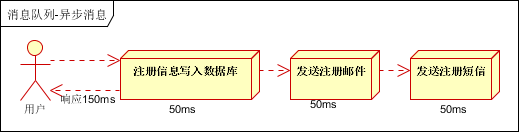
**RabbitMQ**

**重点掌握 消息队列应用场景 几种工作模式**

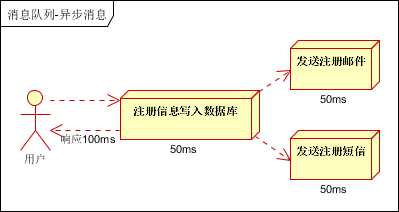
# 消息队列应用场景

## 任务异步处理

场景说明：用户注册后，需要发注册邮件和注册短信。传统的做法有两种 1.串行的方式；2.并行方式



  a、串行方式：将注册信息写入数据库成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后，返回给客户端。

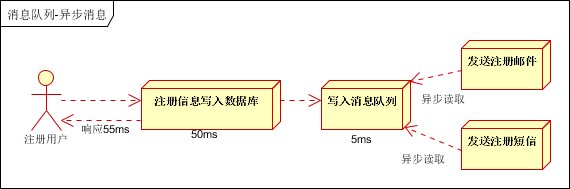


b、并行方式：将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。与串行的差别是，并行的方式可以提高处理的时间

假设三个业务节点每个使用50毫秒钟，不考虑网络等其他开销，则串行方式的时间是150毫秒，并行的时间可能是100毫秒。

因为CPU在单位时间内处理的请求数是一定的，假设CPU1秒内吞吐量是100次。则串行方式1秒内CPU可处理的请求量是7次（1000/150）。并行方式处理的请求量是10次（1000/100）

小结：如以上案例描述，传统的方式系统的性能（并发量，吞吐量，响应时间）会有瓶颈。



引入消息队列，将不是必须的业务逻辑，异步处理。

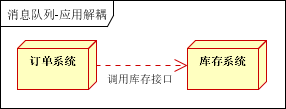
按照以上约定，用户的响应时间相当于是注册信息写入数据库的时间，也就是50毫秒。注册邮件，发送短信写入消息队列后，直接返回，因此写入消息队列的速度很快，基本可以忽略，因此用户的响应时间可能是50毫秒。因此架构改变后，系统的吞吐量提高到每秒20 QPS。比串行提高了3倍，比并行提高了两倍。

## 应用解耦

耦合：应用的相关性

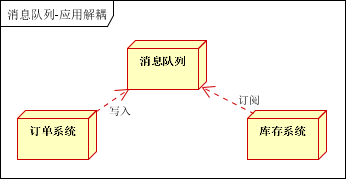
强耦合：应用之间具有很强的相关性

场景说明：用户下单后，订单系统需要通知库存系统。传统的做法是，订单系统调用库存系统的接口。如下图：



传统模式的缺点：假如库存系统无法访问，则订单减库存将失败，从而导致订单失败，订单系统与库存系统耦合

引入应用消息队列后的方案，如下图：



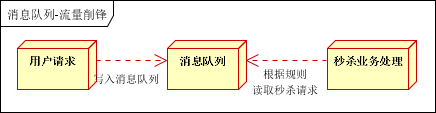
订单系统：用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功

库存系统：订阅下单的消息，采用拉/推的方式，获取下单信息，库存系统根据下单信息，进行库存操作

假如：在下单时库存系统不能正常使用。也不影响正常下单，因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。实现订单系统与库存系统的应用解耦

## 流量削锋

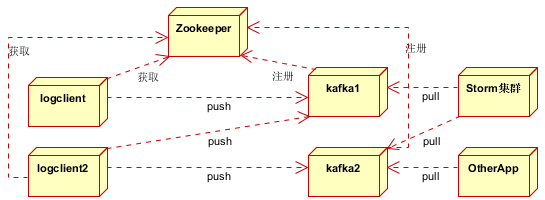
应用场景：秒杀活动，一般会因为流量过大，导致流量暴增，应用挂掉。为解决这个问题，一般需要在应用前端加入消息队列。可以缓解短时间内高流量压垮应用。



