代码之谜（持续更新）

作者: justjavac http://justjavac.iteye.com

代码之谜 - 其实，你不懂代码

目 录

1.1 代码之谜（零）- 其实，你不懂代码

发表时间: 2012-09-26 关键字: javascript, 代码之谜

前世今生

---------2012年9月28日 13时32分 新增-------

最近看本文评论，争议很多，我先说说这篇文章的前世今生吧。

我原文标题是『代码之谜 - 开篇/前言/序』，副标题是『其实，你不懂代码』，本来打算用“其实，代码中的运算符不等价于数学符号”。原文我写于2010年底，当时写在 evernote 中，用了”群“、”域“、”集合“、”关系“的概念解释了计算机中用二进制表示的离散的数和现实中连续的数之间的关系和区别。

前几天群里有人问道，遂打算写一个系列，用比较「贫」的语言把他们讲述出来。

原文首发在我的博客: http://justjavac.com/codepuzzle/2012/09/25/codepuzzle-introduction.html, 因为我也不能保证我的博客空间总是稳定的，所以，将这篇文章发布到了iteye，还可以顺便看看热心网友的评论。

----------正文分割线----------

答应了群里的兄弟们要更新博客，结果回家又是洗衣服做饭的，转眼已经10点多了。

趁洗衣机正在转的功夫，打开 Evernote 找到了以前的几段 javascript 代码，本着人性本贱(咳，咳，该死的输入法，更正「人性本荐」)的精神， 给大家共享一下，不定期更新，算是我「代码之谜」系列的开篇吧。

我喜欢读一些让人震惊的书，比如『哥德尔、艾舍尔、巴赫书：集异璧之大成』，比如『从一到无穷大』，读完后张大嘴巴，「哇噻，太不可思议了，太令我震惊了」。 本系列博客的目的之一就是让每个阅读过的人在思维方式上有所改变，变得更理性，更加会思考，会学习。

本系列说来话长，从10年开始构思，当时写在 evernote 里面，名字叫『理性，像数学家一样思考』，废话少说，言归正传，贴代码吧

第一段代码:

```
function foo1(a){
    return a + '01';
}

foo1(01)
```

第二段代码:

```
function foo2(a){
    return a + '010';
}

foo2(010);
```

第三段代码：（注：这不是 javascript 的问题，而且所有语言的问题，归根结底是 IEEE 标准中二进制浮点运算的问题，关于浮点数的详细问题请阅读 代码之谜 - 浮点数，「~~为什么没有链接呢，呵呵，因为我还没有写，正在整理中~~」。）

```
console.log(0.2 + 0.4);
```

第四段代码就相对来说简单多了：参考我一些发布的这篇为什么 ++[[]][+[]]+[+[]] = 10?。

```
[4,[3,2]][1][0];     // 3
```

----------分割线----------

2012年9月25日 22时25分 更新

还是忍不住，睡前想唠叨几句。

也许很多人第一次接触编程时，对 `i = i + 1` 都感到百撕不得骑姐(咳，我就说了嘛，必须得换一个输入法了，更正「百思不得其解」)。

> “i加上1怎么可能和i相等呢？”

后来慢慢知道了，不，确切的说，是慢慢地接受了，接受了=是赋值（前提是你学的不是pascal，我的入门语言就是它），因为**你可能根本没有思考，只是被动接受**。

再后来，我们学了 if， 开始写分支代码：

```
if (a >3) {
    // do something
}

if (a < 5) {
    // do  something
}
```

但是当我们写出 `if (3 < a < 5)` 时，居然报错了，又是百撕不… 后来被教导了，这么写是错的，应该`if (a>3 && a<5)`。 于是我们又开始接受了，认为这么写是理所当然的，而且以后的代码都是这么写的。

直到有一天，你看了 python 的入门手册，尼玛，居然逆天的出现了 ‘if 3 < a < 5:’，当时绝对又震惊了，“怎么可以这么写？”。 难道你忘了，N年前你就是这么写的，而且当时你不也认为 `3 < a < 5` 是理所当然的吗（任何一个高中生都会同意这种写法）， 为什么你现在又开始觉得 `3 < a < 5` 是种逆天写法呢，因为你在这几年的编程生涯中，已经被动接受了太多太多的东西，而且使你根本就不曾思考过， 这也是我写「代码之谜」系列的初衷。

当你被告知了，在编程中=是赋值的意思(其实他们没有告诉你，只是大部分语言这样，还有很多语言不是这样，比如pascal中:=是赋值，比如basic/VB中=即是赋值也是判断)， 但是=如果不是相等的话，那肯定有表示相等的，对，就是==，或者===。

