Part 1 第一部分

基础知识

本书第一部分将介绍Java 8的基础知识。学完第一部分， 你将会对Lambda表达式有充分的了解，并可以编写简洁而灵活的代码，能够轻松地适应不断变化的需求。

第1章将总结Java的主要变化（Lambda表达式、方法引用、流和默认方法），并为学习本书做好准备。

在第2章中，你将了解行为参数化，这是Java 8非常依赖的一种软件开发模式，也是引入Lambda表达式的主要原因。

第3章全面地解释了Lambda表达式和方法引用的概念，每一步都有代码示例和测验。

第 1 章 为什么要关心Java 8

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
- Java 8的核心新特性：Lambda（匿名函数）、流、默认方法

自1998年JDK 1.0（Java 1.0）发布以来，Java已经受到了学生、项目经理和程序员等一大批活跃用户的欢迎。这一语言极富活力，不断被用在大大小小的项目里。从Java 1.1（1997年）一直到Java 7（2011年），Java通过增加新功能，不断得到良好的升级。Java 8则是在2014年3月发布的。那么，问题来了：为什么你应该关心Java 8？

我们的理由是，Java 8所做的改变，在许多方面比Java历史上任何一次改变都深远。而且好消息是，这些改变会让你编起程来更容易，用不着再写类似下面这种啰嗦的程序了（对inventory中的苹果按照重量进行排序）：

```
Collections.sort(inventory, new Comparator<Apple>() {
    public int compare(Apple a1, Apple a2){
        return a1.getWeight().compareTo(a2.getWeight());
    }
});
```

在Java 8里面，你可以编写更为简洁的代码，这些代码读起来更接近问题的描述：

```
inventory.sort(comparing(Apple::getWeight));
```

本书中第一段Java 8的代码！

它念起来就是“给库存排序，比较苹果的重量”。现在你不用太关注这段代码，本书后面的章节将会介绍它是做什么用的，以及你如何写出类似的代码。

Java 8对硬件也有影响：平常我们用的CPU都是多核的——你的笔记本电脑或台式机上的处理器可能有四个CPU内核，甚至更多。但是，绝大多数现有的Java程序都只使用其中一个内核，其他三个都闲着，或只是用一小部分的处理能力来运行操作系统或杀毒程序。

在Java 8之前，专家们可能会告诉你，必须利用线程才能使用多个内核。问题是，线程用起来很难，也容易出现错误。从Java的演变路径来看，它一直致力于让并发编程更容易、出错更少。

Java 1.0里有线程和锁，甚至有一个内存模型——这是当时的最佳做法，但事实证明，不具备专门知识的项目团队很难可靠地使用这些基本模型。Java 5添加了工业级的构建模块，如线程池和并发集合。Java 7添加了分支/合并（fork/join）框架，使得并行变得更实用，但仍然很困难。而Java 8对并行有了一个更简单的新思路，不过你仍要遵循一些规则，本书中会谈到。

我们用两个例子（它们有更简洁的代码，且更简单地使用了多核处理器）就可以管中窥豹，看到一座拔地而起相互勾连一致的Java 8大厦。首先让你快速了解一下这些想法（希望能引起你的兴趣，也希望我们总结得足够简洁）：

- Stream API
- 向方法传递代码的技巧
- 接口中的默认方法

Java 8提供了一个新的API（称为“流”，Stream），它支持许多处理数据的并行操作，其思路和在数据库查询语言中的思路类似——用更高级的方式表达想要的东西，而由“实现”（在这里是Streams库）来选择最佳低级执行机制。这样就可以避免用`synchronized`编写代码，这一代码不仅容易出错，而且在多核CPU上执行所需的成本也比你想象的要高。[①]

从有点修正主义的角度来看，在Java 8中加入`Streams`可以看作把另外两项扩充加入Java 8的直接原因：*把代码传递给方法的简洁方式*（方法引用、Lambda）和接口中的*默认方法*。

