目录

1. 初识 Java

Java 是一门蓝领语言。它不是博士论文材料，而是用于工作的语言。Java 对各行各业的程序员来说都似曾相识，因为我们更喜欢经过验证的东西。[1]

——Java 之父 James Gosling

1.1 Java 简介

Java 现在取得了巨大的成功，这往往被看作是必然的。有人把 Java 的成功归结为赶上了互联网的浪潮，但人们往往会忽略这样一个事实：在 Java 出生的那个年代，互联网能否成功尚不明确。

印度诗人泰戈尔曾经写过：“有时候爱情不是因为看到了才相信，而是因为相信才看得到。”如果把其中的爱情，换成科技或者计算机编程语言，我认为同样成立。

在 1995 年，Java 的创始人 James Gosling 写了两份文档，在文档中，他描述了当时编程界存在的问题，这些问题会在接下来的 Java 语言的特点那一小节中详细讲述，当时的编程语言并不能解决这些问题，为了解决这些问题，他创造了 Java 语言。其创造的原因就如 ALGOL 语言的设计者艾伦·佩利所言：“在设计程序的过程中，总有一些我们希望表达的东西，而已知的语言都没有做好”。在创造 Java 的过程中，James Gosling 如同泰戈尔的诗里写的一样，他相信他提出的解决方案，并且对编程语言是否应该加入某种特性有强烈的爱憎。[2]

限于篇幅，本书无意详细介绍 Java 的发展历史，只能管中窥豹，讲几个小故事，让大家了解一下 Java 早期的状况。

1.1.1 Java 初次登场

[1] 原话是：Java is a blue-collar language. It’s not PhD thesis material but a language for a job. Java feels very familiar to many different programmers because we preferred tried-and-tested things。这句话来自于 1997 年 James Gosling 写的一篇文章《The Feel of Java》，原文我已经放在本书辅助材料 resources/chapter1 文件夹中。

[2] 在一次访谈中，James 提到他设计 Java 的艰辛，访谈的名字叫《An Interview With "Dr. Java" James Gosling, The Creator Of Java》

1995 年 2 月，John Gage[3]受邀参加一场 TED 会议。他受邀的演讲题目是《科技对教育的作用日益显著》,在演讲的结尾，他想用一个演示软件来展示目前的高科技。为了此次演讲，他花了一个多月来准备这个演示软件。[4]

John Gage 演示的软件是用 Sun 公司尚未透露的编程语言写的,该语言的负责人是 James Gosling。因为事关重大，James Gosling 不放心让 John Gage 独自演示，于是，两人决定合作完成，演讲的部分由 John Gage 来做，展示的部分由 James Gosling 来做。

在演讲的结尾，James Gosling 出现在台上，他打开一个网页，上面有一个三维的红色分子模型，与当时人们常见的图片不同，这个分子模型可以随着 James Gosling 的鼠标放大缩小，旋转跳跃，这种新颖的操作方式，在 1995 年让在座的科技精英都赞叹不已。

演示所用的技术就是本书的主题：Java 编程语言。演示人 James Gosling 被人称为 Java 语言之父。

1.1.2 指导 Java 发展的文档

Java 之父 James Gosling 的故事想必大家已经耳熟能详，如果不太熟悉的话，推荐到计算机历史网站 Computer History Museum 上观看访谈“James Gosling Oral History”。在这份长达近七小时的访谈中，James Gosling 谈了他本人和 Java 的方方面面。

James Gosling

出生于 1955 年，加拿大人，Java 语言创始人。获美国卡内基梅隆大学计算机科学博士学位，1984 年加入 Sun 公司，在此，他研发了 Java 编程语言。

1995 年 2 月，James Gosling 发布了一篇只有 9 页的文档来介绍 Java，这篇文档的名字叫《Java: An Overview》。[5]文档简明扼要的指出了当时计算机界存在的问题，并且给出了 Java 的解决方案。

1995 年 10 月，James Gosling 和 Henry McGilton 发布了 Java 第一版白皮书：《The Java

[3] 关于 John Gage 的介绍，在 1996 年 wired 网站上有一篇名为《Power to the People》的文章专门介绍他。

[4] 由于年代久远，在 1995 年拍摄的视频，其清晰度实在是不够理想，如果大家想看当年演示的画面，请搜索名为《What Were You Doing In 1995? A 20 Year Retrospective of Java》的视频，演示的画面出现的位置大约在 25 分 30 秒。

[5] 这篇名为《Java: an Overview》的文档在本书资料中，详见 chapter 1 文件夹。

Language Environment》。[6]这篇 86 页的文档详细的介绍了 Java 的技术细节。

虽然后来 Java 发布过多份技术白皮书，但我认为这两份白皮书是所有白皮书中最重要的两份。在 Java 尚未被人所知的年代，James Gosling 写的这两份文档，如同明灯一样，指明了 Java 前进的方向。

现在 30 年快过去了，在计算机领域，30 年让计算机软硬件有了长足的发展，让软件工程理论也有了深厚的沉淀，但是这两份文档对学习 Java 仍然非常有指导意义。经过了长时间的洗礼，Java 的优点与缺陷越来越清晰，这两份文档中所提到的问题，也是计算机编程领域一直要解决的问题：**如何管理编程的复杂度**。

编程要掌控的问题是复杂度，而复杂度包含三个方面：项目本身的复杂度，编程语言的复杂度和机器的复杂度。作为程序员，我们一直以来面临的问题都是如何有效的处理复杂度，随着编程语言和编程理论的完善，编程语言方面的复杂度有所缓解，但是机器的复杂度却有了大幅的提升，在以前，程序员可没必要考虑这么多五花八门的设备，家里摆的，包里装的，手里拿的，身上穿的……

这两份文档我读过很多次，坦白来讲，在技术领域，能指出问题的人不少，但是能做到抓住问题核心的同时又能提出有效解决方案的人则凤毛麟角。James Gosling 做到了，从 1995 年到今天，Java 的份额一路攀升到名列前茅，我想，在编程语言领域，最好的赞美是被业界认可，被程序员使用。有句话是这样说的：只有两种编程语言，一种是怨声载道，一种是无人问津。Java 显然不属于后者。

James Gosling 在文档中提到了 Java 产生的来源是对 C++的不满。起初，遇到的问题尚可通过修改编译器来解决，但随着问题的积累，创造一门新的语言成了最优解，以解决项目中遇到的易用性、健壮性、安全性、跨平台性等问题。以史为鉴，让我们看看以前 James Gosling 碰到了哪些难题，他又是如何解决的。在解决这些难题的过程中，Java 形成了自己的特点。

1.1.3 Java 语言的特点

和其它编程语言一样，Java 的发展史也是一个找到痛点并解决的历史。在软件发展史上，这种情况屡见不鲜，甚至有了专属自己词汇的描述：“dogfooding”。[7]起初，James Gosling 并没打算开发一门新的编程语言。

---

[6] Java 第一版的技术白皮书《The Java Language Environment》也在本书资料中，详见 chapter 1 文件夹。

[7] Dogfooding 这个词语来自于“Eating your own dog food”，可以翻译为“吃自己的狗粮”。在 IT 业界这句俚语可能最早是于 1988 年开始使用的。当时微软公司的高级主管保罗·马瑞兹曾写过一封题为“Eating our own Dogfood”（吃我们自家的狗粮）的邮件，在邮件中他向微软局域网管理工具项目的测试主管布莱恩·瓦伦蒂尼提出“提高内部使用自家产品比重”的挑战。而从此以后，这一俚语在公司内就传播开来了。

他参与了一个名为 Green 的项目，这个项目的目标是能让“包括 VCR、电话、游戏机、手机、洗碗机等在内的”消费电子产品能够“透明地交互操作”。由于设备类型太多，Gosling 的团队碰到了一个棘手的问题，用 C++语言支持如此众多的设备，起初可以通过修改 C++编译器来解决，随着设备越来越多，最后成了一个“灾难”。于是，James Gosling 意识到，是时候做一个新的编程语言了。

在上世纪 90 年代，大部分程序员只能在 C 和 C++语言中选择。当时的技术水平与用户需求下，每家厂商都有自己独特的设备，有互不兼容的操作系统。随着设备的增多，设备之间互通的需求逐渐浮现，再加上互联网的兴起，对设备的要求从单机进化到另一个维度：联网。

当时可供选择的编程语言都没有为互联网的爆发提供足够的支持。在非网络化的情况下，各种设备和程序相互独立，但是一旦接触到互联网，这些相互独立的设备就要面对未知的环境，原本无关紧要的程序漏洞，在互联网环境下可能会变成一个灾难。如果能有一门编程语言在满足同时支持多种设备的情况下，又能使这些设备互相联网就好了，在这种需求的召唤下，Java 应运而生。

如何能在减轻程序员负担的同时，还能提高开发软件的质量，是 Java 自始至终一直在解决的问题。那 Java 是如何做到的呢？

1. 模仿 C/C++熟悉的语法

Java 的设计者认为，要避免程序员花大量的时间来学习一门风格迥异的语言，最直接的方法莫过于借鉴 C/C++的语法，毕竟 C/C++是当时最流行的语言。这个策略被证明是非常有效的，大量的 C++程序员甚至不需要重新学习，就能用自己 C++的经验写 Java 程序。

Java 发展的过程并非一帆风顺，在困难的时期，整个项目差点被取消。更不要说 Java 语言某些具体的语言特性了，项目组在是否模仿 C/C++语言特征上产生了诸多分歧，幸亏 Sun 公司的创始人 Bill Joy 对 James Gosling 提供了巨大的支持，Bill Joy 支持 Java 模仿 C/C++的语法。

Bill Joy

出生于 1954 年，加州大学伯克利分校硕士，在校期间和朋友一起创办了 Sun 公司，是公司的联合创始人，在校期间，他是 BSD 系统的主要设计者，同时还是 vi 编辑器，C Shell 的作者。他是 Java 创始人 James Gosling 的上司兼好友，Java 项目曾经几乎被取消，都是 Bill Joy 力排众议，持续支持该项目，最终获得成功。

为了减轻程序员的负担，在语法上，Java 继承并简化了 C++，舍弃了一些诸如多重继承，操作符重载等比较复杂的特性。Java 还增加内存回收的功能，把程序员从原先繁琐的内

存回收中拯救出来，让 Java 自动完成内存的回收。Michael Feldman 曾经这样评价 Java：“Java 从很多方面来说，就是简化版的 C++。”[8]

Java 集百家之长成一家之言。在文档中，Java 从不隐瞒其善于“借鉴”的特点，经常对其它语言的优秀特性大加赞扬，会直言从哪种语言中获得灵感，比如在动态性方面，就从 Objective-C 借鉴了很多想法。在本书中，我会尽量对 Java 语言的特性一一考证，知其根本。唐代的魏征说过这样一句话：“求木之长者，必固其根本；欲流之远者，必浚其泉源。”我认为，对编程语言的历史考证的越多，了解的就越多，知其然又能知其所以然。

2. 强大的可移植性

取得成功的编程语言，一定要兼顾这三个方面：运行速度、可移植性与安全性。同时满足运行速度与可移植性的语言就很少，更不要说还要兼顾安全性了。一般来说，运行速度快的语言，可移植性不高，可移植性高的语言，又难以做到兼容性。

我觉得读本书的读者，肯定知道有一门编程语言叫 C 语言，C 语言的重要特征之一就是有强大的可移植性。正是凭借着这种强大的可移植性，用 C 语言写的 Unix/Linux 操作系统以迅雷不及掩耳的速度攻城略地，但是这种“可移植性”是建立在对每一种机器要修改的基础上。以常见的整数为例，由于 C 语言的标准没有规定整数可以表示的大小，而是依赖不同的硬件平台，比如有的机器是 16 位的，有的机器是 32 位，有的则是 64 位的，那么可能这三种平台上的整数的大小是不同的，需要针对不同的平台做相应的修改，才能做到良好的可移植性。[9]

Java 的出现，真正实现了可移植性。Java 的可移植性是指 Java 程序可以在不同的操作系统和硬件平台上运行，而无需对代码进行修改。还是以整数为例，在 Java 里，所有整数的大小都是相同的，不会随着平台的变化而变化。有关整数的内容，我会在第 4 章详细讲解。Java 之所以能做到“真正的”可移植性，是因为 Java 提供了一种叫做 Java 虚拟机（JVM，Java Virtual Machine）的机制。

Java 虚拟机的内容会在本书第 2 章详细介绍，目前只给出一个粗略的定义：**Java 虚拟机是一个抽象的计算机，它为 Java 程序提供了一个跨平台统一的运行环境。**Java 虚拟机的规范定义了一组明确的指令集和数据类型，这保证了不同平台上的 JVM 实现上是一致的。这意味着只要某个平台有一个 JVM 实现，Java 程序就可以在这个平台上运行。因此，Java

[8] 他评价的原文是：Java is, in many ways，C++-- 。

[9] 在网上有一本神书叫《Unix 痛恨者手册》，是一群 Unix 的讨厌者写的书，在书中，吐槽了很多 C 语言和 Unix 的缺点，包括可移植性，每当说到可移植性，我就想到这本书。书中吐槽 Unix 就是个病毒，给出的理由是 Unix 和病毒的共同特征为：体积小，可传染多种宿主（可移植），变异快速等等。《Unix 痛恨者手册》写完以后，请 C 语言与 Unix 系统的开发者丹尼斯·里奇写序言，丹尼斯·里奇也不客气，痛骂了那本书，那本书的作者把痛骂自己的回信原封不动的当了书的序言。网上有那本书的中文翻译版，如果你找不到，可以到我的个人网站上去下载。

程序不需要考虑底层硬件和操作系统的差异，只需要依赖 JVM 进行解释和执行。

同时我们也要正视 JVM 不可避免的缺点，JVM 曾经被广为诟病的缺点是相比于 C 语言写的软件，性能上有所损失。为了弥补这个缺点，Java 进行了很多创新，这些创新就是接下来要讲 Java 语言的第 3 个特点。

3. 采用解释执行和编译执行相结合的方式运行

从技术角度来说，编程语言可以分为编译型语言和解释型语言。编译型语言将源代码直接编译成 CPU 可执行的机器代码，机器代码是计算机硬件可以理解的语言。常见的编译型语言的包括 C/C++。与编译型语言不同，解释型语言由解释器执行，而不是编译为机器代码由 CPU 直接执行。解释型语言在运行程序的时候，逐行读取源代码，由解释器来执行，这个过程重复进行，直到运行完所有的源代码。常见的解释型语言包括 Python、JavaScript 和 Ruby。

从 Java 第一本白皮书发布的时候算起，快 30 年了，Java 在解释执行方面有了长足的进步，现在 Java 的运行机制已经完全不是 James Gosling 在 1995 年规划的那样，当年 Java 1.0 发布的时候，是一个纯解释型语言。由于现代解释器有了长足的进步，尤其是即时编译器（JIT）的大量应用，解释型语言和编译型语言之间的界限已经像当年那样清晰，如今的 Java 已兼具编译型和解释型语言的特点。

至于 Java 的性能问题，在 1995 年的那份文档里，James Gosling 给出了一个测试数据，在一台 SS10 计算机上，用解释器运行 Java，每秒钟可以调用 30 万次方法，这个数据和 C/C++ 已经没有明显的差距。Java 虚拟机进化了多年，现在人们已经对 Java 的性能没有太多质疑，有很多的测评数据显示，Java 是运行最快的语言之一。

举一个例子，著名的游戏公司 ID Software 开源了其第一人称射击游戏 Quake2 的源代码，这些源代码是用 Java 写的。[10]如果 Java 的性能足够满足第一人称射击游戏的要求，那么肯定可以满足绝大部分商业程序的要求。

4. 反射机制

在第一章就引入“反射机制”这个陌生的名词，对初学者实在是不太友好。我在第 3 章里会讲到面向对象编程为什么这么难，其中一个重要的原因就是“名词搅拌器”，编程界经常引入一些让人丈二和尚摸不着头脑的专业术语。但是对 Java 来说，反射实在是太重要了，没有反射机制的 Java 会让很多功能无法实现，从而走向没落。

我先讲个 Java 历史中发生的故事，前面我提到过 Java 强大的可移植性，只要在某个平

[10] Quake 2 的源代码可以在 github 网站上找到，有兴趣的读者可以搜索一下，关键字为 jake2

台实现了 Java 虚拟机，那么 Java 代码就可以在这个平台运行。这并不是特别的准确，当年 Java 曾经为小型设备(如移动电话、个人数字助理和电视等)提供过一个叫 J2ME 的虚拟机，由于小型设备的性能不强,J2ME 上不支持反射。这导致大量使用反射机制的功能不能使用，比如自动测试就无法在这个平台上运行，当年我记得铺天盖地的吐槽这个缺陷，最后该平台慢慢的消亡了。

还是以自动测试为例,所谓的自动,就是人和电脑之间有一种约定,人做有创意的事情，电脑做繁琐的事情。测试的时候，如果事无巨细，都要告诉电脑，那不叫自动测试，那更应该叫“人动测试”。在 Java 之前，比如 C++测试的时候，测试人员要告诉测试框架要测哪几个方法，怎么测，比较繁琐。在 Java 1.2 引入反射机制之后，测试框架会自动测试某个类的所有方法，而且还可以按照测试人员的要求只测试某类方法，比如只测以“test”开头的方法，从而实现更高级的自动化。

自动测试只是 Java 使用反射机制的一个常见的例子，还有更多的例子不适合放在一本书的开头，只是反射机制太过重要了，如果不提一下，就不能展现出 Java 的精华。

1.2 Java 语言进化史

上面简要的介绍了 Java 两份最重要的文档,这两份文档可以算作 Java 语言的战略纲领，本书在写作的过程中，参考了这两份文档以及后续发表的 Java 白皮书。

有了战略纲领，若想在编程语言的竞争中拔得头筹，还必须要赢下一场一场的战役，这些战役，就是 Java 的每一个发布的版本。接下来，我们再来看看这 20 多年，Java 是如何一步一个脚印的走到今天的位置上的。

除了了解 Java 的历史，知道每个版本有什么样的功能也有现实意义，我们在写 Java 代码的过程中，有时候难以知道我们可以用哪些特性，尤其是对代码所使用的 Java 版本不熟悉的时候。比如我们有一段用了 lambda 功能的代码，那么这段代码在 Java 8 以前的虚拟机上是没法运行的，因为 lambda 是 Java 8 才引入的新特性。

对初学者来说，不用太关注不同 Java 版本的要点。我喜欢读历史，想起当年读《史记》的时候，老师会说，读《史记》先从列传读起，再读世家，最后读表，只有你读懂了表，才说明你《史记》入门了。最初我不以为然，因为《史记》中的表太枯燥了，很多版本的《史记》都直接删掉表这一部分。直到后来很多年，我才意识到表的意义是统领全局，是纲举目张的那个纲，怪不得吕世浩教授说司马迁曾经想把表放在《史记》的最前面，最后不知道什么原因妥协了。

我工作的这些年，经常要开发、维护、部署不同年代的 Java 软件，每当要开展一个项目的时候，我脑海中第一反应就是哪个版本的 Java？要知道，在传统企业里，软件运行在 Java 8 甚至 Java 5 上（我见过的最低版本的 Java 是某个钢铁厂的软件，运行的版本是 Java 1.3），只要软件还能运行，企业领导就没有更新到最新版本的动力。下面要讲的 Java 版本的要点，对我来说就是一张《史记》中的表，提醒我写代码的时候要适应运行环境，小心的避

开不兼容的技术，而不是制造更多的麻烦。

1.2.1 Java 版本要点

- **Java 1.0**: 该版本发布于 1996 年 1 月 23 日，有 212 个类，封装在 8 个包中。这个版本的 Java 速度非常的慢。正是这个版本，宣告了 Java 的诞生，打响了自己的名号：“write-once, run-anywhere” 。[11] 这个宣传口号译成中文是“写一次代码，到处运行”。如果大家能找到用 Java 1.0 写的代码，不用做任何修改，还可以运行在今天的 Java 11 之中。在这个版本中，JDK 1.0 提供了一个纯解释执行的 Java 虚拟机，这个虚拟机的名字叫 Sun Classic VM，这个虚拟机很慢，导致后来 Java 一直被人认为很慢，实际上后来 Java 越来越快了。

> **讽刺 Java 慢的段子**
>
> Java，C，C++，汇编……在一艘船上，船漏水了，为了保证其它人的安全，决定扔两个人下去。规则是讲笑话，只要有一个人不笑，就把讲笑话的人扔下去。
>
> 汇编是讲笑话的高手,他讲了一个笑话,把其它人都笑弯了腰,只有 Java 没笑，按规则，汇编被扔了下去。第二个轮到 C 语言讲了，C 语言还没开口，Java 就笑弯了腰，众人不解，问 Java 笑什么？
>
> Java 回答： “刚才汇编的笑话太好笑了!”

- **Java 1.1**：该版本发布于 1997 年 2 月 19 日，有 477 个类。在这个版本中，第一次引入了反射，一些基础的 Java 技术也是在这个版本中被引入的，比如 JDBC，JAR 文件格式等，并且提供了 GUI 的代码。由于微软和 Sun 公司之间竞争的原因，Java 1.1 曾经长期存在于微软的浏览器中，微软不对其进行更新，而为了兼容，当时的 Netscape 也只能内置 Java 1.1。
- **Java 1.2**：该版本发布于 1998 年 12 月 4 日，有 1524 个类。在这个版本里，引入了诸如 sets，maps 和 lists 这样的容器，将 Swing 集成进来。从该版本开始，Sun 公司将 Java 分拆成三个不同的方向，分别是面向手机的 J2ME，面向桌面的 J2SE 和面向企业的 J2EE。这个版本，也被营销部门称之为 Java 2，这种叫法，导致不少用户混淆了 Java 的版本号和发行号。
- **Java 1.3**：该版本发布于 2000 年 5 月 8 日，有 1840 个类。在该版本中，引入了 JNDI 和处理声音的 javax.sound 库。从该版本开始，HotSpot VM 成了默认的虚拟机，由于 HotSpot 的采用，该版本的 Java 性能有了极大的提升。

[11] 第一次提出这个口号是 1995 年 5 月 23 日，Sun 公司举办第一届 Sun World 大会的时候，当时发布了 Java 1.0 版。

- **Java 1.4：**该版本发布于 2002 年 2 月 6 日，有 2723 个类。在该版本里，引入了断言 assert 这个关键字，增加了对正则表达式的支持，引入了日志功能，增加了对图片处理的库，增加了 XML 解析器和 XSLT 转换器，一些低级的 I/O 操作，对 SSL 的支持等功能。经过了几年的发展，Java 日渐成熟，大批公司采用了 Java，像 IBM，Symbian，SAS，Compaq 这些公司，都参与甚至发布了自己的 JDK。同年，微软公司发布了自己的.NET Framework 来与 Java 正面竞争。

- **Java 5：**该版本发布于 2004 年 9 月 30 日，有 3279 个类。在该版本里，引入了类型安全的枚举类、泛型类、静态导入、还引入了 “for each” 语法。在这个版本里，Java 开始完善并行编程。

> **Java 5 为什么不叫 Java 1.5 了？**
>
> 主要原因是营销。
>
> 技术部门一直按照 1.x（这里的 x 按照阿拉伯数学递增），这种叫开发号，但是营销部门并不这样，而是直接用 Java 2， Java 3……这种叫发行号。开发号与发行号就有了很大的差异，让用户很纠结。因此在 Java 1.5 的时候，直接都用发行号了，Java 1.5 和 Java 5 是一个版本，两个名字。

- **Java 6：**该版本发布于 2006 年 12 月 11 日，有 3739 个类。该版本扩展了 Java 1.2 的 SortedMap 和 SortedSet 接口，引入了 NavigableMap 和 NavigableSet 接口。该版本非常有纪念意义，这是 Sun 公司出品的最后一个版本的 Java，随后 Java 进入了 5 年的断档期。

> **Java 开源大事件**
>
> 2006 年，对 Sun 公司和 Java 来说都是非常重要的一年。在这一年，公司的创始人兼 CEO Scott McNealy 因为公司经营不善离开了公司，新上任的 CEO 是非常会写博客的 Jonathan Schwartz。
>
> 他上任以后，在 2006 年 11 月 13 日的 JavaOne 大会上，宣布开源 Java，随后，Java 在 GPL v2 协议下开源，Java 由新成立的组织 OpenJDK 进行管理。
>
> 与此同时，Sun 再次对 Java 进行了重新规划，当年的 J2EE、J2SE 和 J2ME 分别 改名为 Java EE、Java SE 和 Java ME。

- **Java 7：**该版本发布于 2011 年 7 月 28 日，共有 4024 个类。在这个版本里，switch 语句中可以使用 String 类型了，还增加了 ARM 块（也称自动资源管理）。在这个版本里，Oracle 开始完全支持 Mac OS X 系统，并对 ARM 指令提供了支持。至此，官方的 JDK 才真正支持 Windows，Linux，Solaris，Mac OS 平台。 Java 6 是 Sun 最后一个版本的 Java，Java 7 是 Oracle 接手后的第一个版本的 Java，颇有一点薪火相传的意味。

如果大家从事 Android 开发，Android Studio 3.0 或者更高版本可以支持 Java 7 的所

有特性，但不保证支持 Java 8 的所有特性。

- **Java 8(LTS)：**[12]该版本发布于 2014 年 3 月 1 日，共有 4240 个类。在这个版本里，增加了 lambda 表达式、方法引用、重新设计了与时间相关的 API、Optional 类。在这个版本里，还移除了 PermGen，取而代之的是 MetaSpace 这种新的内存空间。

- **Java 9：**该版本发布于 2017 年 9 月 21 日，共有 6005 个类。Java 8 发布 3 年半后，Java 9 明显又跳票了，这个版本最大的变化是引入了 Java 平台模块系统。使用该特性，Java 应用可以创建出只包含所依赖的 JDK 模块的自定义运行时镜像。这样可以极大的减少 Java 运行时环境的大小。在这个版本中，还增加了 ProcessHandle 接口，让 Java 可以对原生进程进行管理。同时，还引入了 jshell 的功能，这个功能在学习 Java 的时候非常有用，可以使用该功能快速的尝试 Java 的新功能。在这个版本中，Java 还移除了 JavaDB。

Java 10 改发布周期了

现在越来越多的软件采用短期和长期 LST（Long-term support）相结合的方式来更新，比如备受欢迎的 Ubuntu 操作系统就是如此，短期更新的版本是半年，Long-term Support 的更新频率为 18 个月，也就是一年半。

使用这种方式能确保每年都有两次发布。短期版更新频繁，但是支持时间短；长期版更新不频繁，但是支持时间长。当然，Oracle 和 Ubuntu 的发布周期有所区别，大家可以搜索“Oracle Java SE Support Roadmap”，在官方网站上可以查找到详细的发布日期，目前，Java 8 的付费支持可以到 2030 年。

Java 采用这种方式以后，稳定版的名字后面会加 LTS 表示是长期支持版，以目前的规则，Java 8、Java 11 和才发布的 Java 17 都将会是 LTS 版本。

Java 的首席架构师 Mark Reinhold 宣布，Java 以后的版本不会以特性多少来确定版本，而是按照时间来发布，每半年发布一个新的版本。最终选择的命名机制是这样的：

$FEATURE.$INTERIM.$UPDATE.$PATCH

其中我们最需要关心的是$FEATURE，每次发布版本就加 1，为了兼容以前的版本，从 10 开始，比如 2018 年 3 月份是 10，2018 年 9 月份是 11，2019 年 3 月份是 12， 2019 年 9 月份是 13……以此类推。

[12] 对 Oracle 来说，发行 LTS 版本一个更重要的原因可能是为了收费。Oracle 的官网上有一份名为 Oracle Java SE Support Roadmap 的介绍，Java 7 和 Java 8 都提供长期的支持，Java 8 的付费支持到 2030 年，比 2018 年发布的 Java 11 还要久，Java 11 只支持到 2023 年。

- **Java 10：**该版本发布于 2018 年 3 月 20 日，包含 6002 个类。这是 Java 改变发布周期后的第一个版本，这个版本里，增加了局部变量类型推断、并行全垃圾回收器 G1、增加了一种全新的实验性的 JIT 编译器 Graal。

- **Java 11(LTS)：**该版本发布于 2018 年 9 月 25 日，包含 4411 个类。这是新发布周期下第一个长期支持版本。相比于上一个版本，这个版本里的类大幅减少，主要移除了 JavaFX，JavaEE 和 CORBA 模块。如果你的软件依赖这些模块，可以单独安装，除了 CORBA 模块以外，移除的内容都有单独的安装方法。该版本还对标准的 HTTP Client 进行了升级，使之可以完全支持异步非阻塞。另外还引入了两个新属性，一个是 NestMembers 属性，一个是 NestHost 属性。更新了 TLS 1.3 协议。

- **Java 12：**该版本发布于 2019 年 3 月 19 日，包含 4433 个类。该版本引入了一个实验性质的垃圾收集器 Shenandoah。[13]再次改进 Switch 语句，省去了 break 语句，未来某个版本的 switch 肯定将会支持 float、double 和 long。

- **Java 13：**该版本发布于 2019 年 9 月 10 日，包含 4400 个类。该版本继续实验 ZGC 垃圾回收特性、继续探索改进 switch 语句。除此之外，还对 Java 中存在了 20 多年的 Socket API 进行了重构。该版本对 Socket API 带来了新的实现方法，并且默认使用新的 Socket 实现，使其易于使用并提高可维护性。

- **Java 14：**该版本发布于 2020 年 3 月 17 日，该版本将 Java 12 和 Java 13 中的 switch 语句发为正式版。同时引入了记录类，还正式移除了 Pack200 及其 API。

- **Java 15：**该版本发布于 2020 年 9 月 15 日，在该版本中，G1 仍然作为垃圾回收器的默认选择，但是从 Java 12 和 Java 13 一直实验的 ZGC 和 Shenandoah 被正式发布，只需要进行相关的配置就可以选择使用。

- **Java 16：**该版本发布于 2021 年 3 月 16 日，该版本最有趣的两个更新是封闭类（sealed classes）和增强了 Java 14 引入的记录类。同时，Java 16 还正式发布了一个名为 jpackage 的打包工具。

- **Java 17:** 该版本发布于 2021 年 9 月 14 日，是 Oracle 公司发布的最新的长期支持版本的 Java，下个长期支持版本将会是 2023 年 9 月的 Java 21。相比来说，该版本的变化不大，修复了一个长期存在于浮点类型中的 bug，甚至连 Java 的创始人 James Gosling 还专门发文庆祝告别了 strictfp。同时，还增加了对 macOS AArch64 的支持。

  Java LTS 版本的发布周期发生变化，改为每两年发布一次。

  在发布 Java 17 的同时，Java 平台组开发副总裁 Georges Saab 宣布：“在过去的三年里，许多开发人员都很喜欢这些新功能，我们看到生态系统真正适应了

[13] 这个垃圾收集器是 Red Hat 公司从 2014 年就开始研发的，主要解决 JVM 上内存回收效率的问题。Red Hat 对外宣称，该垃圾回收器的暂停时间与堆大小无关。

> 每六个月一次的发布节奏。Java 开发人员目前面临的一大挑战是，他们的组织只允许使用最新的 LTS 版本。现在，LTS 版本将改为每两年发布一次，组织较为保守的开发人员也可以选择和访问他们喜欢和想要使用的功能”。

1.2.2 版本总结

前面罗列了每个 Java 版本的要点和发行时间，我们可以看到，Java 对每一个功能都做了很多次尝试，有些功能如内存回收，HTTP 和 Socket 反复实现改进了几次，这里的 HTTP 和 Socket 是与网络相关的接口，我来解释一下为什么反复实现改进了几次，将来大家工作的时候可能也会碰到类似的情况。这些情况在我的工作生涯中遇到过多次，虽然不难，但是没人帮助的话，还是很难找到头绪的。

Java 是一种跨平台的编程语言，但是网络肯定要依赖其宿主才能实现。在不同的操作系统上网络的实现有些许差异，比如最大并发连接数、最大套接字数等均有差异，甚至某些特定操作系统的底层网络特征在 Java 的标准库中无法使用，要借助第三方库方可实现。如果你写的代码在 Windows 或者不同版本的 Linux 上运行时，无论性能还是运行状态在网络方面有些差异，一定不要着急先从代码上着手改进，先看看运行平台的特征和限制。

下图我列举了部分 Java 发行版中类的数量和发布日期，类的数量稳步增加的时候，对程序员基本没什么影响，但是当类的数量锐减的时候，比如 Java 10 到 Java 11 就减少了 1600 个类，程序员需要确认减少的类会不会影响到软件的运行。

Java版本	发布日期	类的数量
Java 1.0	1996年1月23日	212
Java 1.1	1997年2月19日	477
Java 1.2	1998年12月4日	1524
Java 1.3	2000年5月8日	1840
Java 1.4	2002年2月6日	2723
Java 5	2004年9月30日	3279
Java 6	2006年12月11日	3793
Java 7	2011年7月28日	4024
Java 8	2014年3月1日	4240
Java 9	2017年9月21日	6005
Java 10	2018年3月20日	6002
Java 11	2018年9月25日	4411
Java 12	2019年3月19日	4433
Java 13	2019年9月10日	4400

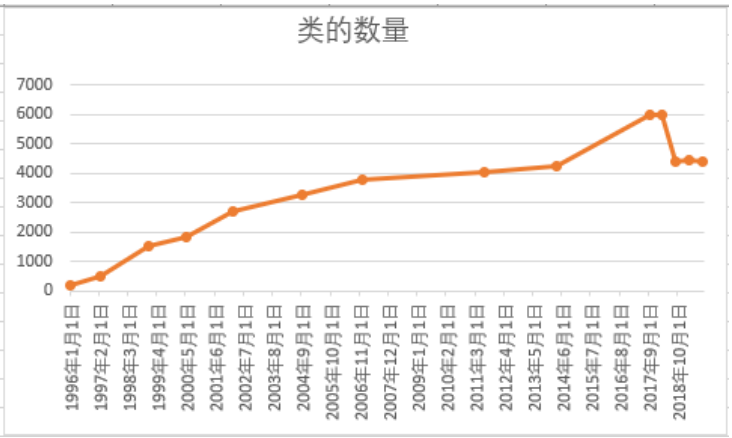


图 1-1. Java 版本和类的数量

从更新的时间上也可以看到，Java 的更新有时候很慢，有时候会跳票，甚至有一段长达 5 年没有任何更新的断档期。时过境迁，最近几年，Java 发布间隔才开始规律起来，从上图可以看到，Java 每半年发布一个新的版本，大量的新功能被快速迭代。仔细研究的话，还是可以从这些更新中找出一些规律：

