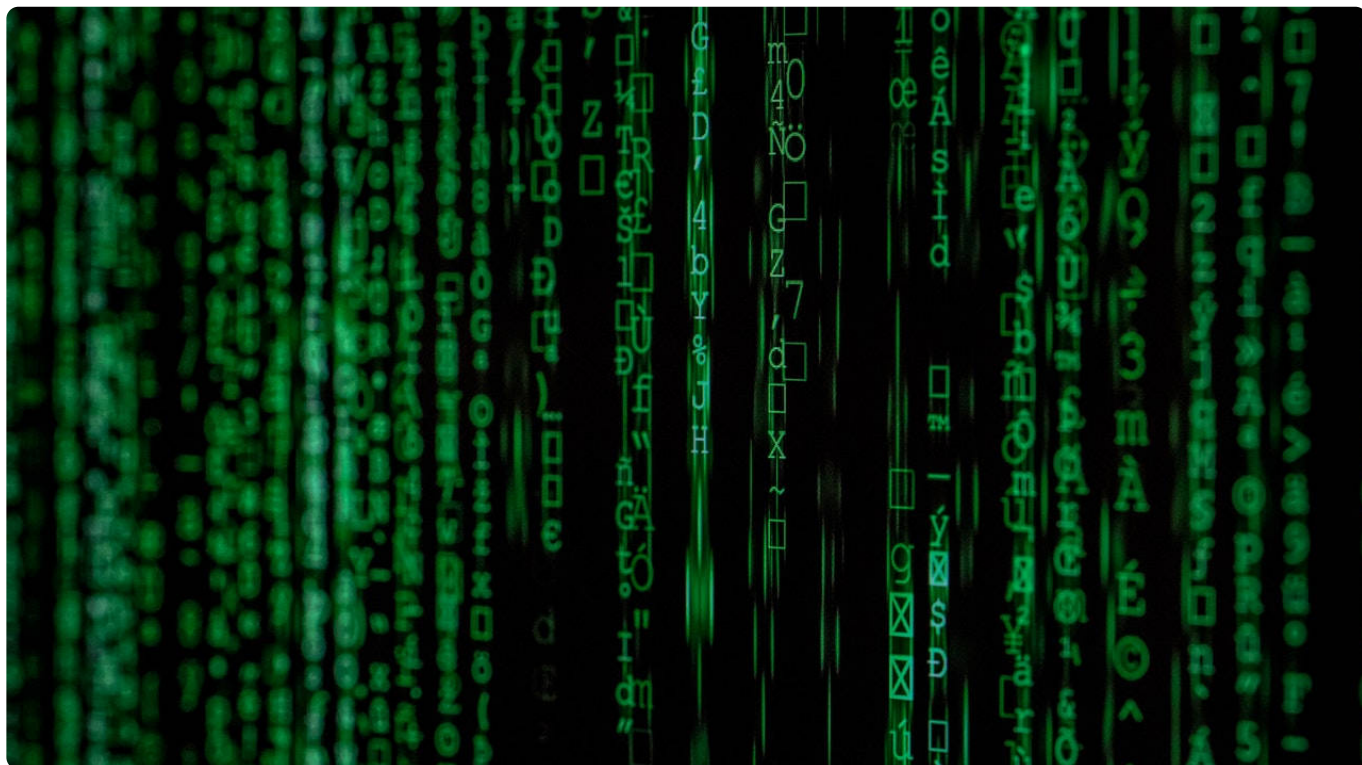


15 | 浮点数和定点数（上）：怎么用有限的Bit表示尽可能多的信息？

2019-05-29 徐文浩

深入浅出计算机组成原理

[进入课程 >](#)



讲述：徐文浩

时长 10:47 大小 9.88M




在我们日常的程序开发中，不只会用到整数。更多情况下，我们用到的都是实数。比如，我们开发一个电商 App，商品的价格常常会是 9 块 9；再比如，现在流行的深度学习算法，对应的机器学习里的模型里的各个权重也都是 1.23 这样的数。可以说，在实际的应用过程中，这些有零有整的实数，是和整数同样常用的数据类型，我们也需要考虑到。

