19 | 单服务器高性能模式：Reactor与Proactor

2018-06-09 李运华

19 | 单服务器高性能模式：Reactor与Proactor
李运华
- 00:00 / 13:15

专栏上一期我介绍了单服务器高性能的PPC和TPC模式，它们的优点是实现简单，缺点是都无法支撑高并发的场景，尤其是互联网发展到现在，各种海量用户业务的出现，PPC和TPC完全无能为力。今天我将介绍可以应对高并发场景的单服务器高性能架构模式：Reactor和Proactor。

Reactor

PPC模式最主要的问题就是每个连接都要创建进程（为了描述简洁，这里只以PPC和进程为例，实际上换成TPC和线程，原理是一样的），连接结束后进程就销毁了，这样做其实是很大的浪费。为了解决这个问题，一个自然而然的想法就是资源复用，即不再单独为每个连接创建进程，而是创建一个进程池，将连接分配给进程，一个进程可以处理多个连接的业务。

引入资源池的处理方式后，会引出一个新的问题：进程如何才能高效地处理多个连接的业务？当一个连接一个进程时，进程可以采用“read -> 业务处理 -> write”的处理流程，如果当前连接没有数据可以读，则进程就阻塞在read操作上。这种阻塞的方式在一个连接一个进程的场景下没有问题，但如果一个进程处理多个连接，进程阻塞在某个连接的read操作上，此时即使其他连接有数据可读，进程也无法去处理，很显然这样是无法做到高性能的。

解决这个问题的最简单的方式是将read操作改为非阻塞，然后进程不断地轮询多个连接。这种方式能够解决阻塞的问题，但解决的方式并不优雅。首先，轮询是要消耗CPU的；其次，如果一个进程处理几千上万的连接，则轮询的效率是很低的。

为了能够更好地解决上述问题，很容易可以想到，只有当连接上有数据的时候进程才去处理，这就是I/O多路复用技术的来源。

I/O多路复用技术归纳起来有两个关键实现点：

- 当多条连接共用一个阻塞对象后，进程只需要在一个阻塞对象上等待，而无须再轮询所有连接，常见的实现方式有select、epoll、kqueue等。
- 当某条连接有新的数据可以处理时，操作系统会通知进程，进程从阻塞状态返回，开始进行业务处理。

I/O多路复用结合线程池，完美地解决了PPC和TPC的问题，而且“大神们”给它取了一个很牛的名字：Reactor，中文是“反应堆”。联想到“核反应堆”，听起来就很吓人，实际上这里的“反应”不是聚变、裂变反应的意思，而是“**事件反应**”的意思，可以通俗地理解为“**来了一个事件我就有相应的反应**”，这里的“我”就是Reactor，具体的反应就是我们写的代码，Reactor会根据事件类型来调用相应的代码进行处理。Reactor模式也叫Dispatcher模式（在很多开源的系统里面会看到这个名称的类，其实就是实现Reactor模式的），更加贴近模式本身的含义，即I/O多路复用统一监听事件，收到事件后分配（Dispatch）给某个进程。

Reactor模式的核心组成部分包括Reactor和处理资源池（进程池或线程池），其中Reactor负责监听和分配事件，处理资源池负责处理事件。初看Reactor的实现是比较简单的，但实际上结合不同的业务场景，Reactor模式的具体实现方案灵活多变，主要体现在：

- Reactor的数量可以变化：可以是一个Reactor，也可以是多个Reactor。
- 资源池的数量可以变化：以进程为例，可以是单个进程，也可以是多个进程（线程类似）。

将上面两个因素排列组合一下，理论上可以有4种选择，但由于“多Reactor单进程”实现方案相比“单Reactor单进程”方案，既复杂又没有性能优势，因此“多Reactor单进程”方案仅仅是一个理论上的方案，实际没有应用。

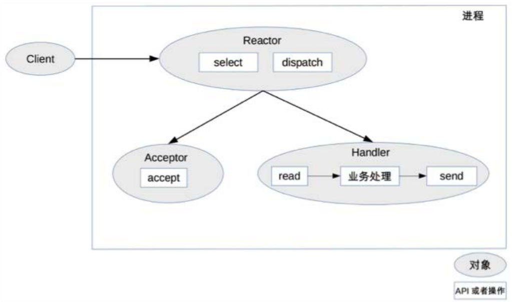
最终Reactor模式有这三种典型的实现方案：

- 单Reactor单进程/线程。
- 单Reactor多线程。
- 多Reactor多进程/线程。

以上方案具体选择进程还是线程，更多地是和编程语言及平台相关。例如，Java语言一般使用线程（例如，Netty），C语言使用进程和线程都可以，例如，Nginx使用进程，Memcache使用线程。

