

# golang 浮点数精度丢失问题详解

作者: [xhaoxiong](#)

原文链接: <https://ld246.com/article/1632905225387>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

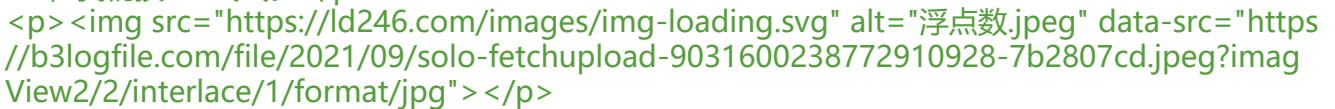
请看以下 Go 代码，会返回 `0.7` 吗？

```
var num float32
for i := 0; i < 7; ++i {
    num = num + 0.1
}
fmt.Println(num)
```

答案可能出人意料，是：`0.70000005`

也许有人会问，是不是 Go 语言的问题？换其他语言试试？

OK，我们换 JS 试试。



答案依然令人意外。

除此之外，你还可以试试 C、C++、Java、PHP 等其他语言的 float 类型相加，看得到的数据是否精确；

还有，除了语言之外，你还可以在 MySQL 等数据库中试试 float 类型数据的字段叠加，得到的数据是否精确。

我可以先告诉你答案：`只要是float类型的数据相加，无论在任何语言、任何数据库、任中间件中进行加法(减法乘法)运算，得到的数据，都不会精确。`

这是浮点类型的精度丢失现象。(Loss of significance)

要了解产生这个现象的原因，就要先了解计算机是如何定义和表示 float 类型的。

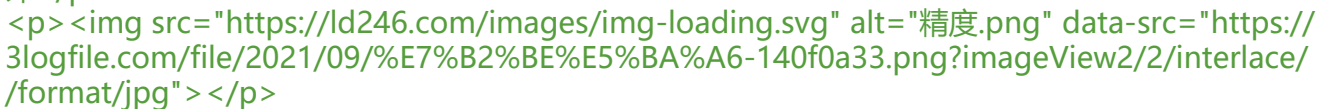
不同于正整数类型的表示方法，float 类型在计算机中的表示略显复杂，遵循的是 `IEEE 754 标准`。

下面，我们就讲一下 `IEEE 754标准`。

我们首先回顾一下整数类型在计算机中的表示。

我们知道：计算机只认识 0 和 1；那么，对于像 6 一样的这种正整数，我们要做十进制到二进制的转换。

即



所以，十进制 `6` 最终转化为二进制为 `110`。

这很好理解，但是，如何表示 `6.1` 等这类小数呢？

有人说了，可以找个特殊的符号，用来表示小数点 `.`，把 `6.1` 中 `6` 和 `1` 隔开；听起来是个不错的办法。其实 `IEEE 754` 还真就是这么做的，只不过思路略有些复杂，总体思路就是：仿照用“科学计数法”！

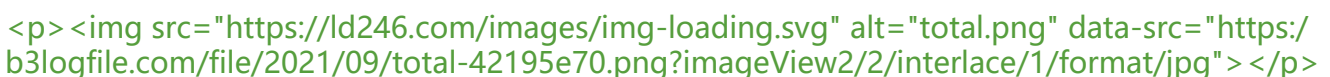
我们再回顾一下什么是 `科学计数法`。

把一个数表示成  $a \times 10^n$  的形式 ( $1 \leq |a| < 10$ ,  $a$  不为分数形式,  $n$  为整数)，这种记数法叫做科学记数法。

也就是：`1.360X10^4` 这种计数方式。

我们可以仿照科学计数法，来表示浮点数，把二进制数统一表示成 `1.0110101 X 2^n` 这种形式。

数据层面怎么表示出这种形式呢？根据 `IEEE 754` 的标准，将数据分为三部分：



从左到右分别表示：符号位(正负数)、指数位和小数位

以单精度浮点数为例，单精度浮点数一共 32 位(双精度 64 位，即平时所说的 `double` 类型)，具体内部表示为：

l01.png" data-src="https://b3logfile.com/file/2020/09/77ab2f3e83214d3783e52eb129059f60.ng?imageView2/2/interlace/1/format/jpg"></p>

<p>这里有个地方要特别注意：因为数据最终要表示成 <code>1.0110101 X 2^n</code> 这种形式，整数位在二进制下，永远都是 <code>1</code>，所以在表示 float 类型的时候，直接把 <code>1</code> 给去掉了，假如有就占据一个 bit 的空间，既然那个 bit 位上永远都是 1，所以干脆去掉。</p>

<p>那么，具体该如何展示呢？例如小数点后的数字怎么表示？<code>6.1</code> 能否写成 <code>110.1</code> 呢？如果能的话小数点后这个 1 代表什么呢？个数一？那添加几个零的话，能否认是十、一百、一千？似乎是不可以，因为这样只能满足“视觉效果”，逻辑层面直接说不通。</p>

<p>要明白在小数点后的数字代表除以 2 后的数字，例如二进制下小数点后的第一位 1 代表 1 / 2 等 <code>0.5</code>，第二位 1 代表 1/2/2 等于 <code>0.25</code>，依次类推第三位 1 则代表 <code>0.125</code>...具体请看下图：</p>