秒杀业务根据消息队列中的请求信息，再做后续处理

## 日志处理

日志处理是指将消息队列用在日志处理中，比如Kafka的应用，解决大量日志传输的问题。架构简化如下



日志采集客户端，负责日志数据采集，定时写受写入Kafka队列

Kafka消息队列，负责日志数据的接收，存储和转发

日志处理应用：订阅并消费kafka队列中的日志数据

# 各种消息队列产品

　（1）ActiveMQ

　　ActiveMQ 是Apache出品，最流行的，能力强劲的开源消息总线。ActiveMQ 是一个完全支持JMS1.1和J2EE 1.4规范的 JMS Provider实现。我们在本次课程中介绍 ActiveMQ的使用。

　　（2）RabbitMQ

　　AMQP协议的领导实现，支持多种场景。淘宝的MySQL集群内部有使用它进行通讯，OpenStack开源云平台的通信组件，最先在金融行业得到运用。

　　（3）ZeroMQ

　　史上最快的消息队列系统

　　（4）Kafka

　　Apache下的一个子项目 。特点：高吞吐，在一台普通的服务器上既可以达到10W/s的吞吐速率；完全的分布式系统。适合处理海量数据。

**JMS 与AMQP区别**

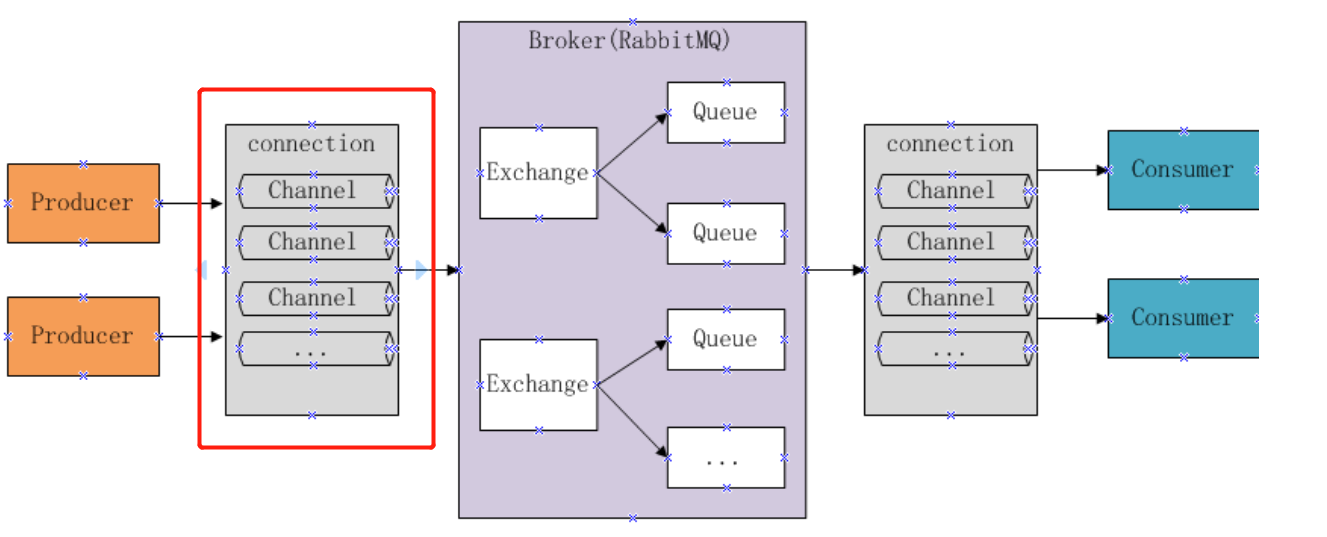
**说到底是协议规范与接口规范的区别**

JMS是java的接口规范，旨在为java应用提供统一的消息操作。类似于JDBC，用户都是根据相应的接口可以和实现了JMS的服务进行通信，进行相关的操作。

JMS是API规范，具有接口开发的特点 ：程序结构清晰，使用方便，有利于程序设计的规范化。实现了程序的可插拔性，对于不同的需求切换不同的实现，降低了耦合度。

AMQP是一种协议，与JMS的本质差别是AMQP不从API层进行限定，而是直接定义网络交换的数据格式。这使得实现了AMQP的provider天然性就是跨平台的，实现不同语言不同平台系统的互操作。意味着我们可以使用Java的AMQP provider，同时使用一个python的producer加一个rubby的consumer。其类似于http协议，不同语言实现的客户端凡是遵守http协议的都可以跟web服务器做通讯。