如果仅仅“把代码传递给方法”看作`Streams`的一个结果，那就低估了它在Java 8中的应用范围。它提供了一种新的方式，这种方式简洁地表达了*行为参数化*。比方说，你想要写两个只有几行代码不同的方法，那现在你只需要把不同的那部分代码作为参数传递进去就可以了。采用这种编程技巧，代码会更短、更清晰，也比常用的复制粘贴更不容易出错。高手看到这里就会想，在Java 8之前可以用匿名类实现行为参数化呀——但是想想本章开头那个Java 8代码更加简洁的例子，代码本身就说明了它有多清晰！

Java 8里面将代码传递给方法的功能（同时也能够返回代码并将其包含在数据结构中）还让我们能够使用一整套新技巧，通常称为*函数式编程*。一言以蔽之，这种被函数式编程界称为*函数*的代码，可以被来回传递并加以组合，以产生强大的编程语汇。这样的例子在本书中随处可见。

本章主要从宏观角度探讨了语言为什么会演变，接下来几节介绍Java 8的核心特性，然后介绍函数式编程思想——其新的特性简化了使用，而且更适应新的计算机体系结构。简而言之，1.1节讨论了Java的演变过程和概念，指出Java以前缺乏以简易方式利用多核并行的能力。1.2节介绍了为什么把代码传递给方法在Java 8里是如此强大的一个新的编程语汇。1.3节对`Streams`做了同样的介绍：`Streams`是Java 8表示有序数据，并能灵活地表示这些数据是否可以并行处理的新方式。1.4节解释了如何利用Java 8中的默认方法功能让接口和库的演变更顺畅、编译更少。最后，1.5节展望了在Java和其他共用JVM的语言中进行函数式编程的思想。总的来说，本章会介绍整体脉络，而细节会在本书的其余部分中逐一展开。请尽情享受吧！

---

① 多核CPU的每个处理器内核都有独立的高速缓存。加锁需要这些高速缓存同步运行，然而这又需要在内核间进行较慢的缓存一致性协议通信。

1.1　Java 怎么还在变

20世纪60年代，人们开始追求完美的编程语言。当时著名的计算机科学家彼得·兰丁（Peter Landin）在1966年的一篇标志性论文[①]中写道，当时已经有700种编程语言了，并推测了接下来的700种会是什么样子，文中也对类似于Java 8中的函数式编程进行了讨论。

之后，又出现了数以千计的编程语言。学者们得出结论，编程语言就像生态系统一样，新的语言会出现，旧语言则被取代，除非它们不断演变。我们都希望出现一种完美的通用语言，可在现实中，某些语言只是更适合某些方面。比如，C和C++仍然是构建操作系统和各种嵌入式系统的流行工具，因为它们编出的程序尽管安全性不佳，但运行时占用资源少。缺乏安全性可能导致程序意外崩溃，并把安全漏洞暴露给病毒和其他东西；确实，Java和C#等安全型语言在诸多运行资源不太紧张的应用中已经取代了C和C++。

先抢占市场往往能够吓退竞争对手。为了一个功能而改用新的语言和工具链往往太过痛苦了，但新来者最终会取代现有的语言，除非后者演变得够快，能跟上节奏。年纪大一点的读者大多可以举出一堆这样的语言——他们以前用过，但是现在这些语言已经不时髦了。随便列举几个吧：Ada、Algol、COBOL、Pascal、Delphi、SNOBOL等。

你是一位Java程序员。在过去15年的时间里，Java已经成功地霸占了编程生态系统中的一大块，同时替代了竞争对手语言。让我们来看看其中的原因。

1.1.1　Java 在编程语言生态系统中的位置

Java天资不错。从一开始，它就是一个精心设计的面向对象的语言，有许多有用的库。有了集成的线程和锁的支持，它从第一天起就支持小规模并发（并且它十分有先知之明地承认，在与硬件无关的内存模型里，多核处理器上的并发线程可能比在单核处理器上出现的意外行为更多）。此外，将Java编译成JVM字节码（一种很快就被每一种浏览器支持的虚拟机代码）意味着它成为了互联网applet（小应用）的首选（你还记得applet吗?）。确实，Java虚拟机（JVM）及其字节码可能会变得比Java语言本身更重要，而且对于某些应用来说，Java可能会被同样运行在JVM上的竞争对手语言（如Scala或Groovy）取代。JVM各种最新的更新（例如JDK7中的新`invokedynamic`字节码）旨在帮助这些竞争对手语言在JVM上顺利运行，并与Java交互操作。Java也已成功地占领了嵌入式计算的若干领域，从智能卡、烤面包机、机顶盒到汽车制动系统。

Java是怎么进入通用编程市场的?