1. 主要版本更新会引入重大新特征

Java 通常会以较长的时间间隔发布一个大版本，例如 Java SE 8、Java SE 9、Java SE 10 等等。主要版本更新通常会引入重大的新特性和语言改进，例如 Lambda 表达式、模块化系统等等。这些新特性可以提高开发者的生产力和编程效率，同时也可以改进 Java 程序的设计和质量。

2. 中间版本更新通常是引入小特征

Java 还会在主要版本之间发布一些中间版本，例如 Java SE 8u45、Java SE 8u60 等等。中间版本更新通常会修复一些 bug 和安全漏洞，同时还可能引入一些小的语言改进和新特性。Java 还会在不同的版本之间进行一些性能优化，例如改进垃圾回收机制、优化 JIT 编译器等等，这些优化可以提高 Java 程序的性能和响应速度。

3. 安全更新主要是修复已知的安全漏洞

Java 还会不定期发布安全更新，用于修复 Java 虚拟机和 Java 类库中的安全漏洞，并增加新的安全特性，以防止黑客攻击和数据泄露。因此，保持 Java 版本更新是保证安全性的重要手段。

我们在做出更新的决定之前，要从安全性、性能、新特性、兼容性综合考虑，仔细检查新版本的文档和升级指南，了解新版本的变化和影响，并进行相关的测试和调整，以确保 Java 应用程序可以在新版本中正常运行，并根据自己的实际情况和需求，选择合适的 Java 版本。

1.3 Java 语言的优势

通过上面的介绍，我们大体了解了 Java 语言的由来。那么，让我们回顾一下，看看 James Gosling 提出的问题，Java 都解决了么？进而，我们思考这样一个问题，在解决问题的过程中，编程语言在其中所占的比重是多少？

大概是 2007 年左右，我那时刚参加工作不久，当时主要用 C 和 C++写软件，那一年发生了一件在程序员界比较轰动的事情，Linux 的开发者 Linus 因为一个叫 Dmitry Kakurin 的开发者，炮轰其写的 git 软件不是采用 C++而是采用 C，于是 Linus 与 Dmitry Kakurin 在网上展开了争论。那场争论没有涉及到 Java，但是我认为涉及到了一个核心问题：**编程语言在解决问题的过程中，到底扮演了多么重要的角色？**

这些年来，我一直考虑这个问题，目前我心中的答案是这样的：编程不应该是神秘莫测的东西，对编程来说，最重要的是设计。编程语言在其中起到的作用是让编程不要神秘莫测，如果编程语言本身太过复杂（特性太多）而难以让程序员掌握，那是编程语言本身的问题，而不是程序员的问题。好的编程语言让软件设计变的简单，而不是把程序员困在编程语言的

泥潭里。

在阅读本书或者以后的编程中，希望大家能权衡利弊，在 Java 这类面向对象的编程语言中，会涉及“抽象”“面向对象”“设计模式”……等术语，这些都是非常重要的东西，但是不要执迷于此，因为最重要的东西是软件本身。以我的见闻，“面向对象”程序员非常容易在一次次“抽象”中丢掉了“真实”。

虽然都说“人教人，教不会；事教人，一次就会”，有时候犯错误就像小孩碰热水杯一样，无伤大雅，但是有时候付出的代价有点大，比如项目投入了很多，最后被迫取消了。太阳底下无新事，以后发生的事，以前也发生过，还是尽量的从历史中学习一下，看看当时的人是怎么处理的。Java 当年面对的情况，今天也要面对。

1.3.1 降低了编程难度

在前文中提到，由于当年设备众多且大多互不兼容，每一类设备都要进行不同的更新，这对程序员是一种沉重的负担。客观来说，仅考虑设备这一项，Java 平台的推出确实解决了当时设备过多导致的问题。

同时Java提供了丰富的类库和工具,这些类库和工具封装了一系列常用的功能和算法，程序员可以通过调用这些类库和工具的接口来完成相应的任务，避免了重复编写代码的麻烦。Java 标准类库（Java Standard Library）包含了大量的类和接口，涵盖了各种应用场景，例如集合、IO、网络、多线程、安全等等，程序员可以通过引入这些类库来快速开发高质量的应用程序。再者，Java 的技术文档非常详实，我工作这些年，觉得只有微软的 MSDN 文档可以和 Java 的文档相媲美，在第 2 章注释的那一小节，我会讲到 Java 的文档与注释以及其所尊崇的文学编程理念。同时，Java 还拥有众多优秀的开发工具和框架，例如 Eclipse、NetBeans、Spring、Hibernate 等等，这些工具和框架可以帮助程序员提高开发效率，降低开发成本。

今天的 Java 已经可以运行在计算机，手机，嵌入式设备上，极大的减轻了程序员面对不同设备时候移植的工作负担。至于说设备联网之后的安全问题,设备之间互相操作的问题,仍然没有很好的解决。网络安全已经成为社会层面的问题,不可能仅通过技术层面上来解决,社会问题很难完全通过技术来解决。

1.3.2 拉平了学习曲线

在软件开发领域中，拉平学习曲线是非常重要的，因为技术和工具的更新换代非常快，程序员需要不断学习新的知识和技能来跟上时代的步伐。Java 作为一种广泛应用于软件开发的编程语言，具有很多特性和工具可以帮助程序员降低学习曲线。

在设计 Java 的过程中， James Gosling 提到要用程序员熟悉的语法和编程思想，我认为这一点也达到了。如果你学会了 Java，那么你再学 C 语言，C++，甚至 Python 语言都不会有太大的难度。相反，如果你已经会了 C/C++、Python 语言，再学 Java 也更容易。

至于说面向对象编程，相比于其它的语言，Java 也算中规中矩，并没有太多特别出奇的用法。在用户界面设计领域，有一个原则叫最小惊讶原则，我认为同样适用于编程语言。编程语言应该尽可能减少“惊讶”，在其它语言中掌握的知识，可以平滑的应用在这门语言中。

我们可以分别来比较一下，因为相同点实在太多了，我只列举一些明显的不同点。

1. C 和 Java 相比

- C 语言是面向过程的，Java 是面向对象的。
- C 语言有指针，Java 没有指针。
- C 语言要程序员手动管理内存，Java 自动管理内存。
- C 语言有预处理机制，Java 没有。
- C 语言移植要重新编译，Java 的字节码可以直接运行。

在访谈里，James Gosling 这样评价 C 语言：“C 语言是伟大的设计，但是环境是不断变化的。当时的计算机不能联网，而是放在屋里彼此隔绝。没人关心不同的计算机使用什么编程语言，只是希望自己的代码能在前面这个又大又慢的机器上顺利运行。”

2. C++和 Java 相比

C 和 C++有很多相似的地方，这里只比较面向对象方面的异同。

- 相比于 C++，Java 的面向对象模型更简单。
- Java 不支持多重继承。
- Java 不支持操作符重载。
- Java 和 C++都支持泛型，Java 的功能要弱一些，但是更安全。

相比于 C/C++，Java 有更多的限制，在前文提到的访谈中，James Gosling 对此的解释是：“对 Java 来说，规矩有很多，一旦你适应了规矩，那么 Java 将是一种自由的语言。”他还以飞机为例打了个比方，在螺旋桨时代，飞机非常的自由，飞行员可以打开窗户，呼吸新鲜的空气，可以通过肉眼来观察方向。到了喷气式飞机时代，飞机的窗户是不能打开的，如果你打开，在 3 马赫的速度下，飞行员的脑袋将会被吹走。最后，他总结说：“如果你想进一步让自己自由，就要放弃一些看起来是自由的东西。”

3. Java 和 JavaScript 相比

这两门语言对初学者来说实在太容易混淆了，实际上，Java 和 JavaScript 的关系，类似于印度和印度尼西亚的关系。这两门语言除了名字有点相似以外，几乎没有什么联系，如果非要拉关系，也只有一点点渊源。Java 是 Sun 公司的产品，JavaScript 是 Netscape 公司的产

品，当年 Sun 公司和 Netscape 公司关系很好，Netscape 公司就让程序员 Brendan Eich[14]设计一种语言，最好参考当时火热的 Java，蹭点热度。

JavaScript 和 Java 的关系考证

Netscape 公司决定让 Brendan Eich 设计一种可以在浏览器上运行的语言，最好是一个“简化版本的 Java”。但是 Brendan 很讨厌 Java。他在访谈里讲了，他的兴趣是研究函数编程，比如 Scheme 语言，为了交差，他只花了 10 天，就设计好了 JavaScript。

JavaScript 借鉴了 C 语言的语法，借鉴了 Scheme 语言的函数，还借鉴了 Self 语言的继承机制，管理层要求的借鉴 Java，他只借鉴了内存管理。

对于每天影响数亿人的 JavaScript 语言，他的自我评价是：“他恨 JavaScript”。他还进一步引用了一位叫 Johnson 的博士的话来评价自己的作品：“它优秀的地方不是原创，它原创的地方并不优秀。”

1.3.3 提高了跨平台能力

在 James Gosling 的文档里，他用整数的例子说明了可移植性的难度。Java 的可移植性是指 Java 程序可以在不同的操作系统和硬件平台上运行，而不需要对程序进行修改或重新编译，例如 Windows、Linux、MacOS 等等。这为开发者提供了更大的开发自由度和灵活性，也为企业带来了很多好处。我认为，包括 Java 在内的绝大部分现代编程语言，基本都解决了这个问题。像 Python，Ruby 都和 Java 一样，宣称自己是跨平台的编程语言。

Java 是最早尝试一劳永逸解决可移植的问题的编程语言，并且解决得还不错。

松本行弘与 Ruby 语言

上面提到了 Ruby 语言，作者本人最喜欢的语言之一就是 Ruby，忍不住多介绍几句。松本行弘是日本计算机科学家与程序员，是 Ruby 语言的主要设计者，他写了《代码的未来》《松本行宏的程序世界》等多部图书。目前在 Heroku 公司当首席架构师。

Ruby 与 Python 类似，区别是 Ruby 的作者希望使用 Ruby 的程序员能感到快乐，以快乐为目标设计的编程语言。

松本行宏说过：“编程语言是在不断的试错中发展起来的。虽然有很多的编程语言已经消亡，只留下了一个名字，但是其中所包含的思想，被后来的编程语言以不同的形式加

[14] 关于 JavaScript 研发的过程，发明人 Brendan Eich 在他的网站上有比较详细的回忆，建议大家自行搜索阅读。

入吸收与借鉴。”

只要认真的学习一门语言，其它的语言大部分都可以触类旁通。

1.4 常见问题

1.4.1 Java 的名字是谁起的？

以目前能找到的资料来看，[15]Java 项目最开始的时候，名字并不是 Java，而是 Oak。之所以叫 Oak，是因为在 James Gosling 的办公室外有一棵橡树。可惜当时有一家已经存在的公司叫 Oak Technologies，由于版权的原因，只能改名。

改名的时候有几个候选，分别是 Silk、Lyric、Pepper、Java。当时的产品经理 Kim Polese 提出了 Java 这个名字并最终采用。

1.4.2 Java 的现状如何？

无论是从找工作还是从学习编程的角度，Java 都是一门不可忽视的语言。

大家只要打开搜索引擎或者 GitHub 网站，查找一下语言的排名，或者在找工作的网站上看一下，就知道 Java 始终是最热门语言之一。写这本书的时候，我在找工作的网站 dice.com 上使用关键字 Java 进行了搜索，有 11,153 个工作，用 Python 当关键字，有 6,395 个工作，用 JavaScript，则有 8,634 个工作。这些结果应该相对直观的反应了市场需求。

从学术和技术角度来看，现在越来越多语言都是基于 Java 实现，运行在 Java 虚拟机之上。比如 Groovy、JRuby、Scala、Jython、Clojure 这些语言，都是如此，只要能运行于 Java 虚拟机，就是站在巨人的肩膀上，有了与生俱来的跨平台特性。从这个角度来看，学会 Java，能更深入的了解这些语言。

业界有一句名言：“没有人因为选择 Java 而被解雇。”这句话的潜台词是：Java 的背后有大公司的支持，这意味着 Java 能得到长期的维护和开发，具有繁荣的生态系统，可以减少技术选型的风险。这也是 Java 长期排名前茅的原因。

[15] Javaworld 曾经写了一篇文章叫《So why did they decide to call it Java? 》，文章里详细介绍了为什么起名叫 Java。

1.5 总结

本章主要的内容要点有：

- Java 有悠久的历史，自从发布以来，不停的修订，现在已经是使用最多的语言之一。
- Java 借鉴了多种语言，比如 C/C++、SmallTalk、Objective-C 等语言的优点，对程序员来说，Java 是一门简单，容易上手的语言。
- 在 1995 年 James Gosling 的两份文档中，就指明了 Java 的发展方向，随后多年，Java 一直沿着这两份文档的方向前进。这两份文档，也是本书的主要架构。
- Java 有良好的兼容性，虽然经历了诸多版本的改进，但是总体而言都是沿着 “碰到问题，解决问题，不增加程序员负担” 为前提改进的。Java 程序，一般情况可以向后兼容。
- 根据历史文档，分析了当年的问题今天是否已经被 Java 解决。
- 考证了 Java 名字的由来。
- Java 是非常值得学习的语言。

1.6 思考拓展

1. 下载本书的配套资料，阅读第 1 章的那两份 James Gosling 在 1995 年写的《Java: an Overview》和《The Java Language Environment》思考一下当年存在的问题是否已经被 Java 解决？如果你认为没有解决，让你来提出解决方案，你会怎么做？
2. 在网上搜索对 James Gosling 的访谈的视频，或者在配套资料中下载访谈的文本，思考这样一个问题：编程语言的特性是被一个人决定好，还是被一个编程委员会共同决定好？[16]

[16] 不止 Java，像 Python，JavaScript 语言，都经历了开始主要由一个人主导，后来一个委员会主导，到最后，语言的创始人离开。2018 年 7 月 12 日，Python 创始人 Guido van Rossum 发邮件决定离开 Python 决策层，不再领导 Python 语言的开发。有兴趣的读者，可以找来那封邮件来读一下。

2. 第一行代码

Hello World 的由来

1974 年，美国贝尔实验室，一位名叫 Brian Kernighan 的研究员，他写了一份名为《Programming in C -- A Tutorial》的文档，在这个文档的第二段，他用 C 语言写了一个经典的程序，程序是输出 hello world 这两个单词。

后来这个研究员又写了一本风靡全球的书叫《The C Programming Language》，在这本书里，他再次用了上面提到的程序，从此，几乎每一个编程语言的第一个程序都是输出 hello world 这两个单词。

2.1 Java 版的 Hello World

2.1.1 输入源代码

程序员通过一系列字符来表达软件的运行逻辑，这种字符被称为源程序（source program），用来保存源程序的文件被称为源文件（source file）。Java 是面向对象的编程语言，类（class）是实现面向对象的机制，本章的下一小节会详细介绍 Java 的类。

首先，新建一个文件 HelloWorld.java，用文本编辑器输入下面代码框中的内容。需要注意的是，由于 Java 源程序区分大小写以及全角半角，所以源代码一定要跟下面的一模一样，不能用全角的括号、空格和引号。另外，空格部分使用空格键或者 Tab 键均可。

代码清单 2-1 在屏幕上显示 hello world 两个单词

```
class HelloWorld {
    public static void main (String[] args) {
        System.out.println("hello world");
    }
}
```

还有一点需要注意，源文件的名字要和类名相同，在这个例子中，类名是 class 之后的 HelloWorld，因此，源文件的名字必须是 HelloWorld.java，其中的后缀. java 是 Java 源文件的扩展名。如果名字不相同，就没法进行下一步的编译。

2.1.2 编译和运行

当源程序编写完成以后，接下来是要把软件运行起来。源程序无法直接运行，对 Java 来说，要运行软件还需要两个步骤：第一步是把源程序编译，第二步是运行编译后的文件。

所谓编译（compile）就是将源代码转换为可以运行的格式，这种格式被称之为字节码（bytecode）。运行则是使用 Java 虚拟机（Java Virtual Machine）将字节码运行起来。要完成这两步，需要在计算机上安装专门的软件：JDK。

JDK 的全称是 Java Development Kit，是由 Oracle 针对 Java 开发人员发布的开发工具包（SDK，Software development kit），其中包含各种工具以及 Java SE (Standard Edition) 的核心类库，使用这些工具可以完成将源代码编译成字节码，将应用程序打包，启动 Java 虚拟机运行字节码等工作。

Oracle 发布的 JDK 主要有两个版本，一个是由 Oracle 公司开发与维护的版本，称之为 Oracle JDK，该版本主要通过 Oracle Binary Code License (OBCL)进行许可，使用该版本进行商业活动要付许可费。另一个是 OpenJDK，这是由 Oracle 公司开发并以 GPL 许可证发布，是一个免费开源项目，由全球开发人员以开源社区的形式进行开发和维护。这使得 OpenJDK 成为许多不同的 JDK 实现的基础，这些不同的 JDK 包括 Amazon Corretto、AdoptOpenJDK、Zulu 等。

这么多不同版本的 JDK，用哪个最好呢？有疑问的人不在少数，甚至有一个网站专门来回答这个问题，网址是 whichjdk.com。网站上对不同 JDK 的优缺点都有涉猎，长话短说，对初学者，在我写书的时候，网站上推荐使用 Adoptium Eclipse Temurin 17 这个版本，本书就采用的这个版本。

如果你对不同 JDK 的来历感兴趣，可以阅读本章后面对 Java 虚拟机的介绍，了解不同虚拟机背后的故事。

1. 安装 JDK

首先，到 Adoptium Eclipse Temurin 的网站（adoptium.net）上下载对应平台的 JDK。我觉得没有必要针对每一个平台都演示一次，所以在本书中仅以 Windows 平台为例进行安装，如果读者使用的是 Mac OS 或者 Linux 平台，可以按照说明一步步安装。

在 Windows 下，根据平台的情况选择 32 位还是 64 位，网站会自动下载相应的安装包，下载完成后，会得到一个 exe 文件，安装过程跟其它的 Windows 软件没有什么差异，一直点下一步直到结束。

要验证是否安装成功，在 Windows 上打开命令提示符，输入 java --version，如果出现下面所显示的结果，证明 JDK 安装成功。

```
PS C:\Users\baoch> java --version
openjdk 17.0.6 2023-01-17
OpenJDK Runtime Environment Temurin-17.0.6+10 (build 17.0.6+10)
OpenJDK 64-Bit Server VM Temurin-17.0.6+10 (build 17.0.6+10, mixed mode, sharing)
```

图 2-1 显示 Java 的版本

编程知识小贴士

如果你因工作或者测试的原因，需要不同版本不同厂家的 JDK，一一手动安装会比较麻烦，在此推荐一个自动的工具：SDKMAN。使用这个工具，可以方便在不同版本（从 Java7 到 Java17），不同厂家（包含 Oracle、IBM、Amazon 等主流厂商）的 JDK 中选择，具体的使用方法可以在官方网站上查询。

2. 编译

在 Windows 下打开命令提示符，进入存放 HelloWorld.java 的文件夹 HelloWorld，输入这行命令：javac HelloWorld.java 。如果没有出现任何错误，在 HelloWorld 文件夹下，会多出一个名为 HelloWorld.class 的文件。

这个过程我们称之为“编译”，所使用的软件为 javac，javac 是 Java 语言的编译器，是一个命令行工具，其作用是将 Java 源文件（后缀为.java 的文件，在本例中为 HelloWorld.java），转换成可以在 Java 虚拟机上运行的 Java 字节码（后缀为.class，在本例中为 HelloWorld.class）。

Java 字节码是一种中间表示形式的代码，它不限定某种平台，而是一种跨平台的二进制代码。它可以在任何平台的 Java 虚拟机上运行，当 Java 源代码编译为字节码后，字节码可直接传输到任何支持 Java 虚拟机的平台上运行，而不用重新编译，这使得 Java 跨平台编程成为可能。

当编译完成以后，打开 HelloWorld 文件夹，会发现其文件结构变成下面这个样子：

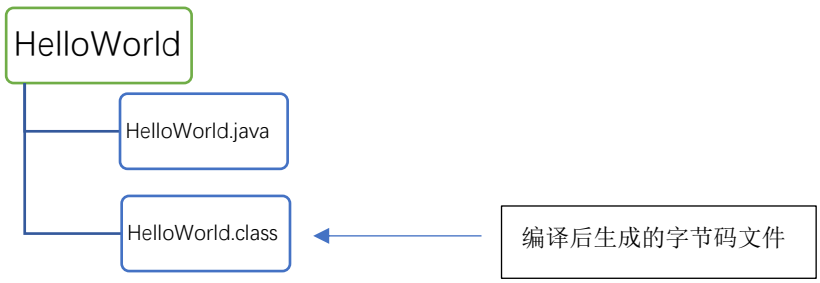


图 2-2 编译后的文件夹结构

3. 运行

上一步编译成功以后，现在可以运行了。运行的命令是：java HelloWorld，然后就在命令行上看到 hello world 这两个单词了。

当我们在命令行中输入 java 命令时，实际上是在调用 Java 虚拟机来解释和执行 Java 字节码。因此这行命令实际上是在 Java 虚拟机中运行名为 HelloWorld 的 Java 字节码。在运行时，Java 虚拟机将这段字节码转换为本机机器代码，并在本机平台上执行该代码。

这里需要稍微注意的是，运行 Java 字节码的时候，通常需要指定类的全名（包括包名和类名，在此处为 HelloWorld），而不需要输入.class 这个 Java 字节码的后缀，Java 虚拟机会自动搜索并加载相应的字节码文件并运行。初学者比较容易犯的错误是多输入了后缀，像 java HelloWorld.class 这样，从而导致程序无法运行的。

2.1.3 回顾写程序的流程

至此，我们已经运行了第一个 Java 程序，输出了两个单词 hello world。麻雀虽小，五脏俱全，从这个小程序中，我们完整的走完了写程序主要的三个流程。

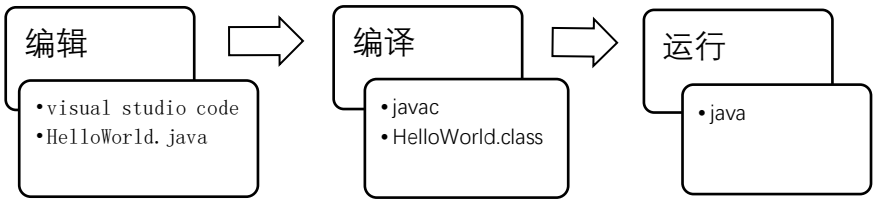


图 2-3 写程序的三个流程

总结一下，在本书中，用 Java 编程的三个阶段包括编辑、编译和运行。下面对每个阶段进行详细一点的回顾：

1. **编辑阶段**

编写 Java 程序的第一步是编辑代码。我们可以使用任何文本编辑器或集成开发环境来编辑 Java 代码文件。在编辑代码时，程序员应该遵循 Java 编码规范和最佳实践，以确保代码的可读性和可维护性。编辑完成后，可以保存代码文件并准备进行编译。

2. **编译阶段**

Java 程序是以源代码的形式编写的，而源代码需要被编译成可执行的字节码文件。在 Java 中，编译器将源代码文件（.java）转换为字节码文件（.class）。我们可以使用 Java 编译器（javac）来编译 Java 代码。以 HelloWorld.java 为例，在命令行中，您可以使用以下命令编译 Java 代码：javac HelloWorld.java，这将编译名为 HelloWorld.java 的 Java 源代码文件，并生成名为 HelloWorld.class 的字节码文件。如果编译过程中没有错误，就可以进入下一个阶段：运行阶段。

3. **运行阶段**

一旦已经编译了 Java 程序，就可以运行它。还是以本章的 HelloWorld 为例，在命令行中，可以使用以下命令来运行 Java 程序：java HelloWorld。这将运行名为 HelloWorld 的 Java 程序。在运行时，Java 虚拟机将读取字节码文件，并将其转换为机器代码。然后，它将执行机器代码，并输出程序的结果。如果 Java 程序中存在错误，JVM 会抛出异常并停止程序的执行。

总的来说，Java 程序的三个阶段是编辑阶段、编译阶段和运行阶段。通过这三个步骤，就可以将 Java 源代码转换为可执行的程序，并在计算机上运行它。

2.2 解析代码

接下来详细的分析一下 Java 版的 hello world。

```
1.  // 显示 hello world 两个单词
2.  class HelloWorld {
3.      public static void main (String[] args) {
4.          System.out.println("hello world");
5.      }
6.  }
```

2.2.1 注释

先从第 1 行开始，这是一种用于解释和说明代码的文本，它不会被编译器识别或执行。这些文本是给程序员看的，而不是让计算机执行的，术语叫注释（comment）。**注释是一种程序员在源代码中用简洁的语言把信息传递给源代码阅读者的方式**。在 Java 中，有三种类型的注释，分别为单行注释、多行注释和文档注释。

单行注释的方法是使用双斜杠：//。单行注释以//开头，后跟注释文本。单行注释可以在代码的任何位置使用，并且从//开始到该行的末尾，所有文本都将被视为注释。

多行注释的方法和 C 语言中注释的相同：/* 这里可以插入多行注释 */。多行注释以/*开头，以*/结尾，可以跨越多行。多行注释通常用于注释大段代码，或者是快速暂时注释掉一段代码。看到这个注释，我忍不住想起我认为最有个性的一个用 C 语言写的注释，这个注释出现在 Unix v6 版的操作系统源代码中，注释是这样写的：

/* You are Not Expected to Understand This */ （我不指望你懂这是啥意思）

写这行注释的人是 Unix 的作者之一 Dennis M. Ritchie，希望有一天，本书的读者也有底气写这样一行注释。如果目前写不出这么霸气的注释，至少记住写注释的格式。后来，Ritchie 本人可能也意识到写这行注释太过于狂妄，尤其是考虑到当时那本 Unix 源码书几乎每个程序员人手一本，他后来解释说这行注释的意思其实是“这些代码不会考试，所以你不用懂这是啥意思”。

还有一种注释被称之为文档注释，所用的方法是：/** 这是一个文档注释 */。文档注释以/**开头，以*/结尾，并位于 Java 类、方法或字段的定义之前。文档注释通常用于生成 Java 文档（Javadoc），可以包含 HTML 标记和特殊注释标记（如@param 和@return），以便生成文档时自动生成说明文档。我从 JDK 的源码中抽取了一个例子，源码有删减，主要用来展示 JDK 的源码中如何使用文档注释的。

代码清单 2-2 JDK 源码示例：展示如何使用文档注释

```
1.  // Attributes.java - attribute list with Namespace support
2.  package org.xml.sax;
3.  /**
4.  * Interface for a list of XML attributes.
5.   */
6.  public interface Attributes
7.  {
8.      ////////////////////////////////////////////////////////////////////
9.      // Indexed access.
10.     ////////////////////////////////////////////////////////////////////
11.     /**
12.      * Return the number of attributes in the list.
13.      *
14.      * <p>Once you know the number of attributes, you can iterate
15.      * through the list.</p>
16.      *
17.      * @return The number of attributes in the list.
18.      */
```

通过上面的例子，可以看到单行注释和文档注释，为了美观，有些会刻意多加一些星号*，比如第 4 行，第 12 到 17 行开始的星号*可有可无。像第 8 行和第 10 行的注释也可有可

无。这些可有可无的符号只是为了美观。在使用 javadoc 生成文档的时候，这些可有可无的星号*会被忽略。

javadoc 能生成 HTML 文档，如第 17 行所示的@return 这种以@开始的标签会被特殊处理。除了@return 之外 ，javadoc 还支持好几个以@开头的标签，这种标签不需要像背英语单词一样背诵，下面的图表中我列出常用的几个。这些标签用多了，自然就记住了，对初学者来说，暂时知道我们写的注释可以被工具提取出来制作漂亮的文档，就足够了，以后工作中用到 javadoc，再查相关文档就好。

需要注意的是，这些以@开始的标签适用于不同的场景，有的适用于 Class，有的适用于 Method，有的适用于 Variable，有的可能适用于以上三者，在表 2-1 中的适用场景一栏中，有详细的标注。

表 2-1 javadoc 中以@开始的标签及适用场景

标签	用途	适用场景
@author	标识作者的名字	Class
@code	源码的内容	Class，Method 和 Variable
@deprecated	被弃用的功能，不建议再使用	Class，Method 和 Variable
@exception	异常名和描述	Method
@link	相关的链接	Class，Method 和 Variable
@param	参数名与描述	Method
@return	返回值	Method
@see	相关的类名	Class，Method 和 Variable
@since	API 加入的日期	Variable
@version	版本信息	Class

写好注释的难度很大，和写好程序的难度一样大。

> 程序员最讨厌的四件事：写注释、写文档、别人不写注释、别人不写文档。

Sun 公司还专门对注释的问题做了研究，对如何写注释，用什么样的风格写注释，给出了许多详细的建议。推荐大家去搜索《How to Write Doc Comments for the Javadoc Tool》这篇文章学习。Javadoc 的这个规范，也许是目前最有说服力的文档之一。

说起 Javadoc，不得不提《计算机程序设计艺术》的作者 Donald Knuth 在 1984 年所提倡的文学编程（Literate Programming），文学编程有自己的官方网站，大家可以自行搜索。文学编程的基本思想是将程序代码和注释混合在一起，形成一个结构化的文档，使得人类读者可以更加容易地理解代码的意图和实现细节。这种方式可以使得代码更加易读、易维护和易扩展，同时也能够促进团队之间的协作和交流。Knuth 曾经说过：“优秀的程序员也能拿普利策奖”。[1]Javadoc 这个工具就是基于文学编程来实现的文档工具。

在计算机图书界，有一本比较著名的书叫《Clean Code》，该书的作者叫 Robert C. “Uncle Bob” Martin，是业界比较出名的人，他对于代码中出现的注释有自己的见解。他认为 “代码注释是一种道歉，为没有选择清晰的名称、合理的参数而道歉，为代码难以维护而道歉，为不使用知名算法而道歉，为编写"取巧的"代码道歉，为没有良好的版本控制系统道歉，为没完成代码编写工作而道歉，为留下漏洞或代码中的缺陷而道歉。”

我引用 Uncle Bob 的观点并不是我认同其观点，而是让大家开阔一下思路，从批评中得到自己想要的信息。当然了，Uncle Bob 大叔经常发表一些“惊世骇俗”的观点，但是，如果观点不够爆炸，在现在这个社会上，难以引起人们的注意。谁会在意平和的观点呢？陈奕迅的《浮夸》里这样唱道 ：“你当我是浮夸吧，夸张只因我害怕，似木头似石头的话，得到注意吗？”

如果你想了解更多与“主流意见”相左的观点，可以搜索参考这篇名为《Don’t comment your code. Refactor it.》的文章。

趣味问题：注释中的代码肯定不运行么？

100%肯定，注释中的代码肯定不会被运行，除非有人搞怪。比如下面的代码：

代码清单 2-3 注释在特定的时候会运行

```
1.  public class HelloWorld {
2.      public static void main(String[] args) {
3.          String name = "hello java";
```

[1] 他的原话是：I’m hoping someday that the Pulitzer Prize committee will agree.” Prizes would be handed out for “best-written program”.

```
4.          // \u000dname = "hello c++";
5.          System.out.println(name);
6.      }
7.  }
```

上面这些代码编译运行后会输出 hello c++而不是 hello java。下面我来解释一下原因，Java 编译器在处理源代码的时候，会从源代码中逐个读取字符，然后判断读取的内容，如果大家学过《编译原理》这门课，应该知道这个过程叫词法分析。尽管代码中存在着第 4 行看似注释掉的代码 \u000dname = "hello c++"，但实际上它会被 Java 编译器视为一条有效的语句，并被执行，因为\u000d 是 Unicode 转义字符，它表示回车符（carriage return），在第 4 章的时候会详细讲转义字符。对 Java 编译器来说，它看到的相当于下面这样：

```
1.  public class HelloWorld {
2.      public static void main(String[] args) {
3.          String name = "hello java";
4.          //
5.          name = "hello c++";
6.          System.out.println(name);
7.      }
8.  }
```

在第 3 行的时候，定义了一个字符串变量 name 并将其赋值为“hello java”,在第 5 行的时候，将 name 重新复制为“hello c++”，因此最终的结果输出为“hello c++”而不是“hello java”。如果对变量和 unicode 有疑问，会在本书的第 4 章详细介绍，看完第 4 章再回头看这个例子，一定会豁然开朗的。

2.2.2 声明类

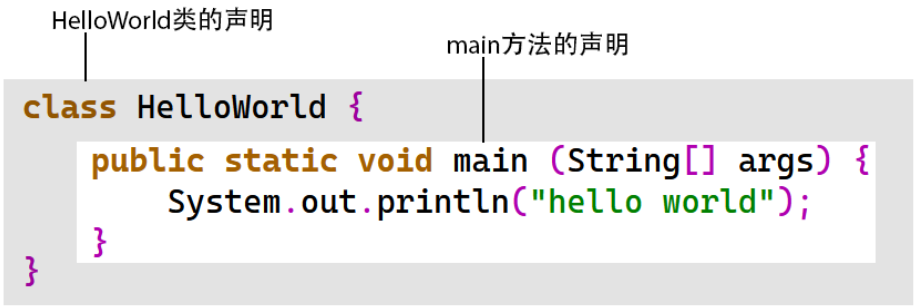


图 2-4 HelloWorld 程序的结构

如上图所示，阴影部分就是整个程序的骨架，所有的代码都被镶嵌到这个骨架之中。这几行定义了一个类，其意义为：声明了一个名为 HelloWorld 的类。

```
// 不管三七二十一，上来就写上这个骨架
class 类名{
//Java 的代码都写在这里面
}
```