# RabbitMQ工作原理



**各角色(如exchage、queue等)作用一定要清楚 请看讲义介绍**

什么是信道？

信道是建立在 Connection 之上的虚拟连接，RabbitMQ 处理的每条 AMQP 指令都是通过信道完成的。

我们完全可以使用 Connection 就能完成信道的工作，为什么还要引入信道呢？

试想这样一个场景，一个应用程序中有很多个线程需要从 RabbitMQ 中消费消息，或者生产消息，那么必然需要建立很多个 Connection，也就是多个 TCP 连接。

然而对于操作系统而言，建立和销毁 TCP 连接是非常昂贵的开销，如果遇到使用高峰，性能瓶颈也随之显现。

当每个信道的流量不是很大时，复用单一的 Connection 可以在产生性能瓶颈的情况下有效地节省 TCP 连接资源。但是信道本身的流量很大时，这时候多个信道复用一个 Connection 就会产生性能瓶颈，进而使整体的流量被限制了。此时就需要开辟多个 Connection，将这些信道均摊到这些 Connection 中，至于这些相关的调优策略需要根据业务自身的实际情况进行调节。

# Rabbbitmq几种工作模式

一个队列可以绑定多个消费者，同一个队列里的消费者不能重复消费一条消息，不同队列里的消费者是可以重复消费消息的。

不同工作模式的最大区别在于使用不同的交换机，不同的交换机有不同的消息转发规则。

work模式：只有默认的交换机，不能指定交换机，一个队列可以绑定多个消费者，每个队列的消息只能被其中一个消费者消费。

订阅模式：交换机会把消息转发给所有与之绑定的queue。

路由模式：交换机会把消息转发给所有与之绑定的且routingkey与该消息routingkey相等的queue。

topic模式：该模式与路由模式类似，在routingkey匹配上使用通配符匹配。

通配符：\*可以匹配一个标识符，#可以匹配0个或多个标识符

header模式：了解

# 面试问题

## 为什么引入消息系统？

简答：说出应用场景

## 如何保证RabbitMQ 的高可用性？

简答：可以搭建镜像集群模式， 镜像模式下queue被同步到多个节点，即使某个节点挂了，还可以通过其它节点对外提供服务。答的好，就详细讲解一下 下面的内容。

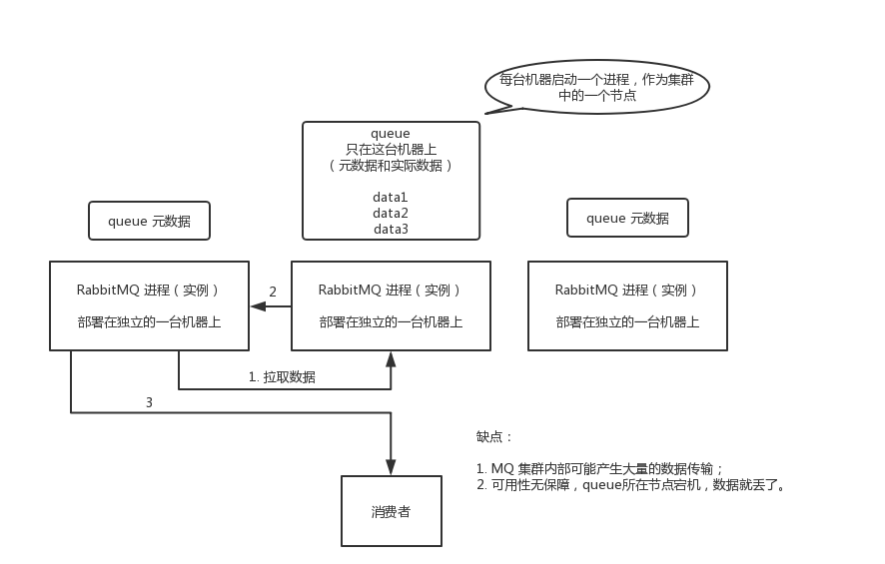
RabbitMQ 有三种模式：单机模式、普通集群模式、镜像集群模式。

**单机模式**

单机模式，只能用于测试，不可用于生产环境

