链滴

如何写出让同事无法维护的代码?

作者：jianzh5

原文链接：https://ld246.com/article/1615336770369

来源网站：链滴

对，你没看错。

本文就是教你怎么写出让同事无法维护的代码。

程序命名

- **容易输入的名字。**比如：Fred，asdf
- **单字母的变量名。**比如：a,b,c, x,y,z（陈皓注：如果不够用，可以考虑a1,a2,a3,a4,....）
- **有创意地拼写错误。**比如：SetPintleOpening， SetPintalClosing。这样可以让人很难搜索代码。
- **抽象。**比如：ProcessData, DoIt, GetData... 抽象到就跟什么都没说一样。
- **缩写。**比如：WTF，RTFSC ..... （陈皓注：使用拼音缩写也同样给力，比如： BT，TMD，TJJTD）
- **随机大写字母。**比如：gEtnuMbER..
- **重用命名。**在内嵌的语句块中使用相同的变量名有奇效。
- **使用重音字母。**比如：int ínt（注：第二个 ínt不是int）
- **使用下划线。**比如：_, __, ___。
- **使用不同的语言。**比如混用英语，德语，或是中文拼音。
- **使用字符命名。**比如：slash, asterix, comma...
- **使用无关的单词。**比如：god, superman, iloveu....
- **混淆l和1。**字母l和数字1有时候是看不出来的。

伪装欺诈

- **把注释和代码交织在一起。**

```
for(j=0; j<array_len; j+ =8)
{
    total += array[j+0 ];
    total += array[j+1 ];
    total += array[j+2 ]; /* Main body of
    total += array[j+3]; * loop is unrolled
    total += array[j+4]; * for greater speed.
    total += array[j+5]; */
    total += array[j+6 ];
    total += array[j+7 ];
}
```

- **隐藏宏定义。**如：#define a=b a=0-b，当人们看到a=b时，谁也想不到那是一个宏。
- **换行。**如下所示，下面的示例使用搜索xy_z变得困难。

```
#define local_var xy\
_z // local_var OK
```

- **代码和显示不一致。**比如，你的界面显示叫postal code，但是代码里确叫 zipcode.
- **隐藏全局变量。**把使用全局变量以函数参数的方式传递给函数，这样可以让人觉得那个变量不是全变量。
- **使用相似的变量名。**如：单词相似，swimmer 和 swimner，字母相似：ill1| 或 oO08。parseInt 和 parseInt， D0Calc 和 DOCalc。还有这一组：xy_Z, xy__z, _xy_z, _xyz, XY_Z, xY_z, Xy_z。
- **重载函数。**使用相同的函数名，但是其功能和具体实现完全没有关系。

● **操作符重载。**重载操作符可以让你的代码变得诡异，感谢CCTV，感谢C++。这个东西是可以把混代码提高到一种艺术的形式。

比如：重载一个类的！操作符，但实际功能并不是取反，让其返回一个整数。

于是，如果你使用！！操作符，那么，有意思的事就发生了—— 先是调用类的重载！操作符，然后把返回的整数给！成了 布尔变量，如果是 !!! 呢？呵呵。

文档和注释

● **在注释中撒谎。**你不用真的去撒谎，只需在改代码的时候不要更新注释就可以了。

● **注释明显的东西。**比如：/* add 1 to i */。

● **只注释是什么，而不是为什么。**

● **不要注释秘密。**如果你开发一个航班系统，请你一定要保证每有一个新的航班被加入，就得要修改2个以上的位置的程序。千万别把这个事写在文档中。

● **注重细节。**当你设计一个很复杂的算法的时候，你一定要把所有的详细细设计都写下来，没有100不能罢休，段落要有5级以上，段落编号要有500个以上，例如：1.2.4.6.3.13 – Display all impacts fo activity where selected mitigations can apply (short pseudocode omitted). 这样，当你写代码时候，你就可以让你的代码和文档一致，如：Act1_2_4_6_3_13()

● **千万不要注释度衡单位。**比如时间用的是秒还是毫秒，尺寸用的是像素还是英寸，大小是MB还是K。等等。另外，在你的代码里，你可以混用不同的度衡单位，但也不要注释。

● **Gotchas。**陷阱，千万不要注释代码中的陷阱。

● **在注释和文档中发泄不满。**

程序设计

● **Java Casts。**Java的类型转型是天赐之物。每一次当你从Collection里取到一个object的时候，你需要把其转回原来的类型。因些，这些转型操作会出现在N多的地方。如果你改变了类型，那么你不定能改变所有的地方。而编译器可能能检查到，也可能检查不到。

● **利用Java的冗余。**比如：Bubblegum b = new Bubblegom(); 和 swimmer = swimner + 1; 注变量间的细微差别。

● **从不验证。**从不验证输入的数据，从不验证函数的返回值。这样做可以向大家展示你是多么的信任司的设备和其它程序员。

● **不要封装。**调用者需要知道被调用的所有的细节。

● **克隆和拷贝。**为了效率，你要学会使用copy + paste。你几乎都不用理解别人的代码，你就可以高地编程了。（陈皓注：Copy + Paste出来的代码bug多得不能再多）

● **巨大的listener。**写一个listener，然后让你的所有的button类都使用这个listener，这样你可以在个listener中整出一大堆if...else...语句，相当的刺激。

● **使用三维数组。**如果你觉得三维还不足够，你可以试试四维。

● **混用。**同时使用类的get/set方法和直接访问那个public变量。这样做的好处是可以极大的挫败维护员。

● **包装，包装，包装。**把你所有的API都包装上6到8遍，包装深度多达4层以上。然后包装出相似的功。

● **没有秘密。**把所有的成员都声明成public的。这样，你以后就很难限制其被人使用，而且这样可以别的代码造成更多的耦合度，可以让你的代码存活得更久。

● **排列和阻碍。**把drawRectangle(height, width) 改成 drawRectangle(width, height)，等release

