函数式编程图解

[波兰] 米哈尔·普瓦赫塔(Michał Płachta) 著
郭 涛 译

清華大學出版社
北 京

北京市版权局著作权合同登记号 图字：01-2024-0883
Michał Płachta
Grokking Functional Programming
EISBN: 978-1-61729-183-8

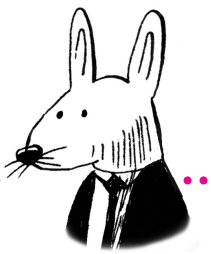
图书在版编目 (CIP) 数据
函数式编程图解 / (波) 米哈尔·普瓦赫塔著 ; 郭涛译 . -- 北京 : 清华大学出版社 , 2025. 2.
ISBN 978-7-302-67928-8
Ⅰ . TP312-64
中国国家版本馆 CIP 数据核字第 2025AD7352 号

责任编辑：王　军　刘远菁
封面设计：高娟妮
装帧设计：恒复文化
责任校对：马遥遥
责任印制：刘　菲

出版发行：清华大学出版社
网　　址：https://www.tup.com.cn，https://www.wqxuetang.com
地　　址：北京清华大学学研大厦A座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：大厂回族自治县彩虹印刷有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　印　　张：30.75　　字　　数：636千字
版　　次：2025年2月第1版　　印　　次：2025年2月第1次印刷
定　　价：168.00元

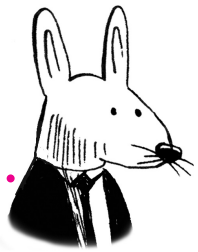
产品编号：101250-01

译者简介



郭涛主要从事人工智能、现代软件工程、智能空间信息处理与时空大数据挖掘分析等前沿领域的研究。翻译了多部计算机书籍，包括《函数式与并发编程》《Effective数据科学基础设施》和《重构的时机和方法》。

译 者 序



早在20世纪50年代，美国计算机科学家约翰•麦卡锡(John McCarthy)为IBM 700/7000系列机器发明了第一门函数式编程语言Lisp。Lisp最初创建时受到阿隆佐•邱奇的lambda演算的影响，在处理数学和逻辑运算方面具有高度的灵活性。因为是早期的高端编程语言之一，它很快成为人工智能研究中最受欢迎的编程语言。作为第一门函数式编程语言，Lisp开创了很多先驱概念。经过几十年的发展，形成了你所看到的现代函数式编程语言，函数式编程是一种编程风格，脱离特定的语言特性，函数式代码易于测试、复用。

与命令式编程相比，函数式编程将计算过程抽象为表达式求值。其中表达式由纯数学函数构成，这些数学函数是第一类对象且无副作用。因此，函数式编程很容易做到线程安全，且具有并发编程的优势。

目前，C++、Scala、Java、C#、Python等高级编程语言也设计了函数式编程语言特性。但函数式编程语言设计思想抽象，特性比较多，这给很多读者带来了很大的困扰，尤其是涉及并发的编程，已成为很多人的梦魇。本书以图解方式，以Scala和Java语言作为实现载体，通过大量的代码示例和案例呈现出了函数式编程语言的特性。本书内容比较基础，建议读完本书的读者阅读译者翻译的另一本著作——《函数式与并发编程》(*Functional and Concurrent Programming*)，该书与本书一脉相承，都以Scala和Java作为示例，主要围绕函数式编程和并发编程高级特性展开讲解。本书适合计算机科学与工程、软件工程、人工智能专业的高年级本科生和企业中对函数式编程感兴趣的工程师阅读。

感谢吉林大学外国语学院研究生吴禹林、电子科技大学外国语学院高丹丹参与本书的翻译、审核和校对工作。由于函数式编程主题涉及的特性广泛且术语众多，国内学者对该主题的翻译常常不一致，为了确保本书的术语一致性，本书的用语参照了以下出版物：张骏温翻译的《C#函数式编程(第2版)》、程继洪等翻译

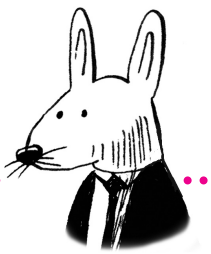
的《C++函数式编程》、王宏江等翻译的《Scala函数式编程》。此外，我要感谢清华大学出版社的编辑、校对和排版工作人员，感谢他们为了保证本书质量所做的一切。

由于本书涉及内容广泛、深刻，加上译者翻译水平有限，本书难免存在不足之处，恳请各位读者不吝指正。

致亲爱的家人：Marta、Wojtek和Ola。
感谢你们给我带来的正能量和启发。

致父母：Renia和Leszek。
感谢你们给予的所有机会。

前　言



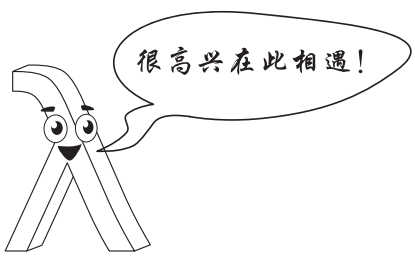
你好！感谢购买《函数式编程图解》。过去十年，我一直在与程序员讨论编程方法、其可维护性以及函数式编程概念逐渐被主流语言所采用这一趋势。许多专业开发人员表示，目前仍然很难从现有资源中学习函数式概念，因为这些资源要么过于简单，要么过于复杂。这就是本书试图填补的空白。本书旨在为那些想要全面了解基本函数式编程概念的程序员提供一种循序渐进的实用指南。