不管是==还是=，「相等」到底是什么意思呢？=或者==或者===，即使以后会出现====，到底和数学的「相等」有多少出入呢？

知道我们遇到了传说中的NaN（很多人认为NaN既然表示Not A Number，那么他就是语言定义的一个东东，根本不存在，这是错误的，NaN是在IEEE浮点数规范中明确定义的，包括本系列后面 后提到的+0和-0问题），它不等于任何值，而且，它居然不等于它自己。

一个数居然不等于它自己，其实确切的说，是 NaN == NaN 居然返回 false， 甚至 NaN === NaN 也返回 false。是 NaN 的问题，还是==或者===的问题，抑或这根本就是相等这个概念的问题。

在集合论中，相等的三要素，不管是==还是===，都无法满足，所以说，===根本就不是相等（如果你读过数学的「群伦」就更明白了）。

相等（等价）的三要素

1. 自反省：A等于A
2. 对称性：如果A等于B，那么B等于A
3. 传递性：如果A等于B，而且B等于C，那么A等于C

当我们看到这几条定理时，我们从来没有怀疑过。 脱离了数学，我们进入了编程领域，当我们遇到了NaN， 我们知道了，在IEEE的数字表示规范里面，「自反省」是不被满足的，那么传递性和对称性呢？ 如果你找到了反例，可以留言。

也许你说，相等/等于/全等/等价这些比较特殊，其它的应该都会满足吧。 我只能告诉你（说通俗一点），以前的所有定理、公理都只适用于一个领域，当它进入另一个领域我们就不能把它当作理所当然的，也许它没有问题，比如 1+2=3，但也许这只是一个巧合， 上面我就提到了 0.2+0.4 就不等于 0.6。

计算机和现实最大的不同（也是问题的根源）就是，世界是连续的，而计算机是非连续的，是离散的。 以前我们学校图书馆有很多「计算机数学」或者「离散数学」之类的书，我现在都不明白，里面写的那些数学，是设计计算机的工程师读呢，还是使用计算机的程序员读呢？里面的内容简直就是大杂烩嘛。 什么是离散数学呢？我的理解，不连续的数学都是离散数学。比如量子论里面用到的数学，就是离散数学。

其它算数定律或者定义有不满足的吗？

再举一例，上小学刚学乘法运算的时候老师就告诉我们，3x4就是4个3相加，下面这个例子再次颠覆你的想法。

```
console.log(0.1 * 10);
console.log(0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1);
```

写完睡觉，如果大家有什么更好的例子，欢迎补充。

1.2 代码之谜（一）- 有限与无限(从整数的绝对值说起)

发表时间: 2012-10-15

一、引子

开始本章之前我先提个问题：“如果一个整数的绝对值等于它自己，那么这个数是几？”如果你回答是 0 和 所有正数，那么请你耐心读完这篇文章吧。

本章是我『代码之谜』系列的第二篇，前一篇『代码之谜 - 开篇/前言/序』简单介绍了计算机与数学的不同。

数学中有许多复杂深刻的矛盾，数学家的工作就是解释或者反驳这些矛盾， 例如有限与无限、连续与离散、存在与构造、逻辑与直观、具体与抽象、概念与计算等等。

在本章中，我们把目标缩小，主要讨论内容

- 概念：有限与无限
- 对象：8bit整数

二、绝对值之谜

终于到主题了，也许你很想知道“负数的绝对值可能等于自己吗？”，也就是 “如果x等于-x，那么x有几个解？”按照我一贯的作风，我是不会轻易告诉你答案的。 《编程珠玑》记载，作者告诉了他同事一个结果，而不是方法，最后追悔莫及。 所以，我在这里要**告诉你方法，而不是告诉你答案**。

告诉你答案之前，首先得回答个问题：“整数(8bit)的表示范围是多少？”，（也许你已经把教科书的知识背下来了，是 -2^7 到 2^7-1，也就是 -128 到 +127，现在的计算机科学都快成为文科了^_^）。

如果你不知道也没关系，至少你知道 8bit 可以表示的整数个数是 2^8 个，这个数等于多少无所谓，但是，它一定是个偶数(256)。

那么这里就有一个很有意思的问题了，0既不是正数也不是负数，把0去掉的话，整数的个数就是奇数了，整数还剩 255 个。 奇数个整数不可能平均分成两部分(正数和负数)，要么负数多一个，要么正数多一个。事实就是，负数比正数多一个，最小的负数是 -128， 最大的整数是 127。

现在的问题是， -128 的绝对值是多少呢？ -(-128)等于多少呢？是溢出呢，还是等于它自己呢？也许计算机课本没有告诉你，**整数是不会出现溢出异常的**，整数的溢出被认为是正常的舍弃（其实只要很合理）。**整数只有被0除才会异常，而浮点数，即使被0除也不会抛出异常**。 浮点数除0的操作将放在本系列浮点数篇讨论。

绝对值等于自己的数有两个，0 和最小的负数。

你可能要像香港电影里女主角那样歇斯底里的大喊“绝对值怎么可能是负数呢？ 不可能，我不信，我不信…”

