**RabbitMQ**

# 消息中间件的定义

其实并没有标准定义。一般认为，消息中间件属于分布式系统中一个子系统，关注于数据的发送和接收，利用高效可靠的异步消息传递机制对分布式系统中的其余各个子系统进行集成。

消息中间件就是消息的搬运工，接收生产者的消息，将消息发送给消费者。

# 为什么要使用消息中间件？

* 解耦
* 异步处理
* 缓冲能力
* 伸缩性:可以不断添加MQ来处理更多的请求。
* 扩展性

# 与RPC的区别

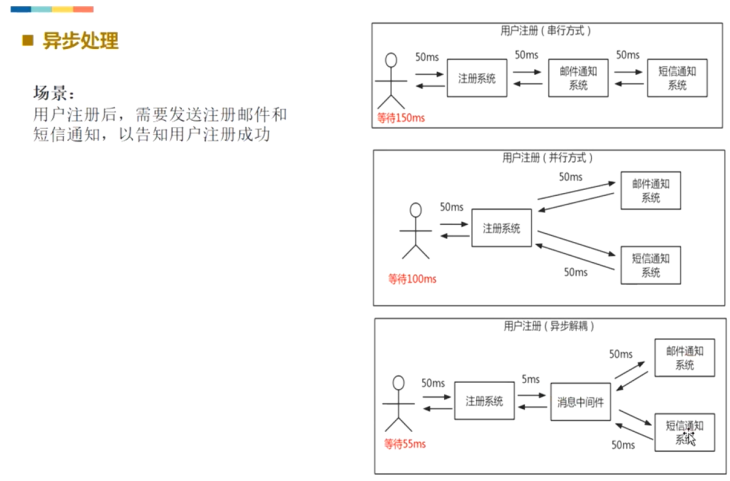
消息中间件是异步的，RPC一般是同步的。

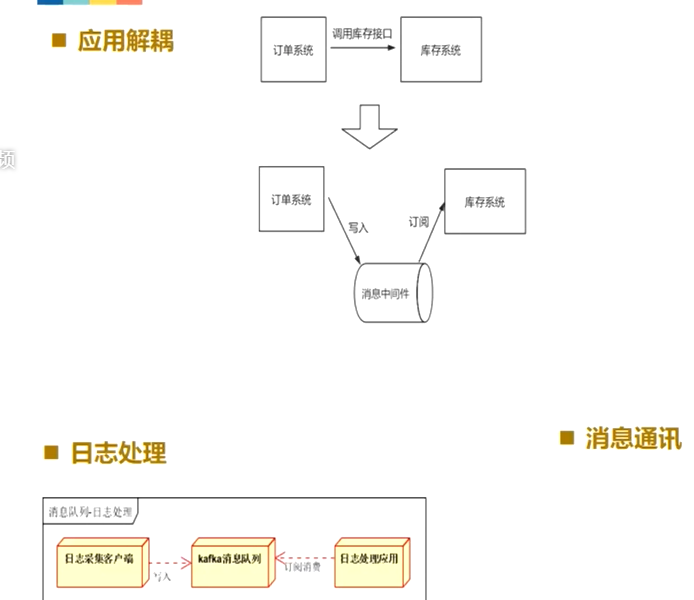
消息中间件是一个松耦合的架构。

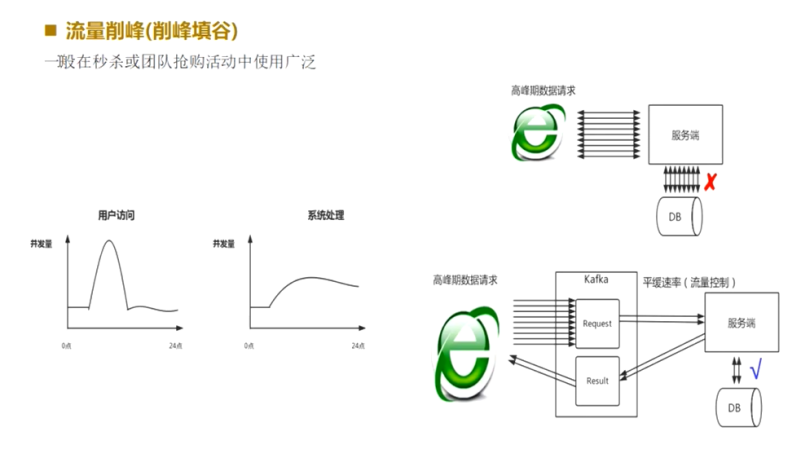
RPC实现服务的调用，调用者需要知道被调用者的接口，需要依赖接口，消息中间件中生产者完全不需要知道消费者的存在。