1.单Reactor单进程/线程

单Reactor单进程/线程的方案示意图如下（以进程为例）：



注意，select、accept、read、send是标准的网络编程API，dispatch和“业务处理”是需要完成的操作，其他方案示意图类似。

详细说明一下这个方案：

- Reactor对象通过select监控连接事件，收到事件后通过dispatch进行分发。
- 如果是连接建立的事件，则由Acceptor处理，Acceptor通过accept接受连接，并创建一个Handler来处理连接后续的各种事件。
- 如果不是连接建立事件，则Reactor会调用连接对应的Handler（第2步中创建的Handler）来进行响应。
- Handler会完成read->业务处理->send的完整业务流程。

单Reactor单进程的模式优点就是很简单，没有进程间通信，没有进程竞争，全部都在同一个进程内完成。但其缺点也是非常明显，具体表现有：

- 只有一个进程，无法发挥多核CPU的性能；只能采取部署多个系统来利用多核CPU，但这样会带来运维复杂度，本来只要维护一个系统，用这种方式需要在一台机器上维护多套系统。
- Handler在处理某个连接上的业务时，整个进程无法处理其他连接的事件，很容易导致性能瓶颈。

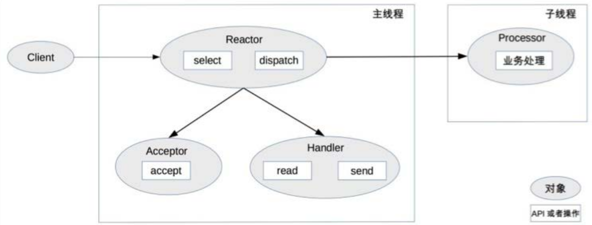
因此，单Reactor单进程的方案在实践中应用场景不多，**只适用于业务处理非常快速的场景**，目前比较著名的开源软件中使用单Reactor单进程的是Redis。

需要注意的是，C语言编写系统的一般使用单Reactor单进程，因为没有必要在进程中再创建线程；而Java语言编写的一般使用单Reactor单线程，因为Java虚拟机是一个进程，虚拟机中有很多线程，业务线程只是其中的一个线程而已。

2.单Reactor多线程

为了克服单Reactor单进程/线程方案的缺点，引入多进程/多线程是显而易见的，这就产生了第2个方案：单Reactor多线程。

单Reactor多线程方案示意图是：



我来介绍一下这个方案：

- 主线程中，Reactor对象通过select监控连接事件，收到事件后通过dispatch进行分发。
- 如果是连接建立的事件，则由Acceptor处理，Acceptor通过accept接受连接，并创建一个Handler来处理连接后续的各种事件。
- 如果不是连接建立事件，则Reactor会调用连接对应的Handler（第2步中创建的Handler）来进行响应。
- Handler只负责响应事件，不进行业务处理；Handler通过read读取到数据后，会发给Processor进行业务处理。
- Processor会在独立的子线程中完成真正的业务处理，然后将响应结果发给主进程的Handler处理；Handler收到响应后通过send将响应结果返回给client。

单Reator多线程方案能够充分利用多核多CPU的处理能力，但同时也存在下面的问题：

- 多线程数据共享和访问比较复杂。例如，子线程完成业务处理后，要把结果传递给主线程的Reactor进行发送，这里涉及共享数据的互斥和保护机制。以Java的NIO为例，Selector是线程安全的，但是通过Selector.selectKeys()返回的键的集合是非线程安全的，对selected keys的处理必须单线程处理或者采取同步措施进行保护。
- Reactor承担所有事件的监听和响应，只在主线程中运行，瞬间高并发时会成为性能瓶颈。

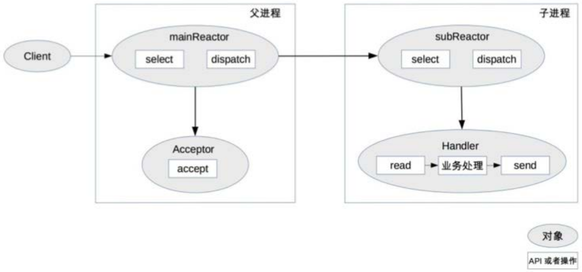
你可能会发现，我只列出了“单Reactor多线程”方案，没有列出“单Reactor多进程”方案，这是什么原因呢？主要原因在于如果采用多进程，子进程完成业务处理后，将结果返回给父进程，并通知父进程发送给哪个client，这是很麻烦的事情。因为父进程只是通过Reactor监听各个连接上的事件然后进行分配，子进程与父进程通信时并不是一个连接。如果要将父

进程和子进程之间的通信模拟为一个连接，并加入Reactor进行监听，则是比较复杂的。而采用多线程时，因为多线程是共享数据的，因此线程间通信是非常方便的。虽然要额外考虑线程间共享数据时的同步问题，但这个复杂度比进程间通信的复杂度要低很多。

