The Pragmatic Programmers

TURING 图灵程序设计丛书 Java系列

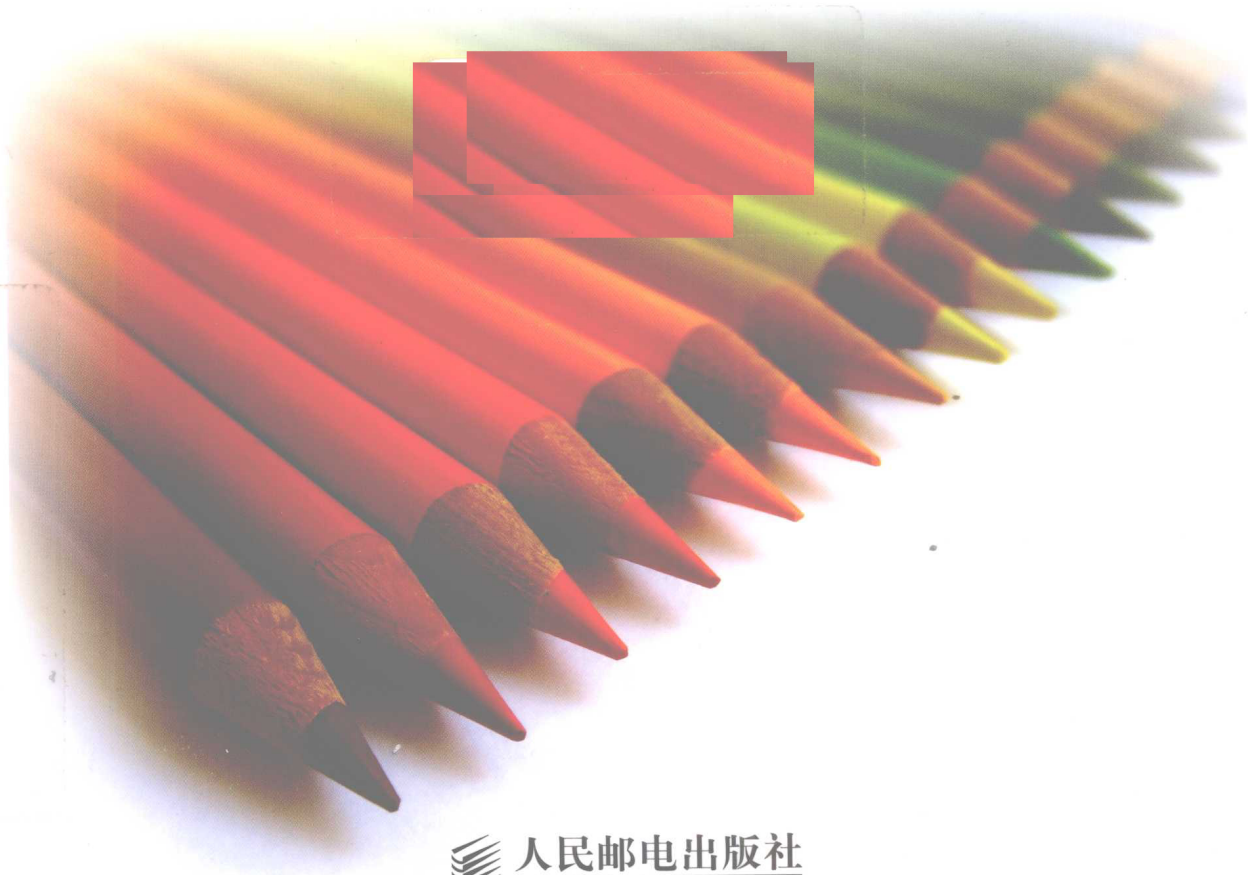
Programming Scala

Tackle Multi-Core Complexity on the Java Virtual Machine

Scala程序设计

Java虚拟机多核编程实战

[美] Venkat Subramaniam 著
郑晔 李剑 译



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 Java系列

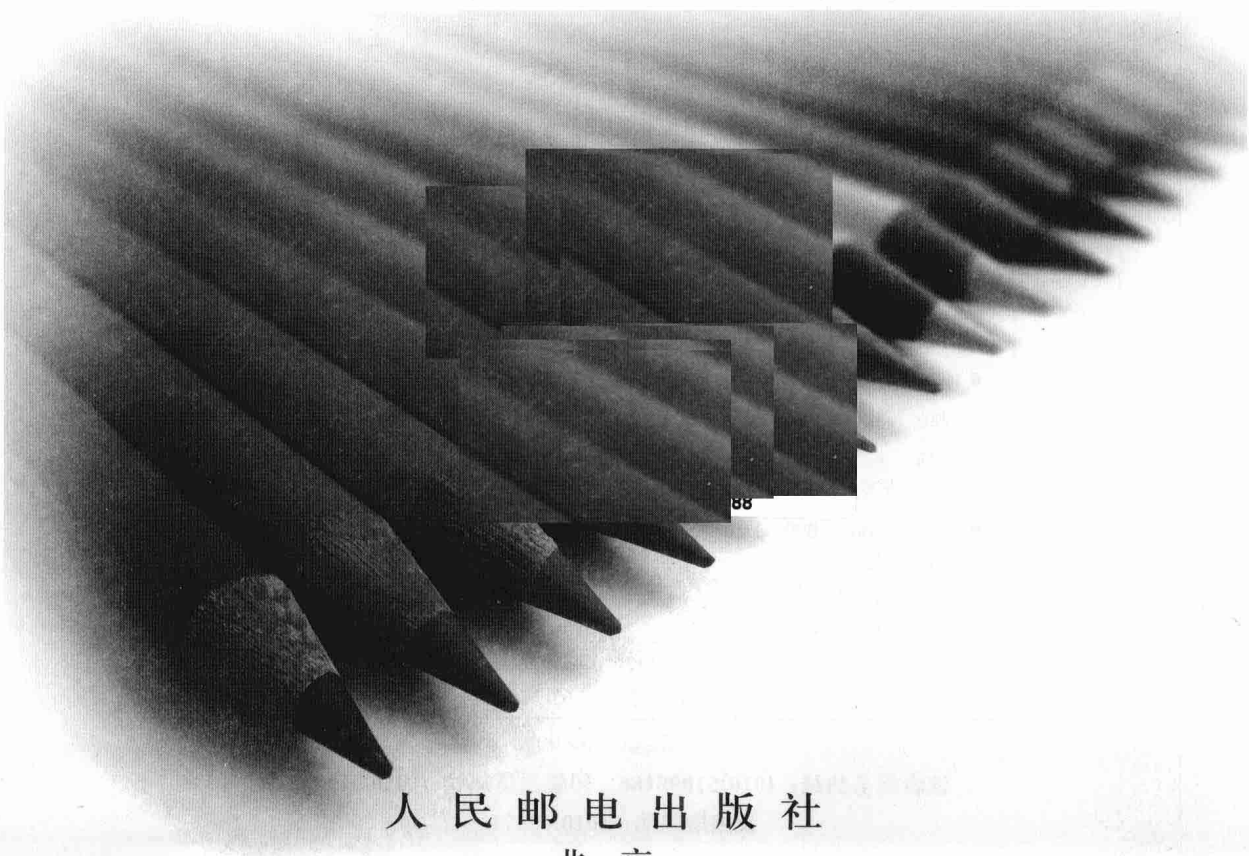
Programming Scala

Tackle Multi-Core Complexity on the Java Virtual Machine

Scala程序设计

Java虚拟机多核编程实战

[美] Venkat Subramaniam 著
郑晔 李剑 译



人民邮电出版社
北京

图书在版编目（CIP）数据

Scala程序设计 : Java虚拟机多核编程实战 / (美)苏帕拉马尼亚姆 (Subramaniam,V.)著 ; 郑晔, 李剑译. — 北京 : 人民邮电出版社, 2010.8
(图灵程序设计丛书)
书名原文: Programming Scala: Tackle Multi-Core Complexity on the Java Virtual Machine
ISBN 978-7-115-23295-3

Ⅰ. ①S… Ⅱ. ①苏… ②郑… ③李… Ⅲ. ①JAVA语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2010)第122110号