实践出真知，这就是本书大量使用实例的原因。读完这本入门书后，你将能够使用函数式方案编写功能齐全的程序，并轻松深入研究其理论基础。

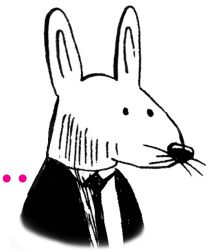
如果你曾使用命令式面向对象语言(如Java或Ruby)创建重要的应用程序，那么你将从本书中获益匪浅。如果你所在的团队曾应对大量错误和可维护性问题，那么本书将是你的一大助力，因为这正是函数式编程的用武之地。

希望你喜欢阅读本书并完成习题，最好能像我写作时一样享受。再次感谢你对本书的喜爱！

——Michał Płachta



致　　谢



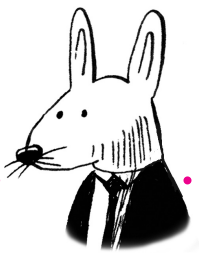
首先，我要感谢 Scala 社区对工具和技术的不懈追求，这些工具和技术有助于构建可维护的软件。本书中介绍的所有想法都源自无数次的代码审查、讨论、多篇涉及大量回复的博客文章、即兴演示和生产故障事后分析。感谢你们的热情。

我要感谢我的家人，尤其是妻子Marta，她在写作过程中给予我大量的鼓励和爱。感谢可爱的孩子 Wojtek 和 Ola，他们确保我不会长时间坐在计算机前。

本书凝聚了许多人的心血。我要感谢 Manning 的工作人员：策划编辑Michael Stephens、编辑Bert Bates、开发编辑Jenny Stout、技术开发编辑Josh White、文稿编辑Christian Berk、出版编辑Keri Hales、技术审校Ubaldo Pescatore、校对Katie Tennant，以及所有参与本书出版的幕后人员。

感谢所有审稿人：Ahmad Nazir Raja、Andrew Collier、Anjan Bacchu、Charles Daniels、Chris Kottmyer、Flavio Diez、Geoffrey Bonser、Gianluigi Spagnuolo、Gustavo Filipe Ramos Gomes、James Nyika、James Watson、Janeen Johnson、Jeff Lim、Jocelyn Lecomte、John Griffin、Josh Cohen、Kerry Koitzsch、Marc Clifton、Mike Ted、Nikolaos Vogiatzis、Paul Brown、Ryan B. Harvey、Sander Rossel、Scott King、Srihari Sridharan、Taylor Dolezal、Tyler Kowallis 和 William Wheeler。感谢你们，你们的建议让这本书变得更完美。

关于本书



本书目标读者

本书假设读者曾使用主流面向对象编程语言(如Java)且至少有一年的商业软件开发经验。本书中的示例将Scala用作教学语言，但本书并不是关于Scala的书。读者不需要事先了解Scala或函数式编程。

本书结构：阅读指南

本书分为三部分。第Ⅰ部分奠定基础，将探讨函数式编程(functional programming，FP)中通用的工具和技术。第1章将讨论如何使用本书学习FP。第2章将展示纯函数和非纯函数之间的区别。第3章将介绍不可变值。最后，第4章将说明纯函数只是不可变值，并展示其强大作用。

在本书的第Ⅱ部分，只使用不可变值和纯函数解决实际问题。第5章将介绍FP中最重要的函数，并展示如何以简洁和易读的方式构建顺序值和程序。在第6章，将学习如何构建可能返回错误的顺序程序。在第7章，将学习有关函数式软件设计的知识。第8章将教你如何以安全和函数式的方案处理非纯、外部、有副作用的问题。然后，第9章将介绍流和流式系统。我们将使用函数式方案构建数十万个项的流。在第10章，最终将创建一些函数式、安全的并发程序。

在第Ⅲ部分，将维基数据用作数据源以实现一个真实的函数式应用程序。将借此回顾前面两部分中学到的所有内容。在第11章，需要创建一个基于不可变值的数据模型，并使用含IO在内的正确类型，与维基数据集成，并使用缓存和多线程提高应用程序的速度。将把所有这些问题都封装在纯函数中，并展示如何在函数式世界中重用面向对象的设计。第12章将展示如何测试在第11章中开发的应用程序，你将看到，即使需求大幅变化，应用程序也易于维护。