3.多Reactor多进程/线程

为了解决单Reactor多线程的问题，最直观的方法就是将单Reactor改为多Reactor，这就产生了第3个方案：多Reactor多进程/线程。

多Reactor多进程/线程方案示意图是（以进程为例）：



方案详细说明如下：

- 父进程中mainReactor对象通过select监控连接建立事件，收到事件后通过Acceptor接收，将新的连接分配给某个子进程。
- 子进程的subReactor将mainReactor分配的连接加入连接队列进行监听，并创建一个Handler用于处理连接的各种事件。
- 当有新的事件发生时，subReactor会调用连接对应的Handler（即第2步中创建的Handler）来进行响应。
- Handler完成read→业务处理→send的完整业务流程。

多Reactor多进程/线程的方案看起来比单Reactor多线程要复杂，但实际实现时反而更加简单，主要原因是：

- 父进程和子进程的职责非常明确，父进程只负责接收新连接，子进程负责完成后续的业务处理。
- 父进程和子进程的交互很简单，父进程只需要把新连接传给子进程，子进程无须返回数据。
- 子进程之间是互相独立的，无须同步共享之类的处理（这里仅限于网络模型相关的select、read、send等无须同步共享，"业务处理"还是有可能需要同步共享的）。

目前著名的开源系统Nginx采用的是多Reactor多进程，采用多Reactor多线程的实现有Memcache和Netty。

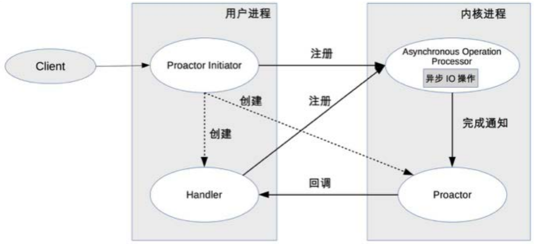
我多说一句，Nginx采用的是多Reactor多进程的模式，但方案与标准的多Reactor多进程有差异。具体差异表现为主进程中仅仅创建了监听端口，并没有创建mainReactor来"accept"连接，而是由子进程的Reactor来"accept"连接，通过锁来控制一次只有一个子进程进行"accept"，子进程"accept"新连接后就放到自己的Reactor进行处理，不会再分配给其他子进程，更多细节请查阅相关资料或阅读Nginx源码。

Proactor

Reactor是非阻塞同步网络模型，因为真正的read和send操作都需要用户进程同步操作。这里的"同步"指用户进程在执行read和send这类I/O操作的时候是同步的，如果把I/O操作改为异步就能够进一步提升性能，这就是异步网络模型Proactor。

Proactor中文翻译为"前摄器"比较难理解，与其类似的单词是proactive，含义为"主动的"，因此我们照猫画虎翻译为"主动器"反而更好理解。Reactor可以理解为"来了事件我通知你，你来处理"，而Proactor可以理解为**"来了事件我来处理，处理完了我通知你"**。这里的"我"就是操作系统内核，"事件"就是有新连接、有数据可读、有数据可写的这些I/O事件，"你"就是我们的程序代码。

Proactor模型示意图是：



详细介绍一下Proactor方案：

- Proactor Initiator负责创建Proactor和Handler，并将Proactor和Handler都通过Asynchronous Operation Processor注册到内核。
- Asynchronous Operation Processor负责处理注册请求，并完成I/O操作。
- Asynchronous Operation Processor完成I/O操作后通知Proactor。
- Proactor根据不同的事件类型回调不同的Handler进行业务处理。
- Handler完成业务处理，Handler也可以注册新的Handler到内核进程。

理论上Proactor比Reactor效率要高一些，异步I/O能够充分利用DMA特性，让I/O操作与计算重叠，但要实现真正的异步I/O，操作系统需要做大量的工作。目前Windows下通过IOCP实现了真正的异步I/O，而在Linux系统下的AIO并不完善，因此在Linux下实现高并发网络编程时都是以Reactor模式为主。所以即使Boost.Asio号称实现了Proactor模型，其实它在Windows下采用IOCP，而在Linux下是用Reactor模式（采用epoll）模拟出来的异步模型。

小结

今天我为你讲了单服务器支持高并发的高性能架构模式Reactor和Proactor，希望对你有所帮助。

这就是今天的全部内容，留一道思考题给你吧，针对“前浪微博”消息队列架构的案例，你觉得采用何种并发模式是比较合适的，为什么？

欢迎你把答案写到留言区，和我一起讨论。相信经过深度思考的回答，也会让你对知识的理解更加深刻。（编辑乱入：精彩的留言有机会获得丰厚福利哦！）

大P

2018-06-10

IO操作分两个阶段1、等待数据准备好(读到内核缓存)2、将数据从内核读到用户空间(进程空间)一般来说1花费的时间远远大于2。1上阻塞2上也阻塞的是同步阻塞IO1上非阻塞2阻塞的是同步非阻塞IO，这讲说的Reactor就是这种模型 1上非阻塞2上非阻塞是异步非阻塞IO，这讲说的Proactor模型就是这种模型

