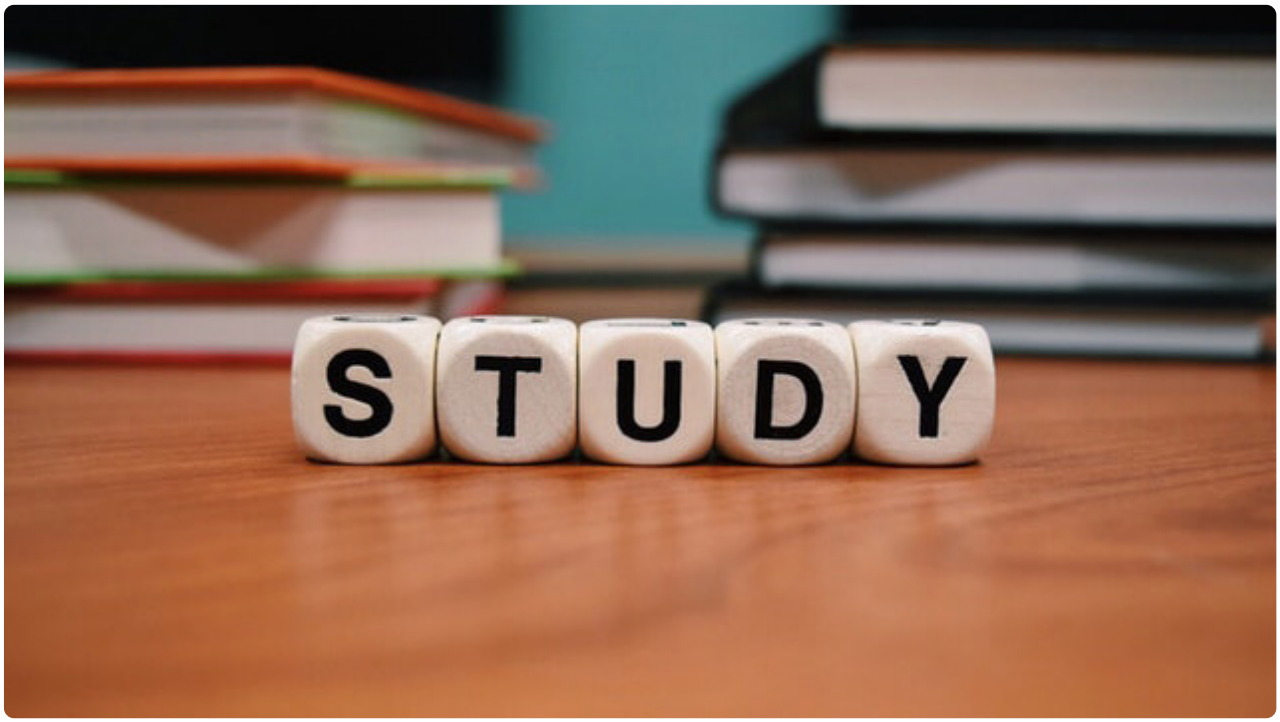
02 绝对不仅仅是为了面试—我们为什么需要学习多线程

更新时间：2019-09-10 15:07:23



> 智慧，不是死的默念，而是生的沉思。
>
> ——斯宾诺莎

此时此刻，是何种原因促使你打开了本篇关于 Java 并发的专栏？实事求是地讲，对于绝大多数研发人员，平时用到多线程的场景并不多。但多线程在我们的日常开发中却无处不在，只不过很多时候，框架已经帮你实现了。比如 web 开发，容器已经帮你实现了多线程；再比如大数据开发，框架也已帮你实现了多线程，甚至分布式计算。那促使你学习多线程的原因是什么呢？我想很大可能你是为了面试打基础、做准备。没错，这真的很现实！从我最近换工作的经历来看，多线程在面试题中出现的概率几乎是 100%。如果你想升职加薪！加入一线大厂！成为互联网精英！多线程的知识储备是必备的。但你想过吗，为什么面试官热衷于问一个平时用到并不多的技术问题？

