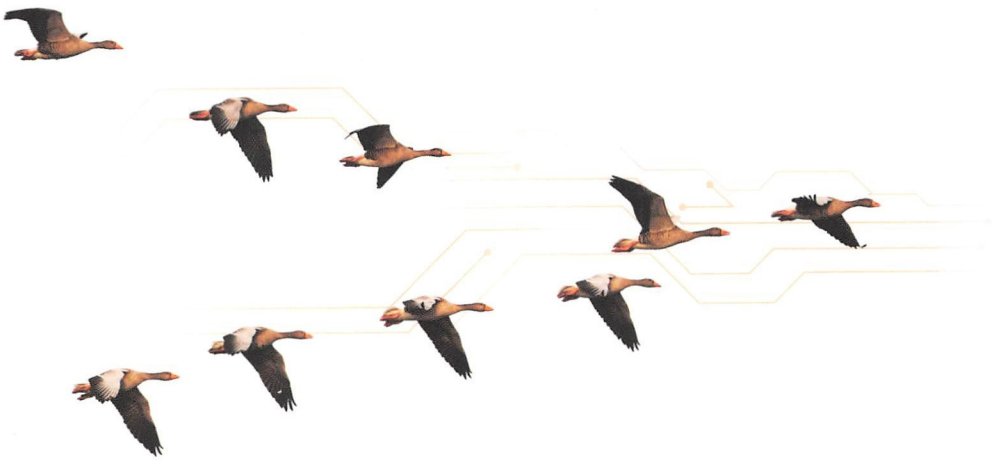
Broadview®
www.broadview.com.cn



码出高效

Java开发手册

杨冠宝（孤尽） 高海慧（鸣莎） 著

Easy Coding

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

杨冠宝

阿里巴巴集团高级技术专家，花名孤尽，取自风清扬“独孤九剑，破尽天下武功”之意。在阿里历任技术研发、架构师、部门主管等不同的角色，承担过双十一、国际化、代码中心等大型项目，有着丰富的一线编程实战和架构经验。目前是集团代码平台负责人，在大数据、高并发、分布式、代码效能等领域均有较深的造诣，乐于分享与总结，在国内外做过多次大型交流和培训，引起强烈共鸣。

高海慧

阿里云资深开发工程师，花名鸣莎。先后在阿里巴巴集团B2B技术部和阿里云任职，承担过商业化、双十一、智能调度及10亿/天的信息采集系统建设等大型项目。在调度匹配、大数据处理和高并发领域具有丰富的实践经验和创新成果。

码出高效

Java开发手册

杨冠宝（孤尽） 高海慧（鸣莎） 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内容简介

《码出高效：Java 开发手册》源于影响了全球 250 万名工程师的《阿里巴巴 Java 开发手册》，作者静心沉淀，对 Java 规约内容的来龙去脉进行了全面而彻底的梳理。本书以实战为中心，以新颖的角度全面阐述面向对象理论，逐步深入地探索怎样成为一位优秀的开发工程师。比如：如何驾轻就熟地使用各类集合框架，如何得心应手地处理高并发多线程问题，如何顺其自然地写出可读性强、可维护性好的优雅代码。

本书旁征博引、文风轻松，秉持“图胜于表，表胜于言”的理念，深入浅出地将计算机基础、面向对象思想、JVM 探源、数据结构与集合、并发与多线程、单元测试等知识客观、立体地呈现出来。紧扣学以致用、学以精进的目标，结合阿里巴巴实践经验和故障案例，与底层源码解析融会贯通，娓娓道来。

本书以打造民族标杆图书为己任，追求极致，打磨精品，在技术广度和深度上兼具极强的参考性，适合计算机相关行业的管理者和研发人员、高等院校的计算机专业师生等阅读。无论是初学者入门，或是中、高级程序员的进阶提升，本书均为不容置疑的选择。

