Broadview
www.broadview.com.cn

分布式消息中间件实践

详解消息中间件的高可用、高性能配置和原理

倪炜 / 著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

分布式消息中间件实践

倪炜 / 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

消息中间件是分布式系统中的重要组件，在实际工作中常用消息中间件进行系统间数据交换，从而解决应用解耦、异步消息、流量削峰等问题，实现高性能、高可用、可伸缩和最终一致性架构。目前市面上可供选择的消息中间件有 RabbitMQ、ActiveMQ、Kafka、RocketMQ、ZeroMQ、MetaMQ 等。本书结合作者近年来在实际项目中使用消息中间件的经历和踩过的一些坑总结整理而成，主要介绍消息中间件使用的背景、基本概念，以及常用的四种消息中间件产品在各个业务场景中的使用案例。作为案例介绍，虽然不能对各种消息中间件产品的所有特性做详细说明，但是希望读者可以通过每章中的案例讨论和实践建议得到启发，为在实际工作中进行产品选型、业务场景方案制定、性能调整等提供思路。

本书适合初、中级软件工程师阅读，不管是有一定工作经验的软件工程师、运维工程师，还是对消息中间件技术感兴趣的在校学生均可参考。由于书中案例主要采用 Java 编写，为了能更好地阅读本书，读者要有 Java 语言的使用能力和基本的 Linux 操作系统使用经历。

图书在版编目（CIP）数据

分布式消息中间件实践 / 倪炜著. —北京：电子工业出版社，2018.9
ISBN 978-7-121-34905-8

Ⅰ. ①分… Ⅱ. ①倪… Ⅲ. ①JAVA 语言－程序设计 Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字（2018）第 187932 号

策划编辑：陈晓猛
责任编辑：葛 娜
印 刷：三河市双峰印刷装订有限公司
装 订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编：100036
开 本：787×980 1/16 印张：17.75 字数：407 千字
版 次：2018 年 9 月第 1 版
印 次：2018 年 9 月第 1 次印刷
定 价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）51260888-819，faq@phei.com.cn。

前　言

大约在五年前，那时我参与的项目到了收尾阶段，工作不太忙，觉得写了好几年业务代码没什么意思，就想找点有技术含量的东西研究。有一次去山西路上的军人俱乐部闲逛，在三楼一家书店的角落里看到一本讲 Tomcat 源码的书，翻了二三十页觉得挺有意思，讲的很多关于 Tomcat 实现的由来以前我从来都不知道，买回家不到一个星期就看了一遍。遗憾的是，这本书讲的 Tomcat 版本有点老，在实际工作中一般都用到 5 以上的版本了。正好当时我也有点时间，就决定分析一下最新的 Tomcat 7 的源码，并发表在了 ITeye 网站上。作为老牌的 Web 服务器，Tomcat 7 的内容非常丰富，写这个系列文章的主要部分，前后花了一年的时间，由此我也就逐渐养成了坚持写博客总结一段时间以来工作和学习到的知识的习惯。

这几年我也常劝很多朋友多写点东西总结，有时作为面试官遇到技术还不错的面试者会问问有没有博客，如果有的话一般会在面试成绩上额外加点分。据我的观察，几乎每一个程序员都知道写博客的好处，但真的动手去写的人实际很少，一个很重要的原因，就是很多人会说“我又不是大牛，写出来的东西没人看，那还有啥意义？”我的回答是，不是牛人一样可以写博客。有一次我碰到一个项目用到一些以前没接触过的新技术，在项目搭建过程中报出错误，但错误信息提示不明确，只说某地方有一个配置校验不通过的异常，到底要通过什么方式解决该问题文档上也没写，只能“Google”一下，在搜索结果的第一页就看到有人遇到了同样的问题、一样的环境和最终的解决办法，一试之下果然奏效。于是就翻了翻他的博客目录，写的大部分内容都是很基础的，有的可能就是对某个技术里的某个特性的介绍，没有高深莫测的东西，也没有长篇大论，但正好里面有篇文章解决了困扰我大半天的问题，显然这篇博客对我来说就非常有价值。在实际工作和学习中我们会遇到很多问题，有的问题经过千方百计翻遍资料甚至查看源码实现，经过一番痛苦折磨终于解决了，那么就可以把这个过程整理成博客，既对自己所学的知识进行了总结、积累了经验，又能给其他可能会碰到类似问题的人提供帮助，分享出去，让更多的人受益，这就是我认为的写博客的价值。

