

**浮点数运算精度问题**

首先看一个例子：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  $a = 0.1;  $b = 0.9;  $c = 1;  var\_dump(($a+$b)==$c);  var\_dump(($c-$b)==$a);  ?> |

$a+$b==$c 返回true,正确  
$c-$b==$a 返回false,错误

**为什么会这样呢？**

运算后,精度为20位时实际返回的内容如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  $a = 0.1;  $b = 0.9;  $c = 1;  printf("%.20f", $a+$b); // 1.00000000000000000000  printf("%.20f", $c-$b); // 0.09999999999999997780  ?> |

$c-$b 为 0.09999999999999997780，因此与0.1比较返回false

出现这个问题是因为浮点数计算涉及精度，当浮点数转为二进制时有可能会造成精度丢失。

**浮点数转二进制方法**

整数部分采用**除以2取余**方法

小数部分采用**乘以2取整**方法

**例如：把数字8.5转为二进制**

**整数部分是8**

8/2=4 8%2=0  
4/2=2 4%2=0  
2/2=1 2%2=0

1比2小，因此不需要计算下去，整数8的二进制为 1000

**小数部分是0.5**

0.5x2 = 1.0

因取整后小数部分为0，因此不需要再计算下去

小数0.5的二进制为 0.1

8.5的二进制为1000.1

**计算数字0.9的二进制**

0.9x2=1.8  
0.8x2=1.6  
0.6x2=1.2  
0.2x2=0.4  
0.4x2=0.8  
0.8x2=1.6

…. 之后不断循环下去，当截取精度为N时，N后的数会被舍去，导致精度丢失。

上例中0.9在转为二进制时精度丢失,导致比较时出现错误。

**所以永远不要相信浮点数已精确到最后一位，也永远不要比较两个浮点数是否相等。**

**正确比较浮点数的方法**

**1.使用round方法处理后再比较**

例子：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  $a = 0.1;  $b = 0.9;  $c = 1;  var\_dump(($c-$b)==$a);          // false  var\_dump(round(($c-$b),1)==round($a,1)); // true  ?> |

**2.使用高精度运算方法**

首先进行运算时，使用高精度的运算方法，这样可以保证精度不丢失。

**高精度运算的方法如下：**

**bcadd**将两个高精度数字相加

**bccomp**比较两个高精度数字，返回-1,0,1

**bcdiv** 将两个高精度数字相除

**bcmod** 求高精度数字余数

**bcmul** 将两个高精度数字相乘

**bcpow** 求高精度数字乘方

**bcpowmod** 求高精度数字乘方求模

**bcscale** 配置默认小数点位数，相当于Linux bc中的”scale=”

**bcsqrt** 求高精度数字平方根

**bcsub**将两个高精度数字相减

例子：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  $a = 0.1;  $b = 0.9;  $c = 1;  var\_dump(($c-$b)==$a);     // false  var\_dump(bcsub($c, $b, 1)==$a); // true  ?> |