图书在版编目（CIP）数据
码出高效：Java 开发手册 / 杨冠宝，高海慧著 .—北京：电子工业出版社，2018.10
ISBN 978-7-121-34909-6

Ⅰ.①码… Ⅱ.①杨… ②高… Ⅲ.① JAVA 语言－程序设计 Ⅳ.① TP312.8

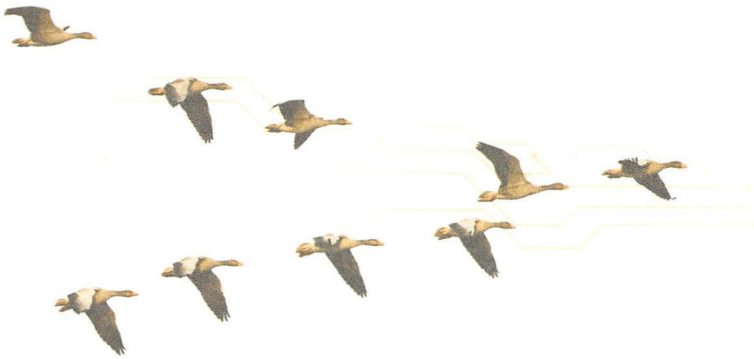
中国版本图书馆 CIP 数据核字 (2018) 第 187939 号

责任编辑：孙学瑛
印　　刷：北京市大天乐投资管理有限公司
装　　订：北京市大天乐投资管理有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：720×1000　1/16　　印张：19　　字数：349 千字
版　　次：2018 年 10 月第 1 版
印　　次：2018 年 10 月第 2 次印刷
定　　价：99.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。



编委会

毕 玄　索 尼　多 隆　叶 渡　至 简
冰 够　晗 光　广 陌　金 戟　骏 烈
昶 乐　曾 侯　胜 燕　默 研　文 龙
楠 山　锦 铭　遁 木　润 谨　玄 坛
虎 仔　息 羽　可 期　澳 明　星 楚
唔 哈　弗 止　嵛 山　辰 颜　别 象
喻 阳

出版团队

郭　立　孙学瑛　康　旭　宋亚东
安　娜　白　涛　李　玲　王　乐

前　言

《码出高效：Java 开发手册》书名中的“码”既是动词，也是名词，希望我们在“码” 出高效的同时编写出高质量的代“码”。本书从立意到付梓，历时超过两年，期间推翻数次写作思路，历经曲折与艰辛，只希望为时代奉献一本好书，打造中国计算机民族标杆图书。愿这本书能陪伴在众多开发工程师的身边，大家一起进步、一起成长、一起感受编程的魅力。

本书缘起

这本书源于影响了全球 250 万名工程师的《阿里巴巴 Java 开发手册》（简称《手册》）。2017 年 2 月 9 日，《手册》以 PDF 文件的方式正式发布第一个版本。一经公布，在各大网络平台引发热议，堪称中国人自己原创的 Java 编程规范，甚至走进硅谷，世界开始听到中国程序员的声音。2017 年杭州云栖大会开源此手册配套的扫描插件后，一度攀升至世界第一，已经有 45 万名开发工程师直接下载，数以千计的企业进行部分修订后在内部推行。《手册》在研发效能、人才培养与系统稳定性领域都产生了巨大而深远的影响，已经成为重要的开发基础标准文件。

从团队协作角度来说。虽然别人都说开发工程师是搬砖的码农，但我们知道自己是追求个性的艺术家。我们骨子里追求着代码的美、系统的美、设计的美，代码规约其实就是一个对程序美的定义。曾经程序员最引以为豪的代码，却因为代码规约的缺失严重制约了相互之间的高效协同，频繁的系统重构和心惊胆战的维护似乎成了工作的主旋律，那么如何走出这种怪圈呢？众所周知，互联网公司的效能是企业的核心竞争力，体现在开发领域上，其实就是沟通效率和研发效率。本书的书名“码出高效”指的就是高效沟通与协作。大雁是一种非常讲究团队配合的鸟类，它们飞翔的队形可以有效地减少空气阻力，所以封面选择大雁作为背景，传递团队沟通与协作的理念，顺利达到共同的目标。