浮点数的不精确性

那么，我们能不能用二进制表示所有的实数，然后在二进制下计算它的加减乘除呢？先不着急，我们从一个有意思的小案例来看。

你可以在 Linux 下打开 Python 的命令行 Console，也可以在 Chrome 浏览器里面通过开发者工具，打开浏览器里的 Console，在里面输入 “0.3 + 0.6”，然后看看你会得到一个什么样的结果。

 复制代码

```
1 >>> 0.3 + 0.6
2 0.8999999999999999
```

不知道你有没有大吃一惊，这么简单的一个加法，无论是在 Python 还是在 JavaScript 里面，算出来的结果居然不是准确的 0.9，而是 0.8999999999999999 这么个结果。这是为什么呢？

在回答为什么之前，我们先来想一个更抽象的问题。通过前面的这么多讲，你应该知道我们现在用的计算机通常用 16/32 个比特 (bit) 来表示一个数。那我问你，我们用 32 个比特，能够表示所有实数吗？

答案很显然是不能。32 个比特，只能表示 2 的 32 次方个不同的数，差不多是 40 亿个。如果表示的数要超过这个数，就会有两个不同的数的二进制表示是一样的。那计算机可就会一筹莫展，不知道这个数到底是多少。

40 亿个数看似已经很多了，但是比起无限多的实数集合却只是沧海一粟。所以，这个时候，计算机的设计者们，就要面临一个问题了：我到底应该让这 40 亿个数映射到实数集合上的哪些数，在实际应用中才能最划得来呢？

定点数的表示

有一个很直观的想法，就是我们用 4 个比特来表示 0~9 的整数，那么 32 个比特就可以表示 8 个这样的整数。然后我们把最右边的 2 个 0~9 的整数，当成小数部分；把左边 6 个 0~9 的整数，当成整数部分。这样，我们就可以用 32 个比特，来表示从 0 到 999999.99 这样 1 亿个实数了。

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

这种用二进制来表示十进制的编码方式，叫作**BCD 编码**（Binary-Coded Decimal）。其实它的运用非常广泛，最常用的是在超市、银行这样需要用小数记录金额的情况里。在超市里面，我们的小数最多也就到分。这样的表示方式，比较直观清楚，也满足了小数部分的计算。