几个版本后，再把其改回去。这样维护程序的程序员们将不能很快地明白哪一个是对的。

- **把变量改在名字上。**例如，把setAlignment(int alignment)改成，setLeftAlignment, setRightAlignment, setCenterAlignment。
- **Packratting。**保留你所有的没有使用的和陈旧的变量，方法和代码。
- **That’s Final。**Final你所有的子结点的类，这样，当你做完这个项目后，没有人可以通过继承来展你的类。java.lang.String不也是这样吗?
- **避免使用接口。**在java中，BS接口，在C++中BS使用虚函数。
- **避免使用layout。**这样就使得我们只能使用绝对坐标。如果你的老大强制你使用layout，你可以考使用GridBagLayout，然后把grid坐标hard code.
- **环境变量。**如果你的代码需要使用环境变量。(getenv() – C++ / System.getProperty() – Java)，么，你应该把你的类的成员的初始化使用环境变量，而不是构造函数。
- **使用全局变量。**1）把全局变量的初始化放在不同的函数中，就算这个函数和这个变量没有任何关，这样能够让我们的维护人员就像做侦探工作一样。2）使用全局变量可以让你的函数的参数变得少些。
- **配置文件。**配置文件主要用于一些参数的初始化。在编程中，我们可以让配置文件中的参数名和实程序中的名字不一样。
- **膨胀你的类。**让你的类尽可能地拥有各种臃肿和晦涩的方法。比如，你的类只实现一种可能性，但你要提供所有可能性的方法。不要定义其它的类，把所有的功能都放在一个类中。
- **使用子类。**面向对象是写出无法维护代码的天赐之物。如果你有一个类有十个成为（变量和方法）可以考虑写10个层次的继承，然后把这十个属性分别放在这十个层次中。如果可能的话，把这十个类别放在十个不同的文件中。

混乱你的代码

- **使用XML。**XML的强大是无人能及的。使用XML你可以把本来只要10行的代码变成100行。而且还要逼着别人也有XML。
- **使用不同的进制。**比如：10 和010不是一样的。再比如：array = new int[]{ 111, 120, 013, 11,};
- **尽量使用void*。**然后把其转成各种类型
- **使用隐式的转型。**C++的构造函数可以让你神不知鬼不觉得完成转型。
- **分解条件表达式。**如：把 a==100分解成，a>99 && a<101
- **学会利用分号。**如：if (a);else;{ int d; d = c;}
- **间接转型。**如：把double转string，写成new Double(d).toString() 而不是 Double.toString(d)
- **大量使用嵌套。**一个NB的程序员可以在一行代码上使用超过10层的小括号（），或是在一个函数使用超过20层的语句嵌套{}，把嵌套的if else 转成 [? :] 也是一件很NB的事。
- **使用C的变种数组。**myArray[i] 可以变成*(myArray + i) 也可以变成 *(i + myArray) 其等价于 i[myrray]。再看一个函数调用的示例，函数声明：int myfunc(int q, int p) { return p%q; } 函数调用myfnc(6291, 8)[Array];
- **长代码行。**一行的代码越长越好。这样别人阅读时就需要来来回回的
- **不要较早的return。**不要使用goto，不要使用break，这样，你就需要至少5层以上的if-else来处错误。
- **不要使用{}。**不要在if else使用{}，尤其是在你重量地使用if-else嵌套时，你甚至可以在其中乱缩进码，这样一来，就算是最有经验的程序员也会踩上陷阱。

原文链接：如何写出让同事无法维护的代码?

● **琐碎的封装。**比较封装一个bool类，类里面什么都做，就是一个bool.

● **循环。**千万不可用for(int i=0; i<n; i++)使用while代替for，交换n和i，把<改成<=，使用 i–调整伐 。

测试

● **从不测试。**千万不要测试任何的出错处理，从来也不检测系统调用的返回值。

● **永远不做性能测试。**如果不够快就告诉用户换一个更快的机器。如果你一做测试，那么就可能会要你的算法，甚至重设计，重新架构。

● **不要写测试案例。**不要做什么代码覆盖率测试，自动化测试。

● **测试是懦夫行为。**一个勇敢的程序员是根本不需要这一步的。太多的程序太害怕他们的老板，害怕去工作，害怕用户抱怨，甚至被起诉。这种担心害怕直接影响了生产力。如果你对你的代码有强大的心，那还要什么测试呢？真正的程序员是不需要测试自己的代码的。

其它

● **你的老板什么都知道。**无论你的老板有多SB，你都要严格地遵照他的旨意办事，这样一来，你会学更多的知识如何写出无法维护的代码来的。

● **颠覆Help Desk。**你要确保你那满是bug的程序永远不要被维护团队知道。当用户打电话和写邮件你的时候，你就不要理会，就算要理会，让用户重做系统或是告诉用户其帐号有问题，是标准的回答。

● **闭嘴。**对于一些像y2k这样的大bug，你要学会守口如瓶，不要告诉任何人，包括你的亲人好友以公司的同事和管理层，这样当到那一天的时候，你就可以用这个bug挣钱了。

● **忽悠。**你会学会忽悠，就算你的代码写得很烂，你也要为其挂上GoF设计模式的标签，就算你的项做得再烂，你也要为其挂上敏捷的标签，让整个团队和公司，甚至整个业界都开始躁动，这样才能真为难维护的代码铺平道路。

总之，我们的口号是——**Write Everywhere, Read Nowhere**

原文：http://mindprod.com/jgloss/unmain.html

译者：陈皓 （@左耳朵耗子）

译文：http://coolshell.cn/articles/4758.html