**普通集群模式**（无高可用性）

普通集群模式，意思就是在多台机器上启动多个 RabbitMQ 实例，每个机器启动一个。你创建的 queue，**只会放在一个 RabbitMQ 实例**上，但是每个实例都同步 queue 的元数据（元数据可以认为是 queue 的一些配置信息，通过元数据，可以找到 queue 所在实例）。你消费的时候，实际上如果连接到了另外一个实例，那么那个实例会从 queue 所在实例上拉取数据过来。



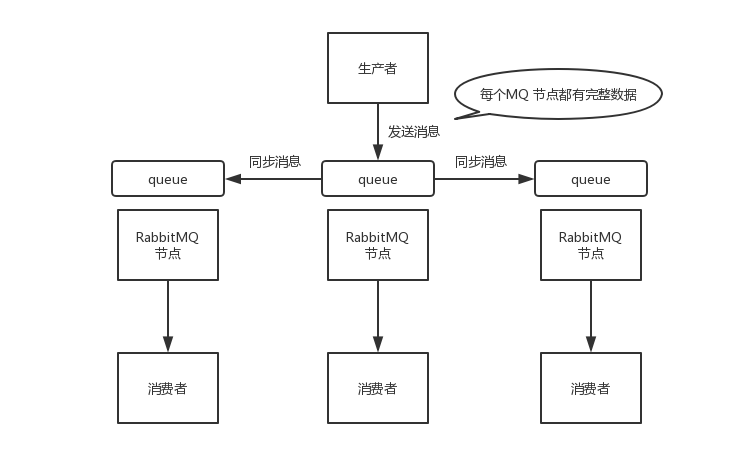
这种方式确实很麻烦，也不怎么好，没做到所谓的分布式，就是个普通集群。因为这导致你要么消费者每次随机连接一个实例然后拉取数据，要么固定连接那个 queue 所在实例消费数据，前者有数据拉取的开销，后者导致单实例性能瓶颈。

而且如果那个放 queue 的实例宕机了，会导致接下来其他实例就无法从那个实例拉取，如果你开启了消息持久化，让 RabbitMQ 落地存储消息的话，消息不一定会丢，得等这个实例恢复了，然后才可以继续从这个 queue 拉取数据。

这个方案没有什么所谓的高可用性，这方案主要是提高吞吐量的，就是说让集群中多个节点来服务某个 queue 的读写操作

**镜像集群模式（高可用性）**<https://blog.csdn.net/vbirdbest/article/details/78740346>

这种模式，才是所谓的 RabbitMQ 的高可用模式。跟普通集群模式不一样的是，在镜像集群模式下，你创建的 queue，无论元数据还是 queue 里的消息都会存在于多个实例上，就是说，每个 RabbitMQ 节点都有这个 queue 的一个完整镜像，包含 queue 的全部数据的意思。然后每次你写消息到 queue 的时候，都会自动把消息同步到多个实例的 queue 上。



那么如何开启这个镜像集群模式呢？其实很简单，RabbitMQ 有很好的管理控制台，就是在后台新增一个策略，这个策略是镜像集群模式的策略，指定的时候是可以要求数据同步到所有节点的，也可以要求同步到指定数量的节点，再次创建 queue 的时候，应用这个策略，就会自动将数据同步到其他的节点上去了。

这样的话，好处在于，你任何一个机器宕机了，其它机器（节点）还包含了这个 queue 的完整数据，别的 consumer 都可以到其它节点上去消费数据。坏处在于，第一，性能开销大，消息需要同步到所有机器上，导致网络带宽压力和消耗很重！第二，，不是分布式的，就没有扩展性可言了，如果某个 queue 负载很重，你加机器，新增的机器也包含了这个 queue 的所有数据，并没有办法线性扩展你的 queue。如果这个 queue 的数据量很大，大到这个机器上的容量无法容纳了，此时该怎么办呢？