从个人发展角度来说。在计算机编程日益普及的今天，程序员群体日益壮大，本书以初级入门、中级进阶、高级修炼为目标，指导工程师的成长之路。涵盖计算机领域基础知识、面向对象理念、数据结构与集合、高并发多线程、异常和日志以及单元测试等多个方面，讲解由浅入深，囊括了一名开发工程师应具备的基本素质。本书以计算机民族标杆图书为自我要求，追求极致，打磨精品，目的是使读者在计算机综合素质上有大幅的提升。

从系统稳定角度来说。稳定是基础设施的关键目标，也是每个开发工程师考核中的重要指标之一。本书搜集线上的真实故障，经过整理后与相应的知识点结合在一起进行讲解，身临其境，阐述了知其不然的背后逻辑，提出更好的实现方案，最终以技术解决技术问题。

本书内容

本书共包括 9 章，每章的主要内容如下：

第 1 章从计算机基础知识说起，介绍基础的 0 与 1 表示与位运算、字符集、CPU 与内存、网络传输与信息安全基础知识，目的是为后文打下知识基座。

第 2 章走入面向对象的世界，介绍具有划时代意义的编程思想理念，覆写、重载等重要概念含义，类、方法等基础模块规范。

第 3 章聊聊代码风格，针对命名、代码展示、控制语句和注释等进行分类定义。虽然代码风格不影响程序运行和执行效率，但是对于团队高效协作来说具有重要意义。通过践行本章，读者可以顺其自然地写出可读性强、可维护性好的优雅代码。

第 4 章揭开 Java 的神秘面纱，探讨底层 JVM 核心。从字节码说起，分析类加载的过程，并结合内存布局，讲解对象创建与垃圾回收等知识点。

第 5 章首先归纳了系统中各类的异常，以及定义各种异常的处理方式，然后定义了日志使用规范，以达到监控运行状况，回溯异常等目的。

第 6 章是重点章节，以数据结构为基础，引申至集合框架，再到重点集合源码分析，最后介绍高并发集合框架，目的是让读者对集合的了解成竹在胸，运用得心应手。尤其是对于集合中使用到的红黑树特性，经过一步步分析，相信使读者不再发怵于树的平衡性与左右旋转。

第 7 章也是重点章节，走进并发与多线程。由并发与并行等基础概念开始，引申到线程安全，介绍几种常见的锁实现，然后讲解线程同步方案，最后扩展到如何正确使用线程池，如何深度解析 ThreadLocal 的安全使用等。目的是让读者深入理解并且

安全规范地实现并发编程，得心应手地处理好高并发多线程问题，提高生产效率。

第 8 章分析了单元测试的重要意义、基本原则、开发规范和评判标准。单元测试的重要意义在于它是一件有情怀、有技术素养、有长期收益的工作，是保证软件质量和效率的重要手段之一。

第 9 章回归初心，聊聊开发工程师的成长方法论，讲解代码规约的起源与落地方法。虽然这更像一个故事，但是它对于推动项目落地与个人成长具有借鉴意义。

本书分析的底层源码基本来自于最新发布的 JDK11，所有示例代码能够正常运行在相应的 OpenJDK 64Bit JVM 上。在阿里巴巴 Java 开发手册相关扫描插件 P3C 的开源网站上，即将公布所有相关源代码，敬请关注：https://github.com/alibaba/p3c。