也就是在这个过程中，有一天我接到了博文视点陈晓猛编辑的邀请，说看了我写的关于消息中间件介绍的几篇文章，想让我出一本与之相关的书，系统介绍一下相关技术。这真是一个让人兴奋的消息，读了这么多年书，现在居然有机会出自己的书了。在欣喜之外还有忐忑，担心我的经验与水平有限，写出来的东西耽误了读者的时间，但是经过陈编辑的多次鼓励，我决定大胆尝试一下。在写这本书之前，其实国内已经有一些介绍消息中间件的书了，但这些书大

都是针对某个具体产品的详细介绍，比如与 RabbitMQ 和 ActiveMQ 相关的中文、英文读物。市面上缺少的是针对主流消息中间件的整体性介绍，以及结合实际消息应用场景的案例说明，因此我打算写一本这样的书。本书选取了我认为市面上应用最广泛的四种消息中间件产品，即 RabbitMQ、ActiveMQ、Kafka 和 RocketMQ，介绍这些消息中间件的来源、特性、Java 语言使用的示例和结合具体业务场景的应用案例，最后给出在实际项目中使用时的一些建议和需要综合权衡的技术要点，希望能对读者的工作有一定的帮助。所以，本书面对的读者主要是具备一定的 Java 功底，尤其是对消息中间件感兴趣并有一定实际使用经验的工程师。本书并不会对每种产品的特性都做非常详细的阐述，因为这些最权威、最详尽的资料都可以从官网中获得，书中内容聚焦于实践，也建议读者能多动手实践一下，这样学到的东西才是自己的。

本书内容

全书共 6 章，第 1 章和第 2 章介绍消息中间件的背景和所涉及的常见概念，第 3 ~ 6 章分别介绍一种消息中间件产品，读者可自行选择阅读。

第 1 章：介绍消息队列技术的背景，包括使用场景和消息队列的功能特点，并设计了一个简单的消息队列。

第 2 章：介绍消息队列中常用的消息协议，包括每个消息协议的历史背景、主要概念和基于该协议的消息通信过程。本章所介绍的协议也是接下来理解各种消息中间件产品的基础。

第 3 章：具体介绍 RabbitMQ 的特点、主要概念和 Java 使用示例，接着通过使用 RabbitMQ 实现异步处理和消息推送的功能，最后给出在工作中使用 RabbitMQ 时的一些实践建议。

第 4 章：具体介绍 ActiveMQ 的特点、基本概念和 Java 使用示例，接着通过使用 ActiveMQ 实现消息推送分布式事务的功能，最后给出在工作中使用 ActiveMQ 时的一些实践建议。

第 5 章：具体介绍 Kafka 的特点、主要概念和 Java 使用示例，接着通过使用 Kafka 实现用户行为数据采集、日志收集和流量削峰的功能，最后给出在工作中使用 Kafka 时的一些实践建议。

第 6 章：具体介绍 RocketMQ 的特点、主要概念和 Java 使用示例，接着通过使用 RocketMQ 的特性实现消息顺序处理和分布式事务的另外一种解决方案，最后给出在工作中使用 RocketMQ 时的一些实践建议。

致谢

虽然这是我写的第一本书，但我不是唯一的作者。书中的很多内容简介分散于各种书籍、标准文档、研究论文、会议演讲、博客，甚至微博、Twitter 中关于某个消息中间件特性的讨论

之上，感谢互联网，是前面许许多多的实践者成就了这本书。工作这么多年，我有幸与很多在软件开发领域有不懈追求的同仁共事，他们扩展了我在很多方面的知识，有很多人帮助我审阅过部分手稿。感谢他们的帮忙，从他们身上我学到了许多宝贵的经验，更感谢他们为本书提供的许多宝贵建议。感谢博文视点的编辑，这本书能够如期出版，离不开你们的敬业精神与一丝不苟的工作态度，我为你们点赞！

倪炜

2018 年 6 月于南京

目　录

第 1 章 消息队列

1.1 系统间通信技术介绍

一般来说，大型应用通常会被拆分成多个子系统，这些子系统可能会部署在多台机器上，也可能只是一台机器的多个进程中，这样的应用就是分布式应用。在讨论分布式应用时，很多初学者会把它和集群这个概念搞混，因为从部署形态上看，它们都是多台机器或多个进程部署，而且都是为了实现一个业务功能。这里有一个简单的区分标准：如果是一个业务被拆分成多个子业务部署在不同的服务器上，那就是分布式应用；如果是同一个业务部署在多台服务器上，那就是集群。而分布式应用的子系统之间并不是完全独立的，它们需要相互通信来共同完成某个功能，这就涉及系统间通信了。