图 2-5 Java 类的架构

在面向对象编程中，类是一种定义对象属性和行为的模板，包含了一组共享相同特征的对象的结构和行为。例如，汽车类可以定义汽车对象的属性，如车型、颜色、重量等，以及方法，如启动、加速、刹车等。目前还没有详细讲怎么设计一个类，会在本书的第 3 章详细介绍什么是类，目前先记住类的关键字 class 以及语法就好。

类名和源文件的名字一定要严格的保持一致。如果类名是 HelloWorld，那么源文件就要是 HelloWorld.java，如果类名是 Helloworld，那么源文件就要是 Helloworld.java。在 Java 语言中，是区分大小写的，HelloWorld 和 Helloworld 是不同的。

虽然和 C++、python 一样是面向对象编程语言，但是 Java 有所不同。C++、Python 可以用面向对象方法编程，也可以不用面向对象方法编程，但 Java 必须使用面向对象的方法来编程，如果没有类，Java 没法运行。正是因为这个原因，Java 的一切都要包在类中，也就是说 Java 要求每个程序都必须包含至少一个类。

还有，Java 对类的名称也有规定，必须是以字母开始，随后可以是数字和字母，并且不能用到 Java 的保留字，比如不能用 public 当类的名字，因为 public 是保留字。

首字母要大写。HelloWorld 首字母就是大写，如果名词是有多个单词组成的，像 HelloWorld 是由两个词组成的，那么这两个词语都要把首字母大写，这种写法有个名字叫“camel case”，翻译成中文叫“骆驼拼写法”。

> **骆驼拼写法**
>
> 我考证过骆驼拼写法的历史，历史上早就有这种写法了，比如 DuPont 公司。只是计算机编程兴起以后，才被大规模的使用。骆驼有两种，一种是只有一个驼峰的，一种是有两个驼峰的。所以，程序员也发明了两种骆驼拼写法，一种是 lowerCamelCase，叫小骆驼拼写法，一种是 UpperCamelCase，叫大骆驼拼写法。在 Java 的类名中，用大骆驼拼写法。

2.2.3 main 方法

图中白色的部分是 **main 方法**（main method）的声明。在 Java 中，**main** 方法是一种特殊的方法，用作 Java 程序的入口点。当 Java 程序运行时，Java 虚拟机从指定为程序入口点的类的 main 方法开始执行。

main 方法前面加了好几个 “修饰词” ，它们分别是 **public static void**，这几个“修饰词”会在本书的第 11 章详细介绍，目前只简单的介绍一下，其意义分别是：

- public: 指定 main 方法可以从类外部访问。
- static: 指定 main 方法属于类本身，而不属于类的实例。
- void: 指定 main 方法不返回值。

main 后面则跟了一个括号，里面又有两个 “固定” 词组为 String [] 和 args，其意义是当 Java 用作命令行软件的时候，指定一个字符串数组，将命令行参数传递给程序。

```
public static void main (String[] args) {


}
```

main 方法里面的语句要放在大括号里，这些语句是程序运行时执行的代码，这些代码包括对其他方法的调用、输入/输出操作以及其它程序逻辑。

当程序运行的时候，会寻找 main 方法，没有 main 方法 Java 没法运行。找到以后，main 方法里的语句（statement）会依次执行。

按照 Java 语言的规则，main 方法必须被声明为 public，但是软件难免会出现 bug。在 Java 1.4 之前的一些版本里，public 不被声明为 public 也可以运行。有兴趣的可以登录 Java 的 bug 系统：https://bugs.java.com/bugdatabase/index.jsp，然后输入 ID 为 4252539 的 ID，就会发现一个 bug。

图 2-6 当 main 方法被声明为 private 也可以运行的 bug

这个 bug 从来没有被修复过，主要原因是如果修复了，可能会导致其它不可预知的结果。[2]

我举这个例子并非想证明 Java 不好，万一你发现 main 方法不加 public ，可能你碰到了一个历史悠久的 bug。

> “没人理”的 bug
>
> 在很多系统中，有一些 bug 没有被修复。在苹果的 iOS 中，也有这样的 bug。如果大家用 iOS 1582 calendar 作为关键字进行搜索，会找到这个 bug 的详细介绍。
>
> 简单来说，在 1582 年，人类的日历从儒略历换成了格里历。这就导致在 1582 年的 2 月份出现了 2 月 31 日。iOS 系统没有恰当的处理这个 bug，以后不知道会不会处理，但是谁会在意 1582 年的日历呢？

2.2.4 语句

[2] 在软件史上，有很多类似的情况，bug 成了 feature，将错就错。比如 Unix 上的 creat，本来应该是 create，但是少写了一个 e，后来所有的 Unix 变种都将错就错的少写了一个 e。后来有人问到 Unix 的作者 Ken Thompson 如果你要重新设计 Unix，你会做哪些改变，他回答说把 creat 写成 create。这个故事来源自 Ken Thompson 在维基百科的词条。

输出字符串“hello world”的那一行被称之为**语句**（statement）。语句是程序最基本的运行单位，是用于执行特定操作的指令，每一个语句都以分号 “;” 结尾。

在 Java 中，语句有相应的执行规则，语句可以根据需要组合在一起，形成复杂的程序逻辑。Java 中有多种类型的语句，包括表达式语句、声明语句、控制流语句和块语句等类型，根据不同的类型，这些语句会按照顺序、分支、循环等方式执行。

```
System.out.println("hello world");
```

先来讲这个语句中的点“.”号，在 Java 中这个点叫“成员访问操作符号”，通常也被称为“点操作符”，用作访问对象的成员变量和成员方法的分隔符。关于操作符，详细的讲解还需要在第 5 章。

System.out.println() 目前可以理解为 “调用 Java 的标准输出流中的 println () 方法”，println 中的 ln 是英文单词 line 的缩写，它的作用是将指定的信息打印到标准输出流中（通常是控制台），并在结尾处自动添加一个换行符。关于方法的具体含义和用法，会在本书第 8 章详细介绍。所谓的流，可以认为字符像水一样流进流出计算机，标准输出流特指用户当前的屏幕。

这个方法可以接受多种类型的参数，如字符串、数字、布尔值等，并会将它们自动转换为字符串输出。在这个例子中，接收的是双引号字符串常量 “hello world”，其中的双引号表示的是字符串的开始与结束，因此在输出的时候不输出引号 ，只输出 hello world 这两个单词，输出结束以后再输出一个换行符。这个方法在调试代码或者测试程序时非常有用，可以帮助开发人员快速地检查代码输出的结果。

2.3 调试程序

上面讲的流程非常理想化，一点错误都没有。写程序不可能一帆风顺，总会碰上一些不顺利的地方。

在我写这本书的时候，我也想过，就这么几行代码，一般是不会出错的，在第 2 章就写如何调试程序，有点太早也太难了，一般的编程书，都不写如何调试程序。但是我想到了我上大学的时候，我拿着一本 C 语言的书，打开 Turbo C 这个当年最流行的 C 语言集成开发环境，在学校的机房里，每小时 1 元钱，我花了 8 元钱，连个 Hello World 也没运行起来。当时不能联网，书上也没写如何调试，我尝试了几乎所有方法，读了几乎所有文档，也没成功。后来才知道，学校的计算机硬盘太小，同学们上机的时候，要玩游戏，为了安装游戏，要删一些东西，而集成开发环境是最没用的，所以优先删集成开发环境，那台计算机上，删了 C 语言的链接库，因此我坐了 8 个小时没让 Hello World 运行起来。

我觉得非常有必要提前写一点调试相关的内容，至少在碰到错误的时候，有个头绪。

调试软件伴随程序员的职业生涯

关于调试，第一台可存储程序的计算机 EDSAC 的设计者、1967 年图灵奖得主 Maurice Wilkes 曾经这样说过：“刚开始编程时我们就惊奇的发现，要想让程序正常运行，并非我们想象的那样简单。你必须认识到调试的重要性。有一天我突然就意识到，我生命中的相当一大部分时间会花在寻找程序的错误上，那一刻，我终生难忘。”

与大家共勉，我们也是这样，只要编程就会出错，就要寻找错误并改正。

编程中的 bug 是程序员写的，修复 bug 也要程序员来做，程序员有点像“侦探”，但同时又是“罪犯”。

2.3.1 调试程序五步骤

当我们写程序碰到错误的时候，应该怎么去应对呢？这时候，程序员的朋友是编辑器和编译器，无论你使用哪一种编辑器或者集成开发环境，主流的软件都有强大的调试功能，并且编译器也会给出相应的信息。可以尝试以下几个步骤来解决问题：

1. 仔细阅读错误消息：Java 编译器和虚拟机会输出有关错误的详细信息。可以读取这些消息，以了解出现问题的原因和位置。在错误消息中通常会提供有关错误类型和位置的提示，这可以帮助您缩小问题的范围。
2. 检查代码：阅读您的 Java 代码，并尝试找到潜在的错误。可能存在语法错误，如缺少括号、分号等等。
3. 利用调试器：使用调试器来查找代码中的错误。通过设置断点，在程序执行到特定位置时停止，然后查看程序的状态和变量的值，可以帮助您理解程序执行期间发生了什么。
4. 借助网络资源：在互联网上有大量的 Java 社区和论坛，您可以在这些地方搜索相关的问题，最后可以向其他 Java 开发人员寻求帮助和建议。
5. 借助大语言模型：现在流行的大语言如 ChatGPT、Bard 等都可以进行代码编写与调试，只需要把相关的代码输入到大语言模型中，大语言模型会有一定概率给出正确的答案，当以上解决方式都无效的时候，可以借助大语言模型来调试程序。

2.3.2 选择一个好用的开发环境

古人云：“工欲善其事，必先利其器。”一个好的开发环境可以减轻我们调试的负担，所以，一定要选带有语法提示的开发环境。开发环境的选择颇为广泛，除了简单的文本编辑器，还有更复杂也更强大的 IDE（Integrated Development Environment，集成开发环境）供我们选择，比如我一直使用的 JetBrains 公司出品的 IDE。该公司出品的 IntelliJ IDEA 对初学者有

点过于强大了，本着“由俭入奢易，由奢入俭难”的原则，我建议初学者先用不那么强大的 IDE 来学习，过于强大的 IDE，仅需要一次点击就完成了所有操作，虽易用，但是难以了解这次点击背后的相关操作。

对初学者，我推荐一个专门为 Java 语言的初学者设计的 IDE：Bluej。这个开发环境是 Oracle 赞助的，包括 James Gosling 在内的人都曾经推荐过，它的目的是为学习和教授 Java 编程。最大的优点是该软件专门针对 Java，可以自动生成类之间的关系，还允许用户通过一个可视化对象交互界面来创建和操作对象，以及查看对象之间的关系。用户界面是基于对象、类和方法的，这使得初学者可以更轻松地理解代码的结构和逻辑。

BlueJ 还提供了许多其他的功能，如语法高亮、自动补全、调试器和交互式控制台等。它还可以与其他工具集成，例如 Git 版本控制和 JUnit 测试框架，以帮助用户更好地管理和测试代码。

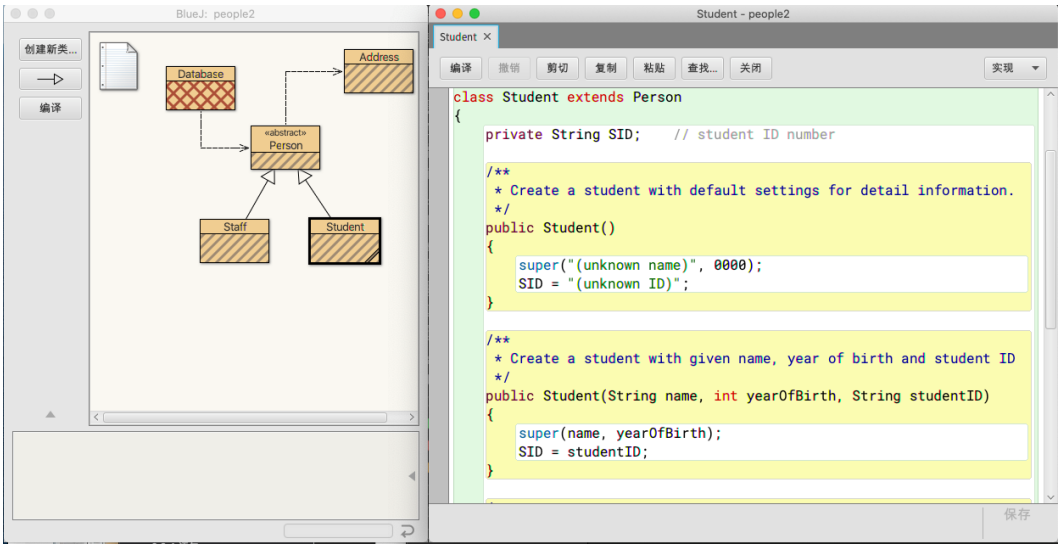


图 2-7 bluej 的用户界面

2.4 Java 虚拟机简史

在上面 2.1.2 小节安装 JDK 的时候，你可能会碰到这样的情况：当你使用 SDKMAN 安装 JDK，会发现有众多的版本供你选择，比如会出现下图让你无所适从。如果你对如此众多的 JDK 有好奇，你可以阅读本节内容，了解不同的 JDK 背后的故事。

以我的工作经验，了解主流的 JDK 在工作中也有用处。现在 Java 已经深深的植根于金融，电子商务，交通等关系国计民生的领域，能为这些领域提供服务的 IT 公司一般都是大型公司。这些大型公司一般都会有自己的 Java 虚拟机，比如企业采购 IBM 的服务，IBM 一般会使用自己家的 Java 虚拟机或者推荐合作伙伴的虚拟机。工作的时候不像在学校里，只需要知道一种虚拟机就够了，工作的环境千差万别，知道的越多，肯定会更从容一点。

我们写的代码最终还是要跑在 Java 虚拟机上，现在业界有很多种 Java 虚拟机，在维基百科的页面上，活跃的 Java 虚拟机还有 9 种，不活跃的 Java 虚拟机有 16 种，我相信还有更多没有统计在维基百科页面上的 Java 虚拟机。就目前的趋势来说，Java 虚拟机会越来越多，稍微大一点的公司，都要维护自己专有的 Java 虚拟机。通常作为程序员，在工作中可能遇到的虚拟机主要有三种：Oracle 公司的 HotSpot、IBM 的 J9 虚拟机、BEA 的 JRockit。这三种之中，又以 Oracle 公司的 HotSpot 最为常见。

Java 虚拟机和 Java 语言是双子星，他们相伴相生，一损俱损，一荣俱荣。在上一章 Java 版本介绍的部分，我们研究了 Java 1.0 开始到 Java 17 的版本，接下来，我想再探讨一下 Java 虚拟机的发展历程。

```
$sdk list java
================================================================================
Available Java Versions for macOS ARM 64bit
================================================================================
 Vendor        | Use | Version      | Dist    | Status     | Identifier
--------------------------------------------------------------------------------
 Corretto      |     | 20           | amzn    |            | 20-amzn
               |     | 20.0.1       | amzn    |            | 20.0.1-amzn
 Gluon         |     | 22.1.0.1.r17 | gln     |            | 22.1.0.1.r17-gln
               |     | 22.1.0.1.r11 | gln     |            | 22.1.0.1.r11-gln
 GraalVM       |     | 22.3.r19     | grl     |            | 22.3.r19-grl
               |     | 22.3.r17     | grl     |            | 22.3.r17-grl
 Java.net      |     | 21.ea.24     | open    |            | 21.ea.24-open
               |     | 21.ea.23     | open    |            | 21.ea.23-open
 Liberica      |     | 20.fx        | librca  |            | 20.fx-librca
               |     | 20.0.1.fx    | librca  |            | 20.0.1.fx-librca
 Liberica NIK  |     | 22.3.r17     | nik     |            | 22.3.r17-nik
               |     | 22.3.r11     | nik     |            | 22.3.r11-nik
 Microsoft     |     | 17.0.7       | ms      |            | 17.0.7-ms
               |     | 17.0.6       | ms      |            | 17.0.6-ms
 Oracle        |     | 20.0.1       | oracle  |            | 20.0.1-oracle
               |     | 20           | oracle  |            | 20-oracle
 SapMachine    |     | 20           | sapmchn |            | 20-sapmchn
               |     | 20.0.1       | sapmchn |            | 20.0.1-sapmchn
 Semeru        |     | 18.0.2       | sem     |            | 18.0.2-sem
               |     | 17.0.6       | sem     |            | 17.0.6-sem
 Temurin       |     | 20           | tem     |            | 20-tem
               |     | 20.0.1       | tem     |            | 20.0.1-tem
 Zulu          |     | 20           | zulu    |            | 20-zulu
               |     | 20.fx        | zulu    |            | 20.fx-zulu
```

图 2-8 使用 SDKMAN 安装 JDK 的部分列表

2.4.1 Oracle 的 HotSpot 虚拟机

1996 年，Sun 发布 Java 语言的同时，还发布了世界上第一个 Java 虚拟机。这个虚拟机之所以重要，是因为是世界上第一个 Java 虚拟机。缺点也很明显，这个 Java 虚拟机的执行效率非常差。

随后，Sun 公司开始对 Java 虚拟机进行改进。改进的成果是一款只可以运行在 Solaris 上的 Java 虚拟机，名叫 Exact VM。如果大家用 Exact VM 在 Oracle 的官网上搜索，仍然可以找到一些信息。大部分信息都是介绍 Exact VM 和 HotSpot 在编译时如何设置参数。

但是 Sun 没有再继续推广这款名为 Exact VM 的虚拟机，这款虚拟机只在 Java SDK 1.3.0 之前使用过，原因是一个更为先进的虚拟机出现了，这个虚拟机也是目前最为流行的 Java 虚拟机，名字就叫 HotSpot。

HotSpot 并不是 Sun 公司做的，而是一家叫 Longview Technologies 的产品，这家公司的创始人，一个叫 Urs Hölzle，一个叫 Lars Bak，两人从 Sun 离职后创业。为了重新获得这两个员工和他们的产品，Sun 收购了这家公司。

该产品原本是针对 SmallTalk 语言做的虚拟机，被收购以后重新设计，转而支持 Java 语言，跟随 Java 1.2 的同时发布。

Java 的 1.2 版本，有 3 个 Java 虚拟机，一个是 Sun 公司第一版的虚拟机，一个是 Solaris 上的 Exact VM，还有一个就是 HotSpot。默认的 Java 虚拟机是最慢的那一个，直到 1.3 版以后，才将默认的虚拟机换成 HotSpot。后来 Sun 公司被 Oracle 公司收购，HotSpot 虚拟机也就成为了 Oracle 的产品。

在早期，只有 Sun 公司有 Java 虚拟机，其它公司没有“版权”来染指 Java。直到 1998 年，Sun 公司成立了 JCP（Java Community Process）组织，这个组织希望让越来越多的公司参与进来，大家都分一杯羹。当然了，Sun 公司肯定是想做分羹的人，大家做 Java 可以，但是要通过 TCK（Technology Compatibility Kit），翻译成中文叫技术兼容性测试。

在 TCK 的约束下，不少公司通过了 TCK，通过以后，自己的虚拟机就有资格叫 Java 虚拟机了。在这个背景下，有很多公司推出了自己的 Java 虚拟机，最著名的厂商有 IBM、BEA、Microsoft 和 Apache。

目前它是 Java Development Kit（JDK）中默认的虚拟机实现之一。HotSpot 虚拟机被广泛应用于 Java 应用程序的开发和部署，包括企业级应用程序、Web 应用程序、移动应用程序等。除了作为 Java 应用程序的运行时环境，HotSpot 虚拟机还被广泛用于 Java 开发工具的实现，例如 Eclipse、NetBeans、IntelliJ IDEA 等。此外，HotSpot 虚拟机还被用于其他语言的实现，例如 JRuby、Jython 等，使它们能够在 Java 虚拟机上运行。

2.4.2 IBM OpenJ9 虚拟机

IBM J9 名称的由来

IBM 官网上介绍，这款 JVM 最早是由 IBM Ottawa 实验室一个 SmallTalk 的虚拟机扩展来的。那时候，这个虚拟机有一个 bug 是因为 8k 值定义错误引起，工程师们花了很长时间终于发现并解决了这个错误，此后这个版本的虚拟机就被称为 K8 了。于是，后来出

现的支持 Java 这个版本的虚拟机就被称为 J9 了。

与 Sun 公司收购 HotSpot 类似，IBM 的 J9 也是收购的。1996 年，IBM 收购了一家叫 OTI（Object Technology International）的公司，该公司有虚拟机产品。更巧合的是，这个产品最初也是为 SmallTalk 语言设计的虚拟机，后来在 Java 流行以后，才改为支持 Java 虚拟机。

2017 年，J9 变成了 IBM 主导的 Eclipse 组织的一个项目，名字已经改成了 Eclipse OpenJ9。现在建立在 Eclipse 开放运行时项目（OMR）之上，IBM 的专有项目大部分基于此，它完全兼容 Java 认证。

目前，OpenJ9 虚拟机在云环境、大规模应用程序、低延迟应用程序、高密度部署和跨平台支持等方面具有优势，其官网的介绍中，着重强调了其低内存占用，快速启动，高吞吐的特征，适用于一些特定的应用场景，例如云原生应用程序、高并发应用程序、容器化、微服务等。

2.4.3 BEA 的 JRockit

BEA 公司目前已经被 Oracle 收购，所以，Oracle 拥有三个最主要的虚拟机中的两个。

BEA 公司是著名的 Java 中间件公司，曾经是 IBM 公司最重要的竞争对手，产品是与 IBM WebSphere 竞争的 WebLogic，IBM 有自己的 Java 虚拟机，BEA 也想有自己的 Java 虚拟机。

收购是最省时间的方式，BEA 就收购了 Appeal Virtual Machines 公司，这个公司的产品是 JRockit。

从名字可以猜一下，应该和火箭一样快吧。JRockit 的特点就是速度快，针对的市场是用专门硬件，专门服务器的商业用户，不针对消费者市场。这个公司宣传自己的产品是：“World's Fastest Java Virtual Machine” ，世界上最快的 Java 虚拟机。

BEA 被 Oracle 收购以后，就被 Oracle 暂停了，Oracle 没必要同时拥有两个 Java 虚拟机。在 2011 年，Oracle 宣布 JRockit 可以免费使用，但是由于多年没开发，JRockit 最高只能支持 Java 6。Oracle 承诺，会将 JRockit 的优秀特性在 OpenJDK 实现。

2.4.4 微软的 Java 虚拟机

在使用 SDKMAN 安装 JDK 的时候，会看到有微软出品的 OpenJDK，实际上，微软至少出过两个 Java 虚拟机，第一次是上世纪 90 年代，另一次是 2021 年。

上世纪 90 年代微软出过 Java 虚拟机，并且性能还相当不错，在 1997 和 1998 年获得过

《PC Magazine》杂志的编辑选择奖，在 1999 年宣称自己是 Windows 上最快的 Java 虚拟机。

微软为什么会花大力气帮助 Sun 来实现 Java 虚拟机呢？答案当然是想控制 Java 了。因为 Java，Sun 在 1997 年控告微软违反协议滥用 Java。直到 2001 年，微软败诉，赔偿 2000 万美元给 Sun 公司。

后来微软模仿了 Sun 的 Java，强推自己的 Visual J++，官司输了以后，又开发了 J#和 C#，再推广 Java 虚拟机对微软已经没任何正面意义，所以，微软的 Java 虚拟机在 2003 年就停止开发，最晚支持到 2007 年。

有个大翻转比较有趣，Sun 在赢了官司以后，按照协议，Windows XP 不能预装 Java 虚拟机。Sun 此时才恍然大悟，如果不预装 Java 虚拟机，那么对于推广 Java 百害而无一利，于是又开始劝微软继续装 Java 虚拟机。

那时 Sun 公司已经发布了 Java 1.4，微软只肯在 Windows XP Service Pack 1 中包含一个 1997 年的，基于 Java 1.1.4 版本的 Java 虚拟机。

最近的一次是 2021 年，根据官网介绍，微软大量的使用 Java，从 Minecraft 游戏到 LinkedIn 网站，再到微软的云计算平台 Azure，因此，微软发布了自己维护的 OpenJDK。如果你是 Azure 用户或者是 Minecraft 玩家，推荐使用微软版本的 OpenJDK，该 JDK 已经集成在 Azure Cloud Shell 中。

2.4.5 Apache 的 Harmony

Apache 也有 Java 虚拟机，但是却不能称之为 Java 虚拟机，前面讲过，想宣传自己为 Java 虚拟机，要先得到 JCP 主导的 TCK 兼容性测试。Apache 得不到这个认证。为什么会得不到这个认证呢？主要还是理念问题。

JCP 的执行委员 Doug Lea 如此评价 Oracle： “虽然 Sun Microsystems 已经制定了可以推动 JCP 创新的规则，但是 Oracle 并不理会这些规则，JCP 也许会成为任 Oracle 摆布的傀儡。”Apache 组织是个非盈利组织，Oracle 是个以盈利为主要目的公司。理念谈不拢。Apache 希望 Java 能够不受任何公司的控制，让 Java 完全开源，做了名为 Harmony 的 Java 版本。

后来，Java 的创始人 James Gosling 也建议 Oracle 应该成立一个独立的 JCP 来控制 Java，但是 Oracle 不为所动。2010 年，JCP 开会讨论 Java 7 和 Java 8 的方向，这次会议双方的矛盾最终爆发，Apache 宣布退出 JCP，Oracle 乐见其成。

本来 IBM，Apache 和 Google 是推动 Harmony 的三巨头，但是 IBM 却发表声明说今后将退出 Harmony，以最大的努力推动 Oracle 的 OpenJDK 的发展。随后，IBM 辞去了 Harmony 项目主席的职位。Apache 一方面无法得到 TCK 认证，另一方面，最坚定的支持者之一 IBM 也跑去了 Oracle 的阵营。在这种境地下，2011 年 12 月 16 日，Apache 宣布取消 Harmony 这个 Java 虚拟机的开发。

Apache 有一个坚定的支持者，就是 Google。Google 早期的安卓系统使用了 Apache Harmony 的类库实现，但是后期的安卓系统最终还是切换到了 OpenJDK 的类库。Apache 的 Harmony 目前仅作为历史资料供大家阅读，虽然已经不更新了，但是作为一种开源的精神，我觉得值得为它写一段文字。

2.4.6 Google 的 Dalvik 和 ART

安卓是 Google 最大的资产之一，凭借安卓，Google 掌握了手机市场。Sun 公司虽然一直想把 Java 推广到手机中，但是应该没有想到 Google 把这事做成了。

在 Apache 宣布不再继续 Harmony 虚拟机以后，Google 从中获取了大量的代码添加到 Google Android SDK 中。早期安卓中不可或缺的组件之一叫 Dalvik，它是与 Java 虚拟机（JVM）类似，它提供了一个运行环境，用于执行用高级编程语言编写的应用程序。然而，Dalvik 并不完全是 JVM，而是专门为安卓操作系统设计的一种独特的 Java 虚拟机。

Dalvic 名字的起源

Dalvik 由 Android 团队中的 Dan Bornstein 编写的，名字来源于他的祖先曾经居住过的小渔村达尔维克（Dalvik），位于冰岛埃亚峡湾。

从 Android 5.0 开始，Android Runtime（ART）取代 Dalvik 成为 Android 应用程序的默认运行时。ART 使用即时编译器（JIT）来提高性能与现有 Java 代码的兼容性，还提供了内存管理、安全性和线程管理等基本功能，但与 JVM 不同，它并不解释和执行 Java 字节码，而是直接执行本地机器码。可以说 Android Runtime 和 Java 虚拟机都是用于运行 Java 代码的软件平台，但它们并不是完全相同的概念，因为它们的实现方式和提供的功能有所不同。

2.4.7 虚拟机小结

前面讲了这几个虚拟机，选择哪种 Java 虚拟机取决于你的具体需求和应用场景。总结一下以上 Java 虚拟机及其特点：

1. Oracle 的 HotSpot

Oracle JDK 是 Java 官方发布的 JDK，它是最广泛使用的 Java 虚拟机之一，具有稳定性和性能优势，同时支持最新的 Java 语言特性和 API。OpenJDK 是一个开源的 Java 开发工具包，它是 Oracle JDK 的开源版本，具有与 Oracle JDK 类似的功能和性能。OpenJDK 提供了免费的使用和分发许可证，因此被广泛用于各种开源项目和商业应用中。

2. IBM 的 OpenJ9

IBM JDK 是由 IBM 公司开发和发布的 Java 开发工具包，它具有良好的稳定性和可伸缩性，适合于大型企业级应用和高负载的 Web 应用。IBM JDK 需要商业许可证，但也提供了免费的开发者版。

3. 微软的 OpenJDK

如果你使用微软的云计算平台 Azure，那么该 OpenJDK 是个非常好的选择。

4. Apache 的 Harmony

虽然该项目已经取消了，我写这一段是出于情怀，希望大家了解一下。如果能再多了解一下 Apache 软件基金会这个非赢利的开源组织就更好了，该组织致力于推广开源软件和开源文化，我们用到的大量的开源软件都是该组织提供的。

5. Google 的安卓系统

前文已经说了，这并非严格意义上的 Java 虚拟机，但是很多学 Java 的程序员，很大比例要给安卓开发相关的软件，毕竟安卓手机是目前市场上占有率最高的手机。

2.5 脱口秀：程序员，编译器，虚拟机

前面讲了 Java 语言，javac 和 Java 虚拟机，了解程序员，javac 和 Java 虚拟机之间的关系非常重要。下面我编了一个小故事，希望对理解这三者之间的关系有帮助。这个小故事的名字叫《表扬与自我表扬》，栋哥是程序员。

栋哥:有件事情我想和两位说一下,项目成功上线以后,咱们会有一个名额去东京旅游。我觉得咱们仨个这几年来一直在一起工作，我非常的感谢 javac 一直以来给我写的程序检查错误，给 Java 虚拟机输出最终执行的文件。当然了，javac 很棒，Java 虚拟机也非常棒，能一直稳定的运行咱们最终的项目。没有 javac 的检查，没有 Java 虚拟机的运行，我肯定没法完成任务，这些年来，我一直拿两位当兄弟看待！不过，我还是觉得我最有资格去东京，毕竟没有我，两位工作的再好，也是无本之木，无源之水。两位应该没什么意见吧？

javac：很好，一起工作有难同当的时候是兄弟，有福同享的时候就不是兄弟了，很好！不瞒两位说，如果不是我 javac 在两位之间做翻译，你们知道你们说的什么么？栋哥，不瞒你说，你写代码，如果有我输出的字节码质量的百分之一，我们还用天天加班么？你应该心里有数吧？从标点符号出错，到变量名出错，还有变量类型中的错误.....这么说吧，如果不是我屏蔽了这些错误，你的代码如果直接给 Java 虚拟机去运行，他能天天死机。没有我，你们两个只能是最陌生的陌生人。

Java 虚拟机：两位真是王婆卖瓜，自卖自夸啊，你们可真是厉害。我只想纠正两位一下，只要项目一上线，我就要 24 小时工作，你们工作再多，也不过是 996 么。两位既然自视甚高，我就不去旅游了，我还要工作呢。我只是提醒两位，没有人比我干活多。刚刚 javac 的发言，真是要笑死我了，你还当我们的翻译，对我来说，你就是个传话筒，栋哥你可以直接写字节码给我，咱们两个直接交流，还有 javac 说的那些错误处理，我都能处理。只要有错误，我就扔给你 ClassCastExceptions，咱们之间，不用有些人传话，有道是，传钱就怕传少了，传话就怕传多了啊。

javac：哈哈，Java 虚拟机你可真是可笑，你和栋哥直接交流，不是我说大话，栋哥用简单的编程语言写的代码都错误百出，你要是让他用字节码和你交流，他一天也写不出一行来。用字节码写程序，那就相当于搞活字印刷，先得让栋哥去学习如何用铅字排版，不，还没有铅字，得让栋哥学习如何将硫化铅提炼成铅。

栋哥：别吵了，没想到我在你们眼里是如此不堪。我并没有否认两位的功劳。我和两位不一样，你们俩只懂 Java 这一门语言，我是程序员，我还懂得 C 语言，Python 语言，PHP 语言.....相比于两位来说，我可以说是站得高，看得远，用其它的语言中，其实也能实现两位的功能，比如说在 C 语言中.....你们要干什么，打人是不对的，有话说话，不要动手啊，君子动口不.....救命啊，打人了.....