本书特色

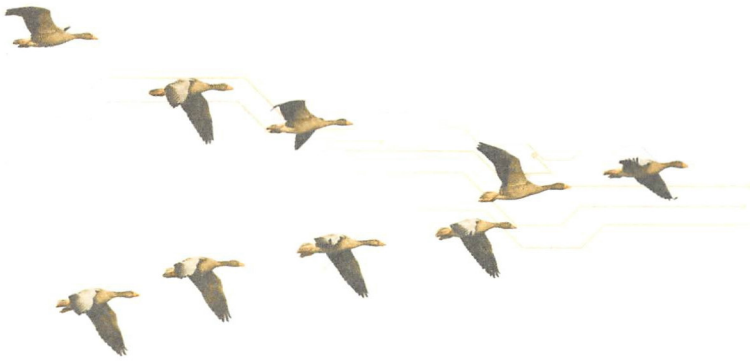
本书旁征博引、文风轻松，坚持朴实的平民化写书理念，为方便理解增加了大量生活化的例子，秉持“图胜于表，表胜于言”的理念，紧扣学以致用、学以精进的目标，结合阿里巴巴实践，与底层源码解析融会贯通，深入浅出地把知识立体、客观、丰富地呈现出来。

友情说明一下，本书的示例代码着重于解释知识点的逻辑与使用技巧，简捷明了为主，并非一一规范。当然，有技术追求的读者可以尝试总结全文不符合规范的代码，然后与我们联系，会有奖品回赠。

致谢

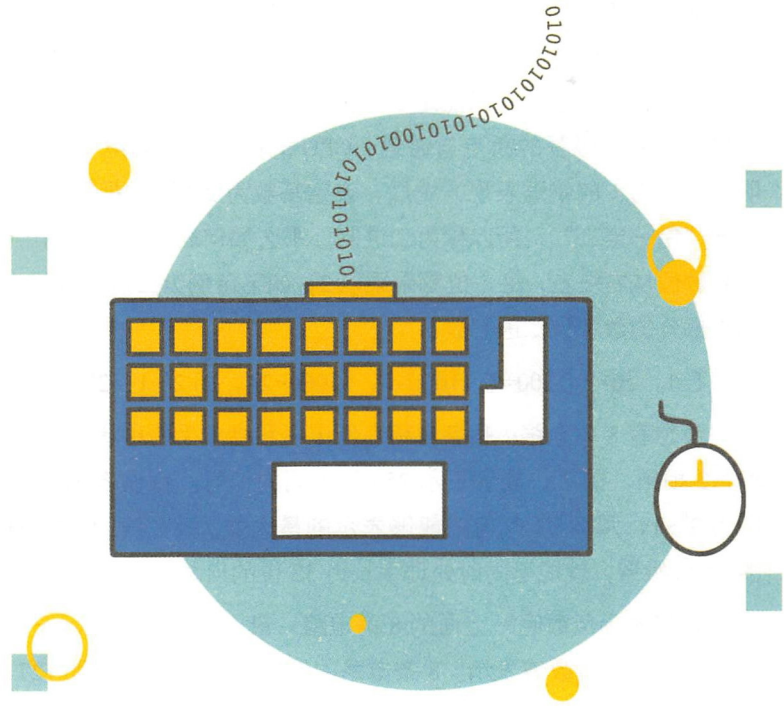
最后，要感谢在本书编写过程中，所有家人、朋友以及伙伴们的支持与帮助，让作者无后顾之忧地投入到写作中。感谢阿里云业务安全团队、研发效能事业部、AJDK、信息平台事业部、技术线 HR、技术战略部、约码项目组、P3C 项目组等团队的倾力奉献和所有支持计算机事业发展的开发工程师们。感谢团队各级 Leader 一如既往地支持。感谢各位编委和电子工业出版社博文视点伙伴们的认真付出，你们的积极参与和认真编校保证了图书的顺利出版。

目　录



第 1 章　计算机基础

大道至简，盘古生其中。计算机的基础世界一切都是由 0 与 1 组成的。



追根究底是深度分析和解决问题、提升程序员素质的关键所在，有助于编写高质量的代码。基础知识的深度认知决定着知识上层建筑的延展性。试问，对于如下的基础知识，你的认知是否足够清晰呢？