最后，将以一组练习作为本书的结尾，以确保你掌握了函数式编程。

关于代码

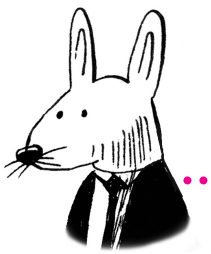
本书包含许多源代码示例，源代码都以等宽字体格式化，以将其与普通文本区分开来。有时代码也会用粗体表示，以突出显示与同一章中的前几步相比发生变化的代码，例如当新功能添加到现有代码行时，新增的代码将以粗体显示。

许多情况下，原始源代码已被重新格式化；我们添加了换行符并重新调整了缩进，以使代码适应书中可用的页面空间。许多代码清单附带注释，以强调重要概念。

你可以扫描本书封底的二维码以获取可执行的代码片段，以及本书完整的示例代码。

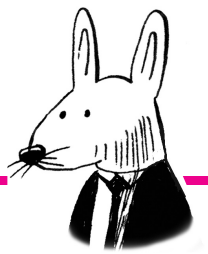
此外，本书所有扩展资源(包括附录和附加材料)均可通过扫描封底二维码获得。

作者简介



Michał Płachta是一位经验丰富的软件工程师，也活跃于函数式编程社区。他经常在技术会议上发言，主持研讨会，组织聚会，并在博客上发表文章，探讨如何创建可维护的软件。

目　录



第I部分　函数式工具包

第II部分 函数式程序

第5章 顺序程序 135

第8章 作为值的IO 269

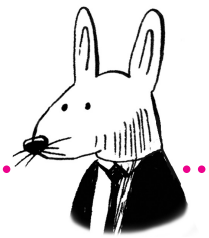
第9章 作为值的流 313

第III部分 应用函数式编程

第11章 设计函数式程序 399

第12章 测试函数式程序 427

第 I 部分
函数式工具包



本书的第 I 部分为本书奠定基础。你将学习函数式编程(FP)中通用的工具和技术。在这一部分学到的所有知识都将在后续章节和你的职业生涯中重复使用。

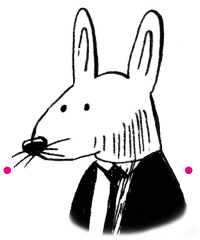
第1章讨论基础知识，并确保你能够适应本书所采用的教学方法。你将设置环境，编写一些代码，并完成一些简单的练习。

第2章讨论纯函数和非纯函数之间的区别。将使用一些命令式示例来展示风险，并通过函数式代码缓解这些风险。

第3章介绍纯函数的搭档：不可变值。将介绍纯函数与不可变值为何缺一不可，它们共同定义了函数式编程。

最后，第4章将展示作为值的纯函数，并说明其强大作用。这将使你能够将所有组件连接在一起，并组装第一个完整的函数式工具包。

第1章 学习函数式编程



本章内容：

- 本书的读者对象
- 函数是什么
- 函数式编程的作用
- 如何安装所需工具
- 如何使用本书

“我只能通过具体例子向你阐述，至于其本质，要由你自行推断。”

——Richard Hamming，《学会学习》

1.1 也许你选择本书是因为……

你对函数式编程很感兴趣 1

你听说过函数式编程，读过维基百科的条目，也看过几本相关主题的书。也许你对代码蕴含的数学解释感到厌烦，但你仍然对它感到好奇。

> 我一直想写一本简单易懂的关于函数式编程的书。就是此书：入门级别，实用性强，尽量不会让你感到厌烦。

你曾试图学习函数式编程 2

你不止一次尝试学习函数式编程，但仍然没有掌握。当你理解了一个关键概念时，又出现了其他问题。若要处理问题，又需要你理解更多内容。

> 学习函数式编程的过程应该是令人愉快的。本书鼓励你进行尝试，相信你有兴趣坚持下去。

你还犹豫不决 3

多年来，你一直使用面向对象或命令式编程语言进行编程。你体验过函数式编程，读过一些博客文章，并尝试编写了一些代码。但你仍然不了解它能如何改善你的编程生涯。

> 本书侧重于函数式编程的实际应用，将向你介绍一些函数式概念。无论你使用什么语言，都能使用这些概念。

或者还有其他原因

不管你的理由是什么，本书都试图以不同的方式满足你的需求。本书注重通过实验和游戏学习，鼓励你提出问题，并通过编程找到答案。本书将帮助你进阶为更高级的程序员。希望你能享受这段旅程。

1.2 你应掌握的背景知识

假设你已经使用任何一种流行的语言开发软件，如Java、C++、C#、JavaScript或者Python。这是一个非常模糊的说法，因此请简单核对下列信息，以帮助你适应本书节奏。

如果你具备以下条件，将会轻松理解本书内容：

- 熟悉类和对象等基本的面向对象概念。
- 能够阅读和理解如下代码：

```
class Book {
  private String title;
  private List<Author> authors;

  public Book(String title) {
    this.title = title;
    this.authors = new ArrayList<Author>();
  }

  public void addAuthor(Author author) {
    this.authors.add(author);
  }
}
```