目前，业界通常有两种方式来实现系统间通信，其中一种是基于远程过程调用的方式；另一种是基于消息队列的方式。前一种就是我们常说的 RPC 调用，客户端不需要知道调用的具体实现细节，只需直接调用实际存在于远程计算机上的某个对象即可，但调用方式看起来和调用本地应用程序中的对象一样（见图 1-1）。

RPC 是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。这句话至少有三个层面的含义。① 它是协议，说明这是一种规范，就需要有遵循这套规范的实现。典型的 RPC 实现包括 Dubbo、Thrift、GRPC 等。② 网络通信的实现是透明的，调用方不需要关心网络之间的通信协议、网络 I/O 模型、通信的信息格式等。③ 跨语言，调用方实际上并不清楚对端服务器使用的是什么程序语言。对于调用方来说，无论其使用的是何种程序语言，调用都应该成功，并且返回值也应按照调用方程序语言能理解的形式来描述。

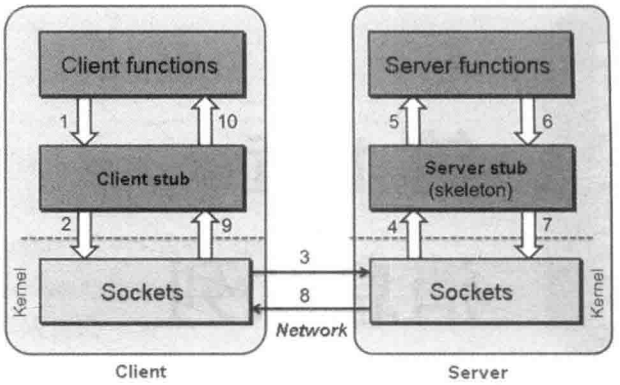

图 1-1

而基于消息队列的方式是指由应用中的某个系统负责发送信息，由关心这条消息的相应系统负责接收消息，并在收到消息后进行各自系统内的业务处理。消息可以非常简单，比如只包含文本字符串；也可以很复杂，比如包含字节流、字节数组，还可能包含嵌入对象，甚至是 Java 对象（经过序列化的对象）。

消息在被发送后可以立即返回，由消息队列来负责消息的传递，消息发布者只管将消息发布到消息队列而不用管谁来取，消息使用者只管从消息队列中取消息而不管是谁发布的，这样发布者和使用者都不用知道对方的存在（见图 1-2）。

图 1-2

1.2 为何要用消息队列

从上一节的描述中可以看出，消息队列（MQ）是一种系统间相互协作的通信机制。那么什么时候需要使用消息队列呢？

举个例子。某天产品人员说“系统要增加一个用户注册功能，注册成功后用户能收到邮件通知”。在实际场景中这种需求很常见，开发人员觉得这个很简单，就是提供一个注册页面，点击按钮提交之后保存用户的注册信息，然后发送邮件，最后返回用户注册成功（见图 1-3）。

图 1-3

该功能上线运行了一段时间后，产品人员说“点击注册按钮之后响应有点慢，很多人都提出这个意见，能不能优化一下”。开发人员首先想到的优化方案是将保存注册信息与发送邮件分开执行，怎么分呢？可以单独启动线程来做发送邮件的事情（见图 1-4）。

图 1-4

没多久，产品人员又说“现在注册操作的响应是快了，但有用户反映没收到注册成功的邮件，能不能在发送邮件的时候先保存所发送邮件的内容，如果邮件发送失败了则进行补发”。

看着开发人员愁眉苦脸的样子，产品人员说“在邮件发送这块平台部门已经做好方案了，你直接用他们提供的服务就行”。开发人员一听，赶紧和平台部门沟通，对方的答复是“我们提供一个类似于邮局信箱的东西，你直接往这个信箱里写上发送邮件的地址、邮件标题和内容，之后就不用你操心了，我们会直接从信箱里取信息，向你所填写的邮件地址发送响应邮件”。

这个故事讲的就是使用消息队列的典型场景——异步处理。消息队列还可用于解决解耦、流量削峰、日志收集、事务最终一致性等问题。

1. 解耦

某天产品人员说“为了便于网站用户之间进行交流、共享信息，解决网站中遇到的各种问题，网站要增加一个论坛功能，在论坛中用户可以发布帖子，其他用户可以在这个帖子下评论和回复”。开发人员觉得这个需求不难实现，就是常见的网上论坛。于是，没过几天就完成了功能开发并转测上线了。