MQ 是生产者消费者模式。  
RPC 是请求响应模式。  
MQ 是面向数据的。  
RPC 是面向动作的。

# 消息中间件使用场景



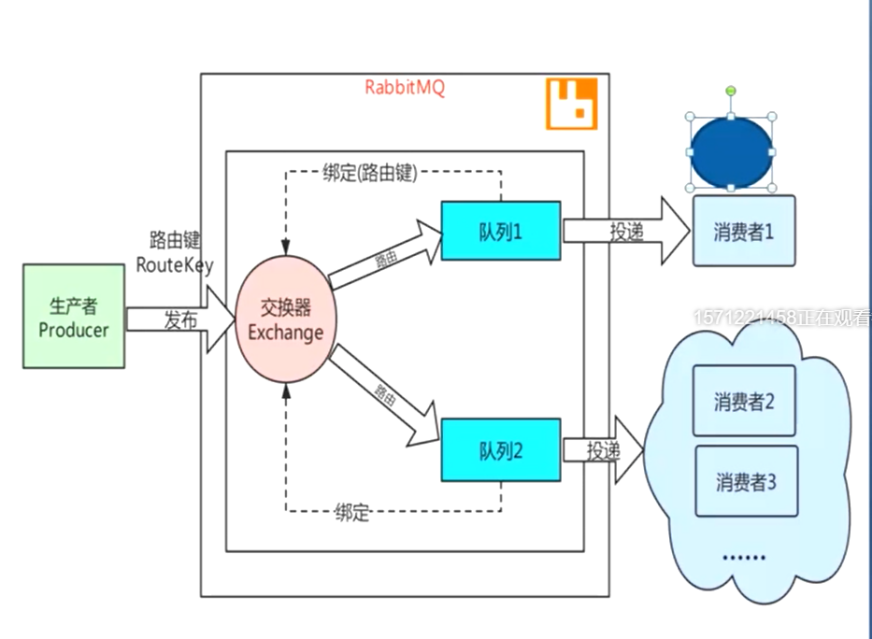




# 如何选择消息中间件



# 内部构造



# 交换器的种类

## Direct exchange（直连交换机）

键必须完全匹配。

## Fanout exchange（扇型交换机）

和路由键没有任何关系。

## Topic exchange（主题交换机）

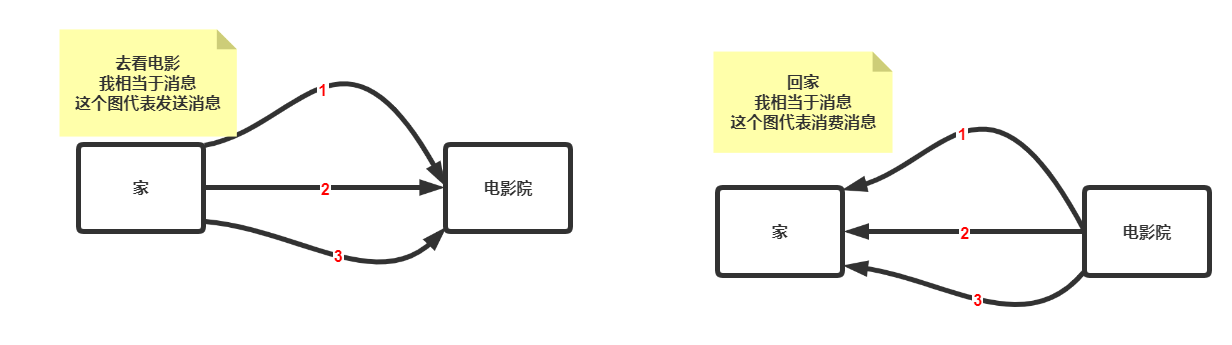
\*匹配一个 cart.info cart.# cart.\*

#可以匹配多个 cart.info.xxx cart.#可以匹配，cart.\*不可以匹配

## Headers exchange（头交换机）

# 路由键

我的观点：路由键类似于我从家到电影院看电影，可以有多条路，我可以选择任何一条路去电影院，同样的，从交换机到队列也可以有多种发送途径，可以选择任意一条途径，接收消息也可以只选择从某条路径发送过来的。