不过，这样的表示方式也有几个缺点。

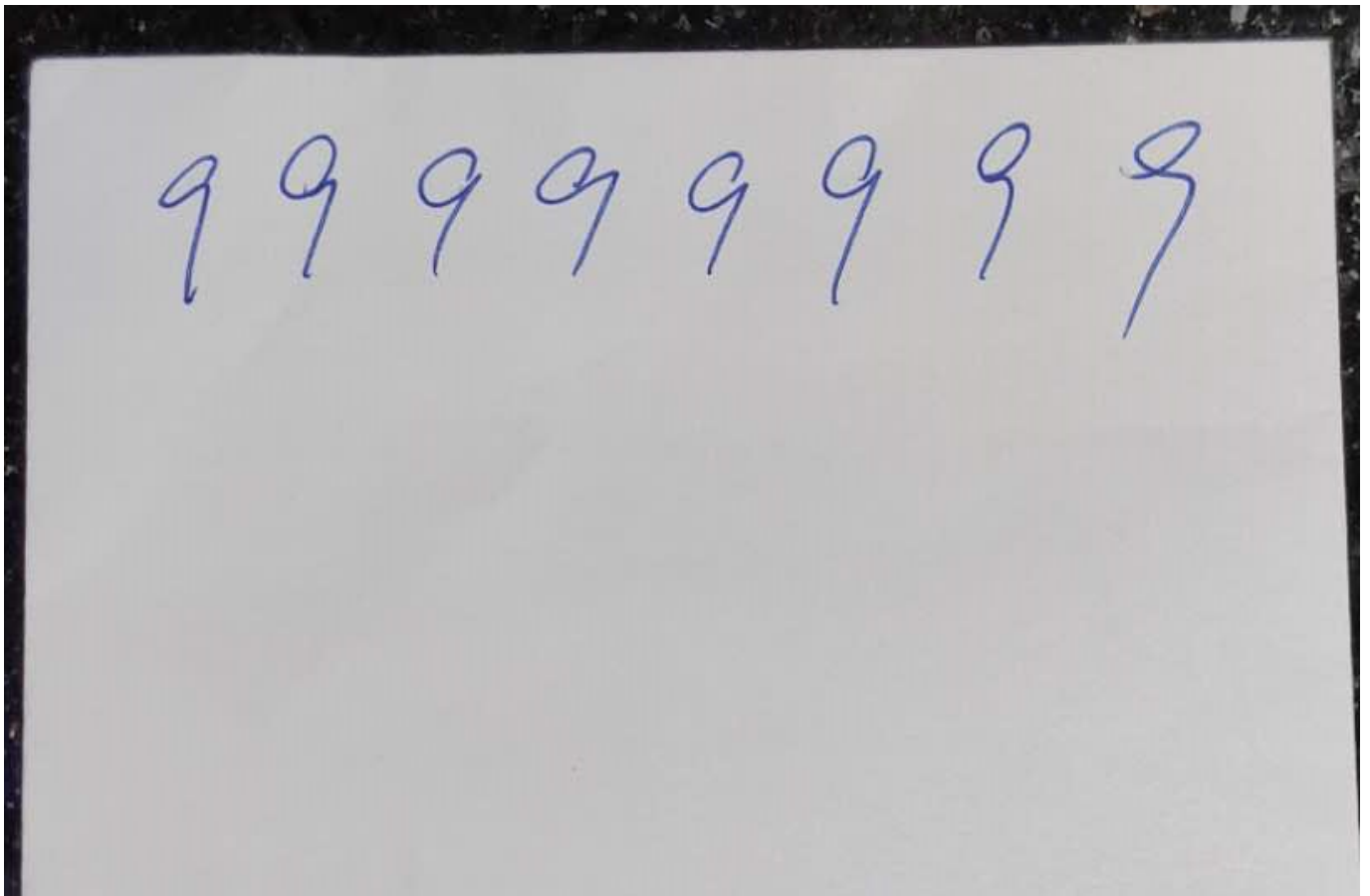
第一，这样的表示方式有点“浪费”。本来 32 个比特我们可以表示 40 亿个不同的数，但是在 BCD 编码下，只能表示 1 亿个数，如果我们要精确到分的话，那么能够表示的最大金额也就是到 100 万。如果我们的货币单位是人民币或者美元还好，如果我们的货币单位变成了津巴布韦币，这个数量就不太够用了。

第二，这样的表示方式没办法同时表示很大的数字和很小的数字。我们在写程序的时候，实数的用途可能是多种多样的。有时候我们想要表示商品的金额，关心的是 9.99 这样小的数字；有时候，我们又要进行物理学的运算，需要表示光速，也就是 3×10^8 这样很大的数字。那么，我们有没有一个办法，既能够表示很小的数，又能表示很大的数呢？

浮点数的表示

答案当然是有的，就是你可能经常听说过的**浮点数**（Floating Point），也就是**float 类型**。

我们先来想一想。如果我们想在一张便签纸上，用一行来写一个十进制数，能够写下多大范围的数？因为我们要让人能够看清楚，所以字最小也有一个限制。你会发现一个和上面我们用 BCD 编码表示数一样的问题，就是纸张的宽度限制了我们能够表示的数的大小。如果宽度只放得下 8 个数字，那么我们还是只能写下最大到 99999999 这样的数字。



有限宽度的便签，只能写下有限大小的数字

其实，这里的纸张宽度，就和我们 32 个比特一样，是在空间层面的限制。那么，在现实生活中，我们是怎么表示一个很大的数的呢？比如说，我们想要在一本科普书里，写一下宇宙内原子的数量，莫非是用一页纸，用好多行写下很多个 0 么？