2.6 常见问题

2.6.1 bytecode 是什么？

我在 IBM Developer 上曾经看过一篇文章，文章的名字叫《Java bytecode: Understanding bytecode makes you a better programmer》。在这篇文章里，作者说：“对 Java 程序员来说，理解 bytecode 类似于让 C/C++程序员理解汇编语言”。

> Kathleen Booth 与汇编语言
>
> Kathleen Booth 女士在 1947 年发明了汇编语言，并且设计了伦敦大学第一个汇编程序与自动解码。
>
> 汇编语言是一种非常低级的语言，一般不能在不同的平台之间移植。可以把汇编语言看作是机器语言的助记符，在不同的设备上有不同的机器语言指令集。

字节码（bytecode）是一种中间语言（intermediate language），它通常是由高级编程语言编写的源代码经过编译器编译后生成的，但不是直接翻译成机器代码，而是翻译成一组可执行的指令序列。“bytecode” 里的 byte 有含义，代表 byte，也就是 8 个 bit，每条指令的长度通常是一个字节（8 位）。8 bits 会产生 256 个组合，Java 虚拟机最多支持 256 个操作符。目前用了大概 80%。可以把字节码理解为 “虚拟机的机器码” 。

javac 的工作就是把 Java 源代码转化成字节码。

不管是在 Linux 上生成的字节码，还是在 Windows 上生成的字节码，都是一模一样的，这构成了 java 跨平台的基石。

除了 Java，还有很多编程语言也采用了字节码技术，比如 Python、Ruby 等。使用字节码可以使得跨平台执行程序变得更加容易，因为不同的计算机平台可以使用不同的虚拟机来解释和执行字节码。

2.6.2 javac 是编译器么？

是编译器，但不是 C 语言所采用的 Gcc 那样的编译器，C 语言的编译器直接产生机器码。机器码是由二进制代码组成的，计算机 CPU 可以直接执行的指令。javac 不产生机器码，而是产生字节码。

在 Java 虚拟机中，负责产生机器码的是 JIT（Just-In-Time）运行时编译器。从这个角度，可以认为 Java 有两种编译器：一个是 Java 字节码编译器，一个是 JIT 编译器。

JIT 编译器的工作原理是这样的：软件在运行的过程中，大部分时间用来运行少量的代码。当软件在解释模式下执行的时候，编译子系统会时刻监控软件的运行，并观察代码中执行最频繁的部分。在整个分析过程中，会捕获一些重要的信息，再根据这些信息进行优化，优化的原则是：把运行最频繁的部分，编译成机器码。这样一来，Java 代码就拥有了可以和 C/C++相媲美的性能。

目前，主流的编程语言都已经支持 JIT 技术，像 PHP 8，JavaScript，Python 都已经开始引入 JIT 技术，这也是开源技术的魅力所在，只要一项技术被证实可靠，就会被其它语言采用。

另外，javac 是用 Java 语言写的，现在主流的语言都用自己写自己的编译器，甚至还有个专门的名称叫编译器的自举（bootstrapping）。如果大家对此感兴趣，可以在 wikipedia 上通过搜索 bootstrapping 查看一下有哪些编程语言可以自举。

其实，这也隐含了一个鸡生蛋和蛋生鸡的问题，既然 javac 编译器是 java 写的，那第一个 javac 是谁编译的呢？这里就又涉及一个安全问题，学术上称之为基于信任的攻击（Trusting Trust Attack）。Unix 操作系统的作者 Ken Thompson 在 1983 年在获得图灵奖的时候，发表了著名的演讲《反思对信任的信任》（Reflect on Trusting Trust），他讲了如何在 C 编译器上植入了后门的一种可能，在 2009 年，该方法真的被一些人利用并发起了攻击，如果想详细了解这次攻击，请搜索 “A Thompson hack virus is found in the wild” 。

2.6.3 不同的 JDK 许可证有什么区别？

主流的 Java 虚拟机都是开源的，除了基于 Eclipse 协议的 IBM J9，其它的大部分都是由从基于 GPL 授权的 HotSpot 衍生出来的。

Oracle 是科技界最懂法律的公司。从 Java 9 开始，Oracle 收购 Sun 后，让情况变得复杂了。虽然 Java 9 也是来自 OpenJDK 的代码库，但是它是专有的，就算你能看到源代码，但是不是开源软件。如果你想对 OpenJDK 贡献代码，你就要和 Oracle 签署一个许可证，该许可证允许 Oracle 对 OpenJDK 的 GPL 和 Oracle 的专有许可证进行双重许可。

Oracle 和 OpenJDK 的真正区别是许可证。从 2017 年开始，Oracle 承诺会对 OpenJDK 提供支持，对个人和中小公司来说，OpenJDK 应该是足够的。但是对商业公司，许可证可能有所不同，比如你在自己公司范围之外重新分发二进制文件是违法的，比如发布了一个包含 Java 二进制的 Docker 映像给外部，有可能就触犯了使用条款。如果没有许可，你也不可以给 Oracle 的二进制文件打补丁，只有买了 Oracle 的商业合同才可以进行补丁修复。

正是因为这些对商业公司的限制，为了不受制于人，大型公司都有自己的 OpenJDK 版本，不过，投入的维护成本和技术成本，不是一般公司能承受的。

2.6.4 Java 虚拟机只能运行 Java 语言么？

以前是，现在不是。

Java 虚拟机和 Java 语言现在齐头并进，虽然 Java 编译的字节码只能跑在 Java 虚拟机上，但是 Java 虚拟机并不是只跑 Java 的字节码。目前来说，Java 语言和 Java 虚拟机在一定程度上是独立的，Java 虚拟机也许应该换个名字，比如叫 “多语言虚拟机” 更贴切一些。

现在 Java 虚拟机可以执行任何语言生成的合法的文件，只要符合 Java 虚拟机的规范就好。比如目前比较热门的 scala 语言用其编译器 scalac 生成的字节码，完全可以运行在 Java 虚拟机上。其它的诸如 Kotlin,、Groovy、Clojure 语言都可以运行于 Java 虚拟机之上。

Java 虚拟机如果要加载类，会先验证它们是不是符合规定的格式，如果符合，就允许其执行。本书不会对类文件的规范讲的太深入，只讲一个有点意思的事情吧。

在 Windows 上，可以用扩展名来识别文件类型，在 Unix 环境下，则要使用一些魔数（magic number）来识别。每个 Java 类文件都以魔数 0xCAFEBABE 开头的，这四个以十六进制表示的字符表示当前文件的类型。大家看到没有，最后四个字母是 BABE，当年 Java 可能没想到会这么火，也没想到这个单词放在今天的舆论下，会涉及性别歧视。现在有转机了，在 Java 9 中，为模块文件（JIMAGE）引入了新的魔数 0xCAFEDADA。也不知道“爸爸（DADA）”这个单词会不会在未来的日子里，也有歧视的意味。

比起社会的文化的变革速度，Java 的更新速度其实很慢的，所以我们要长期使用到 0xCAFEBABE 这个魔数，不过，如果你不用十六进制编辑器打开二进制文件，是不会看到这个涉嫌性别歧视的字串的。

段子

上文讲了 Windows 可以用后缀名来识别文件类型，其实在 Linux 中也可以如此。比

如 Shell 文件的后缀就是 sh 结尾。

我有个同事很爱开玩笑，在一次会议上，他开玩笑说这个以 sh 结尾的文件，代表是在上海（shanghai）的服务器上运行，我们都了解他一直不太正经，喜欢开玩笑，因此没人放在心上，都没有讲话。

到了领导发言的环节，领导着重把这件事拿出来强调，领导说这个想法很好，以后可以在全公司推行，这样就不用担心软件会跑错服务器了。

我们再一次失语。

2.7 总结

本章主要的内容有：

- 写了第一个 java 程序
- 安装必须的 JDK 开发环境
- 分析了第一个 Java 程序的结构
- 如何给 Java 代码写注释
- 选择适合的开发环境，明确自己的需求，是使用 IDE 还是文本编辑器
- 简单介绍了如何调试 Java 程序以及反汇编字节码
- 简要介绍了几种 Java 虚拟机发展的历史
- 明确程序员、编译器和虚拟机之间的关系

2.8 思考拓展

1. 程序可以运行在计算机上，也可以运行在大脑里。伟大的计算机科学家图灵（Alan Turing）在计算机发明之前就已经运行过计算机程序了，他和他的朋友阿里克·格兰尼（Alick Glennie）进行了一场国际象棋比赛。他的朋友正常下棋，他则是用计算机的方式在纸上计算如何下棋。这场棋耗时几个月，最后图灵输掉了比赛。图灵去世多年以后，人们又复现了这场比赛，请大家在网上搜索有关材料，研究一下计算机发明之前的“人工智能”。
2. 国际象棋在计算机发明中扮演了重要的角色。除了上一题提到的图灵，还有 Unix 的作者 Ken Thompson 也热衷于用计算机研发“人工智能”象棋程序，在 1980 年，他与朋友 Joseph Henry Condon 合作研发出来的 Belle 程序，获得了世界冠军。该程序

蝉联了 1978，1980，1981，1982 和 1986 年的美国象棋冠军。协助 Richard Stallman 发起开源运动（GNU）的 Stuart Cracraft 也是象棋爱好者，他写了 GNU Chess 这个广泛存在于 Linux 上的游戏程序，由于是开源的，如果大家碰巧要学习 C/C++语言，可以查看其开源引擎 Fruit。希望大家熟悉一下象棋，尤其是中国象棋的规则，思考一下，如果让你写一个 Java 版的中国象棋软件，你会怎么做？[3]

[3] 初学者可能会觉得，这个可太难了，我不会。其实呢，按我的经验，只要有适度的压力，就能完成你认为不可能完成的任务。我读书的时候，也觉得写不了大型软件，毕竟，我本科最多也就写过几千行级别的小代码。结果我读研究生的第一周，导师让我读 SQLite 的源码，源代码大概有十几万行，那时候，我连 C 语言的指针都搞不太明白，像 B-Tree 这种数据结构，也就会考个试。导师让我每两周写一份报告，我花了一个半学期的时间，写了十几份阅读代码报告，每一份报告，导师都会过目，然后半小时内就把我问的哑口无言，然后搞了十几次后，竟然都读懂了。读懂十几万行代码以后，给了我极大的信心，这是最大的收获。

3. 面向对象

夫以铜为镜，可以正衣冠；以古为镜，可以知兴替；以人为镜，可以明得失。

—— 《贞观政要》

历史是一面镜子，本章通过研究面向对象的历史来学习面向对象编程，如果没有面向对象编程的经验，这一章中碰到不懂的问题直接跳过即可，在后续的章节会详细介绍。

对 C++或者 Python 语言来说，可以使用面向对象编程，也可以不使用面向对象编程。但是对 Java 来说，没有面向对象就没法编程。从这个意义上来说，Java 是一种非常纯粹的面向对象编程。那面向对象有什么作用呢？James Gosling 在接受采访时说： “面向对象可以使你把一个系统分解开，系统的每个部分都能够被分解，这样对更新、调试等很多事情都有帮助。”[1]

既然面向对象如此的重要，我们不妨来研究一下面向对象的历史。

3.1 第一门面向对象编程语言

2001 年，美国计算机学会（ACM）将代表计算机界最高奖的图灵奖颁发给了两位挪威计算机科学家，一位是 Ole-Johan Dahl，另一位是 Kristen Nygaard，以表彰他们 “通过设计编程语言 Simula 1 和 Simula 67，创造了面向对象编程的基本概念” 这一伟大成就。

上世纪 60 年代，Simula 语言从诞生之初，就具备了现在大部分面向对象语言所具备的特性，包括但不限于继承、垃圾回收、动态绑定、错误检查等功能，随后，将详细介绍这几个重要的特征。虽然那个时候还没有“面向对象”这个名称，但是在这个名称出现之前，面向对象的概念已经存在了。

让我们看看当年面向对象语言是如何发展而来的吧。

3.1.1 面向对象的发源项目

1952 年，挪威政府决定成立一个叫 NCC（Norwegian Computing Centre）的组织，把全国零散的计算机资源整合起来。该项目的负责人叫 Jan Garwick，之前在奥斯陆大学当教授，

[1] 来自《Masterminds of Programming》第十二章对 Java 创始人 James Gosling 的访谈。

他招了两个人来当助手。

其中一个助手是 1952 年加入的新兵，他的名字叫 Ole-Johan Dahl，他的任务是给 Mercury 计算机写一个编译器，这个编译器叫 MAC（Mercury Automatic Coding）。

另一个助手也是个士兵，是 1948 年入伍的 Kristen Nygaard，他被安排了另外一个项目，该项目是研究一个开放性的问题：如果有一天挪威要制造核武器了，应该如何提前模拟核武器的爆炸威力？

Kristen Nygaard 以模拟核武器为题写了一篇名为《Theoretical Aspects of Monte Carlo Methods》的论文，随后在军方成了专职的核武器研究员。虽然他的专职工作是研究核武器，但是他兴趣广泛，他的研究范围从单纯的核武器扩大到世间万物，尤其是人力资源方面，他想知道人能不能也用计算机来建模呢？这是一种将社会工程学和管理学相结合的产物，Kristen Nygaard 研究人的行为并加以预测和控制。

后来 Kristen Nygaard 与军方发生了一些摩擦，他于 1960 年离开军方并加入前文提到的 NCC。相比于军方，NCC 更多倾向于民用研究，在这里，他有更大的空间可以自由的发挥。在此，他的兴趣有所转变，他思考能不能将他在军用领域所做的工作转化为民用领域？

一封保留下来的信如实记录了当时的情况，1962 年，Kristen Nygaard 给法国计算机科学家 Charles Salzmann 写了一封信，在信中他透露说他已经有了完整的模拟现实世界的概念，还没动手做语言的编译器，他想等语言先设计好了再动手做这个工作。在信中，他还透露，他认识一位写程序的天才，两人都对这个想法很乐观。在这封信里他提到的那位天才就是他未来的合作伙伴 Ole-Johan Dahl。

1963 年，两人开始研究并实现这个创意，两年后，于 1965 年完成第一阶段工作。这个阶段的成果在当时被称为 Simula，为了区分，后被约定俗成称之为 Simula I。

随后的两年，两人继续研究，于 1967 年发表了第二版本的语言，也就是后来的 Simula 67，Simula 67 已经有了面向对象的雏形，几个与面向对象相关的概念已经被提出。我们看看这几个重要的概念吧。

3.1.2 Simula 核心概念

1. 继承

Ole-Johan Dahl 曾经这样写过：“增量的抽象，对已经抽象过可以加上一个前缀 C，只要有这个前缀 C，就可以使用其所有的属性。这就是继承的雏形，虽然还没有正式叫继承。”

2

其实对任何语言来说，都可以复用代码，至少可以通过简单的“复制粘贴”来完成代码的重复使用，但是这样复用的代码并不优美，而且还很难维护。如果能够直接使用别人已经完成的代码，或者自己先前抽象好的代码，而不是自己再重新开始，那么将会有效的降低工作量。Simula 语言在这个方面进行了探索。

Java 也借鉴了这个思路，Java 围绕类的概念做了很多工作，其中最重要的概念之一就是：“继承”。望文生义，继承的表面意思就是从“长辈”那里得来一些东西，在 Java 中，基本也是这样的，采用已有的类，无需改动这些类，就能获得相应的功能，这种方式在 Java 中也叫“继承”。

2. 内存回收机制

Ole-Johan Dahl 和 C.A.R. Hoare 合作写过一篇论文《Hierarchical Program Structures》，在这篇论文中，他们介绍了当时的想法：“在实现 Simula 的时候，借鉴了 Algol 60 这个编程语言的实现方法，并且对 Algol 60 的 block 进行了必要的改进，将 block 视为数据，其它的程序可以使用这些 block，这后来演化成了类的使用方法。”

Simula 语言的两位作者写的《The Development of the Simula Language》里，也提到了这一点：“改进了存储机制，引入了内存垃圾回收机制。也许很多语言都意识到了 Algol block 的威力，但是第一个意识到并且做出垃圾回收的语言是 Simula，这在当时是一件了不起的创举。虽然内存垃圾回收很难，还有可能对语言的运行速度产生影响，但是为了以后程序员的方便，这点性能缺失不算什么。Simula 还独创了一种名为二维空闲列表的方式来负责内存垃圾回收。”

在计算机科学中，内存泄漏（Memory leak）是一种常见的 bug，由于疏忽或错误造成程序未能释放已经不再使用的内存。内存泄漏并非指内存在物理上的消失，而是应用程序分配某段内存后，由于设计错误，导致在释放该段内存之前就失去了对该段内存的控制，从而造成了内存的浪费。

内存泄漏会因为减少可用内存的数量从而降低计算机的性能。最终，在最糟糕的情况下，过多的可用内存被泄露掉从而导致全部或部分设备停止正常工作，最终导致应用程序崩溃。

现在主流的编程语言如 Java 就提供了内存回收机制，这对保证软件的可靠性和安全性非常有好处。如果大家有 C/C++编程经验的话，很可能为了一个内存泄露花费数小时甚至数周来查找。有了内存回收机制以后，将节省大量的编程与调试时间。

[2] Ole-Johan Dahl 写过一篇名为《Transcript of discussant's remarks》的文章，发表在 1981 年 R. L. Wexelblat 写的《History of programming languages》中，488-490 页。

3. 动态绑定

Simula 开创性的使用了动态绑定技术，虽然当时的名字不叫 dynamic binding，而是叫 virtual。

刚开始的时候，Simula 所有的属性都是静态的。Ole-Johan Dahl 是在最后一分钟才决定做成动态的，如果对属性定义为 virtual，那么就可以动态绑定了。后来的语言如 Smalltalk 更纯粹，直接把所有的属性和方法都定义为 virtual，在 C++中也用相同的关键字 virtual。

Java 语言同样借鉴了 Simula 语言，在默认情况下是可以动态绑定的，如果使用了 final 这个关键字，就是静态绑定从而阻止被覆盖。

4. 错误检查

由于 Simula 语言最初研究的是核爆炸，安全性显得特别重要，与其它同期的语言不同，Simula 格外重视安全，毕竟核爆炸可不是只让计算机死机那么简单了。Simula 在设计之初对错误十分重视，Simula 设计了两种错误检查，一种是编译时检查，一种是运行时检查。

现在我们把这种机制叫做类型安全（type safety）。

类型安全确保代码不会对底层对象执行任何无效操作，它确保任何变量访问只能以明确定义和允许的方式访问其授权的内存位置。由于不同的编程语言对程序员思维的影响，在讲到类型安全的时候，不同程序员对这个概念的理解有些出入，因此讲出来的意义也就十分的宽泛。

在 Java 中，类型安全不仅是对一个 A 类型的变量赋值了一个 B 类型的值，[3]更多的是考虑类的层面，比如每个对象创建后都要初始化，外部对类的访问要受相关的限制，对象抛出异常之前要先将自身重置到合法状态等等。类型安全的思想贯穿于整个编程中，而不仅仅在变量赋值这种技术细节。

不止 Java 借鉴了 Simula 的错误检查，目前像 Haskell 语言中的 inspect 方法或多或少都是从这里学来的。

3.1.3 Simula 的普及

[3] 实际上，我认为在 Java 的类型安全做的并不完美，我们可以对两种不同的引用类型进行转换，在运行时 JVM 可能会失败，从而抛出 ClassCastException 异常。。

Peter Wegner

Peter Wegner 是一位出生于 1932 年的英国科学家，对面向对象编程有很大的贡献。在 1999 年，他被奥地利授予奥地利科学与艺术荣誉奖，在去伦敦领奖的路上，出了车祸，昏迷了好久之后才苏醒，但是有了严重的后遗症。

如果用 Peter Wegner 在 1987 年对对面向对象下的定义：object-oriented＝objects＋classes＋inheritance 来衡量的话，至此，面向对象最重要的几个要素在 Simula 语言中都已经有了雏形。

如果只有技术，没有推广，Simula 可能仍然会像世界上绝大部分的编程语言一样无闻。著名的科技史作家、宾夕法尼亚大学历史学教授 Thomas Parker Hughes 在爱迪生的传记《Networks of Power: Electrification in Western Society》里这样评价：“像爱迪生这样伟大的发明家有这样的特征，为了达成目标，他们不仅有超越普通人和科学家的认知，还能够综合利用自己的社会关系，政治资源，商业手段。”

强有力的推广自己的产品或理念是一种优秀的品质，这种品质在 Simula 两位创始人身上体现的也很明显，两位创始人不仅可以埋头搞科研，还和爱迪生一样，是商业上的天才。他们不仅有雄心壮志，同样也有动员能力、商业运作水平，这两位创始人靠着自己无与伦比的谈判技巧和推广能力，把 Simula 语言从挪威推广到了整个英国、法国、美国，最后影响了全世界。

虽然这是一本编程的书，我还是希望大家能学到比编程更多的东西。要记住，酒香也怕巷子深。能够把一门编程语言推广起来，可不是一件容易的事情。当你写出一款优秀的软件，或者创造了一个编程语言，只是成功走完了第一步。如何推广软件或者语言，让别人用你的软件或者语言，是更重要也是更困难的一步。接下来学习一下 Kristen Nygaard 和 Ole-Johan Dahl 是如何推广 Simula 的吧。

3.1.4 推广 Simula

挪威不是计算机强国，如果要推广自己的 Simula 语言，就要先在计算机上运行。在当时，计算机是非常昂贵的，NCC 有意从英国购买 KDF-9 这台大型计算机，但是这台计算机的价格实在太贵了，远远超出了 NCC 的预算。当时美国已经制造出了 UNIVAC 这样一台机器， NCC 想购买这台计算机。

Kristen Nygaard 表现出了天才般的谈判技巧，他找到了 UNIVAC 在欧洲的负责人 James W. Nickitas，经过了一次谈判，他说服了对方，对方同意把 UNIVAC 打 5 折。还敲定了下一次会谈，要和 UNIVAC 软件的灵魂人物 Robert Bemer——前 IBM 计算机的核心之一——坐下来谈谈软件和编程语言的事情。

UNIVAC 的软件核心 Robert Bemer 在和 Kristen Nygaard 谈过以后，UNIVAC 不仅可以以半价卖给 NCC，而且还给 Simula 项目带来了一笔赞助，还有，Kristen Nygaard 成了北欧

UNIVAC 的销售代理。这样的谈判水平，已经可以说是高手中的高手了。

这种谈判并不是只发生了一次，在 Kristen Nygaard 推广 Simula 的时候，这种事情屡次上演，他总是能把最难谈的谈判像谈天一样搞定，很快 Simula 就可以在 UNIVAC，IBM360/370， CDC 6000， DEC System-10 等一系列当时主流的机器上运行了。在科学计算和仿真领域，Simula 的影响力巨大。不仅如此，施乐帕克研究中心制造的第一台真正意义上的个人计算机 Xerox Alto 上的大量软件，比如第一个图形化的文本编辑器 Bravo，第一个图像编辑器 Drawing，都是使用 Simula 开发的。后来，该计算机的图形用户界面启发了苹果公司和微软公司。

Simula 影响了工业界和学术界，也培养出了一大批 Simula 的拥趸，比如 Smalltalk 的作者 Alan Kay，C++的作者 Bjarne Stroustrup 都曾声称自己的语言深受 Simula 的影响。Simula 的基本思想和设计为现代面向对象编程语言的核心概念，被广泛应用于后续的编程语言中，包括 C++、Java、Python 等。

> **Kristen Nygaard 和 Ole-Johan Dahl**
>
> Kristen Nygaard 和 Ole-Johan Dahl 两人对面向对象的贡献太多了。
>
> 奥斯陆大学是挪威最好的大学，也是世界最好的大学之一，在这个大学里，有两栋隔路相望的楼，一个名为 Kristen Nygaards 楼，一个名为 Ole-Johan Dahl 楼。 这是纪念奥斯陆大学两位杰出的校友，这两个年轻人在编程进入危机的黑暗年代，用努力的工作和杰出的才能，让人们看到了一丝曙光，开启了编程的新时代——面向对象编程的时代。
>
> 在共同获得图灵奖一年后，2002 年，两人先后离世。编程很难，感谢他们照亮了我们前进的路。

支持面向对象的语言有很多，这些语言都是按照设计者的意图来开发的，我们学 Java，那么 James Gosling 的意见是最值得参考的。幸好，James Gosling 给我们留下了大量的参考资料，他本人写过很多本书，比如《The Java Language Specification》、《The Java Language Environment》等。除了文本资料，还接受了大量的访谈。这些资料，可以解释 Java 中几乎所有的为什么。

前面讲了 Simula 语言的历史，但是“面向对象”这个词并不是 Simula 提出的，而是 SmallTalk 的作者 Alan Kay 提出的。对面向对象，Alan Kay 的理解是：“互相传递消息的程序设计就是面向对象。”对先驱 Simula 语言，Alan Kay 有自己的看法：“我不喜欢 Simula 中继承的做法，我并不是不喜欢类，但是我还没有见过不让我感到痛苦的包含类的编程语言。”

> Alan Kay
>
> 在美国施乐公司工作期间，Alan Kay 研发了 SmallTalk 语言，SmallTalk 语言借鉴了很多 Simula 的特性。除此之外，除此之外，Alan Kay 在 IT 领域还做出了很多其它的贡献，比如开发出图形用户界面(Graphical User Interface，GUI)、 提出作为现代笔记本计算机原型的“DynaBook”等。
>
> 他还说过一句特别出名的话，大家可以学一下，在恰当的时候扔出来让别人震惊一下：预测未来最好的方法就是创造未来。（The best way to predict the future is to invent it.）

C++语言则是有不同的看法。在 C++作者 Bjarne Stroustrup 的著作《The Design and Evolution of C++》中，他认为 “Simula 的继承机制是解决问题的关键” ， “类是一种创建用户自定义类型的功能”，“面向对象程序设计是使用了用户定义和继承的程序设计”。我觉得这可能与 C++作者的经历有关，他在发明 C++之前，一直用 C 和 Simula 语言，C 语言不支持类，Simula 又太慢。他最初的想法是实现一个速度很快的 Simula，最好像 C 语言一样快，C++最初的名字是 C with Class 语言。

接下来，我们来看看 Java 是如何权衡利弊，被设计成这样的。

3.2 Java：一切皆对象

写这个小标题的时候我有点犹豫，因为很多语言都这样宣传自己，Java 如此，Python 如此，Ruby 也如此。但是如果仔细的钻起牛角尖来，每门语言又都有一些不是“面向对象”的部分。对 Java 来说，如果不用面向对象的方法来编程，软件根本就没法运行，因为 Java 所有的代码，都要在对象里，所以我还是起了这个标题。

3.2.1 什么是“对象”

1995 年，James Gosling 写了第一份 Java 白皮书《The Java Language Environment》，在这本白皮书的第三章，详细的介绍了什么是对象(object)。

当时是 1995 年，人们对“面向对象”有很多争论，现在 20 多年过去了，争论的声音越来越少。就像现在谈起“面向过程”编程，几乎没有什么争论一样，“面向对象”也要经历同样的过程，尘埃会慢慢落定。

我们的生活中充满了对象：车、咖啡机、鸭子、树都是对象。软件也由对象组成：按钮、菜单、图标也都是对象。无论是生活中，还是软件中的对象，都有自己的状态和行为。**我们在用面向对象编程的时候，需要将现实中的对象或概念映射到程序中的对象中。**如何把现实中的“对象”建立在计算机中，是“面向对象”要解决的问题。

图 3-1 现实世界的“对象”和计算机处理的“对象”

我们可以对一辆现实中的汽车进行建模，让其映射成计算机中的一个对象。一辆汽车有其状态（车速、油耗、颜色、手动挡还是自动挡等等）和行为（启动、转向、停车等）。

当我们开着这辆车去上班后，可能会在办公室里查查自己买的股票。股票也是对象，也可以映射在计算机里，也有自己的状态（最高价、最低价、开盘价、收盘价等）和行为（股价波动、退市等等）。看完不太理想的股票，你头晕眼花，去咖啡机冲一杯咖啡缓一缓。咖啡机也有自己的状态（水温、咖啡种类等等）和行为（加热、搅拌、流出一杯咖啡等等）。

不管对象是什么，都可以把它们进行归类，同一类对象拥有类似的状态与行为。比如汽车，无论品牌、型号如何，都拥有类似的状态与行为。公司的停车场里，可能每辆汽车都有些许区别，但是可以称之为同一类对象，这便是类（class）这个关键字的来历。

在 Java 中，已经内置了很多类，比如 String 类、Date 类等，在面向对象编程中，类就是数据类型。程序员可以定义新的数据类型来解决编程的问题，比如定义新的汽车 Car 类。这些新定义的类与 Java 内置的类没有什么不同，从这个意义上来说，当提到类的时候，要在脑海中想到数据类型，反之亦然。

综上所述：**类是一种抽象的概念，是一组具有相同属性和行为的对象的模板，是蓝图。**可以将类看作是一种数据类型，定义了一组成员变量和成员方法，描述了对象的状态和行为。一旦拥有了类，就可以使用类来创建任意数量的对象。

对象是类的实例，是类的具体化，是根据蓝图制造出来的东西。当使用 new 关键字创建一个类的实例时，就会得到该类的一个对象。比如可以使用汽车 Car 类，来创建一辆特斯拉车的实例。对象有具体的状态和行为，即具有类中定义的成员变量和成员方法。对象可以调用类中定义的成员方法，从而改变自身的状态。接下来，再来看看如何操控对象。

3.2.2 如何操控“对象”

操控对象，可以从操控软件谈起，想这样一个问题：“如果要开发一个软件，需要组合起多少对象才能让软件运行起来？”将问题拆解为一系列对象是面向对象编程中最常用的方

式。

比如，一个社交软件可能需要有“好友”、“聊天群”这些对象，我们还需要一些显示好友列表的对象，显示群组成员的对象与之交互等等。当我们一项一项的细化分解这些对象，最终会发现，这些对象要么已经存在，比如列表对象在安卓的 SDK 中已经存在了，要么这些对象经过分解后，已经简单到可以容易的编写。

站在用户的角度，软件是功能的提供者；站在程序员的角度，对象是功能的提供者。用户的每一步操作的背后，对应的是一系列对象“生死不息”的交互。所谓的生，就是在 Java 中创建一个对象。所谓死，就是对象完成了其交互的使命，所占用的内存被回收。最难的则是对象在“生死之间”所进行的交互，这些交互一般包括：

1. 访问/修改对象的属性

在 Java 中，要访问对象的属性，可以使用点操作符（.）来访问对象的属性。比如如果要显示一个“聊天群”有多少群员，则要访问“聊天群”对象中群员数目这个属性。假设群组的名字为 group 并且有一个群员数目的属性 membersOfGroup，那么可以这样来访问：group.membersOfGroup。当加入新成员以后，也有相应的方法修改 membersOfGroup 属性。

2. 调用对象的方法

在 Java 中，要调用对象的方法，可以使用点操作符（.）来调用对象的方法。点操作符用于调用对象的公共方法。比如在聊天软件中，有“拍一拍”好友的功能，当系统检测到用户在拍好友的时候，就调用相应的方法。假设有名为 friend 的好友对象并且有名为 tickle()的公有方法，当“拍一拍”发生的时候，会调用如下的方法：friend.tickle()。

3. 传递对象作为参数

在 Java 中，可以将一个对象作为参数传递给另一个对象的方法。这使得另一个对象能够访问和操作作为参数的对象的属性和方法。通过这种方式，对象与对象之间可以进行交互，这也是软件得以实现复杂功能的方式。

设计精良的 Java 代码通常比较容易理解，对象对应于要解决的问题中的实体，操作对象则代表了解决问题所采取的具体的活动。

3.3 面向对象编程为什么这么难

很多的程序员并不清楚为什么要用面向对象编程，也不了解面向对象编程的历史，反正大家都在用，有关面向对象编程的工具有很多，有好用的 IDE，有 UML 工具，那我也跟着用用就好了。在这种心态下，面向对象编程就成了一种很神秘的东西，“虽然不理解，但是

大家都在用”。

德国数学家赫尔曼·魏尔曾经对数学发表过这样的评论:“数学具有星光般非人的品质，灿烂、精准，然而，冰冷。”我想，数学和计算机编程在这里有相通之处，都是灿烂且精准，同时，也是冰冷的。不同于数学，在编程行业，有太多过于“炙热”的宣传，如果是出于蓄意的推销，也算是一种恶，如果宣传的人自己也相信了，那就是一种蠢了。

面向对象编程目前来说是最流行的软件开发技术,但是却比较难理解。要掌握这项技术，需要很多的锻炼，就算现在有了很多辅助工具，如果不能从内涵中理解面向对象编程，也无法熟练使用。在这一小节中，我想来讨论一下为什么面向对象这么难以理解，到底难在什么地方？

3.3.1 名词搅拌器

用 2000 多年前孔子的话来讲：“道不远人”。道并不远离人的日常生活，一个人修道的时候如果远离人的日常生活，那他修的就不是道了，用武侠小说中的话来说，可能已经走火入魔了。面向对象编程也是如此，不管这个技术多好，都不能远离程序员太多，如果这种技术已经让大部分程序员搞不懂了，那就不是好的编程技术。

如果面向对象编程让程序员有亲近感，首先在语言上要平易近人。显然，面向对象编程在这一点上做的不好，甚至可以说很差。面向对象有大量的名词，这些名词像把一本新华字典丢进了一个搅拌器，随机搅拌了一堆词语出来，比如下面的词汇：

泛化、特化、父类、多态、属性、委托、注入、构造函数、异常、框架、类库、组件、模式、用例、建模、重构、敏捷、重写、集合、关联……

这只是举了一部分中文的，还有大量英文缩写的没有列出来。这种名词搅拌器一样的编程语言，确实让人很头疼。如果是初学者，一看到这些名词，就已经吓的不敢深入学习了。

存在大量的术语有多方面的原因，如果了解了背后的原因，就不会如此纠结了。

一部分原因是广告的需要，公司推广一项技术，就要写的云山雾罩的，这样显得比较有技术含量。这个不仅是技术行业，在任何行业都是这样，以化妆品为例，我经常盯着老婆化妆品上那些诸如“活泉精华”“抗氧精粹油”发呆，每个字都认识，连起来就是不懂这是什么黑科技。

还有一部分原因是技术人员故意避开以前存在的名字，以便显示自己的独创性，其实没什么独创性，人类总是重复发明旧的科技，然后新瓶装旧酒，用新的名字包装旧的技术。比如 Java 中的多态（polymorphism），这个技术无论是内涵与外延，都和动态绑定（dynamic binding）是一样的，后来又来了一个后期绑定（late binding）和一个运行时绑定（run-time binding）。同一个类似的技术，一下子有了四个名字。这种情况在面向对象的技术发展过程中实在是太普遍了，大家一定要一眼看穿这种鬼把戏。

3.3.2 滥用隐喻

在著名的《代码大全》这本书里，第二章讲了隐喻对理解软件开发的影响。如果选择错了隐喻，那么就会对软件开发有很大的误解。

同样，在面向对象中，如果乱用隐喻，也会让人对面向对象产生误解。

上一小节举的“对象”的例子是参考 James Gosling 写的 Java 白皮书《The Java Language Environment》的第三章。把现实中的对象映射为编程中的类，有助于理解面向对象编程，但是同时也有副作用，会让读者误以为现实世界的物体都可以映射为类，实际情况并不是这样的。

现实中的物体不是由类创建的，而且和面向对象编程中的类大相径庭。现实中的人是父母生的，不是类创建出来的实例。除此之外，现实中的人会根据场景的不同有多种角色，比如我在公司是“员工”，在家里是“父亲”和“丈夫”，对父母则是“儿子”。

但是在 Java 面向对象编程中，一旦根据类创建了实例，那么这个实例就只属于唯一的类，无论时间空间怎么改变，都无法改变这个实例的类型。现实中的我，随着时间的变化，已经从“后浪”成了“前浪”，从“少年”变成了“大叔”。

软件只能涵盖人类工作的一小部分，并不因为引入了面向对象编程而让软件有质的变化。虽然在推广和宣传面向对象编程的过程中，有意无意的夸大了该方法的优点，但是我们程序员切不可认为面向对象真能模拟现实世界。

在面向对象编程以后，还曾经兴起过一个叫“面向代理”的技术浪潮，但是这个技术很快销声匿迹了。“面向代理”的技术在宣传的过程中，有点类似“人工智能”，能够主动的创造软件，自主的响应人类的需求。很可惜，目前的技术好像实现不了所宣传的目标。

有了计算机，大量的工作仍然要用人来实现。目前的计算机架构决定了无法完全顶替人类来完成现实世界的工作。引用《Code: The Hidden Language of Computer Hardware and Software》的作者 Charles Petzold 的话来说：“人类有很多的交流形式不能用非此即彼的可能的选择来表示，但是这些交流形式对我们人类的生存又非常重要。这就是人类为何没有与计算机建立起浪漫关系的原因（无论如何，我们都不希望这种情况会发生）。如果你无法用图画或者声音来表达某种事物的时候，你就无法将这个信息用比特的形式来编码。”

像林俊杰的歌中唱的：“确认过眼神，我遇上对的人。”如果你相信面向对象可以对世间万物进行建模，那么，我们如何能用面向对象的方法对眼神进行建模呢？目前我们是做不到的，眼神中的含义千千万万，不是编程能解决的问题。

隐喻能帮助我们理解面向对象编程，但是也会干扰我们对面向对象编程的理解。这也是面向对象编程难以理解的又一个因素：滥用隐喻。

3.3.3 过度宣传

> 面向对象编程是正确程序的替代品
>
> ---- Edsger W. Dijkstra

Dijkstra 的这句话提醒我们不要教条，[4]过度宣传会给用户不切实际的幻想，经过 30 多年坚持不懈的宣传，让程序员误以为就没有面向对象解决不了的问题，我把这称之为 “面向对象” 综合症。

通过前面介绍 Simula 67 的历史，我们会知道 Simula 67 这个面向对象语言的先驱，当时并没有“发明”出面向对象这个概念，而是后来 Smalltalk 语言的作者提出的。在 Simula 67 中，引入 class 的作用仅仅是汇总子程序和变量的结构，到了 Smalltalk 中才开始使用继承结构来组织类库，使所有的类都继承自 Object 类。

由于 Smalltalk 和 C++的流行，面向对象的概念已经不是某个人能左右的了。这种现象已经不能用技术来解释了，让我想起了郭德刚讲的一段相声，当一个人听到一个段子的时候，会添油加醋的渲染一番，然后再传给下一个人，下一个人也是如此这般，等传到十个人的时候，事情已经面目全非了。

以我工作的经历，那些完全不懂技术的领导也知道面向对象的好处，而且很有主见。领导或者项目经理在给程序员提意见的时候，经常说：“不要想得那么难，用面向对象的方法，对这些东西建个模，很容易的。”然后，我还要装作恍然大悟的样子夸领导：“您真是不但懂管理，还把技术领悟的这么透彻！”

这都是过度宣传导致的结果，面向对象编程被包装成了无所不能的银弹。[5]虽然现实世界是由一个又一个的对象组成的，但是想把现实世界映射在程序中，并不容易。

两届普利策得主约翰·卡雷鲁写过一本书叫《坏血：一个硅谷巨头的秘密与谎言》，在书里他这样写道：“雾件（Vaporware）反映了计算机行业的一种倾向，在涉及市场营销时，做法非常轻率散漫。微软、苹果和甲骨文都曾被谴责某些时候都有类似的做法，过度承诺是硅谷的标志性特征之一。”雾件（Vaporware）这个词现在已经不太出现在媒体上了，但是这种过度吹嘘的行为至今仍然存在，甚至愈演愈烈。

---

[4] 他的原话是：object oriented programs are offered as alternatives to correct ones.

