<https://mp.weixin.qq.com/s?__biz=MzIyNDU2ODA4OQ==&mid=2247484018&idx=1&sn=703b250bfaa480ea737dc4ed51fd11a3&chksm=e80db404df7a3d12968a981bbe597d5934157b3a97c97065af0a2a0b0d8bc8e7b8d6ab2ec1ed&scene=21#wechat_redirect>

# 为什么需要消息队列？使用消息队列有什么好处？

## 目录

一、消息队列的特性  
二、为什么需要消息队列？  
三、使用消息队列有什么好处？  
四、为什么需要分布式？  
五、分布式环境下需要解决哪些问题？  
六、如何实现？  
七、常见消息队列对比和选型

## 一、消息队列的特性

业务无关，一个具有普适性质的消息队列组件不需要考虑上层的业务模型，只做好消息的分发就可以了，上层业务的不同模块反而需要依赖消息队列所定义的规范进行通信。

FIFO，先投递先到达的保证是一个消息队列和一个buffer的本质区别。

容灾，对于普适的消息队列组件来说，节点的动态增删和消息的持久化，都是支持其容灾能力的重要基本特性。当然，这个特性对于游戏服务器中大部分应用中的消息队列来说不是必须的，这个也是跟应用情景有关的，很多时候没有这种持久化的需求。

性能，这个不必多说了，消息队列的吞吐量上去了，整个系统的内部通信效率也会有提高。

## 二、为什么需要消息队列？

当系统中出现“生产“和“消费“的速度或稳定性等因素不一致的时候，就需要消息队列，作为抽象层，弥合双方的差异。“ 消息 ”是在两台计算机间传送的数据单位。消息可以非常简单，例如只包含文本字符串；也可以更复杂，可能包含嵌入对象。消息被发送到队列中，“ 消息队列 ”是在消息的传输过程中保存消息的容器 。

举几个例子

1）业务系统触发短信发送申请，但短信发送模块速度跟不上，需要将来不及处理的消息暂存一下，缓冲压力。就可以把短信发送申请丢到消息队列，直接返回用户成功，短信发送模块再可以慢慢去消息队列中取消息进行处理。

2）调远程系统下订单成本较高，且因为网络等因素，不稳定，攒一批一起发送。

3）任务处理类的系统，先把用户发起的任务请求接收过来存到消息队列中，然后后端开启多个应用程序从队列中取任务进行处理。

## 三、使用消息队列有什么好处？

### 3.1、提高系统响应速度

使用了消息队列，生产者一方，把消息往队列里一扔，就可以立马返回，响应用户了。无需等待处理结果。

处理结果可以让用户稍后自己来取，如医院取化验单。也可以让生产者订阅（如：留下手机号码或让生产者实现listener接口、加入监听队列），有结果了通知。获得约定将结果放在某处，无需通知。

### 3.2、提高系统稳定性

考虑电商系统下订单，发送数据给生产系统的情况。电商系统和生产系统之间的网络有可能掉线，生产系统可能会因维护等原因暂停服务。如果不使用消息队列，电商系统数据发布出去，顾客无法下单，影响业务开展。两个系统间不应该如此紧密耦合。应该通过消息队列解耦。同时让系统更健壮、稳定。

### 异步化、解耦、消除峰值

以上三点其实可以用一个例子来解释——设想有一款MMO游戏，没有人肉写的缓存层或者ORM，所有逻辑节点都直连MySQL，逻辑节点内除了要关注场景、战斗、交互等复杂逻辑以外，还要有个拼SQL语句的模块，想想简直是蛋疼。先考虑一下这样设计的弊端所在：

* 逻辑节点与Db的交互会有大量IO，即使把与Db交互的模块耦合在逻辑节点内，其实现对你来说是黑盒，如果内部是同步实现的，那就直接卡你游戏主逻辑，就因为一次存盘操作，玩家们都掉线了，服务器也可以关掉了。
* 那么我们改进一下，针对1的情况，可以把这个模块做到一个线程里挂在逻辑节点上。这样其实逻辑节点跟这个Db前端模块的交互就会基于一个比较原始的消息队列。但是这样还有一个坏处，那就是这两种任务一种是计算密集的（玩家的逻辑处理）、一种是IO密集的（只负责写入读取MySQL），搞到一个节点中，扩展起来会非常麻烦，而且耦合度太高。比如说现在发现场景放单节点上有瓶颈，要按场景分节点，那么这种挂在上面的数据模块怎么跟其他场景的交互呢？
* 峰值的问题。在分布式系统中，一次分布式事务关联的是多个节点，其中每一个节点出现问题都会成为整个事务处理流程中的瓶颈。如果逻辑节点与数据库之间没有一个起到缓冲作用的节点，那就是每次操作都要访问数据库，对于MMO来说，一个玩家上线load几百K数据，一个服10万个玩家上线已经足够搞垮一个mysql节点了。如果直接搞垮还是比较好的结果，至少是前面的玩家确实登录上去了并且可以正常游戏，后面的玩家登录不上。但是很可惜，十年前开始流行的C10K说法就是在讲：并发量上来之后，会造成chain reaction，大量的并发不会直接挂掉你的mysql节点，但是会拖慢速度，降低吞吐量，一个玩家的请求由于处理时间太长，导致玩家放弃重试，但是对于后端来说，对该玩家之前的处理过程消耗的资源就全部浪费了，陷入恶性循环。

所以，这种情景下，一个介于逻辑节点和db节点之间的缓存节点就是理所当然的事情了。这个缓存节点其实很多时候也可以看作是一个更复杂的消息队列节点。

## 四、为什么需要分布式？

### 4.1、多系统协作需要分布式

消息队列中的数据需要在多个系统间共享数据才能发挥价值。所以必须提供分布式通信机制、协同机制。

### 4.2、单系统内部署环境需要分布式

单系统内部，为了更好的性能、为了避免单点故障，多为集群环境。集群环境中，应用运行在多台服务器的多个JVM中；数据也保存在各种类型的数据库或非数据库的多个节点上。为了满足多节点协作需要，需要提供分布式的解决方案。

## 五、分布式环境下需要解决哪些问题？

### 5.1、并发问题

需进行良好的并发控制。确保“线程安全“。不要出现一个订单被出货两次。不要出现顾客A下的单，发货发给了顾客B等情况。

### 5.2、简单的、统一的操作机制

需定义简单的，语义明确的，业务无关的，恰当稳妥的统一的访问方式。

### 5.3、容错

控制好单点故障，确保数据安全。

### 5.4、可横向扩展

可便捷扩容。

## 六、如何实现？

成熟的消息队列中间件产品太多了，族繁不及备载。成熟产品经过验证，接口规范，可扩展性强。

结合事业环境因素、组织过程遗产、实施运维考虑、技术路线考虑、开发人员情况等原因综合考虑。

## 七、常见消息队列对比和选型