Book有一个标题和一组Author对象

构造器：创建一个有标题但没有作者的新Book对象

为这个Book实例添加一个Author

本书最适合的场景：

- 你的软件模块存在稳定性、可测试性、回归或集成问题。
- 你在调试如下代码时遇到了问题：

这汤可能并不好喝……

```
public void makeSoup(List<String> ingredients)  {
  if(ingredients.contains("water")) {
    add("water");
  } else throw new NotEnoughIngredientsException();
  heatUpUntilBoiling();
  addVegetablesUsing(ingredients);
  waitMinutes(20);
}
```

你不必：

- 是面向对象编程专家。
- 精通Java / C++ / C#/ Python。
- 了解任何函数式编程语言(如Kotlin、Scala、F#、Rust、Clojure或Haskell)。

1.3 函数是什么样的

话不多说，直接进入代码！虽然现在还没有设置好所有必要的工具，但这无法妨碍我们，不是吗？

下面给出一些不同的函数。它们有一个共同点：都以一些值作为输入，进行一些操作，并可能返回值作为输出。

```
public static int add(int a, int b) {
  return a + b;
}
```

取两个int，相加并返回结果

```
public static char getFirstCharacter(String s) {
  return s.charAt(0);
}
```

取一个String并返回其第一个字符

```
public static int divide(int a, int b) {
  return a / b;
}
```

取两个int，将第一个除以第二个并返回结果

```
public static void eatSoup(Soup soup) {
  // TODO: "eating the soup" algorithm
}
```

取Soup对象，对其执行某些操作，并且不返回任何内容

为什么所有函数都用public static

你可能注意到了每个函数定义中的public static修饰符。确实，它的存在是有意义的。本书中使用的函数都是静态的(即它们不需要执行任何对象实例)。它们是自由的——任何人都可以从任何地方调用它们，前提是调用者具有它们所需的输入参数。这些函数只使用调用者提供的数据——再无其他。

当然，这会产生一些重大影响，详见本书后续讨论。现在要记住的是，当提到函数时，指的是可以从任何地方调用的public static函数。

快速练习

执行下面的两个函数：

```
public static int increment(int x) {
  // TODO
}
public static String concatenate(String a, String b) {
  // TODO
}
```

答案：

```
return x + 1;
return a + b;
```

1.4 认识函数

如你所见，函数有不同的类型。基本上，每个函数由一个特征标记和一个执行特征标记的函数体组成。

```
public static int add(int a, int b) {
  return a + b;
}
```

函数特征标记

函数体

本书将重点讨论返回值的函数(见图1-1)，你将了解到，这些函数是函数式编程的核心。不使用不返回任何值(即void)的函数。

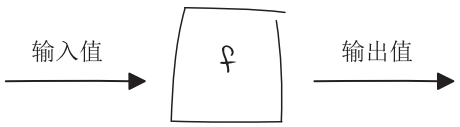


图1-1 返回值的函数

可将函数视为一个盒子，它获取输入值，对其进行处理并返回输出值。函数体便在盒子里。输入值和输出值的类型和名称是特征标记的一部分。因此，可以将add函数表示为图1-2所示形式。

图1-2 add函数

重点！

在函数式编程中，比起使用的函数体，更注重函数的特征标记

特征标记与函数体

在图1-2中，函数的执行(即函数体)隐藏在盒子内，而特征标记则是公开可见的。这是两者之间非常重要的区别。如果仅凭特征标记就能理解盒子内的内容，那么对于阅读代码的程序员来说，这是一大优势，因为他们只有在使用函数时才需要分析盒子内的函数是如何实现的。

快速练习

为下面的函数画一个函数图。盒子内是什么？

```
public static int increment(int x)
```

答案：

有一个名为int x的输入箭头和一个名为int的输出箭头。实现代码是return x+1;

1.5 当代码说谎时……

程序员遇到的一个非常棘手的问题是，代码执行了它不该执行的操作。这种问题通常是特征标记和函数体的不一致导致的。要了解这一点，不妨先简单回顾一下前面的4个函数(见图1-3)：

```
public static int add(int a, int b) {
  return a + b;
}

public static char getFirstCharacter(String s) {
  return s.charAt(0);
}

public static int divide(int a, int b) {
  return a / b;
}

public static void eatSoup(Soup soup) {
  // TODO: "eating a soup" algorithm
}
```