1. 面试官考查多线程的原因

我想除了工作确实需要之外，面试官考察多线程可能有如下原因：

考察你的工作技术深度。

多线程虽然很少用到。但是如果你做底层开发，或者负责基础设施（例如消息队列）研发，肯定会用到多线程。通过面试多线程，可以考察你的工作在技术方面的深度。

考察你的学习、理解能力。

面试大概率会考多线程问题，这已经是公开的秘密了。这其实是一个开卷考试，对所有候选人是公平的。比拼的是候选人的学习能力、理解能力、做事的态度。你可以没用过，但你要有快速掌握的能力，和稳扎稳打的学习态度。

我认为第二点是主要原因。求职者都知道面试官会考查多线程，但为什么还是有的人答非所问，有的人却对答如流，有的人甚至可以深入底层原理？这无外乎两个原因：

1. 对面试的准备和态度。明知道要考察多线程，候选人却不认真准备，这种态度带到工作中是何其的可怕？
2. 学习的能力。短时间内掌握平时不常用到的多线程并不容易。彻底理解多线程，还需要 JVM 的知识。这除了自身的学习能力外，如果配合一本好的教材、几篇好的博客，能够大大加快你的学习速度、提升你的学习深度。

生活在知识爆炸的时代，怕的不是没有选择，而是不知道怎么选。其实市面上关于多线程编程的书籍太多了，那为什么我还要花时间写这个专栏呢？我在准备这个专栏前，买了 7 本多线程相关书籍。全部通读下来后，感觉质量参差不齐。有的讲得比较浅，有的讲得够深入却晦涩难懂，而且每本书的写法和侧重点都不一样。我写这个专栏的目的，是想站在巨人的肩膀上，以更为通俗易懂的方式，把多线程的知识讲出来。让从来没接触过多线程的开发人员也能有兴趣读、能够读懂。并且能够深入到底层原理，而不是蜻蜓点水。

2. 软件世界即现实世界

再回到题目上，虽然可能绝大多数读者是抱着提升自身实力，为面试做准备的初衷来学习多线程。但我想告诉大家，多线程真的很强大，有很多使用场景，能帮你解决很多问题。在学习完多线程后，你手中便多了一样武器，你解决问题的思路也更为宽广。在你以后漫漫的编程生涯中，从此多了一种选择。所以学习多线程，绝对不是仅仅为了面试。

其实多线程并不复杂，其实和现实世界中多人协作是一样的。编程初学者，会觉得软件是无形的，看不见、摸不到，只有冰冷冷的逻辑，学习起来晦涩难懂。其实从面向对象出现开始，软件已经成为现实世界的对等映射。这不光体现在语言本身，其实在软件领域无处不在，例如：

设计模式

23 种经典设计模式，没有哪一种不是从现实世界得来的灵感。如果你看过设计模式的文章，你一定对设计模式中生动有趣的例子所吸引。

软件设计

绝大多数软件的设计，都参考了工业设计或者参考了生活中解决问题的方式，汲取其中的设计思想。其实不管软件还是硬件或者生活中遇到的难题，在解决问题的思路上是一致的。无形的软件设计，可以借助有形世界里的案例来帮助你思考。我最近在看 kafka 的源代码，其中 producer 的设计思想和快递公司发快递的过程很类似。还有 Java NIO，也是类似的原理。可以说软件设计的思想都发源于现实世界。

软件架构

我做个类比，软件架构可以看作现实世界工厂里的机器设计和布置。我们需要考虑很多，比如需要哪些机器，不同机器如何配比、不同工序之间如何衔接、机器出问题如何应对、机器操作日志如何记录、安全如何保障。工厂里遇到的问题在软件架构上也都会遇到。