内 容 提 要

Scala是一门混合了函数式和面向对象的静态类型语言。本书旨在使读者在Scala上达到一定水平，可以用它编写并发、可伸缩、有表现力的程序。主要涉及Scala的函数式风格、自适应类型、闭包、XML处理、模式匹配和并发编程等内容。通过学习本书，你可以使用Scala的强大能力，创建多线程的应用程序。

这本书是为想了解Scala的程序员和有经验的Java程序员准备的。本书可以帮助读者快速领会Scala的精髓，用它构建真实的应用。

图灵程序设计丛书

Scala程序设计：Java虚拟机多核编程实战

◆ 著 [美] Venkat Subramaniam
译 郑 晔 李 剑
责任编辑 傅志红
执行编辑 丁晓昀

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：12
字数：263千字 2010年8月第1版
印数：1－3 000册 2010年8月北京第1次印刷

著作权合同登记号 图字：01-2010-4215号

ISBN 978-7-115-23295-3

定价：39.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

读者推荐

这本书直面那些面临并发困境的开发人员，为在JVM上搭建actor提供了清晰的解决方案。

——John Heintz，Gist Labs总监

Venkat以一种易于追随且讲求实效的方式为（Java）程序员介绍了Scala编程。这本书涵盖了Scala的很多方面，从基础概念直到并发，而后者是如今编程面临的最关键最困难的问题。Venkat轻而易举地触及了问题的核心，我强烈推荐这本书，它能让你快速上手Scala。

——Scott Leberknight，Near Infinity Corporation首席架构师

Venkat又一次让学习变得轻松愉快。你可以像跟人聊天一样，很快学会Scala这门语言，学到它的独一无二，学到如何在多语言环境中充分利用它。

——Ian Roughley，Down & Around咨询师

多核处理器要求开发人员在函数式编程方面有着坚实的基础，而这正是Scala的核心所在。Venkat提供了一个非常棒的指南，让我们得以快速上手这门激动人心的新语言。

——Nathaniel T. Schutta，作家、演说家、教师

这本书真是让我手不释卷啊！这是一本很精彩的Scala简介！有经验的Java程序员都该来看看！这本书从Java面向对象的编程视角来介绍了“Scala之道”。完整而又简洁。

——Albert Scherer，Follett Higher Education Group软件架构师

作为程序员，并发是我们即将面临的巨大挑战，而传统的命令式语言让它更显得难比登天。Scala是JVM上的一个函数式语言，提供了便利的多线程处理、简洁的语法、与Java的无缝互操作。这本书会指引Java程序员畅游于Scala的重要特性和细微之处，让我们看到为什么人们会对这门新语言投入如此多的关注。

——Neal Ford，ThoughtWorks软件架构师/意见领袖

这本书写得很简洁，很容易读懂，也很详尽……这是目前介绍Scala的书中最棒的一本了！当我们进入无所不在的多核处理时代，作为一个程序员，如果你还想不落伍的话，就必须得读一读这本书了。接下来的几年里，我想我会反复温习这本书的。

——Arild Shirazi，CodeSherpas高级软件工程师

译 者 序

写代码的层次

初涉代码之时，我的关注点在于实现功能。初窥门径的我，不了解语言，不熟悉常见的编码技巧。那时，只要代码能够跑出想要的效果，我便欣喜若狂，无暇顾及其他。

积累一定经验之后，对于编写代码，我越来越有感觉，实现一个功能不再高不可攀。我开始了解在工程中编写代码，如何在一个系统而不仅仅是一个局部处理问题，如何解决各种bug，更重要的是，从中汲取教训，在编码中避免这些问题。

读一些软件开发的书，了解一下外面的世界，我知道了，除了自娱自乐外，代码应该是为明天而写。有个说法，对程序员最严厉的惩罚，就是让他维护自己编写的代码。于是，我开始尝试编写干净代码：短小的函数，清晰的结构……所做的一切无非就是让自己明天的日子好过一些。

历经磨练，代码逐渐干净，窃喜之际，我见到了Ruby。孤陋寡闻的我第一次听到了代码的表现力。原来代码不仅仅可以写得让开发人员容易理解，也可以让业务人员看懂。事实上，更容易懂的代码常常也意味着更容易维护。许多人关注的DSL，背后就是对于表现力的追求。