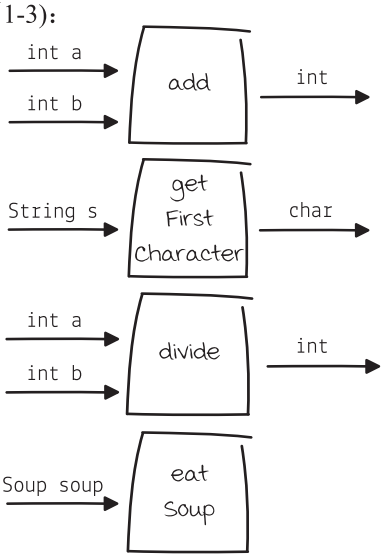


图1-3 4个函数

令人惊讶的是，以上4个函数中有3个存在缺陷。

问： 函数会说谎吗？

答： 很遗憾，会。上面的一些函数就在一本正经地说谎。这通常是因为特征标记没有完全说明实际要执行的函数体。

若输入一个String，getFirstCharacter函数理应返回一个char。然而，若输入一个空String，该函数并不会返回任何字符，而是抛出一个异常！

如果将0作为b输入divide函数，该函数将不会返回预期的int。

eatSoup函数理应喝掉输入的汤，但是当输入汤时，该函数返回void，并未进行其他操作。这可能是大多数新手的默认实现结果。

而对于add函数，无论将什么值作为a和b输入，该函数都将返回预期的int。这样的函数是可信的！

本书将重点讨论可信的函数。希望函数的特征标记能够说明函数体的全部信息。你将学习如何使用这类函数构建真实的程序。

阅读本书后，你将能轻松将函数改写为可信的版本

重点！
可信的函数是函数式编程非常重要的特征

1.6 命令式与声明式

某些程序员将编程语言分为两大范式：命令式和声明式。下面通过一个简单的练习来理解这两种范式之间的差异。

假设我们的任务是创建一个函数，以在某个单词游戏中计算分数。当玩家提交一个单词时，函数将返回一个分数。单词中每个字符得一分。

命令式计算分数

```
public static int calculateScore(String word)  {
  int score = 0;
  for(char c : word.toCharArray()) {
    score++;
  }
  return score;
}
```

开发人员这样读：要计算单词的分数，首先将分数初始化为0，然后遍历单词的字符，并为每个字符增加分数。返回分数

命令式编程关注如何计算结果。重点是按照特定顺序定义特定步骤。通过提供详细的逐步算法来实现最终结果。

声明式计算分数

```
public static int wordScore(String word)  {
  return word.length();
}
```

开发人员这样读：单词的分数是其长度

声明式解法关注需要执行什么操作，而不是如何完成。在本例中，我们需要这个字符串的长度，并将此长度作为特定单词的分数返回。因此，只使用Java的String中的length方法来获取字符数，而不关心它是如何计算出来的。

声明式解法还将函数名从calculateScore改为wordScore。这看起来是细微的差别，但名词的使用会使我们的大脑切换到声明模式，并注重需要完成的内容，而不是如何实现的细节。

声明式代码通常比命令式代码更简洁、易懂。即使许多内部实现(如JVM或CPU)都是命令性的，但应用程序开发人员仍然可以大量使用声明式解法并隐藏命令式内部实现，就像对length函数的处理一样。在本书中，你将学习如何使用声明式解法编写真实的程序。

此外，SQL几乎也是一种声明式语言。通常情况下，你指定所需要的数据，而不用关心获取数据的方式(至少在开发过程中是这样的)

1.7 小憩片刻：命令式与声明式

欢迎来到本书的第一个“小憩片刻”！此练习部分将确保你已经掌握了命令式和声明式方法之间的区别。

本书中的“小憩片刻”意指什么

本书中有几类不同的练习。你已经遇到了第一种：快速练习。这种练习使用一个大问号标记，并分散在本书中。这些练习非常简单，不必借用纸张或计算机即可解决。

第二种类型是小憩片刻。这种练习假设你有一些时间，有一张纸或一台计算机，并且你想开动脑筋。每次练习时，尽量让你熟悉某个主题。这对学习过程至关重要。

一些“小憩片刻”练习可能较难，但即使你无法解决，也不要担心。问题的下一页会给出答案和解释。但在查阅答案之前，请确保你已经尝试思考5～10分钟左右。就算你没有弄清楚，此过程仍有助于掌握当下的内容。

你自己探究的时间越长，你学到的越多

在此练习中，需要增强命令式calculateScore和声明式wordScore函数。新要求规定，单词的分数现在应该等于不同于a的字符数。你的任务如下。可以使用以下代码：

```
public static int calculateScore(String word)  {
  int score = 0;
  for(char c : word.toCharArray()) {
    score++;
  }
  return score;
}

public static int wordScore(String word)  {
  return word.length();
}
```

在查看下一页之前，请务必先自行思考答案。最好的办法是把答案写在纸上或使用计算机

更改以上函数，使以下条件成立：

```
calculateScore("imperative") == 9  wordScore("declarative") == 9
calculateScore("no") == 2          wordScore("yes") == 3
```

1.8 解释：命令式与声明式

但愿你的第一次“小憩片刻”练习一切顺利。现在该检查答案了。先来看命令式解法。

命令式解法

命令式解法强烈鼓励直接实现算法——强调“如何”。因此，需要获取单词，浏览该单词中的所有字符，为每个不同于a的字符增加分数，并在完成时返回最终分数。

```
public static int calculateScore(String word)  {
  int score = 0;
  for(char c : word.toCharArray()) {
    if(c != 'a')
      score++;
  }
  return score;
}
```

就是这样！只是在for循环内添加了一个if语句。

声明式解法

声明式解法侧重于“什么”。在本例中，要求是声明式的：“单词的分数现在应该等于不同于a的字符数。”几乎可以直接实现此要求：

```
public static int wordScore(String word)  {
  return word.replace("a", "").length();
}
```

