zeromq用来怎么玩？

最近看了一本书《ZeroMQ：云时代极速消息通信库》，作者说那是更好用的socket，但感觉用起来相比其他稳定的网络库没有更简单，因为本身有一定的消息格式，所以主要用于集群内部，但是集群内部都可以用自己喜欢的库，因此觉得要使用zeromq的话需要投资一定学习时间并且看不到特别优势。。。请问应该怎么正确地玩zeromq？有没有非用一种消息队列不可的项目？

[**myd7349**](https://www.zhihu.com/people/myd7349)

前花样作死世界男单总冠军。

162 人赞同了该回答

2015-03-19 最近也在学ZeroMQ，先列一些前辈们的学习心得吧： [RabbitMQ, ZeroMQ, Kafka 是一个层级的东西吗， 相互之间有哪些优缺点？ - 消息队列](http://www.zhihu.com/question/22480085) [新世紀通訊函式庫](https://link.zhihu.com/?target=http%3A//blog.ez2learn.com/2011/12/31/transport-lib-of-new-era-zeromq/) （这篇是繁体的，我用firefox的一个插件“同文堂”转换一下看。）

然后这里说一下自己的一些体会： 1. 首先弄明白0MQ中的“socket”和传统意义上socket的一些差异。 我认为ZeroMQ不能算作单纯的socket库，它应该算是一个messaging library，一个“轻量级”的消息队列库（这里所说的“轻量级”是与过去一些企业级的消息队列对比而言的）。正如我前面列出的两个大神所说的：socket只是表现形式，而通信模式才是其本质。 也就是说，在学0MQ的时候，不要把0MQ中的socket和系统本身的socket弄混。我们可以看看zmq\_socket的文档： [zmq\_socket(3)](https://link.zhihu.com/?target=http%3A//api.zeromq.org/3-2%3Azmq-socket) （zguide告诉我们，这份文档要时不时的拿出来要多读几遍） 注意，每一种不同类型的“socket”，都有自己的消息模式。这其中包括该类型套接字可以和哪些套接字连接到一起工作啊（Compatible peer sockets）、消息传递的方向啊（Direction）、消息收发模式啊（Send/receive pattern）、还有Outgoing routing和Incoming routing用到的一些调度算法啊（比如：Round-robin、Fair-queued，这些调度算法在操作系统课程中应该会有涉及）。当我们创建一个0MQ套接字后，我们实际上得到的是一个加了各种特效的“socket”。 上面一段话，我意在说明，0MQ中的“socket”没那么简单。借用@Dirk Chang答案中“模式”一词，0MQ实际上是把一些在实践中总结出来的消息通信模型封装成不同类型的套接字，以供我们使用。

我认为，这个帖子中很好地阐明了0MQ中“socket”的含义： [zeromq - Does PyZMQ handle creating threads for each new client connection?](https://link.zhihu.com/?target=http%3A//stackoverflow.com/questions/13124359/does-pyzmq-handle-creating-threads-for-each-new-client-connection)

2. 我刚学0MQ的时候总是纠结于“客户端”、“服务端”的概念。比如REQ-REP模式中，究竟哪一端该作为客户端，那一端作为服务端呢？ 实际上，这也算是个小误区吧！因为我们不是在用传统TCP中的Server-client。 在传统TCP中的Server-client中，server要先启动，然后bind到一个端口，等待client调用connect连接它。 而0MQ中调用zmq\_bind和调用zmq\_connect的双方没有那么严格的先后顺序。这也是0MQ有趣的特性之一。直接引用zguide的中的文字吧：

To create a connection between two nodes, you use zmq\_bind() in one node and zmq\_connect() in the other. As a general rule of thumb, the node that does zmq\_bind() is a "server", sitting on a well-known network address, and the node which does zmq\_connect() is a "client", with unknown or arbitrary network addresses. Thus we say that we "bind a socket to an endpoint" and "connect a socket to an endpoint", the endpoint being that well-known network address.

Many architectures follow some kind of client/server model, where the server is the component that is most static, and the clients are the components that are most dynamic, i.e., they come and go the most. There are sometimes issues of addressing: servers will be visible to clients, but not necessarily vice versa. So mostly it's obvious which node should be doing zmq\_bind() (the server) and which should be doing zmq\_connect() (the client). It also depends on the kind of sockets you're using, with some exceptions for unusual network architectures.