## 消息的可靠传输

**丢消息怎么办？在使用过程中有没有遇到消息丢失的情况？**

简答：消息发送可以通过事务模式或confirm模式确认机制来确保生产者不丢消息。

消息消费通过手动确认方式来确保消息一定会被消费。

对queue及消息设置为持久化确保服务重启后消息不会丢失。

搭建集群模式确保消息不会因为某个节点顺坏（如硬盘损坏）而导致消息丢失。

哪些地方可能会丢失？

发送消息、消费消息、存储消息。

**生产者发送消息**

从生产者弄丢数据这个角度来看，RabbitMQ提供transaction和confirm模式来确保生产者不丢消息。

transaction模式：发送消息前，开启事物(channel.txSelect())，然后发送消息，如果发送过程中出现什么异常，事物就会回滚(channel.txRollback())，如果发送成功则提交事物(channel.txCommit())。然而缺点就是吞吐量下降了。

//原生客户端

try {

channel.txSelect();

channel.basicPublish(exchange, routingKey, MessageProperties.PERSISTENT\_TEXT\_PLAIN, msg.getBytes());

int result = 1 / 0;

channel.txCommit();

} catch (Exception e) {

e.printStackTrace();

channel.txRollback();

}

comfirm模式**：**将信道设置成confirm模式（发送方确认模式），则所有在信道上发布的消息都会被指派一个唯一的ID。一旦消息被投递到目的队列后，或者消息被写入磁盘后（可持久化的消息），信道会发送一个ack给生产者（包含消息唯一ID）。如果RabbitMQ发生内部错误从而导致消息丢失，会发送一条nack（not acknowledged，未确认）消息。

发送方确认模式是异步的，生产者应用程序在等待确认的同时，可以继续发送消息。当确认消息到达生产者应用程序，生产者应用程序的回调方法就会被触发来处理确认消息。

在事务机制中，只有当前事务在全部镜像queue中执行之后，客户端才会收到ack的消息。同样的，在publisher confirm机制中，向publisher进行当前message确认的前提是该message被全部镜像所接受了。

**channel的confirm模式详解**

<https://blog.csdn.net/xwnxwn/article/details/80462576>

**消费者角度**

①自动确认模式，消费者挂掉，待ack的消息回归到队列中。消费者抛出异常，消息会不断的被重发，直到处理成功。不会丢失消息，即便服务挂掉，没有处理完成的消息会重回队列，但是异常会让消息不断重试。

②手动确认模式，如果消费者来不及处理就死掉时，没有响应ack时会重复发送一条信息给其他消费者；如果监听程序处理异常了，且未对异常进行捕获，会一直重复接收消息，然后一直抛异常；如果对异常进行了捕获，但是没有在finally里ack，也会一直重复发送消息(重试机制)。

③不确认模式，acknowledge="none" 不使用确认机制，只要消息发送完成会立即在队列移除，无论客户端异常还是断开，只要发送完就移除，不会重发。

启用手动确认模式可以解决这个问题

<https://www.cnblogs.com/gordonkong/p/6952957.html?utm_source=itdadao&utm_medium=referral>

**消息数据的存储**

处理消息队列丢数据的情况，一般是开启持久化磁盘的配置。这个持久化配置可以和confirm机制配合使用，你可以在消息持久化磁盘后，再给生产者发送一个Ack信号。这样，如果消息持久化磁盘之前，rabbitMQ阵亡了，那么生产者收不到Ack信号，生产者会自动重发。

那么如何持久化呢，这里顺便说一下吧，其实也很容易，就下面两步

设置持久化有两个步骤：

* 创建 queue 的时候将其设置为持久化

这样就可以保证 RabbitMQ 持久化 queue 的元数据，但是它是不会持久化 queue 里的数据的。

* 第二个是发送消息的时候将消息的 deliveryMode 设置为 2

就是将消息设置为持久化的，此时 RabbitMQ 就会将消息持久化到磁盘上去。

这样设置以后，rabbitMQ就算挂了，重启后也能恢复数据。在消息还没有持久化到硬盘时，可能服务已经死掉，这种情况可以通过引入mirrored-queue即镜像队列，但也不能保证消息百分百不丢失（整个集群都挂掉）