过了一段时间，用户量增加了，对帖子发布功能的使用越来越频繁。产品人员说“信息质量部门期望在发布帖子的时候能检查所发布的内容是不是合规”。没多久，产品人员又说“大数据部门希望能根据帖子的内容提取用户相关信息来丰富用户的画像”。又过了几天，产品人员说“营销部门最近在做活动，如果用户发布的是与营销活动相关的帖子，则能给该用户增加积分”。

经验少的开发人员遇到这种情况，一般是来一个需求叠加一段业务逻辑代码。这当然可以尽快交付上线，但仔细分析需求发现，发布帖子应该作为一个独立的功能，并且这个功能并不需要关心后面不断增加的那些功能，更不需要关心后面功能的执行结果，只需要**通知**其他相应模块执行就可以了。

在大型系统的开发过程中会经常碰到此类情况，随着需求的叠加，各模块之间逐渐变成了相互调用的关系，这种模块间紧密关联的关系就是紧耦合。紧耦合带来的问题是对一个模块的功能变更将导致其关联模块发生变化，因此各个模块难以独立演化。

要解决这个问题，可以在模块之间调用时增加一个中间层来实现解耦，这也方便了以后的扩展。所谓解耦，简单地讲，就是一个模块只关心自己的核心流程，而依赖该模块执行结果的其他模块如果做的不是很重要的事情，有通知即可，无须等待结果。换句话说，基于消息队列的模型，关心的是**通知**，而非**处理**。

2. 流量削峰

在互联网行业中，可能会出现在某一时刻网站突然迎来用户请求高峰期的情况（典型的如淘宝的“双 11”、京东的“618”、12306 的春运抢票等）。在网站初期设计中，可能就直接将用户的请求数据写入数据库，但如果一直延续这样的设计，当遇到高并发的场景时将会给数据库带来巨大压力，并发访问量大到超过了原先系统的承载能力，会使系统的响应延迟加剧。如果在设计上考虑不周甚至会发生**雪崩**（在分布式系统中，经常会出现某个基础服务不可用造成整个系统不可用的情况，这种现象被称为“服务雪崩效应”），从而发生整个系统不可用的严重生产事故。

当访问量剧增时系统依然可以继续使用，该怎么做呢？首先想到的是购买更多的服务器进行扩展，以增强系统处理并发请求的能力。这个想法看起来土，但是很多大公司在高速发展过程中遇到此类问题时也都这么处理过（比如淘宝、京东）。对于很多公司来说，突发流量状况其实并不常见，如果都以能处理此类流量峰值为标准投入大量资源随时待命无疑是很大的浪费。在业界的诸多实践中，常见的是使用**消息队列**，先将短时间高并发的请求持久化，然后逐步处理，从而削平高峰期的并发流量，改善系统的性能。

3. 日志收集

在项目开发和运维中日志是一个非常重要的部分，通过日志可以跟踪调试信息、定位问题等。在初期很多系统可能各自独立记录日志，定位问题也需要到每个系统对应的目录中查看相应的日志。但是随着业务的发展，要建设的系统越来越多，系统的功能也越做越多，每天产生的日志数据量变得越来越大。通过分析海量的日志来实时获取每个系统当前的状态，变得越来越迫切，离线分析当前系统的功能为未来的设计和扩展提供参考，也变得越来越重要。

在这种情况下，利用消息队列产品在接收和持久化消息方面的高性能，引入消息队列快速接收日志消息，避免因为写入日志时的某些故障导致业务系统访问阻塞、请求延迟等。所以很多公司会选择构建一个日志收集系统，由它来统一收集业务日志数据，供离线和在线的分析系统使用。

4. 事务最终一致性

在余额宝刚推出时，由于其收益高，很多人都用过，现在我们从系统设计的角度来思考从支付宝账户向余额宝账户转账的问题。在这个场景中，最简单的情况是至少存在两个账户表。

- 支付宝账户表：A（id，userId，amount）
- 余额宝账户表：B（id，userId，amount）
- userId = 123456

假设要从支付宝账户转 1000 元到余额宝账户，则至少要分两步操作。

① 从支付宝账户表扣除 1000：update A set amount=amount-1000 where userId=123456。

② 余额宝账户表增加 1000：update B set amount=amount+1000 where userId=123456。

那么如何保持支付宝和余额宝两个账户之间的收支平衡呢？显然这里的问题本质是能够让这两步操作在同一个事务中。如果这两个账户表在同一个数据库中，那么处理起来就很简单了。

```
begin transaction

update A set amount=amount-1000 where userId=123456;
update B set amount=amount+1000 where userId=123456;

end transaction
commit;
```