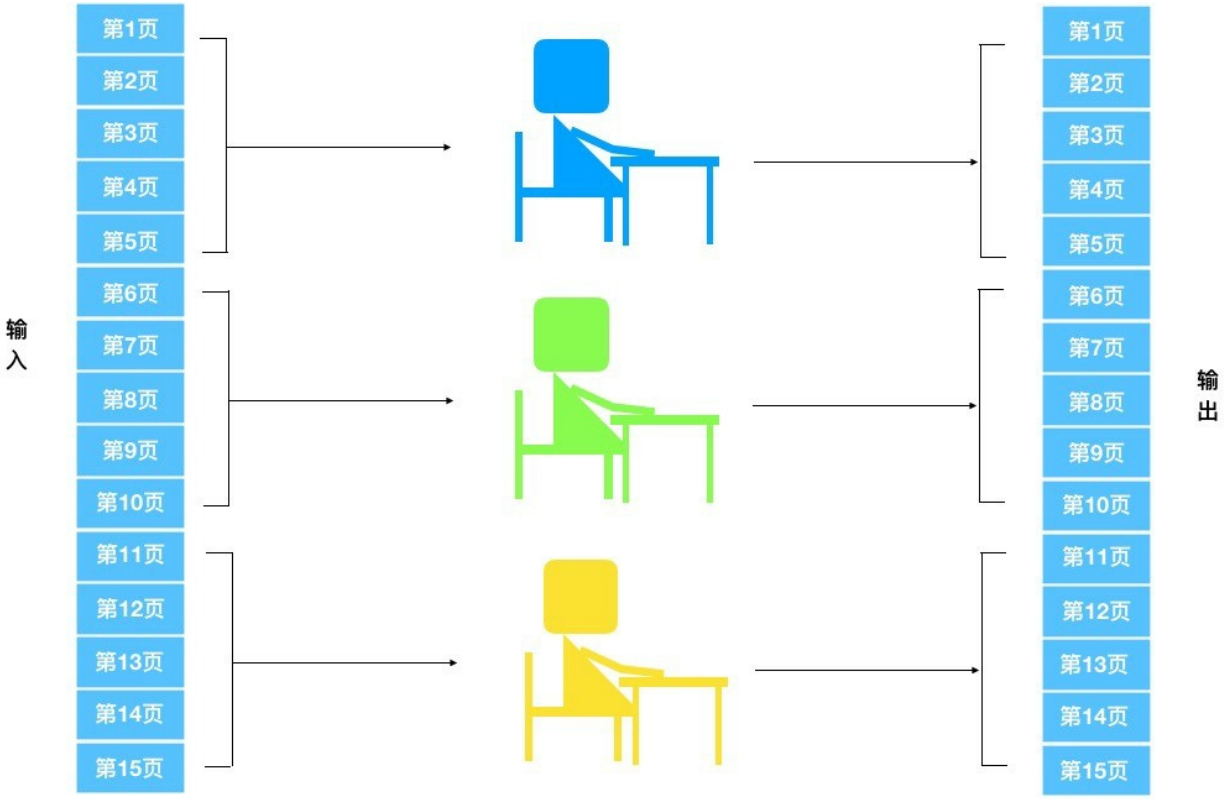
以上举例，足以说明软件和现实世界之间的相似程度。软件其实就是现实世界的映射。我们在学习软件的过程中，要善于找到生活中常见的例子类比，这样理解起来就没有困难了，而且便于记忆。

3. 多线程典型应用场景

啰嗦了这么多，主要是为了介绍我在编程上面的学习心得，希望能对大家有些帮助。下面我们就来看看多线程的几种典型应用场景，以下例子都由现实世界的场景切入讲解。在现实世界中，我们可以认为每个人都是一个线程，当多个人一起完成一项工作，这其实就是多线程。我们来看下面的多线程场景：

