**kafka**

# kafka是什么

消息队列，消息中间件。

Kafka是一种分布式的，基于发布/订阅的消息系统。主要设计目标如下：

O(1)就是最低的时空复杂度了，也就是耗时/耗空间与输入数据大小无关，无论输入数据增大多少倍，耗时/耗空间都不变

* 以时间复杂度为O(1)的方式提供消息持久化能力，并保证即使对TB级以上数据也能保证常数时间的访问性能
* 高吞吐率。即使在非常廉价的商用机器上也能做到单机支持每秒100K条消息的传输
* 支持Kafka Server间的消息分区，及分布式消息消费，同时保证每个partition内的消息顺序传输
* 同时支持离线数据（一周）处理和实时数据处理

# 为什么要用Message Queue

* ****解耦****  提供了一个隐含的，基于数据的接口层，消费者和生产者只需要遵守他们的接口规则就可以  
  在项目启动之初来预测将来项目会碰到什么需求，是极其困难的。消息队列在处理过程中间插入了一个隐含的、基于数据的接口层，两边的处理过程都要实现这一接口。这允许你独立的扩展或修改两边的处理过程，只要确保它们遵守同样的接口约束
* ****冗余**** 队列中的数据会被持久化起来，直到该数据被消息被完全处理完  
  有时在处理数据的时候处理过程会失败。除非数据被持久化，否则将永远丢失。消息队列把数据进行持久化直到它们已经被完全处理，通过这一方式规避了数据丢失 风险。在被许多消息队列所采用的”插入-获取-删除”范式中，在把一个消息从队列中删除之前，需要你的处理过程明确的指出该消息已经被处理完毕，确保你的 数据被安全的保存直到你使用完毕。
* ****扩展性**** 因为解耦，所以可以在不改变代码和配置的情况下，很容易增大消息入队和处理的频率，  
  因为消息队列解耦了你的处理过程，所以增大消息入队和处理的频率是很容易的；只要另外增加处理过程即可。不需要改变代码、不需要调节参数。扩展就像调大电力按钮一样简单。
* ****灵活性 & 峰值处理能力****  消息对列能够使关键组件顶住增长访问的压力  
  在访问量剧增的情况下，应用仍然需要继续发挥作用，但是这样的突发流量并不常见；如果为以能处理这类峰值访问为标准来投入资源随时待命无疑是巨大的浪费。使用消息队列能够使关键组件顶住增长的访问压力，而不是因为超出负荷的请求而完全崩溃。
* ****可恢复性****  加入队列中的消息仍然可以在系统恢复后被处理  
  当体系的一部分组件失效，不会影响到整个系统。消息队列降低了进程间的耦合度，所以即使一个处理消息的进程挂掉，加入队列中的消息仍然可以在系统恢复后被处理。而这种允许重试或者延后处理请求的能力通常是造就一个略感不便的用户和一个沮丧透顶的用户之间的区别。
* ****送达保证**** 每一个消息只能被处理一次，如果客户端没有明确表示处理完成，在一段时间内会重新放到队列中去，在一段可配置的时间之后可再次被处理  
  消息队列提供的冗余机制保证了消息能被实际的处理，只要一个进程读取了该队列即可。在此基础上，IronMQ提供了一个”只送达一次”保证。无论有多少进 程在从队列中领取数据，每一个消息只能被处理一次。这之所以成为可能，是因为获取一个消息只是”预定”了这个消息，暂时把它移出了队列。除非客户端明确的 表示已经处理完了这个消息，否则这个消息会被放回队列中去，在一段可配置的时间之后可再次被处理。
* ****顺序保证****  消息队列本来就是排序的，并且能保证数据会按照特定的顺序来处理  
  在许多情况下，数据处理的顺序都很重要。消息队列本来就是排序的，并且能保证数据会按照特定的顺序来处理。IronMO保证消息浆糊通过FIFO（先进先出）的顺序来处理，因此消息在队列中的位置就是从队列中检索他们的位置。
* ****缓冲**** 消息队列通过一个缓冲层来帮助任务最高效率的执行，写入处理不受从队列读的预备处理的约束  
  在任何重要的系统中，都会有需要不同的处理时间的元素。例如,加载一张图片比应用过滤器花费更少的时间。消息队列通过一个缓冲层来帮助任务最高效率的执行—写入队列的处理会尽可能的快速，而不受从队列读的预备处理的约束。该缓冲有助于控制和优化数据流经过系统的速度。
* ****理解数据流****   
  在一个分布式系统里，要得到一个关于用户操作会用多长时间及其原因的总体印象，是个巨大的挑战。消息系列通过消息被处理的频率，来方便的辅助确定那些表现不佳的处理过程或领域，这些地方的数据流都不够优化。
* ****异步通信****   
  很多时候，你不想也不需要立即处理消息。消息队列提供了异步处理机制，允许你把一个消息放入队列，但并不立即处理它。你想向队列中放入多少消息就放多少，然后在你乐意的时候再去处理它们。