忘掉你那可怜的数学知识吧，“发生这种事，大家都不想的。感情的事呢，是不能强求的。所谓吉人自有天相，做人呢，最要紧的就是开心…”跑题了，赶紧回来。

在经典数学（非皮亚诺算术系统，皮亚诺绝对是欧几里德的铁杆粉丝，要不怎么会有如此天才的构想，这个以后会给大家普及）中，绝对值定义为：“从原点到点A的距离，称为A的绝对值，它是一个非负数”。 既然讲到了距离，不妨剧透一下（本系列“逻辑篇”会涉及到），两个数的大小在数学中如何定义，“距离数轴原点的距离远近”，计算机中大小如何定义的呢？给大家留个作业吧（别告诉我是设计编程语言或者设计电脑的科学家规定的，计算机科学绝对不是文科）。

计算机中没有数轴，绝对值是如何定义的呢？看看java、C、Python的源码（感谢那些开源大牛），和咱们学的小学数学一样。

```
abs(x) :=  (x >= 0) ? x : -x
```

翻译过来就是，x的绝对值定义为：正数和0的绝对值等于它自己，负数的绝对值等于-x。（这里使用的是-x，而没有用0-x，因为在浮点数中，这两者是有区别的。）

三、深入 -x

那么 -x 是如何计算的呢？ 计算是数学概念，在计算机中，我们应该说 -x是如何求值的呢？还得回到源码，我只看了linux中关于C的源码，如果你看过其它语言源码发现和我说的不同，请联系我。 学过计算机原理的都知道，负数在计算机中以补码形式存储，计算补码的方式和取反操作类似。

```
符号位不变，其它位取反，最后加一。
```

比如 -5

```
原码:       1000,0101
其它位取反:  1111,1010
加一:       1111,1011
```

当我们求它的绝对值时如何操作呢

```
补码:      1111,1011      这是-5在计算机中的表示
各位取反:   0000,0100
加一:      0000,0101      此时结果为+5
```

现在我们回到最小的负数问题，最小的负数在计算机中表示为 1000,000，下面我们对这个数操作

```
补码:      1000,0000
各位取反:   0111,1111
加一:      1000,0000
```

神奇吗，尼玛，居然又回到自己了。

1.3 代码之谜（二）- 语句与表达式

发表时间: 2012-11-01

虽然文章标题是『语句与表达式』，在这篇文章中，我将陈述一个观点 **每个表达式都有一个值**。 在此之外，也会继续表述这个『代码之谜』系列的主题——数学与计算机之间被经常忽略的矛盾。

简单的讲

- “表达式”（expression）是一个单纯的运算过程，总是有返回值；
- “语句”（statement）是执行某种操作，没有返回值。

使用表达式也是函数式编程语言所提倡的，而传统命令式编程语言都是语句的堆砌。

表达式和语句如何区分呢？ 最简单最直观的鉴别方法就是， **后面有分号的是语句**， 这是一个充分条件而不是必要条件。 有分号，就是语句；没有分号，就不一定了，也可能是语句，也可能是表达式。

在动态语言——比如javascript——中是通过上下文来区分这两者的。

假如如果 `function foo(){}` 在一个赋值表达式的一部分，则认为它是一个表达式。 **表达式的一部分，也是表达式。** 而如果 `function foo(){}` 被包含在一个函数体内，或者位于程序中，则将它作为一个语句。

```
function foo(){}; // 声明，因为它是程序的一部分
var bar = function foo(){}; // 表达式，因为它是赋值表达式的一部分

new function bar(){}; // 表达式，因为它是New表达式的一部分

(function(){
        function bar(){}; // 声明，因为它是函数体的一部分
})();
```

还有一种不那么显而易见的表达式，就是被包含在一对圆括号中—— `(function foo(){})`。 将这种形式看成表达式同样是因为上下文的关系：（和）构成一个分组操作符，而 **分组操作符只能包含表达式**：

```
(function foo(){}); // 函数表达式：注意它被包含在分组操作符中
(var x = 5);  // error! 分组操作符只能包含表达式，不能包含语句（这里的var就是语句）
```

今天突然有人问我：

```
alert(eval(data));
```

为什么会报错呢？data 是一个对象，按理说应该会弹出 ObjectObject 啊。 这是因为，当我们写

```
{"username" : "justjavac"}
```

时，它并不是一个对象。 因为我们知道有一种表示数据的方法叫做 json（javascript对象表示法）， 所以想当然的认为这应该是一个对象。 其实，在大部分编程语言中，大括号（{}）表示的不是对象，而是代码块，这段代码其实等价于

```
{
        "username" : "justjavac"
}
```

很显然，`"username" : "justjavac"` 并不是合法的语句。 然而解决方法也很简单，就是添加括号——分组操作符

```
({"username" : "justjavac"})
```