工作量太大，需要多人一块干，以缩短工期

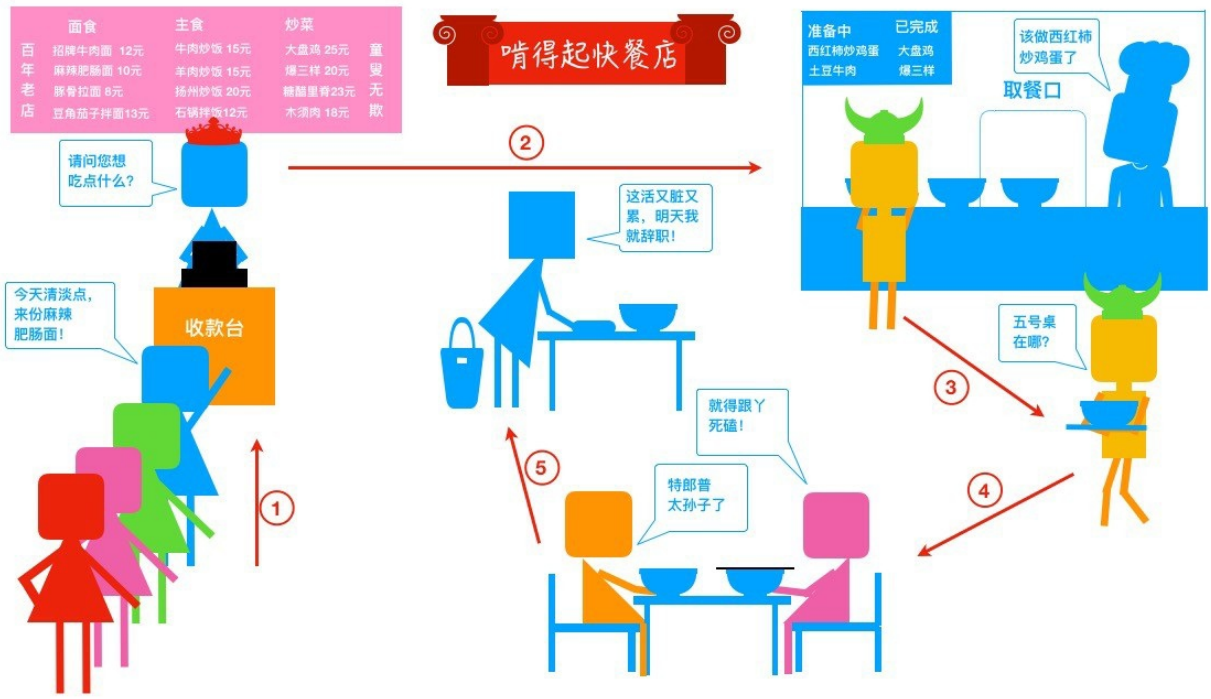
这种场景在现实生活中比比皆是，比如要完成书稿校对工作。显然一个人校对太慢了，那就多叫几个人吧！每个人分一个章节，同时进行校对，速度一下就上来了。如下图所示：



编程上，如果程序需要重复执行一段逻辑，每次执行又互不影响，那么你可以考虑采用多线程，每个线程执行任务总量的一部分，最后再把每个线程执行的结果合并。通过并行处理，能够大大减少执行时间。Java 8 开始出现的 lambda 并行流，就是采用的这种思想，只不过它是 JVM 去实现，而不需要我们做额外的处理。另外在大数据领域，对于海量数据的处理，也可以采用多线程，缩短执行时间。

实现分工

这个场景的例子也很多，而且很贴近我们的生活。例如，我们每天中午都会去吃工作餐，饭馆的工作流程大同小异，如下图所示：



饭店会有这么几类员工，收银员、厨师、传菜员、清洁员。每个人各司其职，大家配合工作。饭店的工作流程如下：

1、顾客在收款台点单；

2、后厨接到系统传过来的订单后开始加工；

3、做好饭菜后传菜员取饭菜；

4、传菜员找到客户所在位置上菜；

5、顾客用餐后，清洁员进行打扫。

每种角色的员工只关心自己的输入和输出。比如厨师的输入就是客户的点菜单，输出就是饭菜。而厨师的输入则是上个环节收银员的输出。这样做的好处是每个人专注于自己的工作，有助于效率的提升。其实好处还很多，我总结如下：

- 每种角色对应一个环节，每个环节在执行上独立分开。这样每个环节的工作就解耦了；
- 每个环节之间有了缓冲。收银员一直在收银，她不需要知道厨师是否空闲，她在不停输出订单。而厨师接到订单就去加工，而不关心积累了多少订单，只要一份菜接一份菜的去加工。订单的列表就是一个缓冲，调节两个环节速率的不匹配及不稳定。如果一个人干所有的事情，那么问题就来了。举个反面的例子，某著名连锁便利店，在早餐时段，收银员即收银又负责做咖啡并配餐，结果导致整个收银的队伍相当的长。我即使只买个面包，也要排队很久；
- 每种角色只做自己的事情，省去了上下文切换的时间。如果你一个人干所有的事情，当你为顾客下单完成后，要跑去后厨炒菜，再端给客户，然后再回到收银台为下一位顾客下单。单单是浪费在路上的时间就会有多少啊！而且每次切换工作，你都要在脑海里想一下接下来的这个工作需要怎么做；
- 我们看下清洁员这个角色。他看到有人吃完饭离开就会去收拾桌子。假如没有分工，而是一个人干所有的工作，那么餐厅员工给客人端上饭菜后，还要一直等到客户吃完饭，才能收拾桌子，效率何其低下。我想没有老板傻到会让自己的员工如此工作；