[5] IBM 大型机之父佛瑞德·布鲁克斯写过一篇论文《没有银弹：软件工程的本质性与附属性工作》，在这篇论文中他强调强调由于软件的复杂性本质，而使真正的银弹并不存在；所谓的没有银弹是指没有任何一项技术或方法可使软件工程的生产力在十年内提高十倍。其中银弹（Silver Bullet）的来历是：在欧洲民间传说及 19 世纪以来哥特小说风潮的影响下，银弹往往被描绘成具有驱魔功效的武器，是针对狼人等超自然怪物的特效武器。后来也被比喻为具有极端有效性的解决方法，作为杀手锏、最强杀招、王牌等的代称。

以上就是面向对象为什么这么难的原因。

本书要做的事情就是尽量把 Java 和面向对象的知识点讲清楚。为尽量避免上面讲到的三种缺点，本书会使用如下的原则：

1. 为了避免名词搅拌器，尽量不过多引入术语。
2. 会使用隐喻，但是不会到处使用隐喻，最终还是会落实到代码上。
3. 不过度的宣传 Java 以及面向对象的强大，而是从为什么引入这种技术以及这些技术能解决什么问题的角度来讲解。

3.4 面向对象编程可以很简单

在大体了解了什么是对象，如何操控对象以后，我们再用例子讲解一个为什么面向对象编程有优点？

无论写什么软件，都要先进行设计，设计有简单有复杂，简单的设计仅需要灵机一动，复杂的设计文档会车载斗量。软件设计最高原则是趋利避害：尽量让好事发生，让坏事不要发生。

设计阶段，自然而然的用面向对象的方法来思考问题。比如考虑一间房子时，不会去想门、窗户和玻璃的具体组成。如果房子很多，我们的视角会放大到小区。如果小区很多，我们的视角会再次发生改变，抽象到城市的层面。城市再多，会抽象成一个国家。

一层又一层的抽象之中，我们实际上是用面向对象的方法隐藏了很多细节信息，这主要是为了让我们的大脑有足够的处理空间。

这是有科学依据的，著名的计算机科学家 Dijkstra 曾经说过：“没有人的大脑可以容纳下一个现代的计算机软件”。目前的软件越来越庞大，数百万数千万行的软件早已经不是什么新鲜事，没有人可以了解一个大型软件的方方面面。我们如何确保在不知道其它功能的前提下，安心的做我们这一块功能？又如何确保我们做完了这个功能，能保证和其它部分和谐运行呢？面向对象编程就是来解决这个问题的。

目前，人们找到的一个解决方法是将整个系统分解为子系统，然后再将子系统分解为类，再将类分解为子程序，最后再把子程序变成可以运行的代码。

如何设计类是一门大学问，下面就先以一个足球游戏来简要的说明一下，先有个大体的概念。

3.4.1 FIFA 足球游戏

足球是世界第一大运动，上一届世界杯说有 22 亿人观看。有两个最著名的足球游戏，

一个是 KONAMI 的《实况足球》，另一个是 EA 的《FIFA》。由于我本人是长期的 FIFA 玩家，以 FIFA 19 来举例吧，假设 EA 把我们请去设计 FIFA 游戏，我们应该怎么做？

这个游戏里，有数百支球队，有数百位教练，有数千名球员，有看台边上举着相机的记者，还有看台上数万球迷.....想想头就大了。幸好，我们可以用面向对象的方法把所涉及的对象抽象出来。

> 软件危机的主要原因，很不客气地说：在没有机器的时候，编程根本不是问题；当我们有了计算机，编程开始变成问题；而现在我们有巨大的计算机，编程就成为了一个同样巨大的问题。
>
> -- Edsger W Dijkstra

为了照顾没玩过 FIFA 足球的读者，我先用文字描述一下这个游戏大概的玩法：玩家用手柄或者键盘控制场上踢球的球员，现实足球场上发生的一切，都尽量的在游戏中加以模拟。当然有所取舍，因为完全模拟是没办法做到的，如果仔细看得话，会看到场边有举着相机的记者，他们的动作是一模一样的，看台上的球迷也都雷同，场边的第四官员和教练动作也比较僵硬。原因就是因为计算力有限，FIFA 使用的寒霜引擎非常耗费资源，好钢用在刀刃上，好运算力用在渲染球员上。

对这个游戏来说，最重要的部分是球员。让我们来看看如何抽象游戏中最重要的球员吧。

3.4.2 对球员进行抽象

下面是 FIFA 19 中的截图，在 FIFA 中，球员有很多的属性，我挑一部分介绍一下。

每个球员都有名字，图里面用的是 Messi，还有球员在场上的位置，Messi 在 FIFA 19 是右边锋(RW)。94 这个数字是能力值，能力值是 FIFA 游戏开发组给每个球员的综合评价，梅西和 C 罗是最高的 94 分。

还有一些属性是国家：阿根廷（显示的国旗），所属的球队（巴塞罗纳）。还有下面数字分别代表的单项能力值（满分是 100）：

PAC(速度)：87

SHO(射门)：92

PAS（过人）：92

DRI（盘带）：96

DEF（防守）：39

PHY（体能）：66

以上列出了球员的状态，除了属性，还要列出球员的行为。

当设计类的时候，主要考虑类的两个方面：

1. 这个类有什么属性？
2. 这个类有什么行为？

球员在球场的行为有很多种，像 FIFA 游戏，场上的每个球员可以响应 86 种动作，[6]本书只列出其中的几种：停球，传球，射门，防守。根据这些属性和行为，可以画出球员的类如下图所示：

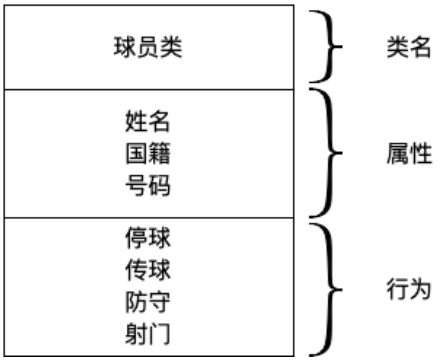


图 3-2 球员类

经过抽象，在编程的时候，我们就可以把球员当成一个整体来考虑，从而减轻大脑的负担。编程首要的任务是控制复杂性，用这种抽象的方法能有效的简化编程过程，可以忽略掉

[6] 如果大家对此有兴趣，可以到本书的补充材料 chapter3 中查看具体的动作。实际上，一般玩家常用的动作在 10 种左右。

次要的属性，而将主要的精力放在重要的属性上。以我们设计的球员类为例，对调用球员的代码来说，实例化的某个球员是可见的，但是球员类的内部信息是不可见的，比如如何实现停球，传球，外部调用代码无需知道球员内部的实现。这种方法，学术一点叫数据封装与信息隐藏。

在 Java 中，数据封装是指限制对类属性与方法的访问（一般设置为 private），使这些属性只能通过类提供的方法来访问和修改，从而保证了数据的安全性。信息隐藏是指将类的实现细节隐藏起来，只对外提供公共接口，使得类的使用者只能通过公共接口来访问类的方法，而无法直接访问类的实现细节，保证类的实现细节不会被外部随意修改，从而提高了代码的安全性。

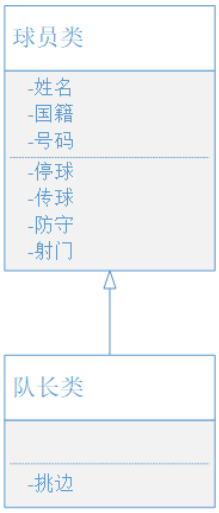
3.4.3 继承

每个球队中都有一个场上队长，像梅西目前担任球队的队长，布教授是球队的副队长。从编程角度来看，队长和普通球员绝大部分属性是一样的，在球场上，队长会戴个队长的袖标。当把队长换下场的时候，有时候会出现一个过场动画，会有球员把队长袖标让副队长戴上。

队长类和球员类几乎是一样的，虽然直接把球员类拿过来是不行的，但是只要把球员类稍微的做一下修改，那就可以了。这时候又产生了一个问题，我们刚刚讲过，类要保证数据封装和数据隐藏，如果直接修改源码的话，修改人员要了解球员类的代码，才能做出修改，这很困难。为此，Java 引入了继承的机制来解决这个问题。

在编程中，如果新的类（比如队长类）可以继承已有的类（比如球员类）的数据和功能，而且新的类可以允许对原有的类增加或者修改一些内容，那么复用将会变得更加便利。用这种方法，程序员就可以从已经完成的类开始，修改得到其子类型，满足新的要求。这还能理顺类之间的关系，队长也是球员，两个类之间是父子关系，球员类是父类，队长类是子类。

我们要给队长加个行为：挑边。挑边的意思是开场前，主裁判会扔个硬币，两队队长会根据硬币的正反来选择进攻的方向。按照前面的理论，只要有个机制确保能继承球员类即可。示意图如下：



Java 的继承是一种面向对象编程的重要特性，它允许一个类（称为子类或派生类）继承另一个类（称为父类或基类）的属性和方法。继承是一个重要的概念，它能够促进代码重用，提高代码的可维护性和可扩展性。在 Java 中，通过使用关键字 extends 来实现类的继承。

从理论上来说，继承可以从多个父类中继承，这种叫多继承；也可以从单个父类中继承，这种叫单继承。不同的语言在这个地方有所取舍，Java 采用了一种较为简单的方式，只支持单继承。这样虽然简单，同时也失去了多重继承的优点，为了弥补这种缺点，Java 语言实际上可以借助接口（Interface）来实现多重继承的功能，这个在以后的章节我们一起学习。

继承的好处我们已经讲过了，现在讲继承的缺点。

如果过度的使用继承，会导致有多层的继承树，这样编程风格的代码，处理起来非常困难。假设我们有这样一个类，由多层继承而来，该类中有一个方法。如何找到这个方法最初的定义位置呢？没有好办法，只能先查父类，如果没有，再查父类的父类……就算查到了，也不敢贸然修改，因为不确定这个类影响的范围有多广，如果修改了这种类，一定要用一种叫回归测试的方法进行广泛测试。[7]

总之，凡事要有个取舍，我在工作中读过 10 来层的继承代码，理解起来非常难。尽量兼顾代码的复用度和代码的清晰度。

[7] 回归测试是指修改了旧代码后，重新进行测试以确认修改没有引入新的错误或导致其他代码产生错误。自动回归测试将大幅降低系统测试、维护升级等阶段的成本。

3.5 脱口秀：面向对象的优点

以下故事纯属虚构。帕克休斯是一个虚构的记者，栋哥是一个虚构的 FIFA 首席设计师。

帕克休斯：你好，栋哥，我们都知道你已经成了 FIFA 的首席设计师，你能谈谈这次设计的目标么？

栋哥：这次设计我只有一个小目标，先赚他一个亿！

帕克休斯：呃.....不是从经济收入上，能从程序设计的角度谈一下么，比如，听说你在设计这款足球游戏的时候用到了面向对象的编程方法，我对面向对象也有所了解，我在大学里的时候老师就讲过，面向对象就是类，包括封装，继承好像还有多态什么的。我有一个问题，听老师讲了这么多，还是有点不清楚，不知道这些具体怎么用，你能给我讲一下么？

栋哥：学校里主要是书本知识，书本知识是一种提炼与抽象，套用一句常用的话来说就是源于编程，又高于编程。在学校中，绝大部分学生最关注的是考试成绩，在现实的编程中，你认为编程最重要的是什么？

帕克休斯：是如期的发布产品？

栋哥：也可以这么说吧，如期发布高质量的产品是软件开发最困难的事情。像 FIFA 每年都是雷打不动的 9 月末发布，去年 9 月 28，今年还是 9 月 28。而每年的 FIFA 都要进行不少的改动，用软件开发大师爱德华·贝拉德话来说就是：“Walking on water and developing software from a specification are easy if both are frozen.”翻译成中文是：“一边走在水上一边开发软件是非常容易的，只要两者都冻结了。”但是现实社会不停的变化，软件需求也要不停的变化，比如在 2019 年 7 月份，我得知我们的竞争对手 KONAMI 在 7 月 3 日宣布和曼联成为合作伙伴，7 月 12 日宣布与拜仁慕尼黑达成协议，7 月 16 日拥有 C 罗的尤文图斯也加入了他们的阵营。为了应对这些变化，在 9 月末发布的《FIFA 20》中将不能采用这些球队的队名、队徽、球衣和球场，只能用虚拟的模型替换，侵犯了版权要赔钱的，还有两个月时间，对我们来说，变化是持续的。

但是，我们不能因为这种变化就延期游戏的发布。

帕克休斯：那么，你们是如何将这种影响减少到最低的呢？

栋哥：这其中最主要的关键是设计，一个优雅的软件，像 FIFA 这样的软件，是可以很容易应对改变的，也非常容易进行扩展和复用。这种设计可以称之为“面向对象设计”，优雅软件设计的关键之一。

帕克休斯：你说的“面向对象设计”和“面向对象编程”是一回事么？

栋哥：面向对象设计不仅仅从微观上着眼于编码，还要从宏观上着眼于产品。所以这两

者不是一回事，但是又相互联系。 面向对象设计需要面向对象编程，但是又不仅包含面向对象编程这一个概念，还要更多内涵，主要有这四个方面：

1. 面向对象编程的方法
2. 代码复用
3. 只改变极少量代码就可以应对需求变化
4. 不改变任何代码就能扩展软件的功能

我分别来讲一下这四个方面，面向对象编程方法已经讲了太多了，这里就先不讲了。讲第二个代码复用，FIFA 已经做成了一年发一版的年货游戏，不可能每次都从头做起，代码的复用显得特别重要。实际上 FIFA 的引擎不仅是 FIFA 自己在用，EA 的很多游戏都在用，目前使用的是寒霜引擎，EA 公司的通用引擎，还用在《战地》、《极品飞车》等其它十二款游戏上。[8]

第三点是只改变极少量代码就可以应对需求的变化。像前面说的，7 月得知公司丢了几个球队的版权，9 月游戏发布，这种调整要非常迅速。

第四点是不改变任何代码就能扩展软件的功能。这一点，FIFA 做的也非常优秀，像 FIFA 19 刚发布的时候，并没有女足世界杯，但是在女足世界杯前夕，官方发布了补丁，该补丁升级了 22 支女足世界杯国家队，球衣队徽，官方比赛用球，真实的球场等等，这些扩展没有修改任何代码就完成了。

帕克休斯：你说的这些听起来不错，是你自己悟出来的么？

栋哥：当然不是我自己悟出来的。计算机行业中有很多人花费大量时间来研究“面向对象设计”，并且总结出了很多行之有效的原则，形成了不少设计模式供我们借鉴。像 SOLID 原则就是其中之一。

帕克休斯：SOLID 原则？

栋哥：是的，SOLID 是面向对象设计和面向对象编程中几个重要的原则，SOLID 是这几个原则的首字母的缩写，共有五个原则。分别是：

1. 单一责任原则（The **S**ingle Responsibility Principle）
2. 开放封闭原则（The **O**pen Closed Principle）
3. 里氏替换原则（The **L**iskov Substitution Principle）

[8] 在 2010 年的时候，EA 通过不停的收购，拥有了很多世界知名的游戏品牌，但是也带来了一个巨大的麻烦，当时 EA 有十三款不同的游戏引擎，有虚幻引擎开发的《质量效应 2》，有变色龙引擎开发的《极品飞车》，这导致暴涨的开发费用。

4. 接口分离原则（The **I**nterface Segregation Principle）
5. 依赖倒置原则（The **D**ependency Inversion Principle）

分别取上面各条原则黑体部分的首字母，就组成了 SOLID 这个单词。因为这几条原则太重要了，不光在面试时候经常被考官问，在实际的编程中，也要用到这五条原则。

帕克休斯：这样说太笼统了，你能详细讲一下么？

栋哥：好的，先来看第一个原则，单一责任原则。这个原则是说一个类只有一种类型责任。就拿球员类来说吧，十一名上场的球员有一名是守门员，守门员的职责和普通球员有所不同，所以，守门员和球员要分成两个类。

帕克休斯：也就是说在划分类的时候要根据类的责任了？

栋哥：可以这么说，像我前面所举的例子，普通球员和守门员大部分情况下都是一样的，甚至可以认为普通球员能做的，守门员都可以做。像德国的诺伊尔就经常跑到禁区外，简直就是个后卫，所以人们称之为门卫。在禁区之外，门将和普通球员一样，他们的区别在禁区之内，在禁区内，门将是可以用手的。所以，在设计的过程中可以这样来设计：

1. Player 类：普通球员类中只要做正常的设定即可。
2. Goalkeeper 类：门将类可以继承 Player 类，然后再添加在禁区内用手的操作。

帕克休斯：好吧，我就假装听懂了，能继续介绍一下这个开放封闭原则么？

栋哥：开放封闭原则也非常的重要，用一句话来解释可以称之为一个类要易于被扩展，但要难于被修改。通俗来说就是核心的类，如果要增加或者改变其功能，最好的方法是扩展这个类，除非万不得已，不要修改这个类，因为一旦修改，众多依赖于这个类的其它类都会受到影响。开放封闭原则的开放是对扩展开放，封闭则是对修改封闭。

帕克休斯：你这个原则让我想到了我们自己也是易于扩展而难于修改。如果我想社会一点，我可以贴个纹身贴，来个爬行动物贴在身上。如果我想显得潮一点，可以来个锡纸烫。这些都没有对我的身体造成影响，如果要动手术整容那可就比较麻烦了。所以，我觉得人类也是易于扩展，而难于修改的。

栋哥：你这个想法非常的好，我还没想到呢。对开放封闭原则来说，最重要的是抽象，把最重要的特性与功能抽象出来，如果抽象做的不好，这个类就设计的不够有扩展性。

帕克休斯：那你能给举个设计的比较有好的例子呢？

栋哥：这种扩展性设计的很好的软件有很多，比如浏览器就有很好的扩展性。无论 FireFox，Chrome 还是 Edge，在开发这几个主流的浏览器的时候，不用关心浏览器要上哪个网站，这个网站用的什么样的服务端，这样就减少了对具体服务器的依赖。如果要扩展浏览器的功能，完全不用改变原有浏览器的代码，而只需要写相应的插件即可。所以，这种设计

又开放又封闭。

帕克休斯：那还有第三个原则叫里氏替换原则，这是什么意思呢？里氏这个名字好奇怪啊。

栋哥：和牛顿定律是牛顿提出来的一样，里氏替换原则也是一位叫 Barbara Liskov 的女士提出的。Liskov 女士发表了一篇名为《数据的抽象与层次》的论文，[9]在论文中她提出了这样一个观点：“如果对每一个类型为 S 的对象 o1，都有类型为 T 的对象 o2，使得以 T 定义的所有程序 P 在所有的对象 o1 都代换成 o2 时，程序 P 的行为没有发生变化，那么类型 S 是类型 T 的子类型。”[10]但是这个表述实在是太科学严谨了，于是有人把这个描述通俗化了：“派生类（子类）对象可以在程序中代替其基类（超类）对象。”

帕克休斯：这个说法还是太严谨了，能再举个通俗的例子么？这些原则是为了科研而提出的么？

栋哥：这些原则都是长期的经验总结，并不是为了科研而科研。我们在做设计的时候，最重要的目标是简化编程的难度，如果设计得当，可以让我们在写一个类的时候，安全的忽略其它的类。我们设计类的时候，也是冲着这个目标去的。

我们在使用类的继承时，不能给自己添麻烦。设想一下，如果我们继承的子类将父类已经实现的方法给重写了，会给我们写程序带来相当的惊喜。为了不让我们惊喜，使用继承，要遵守的原则是，只要出现父类的地方，都可以用子类代替。

比如说前面我举的例子，门将类继承自球员类，只要出现球员的地方，都可以用门将来代替。在现实中也是如此，只要教练愿意，是可以派上 11 个门将的，虽然没人这么做过。球员能拥有的属性和动作，因为门将是子类，所以门将都有，但是在软件中，反之并不如此，在软件中，门将比球员多了一些属性和动作，比如用手扑球，如果用父类来取代子类，那么这个门将就不会扑球了。

帕克休斯：我明白了，也就是说，在做测试的时候，所有针对父类的测试，在子类上都要通过，是么？

栋哥：是的，就是这个意思。所有在父类上执行的动作，在子类上都要执行的一模一样，这就是替换的原则，子类可以代替父类。

---

[9] 论文我已经下载到本书的辅助材料中，存于 resources/chapter3 文件夹中。

[10] 在《Data Abstraction and Hierarchy》中的原话是：If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T.

帕克休斯：那继续说一下第四个原则，接口分离原则吧。

栋哥：我举个例子吧，你应该去过火车站飞机场这种场合，那些地方有那种免费给手机充电的接口，长的就像下面这样。

图 3-3 万能充电器

现在每个人都有手机，你会用这样的手机充电线么？

帕克休斯： 当然不会用了，我的手机只有一种接口，有那么多，根本没用啊。

栋哥：是的，在编程中也是如此。如果接口被设计的大而全，就像上面的充电器一样支持所有手机，会让程序显的非常杂乱，从而难以维护。最好的设计是只用自己有用的接口，将大而全的接口分离。当然了，在特殊的情况下，像机场，大而全的设计也有一定的存在价值，但是总体来说，在编程中，这样的设计基本上是没有好处的。如果有些类用不到大而全的接口，一定要记得分离。

帕克休斯：你这样说我就清楚了，我可不想拿着那么恐怖的充电器。那再讲一下最后一个设计原则吧，叫依赖倒置原则吧。

栋哥：先要理解什么叫依赖，在编程中的依赖和现实中的依赖有些不同。我用依赖造几个句子，比如“她不喜欢依赖别人，自己的事情总是自己做”。再比如“老婆和老公在生活中互相依赖”。在现实中，当我们说某个东西依赖另一个东西的时候，往往有些弱者依赖强者的味道在里面，或者互相依赖。在现实中，可以你依赖我，我依赖他，他依赖你，我们就像一个团结有爱的一家人。

但是，在软件中，你要是设计出了一个 A 类依赖 B 类，B 类依赖 C 类，C 类又依赖 A 类的死循环，那就有你受的了！

当我们说 A 类依赖 B 类的时候，是指 B 类是以局部变量的形式存在于 A 类之中。

帕克休斯：还是举个例子吧，这样说有点摸不着头脑。

栋哥：比如说我要去旅行，假设一个怀着“世界这么大，我想去看看”的旅行者类好了，这个旅行者有一个方法是出行，参数是某种交通工具。当我们出去的时候，是无法得知自己用哪种交通工具的，可能是走路，可能是坐三蹦子、出租车、高铁、飞机、轮船.....

我来举个软件的例子，我开一下计算机，空口无凭，只要稍微看下代码就知道了。你现在还写代码么？

帕克休斯：略懂一些，你讲吧。

栋哥：懂一点就好，看看下面的代码，这里面的 Traveller 旅行者类依赖巴士车类。

```
1.  class Bus {
2.      public String start(){
3.          return "我是巴士车，来不急了，快上车！";
4.      }
5.  }
6.
7.  class Traveller {
8.      public void travel(Bus bus) {
9.          System.out.println("世界这么大，我想去看看");
10.         System.out.println(bus.start());
11.     }
12. }
13.
14. public class TravellerTest{
15.     public static void main(String[] args){
16.         Traveller traveller = new Traveller();
17.         traveller.travel(new Bus());
18.     }
```

帕克休斯：这个我理解了，只要是巴士车，我们的旅行者就能远行。

栋哥：是的，但是还有个问题，如果出行的路上，没有巴士车呢？

帕克休斯：那可以坐其它交通工具啊，交通工具又不止一种。在旅行的时候，碰到什么车也得坐啊，当年我还搭过一辆水罐车的便车呢。

栋哥：但是，我们看看上面的旅行者类，里面的 travel 有个参数是 Bus 类，如果不是 Bus 类，那就没办法了。只能一种交通工具准备一种调用方法。

帕克休斯：这也太不合理了吧！

栋哥：非常的不合理，仅仅是换一种交通工具，就要不停的修改旅行者类，这不是好的设计。原因是 Traveller 类和 Bus 类之间的耦合性太高了。必须降低他们之间的耦合度才行。

帕克休斯：那应该怎么做呢？

栋哥：可以引入一个抽象的接口 IStart。只要是交通工具，都可以跑起来。看看下面的代码：

```
interface IStart {
    public String start();
}

class Bus implements IStart {
    public String start() {
        return "我是巴士车，来不急了，快上车！";
    }
}

class Tank implements IStart {
    public String start() {
        return "我是运水车，上车吧！";
    }
}

class Traveller {
    public void travel(IStart vehicle) {
        System.out.println("世界这么大，我想去看看");
        System.out.println(vehicle.start());
    }
}

public class TravellerTest {
    public static void main(String[] args) {
        Traveller traveller = new Traveller();
        traveller.travel(new Bus());
        traveller.travel(new Tank());
    }
}
```

帕克休斯：我知道了，这样无论怎么扩展，都不用再修改旅行者类了。

栋哥：是的，这就是依赖倒置原则。代表高层的旅行者类一般负责完成更主要的业务，一旦对它进行修改，引入错误的风险极大，使用依赖倒置原则就可以不对这个类进行修改。依赖倒置原则的核心就是要面向接口编程，理解了面向接口编程，也就理解了依赖倒置。

帕克休斯：是不是只要掌握这几个设计原则就能设计出完美的类呢？

栋哥：当然不能这么说，只能说一般情况下设计类的时候都要考虑这几种原则，基于这几种原则，人们还总结了很多经验，这些经验有个名字叫设计模式。

帕克休斯：我知道设计模式，没想到设计模式是建立在这些原则之上呢。

栋哥：其实设计模式并不神秘，算是一些长期以来的经验总结吧，有了设计模式，能让我们少走不少弯路呢。

帕克休斯：今天的采访就到这里吧，下次有机会再来谈谈设计模式的话题。

栋哥：好的，再见。

3.6 常见问题

3.6.1 面向对象的三个基本特征是什么？

面向对象编程（Object-Oriented Programming，OOP）是一种编程范式，其主要特征可以概括为封装，继承和多态。

封装（Encapsulation）：封装是将属性和操作属性的方法捆绑在一起的过程，形成一个称为"对象"的单元。对象的数据可以被外部访问，但必须通过定义好的接口。封装提供了一种用来隐藏对象的内部实现细节的机制，只暴露出一套供外部使用的接口。

继承（Inheritance）：继承是通过创建一个新的子类，继承并扩展一个已存在的父类的属性和方法。这意味着我们可以建立一个类的层次结构，将共享的代码放在父类中，然后在子类中添加或重写方法以满足特定需求，这增加了代码的可重用性。

多态（Polymorphism）：多态是指允许你将父对象设置为和一个或更多的他的子对象相等的技术，它使得父对象可以根据当前引用它的子对象的特性以不同方式运作。这意味着同一个方法可以在不同的对象上有不同的行为，这使得代码更加灵活，易于扩展。

这三个特性是面向对象编程的基础，并且它们相互作用，提供了一种更加强大的方式来组织和设计代码。

3.7 总结

本章的主要内容包括：

- 根据论文介绍了 Simula 语言以及面向对象编程发展的历史
- 介绍了 Simula 语言独创性的发明以及这些发明对当代面向对象编程的影响
- 简略介绍了 Java 面向对象的概念
- 探讨了面向对象难以理解的原因
- 通过一个故事简略介绍了如何使用 Java 进行面向对象编程
- 探讨了面向对象的优点与缺点

3.8 思考拓展

1. 根据本章所学的内容，大略的思考一下，如果你现在是设计师，要设计一个中国象棋的人机对战游戏，要分成哪些类？
2. 棋盘算不算一个类？棋格算不算一个类？棋子算不是一个类？不同的棋子之间的关系是什么？

4. 变量

在软件开发领域很有影响力的 Martin Fowler，同时也是敏捷软件开发理念的推动者之一，在 2009 年 7 月 14 日曾经发过这样一条推，内容如下：There are only two hard things in Computer Science: cache invalidation and naming things —— Phil Karlton。这句话翻译成中文是：“在计算机科学只有两件最难的事，缓存失效和变量命名。”

起初我觉得 Phil Karlton 的这句话是玩笑，笑过以后，觉得并不完全是玩笑，给变量命名真的很难。在编程中，最基本的构建活动是如何使用变量。变量看起来很简单，但是往往看起来简单的东西，深究起来并没有表面那么简单。就像门卫经常问我们三个问题：“你是谁？”“从哪里来？”“到哪里去？”，这三个问题往深处考虑，是人生的终极问题。

在《西游记》的第八十二回《姹女求阳 元神护道》这一章里，有这样一段：

> 好呆子，把钉钯撒在腰里，下山凹，摇身一变，变做个黑胖和尚，摇摇摆摆走近怪前，深深唱个大喏道："奶奶，贫僧稽首了。"那两个喜道："这个和尚却好，会唱个喏儿，又会称道一声儿。"问道："长老，那里来的？"八戒道："那里来的。"又问："那里去的？"又道："那里去的。"又问："你叫做甚么名字？"又答道："我叫做甚么名字。"

这段描写里，八戒的回答相当于什么都没有回答，他既没有回答他从哪里来，又没有回答他向哪里去，又没有回答他是什么名字。仔细想来，我们很多时候都是八戒，好像是明白了，实际上又好像什么也不知道。变量是什么？变量从何处来？又往何处去？我们真的知道么？

变量在现实中无处不在，以致于我们都忽略了，比如小学生发的奖状就变量，变量的类型是奖状，变量的值就是老师在奖状上填的学生姓名。还有无数的证件都等同于变量，出生证，结婚证，学生证，驾驶证……在现实中，证件最重要的是类型，当交警跟你要驾驶证的时候，你拿出结婚证来是不行的，用编程的术语来说叫类型错误。

