链滴

编写可维护的代码 (一)

作者：localvar

原文链接：https://ld246.com/article/1522145653107

来源网站：链滴

可维护性我认为主要由两个方面构成, 一是可读性, 也就是代码要能让人看懂; 二是可调试性, 出了问题以很快的找到原因. 市面上讲设计的书很多, 但大部分侧重于灵活性和可复用性, 比如面向对象设计和计模式等. 灵活和可复用并没有什么错, 但我认为可维护要更重要一些, 试想如果一个模块非常灵活并大量复用, 却不可维护, 岂不是不出问题则已, 一出就是灾难性的吗?

再看C++语言, C++是一个提供了太多特性的语言, 每一件事情都可以用好几种可选的特性去实现, 但们应该选择哪一种呢? 显然应该是最合适, 最实用的, 而不是最新最酷的.

从05年末到09年初, 我得到了一个非常难得的机会: 把一个项目做了两遍. 第一遍的时候用了很多C++高级特性, 也有意无意的引入了一个设计模式的思想, 项目也还成功, 但后续的维护却越来越难. 后来在第二遍的时候, 受《UNIX编程艺术》等的影响(也就是在这时候我成了把C++当C用的程序员), 开始学用最简单直接的方法解决问题. 而结果也相当好, 不光项目成功, 整个系统的可维护性也不错. 而且虽然计时完全没考虑面向对象, 设计模式, 但最终的系统却又带着这些东西的影子, 只是实现方法和书上写不完全一样(呵呵, 吹大了 欢迎大家对这一段扔几个西红柿鸡蛋之类的).

现在由于工作变动开始维护另一个系统, 这是一个很C++, 很面向对象, 也很设计模式的系统, 可是维护来却无比困难, 出问题后简直无从下手. 所以更体会到了可维护性的重要, 进而想到应该把自己的这点验总结一下, 写出来. 目前总共有四五篇的题材, 都是很细节的问题, 希望自己能坚持写完. 我的方法也不那么漂亮, 但应该还实用, 毕竟也算是真刀真枪的实战中总结出来的.

因为一发出来就被批了个体无完肤, 所以加上了这些文字, 说明一下背景. 不太喜欢口水仗, 后续的批评将不再回复, 因为软件开发是工程而不是艺术, 工程讲究实用, 没有绝对的对和错, 一切都应该根据实际况具体问题具体分析. 我写的东西只是供大家参考, 没有也无法强迫大家一定要用.

下面开始正题. 如果我们要实现一个类, 用于从流式缓冲区读出数据(典型应用是网络通讯中的数据包析), 你会用下面哪种实现呢(错误处理用的是异常, 与主题关系不大, 故不详述)?

```
// A实现
class CBufferReaderA
{
 ...
 template<class T>
 CBufferReaderA& operator>>( T& v );
 ...
};

// B实现
class CBufferReaderB
{
 ...
 char ReadChar();
 short ReadShort();
 int ReadInt();
 ...
};
```

我想不少人会选择A, 因为看起来更酷一些, 而且只写了一个模板函数就可以处理一大堆数据类型了, 但际上, 如果从可维护性和实用性来说, B却更好一点. 下面就来对比分析一下.

1. 像cin/cout一样, A实现能把多个操作写在一起.

```
CBufferReaderA br;
br >> a >> b >> c >> d;
```

这一点B是作不到的, 因为它的返回值被用来返回实际读取的内容了. 可是当我们调试A实现支持的那串代码时, 问题就出现了, 整个代码虽然是好几个函数调用, 但一下就执行过去了, 根本没法看到中间结(VC是这样, 其他调试器不清楚). 为了避免这个问题, 只好把这一串操作拆成单个的, 但这样一来A和B就没什么区别了.

2. 如果需要跳过一段数据, 需要怎么做呢? 如果用A实现, 肯定是类似下面的方法:

```
int tmp;
br >> tmp;
```

而B实现则可以直接:

```
br.ReadInt();
```

对比可见, A实现不光多用了一个没有什么实际用处的变量, 而且多写了一行代码. 只看这一点也许没多问题, 但如果程序很大, 类似需求很多, 它带来的混乱就不可忽视了.

3. 缓冲区中是 char型, 但我想用int保存读出的数据, 应该怎么办?

```
// A实现:
char c;
int i;
br >> c;
i = c
// B实现:
int i = br.ReadChar();
```

我想B的优势不用我说了吧.

4. 前面说到的A缺点也许还不算太严重, 下面这个应该就有足够的说服力了.

```
// A实现:
br >> a;
br >> b;
br >> c;
br >> d;
br >> e;

// B实现:
a = br.ReadChar();
b = br.ReadShort();
c = br.ReadChar();
d = br.ReadInt();
e = br.ReadInt();
```

看出问题了吗? 没错, 在B实现中, 我们很容易的知道每步操作从缓冲区中读了多少数据. 而如果用A实现, 这些信息就不那么明显, 必须去检查各个变量的定义, 也许你会说VC里面把鼠标放上去就能看到定义, 也别忘了一次只能看一个, 而B则可统观全局. 如果是个很大的程序, 那B的可读性和可调试性要高很多.

5. 对B的一个批评是暴露了实现细节, 把读了几个字节清晰的写出来了. 但我认为这恰恰是它的优点, 为只有应该隐藏的细节才需要隐藏, 而这里, 知道读几个字节对缓冲区分析来说非常重要的, 是不应该隐藏掉的. 无限制的隐藏细节只会给自己找麻烦. 打个比方, 把路的细节隐藏起来, 方法之一是把眼睛蒙, 我们又怎么走路呢?

6. B相对于A也有一个缺点, 就是A可以通过重定义 >>运算符, 让自定义类型和原生类型使用看起来

全相同的方法被读出来, 但一般来说, 这一点的艺术性远大于实用性, 而且考虑到前面所有的缺点, 它不以成为我们选择A的理由.