# 常用Message Queue对比

* ****RabbitMQ****   
  RabbitMQ是使用Erlang编写的一个开源的消息队列，本身支持很多的协议：AMQP，XMPP, SMTP, STOMP，也正因如此，它非常重量级，更适合于企业级的开发。同时实现了Broker构架，这意味着消息在发送给客户端时先在中心队列排队。对路由，负 载均衡或者数据持久化都有很好的支持。
* ****Redis****   
  Redis是一个基于Key-Value对的NoSQL数据库，开发维护很活跃。虽然它是一个Key-Value数据库存储系统，但它本身支持MQ功能， 所以完全可以当做一个轻量级的队列服务来使用。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。
* ****ZeroMQ****   
  ZeroMQ号称最快的消息队列系统，尤其针对大吞吐量的需求场景。ZMQ能够实现RabbitMQ不擅长的高级/复杂的队列，但是开发人员需要自己组合 多种技术框架，技术上的复杂度是对这MQ能够应用成功的挑战。ZeroMQ具有一个独特的非中间件的模式，你不需要安装和运行一个消息服务器或中间件，因 为你的应用程序将扮演了这个服务角色。你只需要简单的引用ZeroMQ程序库，可以使用NuGet安装，然后你就可以愉快的在应用程序之间发送消息了。但 是ZeroMQ仅提供非持久性的队列，也就是说如果down机，数据将会丢失。其中，Twitter的Storm中默认使用ZeroMQ作为数据流的传 输。
* ****ActiveMQ**** ]98+  
  ActiveMQ是Apache下的一个子项目。 类似于ZeroMQ，它能够以代理人和点对点的技术实现队列。同时类似于RabbitMQ，它少量代码就可以高效地实现高级应用场景。
* ****Kafka/Jafka****   
  Kafka是Apache下的一个子项目，是一个高性能跨语言分布式Publish/Subscribe消息队列系统，而Jafka是在Kafka之上孵 化而来的，即Kafka的一个升级版。具有以下特性：快速持久化，可以在O(1)的系统开销下进行消息持久化；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，Broker、Producer、Consumer都原生自动支持分布式，自动实现复杂均衡；支持Hadoop 数据并行加载，对于像Hadoop的一样的日志数据和离线分析系统，但又要求实时处理的限制，这是一个可行的解决方案。Kafka通过Hadoop的并行 加载机制来统一了在线和离线的消息处理，这一点也是本课题所研究系统所看重的。Apache Kafka相对于ActiveMQ是一个非常轻量级的消息系统，除了性能非常好之外，还是一个工作良好的分布式系统。

# Kafka解析

## **Terminology**

* ****Broker****   
  Kafka集群包含一个或多个服务器，这种服务器被称为broker
* ****Topic****   
  每条发布到Kafka集群的消息都有一个类别，这个类别被称为topic。（物理上不同topic的消息分开存储，逻辑上一个topic的消息虽然保存于一个或多个broker上但用户只需指定消息的topic即可生产或消费数据而不必关心数据存于何处）
* ****Partition****   
  parition是物理上的概念，每个topic包含一个或多个partition，创建topic时可指定parition数量。每个partition对应于一个文件夹，该文件夹下存储该partition的数据和索引文件
* ****Producer**** 生产者  
  负责发布消息到Kafka broker
* ****Consumer**** ****消费者****  
  消费消息。每个consumer属于一个特定的consuer group（可为每个consumer指定group name，若不指定group name则属于默认的group）。使用consumer high level API时，同一topic的一条消息只能被同一个consumer group内的一个consumer消费，但多个consumer group可同时消费这一消息。