简而言之，这段话实际上在讲，在0MQ中，我们会用它提供的套接字构建一个messaging topology，我们通常认为server端应该是这个topology中比较稳定的组件，由这些比较稳定的组件来调用zmq\_bind，而client则是比较动态的部分，它们来来去去，所以我们通常会调用zmq\_connect将它们连接到这个topology中比较稳定的部分。比如，在0MQ中，我们会接触到messaging broker的概念，通常messaging broker会调用zmq\_bind，作为服务端存在。

3. 0MQ的一些特点 (1) 速度快 ZeroMQ的第一个特点就是速度快，这篇帖子中大致给了我们一个这样的概念： [4 款消息队列软件产品大比拼](https://link.zhihu.com/?target=http%3A//www.oschina.net/news/17973/message-queue-shootout) (2) 灵活 首先，再推荐一位大婶的博客： [远程调用服务(RPC)和消息(Message Queue)对比及其适用/不适用场合\_Pragmatistic Guy](https://link.zhihu.com/?target=http%3A//oldratlee.com/post/2013-02-01/synchronous-rpc-vs-asynchronous-message)

0MQ提供的都是一些基本组件，它允许我们自己搭建自己的messaging topology。所以说，0MQ是很灵活的。 本答案的第二个链接中提到，传统的消息队列中有一个“中央集权式”的messaging broker，该messaging broker通常会负责消息在各个节点之间的传输。而0MQ呢，用zguide中的话讲，就是：decentralized。你看，0MQ并不要求你的messaging topology中央必须是一个message broker（这个message broker可能作为消息的存储、转发中心）。在一些简单的通信模型中，省去message broker确实为我们省去了很多工作。而且我们也无需为message broker专门搭建一个服务器。 我们也许会问，如果缺少了message broker，那么未及发送/接受的消息会不会丢失呢？不同担心。因为通常情况下，0MQ中一些套接字本身自带一个buffer，会把这些消息先存下来。

但是ZeroMQ的去中心化不代表完完全全的去中心化。我认为，ZMQ把建立message broker的自由交给了我们。这样，我们可以在有需要的时候建立一个proxy，来简化网络的复杂性和维护城北。zguide中讲到的The Dynamic Discovery Problem、Shared Queue其实都是在教我们在不同场景下应该怎样建立一个broker来降低网络的复杂性而提升其灵活性。 而且，对于一个复杂的消息拓扑，“各自为政”（见：[新世紀通訊函式庫](https://link.zhihu.com/?target=http%3A//blog.ez2learn.com/2011/12/31/transport-lib-of-new-era-zeromq/" \t "_blank)）会可能需要在加入新的节点时重新配置消息拓扑（这会在什么情况下发生，具体可以参考zguide中在介绍The Dynamic Discovery Problem、Shared Queue时引入的例子）。

zguide中描述The Dynamic Discovery Problem这个问题时，拿PUB-SUB模式来举例，说明了使用中间件可以降低两两互联网络的维护成本。中间件的引入使网络更加灵活，因而增加新的节点更加简单。如果不采用中间件，则每次增加新的节点时（比如增加一个新的PUB节点），要重新配置该新节点和现有其他节点之间的关系（比如，把刚才新增的PUB节点和所有现有的SUB节点相连）。

再引用一段zguide中的文字： You might wonder, if all networks eventually get large enough to need intermediaries, why don't we simply have a message broker in place for all applications? For beginners, it's a fair compromise. Just always use a star topology, forget about performance, and things will usually work. However, message brokers are greedy things; in their role as central intermediaries, they become too complex, too stateful, and eventually a problem.

我们通过往网络中加入中间件的方法来降低网络的复杂性，这是一个比较普遍的需求。然而ZMQ并没有因该需求的普遍性而在库中内置形如message borker这样的中间件。该段话阐述了ZMQ这样做的原因：中间件本身是违背ZMQ“去中心化”的设计思想的，而且中间件会变得很复杂。所以，ZMQ把构建中间件的自由给了我们。而且，用ZMQ实现形如message broker这样的消息中间件并不复杂。

通过上面这些讨论，可以看出ZeroMQ极其灵活性（而我们要很好地掌握这种灵活性，需要花些功夫：了解最基本的消息模式、了解彼此之间如何组合起来构建更复杂的消息传递网络拓扑。） (3) 方便 上面提到，0MQ中socket被加了各种特效，所以，我们要实现一些功能的时候，比如：并发、load-balancing，我们需要做的是使用正确的套接字及构建正确的messaging topology。

4. 我认为使用0MQ能带来的好处（实际上，这段主要得益于公司的一位前辈的指导） (1) 跨语言变得简单。 假设，我们有一个模块是用C++实现的，需要提供给另外一个C#/Java/Python/……应用来调用，过去我们可能会使用“C#嵌入C++”/“Java内嵌C++”形如这样的黑科技来实现。但是呢，现在，我们可以用0MQ来实现这种通信。因为0MQ为各种主流语言都提供了bindings。 (2) 模块间的解耦 这里提一下阿里做的RocketMQ。而且，看一下这个issue： [从开发团队获取最新文档，查看哪些用户在使用RocketMQ · Issue #1 · alibaba/RocketMQ · GitHub](https://link.zhihu.com/?target=https%3A//github.com/alibaba/RocketMQ/issues/1) 我们可以看到，很多人拿RocketMQ来做模块间的解耦。

PS：阿里还提供了和阿里云结合的RocketMQ。 (3) 插件化 假设我们在实现一个数据采集、处理系统。数据的处理可能会有多步（比如：A、B、C步），通过0MQ我们可以把每一步处理工作都写成一个模块，类似于插件。这样，传给一个平台的数据可能只需要经过A、C两步，传给另一个平台的可能需要A、B、C三步。各个步骤之间通过0MQ传输数据。这样，当我们想增加新的处理步骤的时候，只需要再写个模块，并加入处理流程就行了。 (4) 并行、负载均衡 0MQ的并行、负载均衡都已经存在于其基因中了。 zguide中有这么一段文字： Multithreading with ZeroMQ To make utterly perfect MT programs (and I mean that literally), we don't need mutexes, locks, or any other form of inter-thread communication except messages sent across ZeroMQ sockets.