这样就构成了一个合法的表达式，当我们进行 json 对象解析的时候可以写如下代码：

```
eval('(' + json + ')')
```

在表达式中，只能存在表达式，不能存在语句。

例如表达式

```
(var a = 4) + 4;
```

这段代码将产生一个错误，因为 `var a = 4` 是一个语句， 而不是表达式—— **对表达式求值必须返回值，但对语句求值则未必有返回值。**

类似的

```
if (var a = 0) {}
```

也产生错误，因为 `var a = 0' 是一条语句，而 **语句没有返回值**。if 语句的语法结构为`

```
if (expression) {
    statement;
    statement;
    ……
}
```

因此

```
if (var a = 0) {}
```

是错误的，但是

```
if (true) {
    var a = 0;
}
```

则是正确的。

最后重申一下，**每个表达式都有一个值**。 理解了这个，就可以很容易的理解 FP（函数式编程）的一些核心思想了。

1.4 代码之谜（三）- 运算符

发表时间: 2012-11-01 关键字: 代码之谜, 运算符

本章是我『代码之谜』系列的第四篇，如果链接打不开可以在咱iteye找到以前的文章，http://justjavac.iteye.com/blog/1685559。

从最简单的运算符加号(+)说起，加号(+)是个二元运算符——也就是说，**加号只把两个数联接起来，从来不把第三个或者更多的联接起来**。

因此，“1加2加3” 在计算机中被表述为：

```
(1 + 2) + 3       // a
```

或者

```
1 + (2 + 3)       // b
```

虽然我们通常写做 `1 + 2 + 3`，但是并不意味这它和我们数学中的 1+2+3 是等价的。

那么数学中的 1+2+3 到底表示的是 a 呢，还是 b 呢？

如果计算机的求值是左结合的，那么此表达式等价于第一种a； 如果是右结合的，那么此表达式等价于第二种b。

`1 + 2 + 3` 简单的理解就是 “把1、2、3加在一起”， 确实，在我们接触到的数学里面，就是把三个数加起来。 但是在编程语言中，却不仅仅这样。

就像前面说的那样，+号无法操作三个或者更多的数，参与加法运算的只能是两个数。

顺便说一句，正号、负号是一元运算符，虽然它们和二元运算符加、减用相同的符号， 但是他们却是不同的，所以不要想当然的认为 +4 就等价于 0+4，其实它们不是等价的，

+4 是一个整数，但是 0+4 是一个加法表达式，这个表达式的求值结果正好是 +4。

在 java 中，我们可以写 `short a = +4`，但是当我们写 `short a = 0 + 4` 时则产生一个警告。

还有一个其它例子，同样是关于 short 的，

```
short b = 1;
short b = b + 4;    // 警告
short b += 4;       // 无警告
```

那么 `1 + 2 + 3` 是如何运算的呢？ 在冯诺依曼体系架构的编程语言中， 这里有一个副作用——我习惯称那些“计算机的运算过程与程序员的大脑思考过程不一样时，则称为副作用”（虽然书本里面没有这么写过，但我一向这么认为）， **本来你以为会是这样，结果计算机偏偏就不是这样做的，我称他为副作用。**

如果看过前面的『语句与表达式』，这可以这么理解：

1 + 2 是一个表达式，它的返回值是 3。 这个表达式的返回值再参加到另一个表达式中 3 + 3，最后得出结果6。

我们用语句（Statement）来改写这段代码：

```
// ?? 1 + 2 + 3
var a = 1 + 2;
var b = b + 3;
```

如果我们用 lisp 语言对这个表达式求值，则没有副作用。

```
(+ (+ 1 2) 3)
```

如果你还没有懂，或者这个例子太有特殊性，那么我们换一个

```
5 > 4 > 3
```

在数学中，这个算式的值为 true。当我们用C语言来写这段代码，它返回的确实 false。

原因和上面的一样，大于号(>)是二元运算，它无法直接比较三个数，5 > 4 返回的结果是 true， 当用 true 和 3 比较时，true 被转换称 1，也就是 `1 > 3`，最终的结果自然就是 false 了。

总之，回归到了『语句与表达式』篇的那个观点：在编程语言中 **每个表达式都有一个值**。

编程语言中的运算符和数学中的运算器虽然一样，但是它们却并不等同。 **当你写程序时，要写给人看； 当你调试程序时，要学会用计算机的方式思考代码的含义。**

我习惯于把运算符理解为函数，比如 2 + 5 其实就是 add(2, 5) 或者 2.add(5)。 难道我会偷偷的告诉你 “其实很多语言都是这么做的”。

1.5 代码之谜（五）- 浮点数（谁偷了你的精度？）

发表时间: 2012-11-13 关键字: 代码之谜, codepuzzle, 浮点数

****光棍节加长版****

如果我告诉你，中关村配置最高的电子计算机的计算精度还不如一个便利店卖的手持计算器，你一定会反驳我：「今天写博客之前又忘记吃药了吧」。

你可以用最主流的编程语言计算 `0.2 + 0.4`，如果你使用的是 Chrome、FireFox、IE 8+，可以按 F12 键，然后找到 「控制台」，输入上面的 **表达式** `0.2 + 0.4`，回车。