如果系统很小，则将两个表存储在同一个数据库中，上面的问题迎刃而解。而且很多框架都提供了事务管理功能，可能都不需要单独写与事务管控相关的代码（比如开启事务、结束事务、提交事务之类的）。假如公司发展得很好很快，系统的规模不断增大，大到支付宝系统和余额宝系统需要分开建设，导致支付宝账户单独有一套数据库，余额宝账户有另一套数据库，此时原先通过数据库提供的事务处理方式则无法解决问题。这就是很多人听说过的所谓分布式事务的问题。

那么该如何做呢？业界曾经提出过一个处理分布式事务的规范——XA。XA 主要定义了**全局**事务管理器（Transaction Manager）和**局部**资源管理器（Resource Manager）之间的接口。XA 接口是双向的系统接口，在事务管理器及一个或多个资源管理器之间形成通信桥梁。XA 引入的事务管理器充当全局事务中的**协调者**的角色。事务管理器控制着全局事务，管理事务生命周期，并协调资源。资源管理器负责控制和管理实际资源（如数据库或 JMS 队列）。目前各主流数据库都提供了对 XA 规范的支持。

对于这种方式实现难度不算太大，适合传统项目中偶尔需要在同一个方法中跨库操作的情况。因为这种方案的每一个事务操作都涉及系统间的多次通信、协调，所以它的最大缺陷是性

能很差，因此并不适合在生产环境下有高并发和高性能要求的场景。

在业界的很多实践方案中，都可以借助消息队列来处理此问题。简单地讲，就是在支付宝账户扣钱的同时发送一条让余额宝账户加钱的消息到消息队列，余额宝系统一旦接收到该消息就操作数据库在自己的账户中加钱。读者可能会问：如何保证数据库操作和消息队列操作在同一个事务中呢？这里可以通过增加一个事件表来解决。本书后面会详细介绍使用事件表和消息队列实现分布式事务的方案。

1.3 消息队列的功能特点

回到消息队列这个术语本身，它包含了两个关键词：消息和队列。消息是指在应用间传送的数据，消息的表现形式是多样的，可以简单到只包含文本字符串，也可以复杂到有一个结构化的对象定义格式。对于队列，从抽象意义上来理解，就是指消息的进和出。从时间顺序上说，进和出并不一定是同步进行的，所以需要一个容器来暂存和处理消息。因此，一个典型意义上的消息队列，至少需要包含消息的发送、接收和暂存功能（见图 1-5）。

图 1-5

- Broker：消息处理中心，负责消息的接收、存储、转发等。
- Producer：消息生产者，负责产生和发送消息到消息处理中心。
- Consumer：消息消费者，负责从消息处理中心获取消息，并进行相应的处理。

但在生产环境应用中，对消息队列的要求远不止基本的消息发送、接收和暂存。在不同的业务场景中，需要消息队列产品能解决诸如消息堆积、消息持久化、可靠投递、消息重复、严格有序、集群等各种问题。

1. 消息堆积

根据消息队列的生产者、消费者处理模型来分析，因为生产者和消费者是两个分开处理消息的系统，所以无法预知两者对消息处理速度的快慢，一旦在某个时间段消费者处理消息的速度没有跟上生产者发送消息的速度，必将导致消息在处理中心逐渐积压而得不到释放。因此，有时需要消息队列产品能够处理这种情况，比如给消息队列设置一个阈值，将超过阈值的消息不再放入处理中心，以防止系统资源被耗尽，导致机器挂掉甚至整个消息队列不可用。

2. 消息持久化

在设计一个消息队列时，如果消息到达消息处理中心后不做任何处理就直接转给消费者，那么消息处理中心也就失去了存在的意义，无法满足流量削峰等需求。所以常规的做法是先将消息暂存下来，然后选择合适的时机将消息投递给消费者。消息暂存可以选择将消息放在内存中，也可以选择放到文件、数据库等地方。将消息放在内存中存在的最大问题是，一旦机器宕掉消息将丢失。如果场景需要消息不能丢失，那么势必要将消息持久化。持久化方案有很多种，比如将消息存到本地文件、分布式文件系统、数据库系统中等。

