<http://book.51cto.com/art/201502/466288.htm>

现在你已经在实践中见过了.MQ，让我们回到“为什么”。

当今的许多应用程序都包含了跨越某种网络的组件，无论这种网络是局域网还是互联网。因此，许多应用程序开发者最终都会处理某种类型的消息传递。一些开发人员使用消息队列产品，但大多数时候，他们使用TCP 或UDP 自己做。这些协议并不难用，但是，

从A 发送几个字节到B 和以任何一种可靠的方式处理消息，这两者之间有很大的区别。

让我们来看看当开始使用原始的TCP 连接部件的时候，我们要面对的典型问题。任何可复用的消息层都需要解决如下所有这些问题或其中的大部分问题：

我们如何处理I/O 呢？是让我们的应用程序阻塞，还是在后台处理I/O 呢？这是一个关键的设计决策。阻塞式I/O 创建的架构不能很好地扩展，但后台I/O 也是非常难以正确做到的。

我们如何处理动态组件（例如，暂时撤除的块）呢？我们需要正式将组件划分为“客户端”和“服务器”，并强制该服务器不能撤除吗？那么，如果我们想将服务器连接到服务器时该怎么办呢？我们需要每隔几秒钟就尝试重新连接吗？

我们如何表示在线路上的消息呢？我们应该怎样将数据组织为帧，才能使得它很容易写入和读取，避免缓冲区溢出，既对小型消息高效，也足以处理非常大的戴着聚会礼帽的跳舞猫的视频呢？

我们如何处理不能立即传递的消息呢？特别是当我们在等待一个组件的联机回应时如何处理呢？我们需要丢弃消息，把它们放入一个数据库，或者把它们放到一个内存队列吗？

我们在哪里存储消息队列呢？如果组件从队列中读取很慢，导致我们的队列堆积，这会发生什么情况？我们的策略是什么呢？

我们如何处理丢失的消息呢？我们应该等待新的数据，要求重发，还是应该建立某种可靠性层，确保信息不会丢失呢？如果该层本身崩溃了该怎么办呢？

如果我们需要使用一个不同的网络传输，比如说，用多播来取代TCP 单播，或IPv6，该怎么办呢？我们需要重写应用程序吗？还是将传输抽象到某个层中呢？

我们如何路由消息呢？我们可以发送同样的消息到多个接收者吗？我们可以发送应答给原来的请求者吗？

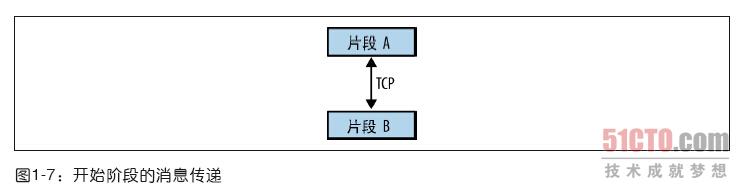
我们如何编写出另一种语言的API 呢？我们应该重新实现一个线路级协议，还是重新包装一个库？如果是前者，我们怎么能保证协议栈的高效稳定呢？如果是后者，我们又怎么能保证互操作性呢？

我们应该如何表示数据，以便它可以在不同的架构之间读取呢？我们应该对数据类型强制执行特定的编码吗？究竟到什么程度，才是消息传递系统的工作，而不是更高一层的工作呢？

我们应该如何处理网络错误呢？是等待并重试，默默地忽略它们，还是终止它们呢？

以Apache ZooKeeper（[http://zookeeper.apache.org](http://zookeeper.apache.org/)）这个典型的开源项目为例，请阅读在src/c/src/zookeeper.c 中的C API 代码（<http://github.com/apache/zookeeper/blob/trunk/src/c/src/zookeeper.c>）。当我在2010 年阅读这段代码时，它是3200 行的神秘代码，并且包含一个未在文档中记录的客户端/ 服务器网络通信协议。我发现，它的高效是因为它使用的是poll() 而不是select()。不过说真的，ZooKeeper 应使用一个通用的消息层和一个在文档中明确记载的线路层协议。对开发团队来说，重复制造特殊的轮子，这是令人难以置信的浪费。

但是，我们如何才能做一个可复用的消息传递层呢？为什么有那么多的项目都需要这项技术，人们都还在通过在他们的代码中驱动TCP 套接字费力地做它，并重复解决一长串清单中的问题呢（参见图1-7）？

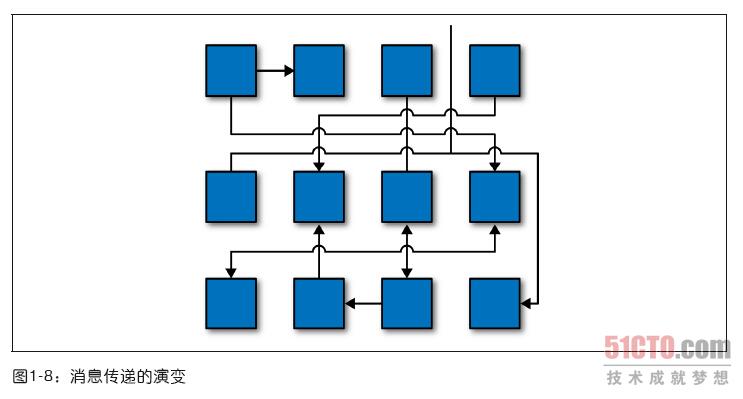
[](http://s2.51cto.com/wyfs02/M00/59/DE/wKiom1TteqbSU7HIAAAw8hKvqTU324.jpg)