然后再用最简陋的计算器（如果你没有手持计算器没关系，手机、电脑都自带一个计算器，打开“运行”，输入`calc`，回车）再计算一下刚才的 **算式** 0.2 + 0.4。

怎么样？同意我的观点了吧！ **再简陋的计算器也比超级计算器的精度高，关键不在于它的频率和内存，而在于它是如何设计、如何表示、如何计算的。**

不能表示 VS 不能精确表示

在上一章『浮点数（从惊讶到思考）』（iteye地址：http://justjavac.iteye.com/blog/1725977）中我们讲到用浮点数表示 **数** 时出现的问题——很多数都 **不能表示**。（注意 **浮点数表示的是数，而不仅仅是小数**。）

如果你数学比较好，或者你确信你身体健康，没有心脏病、高血压，没有受过重大精神创伤，那我告诉你， 在浮点数的表示范围内，有多于 99.999…% 的数在计算机中是 **不能表示** 的。 真的是太令人吃惊，也太令人遗憾了。 真相总是很残忍。

请注意我使用的措辞，区别开 **不能表示** 和 **不能精确表示**。

下面我从数量级分析一下，32bit 浮点数的表示范围是 10 的 38 次方，而表示个数呢，是 10 的 10 次方。 能够被表示的数只有 1/100000000…. （大概有30个零），这个数多大呢？还记得那个国际象棋和麦子的故事吗？

为了让你了解 **指数的威力**，我再举个例子：

有一张很大很大的纸，对折 38 次，会有多高呢？ 一米？一百米？比珠峰还高？再次考验你心脏承受能力的时刻到了：它不仅仅比珠峰高，其实它已经快到达月球了。

回到原来的话题，还有更残忍的真相。 在剩下的可以表示的不到 0.000…1% 的数中，又有多少不能精确表示呢？这就是我写这篇博客的目的。

上一章中我还给出了一种用定点数精确表示小数的方法。 事实上，手持计算器、java 中的 BigDecimal、C# 中的货币类型、MySQL 中的 NUMERIC 类型就是这么干的。 你还记得在数据库中添加字段时的 SQL 语句是如何写的吗？现在明白为什么我说 **再简陋的计算器也比超级计算器的精度高** 了吧。

这篇博客我将为大家讲解为什么很多数 **不能精确表示**，本篇可能比较烧脑子，我会尽量用最通俗的语言，最贴近现实的例子来讲解，不在乎篇幅有多长，关键是要给大家讲明白。下一篇，你将了解到浮点数如何工作，以及为什么很多数 **不能表示**。

热身 —— 问：要把小数装入计算机，总共分几步？你猜对了，3 步。

- 第一步：转换成二进制
- 第二步：用二进制科学计算法表示
- 第三步：表示成 IEEE 754 形式

在上面的第一步和第三步都有可能 **丢失精度**。

十进制 VS 二进制

下面我们讨论如何把十进制小数转换成二进制小数（什么？你不会？请自觉去面壁）。

考虑我们将 1/7（七分之一） 写成小数的时候是如何做的？

用 1 除以 7，得到的商就是小数部分，剩下的余数我们继续除以 7，一直除到什么时候结束呢？ 有两种情况：

1. 如果余数为 0。yeah！终于结束了，洗洗睡吧
2. 当除到某一步时，余数等于 1… 停！stop！等一下，我发现有什么地方怪怪的。余数为 1，余数如果为 1 的话，再继续除下去，不就又是 1/7 了吗？绕了一个大弯，又回来了？对，你猜的很对，它永远不会结束，它循环了。

注意我上面说的 情况2，我们判断他循环，并 **不是从直观看感觉它重复了，而是因为**在计算过程中，它又回到了开头。**为什么这么说呢？当你计算一个分数时，它总是连续出现 5，出现了好多次，例如 0.5555555… 你也无法断定它是无限循环的，比如一亿分之五。**

记得高中时，从一本数学课外书学到了手动开平方的方法，于是很兴奋的去计算 2 的平方根，发现它的前几位是 1.414，哇，原来「2的平方根」等于 1.414141…。很多天以后，当我再次看到我的笔记时，只能苦笑了，「2的平方根」不可能循环啊，它可是一个无理数啊。

你可能不耐烦了，叽哩哇啦说这么多，有用吗？当然有用了，以后如果 MM 问你：你会爱我到什么时候？你可以回答她：我会爱你到 1/7 的尽头。难道我会把我的表白方式告诉你们吗？ **我对你的爱就像圆周率，无限——却永不重复。**

