

（原理：相当于将数据源分离出来，使用统一抽象的类进行处理，这样每次处理结果返回该类型，又可以接着处理。这样不仅可以不影响数据源，又不需要考虑数据源的差异性，其实和传统的方法区别不大。但是好处在于抽象了一个级别出来，屏蔽了数据源差异，且有底层支持效率提升。）

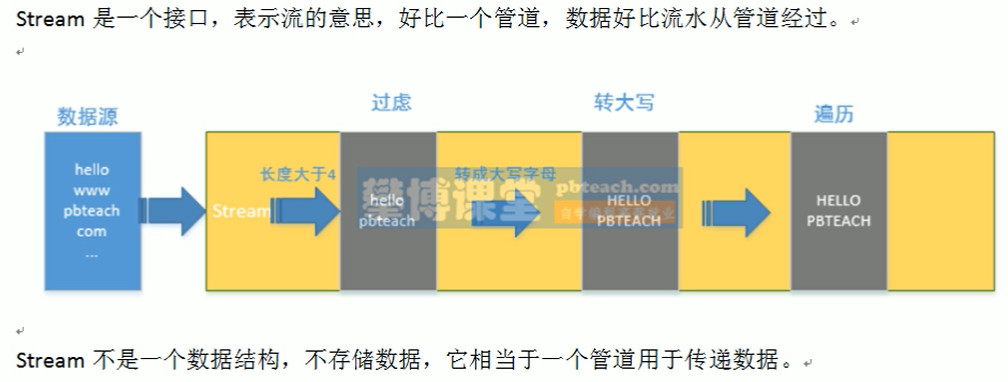
（区别解析：

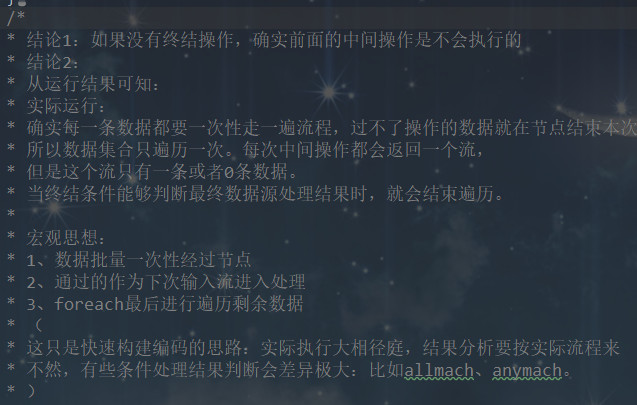
1、传统for是外部遍历，而Stream是内部遍历。也就是说for需要显式的取值放值遍历，而Stream这些都自己干了，也就是封装起来了。

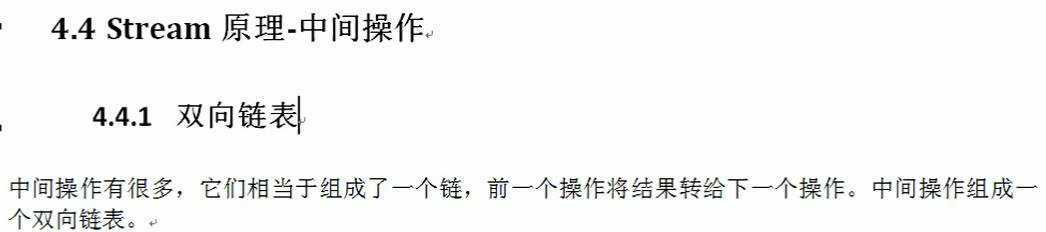
2、for循环实现类“怎么做”，而Stream实现“做什么”。（也就是说，前者是要根据需求，自己考虑具体实现方式，而后者根据需求只有唯一的实现方式，你指定需求，去调用就行。）这就将业务操作代码与遍历实现完全分离。

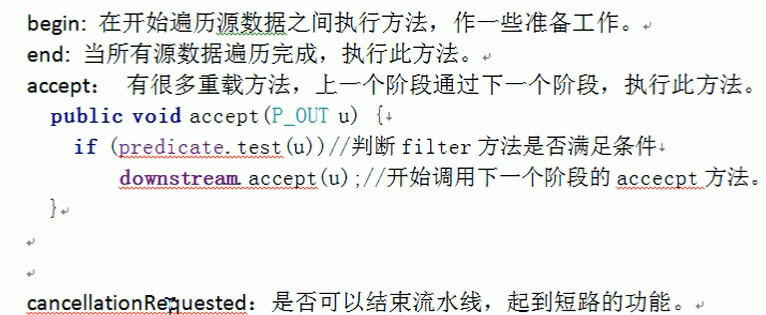
）

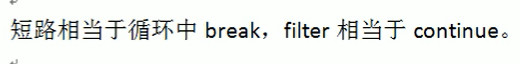
（Stream原理：

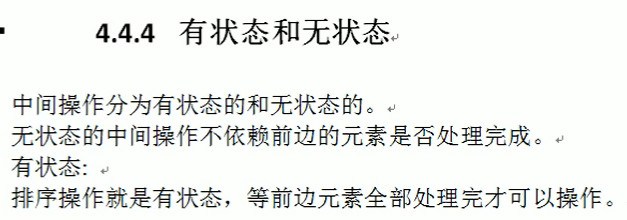












）

像 map 和filter这样的操作从输入流中得到每个元素, 并在输出流中产生零个或一个结果。因此, 这些操作通常是无状态的: 它们没有内部状态 (假设用户提供的 lambda 或方法引用没有内部可变状态)。

