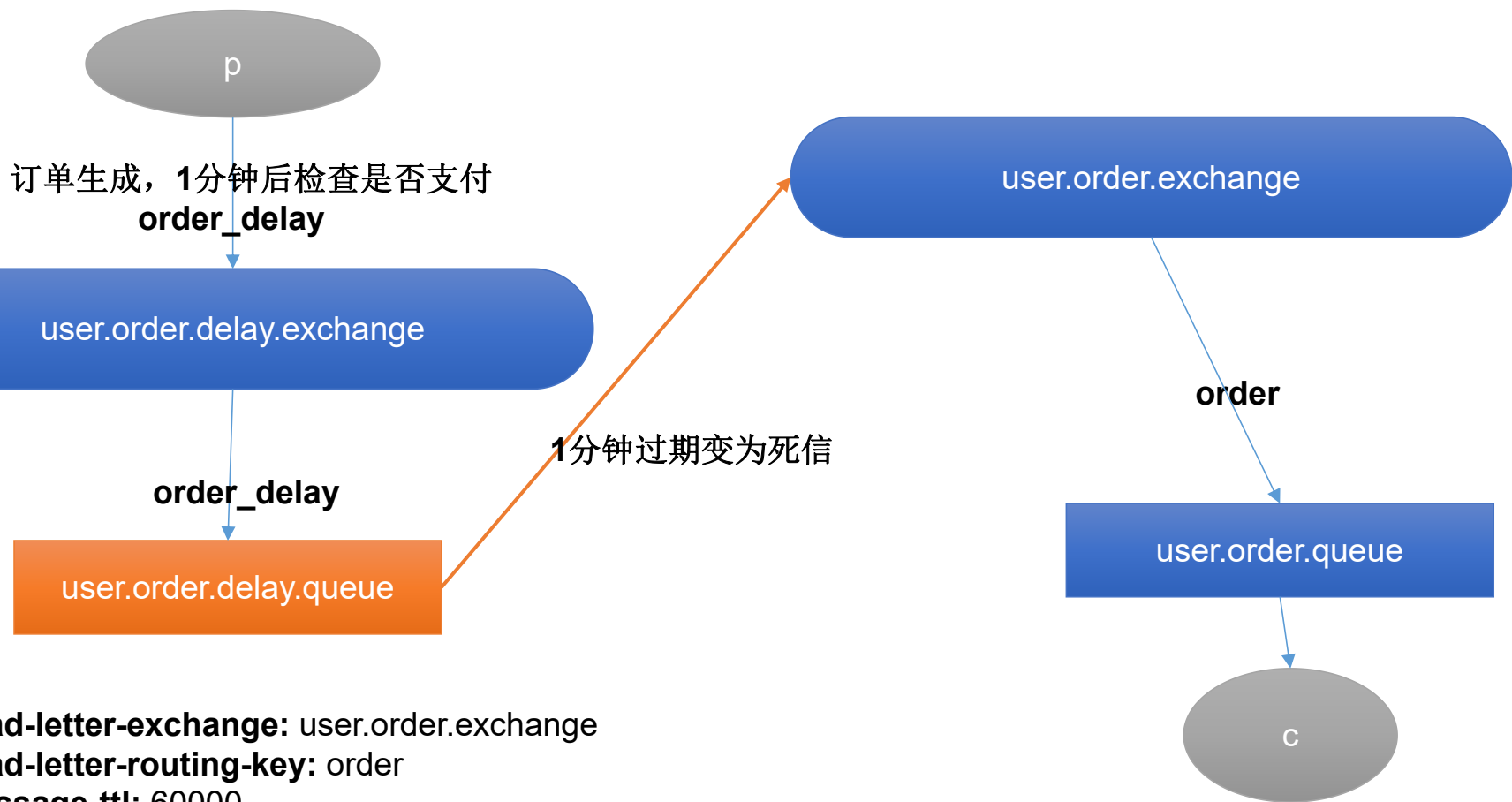
设置队列过期时间实现延时队列



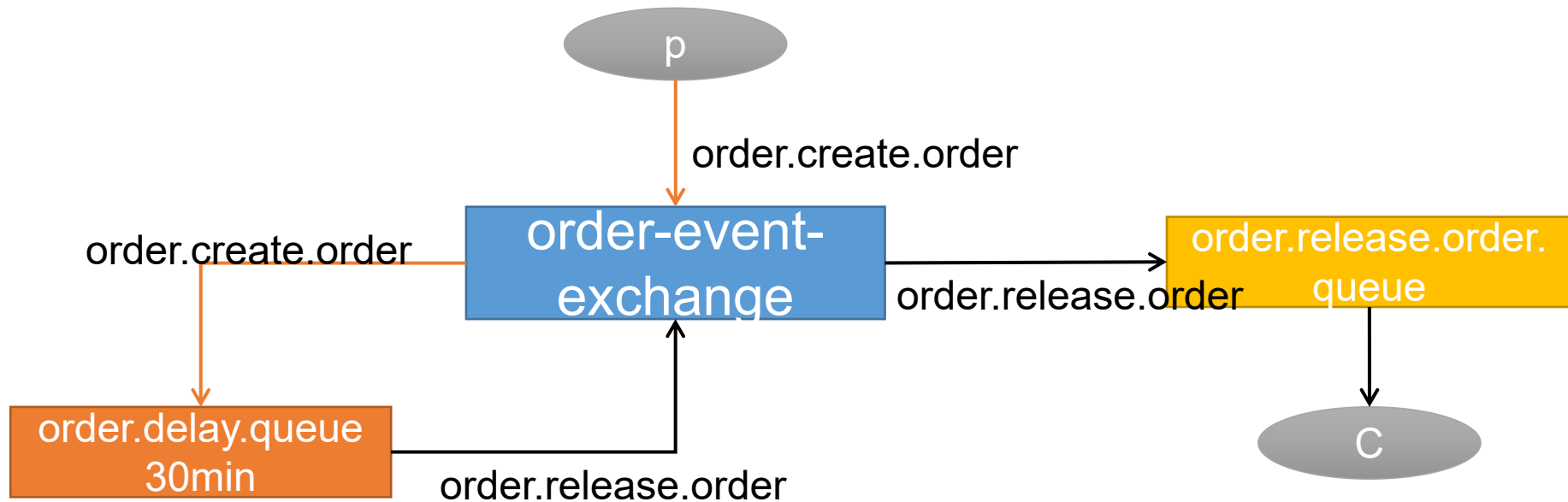


设置消息过期时间实现延时队列





x-dead-letter-exchange: user.order.exchange
x-dead-letter-routing-key: order
x-message-ttl: 60000



x-dead-letter-exchange: order-event-exchange

x-dead-letter-routing-key: order.release.order

x-message-ttl: 60000



- 1、Queue、Exchange、Binding可以@Bean进去
- 2、监听消息的方法可以有三种参数（不分数量，顺序）
 - Object content, Message message, Channel channel
- 3、channel可以用来拒绝消息，否则自动ack；



- 1、消息丢失
 - 消息发送出去，由于网络问题没有抵达服务器
 - 做好容错方法（try-catch），发送消息可能会网络失败，失败后要有重试机制，可记录到数据库，采用定期扫描重发的方式
 - 做好日志记录，每个消息状态是否都被服务器收到都应该记录
 - 做好定期重发，如果消息没有发送成功，定期去数据库扫描未成功的消息进行重发