<p></p>

<p>所以，给定一个小数，譬如 <code>0.1</code>，要想得到对应的二进制数，应该是和小数点边的计算方式相反：<code>乘以2，记录整数位</code></p>

```
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight-cl">0.1 X 2 = 0.2 0
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">0.2 X 2 = 0.4 0
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">0.4 X 2 = 0.8 0
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">0.8 X 2 = 1.6 1
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">(1.6 - 1 = 0.6)
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">0.6 X 2 = 1.2 1
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">(1.2 - 1 = 0.2)
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">0.2 X 2 = 0.4 0
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">0.4 X 2 = 0.8 0
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">0.8 X 2 = 1.6 1
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">(1.6 - 1 = 0.6)
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">0.6 X 2 = 1.2 1
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">(1.2 - 1 = 0.2)
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">0.2 X 2 = 0.4 0
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">0.4 X 2 = 0.8 0
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">0.8 X 2 = 1.6 1
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">...
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">// 无限循环下去
```

```
</span></span></code></pre>
```

<p>所以，<code>0.1</code> 用二进制表示为：<code>0.000110011001100110011...</code>

因此 <code>6.1</code> 用二进制应该表示为：<code>110.000110011001100110011...</code>

用“科学计数法”表示为：<code>1.10000110011001100110011... X 2^2</code>

OK，看来小数位的数可以确定是 <code>10000110011001100110011</code>，即去掉整数位后，向后截取的 23 位数(浮点数不精确的本质原因)。

<p>符号位 0 表示正数，1 表示负数，所以可以确定是 <code>6.1</code> 的符号位是 0；现在符号位有了，小数位有了，只剩下指数 2 如的表示了，该如何表示呢？直接在 8 位的空间内转化为 <code>00000010</code>？</p>

<p>显然不可以，首先，如果指数位用 <code>原码</code> 表示，那么，针对指数位为负的情况就得加一个符号位去表示，而且还会出现两个零的情况：<code>00000000</code> 和 <code>10 0000</code>，操作起来过程复杂~</p>

<p>有人要问那如果使用补码呢？<br>

如果使用补码，会出现以下情况，请看例子：</p>

```
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
```

cl">例如：1.01 X 2<sup>-1</sup> 和 1.11 X 2<sup>3</sup>比较大小？

</span></span><span class="highlight-line"><span class="highlight-cl">  
</span></span><span class="highlight-line"><span class="highlight-cl"> 首先对比指数  
, -1 和 3, 分别转化为二进制数 ``111``和``011``;

</span></span><span class="highlight-line"><span class="highlight-cl">  
</span></span><span class="highlight-line"><span class="highlight-cl"> 如果没有其他  
辑处理, ``111``是"7", ``011``是"3", 7会小于3吗?

</span></span></code></pre>

<p>可见使用补码, 也不是很方便, 于是, 引用了另外一种编码方式——移码。<br>

先说说移码的定义: <code>将每一个数值加上一个偏置常数(Excess / bias), 通常, 当编码位数为n  
时候, bias取 "2<sup>n</sup>-1" 或者 "2<sup>n</sup>-1 - 1"</code></p>

<p>承接以上 1.01 X 2<sup>-1</sup> 和 1.11 X 2<sup>3</sup> 比较大小的例子: </p>

<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight  
cl">例如：1.01 X 2<sup>-1</sup> 和 1.11 X 2<sup>3</sup>比较大小?

</span></span><span class="highlight-line"><span class="highlight-cl">

</span></span><span class="highlight-line"><span class="highlight-cl"> 指数为-1的则表  
为 -1 + 4 = 3, 二进制表示为:011

</span></span><span class="highlight-line"><span class="highlight-cl">

</span></span><span class="highlight-line"><span class="highlight-cl"> 指数为3的则表  
为 3 + 4 = 7 二进制表示为: 111

</span></span><span class="highlight-line"><span class="highlight-cl">

</span></span><span class="highlight-line"><span class="highlight-cl"> 7 &gt; 3, 即 11  
&gt; 011 比较完毕

</span></span></code></pre>

<p>就这样, 浮点数“科学计数法”的指数位比较变得简单了, 而且, 消除了“正零”和“负零”  
相同的问题。</p>

<p>因为: </p>

<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight  
cl">假设偏移量是: 4

</span></span><span class="highlight-line"><span class="highlight-cl">

</span></span><span class="highlight-line"><span class="highlight-cl">则移码表示的0只  
: 0 + 4 = 4, 即 "100"

</span></span></code></pre>

<p>在 <code>IEEE 754</code> 中, 指数位移码的偏移量为指数位数的 <code>2<sup>n</sup>-1 - 1</cod  
>, 为 127。</p>

<p>所以, 回到 <code>6.1</code> 表示的问题上, 指数位为: <code>2 + 127 = 129</code>  
二进制表示为: <code>10000001</code></p>

<p>因此, <code>6.1</code> 在 <code>IEEE 754</code> 单精度浮点数标准的下, 表示为: <  
>

<p>

<p>好了, 现在了解了浮点数 <code>IEEE 754</code> 标准的表示方法, 知道为何浮点数相加总  
不精确了吧? </p>

<p>因为浮点数很多小数在二进制环境下很多都无法完整的表示, 只能截取部分数据来近似的表示,  
个数相加的话, 就是两个近似的数相加的和, 如果相加次数足够多, 精确度自然也就越来越低</p>

<p><a href="https://ld246.com/forward?goto=https%3A%2F%2Fi6448038.github.io%2F202  
%2F09%2F05%2Ffloat-binary%2F" target="\_blank" rel="nofollow ugc">转载</a></p>