物理学家费曼写过一本书叫《别闹了，费曼先生》，在书里他讲如果你碰到了一个新的知识，不要靠背诵来学习它，否则知识的基础非常薄弱，而应该通过已有的知识来类比，重新阐释，了解其历史的来龙去脉，最后确保真正的理解。我觉得如果要类比变量，证件是一个比较贴切的例子。变量有类型，证件也有类型。变量有生命周期，证件也有生命周期，比如第一本驾照是 10 年吧。变量有作用域，证件也有作用域，比如你拿中国的驾照去美国开车是不管用的。

以驾照为例，我无意去考证驾照的来历，但是有一点我确定，驾照一定是在汽车发明出来以后的某一年才开始有的。对变量来说，由于变量依附于内存，那么变量也一定是在内存发明出来之后才产生的。当然了，由于计算机的历史非常短暂，考证起来非常容易。接下来

我就来考证一下变量的历史。

4.1 变量的历史

内存是变量的基础，变量的本质是某一块内存的名字。

计算机起初是没有内存的，没有内存的计算机根本谈不上“编程”。比如很多教科书里都提到的世界上最早的计算机之一 ENIAC 就是没有内存的，所有的数据和指令都依靠外部数千条线缆输入，专业术语叫“插线板编程（Plugboard Programming）”。若想让 ENIAC 换一个功能，就需要重新布置线缆。如果大家对这台没有内存的机器感兴趣，可以阅读一篇名为《When Computers Were Women》的文章，还有 CNN 拍的一个纪录片，名字叫《Rediscovering WWII's female 'computers'》。

为解决“插线板编程”麻烦的问题，1949 年，在冯·诺伊曼的建议和规划下，制造出了一台有内存的计算机 EDSAC。与“插线板编程”不同，这台有了内存，也就有了现代意义上的编程，变量才得以被发明出来。这就是我们教科书上讲的冯·诺依曼架构计算机，这种计算机是现代计算机的基础，几乎所有的通用计算机都采用了这种体系结构。

冯·诺依曼架构，也称为通用存储程序计算机，之所以叫通用，是因为以前的电脑是“不通用”的，比如 ENIAC 只能做“特定”的任务，如果要改 ENIAC 的任务类型，就要重新“插线版编程”，重新更改数千条外部电缆的布线。而“通用”的计算机，更改任务的时候，只需要更改内存中运行的指令和数据即可，不用更改外部电缆。之所以能完成通用的任务，是这种计算机将程序和数据存储在同一块内存中，程序可以按顺序从内存中读取指令，对数据进行执行，并将结果重新存回内存中。

由于内存中同时存放着指令和数据，如果你是程序员的话，如果查找数据呢？答案是只要知道地址就可以了。但是如果只用内存的地址来寻找数据的话，会给程序员带来很大的记忆负担，不如给它取一个名字吧，于是变量随之出现。最初的变量就是用来指向内存中的一段数据。有爱好者做出了 EDSAC 计算机的模拟器，大家可以搜索 The Edsac Simulator 在网页上体验一下世界上第一台可编程的通用计算机。剑桥大学提供的一份名为《EDSAC Initial Orders and Squares Program》的文档，这份文档给出的编程示例，已经出现了变量，当时的变量，更像是帮助记忆的符号。这并不意外，**因为变量最初的作用，就是助记符**。

Perl 语言的设计者 Larry Wall 在其著作《Programming Perl》中说过，优秀程序员的三大美德：懒惰、急躁和傲慢。虽然有点言过其实，但是很多编程语言的特征就是本着让程序员轻松一点、舒服一点的目的发明出来的。如果你不想每次都记住内存的地址，那就给这一段地址发明一个名字，这就是变量名的来历。

如果你看过前文提到的剑桥大学的那份文档，会发现里面的代码实在是太复杂了，通过注释我们还是可以看到变量是什么意思。为了说明变量的来源与作用，我试图用计算机系学生都会学，但是基本不会在工作中使用的汇编语言演示一下，让大家知道为什么变量有用，引入了变量以后，为什么可以让程序员轻松一点。如果你是初学者，这一段可以跳过不用读。

每款 CPU 都有自己的机器语言，只有机器语言才能在 CPU 上运行，但是机器语言对人类实在太不友好了，没人会用 0 和 1 来写软件。我以流行的 X86 汇编举例，如果要执行这样一条命令：将一个 8 位的值移动到寄存器 AL 中，AL 寄存器的地址是 000，这个 8 位的值是十进制 97，用机器语言是这样写的：

10110000 01100001

上面这串数字后面的 01100001 是 97 这个数字，前面的那 8 位二进制要分成两部分，10110 在机器语言中代表“移动”，随后的三位 000 是 AL 寄存器的地址，所以 10110000 这一串的意思是“将一个值移动到 AL 寄存器中”。用这个太麻烦了，一不小心可能连 0 和 1 都写错了，所以，可以用十六进制稍微简化一下，进制的内容在本章后面有详细讲解：

B0 61

这比用二进制好写一点，但是仍然没有太大改观。于是，人们想起了汇编语言，用 mov 来代替 10110，用 AL 寄存器的名字来代替这个寄存器的地址 000，至于 97 这个值，就用十六进制写吧，于是汇编就有了一点可读性：

MOV AL, 61h

用 AL 代替 000 这个地址，意义重大！套用第一个登上月球的阿姆斯特朗的一句话：“这是汇编语言的一小步，却是编程历史上的一大步”，有了变量以后，人们可以不用再记内存真实的物理地址了，只需要记住一个名字即可。

有读者可能会想了，既然冯诺依曼架构的计算机内存中有数据和指令，指向内存中数据的叫变量，那指向内存中指令的部分叫什么？先不要着急，这一章咱们先把变量搞清楚，指向内存中指令的部分有什么用，留到第 7 章数组和第 8 章方法再讲。

Java 也有变量，并且变量被赋予了更多的意义，其内涵不仅仅是一个存储地址的助记符。Java 的变量要从更多的维度来考虑，除了名字，还要从类型、作用域、生存周期和地址这几个方面来考虑。

接下来，将从这几个方面全方位的对 Java 变量进行学习。**变量需要名字才可以使用，要想使用变量，需要先声明，赋予变量“数据类型和变量名”。**数据类型决定了变量中可以存储什么类型的值，变量名则是用来标识变量的名字。变量名遵循一定的命名规则，接下来先学习变量名。

4.2 变量名

在 Java 中声明变量的时候，对变量名有一定的要求，也就是所谓的命名规范。命名规范不仅可以使程序易于阅读，易于理解，还可以附加更多的信息，例如，通过名字可以推测该标识符是变量、常量、包还是类，这对于理解代码非常有帮助。Oracle 的官方网站上也给

出了命名规范的建议，大家可以搜索“Naming Convertions”来找到相应的文档。

4.2.1 变量名以小写字符开始

在 Java 中，字符的内涵要比 C/C++宽泛的多，除了 ‘A’-‘Z’、 ‘a’-‘z’, ‘_’和 ‘$’以外，任何 Unicode 字符都是合法字符，因此在 Java 编程中，实际上可以使用各种奇怪的变量名，接下来的程序中会有展示。**但是官方的建议仍然是变量以小写字符开始，这也符合程序员的工作习惯，带来的好处是每个变量名可以比较容易的用正则表达式来处理。**

当确定了变量名以后，就可以声明变量了，声明变量的方式如下：

数据类型 变量名;

其中的数据类型要到下一小节讲，目前仅需要知道，当变量被指定某种数据类型的时候，变量中只能存储指定的类型。比如当变量被指定 int（整数）类型时，该变量只能存整数。

接下来，通过给变量不同的变量名，测试一下什么样的变量名在 Java 中是“合法”的。

代码清单 4-1 变量名可以用什么字符

```
1  // 变量名可以用什么字符 chapter4/Characters/HelloWorld.java
2
3  public class HelloWorld {
4
5      public static void main(String[] args) {
6          String s = "Hello, World";
7          String 你好 = "你好, ";
8          String prénom = "世界";
9          String 🐶🐱 = "你瞅啥?";
10         String $php = "PHP 风格";
11
12         double π = 3.14;
13
14         System.out.println(s);
15         System.out.println(你好 + prénom);
16         System.out.println(🐶🐱);
17         System.out.println($php);
18         System.out.println(π);
19     }
20 }
```

上面的代码运行之后的结果如下：

```
Hello, World
你好，世界
你瞅啥？
PHP 风格
3.14
```

对变量来说，命名和赋值可以同时进行，以第 6 行代码为例，其意义是：定义一个名为 s 的字符串（String）类型的变量，并将“Hello, World”这个值赋值给变量 s。其中的等号被称之为赋值操作符（assignment operator），该操作符会将右边的值赋值给左边的变量。

上面的程序可以正常运行，这说明只要是合法的 Unicode 字符，都可以当作变量。比如第 7 行的变量是中文，第 8 行的变量中有一个字符 é 是法语的字符。第 9 行中的变量名 🐱_🐱 不是我们常用的字符，而是 Unicode 的表情符号。第 12 行中的 π 是希腊字母。

第 10 行值得强调，在 Java 中确实可以像 PHP 语言中一样以“$”开始定义变量。官方的建议不推荐如此，以美元符号$开始的变量合法，但是该“$”符号最主要的用途是用在 Java 编译器里。同理，以下划线“_”开始的变量也是合法，但是不推荐这样做。PHP 语言是最流行的编程语言之一，肯定有很多人按照 PHP 语言命名变量的方式来给 Java 变量命名，否则 Java 的官方文档不会特意强调这一点。

世界上最好的编程语言是？

Rasmus Lerdorf 出生在格陵兰岛，他曾经在雅虎任职。所说当年他要找工作，就在网上写了一份个人简历，他希望知道多少人访问了他的简历，就在家门口的餐馆用餐后，设计了一个语言：PHP。

PHP 最初是 Personal Home Page 的缩写，后来用了递归 PHP：Hypertext Preprocessor 的缩写。PHP 吸收了很多语言的语法，由于其简单易用，很快被采用，采用 PHP 写的网站一度占到 90%，所以，就有了 “世界上最好的编程语言是谁？” 的这个梗。

Rasmus Lerdorf 经常全球发表各种有关 PHP 的演讲，不过他说： “实际上，我挺讨厌编程的，但是我喜欢解决问题。”

4.2.2 变量名对大小写敏感

在 Java 中，变量名是区分大小写的，这意味着大写字母和小写字母被视为不同的字符。这意味着 helloworld 和 helloWorld 是两个不同的变量。

绝大部分流行的编程语言对大小写敏感，只有少部分如 Fortran 语言可以忽略大小写，还有一部分编程语言是混合型的，比如 PHP 语言，变量对大小写敏感，但是类名和方法名对大小写不敏感。需要注意的是，虽然这些语言对大小写不敏感，但是在编写代码时仍然需要注意一些规范，比如变量名的大小写、方法名的大小写等等。

总之，大小写对 Java 非常重要。现在有一种趋势，越来越多的编程语言（Haskell、Prolog、Go 语言），已经赋予大小写更多的内涵，比如在 Go 语言中，标识符的大小写是有意义的，并且遵循了一定的命名规范，通过字母的大小写来确定访问权限。一般而言，以大写字母开头的标识符是公开的（可以被其他包访问），而以小写字母开头的标识符是私有的（只能在当前包内访问）。目前 Java 没有采取这种机制，仍旧需要使用保留字，我将会在第 11 章的时候讲这方面的内容，现在先提前剧透一下，对比一下 Go 和 Java 的不同。

4.2.3 保留字不能作为变量名

Java 的保留字是指在 Java 语言中具有特定含义和功能的一组单词。这些保留字不能用作变量名，也不能用作方法名或类名等。我从 Oracle 的官网上找了这些关键字：

表 4-1 Java 的保留字列表

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	ex	ends	int	short
try	char	final	interface	static
void	class	finally	long	strictfp
volatile	const	float	native	super
while				

Java 的保留字不用特意去记，随着编程经验的增加，这些保留字会自然而然的记住。当看到这些保留字，脑海中出现相应的使用场景时，就说明已经掌握了 Java 这门编程语言。

4.2.4 什么是好的变量名？

每一种语言都有自己约定成俗的命名规则，像 Google 这种公司会对不同语言给出相应的编程风格指南，其中也包括 Java。在 Java 的编程风格指南里，有一部分关于变量命名的。我们可以学习一下 Google 公司认为的好的命名规则。如果大家想看详细的内容，请自行搜索 Google Style Guides 查看相应的文档，在此我仅列出三条。

1. 变量只能使用 ASCII 字母和数字，这样的话每个变量名可以用正则表达式来处理。

2. 局部变量以 lowerCamelCase 风格编写。

3. 常量名的字母全部是大写，用下划线来分割。

Google 讲的肯定有道理，下面是我认为的好名字的原则，供大家参考，希望能帮助大家形成自己的命名规则。

1. 名字要有意义

取一个好的名字可以给出大量的信息，一个好的名字可以准确完整的描述该变量所代表的内容。这样的名字非常容易阅读，没有晦涩的缩写，不产生歧义，同时也应该容易记忆。

像在第三章中所举的 FIFA 游戏的例子，如果要用一个变量表示球队的球员总数，可以是 numberOfPlayersOnTheTeam。这个名字就比一个单纯的 number 要好得多。如果要表示球员的号码，可以使用 playerNumber。我认为这个名字仍然要比一个单纯的 number 要好。因为 number 可能表示的意思实在太多了。如果要用一个变量代表球员一年的薪水，可以用 salary 或者 annualSalary，而不应该用 s 或者 as，如果用 as 来代表 annualSalary 的缩写，就太短了，也没有描述性，很少有人会想到 as 会代表 annualSalary。

名字有了意义以后，还要注意变量名最好是名词，不应该是动词。在 Java 这种面向对象的编程语言中，变量代表的是对象，对象一般是名词，比如球员，薪水，球队等都是名词。要避免给变量取动词为名，比如射门，停球这种动作。

总之，取个好名字并不容易，需要不停的权衡。幸好一个坏名字也不难被识别，基本上，类似于 temp、x1、x2 这样的名字，几乎都是坏名字。还有一个问题，名字取多长才算合适呢？

2. 变量名要简洁明了

变量名对长度没有要求，最重要的是要清晰，长度不要太长。举个例子，Java 中有一个很出名的框架叫 Spring，我觉得几乎所有的 Java 程序员最终都会了解到这个框架，这个框架里有个类的名字叫 SimpleBeanFactoryAwareAspectInstanceFactory，这个名字就有点太长了。

Spring 框架介绍

只要你是 Java 程序员，那么有很大的可能你要用到一个叫 Spring 的框架，目前该框架是 Java 最受欢迎的框架，在学完 Java 之后，非常有可能马上用到 Spring。

该框架始于 2004 年，最初叫 Spring Framework，以开源形式发布的框架。现在 Spring 有了长足的发展，已经包括 Spring Boot、Spring Cloud 等，已经成了一个 Spring 家族，其设计高度模块化，有强大的生态环境。

其创始人 Rod Johnson 是个传奇人物。他拥有计算机学位和音乐学的博士学位，是计算机界最懂音乐，音乐界最懂计算机的跨界奇才。当时大家都用臃肿的官方正统的 J2EE，效率很低。于是，他写了一本书，该书在 Java 界也是大名鼎鼎：《Expert One-on-One J2EE Development without EJB》，推荐大家阅读。

该书的书名直接点出了不用 EJB，这在当时让众多 J2EE 架构师大跌眼镜，如果 J2EE 中不用 EJB，那类似于鱼丸粗面里没有鱼丸。这是 Rod Johnson 的一贯作风，后来他创建了 Atomist 公司，目前仍然为 Spring 提供更先进的功能。

在 1990 年，三位科学家 Narasimhaian Gorla, Alan C. Benander 和 Barbara A. Benader 在 IEEE 上发表了一篇名为《Debugging Effort Estimation Using Software Metrics》的论文，这篇论文里，作者详细分析了什么样的代码更容易被调试。虽然这篇论文已经过去 30 年了，当时开发软件的工时刚刚超过开发硬件的工时，现在开发软件用的工时早就已经远超开发硬件用的工时了，所以，站在软件更耗费时间的角度上，这篇论文还应该读一下。

除了代码中的空行，goto 语句等因素，论文也提到了变量名的长度。作者认为把变量名限定在 8 到 20 个字符比较容易调试。本着“尽信书不如无书”的原则，如果我们能保证超短名字的意义足够清晰，或者只有超长的名字才能让意义清晰，那也可以用这些名字。

Google 公司建议的 Java 编程风格里有这样一条：“一个项目可以选择一行 80 个字符或 100 个字符的列限制，除了下述例外，任何一行如果超过这个字符数限制，必须自动换行。”很多的编程语言中都有 80 个字符的限制，[30]像上面那种超长的名字，如果严格按照 80 字符的限制，光名字就要写 2 行多，显然是不太合适的。

[30] 之所以有这个限制有两种说法，一种说法是因为在上个世纪计算机显示器不发达，有些终端每列最多只能显示 80 x 24 个字符，所以编程一旦超过 80 个字符，将会没法显示。另一种说法是更早期的 IBM 的纸带就是 80 列。我不知道哪个是真实情况。现在主流的 IDE 上都可以设置在第 80 列的时候显示一条竖线。

4.3 基本数据类型的变量

Java 变量的数据类型可以分成两大类，**一类是基本数据类型（Primitive Data Type），另一类是引用数据类型（Reference Data Type）**。基本数据类型的值是简单的数据值，例如数字、字符、布尔值等。而引用数据类型是一种更为复杂的数据类型，它们可以包含任意数量和类型的数据，例如类、接口等，可以用于存储更为复杂的数据结构，例如图形、列表、树等。

为什么要把数据类型分为这两大类呢？一般的解释是：Java 中将数据类型分为基本数据类型和引用数据类型是为了更好地管理和处理数据。基本数据类型直接存储在内存中，具有较高的效率。但是基本数据类型的这八种类型，又都可以被引用数据类型所处理，对初学者来说，非常容易引起误解。因此，很多计算机界的前辈像 Bruce Eckel 认为这是 Java 语言设计上的重大失误，如果 Java 不提供基本数据类型，那么其数据类型就要比现在统一的多。我认为 Bruce Eckel 说的非常有道理。但是 Java 不是这样设计的，我们还是要学基本数据类型和引用数据类型，下面我画一张图，希望能让读者明白其中的关系。

图中的左边是基本数据类型，图中的右边有包装类型，基本数据类型和包装类型所处理的数据是一样的，功能也是类似的。而且基本数据类型和包装类型在 Java 1.0 的时候就同时发布了，为什么会发布功能类似的两种类型呢？并没有官方解释，一个勉强的解释是基本数据类型运行效率高，还有一个解释是 C/C++中有基本数据类型，索性 Java 中都学习过来了。因为有这两种类型，一部分程序员用这个，另一部分程序员用那个，比较混乱，后来怎么处理的呢？

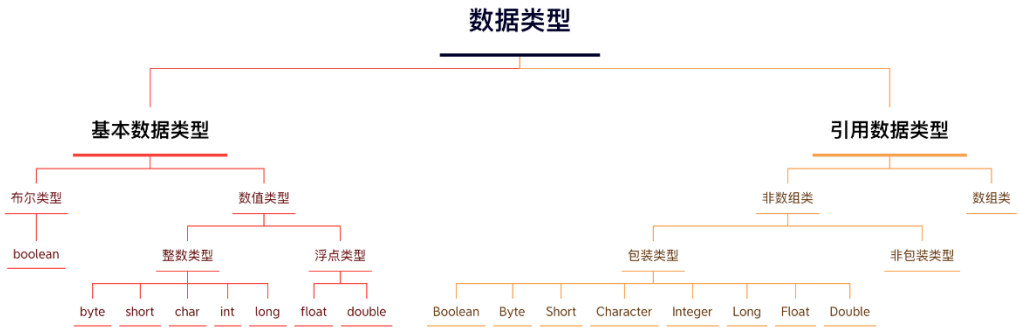


图 4-1 Java 数据类型示意图

鉴于两种基本数据类型一直混用，断然取消掉基本数据类型是不可能的，长期分裂也不合适，于是从 Java 5 开始，提供了一种机制，可以自动的将基本数据类型包装成引用数据类型，这种机制叫作自动装箱/拆箱机制。这种机制采用了一种特殊的引用类型用于封装基本数据类型，这种特殊的引用类型叫包装类型（Wrapper Type）。包装类型是 Java 中用于封装基本数据类型的类，它提供了一些方法和属性，用于对基本数据类型进行操作和处理。Java 中的每个基本数据类型都有对应的包装类型。

基本数据类型总共有八类，分别是：boolean 类型、char 类型、byte 类型、short 类型、int 类型、long 类型、float 类型和 double 类型，其中，byte、short、int、long、float 和 double 是数值类型，而 char 是字符类型，boolean 则是布尔类型。

包装类型的名称与对应的基本数据类型名称相同，首字母大写，分别是：Boolean 类型、Character 类型、Byte 类型、Short 类型、Integer 类型、Long 类型、Float 类型和 Double 类型。

表 4-1 基本数据类型以及对应的包装类型

基本数据类型	对应的包装类型
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

自动装箱指的是将基本数据类型自动包装成对应的包装类对象，而自动拆箱则是将包装类对象自动转换为对应的基本数据类型。通过这种机制，可以提高程序的编写效率和代码的可读性，但在对性能和内存占用要求较高的场合，应该谨慎使用。在编写代码时，应该根据实际需求和场景，选择合适的数据类型和数据转换方式。我认为，除非有特别明确的要求，否则应该一律使用包装类型，而应该放弃使用基本数据类型。只是长期以来形成的习惯，大量的程序员依旧使用基本数据类型，所以我们仍旧要都要学习一下。

接下来，将分别学习这两种变量的数据类型：基本数据类型和引用数据类型，先从基本数据类型讲起。

4.3.1 布尔类型

先从最简单的开始， Java 中的布尔类型只有两个可能的取值：true 和 false。布尔类型用于表示逻辑上的真和假，通常用于表示条件的结果或标志状态。例如，在控制流语句（if、

while、for 是控制流，在第 6 章会详细介绍）中，我们可以使用布尔表达式来决定程序的执行路径。

布尔类型最初是由 ALGOL 60 引入的，在 1960 年后设计的主流编程语言几乎都包含布尔类型。值得一提的是 C 语言，C 语言的第一个标准化版本 C89（也被称之为 ANSI C）中，没有包括布尔类型，采用的方法是用非零的 int 值来表示 true，用 0 表示 false。1999 年发布的另一个 C 语言标准化版本 C99 才增加了布尔类型，以提供一种更自然的方式来表示布尔值。

> Alan Perlis 与 ALGOL 语言
>
> Alan Perlis 是 1966 年首届图灵奖得主，他是 ALGOL 语言的设计者。Alan Perlis 对计算机科学的贡献巨大，在他之前，计算机科学从属于其它学科，在他的倡议下，计算机科学成为一门独立的学科。
>
> 1965 年，卡内基梅隆大学成立了计算机系，这是世界上第一个独立的计算机系。Alan Perlis 担任第一任系主任。多年以后，Java 的创始人就读于卡内基梅隆大学。
>
> ALGOL 语言是最早期的语言之一，今天我们用的 C/Java 都属于 ALGOL 语言家族的。比 ALGOL 语言早一点的语言有 Fortran 和 Lisp。Alan Perlis 发现 Fortran 语言不支持递归，他决定让 ALGOL 完美支持递归。但是这谈何容易，处理递归要小心的安排每个方法每次调用的地址和上下文，即使当时最顶尖的科学家，心里也没底。
>
> 于是，写《计算机程序设计艺术》这套书的 Donald Knuth 给出了一个测试用例，这个测试叫“Man or boy test”，大家可以自行搜索，这个测试是看 ALGOL 的编译器有没有实现正确的递归与外部引用。只是当时这些科学家都没写出正确的编译器，没人知道正确答案，所以 Donald Knuth 自己给出了一个答案是-121，后来，正确的编译器写出来了，正确答案其实是-67。
>
> 我实在想讲清楚“Man or boy test”这个测试，但是这与本书的关系实在是有点远，如果读者有兴趣研究编译器，这个测试非常的有意思。

这一小节不再详细展开布尔类型的介绍，在本书的第 5 章和第 6 章，都会涉及到布尔类型。

为了加深印象，我讲个故事吧。这个故事是真实的，美国有个作家兼演员，她的名字叫 Rachel True。她是苹果用户，于是就注册了一个 iCloud 账户，随后，她的账户被封掉了。

苹果长达半年没有理会她的投诉，给出的原因是苹果的系统中，姓不能设置为布尔类型 true。她在她的社交媒体上公布了 iCloud 的错误提示如下：

Type error: cannot set value `true` to property `lastName`……

这条消息出来以后，有好事者尝试把姓改为 false，也被 iCloud 封了。主要原因还是由于对程序员来讲布尔类型太普遍了，以至于没有程序员认为会有这样的姓，这算是一种意外的 bug，不知道后续苹果公司会如何处理，大家可以在网上关注一下，写本书的时候，苹果公司还没有给出任何解决方案。

4.3.2 字符类型

在 Java 中，char 类型是一个 16 位无符号整数，其取值范围为 0 到 65535，用于表示 Unicode 字符。Unicode 是一种标准字符集，它包含了世界上几乎所有的字符，包括字母、数字、标点符号、特殊符号等等。

每个 char 类型的变量可以存储一个 Unicode 字符，字符要使用单引号，例如：

char c = ‘a’;

上面的代码将字符 ‘a’存储在变量 c 中。注意，单引号中只能包含一个字符，如果包含多个字符或者没有字符，编译器将会报错。

接下来讲一下 Java 中字符类型的变量可存储的内容与可进行的操作，先来讲可存储的值，字符类型的变量可以存储两种类型的值：一是字面量（literal），二是转义字符（Escape Character）。

1. 字面量

   字面量（literal）是指用于表示某种特定类型的常量值的符号或文本，字面量可以表示各种数据类型，例如整数、浮点数、布尔值、字符等。字符类型的字面量要用单引号括起来，例如'A'、'栋'或'\n'（换行符）。也可以使用 Unicode 转义序列表示，例如'\u0041'（等于'A'）。Unicode 转义序列通常以反斜杠（\）开头，并紧随其后的是一个表示 Unicode 码点的十六进制数字。

2. 转义字符

   在计算机编程中，转义字符指的是一些特殊的字符，用于在字符常量或字符串中插入一些特殊的字符或字符序列。转义字符通常以反斜杠（\）开头，后面跟着一个或多个字符，如换行符（\n）、回车符（\r）、制表符（\t）等。这些字符也可以直接存储在字符类型的变量中。

对字符类型的变量，一般可以进行如下三种操作：

1. 算术运算

   字符类型的变量可以进行算术运算，例如+、-、*和/等。这时，字符类型的变量会被隐式转换为整数类型（int），然后进行相应的运算。如果需要将结果重新赋值给一个字符类型的变量，请使用显式强制类型转换，例如：(char)(c + 1) ，其中 c 是一个字符类型的变量。

2. 关系运算

   可以使用关系操作符，如<、>、==等，来比较两个字符类型的变量。这时，比较的是两个字符的 Unicode 编码值。

3. 类型转换

   可以将字符类型的值转换为其他基本数据类型的值，例如 int、long、float、double 等。在进行类型转换时，需要注意可能的精度损失和数值溢出问题。

代码清单 4-2 对字符类型的变量进行的操作

```
public class CharExample {
    public static void main(String[] args) {
        // 定义一个字符类型的变量，并赋值
        char c1 = 'A';
        System.out.println("c1: " + c1);

        // 使用 Unicode 转义序列表示字符
        char c2 = '\u0041';
        System.out.println("c2: " + c2);

        // 使用转义字符表示特殊字符
        char newline = '\n';
        System.out.println("c1 and c2 on separate lines:\n" + c1 + newline + c2);

        // 算术运算
        char c3 = (char) (c1 + 1);
        System.out.println("c3: " + c3);

        // 关系运算
        if (c1 < c3) {
            System.out.println("c1 is less than c3.");
        } else {
            System.out.println("c1 is not less than c3.");
        }
    }
}
```

以上代码运行以后输出的结果如下：

```
c1: A
c2: A
c1 and c2 on separate lines:
A
A
```

```
c3: B
c1 is less than c3.
```

以上就是字符类型经常用到的操作，掌握了这些操作，就可以解决大部分有关字符类型的任务。

但是关于字符类型，其背后的故事远不止这些，如果你想了解字符类型背后被人忽视的历史，可以继续读下去。假如你是一个历史学家，研究早已失传的文字，比如起源于公元前 3200 年前的现在已经失传的埃及象形文字，如何在电脑上处理这些埃及象形文字？如何将下面这一行埃及象形文字通过 Java 输出到屏幕上？

图 4-2 埃及象形文字

要解决这个问题，先从字符编码的历史谈起。

1. 字符编码的历史

我们在看电影的时候，有时候会看到电影里面的人用手电筒打出 SOS 的信号，那种编码叫摩尔斯码，是摩尔斯于 1836 年左右发明的。摩尔斯码是一种主要用于传输电报的编码方式，曾经被广泛的使用，直到二十世纪中叶才被逐渐取代。

随后法国工程师博多（Baudot）于 1870 年发明了一种二进制编码——博多码，使用 5 位二进制数来表示一个字符，因此总共可以表示 $2^5=32$ 种组合。由于 32 种组合能表示的字符集十分有限，因此采用了不同的状态：字母状态和数字状态。在字母状态下，编码表示字母和部分符号；在数字状态下，编码表示数字和另一些符号。博多码后来被广泛的应用于电报通信和电传打字机领域，如今早已被逐渐取代，但是博多码引入用来切换数字状态和字母状态的 Shift 键仍旧占据着键盘上最重要的位置，影响着我们每一个人。

以电传打字机为标志的电传网蓬勃发展了 20 年后，时代进入了计算机时代。1949 年，名为 EDSAC 的计算机被发明出来，EDSAC 计算机使用的编码叫 EDSAC 编码，这个编码借鉴了博多码，同样采用 5 位二进制来表示一个字符，同样有一个 Shift 键来区分数字状态与字母状态。

EDSAC 出现后的十年间，出现了各种各样的计算机，它们使用的字符编码各不相同。

这样导致的结果是不同型号的计算机无法正常读取彼此之间的数据，需要进行转换，十分不方便。

如果有个第三方的编码方案就好了。基于这种想法，ASCII 编码由 IBM 的一个名为 Bob Bemer 的员工在 1961 年向 ANSI（American National Standards Institute）提出。ASCII 编码的英文是 American Standard Code for Information Interchange，翻译为中文是信息交互的美国标准符号，其内涵不言而喻，就是为了信息交互。两年后，ANSI 发布了第一个版本的 ASCII 标准。随后，ASCII 编码发布了几个版本，下面这张图是 1972 年的 ASCII 版本。

USASCII code chart

Bits b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 ↓ b_3 ↓ b_2 ↓ b_1 ↓ Row ↓ Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

图 4-3 1972 年版本的 ASCII 码

如图所示的 ASCII 码图，可以看到主要用于表示英文字符和控制字符。ASCII 编码采用 7 位二进制数表示一个字符，因此最多可以表示 128 个字符，包括 0-9 的数字、26 个英文字母、标点符号和一些控制字符。

在 ASCII 编码中，每个字符都有一个唯一的数字编码，这些编码可以被计算机系统识别和处理。例如，大写字母 A 的编码是 65（二进制编码为 01000001），小写字母 a 的编码是 97（二进制编码为 01100001），数字 0 的编码是 48（二进制编码为 00110000），标点符号 ！的编码是 33（二进制编码为 00100001）等等。虽然 ASCII 编码只能表示有限的字符集，但它仍然是计算机编码的基础，对后来的字符编码方式有着很大的影响。在计算机文本处理、网络通信等方面，ASCII 编码仍然是一种常用的编码方式。

但随着计算机技术的发展和应用领域的扩展，人们需要更多的字符集来表示更多的字符和符号。因此，出现了 ASCII 扩展字符集，包括 ASCII 扩展字符集 1、ASCII 扩展字符集 2 和 ASCII 扩展字符集 3。编码方式随着不同的语言也有所不同，处理简体中文需要 GB2312 编码，处理繁体中文需要 Big5 编码，处理俄语需要 KOI8 编码……诸如此类，不胜其烦。

于是就有了 Unicode 的方案，Unicode 有各种翻译，比如统一码、万国码、国际码、单一码等，从这些翻译中，应该能看出其雄心壮志：要设计一种能编码全世界所有字符的编码。

2. Unicode 编码

1987 年，由施乐公司的 Joe Becker、苹果公司的 Lee Collins 与 MarkDavis 一起，提出了一个宏伟的计划，该计划的目标是涵盖所有主流的字符。并且于 1988 年发布了 Unicode，史称 Unicode 88。[31]在 1988 年发布的这个文档中写道： “Unicode 旨在用可行可靠的方式满足世界范围内对文本的需求。粗略来讲，Unicode 可以认为是宽体的 ASCII 码，已经被扩展为 16 位。”

让我们简单的计算一下，16 位可以容纳 65536 个字符，范围从 U+0000 到 U+FFFF。这 65525 个字符中包括了全世界大多数常用语言的字符集，如拉丁字母，希腊字母，西里尔字母，阿拉伯字母，希伯来字母，汉字，日文平假名和片假名等等。每个字符在 Unicode 编码标准中都对应唯一的数字，这个数字称之为码点（Code Point）。在 Java 中，Unicode 的码点通常用“\u”加上一个十六进制数字来表示，例如“\u0041”表示拉丁字母 A 的码点，而“\u680B”表示汉字“栋”的码点。如下的代码会输出汉字“栋”：

代码清单 4-3 通过码点输出汉字“栋”

```
public class PrintEgyptian {
  public static void main(String[] args) {
    char c = '\u680B';
    System.out.println(c);
  }
}
```

