集合是Java中使用最多的API。几乎每个Java应用程序都会制造和处理集合。集合对于很多编程任务来说都是非常基本的：它们可以让你把数据分组并加以处理。

* 很多业务逻辑都设计类似于数据库的操作，比如对几道菜按照类别进行分组（比如全素菜肴），或查找出最贵的菜。自己用迭代器重新实现过这些操作多少遍？大部分数据库都允许你声明式地指定这些操作。比如，以下SQL查询语句就可以选出热量较低的菜肴名称：SELECT name FROM dishes WHERE calorie < 400。你不需要实现如何根据菜肴的属性进行筛选(比如利用迭代器和累加器)，你只需要表达你想要什么。这个基本的思路意味着用不着担心怎么去显式地实现这些查询语句——都替你办好了! 怎么到了集合这里就不能这样了呢?
* 要使处理大量元素又该怎么办呢？为了提高性能，你需要并行处理，并利用多核架构。但写代码比用迭代器还要复杂，而且调试起来也够受的了。

那Java语言的设计者能做些什么，来帮助你节约宝贵的时间，让你这个程序员活得轻松一点儿呢？答案就是流。

流是Java API的新成员，它允许你以声明性方式处理数据集合（通过查询语句来表达，而不是临时编写一个实现）。就现在来说，可以把它看成遍历数据集的高级迭代器。此外，流还可以透明地并行处理，无需写任何多线程代码。

下面两段代码都是用来返回低热量的菜肴名称的，并按照卡路里排序，一个用Java 7写的，另一个用Java 8的流写的。

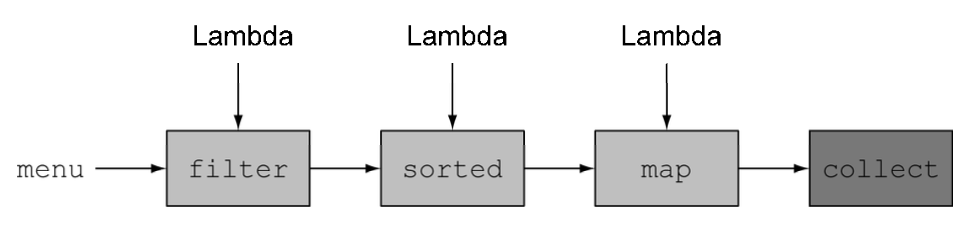
*/\*\**

*\* 使用Stream处理数据集合  
 \*/*public static void java8(List<Person> persons) {  
 List<String> lowWeightPersonName =  
 persons.stream()  
 .filter(p -> p.getWeigth() < 70) // 过滤器：选出体重低于70kg的人员  
 .sorted(Comparator.*comparing*(Person::getWeigth)) // 按照体重排序  
 .map(Person::getName) // 提取人员名称  
 .collect(Collectors.*toList*()); // 将所有名称保存在List中  
  
 System.*out*.println(lowWeightPersonName);  
}

为了利用多核架构并行执行这段代码，你只需要把stream()换成parallelStream()。在调用parallelStream()方法的时候到底发生了什么。用了多少个线程？对性能有多大提升？

* 代码是以声明性方式写的：说明想要完成什么（筛选体重轻的人员）而不是说明如何实现一个操作（利用循环和if条件等控制语句）。这种方法加上行为参数化让你可以轻松应对变化的需求：很容易再创建一个代码版本，利用Lambda表达式来筛选高卡路里的菜肴。
* 可以把几个基础操作链接起来，来表达复杂的数据处理流水线（在filter后面接上sorted、map和collect操作），filter的结果被传给sorted方法，再传给map方法，最后传给collect方法。

因为filter、sorted、map和collect等操作是与具体线程模型无关的高层次构件，所以 它们的内部实现可以是单线程的，也可能透明地充分利用你的多核架构！在实践中，用不着为了让某些数据处理任务并行而去操心线程和锁了，Stream API都替你做好了!



其他库：Guava、Apache和lambdaj

Guava是谷歌创建的一个库。它提供了multimaps和multisets等额外的容器类。Apache Commons Collections库也提供了类似的功能。Mario Fusco编写的lambdaj受到函数式编程的启发，也提供了很多声明性操作集合的工具。

如今Java 8自带了官方库，可以以更加声明性的方式操作集合了。

* 声明性——简洁易读
* 可复合——灵活
* 可并行——性能更好