Scala就是Java平台上追求表现力的探索。

我是通过Java开始真正理解软件开发的，所以，对Java这个平台有一种难以割舍的情结。初见Scala，我看到的是，一个几乎不舍弃任何Java的优点，又能拥有更好表现力的“Java”。当有机会系统地了解这门语言时，我欣然接受了。

翻译向来是一件费力不讨好的事。认真准备的考试不见得能拿到满分，做最大的努力，做最坏的打算。于我，只希望这个译本得到的评价不是太糟糕就好。

感谢我的合作者，李剑，你给我这样的机会，让我知道，我居然还可以做翻译，你

的认真让我受益良多。感谢本书的原作者Venkat Subramaniam，和你讨论让我们对Scala有了更深刻的理解。

最后，感谢我的父母，你们教会我踏实做人，支持着我沿着软件开发这条路一直走下去。

郑 晔

2010年4月18日于成都

目　录

第1章

简　介

可以在JVM上编程的语言有很多。通过这本书，我希望让你相信花时间学习Scala是值得的。

Scala语言为并发、表达性和可扩展性而设计。这门语言及其程序库可以让你专注于问题领域，而无需深陷于诸如线程和同步之类的底层基础结构细节。

如今硬件已经越来越便宜，越来越强大。很多用户的机器都装了多个处理器，每个处理器又都是多核。虽然迄今为止，Java对我们来说还不错，但它并不是为了利用我们如今手头的这些资源而设计的。而Scala可以让你运用这些资源，创建高响应的、可扩展的、高性能的应用。

本章，我们会快速浏览一下函数式编程和Scala的益处，为你展现Scala的魅力。在本书的其他部分，你将学会如何运用Scala，利用这些益处。

1.1　为何选择 Scala

Scala是适合你的语言吗？

Scala是一门混合了函数式和面向对象的语言。用Scala创建多线程应用时，你会倾向于函数式编程风格，用不变状态（immutable state）[①]编写无锁（lock-free）代码。Scala提供一个基于actor的消息传递（message-passing）模型，消除了涉及并发的痛苦问题。运用这个模型，你可以写出简洁的多线程代码，而无需顾虑线程间的数据竞争，以及处理加锁和释放带来的梦魇。把synchronized这个关键字从你的字典中清除，享受Scala带来的高效生产力吧。

然而，Scala的益处并不仅限于多线程应用。你可以用它构建出强大而简洁的单线程

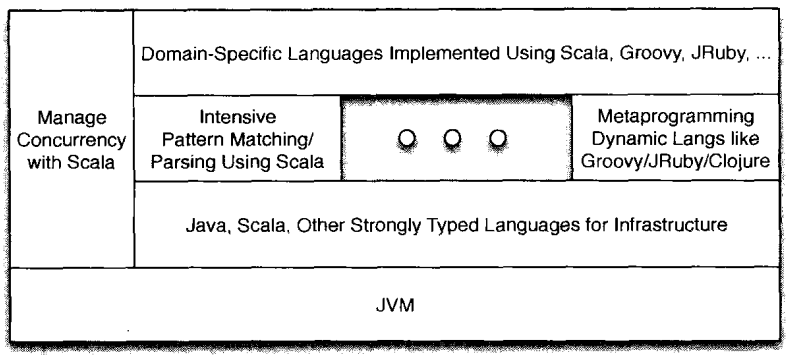
① 对象一旦创建出来，就不再改变其内容，这样的对象就是不变的。这也就无需顾虑多线程访问对象时的竞争管理。Java的String就是不变对象一个非常好的例子。

应用，或是多线程应用中的单线程模块。你很快就可以用上Scala的强大能力，包括自适应静态类型、闭包、不变的容器以及优雅的模式匹配。

Scala对于函数式编程的支持让你可以写出简洁而有表现力的代码。感谢更高层的抽象，它让我们可以用更少的代码做更多的事情。单线程应用和多线程应用都可以从函数式风格中受益。

函数式编程语言也为数不少。比如，Erlang就是一个很好的函数式编程语言。实际上，Scala的并发模型同Erlang的非常相似。然而，同Erlang相比，Scala有两个显著的优势。第一，Scala是强类型的，而Erlang不是。第二，不同于Erlang，Scala运行于JVM之上，可以与Java很好地互操作。