- 位移运算可以快速地实现乘除运算，那位移时要注意什么？
- 浮点数的存储与计算为什么总会产生微小的误差？
- 乱码产生的根源是什么？
- 代码执行时，CPU 是如何与内存配合完成程序使命的？
- 网络连接资源耗尽的问题本质是什么？
- 黑客攻击的通常套路是什么？如何有效地防止？

本章从编程的角度深度探讨计算机组成原理、计算机网络、信息安全等相关内容，与具体编程语言无关。本章将不会讨论内部硬件的工作原理、网络世界的协议和底层传输方式、安全领域的攻防类型等内容。

1.1 走进 0 与 1 的世界

简单地说，计算机就是晶体管、电路板组装起来的电子设备，无论是图形图像的渲染、网络远程共享，还是大数据计算，归根结底都是 0 与 1 的信号处理。信息存储和逻辑计算的元数据，只能是 0 与 1，但是它们在不同介质里的物理表现方式却是不一样的，如三极管的断电与通电、CPU 的低电平与高电平、磁盘的电荷左右方向。明确了 0 与 1 的物理表现方式后，设定基数为 2，进位规则是“逢二进一”，借位规则是“借一当二”，所以称为二进制。那么如何表示日常生活中的十进制数值呢？二进制数位从右往左，每一位都是乘以 2，如下示例为二进制数与十进制数的对应关系，阴影部分的数字为二进制数：

1=1，10=2，100=4，1000=8，11000=24，即 $2^0=1$；$2^1=2$；$2^2=4$；$2^3=8$；$2^4+2^3=24$

设想有 8 条电路，每条电路有低电平和高电平两种状态。根据数学排列组合，有 8 个 2 相乘，即 2^8，能够表示 256 种不同的信号。假设表示区间为 0 ~ 255，最大数即为 2^8-1，那么 32 条电路能够表示的最大数为（$2^{32}-1$）=4,294,967,295。平时所说的 32 位机器，就能够同时处理字长为 32 位的电路信号。

如何表示负数呢？上面的 8 条电路，最左侧的一条表示正负，0 表示正数，1 表示负数，不参与数值表示。8 条电路的最大值为 01111111 即 127，表示范围因有正负

之分而改变为 −128 ~ 127，二进制整数最终都是以补码形式出现的。正数的补码与原码、反码是一样的，而负数的补码是反码加 1 的结果。这样使减法运算可以使用加法器实现，符号位也参与运算。比如 35 + (−35) 如图 1-1（a）所示，35−37 如图 1-1（b）所示。

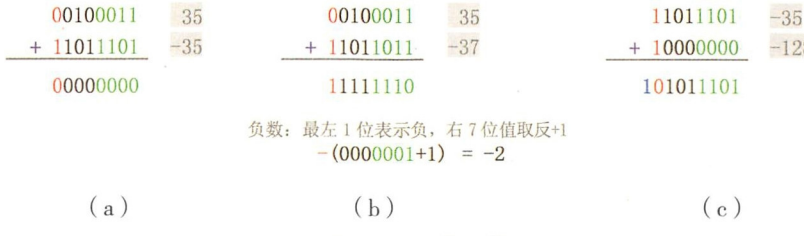


图 1-1　负数运算

加减法是高频运算，使用同一个运算器，可以减少中间变量存储的开销，这样也降低了 CPU 内部的设计复杂度，使内部结构更加精简，计算更加高效，无论对于指令、寄存器，还是运算器都会减轻很大的负担。

如图 1-1（c）所示，计算结果需要 9 条电路来表示，用 8 条电路来表达这个计算结果即溢出，即在数值运算过程中，超出规定的表示范围。一旦溢出，计算结果就是错误的。在各种编程语言中，均规定了不同数字类型的表示范围，有相应的最大值和最小值。

