《Java8实战》笔记

[英]Raoul-Gabriel Urma

[意]Mario Fusco 著

[英]Alan Mycroft

陆明刚 劳佳 译

人民邮电出版社

2017-08-02：开始~Page4

2017-08-04：Page5~Page7

2017-08-09：Page8~Page9

2017-08-10：Page10~Page16

2017-08-11：Page17~Page17

2017-08-14：Page18~Page

# Part1 基础知识

## 为什么要关心Java8

|  |
| --- |
| **本章内容**   * Java怎么又变了 * 日新月异的计算应用背景：多核和处理大型数据集（大数据） * 改进的压力：函数式比命令式更适应新的体系架构 * Java8的核心新特性：Lambda（匿名函数）、流、默认方法 |

## Java怎么还在变

### Java在编程语言生态系统中的位置

### 1.1.2流处理

### 1.1.3 用行为参数化把代码传递给方法

### 1.1.4 并行与共享的可变数据

### 1.1.5 Java需要演变

### Java中的函数

### 1.2.1 方法和Lambda作为一等公民

筛选一个目录中的所有隐藏文件。

Java8前：

File[] hiddenFiles = new File(“.”).listFiles(new FileFilter(){

public Boolean accept(File file){

return file.isHidden();

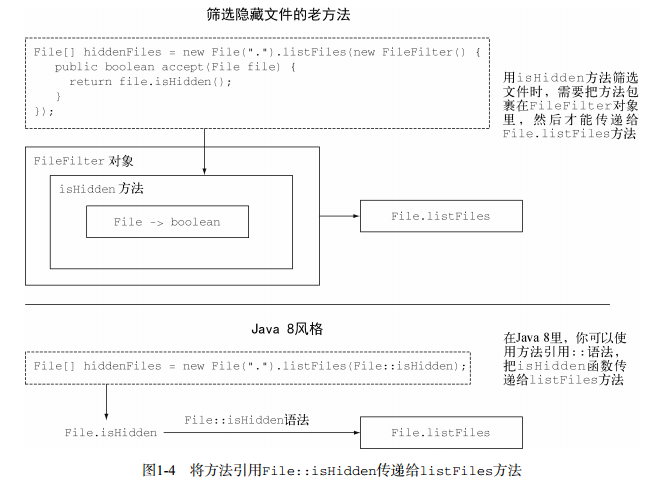
}

});

Java8：

File[] hiddenFiles = new File(“.”).listFiles(File::isHidden);

与用对象引用传递对象（对象引用是用new创建的），在Java8里写下File::isHidden的时候，就创建了一个方法引用，同样可以传递它。



### 1.2.2 传递代码：一个例子

所有的示例代码均可见于本书的GitHub页面（<https://github.com/java8/>）。

筛选绿苹果

Java8之前：

public static List<Apple> filterGreenApples(List<Apple> inventory){

List<Apple> result = new ArrayList<>();

for(Apple apple : inventory){

**if(“green”.equals(apple.getColor())){//筛选绿苹果**

result.add(apple);

}

}

return result;

}

筛选重量超过150克的苹果

Java8之前：

public static List<Apple> filterHeavyApples(List<Apple> inventory){

List<Apple> result = new ArrayList<>();

for(Apple apple : inventory){

**if(apple.getWeight() > 150){//筛选重量超过150克的苹果**

result.add(apple);

}

}

return result;

}

复制粘贴代码的危害：需要更新时容易遗漏。

Java8：

//该方法应放在Apple.java中

public static Boolean isGreenApple(Apple apple){

return “green”.equals(apple.getColor());

}

//该方法应放在Apple.java中

public static Boolean isHeavyApple(Apple apple){

return apple.getWeight() > 150;

}

//PS：实践发现，接口中只能包含一个方法

public interface Predicate<T>{

boolean test(T t);

}

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;

}

filterApples(inventory, **Apple::isGreenApple**); **//筛选绿苹果**

filterApples(inventory, **Apple::isHeavyApple**); **//筛选重量超过150克的苹果**

### 1.2.3 从传递方法到Lambda

filterApples(inventory, (**Apple a**) **-> ”green”.equals(a.getColor())**);

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

filterApples(inventory, (**Apple a**) **-> a.getWeight < 80 || “brown”.equals(a.getColor())**);