就运用在企业级应用的不同层面而言，Scala这两个特性使其成为了首选。只要你愿意，就可以用Scala构建整个企业级应用，或者，也可以把它和其他语言分别用在不同的层上。如果有些层在你的应用中至关重要，你就可以用上Scala的强类型、极佳的并发模型和强大的模式匹配能力。下图的灵感源自Ola Bini的语言金字塔（参见附录A的“Fractal Programming”），它展现了Scala在企业级应用中与其他语言的配合。



JVM上的其他语言Groovy，JRuby，Clojure怎么样呢？

目前为止，能够同时提供函数式风格和良好并发支持的强类型语言，唯有Scala；这正是它的卓越之处。JRuby和Groovy是动态语言，它们不是函数式的，也无法提供比Java更好的并发解决方案。另一方面，Clojure是一种混合型的函数式语言。它天生就是动态的，因此不是静态类型。而且，它的语法类似于Lisp，除非你很熟悉，否则这可不是一种易于掌握的语法。

如果你是个有经验的 Java程序员，正在头痛用Java实现多线程应用，那么你就会发现Scala非常有用。你可以相当容易地就把Java代码封装到Scala的actor中，从而实现线程

隔离。还可以用Scala的轻量级API传递消息，以达到线程通信的目的。与“启动线程，立即用同步的方式限制并发”不同，你可以通过无锁消息传递享受真正的并发。

如果你重视静态类型，喜欢编译器支持所带来的益处，你会发现，Scala提供的静态类型可以很好地为你工作，而不会阻碍你。你会因为使用这种无需键入太多代码的类型而感到惬意。

如果你喜欢寻求更高层次的抽象和具有高度表现力的代码，你会被Scala的简洁所吸引。在Scala里，你可以用更少的代码做更多的事情。了解了运算符和记法，你还会发现Scala的灵活性，这对于创建领域专用语言（domain-specific language）非常有用。

提醒一下，Scala的简洁有时会倾向于简短生硬，这会让代码变得难以理解。Scala的一些运算符和构造对初学者而言可能一时难以适应[①]。这样的语法不是为胆小之人准备的。随着你逐渐精通Scala，你会开始欣赏这种简洁，学会避免生硬，使得代码更易于维护，同时也更易于理解。

Scala不是一种超然物外的语言。你不必抛弃你已经为编写Java代码所投入的时间、金钱和努力。Scala和Java的程序库是可以混合在一起的。你可以完全用Scala构建整个应用，也可以按照你所期望的程度，将它同Java或其他JVM上的语言混合在一起。因此，你的Scala代码可以小如脚本，也可以大如全面的企业应用。Scala已经用于构建不同领域的应用，包括电信、社交网络、语义网和数字资产管理。Apache Camel用Scala做DSL创建路由规则。Lift Web Framework是一个用Scala构建的强大的Web开发框架，它充分利用了Scala的特性，比如简洁、表现力、模式匹配和并发。

1.2 何为 Scala

Scala，是Scalable Language的缩写，它是一门混合型的函数式编程语言。Martin Odersky[②]是它的创始人，2003年发布了第一个版本。下面是Scala的一些关键特性[③]：

- 它拥有基于事件的并发模型；
- 它既支持命令式风格，也支持函数式风格；
- 它是纯面向对象的；
- 它可以很好的与Java混合；
- 它强制使用自适应静态类型；

---

① 我着手学习一门新语言时，还没有哪门语法不让我头疼的，包括Ruby。多多练习，很快语法就变得很自然了。

② 请阅读附录A，了解更多信息。

③ 请参考附录A，获得权威的语言规范。

- 它简洁而有表现力；
- 它构建于一个微内核之上；
- 它高度可扩展，可以用更少的代码创建高性能应用。

下面的小例子突出了这些特性：

Introduction/TopStock.scala

```
import scala.actors._
import Actor._

val symbols = List( "AAPL", "GOOG", "IBM", "JAVA", "MSFT")
val receiver = self
val year = 2008

symbols.foreach { symbol =>
  actor { receiver ! getYearEndClosing(symbol, year) }
}

val (topStock, highestPrice) = getTopStock(symbols.length)

printf("Top stock of %d is %s closing at price %f\n", year, topStock,
  highestPrice)
```