最新的 Unicode 15 中，包含的字符已经接近 15 万，显然早已超出了最初的 65536 个字符。在 Unicode 设计之初，设计者们就预料到了 65535 个字符无法满足要求，已经考虑到了多平面机制。Unicode 定义了 17 个平面（Plane），每个平面都包含了一组字符集合，不同平面的字符范围和编码方式可能不同。其中，第一个平面是基本多文种平面，包含了最常用的字符集合，也是最早的 Unicode 字符集。其他平面则包含了一些较不常用的语言、符号、表情符号等。

那回到前面我们提出的问题，如何表示埃及象形文字呢？

[31] 1988 年发布的 Unicode 文档的 PDF 文件我已经放在本书辅助资料 chapter4 文件中。之所以叫 Unicode，是因为想让这个编码 Unique，Unified 和 Universal。

3. Java 中如何处理埃及象形文字

大家应该想到了，每个埃及象形文字在 Unicode 系统中也都有自己对应的码点。埃及象形文字早就没人使用了，属于不常用的字符，不常用字符在辅助平面中。Unicode 定义了 16 个辅助平面，每个辅助平面包含了 1,048,576 个字符，码点范围从 U+10000 到 U+10FFFF，使用 20 位（4 个字节）来表示每个字符的编码。Java 内部使用的是 UTF-16 编码，它的编码规则非常简单：基本平面的字符占用 2 个字节，辅助平面的字符占用 4 个字节。

通过在 Unicode 官方网站上搜索 Egyptian Hieroglyphs，可以找到埃及象形文字的码点，从 U+130E0 到 U+130E5。

图 4-4 象形文字的码点

这些码点已经超出了基本平面的范围（U+0000 到 U+FFFF），因此需要用 4 个字节来表示。那么问题来了，UTF-16 编码长度要么是 2 个字节，要么是 4 个字节，如果遇到 2 个字节，Java 把这 2 个字节当成一个字符还是 4 个字节中的一半呢？

Unicode 的解决方法很简单，在基本平面内，U+D800 到 U+DFFF 这段长为 20 位的空间内不对应任何字符，这段空间用来映射辅助平面的字符。将辅助平面内的 20 个二进制分为两半，前 10 位映射在 U+D800 到 U+DBFF 之间，称之为高位；后 10 位映射在 U+DC00 到 U+DFFF 之间，称之为低位。这样，一个辅助平面的字符，就可以映射为两个基本平面的字符，在 Java 中，只要碰到码点在 U+D800 到 U+DBFF 之间，就可以断定这个码点要与后面两个码点一起解读。

当我们查询后得知某个辅助平面的字符的 Unicode 码点以后，可以将其转换为高位与低位，转换的方法不再赘述，在网上有很多现成的在线工具供大家使用。也可以通过下面的代码完成转换：

代码清单 4-4 将码点转化为代理字符对

```
import java.util.Scanner;


public class SurrogatePairDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("输入十六进制的码点：");
        int codePoint = Integer.parseInt(scanner.nextLine(), 16);
        scanner.close();
```

```
        if (Character.isSupplementaryCodePoint(codePoint)) {
            char[] surrogatePair = Character.toChars(codePoint);
            char highSurrogate = surrogatePair[0];
            char lowSurrogate = surrogatePair[1];
            System.out.printf("码点 U+%04X 转化为代理字符对的高位与低位分别
为: %04X %04X%n", codePoint, (int)highSurrogate, (int)lowSurrogate);
        } else {
            System.out.printf("U+%04X 不是合法的 Unicode 码点.%n", codePoint);
        }
    }
}
```

运行上面的程序，输入十六进制 130E0，可以得到如下的结果：

```
输入十六进制的码点: 130E0
码点 U+130E0 转化为代理字符对的高位与低位分别为: D80C DCE0
```

同样的方式，可以得到前面提到的希腊象形文字的代理字符对。由于单个字符类型只能表示基本平面中的代码点，因此不能直接使用字符类型打印或操作补充平面中的字符，需要用如下的代码来转出补充平面的 Unicode 字符。

代码清单 4-5 输出希腊象形文字

```
public class PrintEgyptian {
  public static void main(String[] args) {
    System.out.print("\uD80C\uDCE0 ");
    System.out.print("\uD80C\uDCE1 ");
    System.out.print("\uD80C\uDCE2 ");
    System.out.print("\uD80C\uDCE3 ");
    System.out.print("\uD80C\uDCE4 ");
  }
}
```

将上面的代码编译运行以后，会有输出如下的输出：

图 4-5 运行后的结果

如果你运行后没有显示出上面的图形，而是显示问号或者乱码，那可能是你的电脑没有安装相应的字库文件。一般来说，电脑里不会安装显示希腊象形文字的字库，用关键字“Egyptian Unicode Font”来搜索一下，会发现有几个不同的设计师设计的字体，我安装的是：Noto Sans Egyptian Hieroglyphs 这个字体。

好了，你现在已经了解了如何处理希腊象形文字，那处理起相对比较常用的 emoji 表情符号，就更不在话下了。

4.3.3 整数类型

整数类型是最常见的数据类型，用于表示一定范围内连续的整数。为了满足不同的应用场景和需求，Java 提供了四种不同的整数类型，分别是 byte、short、int 和 long，每种整数类型都有其特定的取值范围和存储空间大小。

为什么 Java 要提供四种不同的整数类型？提供不同的整数类型，最主要的是为了节约存储空间以及提高性能，占用的空间越小，运算的速度越快。比如在进行文件与网络传输的时候，使用 byte 类型可以节省传输的数据量。

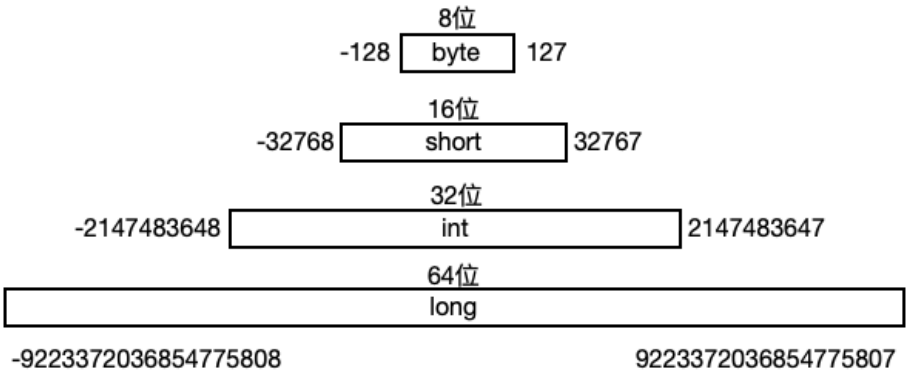


图 4-6 不同整数类型的存储空间和表示数值的范围

```
public class IntegerTypes {
    public static void main(String[] args) {
        byte b = 127;       // byte 类型
        short s = 32767;    // short 类型
        int i = 2147483647; // int 类型
        long l = 9223372036854775807L; // long 类型

        // 输出各个整数类型的值
        System.out.println("byte: " + b);
        System.out.println("short: " + s);
        System.out.println("int: " + i);
        System.out.println("long: " + l);
```

```
        // 进行加法运算
        System.out.println("b + 1: " + (b + 1));
        System.out.println("s + 1: " + (s + 1));
        System.out.println("i + 1: " + (i + 1));
        System.out.println("l + 1: " + (l + 1));
    }
}
```

上面的代码运行以后，会输出如下的结果：

```
byte: 127
short: 32767
int: 2147483647
long: 9223372036854775807
b + 1: 128
s + 1: 32768
i + 1: -2147483648
l + 1: -9223372036854775808
```

在上面的代码中，我们定义了四个变量分别表示不同类型的整数。注意，在定义 long 类型的变量时，需要在数字后面加上 L 或 l，表示这是一个长整数。然后，我们输出了各个整数类型的值。随后对它们进行了加法运算，分别对这四个变量进行了加 1 操作，然后再输出加 1 后的值。

对变量进行加 1 操作以后，会发现对 byte 类型和 short 类型来说，输出的结果是正确的，但是对 int 类型和 long 类型来说，输出结果是错误的，为什么会有这样反常的结果呢？这时候就要了解整数类型一个常见的错误：溢出。

前面已经提到，对四种整数类型来说，它们主要的区别是不同的数值类型所定义的存储空间不同。记得国内有一部电影，有一个桥段发生在星巴克咖啡店，主人公和星巴克的员工争论大杯、中杯和小杯的问题。作为常识，大杯的咖啡倒入小杯子，会溢出。类似的，在计算机中，大的数值，存入小的存储空间中，也会溢出。

图 4-7 星巴克的杯子

在 Java 中，溢出是指当一个变量超出了其数据类型所能表示的最大值或最小值时发生的现象。由于数据类型的限制，变量无法表示比其数据类型的最大值还要大的数值，或比其数据类型的最小值还要小的数值。当变量的值超出了这个范围时，就会发生溢出。溢出是最难调试的 bug 之一，溢出产生的 bug 已经造成了无数的损失。

比如赫赫有名的 Y2K 千年虫问题就是整数的溢出。Y2K 千年虫是指 1999 年底至 2000 年初，由于计算机中使用了只有两位表示年份的日期格式，导致在 2000 年时计算机会将年份错误地解释为 1900 年，从而引发的一系列可能导致计算机系统故障的问题。因为计算机中很多程序都使用了这种日期格式，因此人们一度担心 Y2K 千年虫可能引发全球范围内的灾难，包括航空、金融、交通等领域的系统瘫痪。为了避免这种情况发生，全球各地的计算机专家和程序员投入了大量的工作，对涉及到日期计算的计算机程序进行了修复和升级，以使其能够正确处理 2000 年及以后的日期。最终，全球范围内的计算机系统在 2000 年 1 月 1 日顺利地度过了 Y2K 千年虫的威胁。

再回到我们的前面所说的程序，为什么 byte 类型和 short 类型不会溢出，而 int 和 long 会溢出呢？除了溢出，在 Java 中还有另一个规则：**byte 类型和 short 类型在进行算术运算时会自动提升为 int 类型。**为什么会有这样一条规则呢？当年在 Java 语言和 Java 虚拟机设计时，32 位是计算机的标准字长，在使用较小的类型进行基本算术运算时，并没有性能优势。Java 虚拟机利用了这一特点，既然较小的类型没有性能优势，也就没有必要对较小的类型提供专用指令。于是 Java 虚拟机将 32 位作为 int 类型的大小，只对 int、long、float 和 double 进行算术运算提供了专用指令，并没有为任何较小的数值类型（byte、short 和 char）提供专用指令，这些较小的数值类型都提升为 int 进行操作。

当 int 类型和 long 类型进行加法操作的时候，类型并没有进行提升，所以导致上面程序中 int 和 long 溢出。

当溢出发生的时候， Java 不给出任何的异常提示，这也导致了溢出非常难以察觉。在编程的时候，一定要先估算自己能处理的数据范围，以防止溢出的发生。如果意识到所用的类型可能会溢出，就用一个更大的类型。比如 int 不够的时候，可能考虑用 long 类型。如果

long 类型还不够用，再考虑用 BigInteger 类型，BigInteger 是 Java 提供的类，不是原始变量。

BigInteger 是 Java 中的一个大整数类，用于表示任意大小的整数，可以进行高精度的整数运算。由于基本数据类型（如 int、long）的取值范围是有限的，当需要处理超出这些范围的整数时，就需要使用 BigInteger 类。需要注意的是，由于 BigInteger 对象可以表示的数字范围非常大，因此它们的计算速度可能比使用基本数据类型进行计算要慢得多。

BigInteger 类提供了各种算术运算、位运算等操作，可以进行加、减、乘、除、取模等运算，还可以进行比较、求绝对值、求幂等操作。BigInteger 类型没有严格的长度，所能表达的数字只与 Java 虚拟机能使用的内存数量有关系，只要内存够大，那么所能表示的整数就可以无限大。

下面这个例子是使用 BigInteger 的例子，如果要表示比 long 类型还要大的数，可以考虑使用这种类型。

```
1  // 表示超大整数以防溢出 chapter4/big_integer/BigIntegerDemo.java
2  import java.math.BigInteger;
3
4  public class BigIntegerDemo {
5      public static void main(String[] args) {
6          BigInteger largeValue = new BigInteger(Long.MAX_VALUE + "");
7          for (int i = 0; i < 4; i++) {
8              System.out.println(largeValue);
9              largeValue = largeValue.add(BigInteger.ONE);
10         }
11     }
12 }
```

代码中的第 6 行中 Long.MAX_VALUE 是一个常量，它代表了 long 数据类型的最大值。具体来说，它的值是 2 的 63 次方减 1。第 7 行为 for 循环，在第 6 章中详细介绍 for 循环。在本例中，for 循环的作用是每次给 largeValue 这个变量加 1，总共运行了 4 次。运行以后，输出的结果如下：

```
9223372036854775807
9223372036854775808
9223372036854775809
9223372036854775810
```

可以看到使用 BigInteger 后，没有再发生溢出的情况。

如果我们不想让整数加法悄无声息的溢出，在碰到溢出的时候给出一些错误提示，那应该怎么办呢？这种机制在 Java 中被称之为异常，在本书的第 14 章会详细介绍。在 Java 中，

异常是指在程序运行期间出现的错误或意外情况，它会导致程序无法正常执行。当出现异常时，程序会抛出一个异常对象，这个异常对象包含了异常的类型、原因等相关信息。基于这种机制，Java 8 增加了 addExact 方法，确保发生溢出的时候有异常抛出。

```
1  // 整数溢出后抛出异常 chapter4/add_exact/AddExact.java
2  public class AddExact {
3      public static void main(String[] args) {
4          int value = Integer.MAX_VALUE - 1;
5          for (int i = 0; i < 4; i++) {
6              System.out.println(value);
7              value = Math.addExact(value, 1);
8          }
9      }
10 }
```

代码的第 4 行中， Integer.MAX_VALUE 是 Java 的一个常量，它代表了 int 数据类型的最大值。第 3 次循环时变量会溢出，其结果如下：

```
2147483646
2147483647
Exception in thread "main" java.lang.ArithmeticException: integer overflow
    at java.base/java.lang.Math.addExact(Math.java:825)
    at AddExact.main(AddExact.java:7)
```

可以看到，当发生溢出的时候，程序抛出了异常，这比悄无声息要好的多。这个方法要求程序员在使用加法时考虑到潜在的溢出问题，并作出处理，这样可以提高代码的健壮性。

相应的，Java 还提供了以下精确计算方法:

- addExact() - 精确加法

- subtractExact() - 精确减法

- multiplyExact() - 精确乘法

- incrementExact() - 精确递增

- decrementExact() - 精确递减

这些方法都会在计算结果发生溢出时抛出 ArithmeticException。

4.3.4 浮点类型

以上讲的都是整数，在现实中，大量的数据都是有小数部分的实数，因此如何正确地在计算机中表示并操作实数显得尤为重要。**Java 中的浮点数是一种带小数部分的数值类型，用于表示实数。**Java 支持两种浮点数类型，分别是单精度浮点数（float）和双精度浮点数（double）。编程中，有大量的 bug 来自于实数的转换错误或者精度不够，我举几个严重的 bug 来加深大家对浮点数的印象。

在 1996 年，欧洲的 Ariane 5 重型运载火箭首次测试发射，火箭在发射后 37 秒被迫自行引爆。事后调查发现是一个 64 位的浮点数在转换的过程中发生故障，从而导致火箭翻转，进而自毁程序启动。

还有一个损失重大的浮点错误发生在 Intel 公司身上，1994 年 10 月，美国 Lynchburg College 数学系的教授 Thomas Nicely 为研究孪生质数，发现用计算机处理除法时一直出错，事后发现原因是英特尔的浮点运算出现了 bug。为此，Intel 回收了出现 bug 的所有处理器，造成了巨大经济和名誉损失。

第三个例子，美国政府问责局（Government Accountability Office）发布过一份文件，这份文件里写到，在 1991 年 2 月 25 日海湾战争期间，爱国者导弹防御系统出现问题，让一颗飞毛腿导弹击中营房，导致 28 名美军士兵阵亡。对于此次事件，Robert Skeel 写了一篇详细的分析报告，[32]爱国者导弹防御系统，由于设计原因，计时系统的精确度有一个很微小的误差，对浮点数，计算机没有办法避免误差。结果每 8 个小时，系统会有 0.0275 秒的误差，当时防御系统已经运行了 100 个小时，累积了大约 0.34375 秒的误差。飞毛腿导弹的速度高达 5 马赫，每秒飞行 1.7km，这 0.3 秒多的误差，足够导弹飞行 600 米，因此造成了巨大的灾难。

在计算机发展的历史上，曾经有两件事情特别困难，一是浮点计算，二是数组下标的寻址，数组下标寻址到第 7 章数组的时候我会详细讲解。以前浮点难，现在浮点也不容易。在以前，设计浮点非常困难，Intel 发布的 80386 芯片，浮点乘法是用软件模拟的，如果想要硬件浮点乘法，需要再购买一块 80387 浮点数协处理器，如果没有 80387 芯片，想要在电脑上使用高度依赖浮点运算的多媒体是非常困难的。现在有个网站叫 top500.org，这个网站上有世界上最快的 500 台超级计算机的列表，本书写作的时候排名第一的计算机是日本富士通研发的“富岳”，最大计算性能是 442 petaflops。超级计算机的排名标准是什么呢？是每秒钟进行多少次浮点运算，单位是 FLOPS（Floating Point Operations Per Second 的缩写）。

为了更直观的感受浮点数，看看下面两个浮点数相加的代码。下面这段代码定义了两个 double 类型的变量，将 0.1 和 0.2 这两个数加起来，计算结果“应该”为 0.3。

```
public class DoubleAdder {
```

[32] 这篇报告名为《Roundoff Error and the Patriot Missile》，已经下载到补充资料中了，在 charpter4 文件夹下，该报告详细讲了事故原因。

```
    public static void main(String[] args) {
        double num1 = 0.1;
        double num2 = 0.2;
        double sum = num1 + num2;
        System.out.println("0.1 + 0.2 = " + sum);
    }
}
```

当上面的代码运行之后，实际结果输出的结果如下：

0.1 + 0.2 = 0.30000000000000004

为什么会产生微小的误差呢？这是由于浮点数在计算机中以二进制形式存储，二进制无法精确地表示十进制小数（例如 0.1 和 0.2），这就导致了在进行浮点数计算时出现了精度误差。

目前在计算机中，大部分的平台用 IEEE 754 标准来表示有小数点的实数。[33]Java 中的 float 类型的名称源于词组 floating-point，而 double 的名称则源于其精度大约是 float 类型的两倍。floating-point 的意思为浮点数，大家可能会想，既然有了浮点，那有没有定点数（fixed-point）？还真有！接下来我来讲讲定点数是什么，了解了定点数之后，再了解浮点数会简单一些。

1. 定点数

定点数是一种用于表示数字的数值系统，它在计算机科学中得到广泛应用。在定点数系统中，实数被表示为一个整数位和一个小数位，小数点的位置在特定的位置上固定不变，不像浮点数那样可以移动。

定点数通常使用固定位数的二进制表示，其中整数位和小数位的数量是事先确定的。例如，一个 16 位的定点数可以被表示为一个包含 8 个整数位和 8 个小数位的二进制数，其中小数点固定在第 8 位处。定点数的整数位和小数位的值都用 BCD 码（Binary-Coded Decimal，二进制编码的十进制）来表示。BCD 编码非常简单，就是用 4 位二进制来表示一位十进制，其对应关系如下表所示。

表 4-2 二进制编码的十进制对应表

十进制数字	二进制数字

[33] IEEE 754 官方的名字叫 IEEE Standard for Floating-Point Arithmetic (ANSI/IEEE Std 754-2008) 。该标准于 1985 年制定，后来在 2008 年和 2019 年进行了修订。这份文档里给出了 5 种标准类型。我们常用的是两种，一种是 4 个字节表示的单精度格式 float 类型，还有一种是 8 个字节表示的双精度格式 double 类型。

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

由上面的对应表可知，每个十进制数对应 4 位二进制数字。如果要表示一位十进制数 8，则可以用 1000 来表示，如果要表示两位十进制数 10，则要分别表示 1 和 0，其中的是 1 表示为 4 位二进制 0001，0 则表示为 4 位二进制 0000。可以把两个十进制数字放一个 8 位二进制中，正好一个字节中，这种把两个十进制放入一个字节的方法被称之为压缩 BCD（packed BCD）。那么十进制数字 10 用压缩 BCD 的方式表示为 00010000。

以日常生活中去超市为例，我们会经常看到商品的标价为 19.99 元、9.99 元这样的价格。在超市场景下，计价系统的小数位事先确定为两位小数，也就是精确度保留到分即可。假设我开了一家超市，要处理的账目是±1000 万，精度精确到分，这意味着系统可以表示如下范围的数字：-10,000,000.00 元到 +10,000,000.00 元。

当我要表示一笔 3,898,190.25 元的记录，可以从最后开始，以两位十进制数字为单位进行拆分，写成如下的方式：

0011	10001001	10000001	10010000	00100101
3	89	81	90	25

定点数的优点是可以节省存储空间和计算开销，因为它们不需要浮点数那样复杂的运算和存储。但是，定点数的缺点是精度有限，因为其表示的数字范围和精度是固定的，不能随着需要进行动态调整。如果我们知道自己处理的数字范围，那么使用这种定点格式的小数是一个不错的选择，不会出现意想不到的精度问题。

在 Java 标准库中没有直接支持定点数的类型，而是提供了原生支持的浮点数类型。浮点数类型的计算速度可能较慢，同时也存在精度损失的问题，但是具有比定点数更高的精度和更大的表示范围，接下来，我们一起学习浮点数类型。

2. 单精度浮点数 float

在讲浮点数之前，先来讲一下如何用二进制表示小数。在计算机中，小数可以使用二进制表示法来存储和处理。二进制小数表示法基于科学计数法，将小数部分转换为二进制数，并使用科学计数法的形式来表示数值。

将十进制小数转化为二进制小数的步骤如下：

1. 将小数部分乘以 2，得到积和整数部分。
2. 将积的整数部分作为二进制数的一位，并将小数部分保留。
3. 将小数部分重复第 1 步和第 2 步，直到小数部分为 0 或达到所需的精度。

例如，将十进制小数 0.625 转化成二进制小数的过程如下：

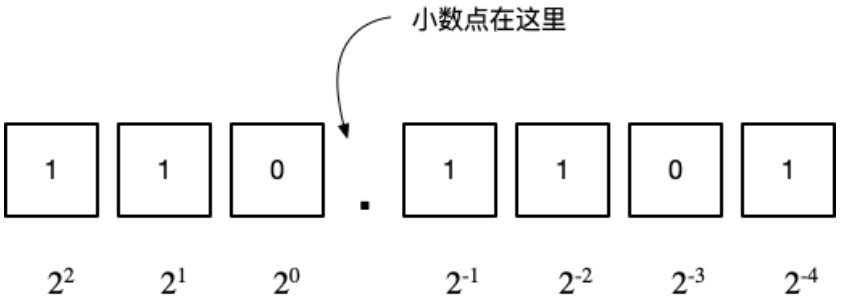
```
0.625 × 2 = 1.25，整数部分为 1，小数部分为 0.25
0.25 × 2 = 0.5，整数部分为 0，小数部分为 0.5
0.5 × 2 = 1.0，整数部分为 1，小数部分为 0
```

因此，0.625 的二进制表示为 0.101。

那如何将二进制小数转换为十进制小数呢？其方法与将整数从二进制转换为十进制的方法类似，只不过需要将每一位的权值改为 2^{-n}，其中 n 表示该位在二进制数中的位数。以下是将二进制小数转换为十进制小数的方法：

1. 将二进制小数的整数部分和小数部分分别转换为十进制数。
2. 对于小数部分，将每一位的权值改为 2^{-n}，其中 n 表示该位在二进制数中的位数。
3. 将小数部分的每一位乘以对应的权值，并将所有结果相加，得到十进制小数的值。

举个例子，以 110.1101 这个带有小数点的二进制数为例，



$(110.1101)_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 0^{-4}$

$$= 4 + 2 + 0 + 0.5 + 0.25 + 0 + 0.0625$$

$$= (6.8125)_{10}$$

当我们在表示一个很大的数的时候，比如美国国债是$25,000,000,000,000，数学上会采用科学计数法来计数，写成这样：2.5×10^{13}。在科学计数法中，我们称 2.5 为首数，10 的幂是 13，我们称 13 为指数。在计算机中也类似，只是给首数改了个名字叫有效数，指数没改名字，还是叫指数。

当我们看到 2.5×10^{13} 的时候，知道把小数点向右移动 13 位，当看到 4.5×10^{-3} 的时候，把小数点向左移动 3 位成为 0.00045。因此只要知道了指数，就知道怎么浮动小数点了，指数是正的就向右移，指数是负的就向左移。这就是浮点数的基本原理了，小数点浮动的方向和浮动多少，都由指数决定。

浮点数还有一个规定叫写法的规范化，什么叫规范化呢？还是以美国的国债为例，可以写成 2.5×10^{13}，虽然写成 25×10^{12}、250×10^{11} 都可以理解，但是后面这两种不规范。对十进制来说，规范化是规定有效数只能介于 0 和 10 之间。对二进制浮点数来说，规范化的写法第一位只能是 1。以 110.1101 为例，规范化的二进制浮点数只能是 1.101101×2^2 这一种形式，其余的形式都是不规范的。

因为规范化的二进制浮点数第一位永远只能是 1，所以根本就没必要存储这一位。因此，对实数来说，IEEE 754 规定，只需要存储三个信息即可：代表正负的符号位、代表如何浮动小数点位置的指数和有效值。以 IEEE 定义的 4 个字节的 float 单精度格式为例。

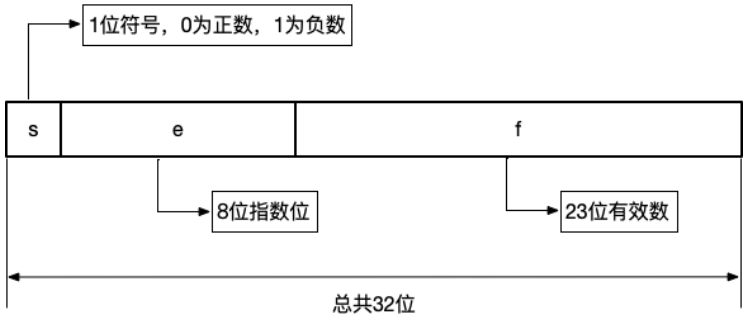


图 4-8 IEEE 754 规定的单精度浮点数 float

长度为 1 位的符号位 s 代表正数还是负数，0 表示正数，1 表示负数。

随后的 8 位是指数位 e，8 位可以代表 0 到 255 的数值。在此，我们取 1 到 254，0 和 255 留作他用。小数点有可能向左浮动，也有可能向右移动，我们规定这个数减去 127，正好可以左移或者右移，此处的 127 我们称之为偏移量。

接下来是 23 位有效数 f，前面已经讲过，有效数的第一位永远是 1，所以不用存这一位，只需要在计算的时候补上这个 1 即可。综上，这个特定的数可以用这个公式来计算：

$$(-1)^s \times 1.f \times 2^{e-127}$$

还有几种特殊情况要单独考虑，比如指数位都是 0 或者都是 1 的时候，这也是为什么指数位要保留了 0 和 255 两个数用作其它用途的原因，表示无穷大（Positive Infinity），无穷小（Negative Infinity）或者不合法 NaN（Not a Number），比如当 0/0 时或者对一个负数进行平方根操作会返回不合法。这三种情况在实际应用中并不多见，Java 为其提供了相应的常量分别是 Float.POSITIVE_INFINITY、Float.NEGATIVE_INFINITY 和 Float.NaN（相应的 double 类型也有这三个常量）。

接下来我们来看看单精度浮点的表示范围，理解了原理，套用上面的公式$(-1)^s \times 1.f \times 2^{e-127}$，我们知道最小的正负二进制是：

$$(-1)^s \times (1.00000000000000000000000)_2 \times 2^{-126}$$

指数位为$(1)_{10}$，也就是让小数点移动到最左侧，有效值是一个 1，后面跟 23 个 0。这个数用十进制表示，近似的等于$\pm 1.17549435 \times 10^{-38}$。最大表示的数是

$$(-1)^s \times (1.11111111111111111111111)_2 \times 2^{127}$$

指数位为$(254)_{10}$，让小数点移动到最右边。有效值为 24 个 1。这个数如果用十进制表示的话，近似的等于$\pm 3.40282347 \times 10^{38}$。所以，只要是用 IEEE 754 规定的浮点数，float 类型的取值范围是 $\pm 1.17549435 \times 10^{-38}$ 到 $\pm 3.40282347 \times 10^{38}$。

以上就是单精度浮点数的原理。

单精度浮点数的不足

在数学上，任意两个不同的实数，我们都可以算出平均值。但是计算机不是数学，计算机的物理特性决定了只能用 0 和 1 来表示数学，只能用离散的数据模拟连续的有理数，这种模拟会导致精度丢失。

前面讲的 float 单精度用 24 位来表示有效值，只能表示大约相当于 7 位十进制的数字，单精度浮点数的精度是 $1/2^{24}$，也就是 $1/16777216 = 5.96046 \times 10^{-8}$，相当于百万分之六左右。那这个精度意味着什么呢？看看下面这个程序：

```
public class FloatDemo {
    public static void main(String[] args) {
        float a = 66999888.0f;
        float b = 66999887.0f;
        System.out.println(a - b);
    }
}
```

在这段代码中，变量 a 和 b 都是 float 类型，在输出 a-b 的结果时，实际上输出的是 0.0，而不是期望的 1.0，这是因为精度丢失的问题，百万分之六左右的误差。

如果想精度更高一点，只能考虑把单精度浮点数换成双精度浮点数。了解了单精度浮点数，双精度浮点数的原理是一样的，只是 4 个字节的长度换成了 8 个字节的长度。

3. 双精度浮点数 double

双精度浮点数的原理和单精度的一样，在此不再重复。下图是 IEEE 中规定的双精度浮点数格式。

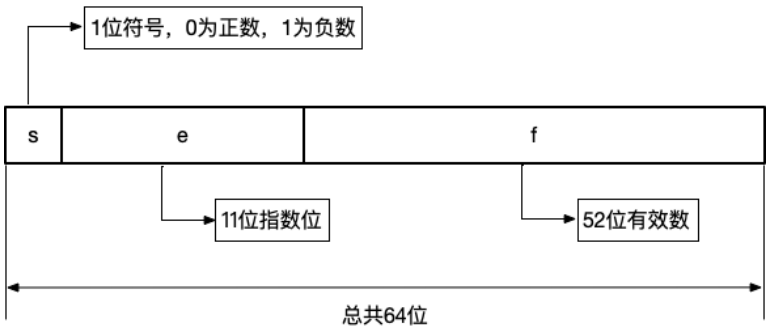


图 4-9 IEEE 中规定的双精度浮点数 double

因为指数有 11 位了，所以偏移量是 1023，所以双精度浮点数存储的数可以表示为：

$$(-1)^s \times 1.f \times 2^{e-1023}$$

所能表示的最大最小规则同样适用，计算以后可以知道，双精度所能表示的范围用十进制来表示大概是：$\pm 2.2250738585072014 \times 10^{-308}$ 到 $\pm 1.7976931348623158 \times 10^{308}$ 。这是一个非常巨大的数字，2 后面跟着 308 个 0。

如果把上面那个展示 float 精度的代码中的 float 换成 double，看看会发生什么吧。

```
public class DoubleDemo {
    public static void main(String[] args) {
        double a = 66999888.0;
        double b = 66999887.0;
        System.out.println(a - b);
    }
}
```

上面的代码运行以后就不像 float 那样输出 0.0，而是输出 1.0。在科学研究和工程运算

上，浮点数的精度已经够了。

金融、证券等这些对精度要求特别高的行业，他们需要完全的精确，不能有任何的精度损失。对这种要求，Java 提供了 BigDecimal 类，BigDecimal 类型使用字符串表示数值，因此可以避免二进制浮点数带来的精度损失。当然，使用 BigDecimal 类型进行计算也会带来一些额外的开销，因为它需要更多的内存和处理时间来完成计算。

4.3.5 类型转换

以上讲的 8 种原始数据类型中，除了布尔类型，其余的 7 种之间有可能进行数据类型转换。在现实生活中，我们实际上是不太区分数字的类型，比如我们早上去菜市场，卖菜的说 2 元钱一斤，买菜的大妈还价 1.8 行不行？实际上这个大妈很富，家里拆迁分了 16 套房，银行里的存款高达 1.75 亿人民币。当我们在说 2 元，1.8 元，16 套，1.75 亿这些数字的时候，脑海中并没有出现什么 int 类型，double 类型。如果在写程序的时候，编程语言也能自动处理这些类型的转换就好了。实际上，编程语言确实这么做了，编程语言会根据情况做自动的类型转换。

Java 中的数据类型转换是将一个数据类型的值转换为另一个数据类型的过程。在程序中，可能需要将一个数据类型的值转换为另一个数据类型的值，以便进行算术运算、比较或赋值等操作。类型转换可以分为两种情况：自动类型转换和强制类型转换。下面分别介绍一下这两种情况。

1. 自动类型转换

Java 中的自动类型转换是指在不需要显式转换的情况下，将一个数据类型转换为另一个数据类型。通常，自动类型转换发生在两种数据类型存在类型层次结构（Hierarchy）时，即将一个小的数据类型转换为一个大的数据类型。有效值范围小的数据类型可以自动转换为有效值范围大的数据类型，这种类型的转换不会丢失精度。如下图所示，实线箭头所表示的就是不会丢失精度的转换。