当然不是了，我们会用科学计数法来表示这个数字。宇宙内的原子的数量，大概在 10 的 82 次方左右，我们就用 1.0×10^{82} 这样的形式来表示这个数值，不需要写下 82 个 0。

在计算机里，我们也可以用一样的办法，用科学计数法来表示实数。浮点数的科学计数法的表示，有一个**IEEE**的标准，它定义了两个基本的格式。一个是用 32 比特表示单精度的浮点数，也就是我们常常说的 float 或者 float32 类型。另外一个是用 64 比特表示双精度的浮点数，也就是我们平时说的 double 或者 float64 类型。

双精度类型和单精度类型差不多，这里，我们来看单精度类型，双精度你自然也就明白了。

s = 符号位	e = 指数位	f = 有效数位
1个比特	8个比特	23个比特

单精度的 32 个比特可以分成三部分。

第一部分是一个**符号位**，用来表示是正数还是负数。我们一般用**s**来表示。在浮点数里，我们不像正数分符号数还是无符号数，所有的浮点数都是有符号的。

接下来是一个 8 个比特组成的**指数位**。我们一般用**e**来表示。8 个比特能够表示的整数空间，就是 0~255。我们在这里用 1~254 映射到 -126~127 这 254 个有正有负的数上。因为我们的浮点数，不仅仅想要表示很大的数，还希望能够表示很小的数，所以指数位也会有负数。

你发现没，我们没有用到 0 和 255。没错，这里的 0（也就是 8 个比特全部为 0）和 255（也就是 8 个比特全部为 1）另有它用，我们等一下再讲。

最后，是一个 23 个比特组成的**有效数位**。我们用**f**来表示。综合科学计数法，我们的浮点数就可以表示成下面这样：

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

你会发现，这里的浮点数，没有办法表示 0。的确，要表示 0 和一些特殊的数，我们就要用上在 e 里面留下的 0 和 255 这两个表示，这两个表示其实是两个标记位。在 e 为 0 且 f 为 0 的时候，我们就把这个浮点数认为是 0。至于其它的 e 是 0 或者 255 的特殊情况，你可以看下面这个表格，分别可以表示出无穷大、无穷小、NAN 以及一个特殊的不规范数。

e	f	s	浮点数
0	0	0 or 1	0
0	$\neq 0$	0 or 1	0.f
255	0	0	无穷大
255	0	1	无穷小
255	$\neq 0$	0 or 1	NAN

我们可以以 0.5 为例子。0.5 的符号为 s 应该是 0, f 应该是 0, 而 e 应该是 -1, 也就是

$0.5 = (-1)^0 \times 1.0 \times 2^{-1} = 0.5$, 对应的浮点数表示, 就是 32 个比特。

s	e								f												
0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	...	0	
1位	8位								23位												

$s = 0$, $e = 2^{-1}$, 需要注意, e 表示从 -126 到 127 个, -1 是其中的第 126 个数, 这里的 e 如果用整数表示, 就是 $2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 = 126$, $1.f = 1.0$ 。

在这样的浮点数表示下, 不考虑符号的话, 浮点数能够表示的最小的数和最大的数, 差不多是 1.17×10^{-38} 和 3.40×10^{38} 。比前面的 BCD 编码能够表示的范围大多了。

总结延伸

你会看到, 在这样的表示方式下, 浮点数能够表示的数据范围一下子大了很多。正是因为这个数对应的小数点的位置是“浮动”的, 它才被称为浮点数。随着指数位 e 的值的不同, 小数点的位置也在变动。对应的, 前面的 BCD 编码的实数, 就是小数点固定在某一位的方式, 我们也就把它称为**定点数**。

回到我们最开头, 为什么我们用 $0.3 + 0.6$ 不能得到 0.9 呢? 这是因为, 浮点数没有办法精确表示 0.3、0.6 和 0.9。事实上, 我们拿出 0.1~0.9 这 9 个数, 其中只有 0.5 能够被精确地表示成二进制的浮点数, 也就是 $s = 0$ 、 $e = -1$ 、 $f = 0$ 这样的情况。