面向对象在20世纪90年代开始时兴的原因有两个：封装原则使得其软件工程问题比C少；作为一个思维模型，它轻松地反映了Windows 95及之后的WIMP编程模式。可以这样总结：一切都是对象；单击鼠标就能给处理程序发送一个事件消息（在`Mouse`对象中触发`Clicked`方

① P. J. Landin, "The Next 700 Programming Languages," *CACM* 9(3):157–65, March 1966.

法）。Java的“一次编写，随处运行”模式，以及早期浏览器安全地执行Java小应用的能力让它占领了大学市场，毕业生随后把它带进了业界。开始时由于运行成本比C/C++要高，Java还遇到了一些阻力，但后来机器变得越来越快，程序员的时间也变得越来越重要了。微软的C#进一步验证了Java的面向对象模型。

但是，编程语言生态系统的气候正在变化。程序员越来越多地要处理所谓的*大数据*（数百万兆甚至更多字节的数据集），并希望利用多核计算机或计算集群来有效地处理。这意味着需要使用并行处理——Java以前对此并不支持。

你可能接触过其他编程领域的思想，比如Google的map-reduce，或如SQL等数据库查询语言的便捷数据操作，它们能帮助你处理大数据量和多核CPU。图1-1总结了语言生态系统：把这幅图看作编程问题空间，每个特定地方生长的主要植物就是程序最喜欢的语言。气候变化的意思是，新的硬件或新的编程因素（例如，“我为什么不能用SQL的风格来写程序？”）意味着新项目优选的语言各有不同，就像地区气温上升就意味着葡萄在较高的纬度也能长得好。当然这会有滞后——很多老农一直在种植传统作物。总之，新的语言不断出现，并因为迅速适应了气候变化，越来越受欢迎。

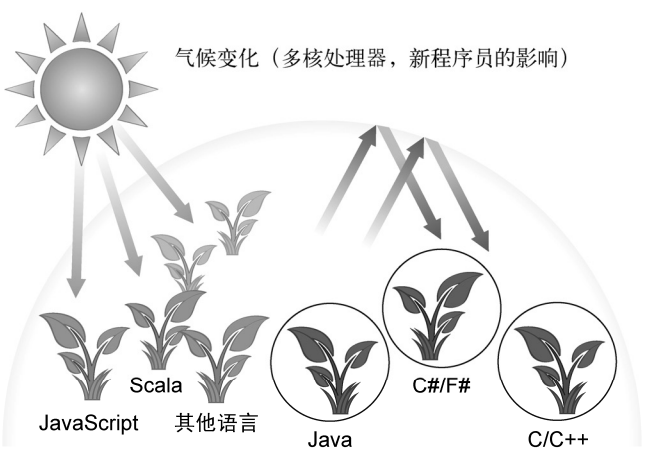

图1-1 编程语言生态系统和气候变化

Java 8对于程序员的主要好处在于它提供了更多的编程工具和概念，能以更快，更重要的是能以更为简洁、更易于维护的方式解决新的或现有的编程问题。虽然这些概念对于Java来说是新的，但是研究型的语言已经证明了它们的强大。我们会突出并探讨三个这样的编程概念背后的思想，它们促使Java 8中开发出并行和编写更简洁通用代码的功能。我们这里介绍它们的顺序和本书其余的部分略有不同，一方面是为了类比Unix，另一方面是为了揭示Java 8新的多核并行中存在的“因为这个所以需要那个”的依赖关系。