作者回复

2018-06-10

解释很清楚����

大P

2018-06-11

Reactor与Proactor能不能这样打个比方：
1、假如我们去饭店点餐，饭店人很多，如果我们付了钱后站在收银台等着饭端上来我们才离开，这就成了同步阻塞了。
2、如果我们付了钱后给你一个号就可以离开，饭好了老板会叫号，你过来取。这就是Reactor模型。
3、如果我们付了钱后给我一个号就可以坐到坐位上该干啥干啥，饭好了老板会把饭端上来送给你。这就是Proactor模型了。

作者回复

2018-06-11

太形象了������

正是那朵玫瑰

2018-06-09

根据华仔之前对前浪微博消息中间件的分析，TPS定位在1380，QPS定位在13800，消息要高可靠（不能丢失消息），定位在常量连接海量请求的系统吧。基于此来分析下吧。

1、单Reactor单进程/线程
redis采用这种模式，原因是redis是基于内存的数据库，在处理业务会非常快，所以不会对IO读写进行过长时间的阻塞，但是如果redis开启同步持久化后，业务处理会变慢，阻塞了IO线程，也就无法处理更多的连接了，而我们的消息中间件需要消息的高可靠，必定要同步持久化，如果异步的话，就看异步持久化的时间间隔了，假设500ms持久化一次，那就有可能会丢失500ms的消息。当然华仔分析的无法利用多核cpu的特性也是一大缺点；虽然我们要求的TPS不算很高，但是QPS很高了，所以我觉得这种模式不合适
2、单Reactor多进程/线程
这种模式我觉得也不是和合适，虽然真正的业务处理在独立的线程了，IO线程并没有被阻塞，可以处理更多的连接和读写事件。我们的中间件可能不会面对海量的连接数，但是会面对大量的读请求，瓶颈是在处理读操作上，跟单Reactor单进程/线程差别不大；我倒觉得前一讲说的TPC prethread 模式是合适的，有独立的线程负责read-业务处理-send。
3、多Reactor多进程/线程
这种模式是最合适的了，不过华仔在讲解是read→业务处理→send，业务处理还是在IO线程上，如果业务处理的慢，还是会阻塞IO线程的，我觉得最好是业务处理放到独立的线程池里面去，这就变成了mainReactor负责监听连接，subReactor 负责IO读写，后面的业务线程池负责真正的业务处理，这样就既可以面对海量的连接，海量的请求也可以支撑。

不知理解的是否正确？

作者回复

2018-06-10

1. 分析正确，redis不太适合有的key的value特别大，这种情况会导致整个redis变慢，这种场景mc更好
2. prethread确实可以，mysql就是这种模式
3. 多reactor多线程再拆分业务线程，性能没有提升，复杂度提升不少，我还没见过这种方式。

空档滑行

2018-06-10

消息队列系统属于中间件系统，连接数相对固定，长链接为主，所以把accept分离出来的意义是不大的。消息中间件要保证数据持久性，所以入库操作应该是耗时最大的操作。综合起来我觉得单reactor，多线程/进程的方式比较合适。

作者回复

2018-06-10

分析正确

赵正 Allen

2018-06-10

一直做网络通讯相关的开发，用过ACE，boost asio。谈谈我的一些愚见，reactor pattern 主要解决io事件检测和事件分派，其中，事件检测一般都是通过封装OS提供API实现，在Linux下最常用epoll，事件分派是将检测到的事件委托给特定的方法，一般通过接口继承或函数指针实现。除此之外，定时器，信号量也会集成到reactor框架中。
多线程or多进程，实际工作中，基本多线程模型，可以单线程事件检测，多线程分派，也可以多线程轮流事件检测和分派。可以参考leader-follwers pattern。
io模式，一般都使用non-block。
与acceptor-connector模式结合使用，可进一步分离模块职责（即将 服务初始化与 服务逻辑分离. 由reactor统一进行事件驱动）

附一个自己开发的reactor框架

https://github.com/zhaozhencn/cute

赵正 Allen

2018-06-11

对于两组概念的理解，欢迎吐槽

阻塞&非阻塞

这一组概念并偏向于系统底层的实现，常与OS进程调度相关。 以socket为例，在阻塞模式下线程A调用recv函数，若此时接收缓冲区有数据，则立即返回，否则将进入”阻塞状态“（主动释放CPU控制权，由OS CPU调度程序重新调度并运行其它进程），直到”等待条件”为真，再由OS将此进程调度并重新投入运行。非阻塞模式则另辟蹊径，无论有无数据均立即返回（有数据则返回数据，无数据则返回错误值）， 不会影响当前线程的状态。 从某种意义上讲，阻塞模式下，一个线程关联一个文件fd，常引起线程切换与重新调度，对于高并发环境，这种代价太大。而非阻塞模式则解耦了“1线程关联1文件fd"。