不用想语法，我们先从大处着眼。symbols指向一个不变的List，其中持有股票代码。我们对这些股票代码进行循环，调用actor。每个actor在单独的线程中执行。因此，同actor关联的代码块({})运行在其自己的线程上。它调用（尚未实现的）函数getYearEndClosing()。这个调用的结果返回发起请求的actor。这由特殊的符号（!）实现。回到主线程，我们调用（尚未实现的）函数getTopStock()。在上面的代码完全实现之后，我们就可以并发地查询股票收盘价了。

现在，我们看看函数getYearEndClosing()：

Introduction/TopStock.scala

```
def getYearEndClosing(symbol : String, year : Int) = {
  val url = "http://ichart.finance.yahoo.com/table.csv?s=" +
    symbol + "&a=11&b=01&c=" + year + "&d=11&e=31&f=" + year + "&g=m"

  val data = io.Source.fromURL(url).mkString
  val price = data.split("\n")(1).split(",")(4).toDouble
  (symbol, price)
}
```

在这个短小可爱的函数里面，我们向http://ichart.finance.yahoo.com发出了一个请求，收到了以CSV格式返回的股票数据。我们解析这些数据，提取年终收盘价。现在，先不必为收到数据的格式操心，它并不是我们要关注的重点。在第14章，我们还将

再用到这个例子，提供所有与Yahoo服务交流的细节。

还需要实现getTopStock()方法。在这个方法里，我们会收到收盘价，确定最高价的股票。我们看看如何用函数式风格实现它：

Introduction/TopStock.scala

```
def getTopStock(count : Int) : (String, Double) = {
  (1 to count).foldLeft("", 0.0) { (previousHigh, index) =>
    receiveWithin(10000) {
      case (symbol : String, price : Double) =>
        if (price > previousHigh._2) (symbol, price) else previousHigh
    }
  }
}
```

在这个getTopStock()方法中，没有对任何变量进行显式赋值的操作。我们以股票代码的数量作为这个方法的参数。我们的目标是找到收盘价最高的股票代码。因此，我们把初始的股票代码和高价设置为（"",0.0），以此作为foldLeft()方法的参数。我们用foldLeft()方法去辅助比较每个股票的价格，确定最高价。通过receiveWithin()方法，我们接收来自开始那个actor的股票代码和价格。如果在指定时间间隔没有收到任何消息，receiveWithin()方法就会超时。一收到消息，我们就会判断收到的价格是否高于我们当前的高价。如果是，就用新的股票代码及其价格作为高价，与下一次接收的价格进行比较。否则，我们使用之前确定的（previousHigh）股票代码和高价。无论从附着于foldLeft()的代码块（code block）中返回什么，它都会作为参数，用于在下一元素的上下文中调用代码块。最终，股票代码和高价从foldLeft()返回。再强调一次，从大处着眼，不要管这里的方法的细节。随着学习的深入，你会逐步了解它们的详细内容。

大约25行代码，并发地访问Web，分析选定股票的收盘价。花上几分钟，分析一下代码，确保你理解了它是如何运作的。重点看方法是如何在不改变变量或对象的情况下，计算最高价的。整个代码只处理了不变状态；变量或对象在创建后就没有修改。其结果是，你不需要顾虑同步和数据竞争，代码也不需要有显式的通知和等待序列。消息的发送和接收隐式地处理了这些问题。

如果你把上面所有的代码放到一起，执行，你会得到如下输出：

```
Top stock of 2008 is GOOG closing at price 307.650000
```

假设网络延迟是d秒，需要分析的是n个股票代码。如果编写代码是顺序运行，大约要花n×d秒。因为我们并行执行数据请求，上面的代码只要花大约d秒即可。代码中最大的延迟会是网络访问，这里我们并行地执行它们，但并不需要写太多代码，花太多精力。

想象一下，用Java实现上面的例子，你会怎么做。

上面的代码的实现方式与Java截然不同，这主要体现在下面3个方面。

- 首先，代码简洁。Scala一些强大的特性包括：actor、闭包、容器（collection）、模式匹配、元组（tuple），而我们的示例就利用了其中几个。当然，我还没有介绍过它们，这还只是简介！因此，不必在此刻就试图理解一切，通读本书之后，你就能够理解它们了。
- 我们使用消息进行线程间通信。因此不再需要`wait()`和`notify()`。如果你使用传统Java线程API，代码会复杂几个数量级。新的Java并发API通过使用executor服务减轻了我们的负担。不过，相比之下，你会发现Scala基于actor的消息模型简单易用得多。
- 因为我们只处理不变状态，所以不必为数据竞争和同步花时间或精力（还有不眠夜）。