 - 消息抵达Broker，Broker要将消息写入磁盘（持久化）才算成功。此时Broker尚未持久化完成，宕机。
 - publisher也必须加入确认回调机制，确认成功的消息，修改数据库消息状态。
 - 自动ACK的状态下。消费者收到消息，但没来得及消息然后宕机
 - 一定开启手动ACK，消费成功才移除，失败或者没来得及处理就noAck并重新入队



• 2、消息重复

- 消息消费成功，事务已经提交，ack时，机器宕机。导致没有ack成功，Broker的消息重新由unack变为ready，并发送给其他消费者
- 消息消费失败，由于重试机制，自动又将消息发送出去
- 成功消费，ack时宕机，消息由unack变为ready，Broker又重新发送
 - 消费者的业务消费接口应该设计为**幂等性的**。比如扣库存有工作单的状态标志
 - 使用**防重表**（redis/mysql），发送消息每一个都有业务的唯一标识，处理过就不用处理
 - rabbitMQ的每一个消息都有redelivered字段，可以获取**是否是被重新投递过来的**，而不是第一次投递过来的



- 3、消息积压
 - 消费者宕机积压
 - 消费者消费能力不足积压
 - 发送者发送流量太大
 - 上线更多的消费者，进行正常消费
 - 上线专门的队列消费服务，将消息先批量取出来，记录数据库，离线慢慢处理



- ActiveMQ、 RabbitMQ、 RocketMQ、 Kafka



谢谢观看