同步&异步

调用与执行的分离即为异步，否则为同步。其实包括两个层面，其一为请求方（客户方），其二为执行方（服务方），抛开这两个概念单独讨论同步或异步是片面的。若请求方调用执行方的服务并等待服务结果，则为同步过程。但对于一些耗时或IO服务，服务执行时间往往较长或不可控，并可能导致降低整体服务质量，此时需要将调用与执行解耦。 有些经典设计模式常用于解决此问题： 1 command（命令模式）-- 将请求封装成命令对象，实现请求方对命令执行的透明化， 2 Active Object（主动对象）-- 对象内部驻留一个线程或线程池，用于执行具体服务，同时，对象对外提供服务接口，供请求方发起调用（可能获得Future对象）。

Bayern

2018-06-10

我能不能这样理解reactor，IO操作用一个连接池来获取连接，处理用线程池来处理任务。将IO和计算解耦开。这样就避免了在IO和计算不平衡时造成的浪费，导致性能低下。老师，我这样理解对吗

周飞

2018-07-05

nodejs的异步模型是io线程池来监听io，然后通过管道通信来通知事件循环的线程。事件循环线程调用主线程注册的回调函数来实现的。不知道这种模式跟今天说的两种相比有什么优缺点啊

作者回复

2018-07-05

我理解这就是Reactor模式

LinMoo

2018-06-09

请教两个问题 谢谢
之前学习NIO和AIO的时候是这么描述的：进程请求IO（无论是硬盘还是网络IO），先让内核读取数据到内核缓存，然后从内核缓存读取到进程。这里面就有2个IO等待时间，第一个是读取到内核缓存，第二个是读取到进程。前者花费的时间远远大于后者。在第一个时间中进程不做等待就是NIO，即非阻塞。第二个时间中进程也不需要等待就是AIO，即异步。
第一个问题：文章中说Reactor 是非阻塞同步网络模型，因为真正的 read 和 send 操作都需要用户进程同步操作。这里的read和send指的是我上面说的第二个时间吗?
第二个问题：因为我理解你的“来了事件我来处理，处理完了我通知你”。这里的我来处理就是包括第一和第二个时间吗?

感觉我之前被误解了，是我哪个地方理解不对吗？麻烦解答一下。

作者回复

2018-06-10

你两个问题的理解都正确

Geek_8242cb

2018-06-09

如果为了保证消息入库的顺序性，最好采用单Reactor单线程的模式。

作者回复

2018-06-10

这也是一种方式，但性能就需要好好设计了

码钉

2018-06-09

Proactor 可以理解为“来了事件我来处理，处理完了我通知你”。

请问一下这里的“处理”具体指什么? 把数据从内核层复制到应用层么?

作者回复

2018-06-10

包括从驱动读取到内核以及从内核读取到用户空间

沙亮亮

2018-07-17

两个轮询不是一个意思，select和poll是收到通知后轮询socket列表看看哪个socket可以读，普通的socket轮询是指重复调用read操作

感谢大神回复，那对于select实现的I/O多路复用技术，和普通的轮询区别在于，一个是socket上有数据时，系统通知select，然后select去轮询所有的socket，而普通的轮询就是一直不停的轮询所有socket。

还有一个有关真实应用场景中的问题，对于nginx+php-fpm这样一个场景，对于I/O多路复用技术，在nginx处理外部请求的时候用到了Reactor模式。当nginx把请求转发给php-fpm，然后php通过读数据库，代码业务逻辑处理完后，再从php−fpm读取数据，返回给客户端请求过程中，有没有再使用Reactor模式了?

作者回复

2018-07-17

1. 轮询理解OK
2. php-fpm我没有深入研究，你可以自己研究一下，这样学习效果会更好��

---

darrykinger.com

2018-07-16

发现这些对于我们这些学PHP开发的比较偏实现原理，在架构设计中，你说的这些貌似对于我们来说只是些软件配置的调优吧？或者说在以后得工作中遇到的问题的一种解决思路或者解决问题的思路？中间介绍的模型都了解过，那么作为PHPer不知道如何实现调试模拟?