# 消息发送时的权衡

失败重试

## 发送方确认

一般确认、批量确认。

启用发送者确认模式

|  |
| --- |
| channel.confirmSelect(); |

批量确认：

|  |
| --- |
| channel.waitForConfirmsOrDie(); |

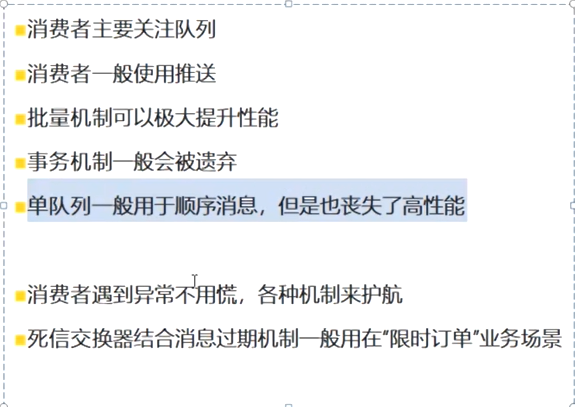
添加消息确认监听器：

|  |
| --- |
| *// 添加消息确认监听器* channel.addConfirmListener(**new** ConfirmListener() {  **public void** handleAck(**long** deliveryTag, **boolean** multiple) **throws** IOException {  System.***out***.println(**"ack message: deliveryTag = "** + deliveryTag + **", multiple = "** + multiple);  }   **public void** handleNack(**long** deliveryTag, **boolean** multiple) **throws** IOException {  System.***out***.println(**"nack message: deliveryTag = "** + deliveryTag + **", multiple = "** + multiple);  } }); |

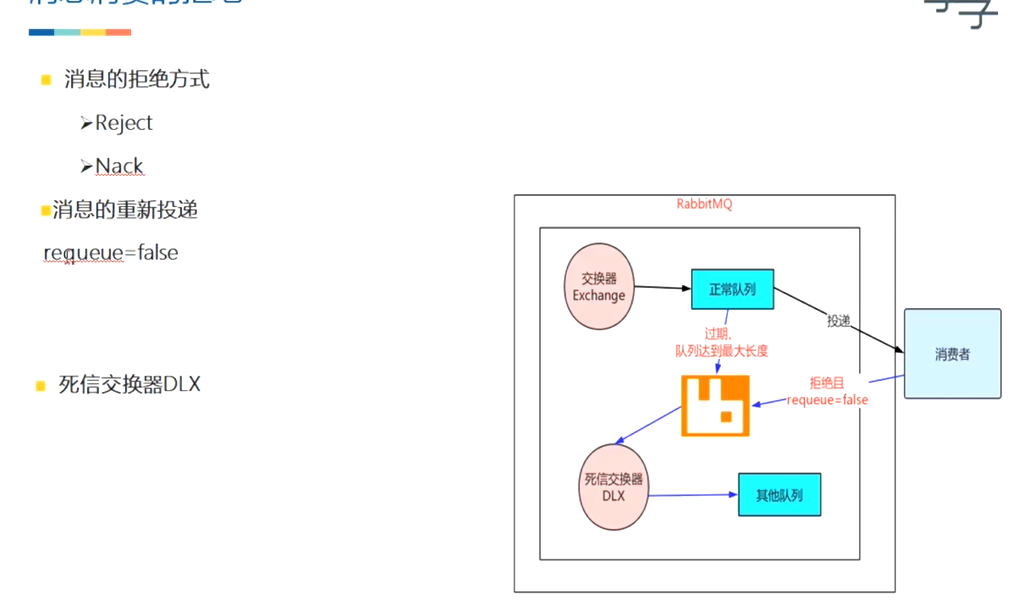
### 备用交换器



## 消费者消费时的权衡



# 消息消费的拒绝



# 队列

## 临时队列

自动删除的判断条件：消费者断开

消费者独占：



过期自动删除：



## 持久队列

# 消息的属性

