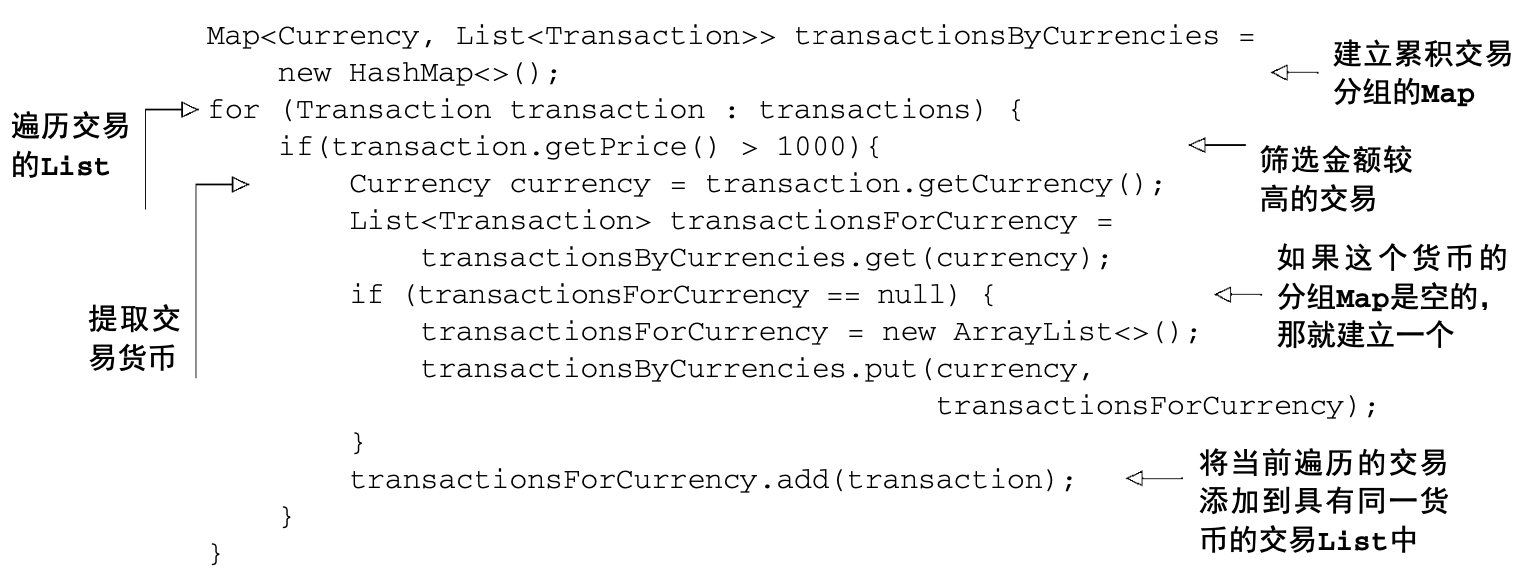
几乎每个Java应用都会制造和处理集合。但集合用起来并不总是那么理想。比方说，你需要从一个列表中筛选金额较高的交易，然后按货币分组。你需要写一堆套路化的代码来实现这个数据处理命令，如下命令：



因为有好几个嵌套的控制流指令很难看出代码是做什么的。有了Stream API，就可以这样解决这个问题了：

和Collection API相比，Stream API处理数据的方式非常不同。用集合的话，你得自己去做迭代的过程。相反，有了Stream API，你根本用不着操心循环的事情。数据处理完全是在库内部进行的。我们把这种思想叫做内部迭代。

使用集合的另一个头疼的地方是，要是你的交易量非常庞大，你要怎么处理这个巨大的列表呢？单个CPU根本搞不定这么大量的数据。但你很可能已经有了一台多核电脑。理想的情况下，你可能想让这些CPU内核共同分担处理工作，以缩短处理时间。理论上来说，要是你有八个核，那并行起来，处理数据的速度应该是单核的八倍。

