要讨论流，先谈集合。Java 8中的集合支持一个新的stream方法，它会返回一个流（接口定义在java.util.stream.Stream里）。还有很多方法可以得到流，比如利用数值范围或从I/O资源生成流元素。

流：从支持数据操作的源生成的元素序列。

* 元素序列——就像集合一样，流也提供了一个接口，可以访问特定元素类型的一组有序值。因为集合是数据结构，所有它的主要目的是以特定的事件/空间复杂度存储和访问元素（如ArrayList与LinkedList）。但流的目的在于表达计算，比如filter、sorted和map。集合讲的是数据，流讲的是计算。
* 源——流会使用一个提供数据的源、如集合、数组或输入/输出资源。从有序集合生成流时会保留原有顺序。由列表生成的流，其元素顺序与列表一致。
* 数据处理操作——流的数据处理功能支持类似于数据库的操作，以及函数式编程语言中的常用操作，如filter、map、reduce、find、match、sorted等。流操作可以顺序执行，也可并行执行。

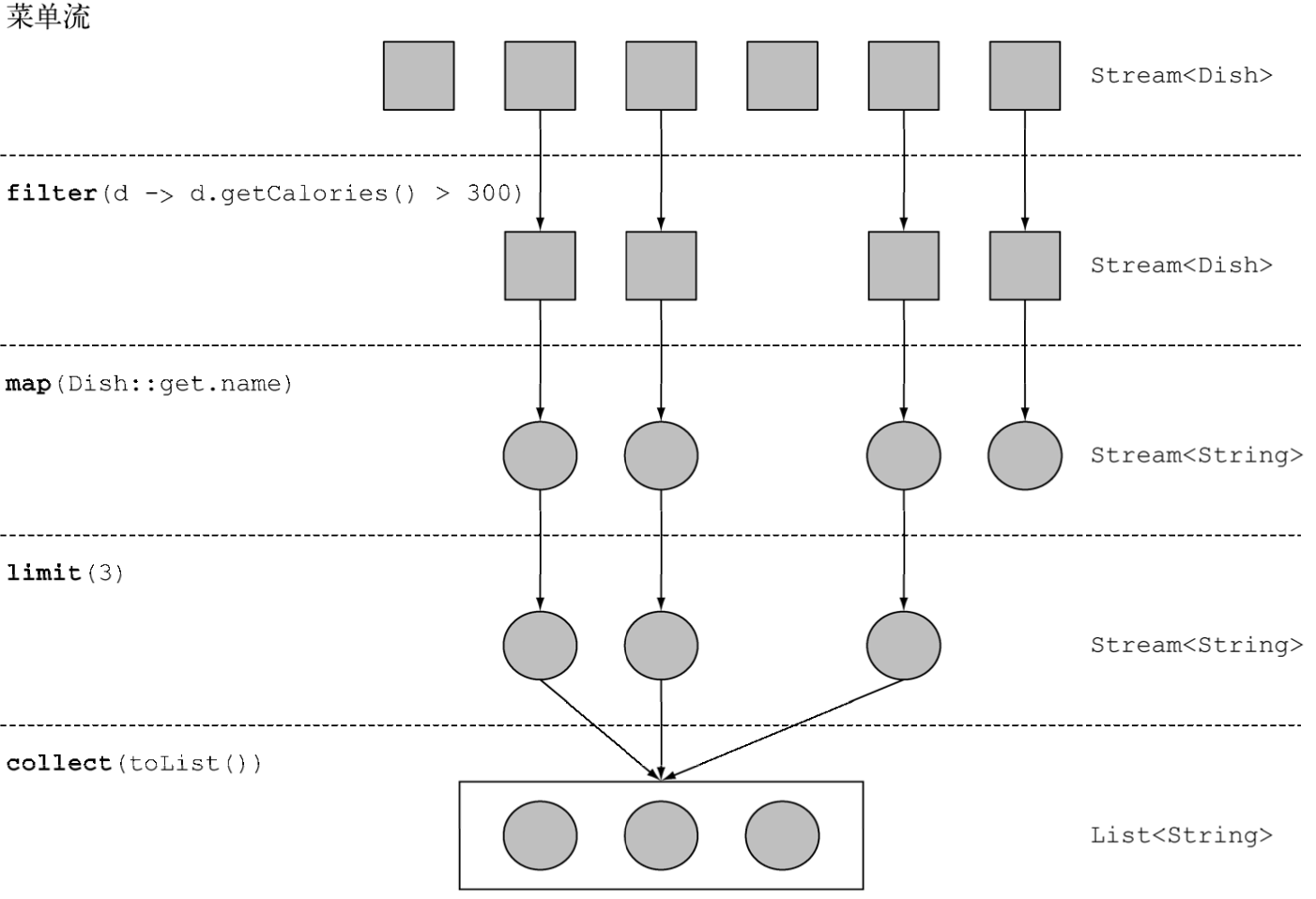
流操作特点：

* 流水线——很多流操作本身会返回一个流，这样多个操作就可以链接起来，形成一个大的流水线。这让一些优化（延迟和短路）成为可能。流水线的操作可以看作对数据源进行数据库式查询。
* 内部迭代——与使用迭代器显式迭代的集合不同，流的迭代操作是在背后进行的。

List<String> lowWeightPersonName =

persons.stream() // 从人员列表获得流  
 .filter(p -> p.getWeigth() < 70) // 建立操作流水线，首先选出低于70的人员   
 .map(Person::getName) // 获取人员姓名  
 .limit(2) // 只选头2个  
 .collect(Collectors.*toList*()); // 将结果保存在另个List

对流应用一系列数据处理操作：filter、map、limit 和collect。除了collect之外，所有这些操作都会返回另一个流，这样它们就可以接成一条流 水线，于是就可以看作对源的一个查询。最后，collect操作开始处理流水线，并返回结果(它和别的操作不一样，因为它返回的不是流，在这里是一个List)。在调用collect之前，没有任何结果产生，实际上根本就没有从persons里选择元素。你可以这么理解:链中的方法调用都在排 队等待，直到调用collect。下图显示了流操作的顺序:filter、map、limit、collect， 每个操作简介如下。



* filter——接受Lambda，从流中排除某些元素。在本例中，通过传递lambda p -> p.getWeight < 70，选择出体重低于70kg的人员。
* map——接受一个Lambda，将元素转换成其他形式或提取信息。在本例中，通过传递方法引用Person::getWeight，相当于Lambda p -> p.getName，提取了人员姓名。
* limit——截断流，使其元素不超过给定数量。
* collect——将流转换为其他形式。