以上示例中的一条电路线在计算机中被称为 1 位，即 1 个 bit，简写为 b。8 个 bit 组成一个单位，称为一个字节，即 1 个 Byte，简写为 B。1024 个 Byte，简写为 KB；1024 个 KB，简写为 MB；1024 个 MB，简写为 GB，这些都是计算机中常用的存储计量单位。

除二进制的加减法外，还有一种大家既陌生又熟悉的操作：位移运算。陌生是指不易理解且不常用，熟悉是指“别人家的开发工程师”在代码中经常使用这种方式进行高低位的截取、哈希计算，甚至运用在乘除法运算中。向右移动 1 位近似表示除以 2（如表 1-1 所示），十进制的奇数转化为二进制数后，在向右移时，最右边的 1 将被直接抹去，说明向右移对于奇数并非完全相当于除以 2。在左移 << 与右移 >> 两种运算中，符号位均参与移动，除负数往右移动，高位补 1 之外，其他情况均在空位处

补 0，红色是原有数据的符号位，绿色仅是标记，便于识别移动方向。

表 1-1　带符号位移运算

正数 / 负数	向左移 << 1 位	向右移 >>1 位
正数（35 的补码 00100011）	01000110 = $2^6+2^2+2^1$ = 70	00010001 = 2^4+2^0=17（近似除 2）
负数（−35 的补码 11011101）	10111010 = 1(1000101+1) = −70	11101110 = 1(0010001+1) = −18
正数（99 的补码 01100011）	11000110 = −58	00110001 = 49
负数（−99 的补码 10011101）	00111010 = 58	11001110 = −50

左移运算由于符号位参与向左移动，在移动后的结果中，最左位可能是 1 或者 0，即正数向左移动的结果可能是正，也可能是负；负数向左移动的结果同样可能是正，也可能是负。

对于三个大于号的 >>> 无符号向右移动（注意不存在 <<< 无符号向左移动的运算方式），当向右移动时，正负数高位均补 0，正数不断向右移动的最小值是 0，而负数不断向右移动的最小值是 1。无符号意即藐视符号位，符号位失去特权，必须像其他平常的数字位一起向右移动，高位直接补 0，根本不关心是正数还是负数。此运算常用在高位转低位的场景中，如表 1-2 所示分别表示向右移动 1 ~ 3 位的结果，左侧空位均补 0。

表 1-2　无符号位移运算

正数 / 负数	向右移 >>>1 位	向右移 >>>2 位	向右移 >>>3 位
正数（35 的补码 00100011）	00010001 = 17	00001000 = 8	00000100 = 4
负数（−35 的补码 11011101）	01101110 = 110	00110111 = 55	00011011 = 27

为何负数不断地无符号向右移动的最小值是 1 呢？在实际编程中，位移运算仅作用于整型（32 位）和长整型（64 位）数上，假如在整型数上移动的位数是 32 位，无论是否带符号位以及移动方向，均为本身。因为移动的位数是一个 mod 32 的结果，即 35>>1 与 35>>33 是一样的结果。如果是长整型，mod 64，即 35<<1 与 35<<65 的结果是一样的。负数在无符号往右移动 63 位时，除最右边为 1 外，左边均为 0，达到最小值 1，如果 >>>64，则为其原数值本身。

位运算的其他操作比较好理解，包括按位取反（符号为 ~）、按位与（符号为 &）、

按位或（符号为 | ）、按位异或（符号为 ^ ）等运算。其中，按位与（ & ）运算典型的场景是获取网段值，IP 地址与掩码 255.255.255.0 进行按位与运算得到高 24 位，即为当前 IP 的网段。按位运算的左右两边都是整型数，true&false 这样的方式也是合法的，因为 boolean 底层表示也是 0 与 1。

按位与和逻辑与（符号为 && ）运算都可以作用于条件表达式，但是后者有短路功能，表达如下所示：

```
boolean a  = true;
boolean b = true;
boolean c = (a=(1==2)) && (b=(1==2));
```