或者，可以引入一个辅助函数。

```
public static String stringWithoutChar(String s, char c) {
  return s.replace(Character.toString(c), "");
}

public static int wordScore(String word)  {
  return stringWithoutChar(word, 'a').length();
}
```

重点！

在函数式编程中，更多地关注需要发生什么事情，而非事情应该如何发生

你可能有不同解法。如果解法的重点是没有a的字符串(强调“什么”)，而不是for和if(强调“如何”)，则可以接受。

1.9 学习函数式编程的益处

函数式编程(FP)是指使用具有以下特征的函数进行编程：

- 不说谎的特征标记。
- 最好为声明式的函数体。

本书将逐步探讨这些主题，最终你将能够轻松用本书介绍的方案构建真实的程序。仅这一点就能产生巨大影响。然而，好处不止于此。通过本书学习函数式编程，还可获得其他益处，如图1-4所示。

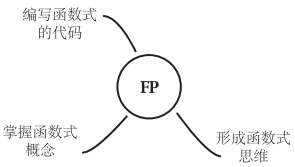


图1-4 学习函数式编程的益处

可以使用任何一种语言编写代码

到目前为止，我们已经使用Java编写了函数，尽管它被认为是一种面向对象的命令式语言，仍可以用来编写函数式代码。事实证明，声明式和函数式编程的技术和特性也能在Java和其他传统的命令式语言中使用。你可使用自己选择的语言中的一些技术。

FP语言中的函数式概念是相同的

本书重点是函数式编程的通用特性和技术。这意味着，如果你在本书中学习了某个概念(使用Scala)，那么你将能将其应用到许多其他函数式编程语言中。更多地关注FP语言之间的共通之处，而不是单一语言的特殊性。

函数式和声明式思维

你将学习的最重要的技能之一是如何使用不同方案解决编程问题。这些函数式技术将成为你的软件工程工具箱中的强大工具。无论你此前如何，这一新视角都一定会帮助你在职业上有所成长。

1.10 进入Scala

本书中的大部分示例和练习都使用Scala。如果你不了解这门语言，不用担心，你很快就会掌握所有必要的基础知识。

问： 为什么使用Scala?

答： 这是实用之选。Scala具有所有的函数式编程特性，同时其语法仍与一门主流命令式语言相似。这可以使学习过程更加顺畅。请记住，要节约在语法上花费的时间。要适当学习Scala知识，使你能够讨论函数式编程的更复杂概念。还要适当学习语法，使你能够用函数式方案解决大型的真实编程问题。最后，将Scala视为一个教学工具。通读本书后，你将自行判断Scala是否足以帮你应对日常编程任务，或者你是否想要学习语法更复杂但概念相同的其他函数式编程语言。

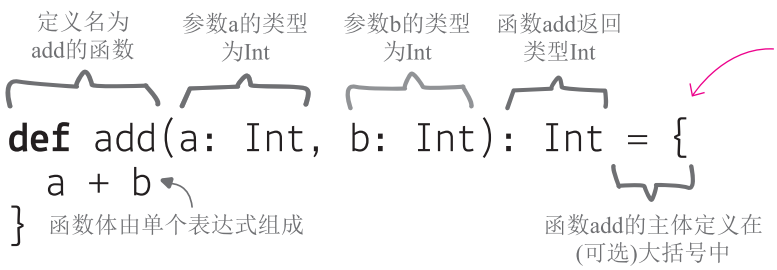
仍然会偶尔使用Java来呈现命令式的示例。打算仅在完全函数化的代码片段中使用Scala

用Scala编写的函数……

在本章的前面部分，你认识了第一个用Java编写的函数。该函数取两个整数参数，并返回它们的和。

```
public static int add(int a, int b) {
  return a + b;
}
```

现在，是时候用Scala重新编写这个函数，并学习一些新语法了，如图1-5所示。



Scala允许省略大括号(可选)。如果程序员不使用大括号，则编译器会认为缩进是有意义的，就像在Python中一样。如果你喜欢，可以使用此特性。但是，本书将包含大括号，因为希望节约花在语法差异上的时间，如前所述

图1-5 用Scala编写的函数

1.11 练习用Scala编写函数

既然了解了用Scala编写的函数的语法，可以尝试将之前的一些Java代码片段重写成Scala代码。希望这有助于知识过渡。

本书中的“练习……”部分是什么

本书中有三种练习。你已经遇到了其中两种：“快速练习”(标有大问号的小练习，可以很容易地在脑海中解决)和“小憩片刻”(所需时间更长、难度更大，旨在让你从不同的角度思考概念，需要使用纸张或计算机)。

第三种是“练习……”。这是三种练习中最枯燥的，因为它在很大程度上是重复性的。通常，你需要完成三到五个练习，其求解方式完全相同。之所以故意这样安排，是为了训练你的肌肉记忆。这部分内容将广泛应用于整本书，因此你需要尽快掌握。