但是像reduce、sum和max 这样的操作需要有内部状态来累积结果。在这种情况下, 内部状态是小的。在我们的例子中, 它包括了一个 int 或double。无论正在处理的流中有多少元素, 内部状态都是有界大小的。

相比之下, 某些操作 (如sorted或distinct) 首先看起来像filter或map–所有这些都采用流并生成另一个流 (中间操作), 但有一个关键的区别。从流中排序和删除重复项都需要了解以前的历史记录才能完成其工作。例如, 排序要求在将单个项添加到输出流之前对所有元素进行缓冲。操作的存储要求是无限制的。如果数据流是大的或无限的, 这可能是问题。(什么应该逆转所有质数的流？它应该返回最大的质数, 数学告诉我们不存在。我们称这些操作为有状态的操作。

（所谓有状态操作就是：这类型操作需要先将开始的**数据源**中的数据遍历一遍，每条数据依次执行前面的操作，而到了该操作就需要停止，将数据积累起来，当数据积累完全后再执行该操作，形成新的**数据源**。这就意味着有状态操作很可能会导致遍历多次集合。）

（[深入理解Java Stream流水线 - CarpenterLee - 博客园 (cnblogs.com)](https://www.cnblogs.com/CarpenterLee/p/6637118.html)）

（[java8Stream原理深度解析 - Dorae - 博客园 (cnblogs.com)](https://www.cnblogs.com/Dorae/p/7779246.html)）



（那么也就是说，我们的集合类都能够获取到一个stream流，通过这个流的API来处理出我们需要的数据）

（流式编程-链式编程，将集合数据放入流中，每次通过一个个的过滤器进行过滤，过滤完成后形成新的流）（在这些流中间我们能做很多的中间层操作）

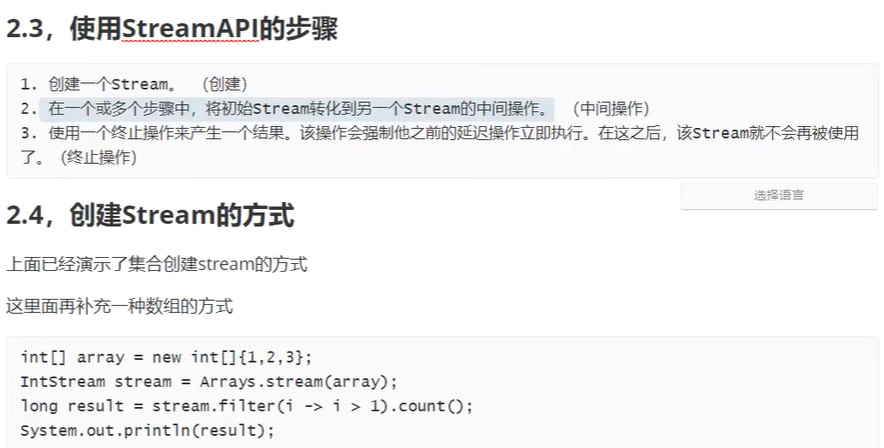
**Arrays类**：

该类包含用于操作数组的各种方法（例如排序和搜索）。 此类还包含一个静态工厂，允许将数组视为列表。

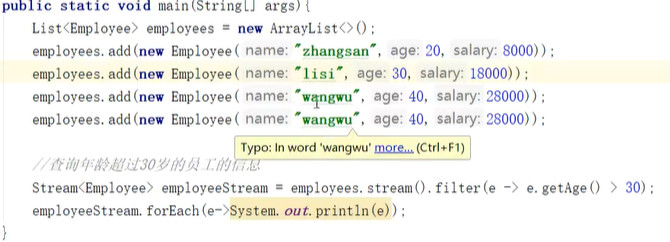
如果指定的数组引用为null，则此类中的方法都抛出NullPointerException ，除非另有说明。

包含在此类的方法中的文件包括的实施方式的简要描述。 这些描述应被视为实施说明 ，而不是规范的一部分 。 只要遵守规范本身，实现者就可以随意替换其他算法。 （例如， sort(Object[])使用的算法不一定是sort(Object[]) ，但它必须是稳定的 。）

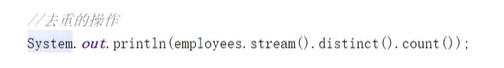
该班级是Java Collections Framework的成员。



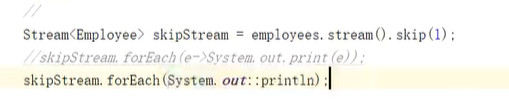
1、流式编程之一：filter（参数->过滤条件）（过滤器），从过滤后的流中获取指定数据。



2、去重(distinct)



3、跳过（skip跳过几条数据）



4、截取输出(limite)

