编程语言中的函数一词通常是指方法，尤其是静态方法；这是在数学函数，也就是没有副作用的函数之外的新含义。Java 8谈到函数时，这两种用法几乎是一致的。

Java 8中新增了函数——值的一种新形式。它有助于使用流，有了它，Java 8可以进行多核处理器上的并行编程。

想想Java程序可能操作的值吧。首先有原始值，比如42（int类型）和3.14（double类型）。其次，值可以是对象（更严格地说是对象的引用）。获得对象的唯一途径是利用new，也许是通过工厂方法或库函数实现的；对象引用指向类的一个实例。

例子包括“abc”（String类型），new Integer(1111)（Integer类型），以及new HashMap<Integer,String>(100)的结果——它显然调用了HashMap的构造函数。甚至数组也是对象。那么有什么问题呢？

编程语言的整个目的就在于操作值，要是按照历史上编程语言的传统，这些值因此被称为一等值（或一等公民，这个术语是从20世纪60年代美国名犬运动中借用来的）。编程语言中的其他结构也许有助于我们表示值的结构，但在程序执行期间不能传递，因而是二等公民。前面所说的值是Java中的一等公民，但其他很多Java概念（如方法和类等）则是二等公民。用方法来定义类很不错，类还可以实例化来产生值，但方法和类本身都不是值。这又有什么关系呢？在运行时传递方法能将方法变成一等公民。这在编程中非常有用，因此Java 8的设计者把这个功能加入到了Java中。你可能会想，让类等其他二等公民也编程一等公民可能也是个好主意。有很多语言，如Smalltalk和JavaScript，都探讨过这条路。

#### 1、方法和Lambda作为一等公民

Java 8允许方法作为值，让编程更轻松。此外，让方法作为值也构成了其他若干Java 8功能（如Scream）的基础。

Java 8的第一个新功能是方法引用。比方说，你想要筛选一个目录中的所有隐藏文件。你需要编写一个方法，然后给它一个File，它就告诉你文件是不是隐藏的。幸好，File类里面有一个叫作isHidden的方法。我们可以把它看作一个函数，接受一个File，返回一个布尔值。但要用它做筛选，你需要把它包在一个FileFilter对象里，然后传递给File.listFiles方法，如下所示：