事实证明，构建可复用的消息传递系统是非常困难的，这就是为什么很少有自由和开放源码（FOSS）项目尝试做这项工作的原因，它也是商业通信产品复杂、昂贵、灵活性差，而且脆弱的原因。2006 年，iMatix 设计了高级消息队列协议，或AMQP（[http://www.amqp.org](http://www.amqp.org/)），开始给FOSS 开发者提供也许是首个可复用的消息传递系统方法。AMQP 工作得比许多其他设计更好，但仍然相对复杂、昂贵，而且脆弱（<http://www.imatix.com/articles:whats-wrong-with-amqp>）。学会如何使用它需要几个星期，而要用它来建立在事情变得很麻烦时也不会崩溃的稳定架构则需要几个月。

大多数消息传递项目（如AMQP）都在尝试以可重用的方式解决上面这个冗长的清单上的问题，它们通过发明一种负责寻址、路由和排队的新概念——代理，来做到这一点。这将导致一个客户端/ 服务器协议或一些未在文档中记录的协议之上的一组API，它允许应用程序与这个代理交流。在降低大型网络的复杂性方面，代理是一个很好的东西。但把以代理为基础的消息传递添加到像ZooKeeper 这样的产品会使情况变得更糟，而不是更好。这将意味着增加一台额外的大电脑和一个新的单点故障。代理迅速成为一个瓶颈和一个要管理的新风险。如果软件支持的话，我们可以添加第二个、第三个和第四个代理，并提出一些故障切换方案。人们这么做了。然而，它产生了更多的变动部件，变得更复杂，有更多的东西会被破坏。

此外，以代理为中心的设置都需要自己的运营团队。你真的需要日夜注意代理，并在它们开始“行为不端”时，用棍子敲打它们。你需要电脑，你需要备份的电脑，你需要人来管理那些电脑。只有那些由多个团队在数年内建成的，带有许多变动部件的大型应用程序，才值得这样做。

因此，中小型应用程序开发人员陷入了困境。他们要么避免网络编程，制作不可扩展的单一应用程序，要么跳进网络编程，制作脆弱、复杂、很难维护的应用程序。他们还可以把赌注压在消息传递产品上，并最终获得依赖于昂贵且易破坏的技术的可扩展的应用程序。目前还没有非常好的选择，这也许可以解释为什么消息传递主要还停留在上个世纪，并激起强烈的情绪——对用户是消极的，而对那些销售技术支持和许可的厂商则是欢乐的、喜悦的（参见图1-8）。

[](http://s4.51cto.com/wyfs02/M01/59/DB/wKioL1Tte7aTxxD0AABnLRB-la0909.jpg)

我们需要的是做消息传递工作的东西，但需要它以下面这种简单和廉价的方式完成工作，它可以在任何应用程序中以接近零的消耗开展工作。它应该是不需要任何其他依赖就可以链接的库。无须额外的变动部件，所以没有额外的风险。它应该能运行在任何操作系统上，并能用任何编程语言开展工作。

而这就是.MQ ：一个高效的可嵌入库，它解决了大部分应用程序需要解决的问题，变得在网络上有良好的可伸缩性，而没有多少成本。

具体做法是：

它在后台线程异步处理I/O。这些线程使用无锁数据结构与应用程序线程进行通信，所以并发.MQ 应用程序不再需要锁、信号量，或其他等待状态。

组件可以动态地来去自如，而.MQ 会自动重新连接。这意味着你可以以任何顺序启动组件。你可以创建“面向服务的架构”（SOA），其中的服务可以在任何时间加入和离开网络。

它根据需要自动对消息排队。为此，它会智能地在对消息排队之前，将消息尽可能地推进到接收者。

它有一个处理过满队列（称为“高水位标志”）的方法。当队列满时，.MQ 会自动阻止发件人，或丢弃消息，这取决于你正在做的是哪种消息传递（即所谓的“模式”）。

它可以让你的应用程序通过任意传输协议来互相交流，这些协议可以是：TCP、多播、进程内、进程间。你不需要更改代码以使用不同的传输工具。

它使用依赖于消息传递模式的不同策略，安全地处理速度慢/ 阻塞的读取者。

它可以让你采用多种模式，如请求- 应答和发布- 订阅来将消息路由。这些模式是指你如何创建拓扑结构和网络结构。

它可以让你创建代理（proxy）来排队、转发，或通过一个调用来捕获消息。代理可以降低网络互联的复杂性。

它使用在线路上的简单组帧原封不动地传递整个消息。如果你写了一个10KB 的消息，那么你将收到一个10KB 的消息。

它不对消息强加任何格式。它们是零字节到千兆字节的二进制大对象。当你想表示你的数据时，可以选择其上的其他一些产品，如谷歌的协议缓冲区、XDR 等。

它能智能地处理网络错误。有时候它会重试，有时它会告诉你某个操作失败。

它可以减少你的能源消耗。少花CPU 多办事意味着使用电脑更少的能源，你可以让你的旧电脑使用更长的时间。戈尔（译者注：美国前副总统，环保主义者）也会爱上.MQ 的。

实际上，.MQ 做的比这更多。它对你如何开发网络功能的应用程序有颠覆性的影响。从表面上看，这是一个在其上做zmq\_msg\_recv() 和zmq\_msg\_send() 的套接字风格的API。但该消息处理循环迅速成为中心循环，而你的应用程序很快就会分解成一组消息处理任务。它是优雅和自然的。而且，它可扩展：每个任务对应一个节点，节点通过任意传输方式互相交谈。在一个进程中的两个节点（节点是一个线程），在一台电脑中的两个节点（节点是一个进程），或一个网络上的两台电脑（节点是一台电脑），所有的处理方式都是相同的，不需要更改应用程序代码。