而 0.3、0.6 乃至我们希望的 0.9，都只是一个近似的表达。这个也为我们带来了一个挑战，就是浮点数无论是表示还是计算其实都是近似计算。那么，在使用过程中，我们该怎么来使用浮点数，以及使用浮点数会遇到些什么问题呢？下一讲，我会用更多的实际代码案例，来带你看看浮点数计算中的各种“坑”。

推荐阅读

如果对浮点数的表示还不是很清楚，你可以仔细阅读一下《计算机组成与设计：硬件 / 软件接口》的 3.5.1 节。

课后思考

对于 BCD 编码的定点数，如果我们用 7 个比特来表示连续两位十进制数，也就是 00 ~ 99，是不是可以让 32 比特表示更大一点的数据范围？如果我们还需要表示负数，那么一个 32 比特的 BCD 编码，可以表示的数据范围是多大？

欢迎你在留言区写下你的思考和疑问，和大家一起探讨。你也可以把今天的文章分享给你朋友，和他一起学习和进步。




深入浅出计算机组成原理

带你掌握计算机体系全貌

徐文浩 bothub 创始人



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

上一篇 14 | 乘法器：如何像搭乐高一样搭电路（下）？

下一篇 16 | 浮点数和定点数（下）：深入理解浮点数到底有什么用？

精选留言 (16)

写留言



小崔

2019-05-29

6

对于0.5，按照老师的说法，可以用 $s = 0$ 、 $e = -1$ 、 $f = 0$ 来表示。
但是对照表格，似乎 $s = 0$ 、 $e = 0$ 、 $f = 5$ 也可以表示？请解惑

作者回复: $s = 0$, $e = 0$ 的时候，无论 f 是多少，都是表示浮点数 0
 $f = 5$ ，底数是1.5 而不是 0.5



陆离

2019-05-29

2

如果觉得没有理解老师讲的可以参考阮一峰的一篇文章
http://www.ruanyifeng.com/blog/2010/06/ieee_floating-point_representation.html
展开

作者回复:

其实这一讲还有下篇，具体 s , e , f 怎么计算大家可以看一下周五的下篇，以及里面给的交互演示网页。



龙猫

2019-05-30

1

0.3无法被精确表达：

1、首先想到会使用这种情况： $s=0$ ， $e=0$ ， $f=11$

但却触发了这个特殊规则：当 s 和 e 都为0， f 不为0时，表达的是 $0.f$

但是， $0.f$ 的时候，无论 f 怎么取值，都无法精确表达0.3。因为0.3的精确二进制表达式1.1

展开



范宁

2019-05-29

1

老师可以讲一下计算机怎么识别规格化浮点数和非规格化浮点数吗

展开



Geek

2019-05-29

1

7个比特的话，99的二进制是1100011，32位里有四个7，那就是99999999，还剩4个比特，正好用来表示一个9，所以最大应该是9999999.99，如果表示负数，第一位是符号位，所以之前剩余的四位，最大是（正）0111和（负）1111，也即是 ± 7 ，所以结果是-7999999.99-7999999.99



愤怒的虾干

2019-05-29

1

老师，我在java里验证了下，譬如1.9999999f，小数点后的位数，即“9999999”七个9是没办法用8个bit位表示的，我猜测会失去精度变成2.0f，但是调用Float.toHexString发现是0x1.ffffep0，ffffe怎么看都不可能是9999999。于是我换了个数1.5f，16进制浮点数表示为0x1.8p0，可以看到小数点后是8,16进制的一半。这样看的话，上面的小数部分十进制显示是： $\text{ffffe}/2^{23} = 0.9999999$ ，加上小数点前的1就是1.9999999了。...