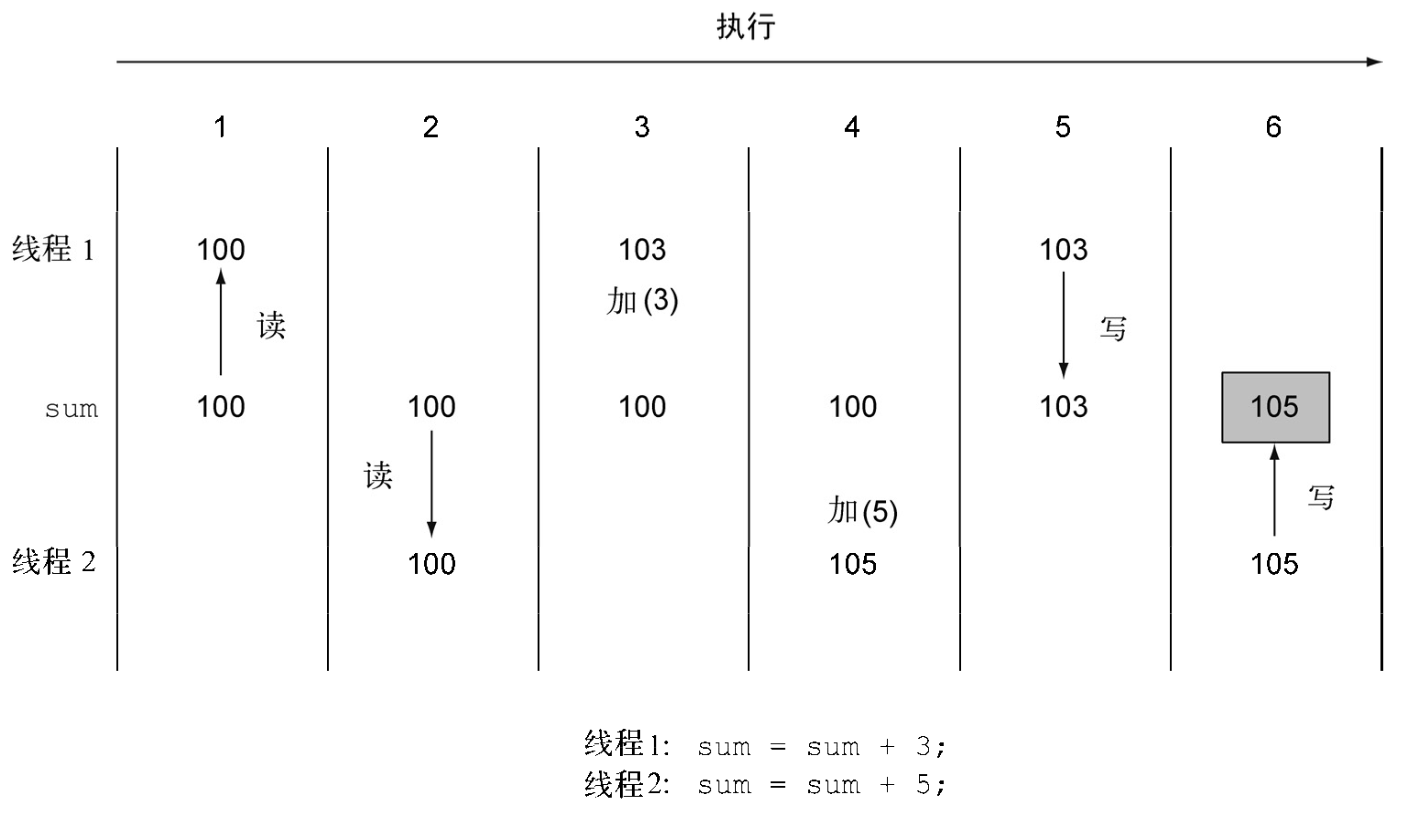
多核

现在电脑都是多核的，不仅有一个CPU，通常称为内核。经典的Java程序只能利用其中一个核，其他核的处理能力都浪费了。类似地，很多公司利用计算集群（用高速网络连接起来的多态计算机）来高效处理海量数据。Java 8提供了新的编程风格。

Google的搜索引擎就是一个无法在单台计算机上运行的代码的例子。它要读取互联网上每个页面并建立索引，将每个互联网网页上出现的每个词都映射到包含这些词的网页集合。想想看，你会如何在Java中实现这个算法，哪怕是比Google小的引擎也需要你利用计算机上的所有的核。