- 便于对原有流程进行改变。假如老板想在点餐前，增加向客户推销关注店铺公众号，并注册会员的环节。如果没有分工，老板要向所有员工通知这个事情，并且组织所有人学习。但是有了分工后，只有收银员需要进行学习。而其他角色的员工完全不需要知道这件事情。老板是不是轻松多了？

通过分工，多人协作，餐厅的工作才能高效运转起来。我们开发的程序也是如此，如果你所有的工作都在一个线程里，那么首先这段主逻辑会相当复杂，而且难于维护和扩展，另外相信效率也会相对低下。如果我们的程序通过多线程 + 缓冲的方式，把不同步骤解耦，那么将大大提高效率。

还是拿 kafka 举例，Kafka 的 producer 发送消息的机制就是如此，首先不同的发消息线程会往缓存中累积消息，此时消息没有被真正发送出去，只是累积在本地缓存中。Kafka 有专门负责网络 IO 的 sender 线程，当缓存满了，sender 线程被唤醒，它真正把消息发送出去，而此时新的消息还会被累积进来。

Kafka 的这种多线程设计，使得收集消息和 IO 发送消息解耦。sender 线程可以根据消息发往主机的不同，把消息分类打包，一次网络 IO 可以发送出多条消息，从而大大减少了网络 IO 的消耗。

我们再想想清洁员所做的工作，是不是很熟悉？没错，其实 JVM 中的 GC 线程就相当于清洁员。

分头行动，最后汇合

这也是分工协作的一种，只不过是分头行动后，大家要把行动的结果汇总，才能执行接下来的任务。接下来这个例子，作为研发同学再熟悉不过了。现在 BS 软件开发，前后端分离已经成为了趋势，在这种开发方式下，一般分为三步：

1、前、后端研发定义接口；

2、前、后端开发分头开发；

3、前、后端联调。

第 3 步的前提是第 2 步。在第 2 步中，前后端程序员分头进行开发，谁先开发完都没有用，只有二者都开发完了，才能进入第三步。

在微服务大行其道的时代，类似上面这种场景的多线程应用很常见。例如，你的一个业务接口中，可能会调用数个微服务接口获取数据。如果你没有采用多线程，那么每次请求时，主线程都会被阻塞。但是假如你采用了多线程开发，对微服务的几个请求可以同时发出，主线程阻塞时间只取决于几个请求中最长的那个，而不是所有请求阻塞时间之和，这样会极大地提高响应速度。

排队的同时，不耽误做其他事情

我平时很讨厌排队，所以看到做什么事情要排队，我就放弃了，因为排队给我的感觉就是在浪费时间，什么都做不了。当然，在移动互联网时代，排队时刷刷手机还是可以的。有没有一种方式，能让我把队排了，但不耽误我做别的事情呢？当然有，我举个例子。我们都应该做过体检，检查项目中最慢的就是 B 超。我说过我最不爱排队，所以我都是最后才去做 B 超，但每次还是要排队半个小时以上。近几年我去体检时，发现流程优化了。你先去 B 超排个号，然后可以先去做其他项目的检查，当你听到叫号你所在的区间时，再去做 B 超。这个流程改进只需要添加一个要素，就是发你一个号码。拿到你的序号牌后，你可以去做其他事情。等叫到你的号，你可以凭号进行 B 超检查。

这其实就是 java 多线程中的 Future 模式。这种模式下，主线程不会因为一个耗时的业务操作而被阻塞住，主线程可以单起一个线程去处理耗时的操作，主线程逻辑继续执行，等用到另外线程返回的数据时，再通过 Futrue 对象获取。Future 就是你的一张旧船票，你凭借这张旧船票，还能登上那艘客船。

4. 总结

本节我们了解了多线程的应用场景。其实除了文中列举的，还有许多其它使用多线程的场景。现实世界中几乎所有的工作都需要多人协作，而计算机的世界亦是如此。了解完多线程各种应用场景，下面就让我们开启 Java 多线程的学习之旅吧！

}

01 开篇词-多线程为什么是你必需要掌握的知识

03 多线程开发如此简单—Java中如何编写多线程程序