作者回复

2018-07-16

学习下php-fpm的实现

沙亮亮

2018-07-16

根据unix网络编程上说的，select和poll都是轮询方式，epoll是注册方式。为什么您说select也不是轮询方式

作者回复

2018-07-16

两个轮询不是一个意思，select和poll是收到通知后轮询socket列表看看哪个socket可以读，普通的socket轮询是指重复调用read操作

孙振超

2018-07-14

ppc和tpc模式是每一个连接创建一个进程或线程，处理完请求后将其销毁，这样的性能比较低，为提升性能，首先考虑是链接处理完后不再销毁进程或线程，将这一部分的成本给降下来。改进后存在的问题是如果当前的链接没有请求又把进程或线程都给占住的情况下，新进来的链接就没有处理资源了。对此的解决方法是把io处理从阻塞改为非阻塞，这样当链接没有请求的时候可以去其他有请求的链接，这样改完后存在的问题有两个：一是寻找有请求的链接需要轮询需要耗费cpu而是当请求特别多的时候轮询一遍也需要耗费很长时间。基于这种情况引出了io多路复用，在处理进程和链接这之间加了一个中间人，将所有的链接都汇总到一个地方，处理进程都阻塞在中间人上，当某一个链接有请求进来了，就通知一个进程去处理。在具体的实现方式上根据中间人reactor的个数和处理请求进程的个数上有四种组合，用的比较多的还是多reactor和多进程。
之前的留言中有一个类比成去餐厅吃饭的例子还是蛮恰当的，肯德基麦当劳里面是reactor模式，需要用户先领个号然后等叫号取餐；海底捞和大多数中餐厅就是paractor模式，下完单后服务员直接将食品送过来。

回到文章中的问题，消息中间件的场景是链接数少请求量大，采用多进程或多线程来处理会比较好，对应单reactor还是多reactor应该都可以。

作者回复

2018-07-16

功能是ok的，但复杂度不一样，参考架构设计的简单原则

老北

2018-07-07

华仔，请教个问题。
redis是使用单reactor单进程模式。缺点是handler在处理某个连接上的业务时，整个进程无法处理其他连接的事件。
但是我做了个测试，在redis里面存放了一个1000w长度的list，然后使用lrange 0 -1全取出来，这会用很久。
这时候我新建个连接，继续其他key的读写操作都是可以的。不是应该阻塞吗?

作者回复

2018-07-09

很好的一个问题，这就是你去研究源码查看细节的最好时机了，参考特别放松"如何学习开源项目"。

周飞

2018-07-06

晚上看了朴灵写的 深入浅出nodejs和 unix网络编程，我认为node的异步模型其实是reactor 和proactor的结合。Windows下node用iocp，linux下用多线程和epoll来做多reactor。然后事件循环来执行主线程注册的回调函数，实现异步的io，这部分是proactor。

作者回复

2018-07-06

那还是Proactor，只是windows上的IOCP是标准的Proactor, linux 上是用epoll模拟Proactor

---

tiger

2018-06-24

老师，你好，你有说到单线程单连接的时候，线程会阻塞，。这种模型还可以通过协程(比如go语言的goroutine)方式解决，一个协程一个连接，因为协程属于用户态线程，轻量，所以同一时间可以产生大量的协程。这种方式就可以实现同步阻塞并发模型。

当然协程会增加额外的内存使用，另外也是通过线程轮询协程状态的方式驱动的

作者回复

2018-06-25

用户态线程也可以算作线程，因此这还是单线程单连接的

---

luck bear

2018-06-23

你好，我是小白，针对单reactor, 多线程的方式，负责处理业务的processor的子线程，是在什么时候创建，由谁创建，每来一个新链接，都要创建一个新的子线程吗?

作者回复

2018-06-23

代码实现请参考Doug Lee关于NIO的PPT

成功

2018-06-16

Reactor多线程模式，比较合适。父进程管理监听和accept，子进程负责创建Handle和处理

正是那朵玫瑰

2018-06-15

感谢华仔，我也再实验了下netty4，其实handler的独立的线程池里面执行其实也没有问题，netty已经帮我们处理好了，当我们处理完业务，write数据的时候，会先放到一个队列里面，真正出站还是由io线程统一调度，这样就避免了netty3的问题!

作者回复

2018-06-15

非常感谢，我明白了你说的情况，我再次验证了一下，写了一个独立线程处理业务的，确实如你所说，netty4两者都支持，并且做了线程安全处理，最终发送都是在io线程里面。

如果我们用这种模式，可以自己控制业务线程，因为netty4已经帮我们封装了复杂度，看来我孤陋寡闻了��

不过我建议还是别无条件用这种模式，我们之前遇到的情况就是短时间内io确实很快，并发高，但如果业务处理慢，会积压请求数据，如果客户端请求是同步的，单个请求全流程时间不会减少；如果客户端请求是异步的，如果积压的时候宕机会丢较多数据。

其实这种情况我理解单纯加大线程数就够了，例如5个io线程加20个业务线程能达到最优性能的话，我理解25个融合线程性能也差不多。

我们之前有一个案例，http服务器业务处理线程配置了512个，后来发现其实配置128是最好的(48核)，所以说并不是线程分开或者线程数量多性能就一定高。

再次感谢你的认真钻研，我也学到了一个技术细节��

大光头

2018-06-15

proactor明显比reactor更好，但是很多都没有实现，只能用reactor。对于前浪微博这么大吞吐量的业务，用多reactor多进程方式更好，因为业务比较复杂，同时要求稳定性，又高吞吐量。单reactor单进程和单reactor多线程都无法满足这个。

