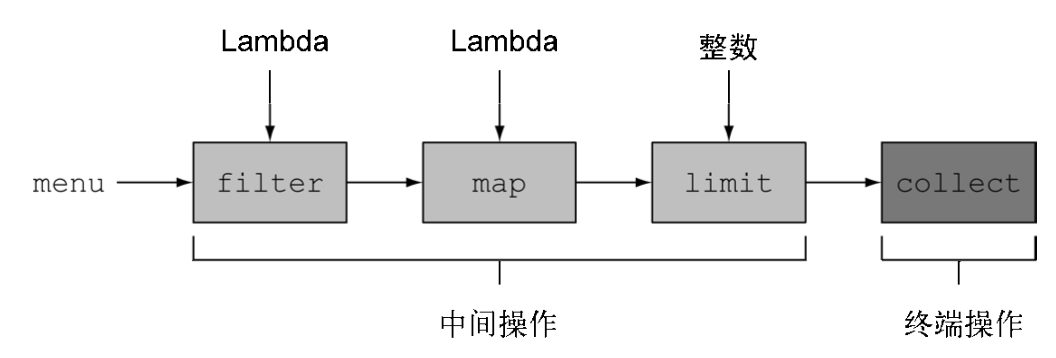
java.util.stream.Stream中的Stream接口定义了许多操作。它们可以分为两大类：

* 中间操作：filter、map和limit可以连成一条流水线；
* 终端操作：collect触发流水线执行并关闭它。



#### 1、中间操作

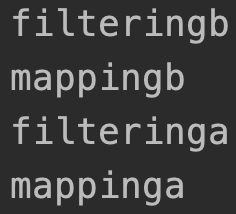
诸如filter或sorted等中间操作会返回另一个流。这让多个操作可以连接起来形成一个查询。除非流水线上触发一个终端操作，否则中间操作不会执行任何处理。

为了搞清楚流水线中到底发生了什么，我们把代码改一改，让每个Lambda都打印出当前处理的菜肴(就像很多演示和调试技巧一样，这种编程风格要是搁在生产代码里那就吓死人了，但是学习的时候却可以直接看清楚求值的顺序):

List<String> lowWeightPersonName =

persons.stream()  
 .filter(p -> {  
 System.*out*.println("filtering" + p.getName());  
 return p.getWeigth() < 70;  
 })

.map(p -> {  
 System.*out*.println("mapping" + p.getName());  
 return p.getName();  
 }) // 提取人员名称  
 .limit(2)  
 .collect(Collectors.*toList*());



有好几种优化利用了流的延迟性质。第一，尽管很多人的体重都低于70kg，但只选出了前三个！这是因为limit操作和一种称为短路的技巧。第二，尽管filter和map是两个独立的操作，但它们合并到同一次遍历中了（称为循环合并）。

#### 2、终端操作

终端操作会从流的流水线生成结果。其结果是任何不是流的值，比如List、Integer，甚至void。

#### 3、使用流

流的使用一般包括三件事：

1. 一个数据源（如集合）来执行一个查询；
2. 一个中间操作链，形成一条流的流水线；
3. 一个终端操作，执行流水线，并生成结果。

流的流水线背后的理念类似于构建器模式。在构建器模式中有一个调用链来设置一套配置（对流来说这就是一个中间操作链），接着是调用built方法（对流来说就是终端操作）。

常用中间操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 操作 | 类型 | 返回类型 | 操作参数 | 函数描述符 |
| filter | 中间 | Stream<T> | Predicate<T> | T -> boolean |
| map | 中间 | Stream<T> | Function<T, R> | T -> R |
| limit | 中间 | Stream<T> |  |  |
| sorted | 中间 | Stream<T> | Comparator<T> | (T, T) -> int |
| distinct | 中间 | Stream<T> |  |  |

常用终端操作

|  |  |  |
| --- | --- | --- |
| 操作 | 类型 | 目的 |
| forEach | 终端 | 消费流中的每个元素并对其应用Lambda。返回void |
| count | 终端 | 返回流中的元素个数。返回long |
| collect | 终端 | 把流规约成一个集合，比如List、Map等 |