还没有讨论需要在计算机上安装哪些工具以编写Scala代码，所以请在一张纸上完成这个任务

你的任务是使用Scala重写以下三个Java函数：

```
public static int increment(int x) {
  return x + 1;
}

public static char getFirstCharacter(String s) {
  return s.charAt(0);
}

public static int wordScore(String word)  {
  return word.length();
}
```

答案：

```
def increment(x: Int): Int = {
  x + 1
}

def getFirstCharacter(s: String): Char = {
  s.charAt(0)
}

def wordScore(word: String): Int = {
  word.length()
}
```

注意：

- Scala中的String与Java中的String具有完全相同的API
- Scala中的字符类型为Char
- Scala中的整数类型为Int
- 在Scala中不需要使用分号

1.12 准备工具

现在是时候开始在真实计算机上编写一些函数式Scala代码了。为此，需要安装一些工具。由于每个计算机系统都不同，请谨慎按照以下步骤操作。

如果你喜欢以自动化的方式安装JDK/Scala，或者你更喜欢使用Docker或Web界面，请务必访问本书的网站以了解本书中练习的其他编码方案

下载本书的配套源代码项目

本书中的每个代码片段也可在本书配套的Java/Scala项目中找到。可以通过扫描封底二维码或访问https://michalplachta.com/book进行下载或检查。该项目附带README文件，其中包含有关如何入门的最新详细信息。

安装Java开发工具包(Java Development Kit，JDK)

请确保你已在计算机上安装了JDK。这将使你能够运行Java和Scala(这是一种JVM语言)代码。如果你不确定是否已安装，请在终端中运行javac -version，你应该会得到类似于javac 17的结果。如果没有，请访问https://jdk.java.net/17/。

在本书编写之时，JDK 17是最新的长期支持(long-term-support，LTS)版本。其他LTS版本也应该可用

安装sbt(Scala构建工具)

sbt是Scala生态系统中使用的构建工具。它可用于创建、构建和测试项目。有关如何在你的平台上安装sbt的说明，参见https://www.scala-sbt.org/download.html。

运行它！

在你的shell中，你需要运行一个sbt console命令，该命令将启动Scala读取–求值–输出循环(read–evaluate–print loop，REPL)。这是在本书中运行示例和做练习的首选工具。你只要编写一行代码，按下Enter键，就能立即获得反馈。如果你在本书的源代码文件夹中运行此命令，则还将获得所有练习的访问权限，尤其是在本书的后半部分，练习变得更加复杂，这将会非常有用。不过，暂时先别急，先来练习一下REPL。请查看以下内容，直观地了解如何使用此工具。运行sbt console后：

```
Welcome to Scala 3.1.3
Type in expressions for evaluation. Or try :help.

scala>

scala> 20 + 19
val res0: Int = 39
```

使用的Scala版本

这是Scala提示符，在其中输入命令和代码。请继续编写一些数学表达式，然后按下Enter键

该表达式的计算结果为一个名为res0的值，其类型为Int，值为39

注意：
建议在本书中使用REPL(sbt console)，特别是在开始阶段，因为它不涉及函数体，不会让你分心。你可将所有练习直接载入你的REPL。但是，在熟悉练习的套路后，可以自由切换到IDE。最适合初学者的是IntelliJ IDEA。安装Java后，你可以从https://www.jetbrains.com/idea/下载此IDE

1.13 了解REPL

下面进行一个简略的REPL会话，顺便学习一些新的Scala技巧！

在此处输入代码，然后按下Enter键，立即执行它

```
scala> print("Hello, World!")
Hello, World!
```

REPL将输出打印到控制台

val是Scala关键字，用于定义常量值。注意，val是语言的一部分，而不是REPL命令

```
scala> val n = 20
val n: Int = 20
```

REPL创建一个Int类型的n，其值为20。在REPL会话的持续时间内，该值将在作用域中

可以引用以前定义的任何值

```
scala> n * 2 + 2
val res1: Int = 42
```

每当不为结果分配名称时，REPL都会生成一个名称。在本例中，res1是REPL创建的名称。它是Int类型，并且值为42

可以像引用任何其他值一样引用REPL生成的任何值

```
scala> res1 / 2
val res2: Int = 21
```

在这里，REPL生成了另一个名为res2的Int类型的值

只需要输入以前定义的名称以检查其值

```
scala> n
val res3: Int = 20
```

你可使用:load从本书的配套代码库加载任何Scala文件。此处加载第1章的代码。REPL显示加载的内容：你在上一个练习中编写的三个函数！请务必在文本编辑器中查看此文件来进行确认

```
scala> :load src/main/scala/ch01_IntroScala.scala
def increment(x: Int): Int
def getFirstCharacter(s: String): Char
def wordScore(word: String): Int
// defined object ch01_IntroScala
```

所有针对REPL本身(而不是代码)的命令都以:开头。使用:quit或:q退出REPL

```
scala> :quit
```

有用的REPL命令

:help 显示所有带有说明的命令
:reset 取消一切，重新开始
:quit 结束会话(退出REPL)

有用的键盘快捷键

使用上/下箭头循环浏览以前的项
使用Tab显示自动完成选项(如果有选项的话)