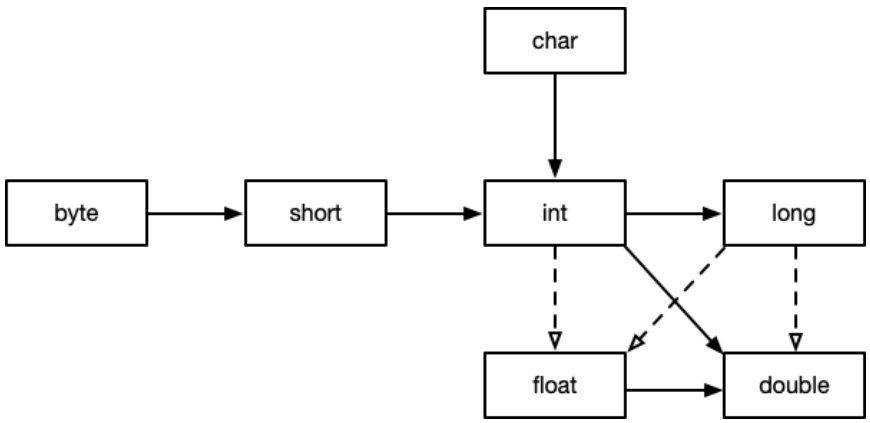


图 4-10 Java 原始类型之间的转换

自动类型转换依次遵从如下的规则：

- 如果有一个操作数为 double 类型，那么先要将另一个操作数转换为 double 类型
- 如果有一个操作数为 float 类型，那么先要将另一个操作数转换为 float 类型
- 如果有一个操作数为 long 类型，那么将另一个操作数转换为 long 类型
- 如果以上都不满足，那两个操作数都转换为 int 类型

了解了以上四条规则，我们通过几个例子来验证一下这些规则。

```
1  // 对两个整数求平均值 chapter4/average_demo1/AverageDemo1.java
2
3  class AverageDemo1 {
4      public static void main(String[] args) {
5          int a = 5;
6          int b = 6;
7
8          double average = (a + b) / 2;
9          System.out.println("a 和 b 的平均值为：" + average);
10     }
11 }
```

上面代码的第 8 行是计算整数 5 和 6 的平均值，如果是没学过编程的小学生来算的话，肯定能得出平均值为 5.5。但是在 Java 中，得到的是一个与常识相反的结果 5.0。这是为什么呢？在第 8 行里，a 和 b 都是整数，适用于两个操作数都是 int，结果也是 int，因此(a + b)/2 中，int + int 结果是 int，int/int 的结果也是 int，会舍弃小数部分得到 5。

由于我们定义 average 是 double 类型，整数 5 存入其中没有小数，因此最终结果是 5.0。那我们应该如何等到想要的结果 5.5 呢？一个比较简单的方式是使用上面的规则，只要有一个操作数为浮点数即可，因此，我把第 8 行的代码修改为如下的样子：

double average = (a + b) / 2.0;

这样运行以后，就可以得到我们想要的结果 5.5 了。当执行（5+6）/2.0 的时候，由于 2.0 是 double 类型，那么 5+6 得到的结果 11 会时行类型转换成 double 类型，然后再计算。这种方式有个专有名字叫双目数值提升（binary numerical promotion），其内涵就是把数据类型进行自动提升。

但是在实际的计算中，如果我们算两个数的平均值是除以 2.0，这样显得非常的不合理。我们可以考虑把（5+6）强制转化为浮点数。

2. 强制类型转换

顾名思义，强制类型转换的意思就是不管三七二十一，直接将一个数据类型强制转换为另一个数据类型。强制类型转换通常发生在两种数据类型不兼容的情况下，需要将一个大的数据类型转换为一个小的数据类型。

把上面那个算平均值的强制类型转换一下，代码如下：

```
1  // 对两个整数求平均值
2
3  class AverageDemo1 {
4      public static void main(String[] args) {
5          int a = 5;
6          int b = 6;
7
8          double average = (double)(a + b) / 2;
9          System.out.println("a 和 b 的平均值为: " + average);
10     }
11 }
```

看上面代码的第 8 行，对一个值或者表达式进行强制类型转化的格式如下：

(类型) 表达式

用这种方法可以把表达式强制转化为想要的类型，像本例所示，就是把后面的表达式（a+b）强制转化为 double 类型。在英语中这种方法被称之为 cast，翻译成中文有的翻译为强制转换，有的翻译为造型。还有一点需要注意，第 8 行中有两个括号，后面的(a+b)这个括号是用来指定优先级的符号，前面(double)的括号叫 cast operator，翻译成中文为强制转换操作符或者造型操作符。

强制类型转换又分成两种，一种是精度低的向精度高的转换，这种情况可以称之为基本数据类型扩大的强制转换，这种情况不会出现问题，可以显式的写上强制类型转换，也可以不写，如果不写，编译器会采用自动类型转换的方式提升精度。还有一种是精度高的向精度低的转换，也可以称之为类型缩小的强制转换，这种情况要小心，会丢失精度。下面分别来介绍一下。

先来说简单的，精度低的向精度高的转换。看下面的代码：

```
1       int a = 'a';
2       long b = a;
3       double c = 2.7182f;
```

第 1 行的‘a’是字符类型，赋值给 int 类型的变量属于基本数据类型的扩大，完全没问题。如果再把该变量赋值给 long 类型，也没问题。第 3 行的 2.7182f 是 float 类型，赋值给 double 类型的 c 也没问题。

由以上的几个例子可以看出，当表示范围小，精度底的类型向表示范围大，精度高的类型转化的过程中，“一般”是不会有问题的。下面这张图是大体“没问题”的方向。

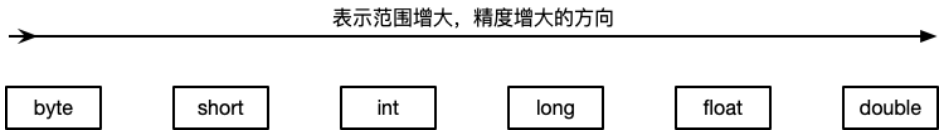


图 4-11 基本数据类型转换方向

之所以说“大体”没问题，是因为还是有一些问题，比如说 int 或者 long 类型的值到 float 类型转换，或者 long 类型的值到 double 类型转换，仍然会有精度方面的变化。原理在浮点数那一节已经说过了，浮点是近似值，转换的结果只能是最近似的那个值。下面这个例子：

```
// 从小转大 chapter4/type_to_type1/TypeToType1.java

public class TypeToType1 {
    public static void main(String[] args) {
        int a = 666666666;
        long b = 33333333333333333L;

        System.out.println( "(float) a = " + (float)a );
        System.out.println( "(double)b = " + (double)b );
    }
}
```

以上这段代码编译运行以后结果如下：

```
(float) a = 6.6666669E8
(double)b = 3.3333333333333332E16
```

可以看到，在转换的过程中，精确度确实有所丢失。在一般的计算之中，并不要求浮点精确，如果这段程序要用在火箭发射上，可能需要特别的注意一下，毕竟 Ariane 5 火箭就是浮点转换的时候精度不够而炸掉的。

如果我们的软件严格要求浮点精确，Java 也可以做到。Java 有一个关键字叫 strictfp（strict float point），使用 strictfp 关键字可以来声明一个类、接口或者方法，那么在所声明的范围内，Java 编译器以及运行环境会完全依照 IEEE 二进制浮点数算术标准来执行，声明的范围内所

有浮点数的计算都是精确的。在实际应用中，我们一般用不到这个参数，在此就不再赘述。

还有一种情况是表示范围大，精度高的向表示范围小，精度低的转换。这种转化必须显示的写明强制类型转换，转换以后的数值，不管是“精度”还是“长度”，都非常有可能丢失。看下面的例子：

```
1       int a = 3.14;
2       int b = (int)3.14;
```

上面代码的第 1 行，我把一个 double 类型的数字 3.14 赋值给 int 类型的变量，在编译时候会报错，并且提示从 double 类型转换到 int 类型可能会有损失。

```
TypeToType.java:6: 错误：不兼容的类型：从 double 转换到 int 可能会有损失
       int a = 3.14;
```

这说明表示范围大、精度高的类型向表示范围小、精度低的类型转换时，不会自动转换。如果要强行转换，必须要使用第 2 行的格式，强行转换以后，变量 b 的值将会成为 3。

其它语言中是如何处理类型转换的？

因为 Java 的语法借鉴了 C 语言，C 语言又被广泛的应用，因此在处理类型转换的时候，Java 参考了 C 语言的规则。实际上，并不是所有语言都采用这种规则。比如前面举的例子 x/y 中，如果 x 与 y 都是整数，那结果也是整数。

这种规则有些不合理，所以在 python 3.0 中，当 x/y 时，如果结果是浮点数，就算 x 与 y 都是整数，也会输出浮点数。但是在 python 3.0 之前的版本，又与 C 语言的风格相同。如果在 python 3.0 中兼容 C 语言的做法，又引入了一种新的语法

4.4 引用数据类型的变量

上一节讲过 Java 中的数据类型分为两大类：基本数据类型和引用数据类型。并且在上一节中，对基本数据进行了详细的讲解，我们已经了解到基本数据类型直接存储在变量中，它们具有固定的大小和范围。当我们声明一个基本数据类型的变量时，内存中会为该变量分配一定的空间，用于存放实际的值。

在这一小节中，我们将目光转向另一种数据类型：引用数据类型。引用数据类型主要包括以下几种：类、数组（本书第 7 章内容）和接口（本书第 13 章内容）。从两个方面来介绍引用数据类型的变量：一是变量里面存储的什么内容，二是变量的大小。

4.4.1 引用数据类型的变量里面存储着什么内容

引用数据类型的变量存储的是对象的引用（内存地址），而非对象本身。当我们声明一个引用数据类型的变量时，实际上分配了一个指针。这个指针指向对象的内存地址，而对象本身存储在堆内存（heap memory）中。

指针是一种变量类型，它存储了一个内存地址。这个内存地址指向某个存储在内存中的值。使用指针可以间接地访问和修改这个值。指针在很多编程语言（如 C 和 C++）中广泛使用，它们提供了对内存的底层操作能力。

Java 中没有明确的指针概念，但引用数据类型的变量在某种程度上类似于指针。当我们创建一个引用数据类型的变量时，变量存储的是对象的内存地址（引用），我们可以通过这个引用间接地访问和修改对象的内容。

下面是一个引用数据类型的例子：

String name = "Java";

在这个例子中，name 是一个引用变量，它指向一个 String 类型的对象。对象的内容（"Java"）存储在堆内存中，而 name 变量存储的是对象的引用。下面的例子中有两行代码，第一行是定义了一个基本数据类型的变量，第二行定义了一个引用数据类型的变量：

```
int i = 10;
String name = "Java";
```

这两个变量在内存中的情况如下图所示：

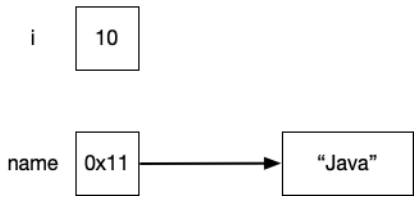


图 4-12 基本数据类型变量与引用数据类型变量内存中的区别

上图所示的 name 中所存储的是一个内存地址，我随便写了一个假的地址，因为在 Java 中，程序员不能直接获得变量在内存中的地址。这是因为 Java 的设计目标之一是提供一种安全、简单且跨平台的编程语言。为了实现这个目标，Java 引入了一个抽象层，使开发人员无法直接访问底层内存地址。

接下来，我们再来看看引用数据类型的变量在内存中占用的空间有多大。

4.4.2 引用数据类型的变量占用的内存大小是多少

像前面讲的基本数据类型的变量，有的占两字节，有的占四个字节。在 Java 中，引用数据类型占用的内存大小由对象头和对象实例两部分组成，因此引用数据类型的变量占用内

存的大小等于对象头和对象实例两部分内存大小之和。

对象头包含了用于垃圾回收和线程同步的信息，其大小根据虚拟机的不同有所区别。通过搜索 Compressed oops in the Hotspot JVM 这份文档，得知在 Hotspot 虚拟机中，如果是 32 位的 Java 虚拟机，占用 32 位，也就是 4 个字节。如果是 64 位的 Java 虚拟机，有可能是 32 位的，也有可能是 64 位。这取决于是否开启了-XX:-UseCompressedOops 参数，如果开启了，就是 32 位，如果没开启，就是 64 位。

对象实例则包含了对象的成员变量和实例方法，其大小不固定。因此，对引用数据类型的变量而言，如果所引用的对象是一个空对象，则占用的内存大小为对象头的大小。如果所引用的对象不是空对象，则占用的内存大小为对象头的大小加上对象实例的大小。

由于不同类型的 JVM 的实现有所不同，因此通常很难准确地知道一个引用数据类型的变量占用的内存大小。不同的 JVM 实现可能有不同的对象头大小和对齐方式，而对象的实际大小还受到对象的成员变量、继承关系、对齐方式等因素的影响。因此，即使是相同类型的对象，其占用的内存大小在不同的 JVM 中也可能会有所不同。

虽然无法准确地知道一个引用数据类型的变量占用的内存大小，但可以通过一些手段来估算它的大小，例如使用 Java 的内存分析工具（如 VisualVM、JProfiler 等）来观察内存使用情况，或者使用 Java 的代理机制来动态获取对象的大小信息。但需要注意的是，这些方法并不是绝对准确的，因为它们都受到 JVM 实现、垃圾回收机制等因素的影响。

4.4.3 小结

最后，比较一下基本数据类型的变量和引用类型的变量有什么区别与联系吧。

表 4-3 基本数据类型变量和引用类型变量的区别

	基本数据类型（primitive Types）	引用数据类型 （Reference Types）
种类	8 种：boolean、char、byte、short、int、long、float 和 double 类型。	用户可以自定义类，所以，可以有无穷种。
变量的值	对基本数据类型来说，变量中存的值就是变量实际的值。	变量的值是指向对象的引用。
赋值给另一个变量	当一个基本数据类型的变量赋值给另一个变量的时候，会在内存里复制一个新数据。这种情况有个学名叫值传递（pass by value）。	当引用类型的变量赋值给另一个变量的时候，新的变量会指向原来的变量，不会重新复制一个新的。这种情况有个学名叫引用传递（pass by reference）。

方法调用	当在方法中处理基本数据类型的时候，和赋值一样，会复制一份新数据，因此，操作结束以后，不会改变原数据。	当方法处理的是引用类型，引用类型的值会发生改变。

4.5 变量作用域

在 Java 中，变量作用域指的是变量在程序中可以被访问的范围，如果在一个作用域内定义了一个同名的变量，它会覆盖外层作用域内同名的变量，这种情况称为变量的遮蔽。无论对人名还是变量来说，朗朗上口的名字比较稀缺，非常容易重名。

重名问题普遍存在

《2019 年全国姓名报告》报告显示，全国户籍人口中，使用频率最高姓名“张伟”有近 30 万人，全国最多。但是这 30 万“张伟”并没有对我们造成过多困扰。对我们每个人来说，“张伟”是有作用域的。

更普遍的例子，每个人都有“爸爸”，但是当喊“爸爸”的时候，并没有造成什么困扰，也是因为“爸爸”有作用域，仅限于家庭中。每个人都知道别人喊“爸爸”时，应不应该答应。

假设我们有幸站在编程语言刚起步的几年，会发现早期的语言经常会发生名字冲突，因为早期的编程语言中，程序可以访问所有的变量名，名字总难免冲突，就像同一个班里有两个“张伟”，点名的时候，不知道叫的谁。

如果在同一个班里，解决方法是让其中一个人改名。虽然这种“简单粗暴”的方法也可以用于编程中，但是要想在巨量的代码中确保变量不重名，几乎是不可能的，只能想另外的办法。计算机科学家想到的办法是让名字有“作用域”，指定名字的有效范围，保证程序不出现名字冲突。目前有两种解决方案：一种叫动态作用域，一种叫静态作用域。Java 采用的是静态作用域。

Java 变量的作用域由变量声明的位置决定。变量会出现在类中，出现在方法中（本书第 8 章会讲方法），出现在代码块中，出现在循环体中（本书第 6 章会讲循环）。根据变量出现的位置不同，有以下几种作用域:

- 类作用域

在类内部定义的变量，可以在整个类内部访问，包括类的方法、构造函数和代码块。

- 方法作用域

在方法内部定义的变量，只能在该方法内部访问，方法执行结束后变量会被销毁。

- 代码块作用域

在代码块内部定义的变量，只能在该代码块内部访问。Java 的代码块是一段被大括号{}包围的代码片段，可以包含任意数量的语句。

- 循环体作用域

在 for 循环体内定义的变量，在循环体内部访问，循环结束后变量会被销毁。

代码清单 4-6 不同位置的变量作用域不同

```
public class ScopeExample {
    int a = 1; // 类作用域变量

    public void method() {
        int b = 2; // 方法作用域变量
        if (b > 1) {
            int c = 3; // 代码块作用域变量
            System.out.println("c=" + c);
        }
        // System.out.println("c=" + c); // c 只能在 if 代码块内部访问，这里会报错

        for (int i = 0; i < 3; i++) { // 循环体作用域变量
            System.out.println("i=" + i);
        }
        // System.out.println("i=" + i); // i 只能在 for 循环体内部访问，这里会报错
    }

    public static void main(String[] args) {
        ScopeExample example = new ScopeExample();
        System.out.println("a=" + example.a);
        example.method();
    }
}
```

现在绝大部分语言采用的都是静态作用域，由于很多语言都采用了这项技术，所以我们都觉得理所当然，但作用域的演化远比现在要复杂，其中最典型的例子就是 Perl 语言，我当过几年 Perl 程序员，对 Perl 语言相对比较了解，Perl 语言是我所知的把各种作用域的坑都踩了一遍的语言。

这是一本讲 Java 的书，但是我觉得了解其它语言对编程爱好者是件很有趣的事情。我

就来以 Perl 为背景来编个故事吧，因为是故事，只能隐去真实的名字了。这个故事是我模仿 Perl 语言的作者 Larry Wall 创建 Perl 来写的。Larry Wall 毕业于加利福尼亚大学洛杉矶分校，在 Unisys 公司，希捷公司工作过。他是一个系统管理员，同时也是系统程序员，曾经给 NSA（国家安全局）做不可以泄露机密的工作。Perl 语言最初是为了从大量文本中筛选出重要的信息。通过这个故事，希望你可以理解动态作用域与静态作用域的区别，以及为什么 Python，Javascript 这些选择动态作用域的语言需要更多的关键字来处理作用域。

故事：动态作用域和静态作用域

我是个程序员，我是计算机的先驱，站在 1986 年人文和科技的十字路口上。

今天是 1986 年 6 月 22 日，马拉多纳带领阿根廷打了英格兰两个球，他自己就进了两个球。马拉多纳疯狂的在球场上挑衅，而英国人则绅士一样的站在球场里，任由潘帕斯草原上的雄鹰翱翔。哦，忘记说一件事了，英国是个绅士国家，挑衅英国人只有一个结果，英国人会上诉到法院。

我不是阿根廷球迷，甚至，我不是足球球迷，但是今天我看到英格兰惨败，心里太高兴了，因为我的上司是个英国绅士，我的合作伙伴是英国公司，我已经快被工作压垮了，而这个英国公司却威胁我如果不能按期完成工作，那就法院见。

对了，我还没有说我的工作是什么呢。我的工作叫数据处理程序员，就是坐在计算机前面处理数据的那种人，现在是 1986 年，干我们这一行的还没多少人，以后也许会多吧，谁知道呢。我要给伦敦证券交易所的那帮混蛋处理数据，他们的数据太多，变化更多，不是每天都要数据，而是每秒都要数据，我真的快要被逼疯了。

我有一台最先进的计算机，但是只有几个蹩脚的工具，有两个处理文本数据的软件叫 sed 和 awk，[34]还有一个不好用的编程语言叫 C 语言，还有个 shell，没有其它的了，这些工具像石器时代过来的，面对伦敦证券交易所每天产生的巨量数据，这些工具早晚会顶不住。

上帝保佑，如果我能有个更趁手的编程语言就好了，这个编程语言能够不像 C 语言一样每次使用前先编译，而是可以直接执行，这个语言还能和 sed 与 awk 完美的结合。

我知道现在没有，求人不如求已，只要有空闲，我自己做一个这样的编程语言吧。

现在 Apple 公司很火，我也给我的编程语言起一个名字吧，就叫梨子语言，Pear 语言

[34] AWK 是一种处理文本文件的语言，是一个强大的文本分析工具。特点是处理灵活，功能强大。可实现统计、制表以及其他功能。之所以叫 AWK 是因为其取了三位创始人 Alfred Aho，Peter Weinberger, 和 Brian Kernighan 的 Family Name 的首字符。sed：Stream EDitor，流编辑器，它以行为单位处理字符流。

吧。这个语言是我自己用的，所以设计的没那么精密，只要能完成工作就好。

我有个优势，我有设计语言的条件，我妻子是个语言学家，她设计了一套语言，可以翻译多种版本的圣经，比我这个 Pear 语言可复杂多了。花了没多长时间，大概 3 个月吧，我这个语言就能运行了，至少再也不用写 C 语言了，伦敦证券交易所的那些数据每次都会有变动，再微小的修改，都要编译一次 C 语言，现在不用每次都编译了，我用的还是很开心的。

我把我这个语言告诉了我的同事，他们提了一些意见，总归不用 C 语言了，还有一些同事提出要把 sed 和 awk 也整合进来，我觉得挺好的。

又过了几个月，用我编程语言的人越来越多了，项目也快一年了，我打算把这个语言发到网上，如果有人喜欢，自己可以去编译一下，就发到 unix 的新闻组好了，就叫 Pear kit 1.0 吧。

我获得了很多的赞扬，但是也有人提出了一些 bug，说这个语言不能很好的处理变量和作用域的问题。这个家伙发给我了一封邮件，我贴在这里，因为我的团队人不多，我还没遇到这个情况。

亲爱的 Pear 创作人，你好：

我非常喜欢你写的这个编程语言，每天都用得特别开心。我在工作中碰到了这样一个问题，因为我们是一个大型网站，有很多员工，Pear 语言在设计的时候对变量使用的是对照表，我理解你这样设计的优点是很容易实现，对个人来说已经够快了，但是我的同事太多了，经常会给某个变量起相同的名字，所以，每次都要处理名字相同的问题。

请问，有没有什么好办法或者好的机制来处理么？

邮件就是这样，我读了好几遍。我没想到竟然有商业公司用 Pear 语言写程序，如果人一多，确实会出现他说的那个问题。我最初设计的时候确实没考虑这么多，为了说明这个情况，我来解释一下，以便大家知道这到底是个什么问题，看看下面这段代码：

代码清单 4-7 变量作用域设计失误案例

```
for( i = 0; i < 10; i++) {
    process();
    print i;
}


process() {
    i = 0;
}
```

由于我设计的不严谨，前面的循环里用到了一个变量 i，在循环里还调用了一个方法 process，结果这个 process 方法里也用到了变量 i，这样就把 i 的值改变了，导致上面的循环成了死循环。

我设计这个语言的时候，根本没有考虑这么多，我用了最简单的方法，用对照表，对照表的方法类似于身份证，出现一个变量就去一个列表中去查找，并且不能出现相同名字的变量，这个方法最简单粗暴有效了。

我暂时没想到好的解决方法，只好建议他尽量不要取相同的名字，或者如果有很多开发者的话，可以考虑每个开发者加上自己名字的前缀，比如 lyd_i 这样。

或者用下面这种方法，在方法入口处先保存原来的值，等到处理完以后，在方法出口的时候再把保存的值返回给变量。

```
process() {
  old_i = i;
  i = 0;
  i = old_i;
}
```

这是个权益之计，我知道我要想办法解决这个问题了。这样写太麻烦了，人为的增加了难度，也会增加出错的几率。可以把这种写法加上一个关键字 local，有了这个关键字，可以让编程语言自己在程序入口处把值存起来，在方法出口处把值再复原回去。这样稍微减轻了一点工作量，上面这段代码可以这样写:

```
process() {
  local i;
  i = 0;
}
```

直到又有一个人提出这个方法仍然有 bug，这是我没想到的。我本以为是个初学者提出的又一个无聊的问题，在我仔细看了邮件之后，才发现动态作用域的方法真有严重的缺陷。

看看下面的代码：

```
x = "武大郎"; // 全局变量


panjinlian_husband() {
    print x; // 输出武大郎
}
```

```
wangposhuomei() {
    local x = "西门庆";
    panjinlian_husband(); // 竟然输出了西门庆
}
```

这段代码中，有一个变量 x 是全局变量，里面的值是“武大郎”，有两个方法，一个方法是输出潘金莲的丈夫 panjinlian_husband()，还有一个方法是王婆说媒 wangposhuomei()。按照常理，panjinlian_husband()这个方法没有参数传递，在任何情况下都应该是输出确定的值。但是在 wangposhuomei()这个方法中，却会输出错误的结果。

方法调用方竟然修改了方法运行的结果，这种隐含的错误是动态作用域无法克服的困难。有没有更好的解决方法呢？我冥思苦想以后，决定还是借用 C 语言中的静态作用域的方法吧。好了，我的故事讲完了，设计一门编程语言还真是不容易呢。

之所以会出现故事里的那种 bug，主要原因是动态作用域中变量的对照表会被所有的源代码读写，如果对每个方法都建立自己的对照表，只能由本方法来读写，那就解决问题了。

那是不是可以说动态作用域不如静态作用域好呢？不能！像 Java 这样的面向对象编程语言，可将属性设置为 private，private 可以将属性限制在类之内，这又有点动态作用域的意思了。还有 Java 的异常处理机制也与动态作用域有些类似。这些放在以后的章节再说，学编程，多考虑一下，不要一根筋的认为某个语言比另一个语言好，也不要认为一个技术比另一个技术要高明。

他山之石

程序员不可能只用一种编程语言，老板让用啥，维护的项目用啥，就得学啥，所以，比较的学习一下会比较好。

JavaScript 语言中，没有任何声明的变量默认为全局作用域，使用 var 声明的变量为静态作用域变量。

Python 语言的情况更复杂一些，在 Python 2.0 中没有解决的问题到了 Python 3.0 中试图解决，比如引入了 nonlocal 这个关键字。当年为引入哪个关键字也有巨大的争论，后来考虑再三选择了 nonlocal 这个关键字，有兴趣的读者可以在 python 官网搜 PEP 3104 这个关键字，看看当年的情况。

4.6 脱口秀：值传递与引用传递

以下故事纯属虚构。小潘是我新来的同事，我每写完一章书，都会先拿给几个同事看。于是就有了这段对话。

小潘：栋哥，你的书里为什么没写值传递和引用传递的内容呢？

栋哥：还没到时候，我想在第 8 章方法那一章中写。

小潘：我一直没搞明白值传递和引用传递到底发生了什么，你应该在书里详细写一下。

栋哥：那你认为这两者有什么区别呢？

小潘：我是这样理解的，如果我有一个变量，我将这个变量传递给一个方法来处理，如果该方法有办法把这个变量进行修改，那就是引用传递，如果该方法没有办法修改我这个变量，那就是值传递。

栋哥：你这样理解的不错，那你觉得 Java 是值传递还是引用传递呢？

小潘：一般书上都说，对基本数据类型的变量来说是值传递，对引用数据类型的变量来说是引用传递。是这样么？

栋哥：并不是，不管书上怎么说，对 Java 来说，实际上都是值传递。

小潘：不会吧，你又想标新立异。

栋哥：你仔细思考这样一个问题，在 Java 中，一个字符串 String 类型是基本数据类型还是引用数据类型？

小潘：那还用说么，你这一章不是写了么，基本数据类型只有那 8 种，String 类型肯定不在其中，按书上的定义，传递一个引用数据类型，肯定是引用传递。

栋哥：是的，那你来看看这段代码应该输出什么结果？

```
public class PassByReference {
    public static void main(String[] args) {
        String myString = "Hello, World!";
        System.out.println("Before: " + myString);
        modifyString(myString);
        System.out.println("After: " + myString);
    }

    static String modifyString(String input) {
        input = input + " Again!";
        return input;
    }
}
```

小潘：第一行输出 Before: Hello World! ，第二行输出 After: Hello World! Again !

栋哥：是么，你解释一下为什么？

小潘：第一行就不用解释了，第二行是因为 myString 通过引用传递的方式传递给 modifyString 方法，由于是引用传递，modifyString 修改了传递进来的值，从而使 myString 的值变为 Hello World! Again! ，因此输出第二行。

栋哥：错的有点离谱了，你一会去运行一下，你会发现这两行代码输出的结果是一样的，都是 Hello，World！

小潘：为什么？

栋哥：因为 Java 都是值传递啊，只是对引用类型的变量来说，它的值恰好是引用。于是，大家都称之为引用传递了，但是 Java 实质上并不支持纯粹的引用传递。在前面的例子中，String 对象是不可变的，这意味着一旦创建了一个 String 对象，它的内容就不能改变。因此，尽管我们可以尝试编写一个传递 String 类型引用的示例，但在方法内部我们无法直接修改原始字符串的内容。在这个示例中，我们创建了一个名为 modifyString 的方法，该方法接收一个 String 类型的参数 input。在方法的内部，我试图用再对 input 进行一些修改，但是 Java 只是重新在堆中建立了一个新的字符串”Hello World! Again!”。

小潘：我好像有点懂了，如果不是 String 类型的变量，我认为结果应该会不同。

栋哥：是的，比如下面这个例子，跟上面的例子相似，但是结果却截然不同。

```
public class ArrayExample {
    public static void main(String[] args) {
        int[] myArray = {1, 2, 3, 4, 5};
        System.out.println("数组在传递给方法之前的内容：");
        printArray(myArray);

        modifyArray(myArray);

        System.out.println("数组在传递给方法之后的内容：");
        printArray(myArray);
    }

    public static void modifyArray(int[] arrayToModify) {
        // 修改数组的第一个元素
        arrayToModify[0] = 99;
    }

    public static void printArray(int[] arrayToPrint) {
        for (int i : arrayToPrint) {
            System.out.print(i + " ");
```

```
        }
        System.out.println();
    }
}
```

小潘：这个例子中，数组发生改变了么？

栋哥：是的，在这个例子中，我们首先创建了一个包含 5 个整数的数组 myArray，然后将其传递给 modifyArray 方法。在 modifyArray 方法中，我们修改了数组的第一个元素。当我们再次打印 myArray 时，可以看到其内容已经被修改。

小潘：那你给我详细解释一下原因好么？

栋哥：不要着急，第 7 章我会讲数组，第 8 章会讲方法。我还要请你帮我审稿呢，到时候，我会详细解释给你的。

小潘：好吧，很有你做事的风格。

4.7 常见问题

4.7.1 Java 的字符类型与 Unicode 之间有什么关联？

Java 的字符类型和 Unicode 之间有密切的关系。Unicode 是一种字符集和编码的标准，旨在为全世界的所有文字和符号分配唯一的编码方案，Unicode 可以表示许多不同的语言和符号，包括英文、中文、日文、韩文、数学符号等。Unicode 的目标是为每个字符或符号提供一个唯一的数字，称为 Unicode 代码点（Code Point）。

Java 中的字符类型是一个 16 位无符号整数，用于表示 Unicode 字符。字符类型可以表示 Unicode 基本多文本平面（Basic Multilingual Plane，BMP）中的字符，即 U+0000 到 U+FFFF 的代码点，基本多文本平面中包括了大多数常用字符。

针对 Unicode 辅助平面（Supplementary Planes）中的字符（U+10000 到 U+10FFFF），由于字符类型的取值范围不足以表示辅助平面中的字符，因此 Java 使用两个字符类型的值（称为代理对，Surrogate Pair）来表示这些字符。

4.7.2 在计算机中，如何处理无法用有限的二进制小数表示的十进制小数？

在计算机中，有些十进制小数无法用有限的二进制小数表示，例如 0.1。这是因为对于某些十进制小数，它们的二进制表示在小数部分会出现无限循环的情况，例如 1/3 在十进制下为 0.33333……，在二进制下为 0.01010101……

为了避免精度损失，可以使用更高精度的数值类型，如 BigDecimal 类型来处理小数。

BigDecimal 类型可以表示任意精度的十进制小数，不会出现精度损失的问题。

4.8 总结

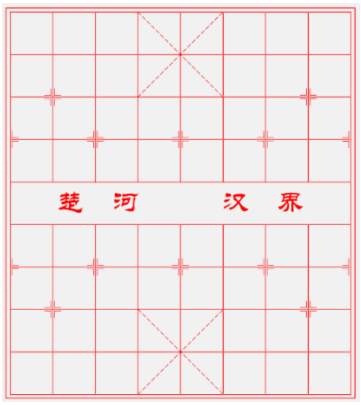
这一章的内容是围绕着变量展开的，本章的主要内容有：

- 为什么会有变量，变量的历史
- 什么样的变量名是好名字，如何给变量起个好名字
- Java 中变量的类型以及如何把变量存于内存中什么地方
- 介绍了基本数据类型和引用类型这两种类型的 Java 变量
- 详细介绍了计算机中整数类型的表示方法
- 详细介绍了计算机中浮点类型的表示方法与 IEEE 标准
- 介绍了不同类型变量之间的转换
- 介绍了变量的作用域，分析了静态作用域与动态作用域之间的区别
- 变量的生命周期与内存回收的问题

4.9 思考拓展

1. 中国象棋有多少个棋子？如果我们写象棋软件的时候，棋子的数目，每一行有多少个格子，是设置为普通的变量好，还是设置为 final 类型的变量好？你能给出你选择的理由么？
2. 如下图所示，中国象棋的棋盘有 90 个点格。如果要在程序中表示这 90 个点格，会有很多方案，最简单的有从 0 到 89 这 90 个整数来表示。[35]除了这种表示棋盘的方法，你能想出几种其它的方法么?

[35] 在早期的国际象棋程序中，确实采用过这种方法，虽然该方法简单易懂，但是有很多缺陷。在 1984 年，David Welsh 曾经写过一本名为《Computer Chess》的书，在该书中详细介绍了早期计算机中如何在计算机上写象棋程序。这本书有点老了，但是仍不失为一本优秀的作品，书里讲了当年计算机编程的困境，如何有效的利用每一个字节。


楚 河
汉 界