//返回一个抽象路径名数组，表示由此抽象路径名表示的满足指定过滤器的目录中的文件和目录

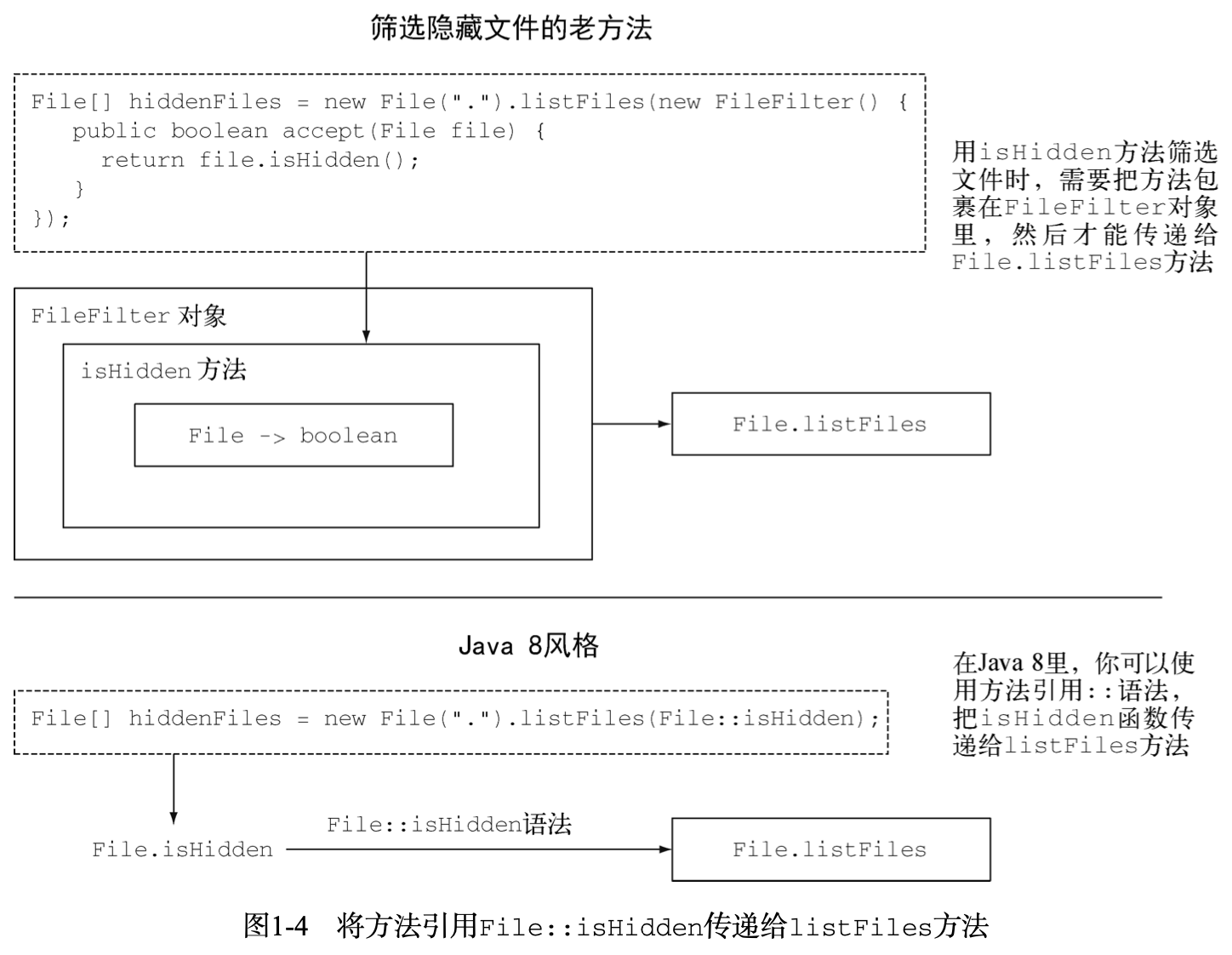
File[] hiddenFiles = new File(".").listFiles(new FileFilter() {  
 @Override  
 public boolean accept(File file) { //接受一个File，返回一个布尔值  
 return file.isHidden();  
 }  
});

在Java 8之前必须这么做。现在可以这样写：

//Java 8的实现方法

File[] files = new File(".").listFiles(File::isHidden);

已经有了函数isHidden，因此只需用Java 8的方法引用::语法（即“把这个方法作为值”）将其传给listFiles方法；请注意我们也开始用函数代表方法了。一个好处是，你的代码现在读起来更接近问题的陈述了。方法不再是二等值了。与用对象引用传递对象类似（对象引用是用new创建的），在Java 8里写下File::isHidden的时候，你就创建了一个方法引用，你同样可以传递它。只要方法中有代码（方法中的可执行部分），那么用方法引用就可以传递代码。



**Lambda——匿名函数**

除了允许（命名）函数成为一等值外，Java 8还体现了广义的将函数作为值的思想，包括Lambda（或匿名函数）。

比如，你现在写（int x） -> x + 1，表示“调用时给定参数x，就返回x + 1值的函数”。这有什么必要呢？因为你可以在MyMathsUtils类里面定义一个add1方法，然后写MyMathsUtils::add1。确实可以这样，但要是你没有方便的方法和类可用，新的Lambda语法更简洁。

#### 2、传递代码：一个例子

假设你有一个Apple类，它有一个getColor方法，还有一个变量inventory保存着一个Apples的列表。你可能想要选出所有的绿苹果，并返回一个列表。通常我们用筛选的（filter）一词来表达这个概念。在Java 8之前，你可能会写这样一个方法filterGreenApples：

*/\*\**

*\* 筛选出绿苹果  
 \*  
 \** ***@param*** *inventory  
 \** ***@return*** *\*/*public static List<Apple> filterGreenApples(List<Apple> inventory) {  
 //result用来累积结果的list  
 List<Apple> result = new ArrayList<>();  
 for (Apple apple : inventory) {  
 if ("green".equals(apple.getColor())) {  
 result.add(apple);  
 }  
 }  
 return result;  
}  
  