扯远了，现在会到主题。 你也许会说：我明白了，循环小数不能精确表示，放到计算机中会丢失精度； 那么有限小数可以精确表示吧，比如 0.1。

对于无限小数，不只是计算机不能精确表示，即使你用别的办法（省略号除外），比如纸、黑板、写字板…都无法精确表示。什么？手机？也不能，当然不能了。不，不，iPad也不行，1万买的也不行，真的，再贵的本子也写不下。

哪些数能精确表示？

那么 0.1 在计算机中可以精确表示吗？

答案是出人意料的， **不能。**

在此之前，先思考个问题： **在 0.1 到 0.9 的 9 个小数中，有多少可以用二进制精确表示呢？**

我们按照乘以 2 取整数位的方法，把 0.1 表示为二进制（我假设那些不会进制转换的同学已经补习完了）：

```
(1) 0.1 x 2 = 0.2  取整数位 0 得 0.0
(2) 0.2 x 2 = 0.4  取整数位 0 得 0.00
(3) 0.4 x 2 = 0.8  取整数位 0 得 0.000
(4) 0.8 x 2 = 1.6  取整数位 1 得 0.0001
(5) 0.6 x 2 = 0.2  取整数位 1 得 0.00011
(6) 0.2 x 2 = 0.4  取整数位 0 得 0.000110
(7) 0.4 x 2 = 0.8  取整数位 0 得 0.0001100
(8) 0.8 x 2 = 1.6  取整数位 1 得 0.00011001
(9) 0.6 x 2 = 1.2  取整数位 1 得 0.000110011
(n) ...
```

我们得到一个无限循环的二进制小数 0.000110011…

我为什么要把这个计算过程这么详细的写出来呢？就是为了让你看，多看几遍，再多看几遍，继续看… 还没看出来，好吧，把眼睛揉一下，我提示你，把第一行去掉，从 (2) 开始看，看到 (6)，对比一下 (2) 和 (6)。 然后把前两行去掉，从 (3) 开始看…

明白了吧，0.2、0.4、0.6、0.8 都不能精确的表示为二进制小数。 难以置信，这可是所有的偶数啊！那奇数呢？ 答案就是：

0.1 到 0.9 的 9 个小数中，只有 0.5 可以用二进制精确的表示。

如果把 0.0 再算上，那么就有两个数可以精确表示，一个奇数 0.5，一个偶数 0.0。 为什么是两个呢？因为计算机二呗，其实计算机还真够二的。

> **世界上有 10 种人，一种是懂二进制的，一种是不懂二进制的。**

其实答案很显然，我再领大家换个角度思考，0.5 就是一半的意思。 在十进制中，进制的基数是 10，而 5 正好是 10 的一半。 2 的一半是多少？当然是 1 了。 所以，十进制的 0.5 就是二进制的 0.1。如果我用八进制呢？ 不用计算你就应该立刻回答：0.4；转换成十六进制呢，当然就是 0.8 了。

$(0.5)_{10} = (0.1)_2 = (0.4)_8 = (0.8)_{16}$

如果你还想继续思考，就又会发现一个有趣的事实，我们称之为 定理A。 我们上面的数，都是小数点后面一位小数，因此，在十进制中，这样的小数有 10 个（就是 0 到 9）； 同理，在二进制中，如果我们让小数点后面有一位小数，应该有多少个呢？当然是 2 个了（0 和 1）。

哇，好像发现了新大陆一样，很兴奋是吧。那我再给你一棒，其实定理A是错的。再重申一遍 **尽信书，则不如无书**。我写博客的目的 **不是把我的思想灌输到你的脑子里，你应该有自己的思想，自己的思考方式**，当我得出这个结论时，你应该立刻反驳我：“按照你的思路，如果是 16 进制的话，应该可以精确表示所有的 0.1 到 0.9 的数甚至还可以精确表示其它的 6 个数。而事实呢，16 进制可以精确表示的数 和 2 进制可以精确表示的数是一样的，只能精确表示 0.5。”

那么到底怎么确定一个数能否精确表示呢？还是回到我们熟悉的十进制分数。

1/2、5/9、34/25 哪些可以写成有限小数？把一个分数化到最简（分子分母无公约数），如果分母的因式分解只有 2 和 5，那么就可以写成有限小数，否则就是无限循环小数。为什么是 2 和 5 呢？因为他们是 10 的因子 10 = 2 x 5。

二进制和十六进制呢？他们的因子只有 2，所以十六进制只是二进制的一种简写形式，它的精度和二进制一样。

如果一个十进制数可以用二进制精确表示，那么它的最后一位肯定是 5。

备注：这是个必要条件，而不是充分条件。一位热心网友设计出了下面的解决精度的方案。我就不解释了，同学们自己思考一下吧。

> **我有一个观点，针对小数精度不够的问题（例如 0.1），软件可以人为的在数据最后一位补 5， 也就是 0.15，这样牺牲一位，但是可以保证数据精度，还原再把那个尾巴 5 去掉。**