**硬盘可能损坏？**

**镜像集群**

RabbitMQ主从之间的数据复制是异步的

<https://www.cnblogs.com/wangzhongqiu/p/7831854.html>

## 如何保证消息顺序消费？

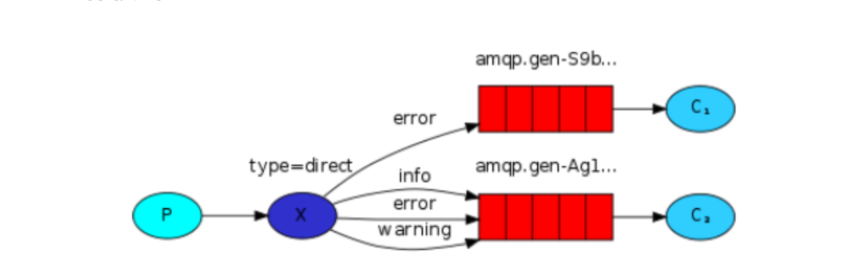
简答：确保需要顺序消费的一组消息进入同一队列，且队列只能绑定一个消费者。具体讲一个某个工作模式如何保证需要顺序消费的一组消息进入同一队列。

**顺序消费**

RabbitMQ：一个 queue，多个 consumer。比如，生产者向 RabbitMQ 里发送了三条数据，顺序依次是 data1/data2/data3，压入的是 RabbitMQ 的一个队列。有三个消费者分别从 MQ 中消费这三条数据中的一条，结果消费者2先执行完操作，把 data2 存入数据库，然后是 data1/data3，这就导致数据没有按时消费。

**顺序消费场景**

在网购的时候，我们需要下单，那么下单需要假如有三个顺序，第一、创建订单 ，第二：订单付款，第三：订单完成。



解决：

可以使用路由模式，保证同一个订单的一组消息具有相同的routingkey，

假如queue设置五个

routingkey=order\_0,order\_1,order\_2,order\_23,order\_24

cal(orderid) %5= [0,1,2,3,4]

## 如何避免消息重复的被消费

简答：

什么时候会重复消费？

处理业务逻辑后，向rabbitmq发送ack消息，由于网络等原因，造成rabbitmq无法接受到ack消息，导致rabbitmq会重复发送该消息给消费者。

解决：

消费端处理消息的业务逻辑保持幂等性。

**方案1：**

在消息处理前进行去重判断，但每个消息需要需有一个消息的唯一ID作为去重的依据，如果消息已处理这不再进行消费，直接ack回复mq。这里我们可以使用redis存储已处理的消息id来进行去重。

步骤

1、查询redis是否已存在该消息id，如果存在则该消息已处理过，直接跳到第5步骤

2、如果redis中不存在该消息的唯一id，则进行业务逻辑处理

3、处理的业务结果保存至数据库（支持事务的数据库）

4、将消息的id存储至redis中。

5、发送ack至rabbitmq。

注意：这个方案业务结果保存一定要放在支持事务的数据库中，否则在业务成功情况下，redis保存失败就会导致重复消费的问题。如果数据库支持事务，在redis操作失败的情况下可以回滚数据库保存的处理结果信息，也就无所谓再处理一次消息了。

**方案2：**

在方案1中数据库需要支持事务，假如我们使用的是mongodb，就会有重复消费的问题。

此时我们可以把消息唯一id和处理结果一起保存到一个数据库表中，这样一起成功一起失败，也就不会有重复消费的问题了。

前提：保证消息有一个唯一id

步骤：

1. 如果数据库中不存在该消息的唯一id，则进行业务处理
2. 如果数据库中不存在该消息id，则进行业务处理

3、处理的业务结果与消息唯一id保存至数据库

4、发送ack至rabbitmq.