3. 可靠投递

可靠投递是不允许存在消息丢失的情况的。从消息的整个生命周期来分析，消息丢失的情况一般发生在如下过程中：

- 从生产者到消息处理中心。
- 从消息处理中心到消息消费者。
- 消息处理中心持久化消息。

由于跨越不同的系统，中间会碰到诸如网络问题、系统宕机等各种不确定的情形，但对于消息发送者来说都是一件事，就是消息没有送达。在有些场景下需要保证消息不能丢失，比如网购时订单的支付成功消息不能丢失，否则该笔订单将会卡在未支付环节，用户肯定会抱怨。

4. 消息重复

有些消息队列为了支持消息可靠投递，会选择在接收到消息后先持久化到本地，然后发送给消费者。当消息发送失败或者不知道是否发送成功时（比如超时），消息的状态是待发送，定时任务不停地轮询所有的待发送消息，最终保证消息不会丢失，这就带来了消息可能会重复的问题。其实并不是所有场景都需要消息可靠投递，比如在论坛系统或招聘系统中，话题被重复发布或简历被重复投递，可能比丢失一个话题或一份简历更让用户不舒服。

5. 严格有序

在实际的业务场景中，经常会碰到需要按生产消息时的顺序来消费的情形。比如网购时产生的订单，每一笔订单一般都经过创建订单、支付完成、已发货、已收货、订单完成等环节，每个环节都可能产生消息，但会要求严格按照顺序消费消息，否则在业务处理上就是不正确的。比如为了提高用户体验，订单流转到每个环节时都会给用户发送一个提醒，提醒用消息队列来处理，但在业务上可能不允许还没收到订单支付完成的提醒就先处理订单完成的提醒。这就需要消息队列能够提供有序消息的保证。但顺序消费却不一定需要消息在整个产品中全局有序，有的产品可能只需要提供局部有序的保证。就拿上面的例子来说，只要该笔订单的消息能够都投递到该产品的局部，其实就算满足了业务需求。

6. 集群

在大型应用中，系统架构一般都需要实现高可用性，以排除单点故障引起的服务中断，保证 7×24 小时不间断运行，所以可能需要消息队列产品提供对集群模式的支持。集群不仅可以让消费者和生产者在某个节点崩溃的情况下继续运行，集群之间的多个节点还能够共享负载，当某台机器或网络出现故障时能自动进行负载均衡，而且可以通过增加更多的节点来提高消息通信的吞吐量。

7. 消息中间件

非底层操作系统软件、非业务应用软件，不是直接给最终用户使用的，不能直接给客户带来价值的软件统称为中间件。消息中间件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集成分布式系统。中间件是一种独立的系统软件或服务程序，分布式应用系统借助这种软件在不同的技术之间共享资源，管理计算资源和网络通信。中间件在计算机系统中是一个关键软件，它能实现应用的互联和互操作，能保证系统安全、可靠、高效运行。中间件位于用户应用和操作系统及网络软件之间，它为应用提供了公用的通信手段，并且独立于网络和操作系统。中间件为开发者提供了公用于所有环境的应用程序接口，当在应用程序中嵌入其函数调用时，它便可利用其运行的特定操作系统和网络环境的功能，为应用执行通信功能。

目前中间件的种类有很多，比如交易管理中间件（如 IBM 的 CICS）、面向 Java 应用的 Web 应用服务器中间件（如 IBM 的 WebSphere Application Server）等。而消息传输中间件（MOM）是其中的一种，它简化了应用之间数据的传输，屏蔽了底层的异构操作系统和网络平台，提供了一致的通信和应用开发标准，确保在分布式计算网络环境下可靠、跨平台的信息传输和数据交换。它基于消息队列的存储-转发机制，并提供了特有的异步传输机制，能够基于消息传输和异步事务处理实现应用整合与数据交换。

IBM 消息中间件 MQ 以其独特的安全机制，简便、快速的编程风格，卓越不凡的稳定性、可扩展性和跨平台性，以及强大的事务处理能力和消息通信能力，成为市场占有率最高的消息中间件产品。

1.4 设计一个简单的消息队列

看了那么多文字论述，不如自己动手实践一遍体会深刻。下面我们用 Java 语言写一个简单的消息队列。从 1.3 节中的描述可知，在消息队列的完整使用场景中至少包含三个角色。

- 消息处理中心：负责消息的接收、存储、转发等。
- 消息生产者：负责产生和发送消息到消息处理中心。
- 消息消费者：负责从消息处理中心获取消息，并进行相应的处理。