请同学们思考一下。

精度在哪儿丢失？

一位热心网友 独孤小败 在 OSC 上回复了我上一篇文章，提出了一个疑问：

在 java 中计算 0.2 + 0.4 得到的结果是

```
// 代码(a)
double d = 0.2 + 0.4;  // 结果是 0.6000000000000001
```

但是当直接输出 0.6 的时候，确实是 0.6

```
// 代码(b)
double d = 0.6;  // 结果是 0.6
```

好像很矛盾。很显然，通过代码(b)可以知道，在 java 中，可以精确 **显示** 0.6，哪怕 0.6 不能被精确表示，但至少能精确把 0.6 显示出来，这不是和代码(a)矛盾了吗？

这又是一个 **想当然的错误**，在直观上认为 0.2 + 0.4 = 0.6 是必然成立的（在数学上确实如此），既然(a)的结果是 0.6，而且 java 可以精确输出 0.6，那么代码(a)的结果应该输出 0.6。

其实在计算机上 0.2 + 0.4 根本就不等于 0.6 （为什么？可以查看本系列『运算符』），因为 0.2 和 0.4 都不能被精确表示。 **浮点数的精度丢失在每一个表达式，而不仅仅是表达式的求值结果。**

我们用数学中的概念类比一下，比如四舍五入，我们计算 1.6 + 2.8 保留整数。

```
1.6 + 2.8 = 4.4
```

四舍五入得到 4。我们用另一种方法

```
先把 1.6 四舍五入为 2
再把 2.8 四舍五入为 3
最后求和 2 + 3 = 5
```

通过两种运算，我们得到了两个结果 4 和 5。同理，在我们的浮点数运算中，参与运算的两个数 0.2 和 0.4 精度已经丢失了，所以他们求和的结果已经不是 0.6 了。

后记

上面一直在讨论小数，整数呢？在博客园，一位童鞋为下面的代码抓狂了：

```
JSON.parse('{"status":1,"id":9986705337161735,"name":"test"}').id;
```

把这段代码复制到 Chrome 的 Console 中，按回车， 诡异的问题出现了 9986705337161735 居然变成了 9986705337161736！原始数据加了 1。

```
9986705337161735
9986705337161736
```

一开始以为是溢出，换了个更大的数：9986705337161738 发现不会出现这个问题。

但是 9986705337161739 输出又变成了 9986705337161740！

```
9986705337161739

9986705337161740
```

测试几次之后发现浏览器输出数字的一个规律（justjavac注：其实这个规律是错误的）：

1. 十位数为偶数，个位数为奇数时会减 1，个位数为奇数时会加1
2. 十位数为奇数，个位数为奇数时会加 1，个位数为奇数时会减1

又多测了几次，发现根本没有规律，很混乱！！有时候是加，有时候是减！！

解析：

这显然不仅仅是丢失精度的问题，欲知后事如何…咳咳…静待下一篇吧。

1.6 代码之谜（四）- 浮点数（从惊讶到思考）

发表时间: 2012-11-15 关键字: 代码之谜, 浮点数, 精度, IEEE

在『代码之谜』系列的前几篇文章中，很多次出现了浮点数。 浮点数在很多编程语言中被称为简单数据类型，其实，浮点数比起那些复杂数据类型（比如字符串）来说， 一点都不简单。

单单是说明 IEEE浮点数 就可以写一本书了，我将用几篇博文来简单的说说我所理解的浮点数，算是抛砖引玉吧。

一次面试

记得多年前我招聘 Java 程序员时的一次关于浮点数、二分法、编码的面试， 多年以后，他已经称为了一名很出色的程序员。每次聚会他都会告诉我，“那次面试彻底改变了我的过去的学习方式， 我以前 **只是盲目接受知识，根本就没有自己思考过**，那次对话，比我大学四年学到的知识都多”。

我看他简历上写到读过《信息论》才谈了很多关于二分法以及编码的话题， 整个过程大概3个小时——这是我面试时间最长的一次。

因为时间久远，我把一些我能回忆起来的关于浮点数的内容整理在这篇博客中。

格式说明：

所有我说的话，都放在引用里面。他的话放在了引号（“”）里面。没有加引号的是我的心理活动或者说明。

浮点数个数

> 在 8 位计算机上，浮点数一共有多少个呢？

“8 位的好像太过时了，现在主流的是 32 位的，好像可以表示3 x 10^38。”

果然不出我所料， **很多毕业生都把计算机学成了文科，他们不是在学习理论知识，而是接受/背诵这些知识。**

> 8 位计算机可以表示的整数是多少个呢？

“这个简单，2的8次方，应该是 256 个。N 位计算机表示的整数就是 2 的 N 次方。”