*/\*\*  
 \* 筛选出重量超过150g的苹果  
 \*  
 \** ***@param*** *inventory  
 \** ***@return*** *\*/*public static List<Apple> filterHeavyApples(List<Apple> inventory) {  
 List<Apple> result = new ArrayList<>();  
 for (Apple apple : inventory) {  
 if (apple.getWeight() > 150) {  
 result.add(apple);  
 }  
 }  
 return result;  
}

以上两个方法只有一行不同：if里面高亮的哪行条件。如果这两个高亮的方法之间的差异仅仅是接受的重量范围不同，那么你只要把接受重量上下限作为参数传递给filter就行了，比如指定(150,1000)来选出重的苹果（超过150克），或者指定（0，80）来选出轻的苹果（低于80克）。

但是我们前面提过了，Java 8会把条件代码作为参数传递进去，这样可以避免filter方法出现重复的代码。

public static boolean isGreenApple(Apple apple) {

return "green".equals(apple.getColor());  
}  
  
public static boolean isHeavyApple(Apple apple) {  
 return apple.getWeight() > 150;  
}

*/\*\*  
 \* 筛选苹果  
 \*  
 \** ***@param*** *inventory  
 \** ***@param*** *p  
 \** ***@return*** *\*/*public static List<Apple> filterApples(List<Apple> inventory, Predicate<Apple> p) {  
 List<Apple> result = new ArrayList<>();  
 for (Apple apple : inventory) {  
 if (p.test(apple)) {  
 result.add(apple);  
 }  
 }  
 return result;  
}

用它的话，你可以写：

List<Apple> greenApples = *filterApples*(inventory, FilteringApples::*isGreenApple*);

List<Apple> heavyApples = *filterApples*(inventory, FilteringApples::*isHeavyApple*);

现在重要的是你可以在Java 8里面传递方法了！

什么是谓词？

前面的代码传递了方法Apple::isGreenApple（它接受参数Apple并返回一个boolean）给filterApples，后者则希望接受一个Predicate<Apple>参数。谓词（predicate）在数学上常常用来代表一个类似函数的东西，它接受一个参数值，并返回true或false。你在后面会看到，Java 8也会允许你写Function<Apple,Boolean>——在学校学过函数却没学过谓词的读者对此可能更熟悉，但用Predicate<Apple>是更标准的方式，效率也会更高一点儿，这避免了把boolean封装在Boolean里面。

#### 3、从传递方法到Lambda

把方法作为值来传递显然很有用，但要是为类似于isHeavyApple和isGrrenApple这种可能只用一两次的短方法写一堆定义有点儿烦人。不过Java 8也解决了这个问题，它引入了一套新记法（匿名函数或Lambda）,让你可以写:

List<Apple> greenApples2 = *filterApples*(inventory, (Apple a) -> "green".equals(a.getColor()));

List<Apple> heavyApples2 = *filterApples*(inventory, (Apple a) -> a.getWeight() > 150);

List<Apple> weirdApples = *filterApples*(inventory, (Apple a) -> a.getWeight() < 80 || "brown".equals(a.getColor()));

所以,甚至都不需要为只用一次的方法写定义；代码更干净、更清晰，因为你用不着去找自己到底传递了什么代码。但要是Lambda的长度多于几行（它的行为也不是一目了然）的话，那你还是应该用方法引用来指向一个有描述性名称的方法，而不是使用匿名的Lambda。你应该以代码的清晰度为准绳。

本来Java加上filter和几个相关的东西作为通用库方法就足以让人满意了，比如：

static <T> Collection<T> filter(Collection<T> c, Predicate<T> p);

这样你甚至都不需要写filterApples了，因为比如先前的调用：

*filterApples*(inventory, (Apple a) -> a.getWeight() > 150);

就可以直接调库方法filter：

*filter*(inventory, (Apple a) -> a.getWeight() > 150);

但是，为了更好地利用并行，Java的设计师没有这么做。Java 8中有一整套新的类集合API——Stream，它有一套函数式程序员熟悉的、类似于filter的操作，比如map、reduce，还有我们接下来要讨论的在Collections和Stream之间做转换的方法。