# 进程

# 线程

# 协程

## Java协程

Quasar

一个比较成熟的java三方协程库，熟悉之后可用于生产环境。

Alibaba ajdk

Project Loom

oracle的jvm级项目，重新实现线程模型，里面包含协程方案，目前Quasar作者已经加入。（oracle忙着发JDK呢，这个还在无限延期）

kotlin

kotlin原生支持携程，且也是基于jvm运行的语言并且可以相互调用，可以考虑相互协作。

## go协程

## 为什么 Java 仍然是多线程而不选择协程？是否无法解决高并发问题?

作者：大宽宽  
链接：https://www.zhihu.com/question/332042250/answer/734115120  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

先说结论：协程是非常值得学习的概念，它是多任务编程的未来。但是Java全力推进这个事情的动力并不大。

先返回到问题的本源。当我们希望引入协程，我们想解决什么问题。我想不外乎下面几点：

* 节省资源，轻量，具体就是：
  + 节省内存，每个线程需要分配一段栈内存，以及内核里的一些资源
  + 节省分配线程的开销（创建和销毁线程要各做一次syscall）
  + 节省大量线程切换带来的开销
* 与NIO配合实现非阻塞的编程，提高系统的吞吐
* 使用起来更加舒服顺畅（async+await，跑起来是异步的，但写起来感觉上是同步的）

我们分开来讲下。

先说内存。拿Java Web编程举例子，一个tomcat上的woker线程池的最大线程数一般会配置为50～500之间（目前springboot的默认值给的200）。也就是说同一时刻可以接受的请求最多也就是这么多。如果超过了最大值，请求直接打失败拒绝处理。假如每个线程给128KB，500个线程放一起的内存占用量大概是60+MB。如果真的有瓶颈，也许CPU，IO，带宽，DB的CPU等会有瓶颈，但这点内存量的增幅对于动辄数个GB的Java运行时进程来说似乎**并不是什么大问题**。

换一个场景，比如IM服务器，需要同时处理大量空闲的链接（可能要几十万，上百万）。这时候用connection per thread就很不划算了。但是可以直接改用netty去处理这类问题。你可以理解为NIO + woker thread大致就是一套“协程”，只不过没有实现在语法层面，写起来不优雅而已。问题是，你的场景真的处理了并发几十万，上百万的连接吗？

再说创建/销毁线程的开销。这个问题在Java里通过线程池得到了很好的解决。你会发现即便你用vert.x或者kotlin的协程，归根到底也是要靠线程池工作的。goroutine相当于设置一个全局的“线程池”，GOMAXPROCS就是线程池的最大数量；而Java可以自由设置多个不同的线程池（比如处理请求一套，异步任务另外一套等）。kotlin利用这个机制来构建多个不同的协程scope。这看起来似乎会更灵活一点。

然后是线程的切换开销。线程的切换实际上只会发生在那些“活跃”的线程上。对于类似于Web的场景，大量的线程实际上因为IO（发请求/读DB）而挂起，根本不会参与OS的线程切换。现实当中一个最大200线程的服务器可能同一时刻的“活跃线程”总数只有数十而已。其开销没有想象的那么大。为了避免过大的线程切换开销，**真正要防范的是同时有大量“活跃线程”**。这个事情我自己上学的时候干过，当时是写了一个网络模拟器。每一个节点，每一个链路都由一个线程实现。模拟跑起来后，同时的活跃线程上千。当时整个机器瞬间卡死，直到kill掉这个程序。

此外说说与NIO的配合。在Java这个生态里Java NIO/Netty/Vert.X/rxJava/Akka可以任意选择。一般来讲，Netty可以解决绝大部分因为IO的等待造成资源浪费的问题。Vert.X/rxJava。可以让程序写的更加“优雅”一点（见仁见智）。Akka就是Java世界里对“原教旨OO“的实现，很有特色。的确，用NIO + completedFuture/handler/lambda不如async+await写起来舒服，但起码是可以干活的。

如果真的要较真Java的NIO用于业务的问题，其**核心痛点应该是JDBC**。这是个诞生了几十年的，必须使用Blocking IO的DB交互协议。其上承载了Java庞大的生态和业务逻辑。Java要改自己的编程方式，必须得重新设计和实现JDBC，就像[https://github.com/vert-x3/vertx-mysql-postgresql-client](https://link.zhihu.com/?target=https://github.com/vert-x3/vertx-mysql-postgresql-client" \t "_blank) 那样做。问题是，社区里这种“异步JDBC”还没有支持oracle、sql server等传统DB。对mysql和postgres的支持还需要继续趟坑～

如果认真阅读上面这些需要“协程”解决的问题，就会发现基本上都可以以各种方式解决。觉得线程耗资源，可以控制线程总数，可以减少线程stack的大小，可以用线程池配置max和min idle等等。想要go的channel，可以上disruptor。可以说，Java这个生态里尽管没有“协程”这个第一级别的概念，但是要解决问题的工具并不缺。

Java仅仅是**没有解决”协程“在Java中的定义，以及“写得优雅“这个问题**。从工程角度，“写得优雅”的优势并没有很多追新的人想象的那么关键。C#也并非因为有了async await就抢了Java的市场分毫。而反过来，如果java社区全力推进这个事情，Java历史上的生态的积累却因为协程的出现而进行大换血。想像一下如果没有thread，也没有ThreadLocal，@Transactional不起作用了，又没有等价的工具，是不是很郁闷？这么看来怎么着都不是个划算的事情。我想Oracle对此并不会有太大兴趣。OpenJDK的loom能不能成，如果真的release多少Java程序员愿意使用，师母已呆。据我所知在9012年的今天，还有大量的Java6程序员。

其他新的语言历史包袱少，比较容易重新思考“什么是现代的multi-task编程的方式“这个大主题。kotlin的协程、go的goroutine、javascript的async await、python的asyncio、swift的GCD都给了各自的答案。如果真的想入坑Java这个体系的“协程”，就从kotlin开始吧，毕竟可以混合编程。

最后说一句，多线程容易出bug主要因为：

* “抢占“式的线程切换 —— 你无法确定两个线程访问数据的顺序，一切都很随机
* “同步“不可组装 —— 同步的代码组装起来也不同步，必须加个更大的同步块

协程能不能避免容易出bug的缺陷，主要看能不能避免上面两个问题。如果协程底层用的还是线程池，两个协程还是通过共享内存通讯，那么多线程该出什么bug，多协程照样出。javascript里不出这种bug是因为其用户线程就一个，不会出现线程切换，也不用同步；go是建议用channel做goroutine的通讯。如果go routine不用channel，而是用共享变量，并且没有用Sync包控制一下，还是会出bug。