作者回复

2018-06-15

proactor没有明显好，理论上来说性能差异不会很大，更不用谈实际实现受各种因素影响了

周龙亭

2018-06-15

多reactor多线程模式中，如果是阻塞的业务处理也需要放到独立线程池去处理，不能直接在reactor线程处理。

作者回复

2018-06-15

可以的，独立线程处理业务，系统总吞吐量不会增加，因为瓶颈在业务处理部分，但复杂度增加很多，参考我另外一个回答，里面分析了netty4的实现

正是那朵玫瑰

2018-06-14

感谢华仔的解答，我看到在针对多reactor多线程模型，也有同学留言有疑问，我想请教下华仔，多reactor多线程模型中IO线程与业务处理在同一线程中，如果业务处理很耗时，定会阻塞IO线程，所以留言同学“衣申人”也说要不要将IO线程跟业务处理分开，华仔的答案是性能没有提升，复杂度提升很多，我还没见过这种处理方式，华仔对netty应该是很熟悉的，我的疑问是：
在netty中boss线程池就是mainReactor，work线程池就是subReactor，正常在ChannelPipeline中添加ChannelHandler是在work线程即IO线程中串行执行，但是如果pipeline.addLast(group, "handler", new MyBusinessLogicHandler());这样的话，业务hangdle就会在group线程池里面执行了，这样不就是多reactor多线程模型中把IO线程和业务处

理线程分开么？而且我在很多著名开源项目里面看到使用netty都是这样处理的，比如阿里的开源消息中间件rocketmq使用netty也是如此。华仔说没有见过这种处理方式，能否解答下？不知道是不是我理解错了

作者回复

2018-06-15

非常感谢你的认真研究和提问，我对netty的原理有一些研究，也用过netty3，也看过一些源码，但也还达不到非常熟悉的地步，所以不管是网上的资料，还是我说的内容，都不要无条件相信，自己都需要思考，这点你做的很好��

回到问题本身，由于netty4线程模型和netty3相比做了改进，我拿netty4.1源码中的telnet样例，在handler和NioEventloop的processSelectedKey函数打了输出线程id的日志，从日志结果看，StringEncoder, StringDecoder, TelnetServerHandler 都在NioEventLoop的线程里面处理的。

如果handler在独立的线程中运行，返回结果处理会比较麻烦，如果返回结果在业务线程中处理，会出现netty3存在的问题，channel需要做多线程同步，各种状态处理很麻烦；如果返回结果还是在io线程处理，那业务线程如何将结果发送给io线程也涉及线程间同步，所以最终其实还不如在一个线程里面处理。

LouisLimTJ

2018-06-14

你好，老师，可不可以系统的整理出一个性能数据表？由于时间和经验的原因，当进行架构选择时，我不太可能每个选择都去做测试。我相信这样一张表格会给大家有帮助。我知道影响性能的因素和变量很复杂。我自己在总结云预算管理的时候，我会定义自己的一套标准的的硬件资源，然后不同的选型和业务给出一个数值。如果要更细致的话，在资源方面可以定义高中低配置表现。性能数据的话，可以通过一个基准业务，复杂业务是基准的多少倍或者几分之几。

作者回复

2018-06-15

可以认为目前开源系统对外宣称的性能都是TPS/QPS上万，例如nginx，mc ,redis都是3～5万，mysql简单的k-v存储性能也能达到这个量级，kafka更高，10几万都有

爱逃课的橡皮擦

2018-06-14

“当多条连接共用一个阻塞对象后，进程只需要在一个阻塞对象上等待，而无须再轮询所有连接“ 华仔你好，这句话能详细解释下吗，多条连接怎么公用一个阻塞对象，为什么解决了轮训的问题，连接处于什么事件不去轮询怎么知道，是说自己不去轮询系统帮你轮询吗

作者回复

2018-06-14

看看epoll的原理，一个epoll对象可以注册很多连接，不用轮询的原因在于epoll注册了一个回调函数到内核驱动

正直D令狐勇

2018-06-13

“java用线程，c/c++用进程多些”，指的开发效率嘛？性能上来说还是线程有优势吧

作者回复

2018-06-14

这只是理论上的差异，实际上的性能取决很多因素，例如线程的并发处理就是性能容易出问题的地方

Tom

2018-06-13

PPC、TPC、Reactor和Proactor这四种模式在linux下和windows 下都是适用的吗，两个平台下有没有什么区别？

作者回复

2018-06-13

windows用tpc和Proactor, linux 用ppc和reactor多些，java用线程，c/c++用进程多些

---

hb

2018-06-11

业务处理假如是跟数据库打交道的一个线程同时处理io跟业务不会有问题吗

作者回复

2018-06-12

没有完美的方案，如果分开，io线程很快但写数据库慢的话，会积压请求数据，此时系统故障会丢很多数据

Geek_8242cb

2018-06-10

我理解nio框架Mina 就是多reactor多线程的模式

古德

2018-06-10

无论nio还是aio，都是操作系统自生的机制，java只是对底层api的封装，其他语言都有类似的封装。这样理解没错吧

星火燎原

2018-06-10

自然是多reactor模式啦

衣申人

2018-06-10

华仔，我有两个疑问：
1.单reactor多线程模式，业务处理之后，处理线程将结果传输给reactor线程去send，这个具体能怎么实现？reactor既要等待网络事件，又要等待业务线程的处理结果，然后作出响应，这个除了两边轮询还有更直接的方式吗？
2.多reactor多线程模型，现在你给出的方案是连接线程与io线程分开，但io线程与业务处理在一起的。而有的资料建议将io线程和业务线程分开，你认为有这个必要吗？