这些益处为你卸下了沉重的负担。要详细地了解使用线程到底有多痛苦，请参考Brian Goetz的*Java Concurrency in Practice* [Goe06]。运用Scala，你可以专注于你的应用逻辑，而不必为低层的线程操心。

你看到了Scala并发的益处。Scala也并发地[①]为单线程应用提供了益处。Scala让你拥有选择和混合两种编程风格的自由：Java所用的命令式风格和无赋值的纯函数式风格。Scala允许混合这两种风格，这样，你可以在一个线程范围内使用你最舒服的风格。Scala使你能够调用和混合已有的Java代码。

在Scala里，一切皆对象。比如，`2.toString()`在Java里会产生编译错误。然而，在Scala里，这是有效的——我们调用`Int`实例的`toString()`方法。同时，为了能给Java提供良好性能和互操作性，在字节码层面上，Scala将`Int`的实例映射为32位的基本类型`int`。

Scala编译为字节码。你可以按照运行Java语言程序相同的方式运行它。[②]也可以很好的将它同Java混合起来。你可以用Scala类扩展Java类，反之亦然。你也可以在Scala里使用Java类，在Java里使用Scala类。你可以用多种语言编写应用，成为真正的多语言程序员[③]——在Java应用里，在需要并发和简洁的地方，就用Scala（比如创造领域特定语言）吧！

Scala是一个静态类型语言，但是，不同于Java，它拥有自适应的静态类型。Scala

① 这里一语双关。——编者注

② 你可以把它当作脚本运行。

③ 参见附录A，也请阅读Neal Ford著的*The Productive Programmer* [For08]。

在力所能及的地方使用类型推演。因此，你不必重复而冗繁地指定类型，而可以依赖语言来了解类型，在代码的剩余部分强制执行。不是你为编译器工作；相反，编译器为你工作。比如，我们定义var i = 1，Scala立即就能推演出变量i是Int类型。现在，如果我们将某个字符串赋给那个变量，比如，i = "haha"，编译器就会给出如下的错误：

```
error: type mismatch;
 found   : java.lang.String("haha")
 required: Int
       i = "haha"
```

在本书后面，你会看到类型推演超越了简单类型定义，也进一步超越了函数参数和返回值。

Scala偏爱简洁。在语句结尾放置分号是Java程序的第二天性。Scala可以为你的小拇指能从多年的虐待中提供一个喘息之机——分号在Scala中是可选的。但是，这只是个开始。在Scala中，根据上下文，点运算符（.）也是可选的，括号也是。因此，不用写成s1.equals(s2);，我们可以这么写s1 equals s2。去掉了分号、括号和点，代码会有一个高信噪比。它会变成更易编写的领域特定语言。

Scala最有趣的一个方面是可扩展性。你可以很好享受到函数式编程构造和强大的Java程序库之间的相互作用，创建高度可扩展的、并发的Java应用，运用Scala提供的功能，充分发挥多核处理器的多线程优势。

Scala真正的魅力在于它内置规则极少。相比于Java，C#和C++，Scala语言只内置了一套非常小的内核规则。其余的，包括运算符，都是Scala程序库的一部分。这种差异具有深远的影响。因为语言少做一些，你就能用它多做一些。这是真正的可扩展，它的程序库就是一个很好的研究案例。

1.3 函数式编程

我已经提过几次，Scala可以用作函数式编程语言。我想花几页的篇幅给你一些函数式编程的感觉。让我们从对比Java编程的命令式风格开始吧！如果我们想找到给定日期的最高气温，可能写出这样的Java代码：

```
//Java code
public static int findMax(List<Integer> temperatures) {
  int highTemperature = Integer.MIN_VALUE;
  for(int temperature : temperatures) {
   highTemperature = Math.max(highTemperature, temperature);
  }
  return highTemperature;
}
```