1.14 编写你的第一个函数

时机已到！你将要用Scala编写(并使用)第一个函数。你将使用已经熟悉的函数。

启动Scala REPL(sbt console)，并编写：

每当你在编写多行表达式时按下Enter键，|字符都会出现在REPL输出中

```
scala> def increment(x: Int): Int = {
     |   x + 1
     | }
def increment(x: Int): Int
```

如你所见，REPL响应了一行代码。这表明它理解输入的内容：名称为increment，该函数取类型为Int的参数x并返回一个Int。下面将使用该函数！

```
scala> increment(6)
val res0: Int = 7
```

通过将6作为参数，调用函数。该函数按预期返回了7！此外，REPL将此值命名为res0。

使用本书的代码片段

为了尽可能使代码清单易于阅读，将不再在本书中打印REPL提示符scala>。也不会再打印REPL的详细响应。上面的示例是你应在REPL会话中执行的操作。但在本书中，仅会打印：

```
def increment(x: Int): Int = {
  x + 1
}

increment(6)
→ 7
```

如你所见，使用→表示来自REPL的答案。它的意思是，“输入以上代码并按下Enter键后，REPL应该响应值7。”

> 将使用此图形表示你应该尝试在自己的REPL会话中编写代码

现在，尝试编写并调用之前遇到的另一个函数：

```
> def wordScore(word: String): Int = {
    word.length()
  }

  wordScore("Scala")
  → 5
```

同样，左侧片段在你的REPL中应该显示为右侧的形式

```
scala> wordScore("Scala")
val res1: Int = 5
```

1.15 如何使用本书

在结束本章之前，向你介绍使用本书的方法。记住，这是一本技术性书籍，所以不要指望一口气从头读到尾。相反，将书放在你的桌子上，以便你使用计算机或者在纸上写代码的时候查阅。转换视角，从被动的知识接收者转变为积极主动的参与者。以下是一些补充建议。

做练习

确保做好每一个练习。不要复制和粘贴代码，也不要盲目地把代码从书中转移到REPL。

不要提前查答案，对于“小憩片刻”练习，尤其如此。查答案以快速解决练习可能会让你有一时快感，但会影响长期学习效果

快速练习、小憩片刻和练习……

本书中有三种类型的练习：

- “快速练习”是不必借助任何外部工具，可以在脑海中完成的小练习。
- “小憩片刻”所需时间较长，难度较大，旨在让你从不同角度思考一个概念。这通常需要使用纸张或计算机。
- “练习……”主要基于重复。这种练习用来训练你对书中重要概念和技巧的肌肉记忆。

创建一个学习环境

在手边放一些纸和几支不同颜色的铅笔或钢笔。记号笔也可以。希望你的工作环境充满信息——而不是沉闷、枯燥的。

不要匆忙

以舒适的节奏工作。即使没有持续、稳定的节奏，也没关系。有时奔跑，有时爬行。有时什么也不做。休息非常重要。记住，有些主题可能较难。

但如果感觉很容易，就没有收获

尽量多编写代码

本书有成百上千的代码片段，你可以直接将它们转入你的REPL会话中。每一章都是按照“REPL方式”编写的，但是鼓励你玩转代码，编写自己的版本，尽情享受其中的乐趣！

记住，你在本书中遇到的所有代码都可以在本书的配套源代码仓库中找到

小结

在本章中，你学习了五个非常重要的技能和概念，这是本书后续内容的基础。

本书的读者对象

首先定义你——读者。你选择本书的原因主要有三个：也许你只是对函数式编程(FP)感兴趣；也许你以前没有足够的时间或机会全面学习FP；也许你之前学过它，但并不喜欢它。无论原因如何，本书的读者都应是希望通过实验和游戏学习一些创建实际应用的新方案的程序员。读者应熟悉面向对象语言(如Java)。

函数是什么

然后，介绍本书的主角——函数，并探讨特征标记和函数体。本章还提及了当特征标记没有说明函数体的全部情况时会遇到的问题，以及这为何会加大编程难度。

> 在整本书中，REPL会话都用这个图标标记。在开始新章节之前，记得重置(:reset)你的会话

函数式编程的作用

讨论命令式编程和声明式编程之间的区别，大致定义什么是函数式编程，以及它对你成长为软件专家有什么帮助。

如何安装所需工具

安装sbt，并使用Scala REPL编写第一个函数。学习书中的代码片段在REPL中的工作方式，以及如何使用→来表示代码片段中的REPL响应。

代码：CH01_*
通过查看本书代码库中的ch01_*文件，探索本章的源代码

如何使用本书

最后，介绍本书的所有辅助学习功能。描述三种类型的练习(快速练习、小憩片刻和练习……)，讨论如何准备你的工作空间以促进学习，并描述如何处理代码片段。你可以将它们复制、粘贴到REPL会话中，手动传输它们，或者从本书的仓库附带的Scala文件中对它们进行:load操作。请扫描封底二维码或访问https://michalplachta.com/book以获取源代码。其中还有一个README文件，可以帮你完成设置。