If you've spent years learning tricks to make your MT code work at all, let alone rapidly, with locks and semaphores and critical sections, you will be disgusted when you realize it was all for nothing. If there's one lesson we've learned from 30+ years of concurrent programming, it is: just don't share state. It's like two drunkards trying to share a beer. It doesn't matter if they're good buddies. Sooner or later, they're going to get into a fight. And the more drunkards you add to the table, the more they fight each other over the beer. The tragic majority of MT applications look like drunken bar fights. 我认为，值得深思。

你看，学了0MQ，我们的程序设计思路（思考模式）在发生着变化。

5. 一些八卦 github上libzmq的第一贡献者： [Contributors to zeromq/libzmq · GitHub](https://link.zhihu.com/?target=https%3A//github.com/zeromq/libzmq/graphs/contributors) 是这位大婶： [sustrik · GitHub](https://link.zhihu.com/?target=https%3A//github.com/sustrik) 而zguide的作者，以及libzmq目前主要维护者，是这位大婶（iMatix CEO）： [hintjens (Pieter Hintjens) · GitHub](https://link.zhihu.com/?target=https%3A//github.com/hintjens)

据不负责任传言，他们本是一家，后来意见发生分歧，前者就出走，另立项目：Crossroads I/O： [ZeroMQ vs Crossroads I/O](https://link.zhihu.com/?target=http%3A//stackoverflow.com/questions/13494033/zeromq-vs-crossroads-i-o) 后来，Crossroads I/O挂掉之后，前者重写一个项目：[nanomsg/nanomsg · GitHub](https://link.zhihu.com/?target=https%3A//github.com/nanomsg/nanomsg" \t "_blank)。

这个PPT透露出了个中八卦，以及zeromq和nanomsg之间的一些差别（其中提到0MQ中“0”的奥义）： [Nanomsg: ZeroMQ done right](https://link.zhihu.com/?target=https%3A//hguemar.fedorapeople.org/slides/nanomsg/presentation.html%23slide1) 里边插入了奇怪的照片。

还有一个有趣的问题是，sustrik曾说当年选用C++实现libzmq是一个“美丽的”错误（“美丽的”是我加的） [为什么我希望用C而不是C++来实现ZeroMQ](https://link.zhihu.com/?target=http%3A//blog.jobbole.com/19647/) [Why should I have written ZeroMQ in C, not C++ (part I)](https://link.zhihu.com/?target=http%3A//250bpm.com/blog%3A4) 所以你看，nanomsg就是用C实现的了。

另，nanocat貌似不错。

不过不知道nanomsg示例是否丰富、文档是否全。相信学了ZeroMQ，nanomsg也很容易上手。

6. 一些资源 (1) zguide 官网上的[http://zguide.zeromq.org/page:all](https://link.zhihu.com/?target=http%3A//zguide.zeromq.org/page%3Aall" \t "_blank)只是文档 [imatix/zguide · GitHub](https://link.zhihu.com/?target=https%3A//github.com/imatix/zguide) 里面有各种语言的示例。 (2) zguide-cn 这是国内的大婶翻译的，不过好久没更新了。如果一些术语实在不知道啥意思，不妨翻阅一下，和英文版的zguide对比阅读。 [anjuke/zguide-cn · GitHub](https://link.zhihu.com/?target=https%3A//github.com/anjuke/zguide-cn) (3) [ZeroMQ (豆瓣)](https://link.zhihu.com/?target=http%3A//book.douban.com/subject/11622703/) 这本书的第一部分就是zguide中的内容，第二部分更偏向实践应用。 (4) [网络编程 - kaka11的专栏](https://link.zhihu.com/?target=http%3A//blog.csdn.net/kaka11/article/category/619873) ZeroMQ源码分析。orz。

2016-08-29T17:17+08:00 nanomsg 1.0 版已发布。

2016-10-14T13:28+08:00

Pieter Hintjens 最终选择了安乐死。R.I.P.

2017-04-09T13:22:29+08:00

《代码的未来》最后一节讲了 ZeroMQ。

建议也了解一下原作者新重新实现的nanomsg  
[http://nanomsg.org/documentation.html](https://link.zhihu.com/?target=http%3A//nanomsg.org/documentation.html)