作者回复

2018-06-10

1. 处理线程将返回结果包装成一个事件，触发write事件，详细可以看看Doug Lee的NIO PPT，处理线程和业务线程共享selector，key这些对象
2. io线程与业务线程分开就是单reactor多线程，多reactor如果再分开的话，性能没有提升，复杂度提升很多，我还没见过这种处理方式

咖飞

2018-06-10

解决线程池利用率，文中说：解决这个问题的最简单的方式是将 read 操作改为非阻塞，然后进程不断地轮询多个连接。
请问只要将read操作改为非阻塞不就可以了吗？为啥还要轮询？

作者回复

2018-06-10

改为非阻塞，没有数据可读就立刻返回了，那什么时候才有数据可读呢？只能循环调用read操作了

咖飞

2018-06-10

老师你好，IO多路复用的两个关键点能否说的更具体一点，实在是很难理解。

作者回复

2018-06-10

《unix网络编程 卷一》有详细的说明，包括阻塞非阻塞，同步异步等

谭斌

2018-06-10

还是没有理解reactor中同步的意思...

作者回复

2018-06-10

参考Doug Lee的NIO PPT，亲手把里面的代码写一遍，测试一遍

星火燎原

2018-06-10

我的理解是前浪微博适合于 用一个主线程去监听连接请求，然后把请求丢到业务线程池后立即返回 通过回调函数处理业务逻辑。

作者回复

2018-06-10

那具体是哪种模式呢？单reactor多线程？

---

feifei

2018-06-10

我认为消息队列适合reactor单进程多线程方案，消息队列的连接不多，消息处理工作量大！这个特性与单进程多线程的方案特点匹配，连接在一个线程中完成，业务处理在其他线程中完成，这样能最大限度地利用资源

作者回复

2018-06-10

这个方案可以的

天天平安

2018-06-09

老师您好，一个短信计分析算平台的需求：一集大数据分析、数据挖掘、机器学习为一体的智能分析平台。比如一次一条或提交1万条短信，要根据如下七八个数据库（每个库的数据有的亿级别，并且每次对短信发送的结果来维护这些分析库）：客户数据库，
失败号码库，客户归属地市数据库，白名单库，黑名单库，买家信息，实号库和新号码库 等等分析出来这1万的短信哪些需要发送，哪些不需发送 做到客户的精细化运营。
这系统每天发送短信的量大概是5000万，短信的信息流程是：短信平台-->短信分析计算平台-->短信平台-->发送到运营商 。请问短信分析计算平台的架构怎么设计？

作者回复

2018-06-10

感觉是标准的流式处理，如果实时性要求高，用storm，如果实时性要求没那么高，spark streaming可能也可以。
不过我理解发送短信的实时性要求不会很高，因此方案可以简单一些

王磊

2018-06-09

我们用的是Vertx，我了解它是多Reactor，多线程的模式，Reactor只负责消息的分发，耗时的操作都在专有的线程池内操作，也可以方便的指定新的线程池的名字在自己的线程池内运行。
问题，这里明确一个定义，说单Reactor单进程/线程的时候，是否包含了Reactor的进程/线程？理解应该没有，所以'单Reactor单进程/线程'应该有2个进程/线程在工作？

作者回复

2018-06-10

vert.x在linux平台应该是reactor的。
单reactor单进程/线程是指reactor和handler都在同一进程/线程中运行，不是reactor占一个线程，handler占一个线程，redis的模型就是这样的，redis业务处理是单进程的

云学

2018-06-09

曾经用c++11实现了多reactor多线程网络库，微博应该用这种模式吧

作者回复

2018-06-10

微博我不清楚呢，不过类似的库很多

刘岚乔月

2018-06-09

感觉其根本都是从io nio nio2演变的
从同步阻塞到同步非阻塞(利用轮循机制 仅仅在select阻塞)在到异步非阻塞 (基于注册事件和回调机制)都是基于java基础

作者回复

2018-06-10

其实java都是基于操作系统来的呢，c/c++一样可以实现这些模式

---

olaf

2018-06-09

消息队列的业务的话，数据共享和同步应该设计很少所以主要发挥机器性能。采用nginx模式不错

作者回复

2018-06-10

类似nginx，nginx模式与标准的多reactor多线程有点差别，nginx每个子进程都可以accept