展开

作者回复: 愤怒的虾干同学你好，

toHexString表示的是把10进制转换成16进制表示。

0.9999999的小数部分转换成16进制，采用的是 乘以2 然后如果大于1去减1这样的操作过程。你试一下就知道就会是1111111...因为一共有23位长，所以最后有一位可能是0，所以就是ffffe，就是表示0.999999

以1.5f为例，小数部分是0.5

乘以2就是1.0，减1就是0

那么0.5表示成2进制就是 0.1000000

4位表示1个16进制数第一位就是8，后面都是0会截断显示。

你可以照着接下来第16讲的转换过程试一下，看看小数部分会变成什么样子。

然后把二进制转换成16进制，就能知道为什么了。



任雪龙

2019-05-29

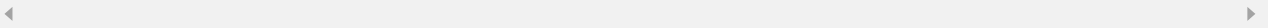
👍 1

老师，感觉今天这个讲的太粗糙了，很多东西都是用结果解释结果，比如对 0.5 这个数 s、e、f 的值，值是从哪里推导得来的都没有解释，希望可以详细解释下

展开 ▾

作者回复: 任雪龙同学你好，

这个在第16讲里面会讲解一下计算过程，因为一讲的篇幅有限，所以没有放在15讲里面讲完。



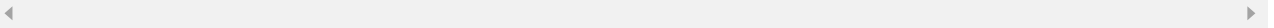
humor

2019-05-29

👍 1

如果7位表示0-99的话，32位的取值范围是0-9999999.99。如果需要负数，第一位表示符号位，取值范围是-7999999.99-7999999.99

作者回复: 👍



曾轼麟

2019-06-03

👍

老师希望您可以讲一下有关高位int32转低位bit8中溢出的过程

展开 ▾



Ant

2019-05-29

👍

打卡

展开 ▾



徐小晋

2019-05-29

👍

老师你好，我想请教一下。如何判断是否溢出。例如无符号八位十进制数185-122。这个是否溢出？以及如何去判断？



Only now

2019-05-29

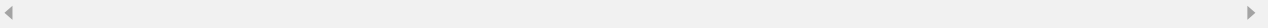


IEEE754?

展开 ▾

作者回复: 对, 整个是浮点数的标准

https://zh.wikipedia.org/zh-hans/IEEE_754



活的潇洒

2019-05-29



“算出来的结果居然不是准确的 0.9, 而是 0.8999999999999999”
经实际在python控制台和浏览器测试确实如此, 实战笔记:
<https://www.cnblogs.com/luoahong/p/10942468.html>

展开 ▾



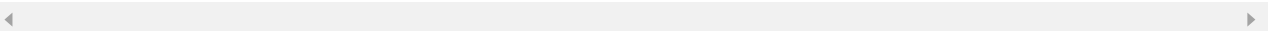
鱼向北游

2019-05-29



老师可以扩展讲一下 移码 毕竟阶码部分并不是我们常见的原码或者补码 也不是移码的常见表示 还有非规格化表示法的由来

作者回复: 这个想法不错, 我看是否搞一章加餐



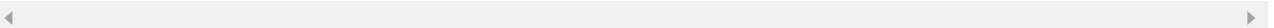
古夜

2019-05-29



对于那个公式, 底数怎么表示? 32位都给了符号位, 指数位, 小数位, 底数怎么办?

作者回复: 底数就是 1.小数位, 也就是1.f。因为是二进制, 所以底数的“整数”部分可以认为必然是1啊, 不存在其他情况





在这样的浮点数表示下，不考虑符号的话，浮点数能够表示的最小的数和最大的数，差不多是 1.17×10^{-38} 、 1.7×10^{-38} 和 3.40×10^{38} 、 3.40×10^{38} 。比前面的 BCD 编码能够表示的范围大多了

...

展开 ▾

作者回复: 最大的数，会是小数位全部为1，指数位二进制表示成127

表示成二进制就是 $1.11111... \times 2^{127}$

差不多就是 1.9999999×2^{127}

差不多正好是 $3.4028235 \times (10^{38})$

最小的数就是 $1.000... \times 2^{-126}$

差不多就是 1.0000×2^{-126}

差不多正好就是 $1.17549435 \times (10^{-38})$

