[什么情况下的异步操作使用消息队列而不是多线程?](https://segmentfault.com/q/1010000011291800)

1.

消息队列和多线程两者并不冲突，多线程可以作为队列的生产者和消费者。  
使用外部的消息队列时，**第一是可以提高应用的稳定性，当程序fail后，写入外部消息队列的数据依旧是保存的，**如果使用两步commit的队列的话，可以更加提高这个项目。

2.

**用线程的话，会占用主服务器资源， 消息队列的话，可以放到其他机器上运行**， 让主服务器尽量多的服务其他请求。

3.

**解耦更充分，架构更合理**多线程是在**编程语言层面解决问题  
消息队列是在架构层面解决问题**我认为**架构层面解决问题是“觉悟比较高的方式“**，理想情况下应该限制语言层面滥用多线程，能不用就不用

4.

用线程池ExecutorService异步处理：我理解ExecutorService其实也是内部使用了队列（如LinkedBlockingQueue），所以从设计上，其实和使用中间价的消息队列是差不多一致的。只是这里**应用服务器**既充当生产者又充当消费者，也是消息队列中间价的实现者。这种应该适合非分布式的架构，比如简单的只有一台服务器。

使用消息队列：消息队列（指activeMQ，rabbitMQ，kafaKa，Redis等）因为一般都是中间件，部署在其他机器，需要一定的网络消耗。

**本着解耦的目的，使用后者更合理，因为应用服务器一般内存也不会太多，队列长度不易太长。让应用服务器只处理逻辑比较合理。适合分布式架构。**

将redis发布订阅模式用做消息队列和rabbitmq的区别：

* **可靠性**
  + redis ：没有相应的机制保证消息的可靠消费，如果发布者发布一条消息，而没有对应的订阅者的话，**这条消息将丢失，不会存在内存中；**
  + rabbitmq：具有消息消费确认机制，如果发布一条消息，还没有消费者消费该队列，那么这条消息将一直存放在队列中，直到有消费者消费了该条消息，以此**可以保证消息的可靠消费.**
* **实时性**
  + redis:**实时性高，**redis作为高效的缓存服务器，所有**数据都存在在服务器**中，所以它具有更高的实时性
* 消费者负载均衡：
  + rabbitmq队列可以被多个消费者同时监控消费，但是每一条消息只能被消费一次，由于**rabbitmq的消费确认机制**，因此它能够根据消费者的消费能力而调整它的负载；
  + redis发布订阅模式，**一个队列可以被多个消费者同时订阅，当有消息到达时，会将该消息依次发送给每个订阅者；**
* 持久性
  + redis：redis的持久化是针对于整个redis缓存的内容，它**有RDB和AOF两种**持久化方式（redis持久化方式，后续更新），可以将整个redis实例持久化到磁盘，以此来做数据备份，防止异常情况下导致数据丢失。
  + rabbitmq：**队列，消息都可以选择性持久化，**持久化粒度更小，更灵活；
* **队列监控**
  + rabbitmq实现了**后台监控平台**，可以在该平台上看到所有创建的队列的详细情况，良好的后台管理平台可以方面我们更好的使用；
  + redis没有所谓的监控平台。
* **出入队性能**
* 对于RabbitMQ和Redis的入队和出队操作，各执行100万次，每10万次记录一次执行时间。  
  测试数据分为128Bytes、512Bytes、1K和10K四个不同大小的数据。  
  实验表明：  
  入队时，当数据比较小时Redis的性能要高于RabbitMQ，而如果数据大小超过了10K，Redis则慢的无法忍受；  
  出队时，无论数据大小，Redis都表现出非常好的性能，而RabbitMQ的出队性能则远低于Redis。
* 总结
  + redis：       **轻量级，低延迟，高并发，低可靠性；**
  + rabbitmq：**重量级，高可靠，异步，不保证实时；**
  + rabbitmq是一个专门的AMQP协议队列，他的优势就在于**提供可靠的队列服务，并且可做到异步，而redis主要是用于缓存的，redis的发布订阅模块，可用于实现及时性，且可靠性低的功能。**

**所以项目中所采用的Redis可能只是用于一个轻量级的应用。只是用于简单的发短信，邮件等通知，如果业务要进一步的扩展，如果需要消息队列的话，可能还需要用到专用的那些重量级的消息中间件比如rabbitmq等，他们的高可靠，负载均衡这些特性都是Redis所没有的。**