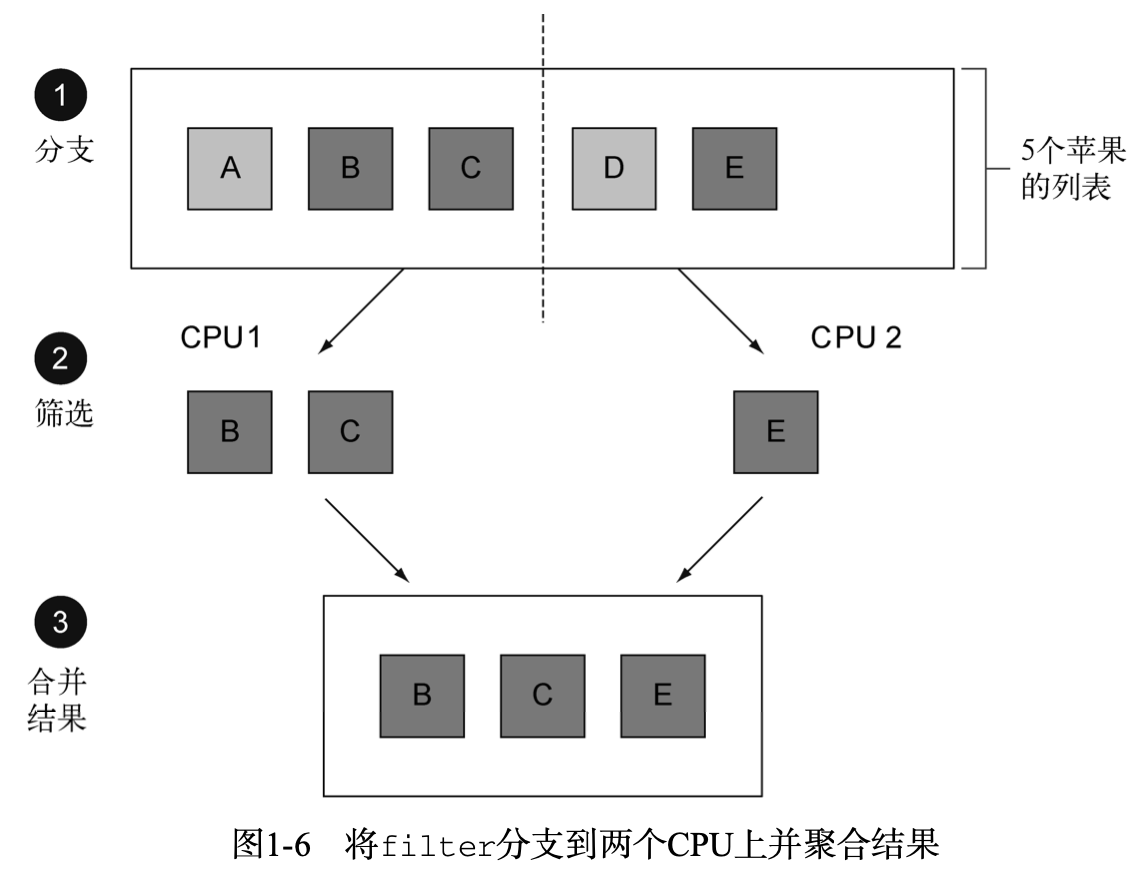
#### 多线程并非易事

问题在于，通过多线程代码来利用并行（使用先前Java版本中的Thread API）并非易事。得换一种思路：线程可能会同时访问并更新共享变量。因此，如果没有协调好，数据可能会被意外改变。相比一步步执行的顺序模型，这个模型不太好理解。如图所示，展示了如果没有同步好，两个线程同时向共享变量sum加上一个数时，可能出现的问题。



两个线程对共享的sum变量做加法的一种可能方式。结果是105，而不是预想的108。

Java 8也用Stream API（java.util.stream）解决了这两个问题：集合处理时的套路和晦涩，以及难以利用多核。这样设计的第一个原因是，有许多反复出现的数据处理模式，类似于前一节所说的filterApples或SQL等数据库查询语言里熟悉的操作，如果在库中有这些地方就会很方便：根据筛选数据（比如较重的苹果），提取数据（例如抽取列表中每个苹果的重量字段），或给数据分组（例如，将一个数字列表分组，奇数和偶数分别列表，可以让一个CPU处理列表的前一半，第二个CPU处理后一半，这称为分支步骤①。CPU随后对各自的半个列表做筛选②。最后③，一个CPU会把两个结果合并起来（Google搜索这么快就与此紧密相关，当然他们用的CPU远远不止两个）。



到这里，我们这只是说新的Stream API和Java现有的集合API的行为差不多：它们都能够访问数据项目的序列。不过，现在最好记得，Collection主要是为了存储和访问数据，Stream则主要用于描述对数据的计算。这里的关键点在于，Stream允许并提倡并行处理一个Stream中的元素。下面举的例子如何利用Stream和Lambda表达式顺序或并行地从一个列表里筛选比较重的苹果。

//顺序处理

//将其转换为Stream，进行并行处理，然后再转换回List

List<Apple> heavyApples3 = inventory.stream()

.filter((Apple a) -> a.getWeight() > 150)  
 .collect(Collectors.*toList*());

//并行处理

//将其转换为parallelStream，进行并行处理，然后再转换回List

List<Apple> heavyApples4 = inventory.parallelStream()

.filter((Apple a) -> a.getWeight() > 150)  
 .collect(Collectors.*toList*());

Java 8设计者发现的一个现实问题就是现有的接口也在改进。比如，Collections.sort方法真的应该属于List接口，但却从来没有放在后者里。理想情况下，你会希望做list.sort(comparator)，而不是Collections.sort(list, comparator)。这看起来无关紧要，但是在Java 8之前，你可能会更新一个接口，然后发现你把所有实现它的类也给更新了——简直是逻辑灾难！这个问题在Java 8里由默认方法解决了。

Java中的并行与无共享可变状态

在Java 8里，首先，库会负责分块，即把大的流分成几个小的流，以便并行处理。其次，流提供的这个几乎免费的并行，只有在传递给filter之类的库方法的方法不会互动（比方说有可变的共享对象）时才能工作。但是其实这个限制对于程序员来说挺自然的，举个例子，我们的Apple::isGreenApple就是这样。确实，虽然函数式编程中的函数的主要意思是“把函数作为一等值”，不过它也常常隐含着第二层意思，即“执行时在元素之间无互动”。