因为 && 前边的条件表达式，即如上的红色代码部分的结果为 false，触发短路，直接退出，最后 a 的值为 false，b 的值为 true。假如把 && 修改为按位与 &，则执行的结果为 a 与 b 都是 false。

同样的逻辑，按位或对应的逻辑或运算（符号为 || ）也具有短路功能，当逻辑或 || 之前的条件表达式，即如下的红色代码部分的结果为 true 时，直接退出：

```
boolean e  = false;
boolean f = false;
boolean g = (e=(1==1)) || (f=(1==1));
```

最后 e 的值为 true，f 的值为 false。假如把 || 修改为按位或符号 |，执行的结果为 e 与 f 都是 true。

逻辑或、逻辑与运算只能对布尔类型的条件表达式进行运算，7&&8 这种运算表达式是错误的。

异或运算没有短路功能，符号在键盘的数字 6 上方，在哈希算法中用于离散哈希值，对应的位上不一样才是 1，一样的都是 0。比如，1^1=0 / 0^0=0 / 1^0=1 / true^true=false / true^false=true。

基于 0 与 1 的信号处理为我们带来了缤纷多彩的计算机世界，随着基础材料和信号处理技术的发展，未来计算机能够处理的基础信号将不仅仅是二进制信息。比如，三进制（高电平、低电平、断电），甚至十进制信息，届时计算机世界又会迎来一次全新的变革。

1.2 浮点数

计算机定义了两种小数，分别为定点数和浮点数。其中，定点数的小数点位置是固定的，在确定字长的系统中一旦指定小数点的位置后，它的整数部分和小数部分也随之确定。二者之间独立表示，互不干扰。由于小数点位置是固定的，所以定点数能够表示的范围非常有限。考虑到定点数相对简单，本节不再展开。下面重点介绍应用更广、更加复杂的浮点数。它是采用科学计数法来表示的，由符号位、有效数字、指数三部分组成。使用浮点数存储和计算的场景无处不在，若使用不当则容易造成计算值与理论值不一致，如下示例代码：

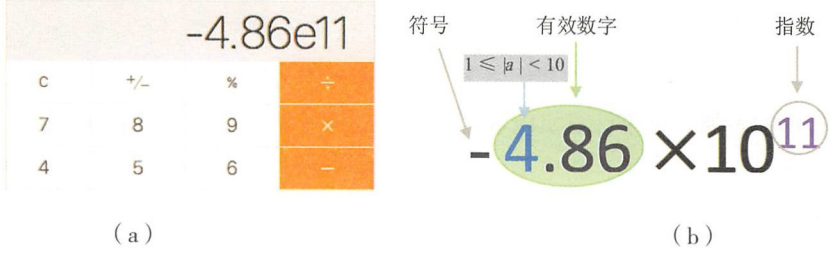
```
float a = 1f;
float b = 0.9f;

// 结果为：0.100000024
float f = a - b;
```

执行结果显示计算结果与预期存在明显的误差，本节将通过深入剖析造成这个误差的原因来介绍浮点数的构成与计算原理。由于浮点数是以科学计数法来表示的，所以我们先从科学计数法讲起。

1.2.1 科学计数法

浮点数是计算机用来表示小数的一种数据类型。在数学中，采用科学计数法来近似表示一个极大或极小且位数较多的数。如 $a \times 10^n$，其中 a 满足 $1 \leqslant |a| < 10$，10^n 是以 10 为底数，n 为指数的幂运算表达式。$a \times 10^n$ 还可以表示成 aen，如图 1-2（a）中计算器的结果所示。−4.86e11 等价于 -4.86×10^{11}，它们都表示真实值 −486000000000，具体格式说明如图 1-2（b）所示。



（a）　　（b）

图 1-2　科学计数法

科学计数法的有效数字为从第 1 个非零数字开始的全部数字，指数决定小数点的位置，符号表示该数的正与负。值得注意的是，十进制科学计数法要求有效数字的整