他回答时显得很兴奋，因为他终于可以反驳我的观点了，他没有把计算机当作死记硬背的学科。

> 8 位计算机，或者说 **8bit 可以表示 2^8 个整数。** 如果用这 8bit 来表示字符，可以表示多少个呢？

“呵呵，当然也是 2 的 8 次方了，否则就没有必要再发明16位或者32位的 unicode 去表示汉字了。”

> 如你刚才所说，8bit 可以表示 3^38 个浮点数。那么你估算一下，2bit 可以表示多少个浮点数呢？

“既然 2bit 可以表示 4 个整数，浮点数嘛肯定比这个多，最少也得能表示 10 几个浮点数吧。”

> 好吧，按照你的思路，我说几个数。

- 0总该有吧，用 00 表示。
- 1 用 01 表示

- 2 用 10 表示
- 3 用 11 表示

现在你把 0.4 给我表示出来？

『他思索了片刻』“哦。我明白了， 2bit 可以表示 4 个数，不管是整数、小数或者字符，就算是用 2bit 表示苹果，我们也只能表示 4 个，如果想要表示更多，就得用更多的 bit 位。”

虽然他在简历中写到读过《信息论》，他对 N bit可以表示的信息量是 2^N 肯定没有完全理解，或者只是被动接受了这个定理。

过了一会儿他又继续说：“按照这个逻辑，8bit 只能表示 256 个浮点数了，这也太少了。 我有点糊涂了，浮点数的表示范围一般都得几万甚至几亿啊。”

浮点数精度

于是我在 firebug 里面写了几行代码（可以在本系列第一篇的 序言 部分找到这些代码）。

```
0.2 + 0.4


0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
```

“这怎么可能呢？JS 居然这么不严格？”

显然他把这种现象归结于 js（谢天谢地，他没有把罪过加在 firebug 身上）。 于是我用 Java 重写了上面的代码，这回他只剩目瞪口呆了。

既然他已经开始 惊讶，那么下一步就是 思考。我又稍作了解释：

> 任何语言都宣称他们的浮点数的表示范围是 3 x 10^38，这个数到底多大呢？目前所知宇宙的年龄是 1.373 x 10^10 年。
>
> 但是 32bit 最多只能表示 2^32 个数，大约是 4 x 10^9。
>
> 对比一下你就会发现令人震惊的结果。 如果把浮点数的范围比做地球，那么可以精确表示的浮点数还不到一粒芝麻大。

“这么说，0.2+0.4 是因为他不能够精确表示，所以出现了计算错误的现象。那在编程中如何避免这种问题呢？”

> 用 **定点数表示小数。**

浮点数等价于小数吗

“定点数不是整数吗？定点数怎么表示小数啊？”

很显然，有一个理论性概念错误。他没有真正理解什么是定点，什么是浮点。

> 浮点数可以表示整数吗？比如，float a = 2 可以吗？

“可以是可以，这个 2 在计算机里面应该存储的是 2.0 吧？”

计算机肯定没有存储 2.0。百分之一万的肯定。计算机存储的是0、1串。呵呵。

“我觉得浮点数应该不会存储整数的2，他存储的应该是小数的2.0，然后转换成0、1串，是这样吗？”

他一连问了我几个问题，使我感觉到，我不是在面试，而是在上课。

整数和小数是数学里面的概念，在计算机中，只有定点数和浮点数，没有整数和小数。

定点数在课本里如何定义的？

“忘了，只知道定点数就是整数，浮点数就是小数。好像老师也是这么讲的。”

那是因为你们老师不是我，如果我当老师，肯定不会这么教学生。『笑』

定点、浮点，“点”是什么意思？“点”就是小数点。 把小数点固定，通常固定在最右面，就是定点数。 把小数点浮动，就是浮点数。浮点在哪儿？这个在 IEEE 浮点数标准里面定义的。

回到前面话题，如何精确的表示小数呢？其中一种方案就是定点数。

拿 8bit 举例吧。我们可以把小数点定在中间，用 4bit 表示整数部分，4bit 表示小数部分。 这样构造方式（专业点我们称他为数据结构，一般语言把整数和小数称为简单数据类型，其实他们一点都不简单，而且比那些成了复合数据类型的字符串都要复杂的多），~~我们可以精确的表示64个小数~~，我们可以精确的表示 2^8 = 256 个小数（谢谢 mfkvfn 在 iteye 上的指正）。

在下一章，我们将构造一个 8bit 的浮点数表示形式，来深入探索浮点数不为人知的秘密。 我称它为 JJFN-134(JustJavac Float Notation，justjavac浮点数表示法)，1bit符号，3bit指数，4bit尾数。

微博 http://www.weibo.com/justjavac

代码之谜（持续更新）

作者: justjavac
http://justjavac.iteye.com