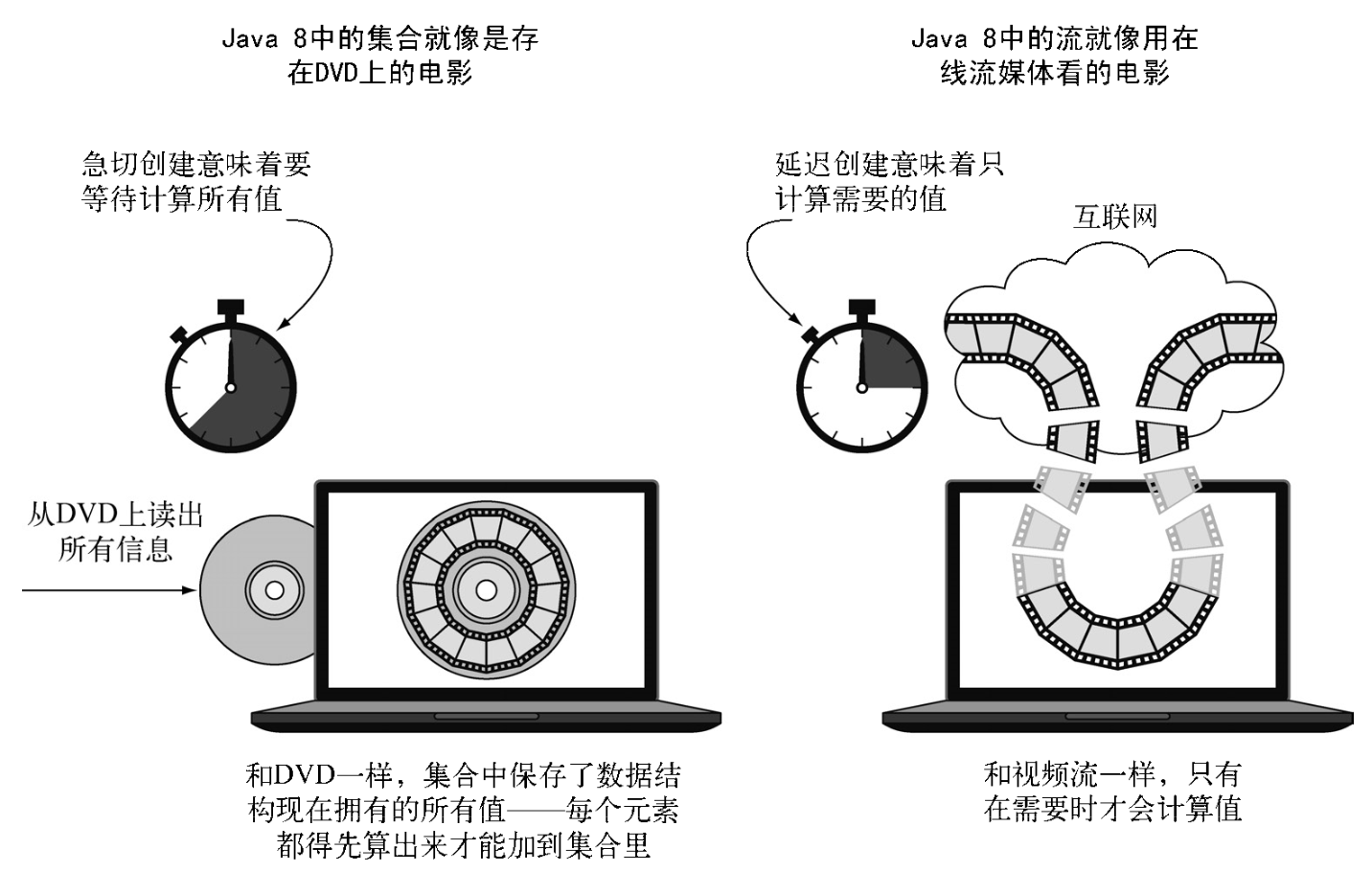
Java现有的集合概念和新的流概念都提供了接口，来配合代表元素型有序值的数据接口。所谓有序，就是说我们一般是按顺序取用值，而不是随机取用的。

比如说存在DVD里的电影，这就是一个集合，因为它包含了整个数据结构。现在再来想想在互联网上通过视频流看同样的电影。现在这是一个流。流媒体视频播放器只要提前下载用户观看位置的那几帧就可以了，这样不用等到流中大部分值计算出来，你就可以显示流的开始部分了(想想观看直播足球赛)。

流和集合之间的差异就在于什么时候进行计算。

1. 集合是一个内存中的数据结构，它包含数据结构中目前所有的值——集合中的每个元素都得先算出来才能添加到集合中；
2. 流则是在概念上固定的数据结构（你不能添加或删除元素），其元素则是按需计算的。



#### 1、只能遍历一次

注意：和迭代器类似，流只能遍历一次。遍历之后，我们就说这个流已经被消费掉了。你可以从原始数据源那里再获得一个新的流来重新遍历一遍。否则代码会抛出一个异常，说流已经被消费掉了。

