**为什么数据库连接池不采用 IO 多路复用？**

<https://mp.weixin.qq.com/s/V9D7QyQDy7luLM5NURJgnQ>

今天我们聊一个不常见的 Java 面试题：为什么数据库连接池不采用 IO 多路复用？

这是一个非常好的问题。IO多路复用被视为是非常好的性能助力器。但是一般我们在使用 DB 时，还是经常性采用c3p0，tomcat connection pool等技术来与 DB 连接，哪怕整个程序已经变成以Netty为核心。这到底是为什么？

首先纠正一个常见的误解。IO多路复用听上去好像是多个数据可以共享一个IO（socket连接），实际上并非如此。**「IO多路复用不是指多个服务共享一个连接，而仅仅是指多个连接的管理可以在同一进程」**。在网络服务中，IO多路复用起的作用是**「一次性把多个连接的事件通知业务代码处理」**。至于这些事件的处理方式，到底是业务代码循环着处理、丢到队列里，还是交给线程池处理，由业务代码决定。

对于使用DB的程序来讲，不管使用多路复用，还是连接池，都要维护一组网络连接，支持并发的查询。

为什么并发查询一定要使用多个连接才能完成呢？因为DB一般是使用连接作为Session管理的基本单元。**在一个连接中，SQL语句的执行必须是串行、同步的。**这是由于对于每一个Session，DB都要维护一组状态来支持查询，比如事务隔离级别，当前Session的变量等。只有单Session内串行执行，才能维护查询的正确性（试想一下一组sql在不断的增减变量，然后这组sql乱序执行会发生什么）。维护这些状态需要耗费内存，同时也会消耗CPU和磁盘IO。这样，限制对DB的连接数，就是在限制对DB资源的消耗。

因此，对DB来说，关键是要限制连接的数目。这个要求无论是DB连接池还是NIO的连接管理都能做到。

这样问题就绕回来了，为什么DB连接不能放到IO多路复用里一并执行吗？为啥大家都用连接池？

答案是，可以用IO多路复用——但是**「使用JDBC不行」**。JDBC是一个出现了近20年的标准，它的设计核心是BIO（因为199X年时还没有别的IO可以用）：调用者在通过JDBC时执行比如query这样的API，在没有执行完成之前，整个调用线程被卡住。而类似于Mysql Connector/J这样的driver完备的实现了这套语义。

当然如果DB Client的协议的连接处理和解析稍微改一下：

1. 将IO模式调整为Non-Blocking，这样就可以挂到IO多路复用的内核上（select、epoll、kqueue……）
2. 在Non-Blocking实现的基础之上实现数据库协议的编码和解析

就可以实现用IO多路复用来访问DB。实际上很多其他语言/框架里都是这么干的。比如 Nodejs，see https://github.com/sidorares/node-mysql2；或者 Vert.X 的 db 客户端https://github.com/mauricio/postgresql-async，不要在意这个名字，它实际上同时支持mysql和postgres）。只不过对于IO多路复用，数据库官方似乎都没做这种支持——他们只支持JDBC、ODBC等等这些标准协议。

那么为什么基于 IO 多路复用的实现不能成为默认的，官方的，而要成为偏门呢？

对于数据库开发者来说。这种用法在整体的用户里占有量非常小，所以也许不值当的花大力气。只需要把协议写清楚（比如https://dev.mysql.com/doc/internals/en/client-server-protocol.html），就可以做实现。那么社区的有兴趣的人自然就可以去做。

另外一个原因是体系的支持。简单来讲，如果没有一个大的 Reactive 的运行环境，IO 多路复用的使用会非常受限。

IO 多路复用之所以能成立，是需要**「整个程序要有一个IO多路复用的驱动代码」**——就是 select 那句调用——等待事件来临，一个 blocking 的 API。整个程序必须以这个驱动代码为核心。这样就对整个代码的结构产生重大的影响。这种影响是没法用简单的接口抽象的。

Java Web 容器之所以可以使用 NIO 是因为 NIO 可以被封装到容器内部。Web 容器对外暴露的还是传统的多线程形式的Java EE接口。

如果 DB 和 Web 容器同时使用 NIO，那么调用的DB连接库与必须与容器有一个约定描述**「DB的连接管理如何接入Web容器的NIO的驱动代码」**。在 Java 这个大环境下，不同人，不同的容器写的代码不同；又或者，不使用任何常见的容器，而是自己用 NIO 去封装一个。这样是无法形成代码上的约定的。那么多个独立的组件就不能很好的共享 NIO 的驱动代码。

上面这个用法假设整个程序应该共享一个 NIO 驱动代码。那么 Web 和 DB 可不可以各用各的呢？也是可以的，但是为了保证这两个 NIO 驱动代码不会相互 block，最好要分开两个线程。这样一来就会打破一般 Web 服务一个请求处理用一个线程的一般做法，会让程序边的更复杂——你的业务代码和DB查询之间必须做跨线程数据交换。

相反，连接池的实现就相对独立的多，也简单的多。外界只要配好 DB URL，用户名密码和连接池的容量参数，就可以做到自行管理连接。

而Nodejs和Vert.X是完全不同的。他们本质就是Reactive的。他们的NIO的驱动方式是其运行时的基础——所有要在这个基础上开发的代码都必须遵守同样的NIO+异步开发规范，使用同一个NIO的驱动。这样DB与NIO的协作就不成问题了。

最后，**「有大量场景是需要BIO的DB查询支持的」**。批处理数据分析代码都是这样的场景。这样的程序写成NIO就会得不偿失——代码不容易懂，也没有任何效率上的优势。类似于Nodejs这样的运行时在此场景下，反而要利用async或等价的语法来让代码看起来是同步的，这样才容易写。

总结一下。DB 访问一般采用连接池这种现象是生态造成的。历史上的 BIO + 连接池的做法经过多年的发展，已经解决了主要的问题。在 Java 的大环境下，这个方案是非常靠谱的，成熟的。而基于 IO 多路复用的方式尽管在性能上可能有优势，但是其对整个程序的代码结构要求过多，过于复杂。当然，如果有特定的需要，希望使用 IO 多路复用管理 DB 连接，是完全可行的。