Lambda的长度多于几行时，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。

## 流

从一个列表中筛选金额较高的交易，然后按货币分组。

Java8之前：

//外部迭代

Map<Currency, List<Transaction>> transactionsByCurrencies = new HashMap<>();

for(Transaction transaction : transactions){

if(transaction.getPrice() > 1000){

Currency currency = transaction.getCurrency();

List<Transaction> transactionsForCurrency = transactionsByCurrencies.get(currency);

if(transactionsForCurrency == null){

transactionsForCurrency = new ArrayList<>();

transactionsByCurrencies.put(currency, transactionsForCurrency);

}

transactionsForCurrency.add(transaction);

}

}

Java8：

//内部迭代

import static java.util.stream.Collectors.toList;

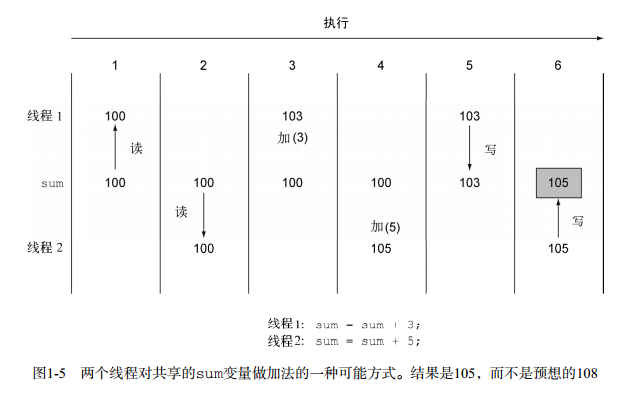
Map<Currency, List<Transaction>> transactionsByCurrencies =

transactions.stream()

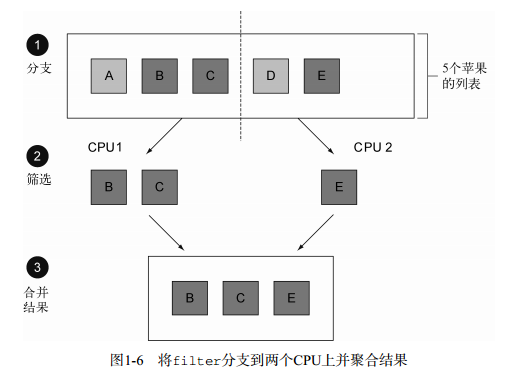
.filter((Transaction t) -> t.getPrice() > 1000)

.collect(groupingBy(Transaction :: getCurrency));

### 多线程并非易事



**Java8利用Stream API（java.util.stream）解决了：集合处理时的套路和晦涩，以及难以利用多核。**



**Stream允许并提倡并行处理一个Stream中的元素。**

利用Stream和Lambda表达式顺序或并行地从一个列表里筛选比较重的苹果。

顺序处理：

import static java.util.stream.Collectors.toList;

List<Apple> heavyApples =

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

.collect(toList());

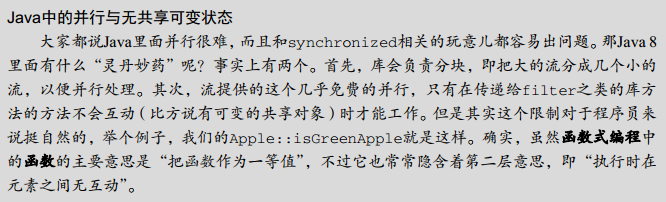
并行处理：

import static java.util.stream.Collectors.toList;

List<Apple> heavyApples =

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

.collect(toList());



## 默认方法

默认方法解决了什么问题？

当JDK中的接口新增方法时，已应用于项目的实体类必须为其提供一个实现，而在Java8中，接口如今可以包含实现类没有提供实现的方法签名，缺失的方法主体随接口提供，而不是由实现类提供，这个就是默认方法。

例如，在Java8里，可以直接对List调用sort方法。它是用Java8 List接口中如下所示的默认方法实现的，它会调用Collections.sort静态方法：

default void sort(Comparator<? super E> c){

Collections.sort(this, c);

}

一个类可以实现多个接口，如果在好几个接口里有多个默认实现，就意味着Java中有了某种形式的多重继承。

Java8如何避免类似于C++的菱形继承问题。

## 来自函数式编程的其他好思想