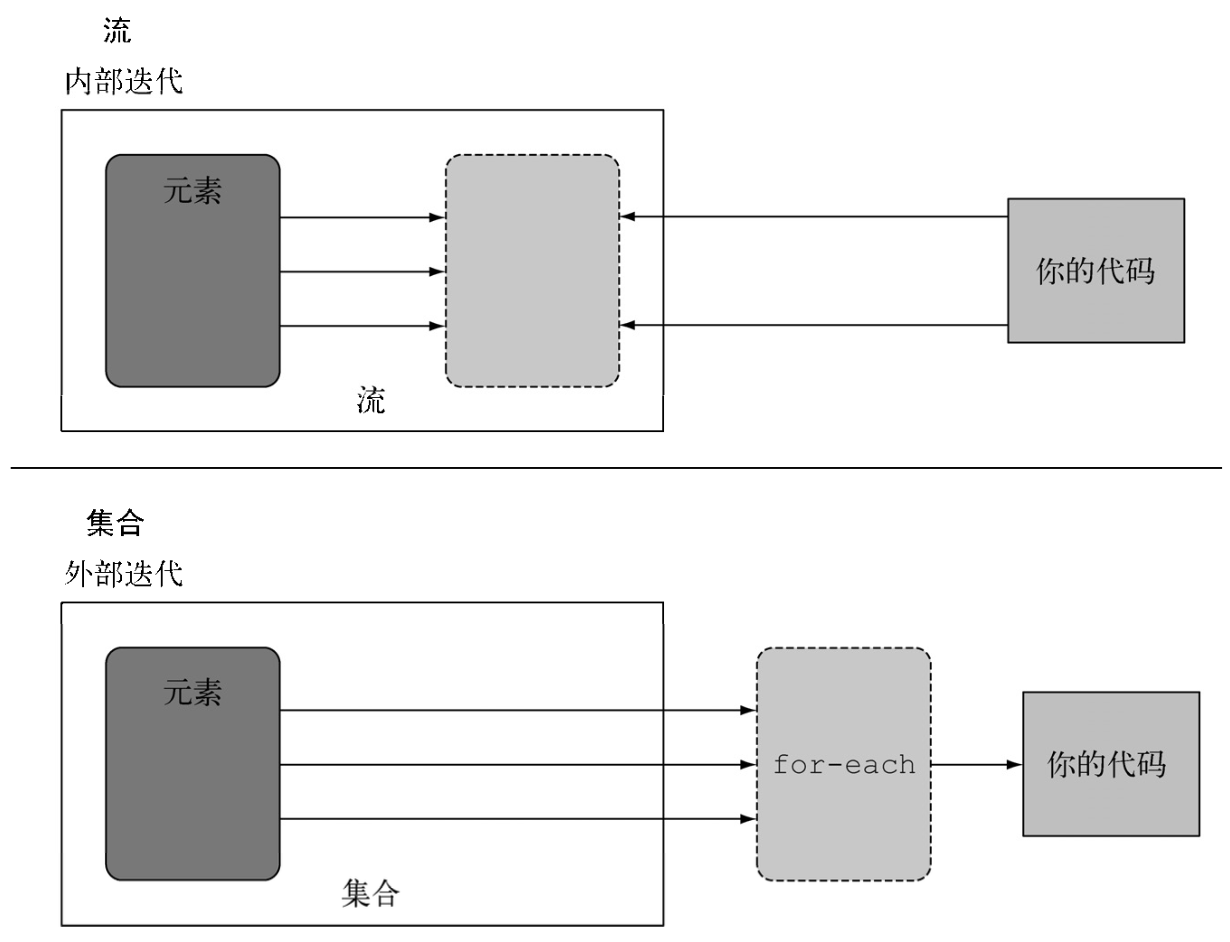
可以把流看作在时间中分布的一组值。相反，集合则是空间(这里就是计算机内存)中分布的一组值，在一个时间点上全体存在——你可以使用迭代器来访问 for-each循环中的内部成员。

#### 2、外部迭代和内部迭代

使用Collection接口需要用户去做迭代(比如用for-each)，这称为外部迭代。相反，Streams库使用内部迭代——它帮你把迭代做了，还把得到的流值存在了某个地方，你只要给出一个函数说要干什么就可以了。

1. 集合：用for-each循环外部迭代
2. 集合：用背后的迭代器做外部迭代
3. 流：内部迭代

内部迭代时，项目可以透明地并行处理，或者用更优化的顺序进行处理。要是用Java过去的那种外部迭代方法，这些优化都是很困难的。这似乎有点儿鸡蛋 里挑骨头，但这差不多就是Java 8引入流的理由了——Streams库的内部迭代可以自动选择一种适合你硬件的数据表示和并行实现。与此相反，一旦通过写for-each而选择了外部迭代，基本上就要自己管理所有的并行问题了。Java 8需要一个类似于Collection却没有迭代器的接口，于是就有了Stream！下图说明了流(内部迭代)与集合(外部迭代)之间的差异。



流利用了内部迭代：替你把迭代做了。但是，只有你已经预先定义好了能够隐藏迭代的操作列表，例如filter或map，这个才有用。大多数这类操作都接受Lambda表达式作为参数，因此可以参数化行为。Java语言的设计者给Stream API配上了一大套可以用来表达复杂数据处理查询的操作。