切换导航 Farseerfc的小窝

- 文A
  - 繁體
  - 简体
  - English
  - 日本語
  - 文A繁體
  - 文A简体
  - 文AEnglish
  - 文A日本語
-
  - About
  - Links
  - About
  - Links
-
  - Import
  - Life
  - Tech

C++ Tricks 1.2 逗号运算符(,)、逻辑运算符(&&,||)与运算符重载的陷阱

从 farseerfc.wordpress.com 导入

1.2 逗号运算符(,)、逻辑运算符(&&,||)与运算符重载的陷阱

很多人甚至不知道逗号(,)也是个C++运算符。与语法上要求出现的逗号(比如分隔函数参数的逗号)不同的是，出现在表达式中的逗号运算符在语义上表示多个表达式操作的连续执行，类似于分隔多语句的分号。比如：

```
for(int i=0,j=9;i<10;++i,--j)std::cout<<i<<"+"<<j<<"=9\n";
```

在这句语句中，出现了两个逗号，其中前者是语法上用来分隔声明的变量的，并非逗号运算符，而后者则是一个逗号运算符。根据C++标准，逗号运算符的执行顺序为从左到右依次执行，返回最后一个子表达式的结果。由于只有最后一个表达式返回结果，所以对于一个语义正常的逗号表达式而言，前几个子表达式必须具有副作用。同时，从语言的定

义中也可以看出，逗号表达式对求值的顺序有严格要求。

对求值顺序有要求的，除了逗号表达式和条件表达式(参见1.1)，在C++中还有逻辑运算符(&&和||)。逻辑运算相较于数学运算和位运算而言，有个显著的不同点：逻辑运算在计算到一半时，就有可能已经得到结果，这样继续运算另一半就不是必需的。对于A&&B，如果A=false，那么无论B为何值，整个的结果都是false；同样的A||B，如果A=true，那么不考虑B，结果一定是true。

C++标准规定，如果逻辑运算到一半(算出A)时，就已经可以确定运算的结果，那么就不运算剩下的另一半(B)。这种执行语义被称作“短路”。在其它一些编程语言中，短路语义是可以选择的：在Ada里非短路的逻辑运算符为and和or，短路的逻辑运算符为and_then和or_else。但是在C++中，逻辑运算符的短路语义是语法上强制的，我们没有非短路版本的运算符。如果确实需要非短路语义，我们总是可以通过增加一个bool中间变量加以解决。有时，短路对于保证正确执行是必须的，比如：

```
char*p=getString();
```

```
if(p&&*p)std::cout<<p;
```

这段代码在得到了一个字符串后，在字符串不为空时输出它。在C++中判断一个字符串不为空需要两个步骤：判断指针是否为0，以及指针不为0时判断指针指向的内容是否为”。就像条件表达式中讨论到的(参见1.1)，在p为空时提领p是个极其危险的操作。逻辑运算符的短路语义则避免了这种危险。

以上对逗号运算符与逻辑运算符的讨论，仅限于C++标准所定义的运算符语义。为什么这样说呢？这是因为在C++中，运算符的语义是可以由程序员自行定义的，这种机制叫做运算符重载(operator overload)。运算符重载可以将人们熟悉的运算符表达式转换成函数调用，使编程灵活而直观，是个方便的语言特性。不过有时运算符重载也会使人困扰，那就是当运算符重载遇到求值顺序问题时。

C++中，并不是所有合法运算符都可以被合法地重载。条件运算符虽然对求值顺序有要求，但它并不在可重载运算符之列，所以运算符重载机制对它没有影响。问题在于，逗号运算符和逻辑运算符都可以被合法地重载：

class BadThing{/* Some Bad and Stupid Thing*/};

BadThing& **operator**,(BadThing&, BadThing&);//重载了逗号运算符

bool operator&&(BadThing&, BadThing&);//重载了&&

BadThing b1,b2;

if(b1&&b2)b1,b2;//被替换成如下形式：

if(**operator**&&(b1,b2))**operator**,(b1,b2);

可以看到，重载了运算符之后，对运算符的使用被替换为相应的函数调用形式。因此，旧有的运算符的执行顺序不再适用，取而代之的是函数参数的压栈顺序。

根据C++标准规定，任何参数必须在进入函数之前压栈，所以在进入**operator**&&之前，b1、b2就会被求值，这里不再有短路规则，任何依赖于短路语义的不知不觉间操作BadThing的代码(可能通过模板)都会混乱。

短路语义只是一个方面，更重要的在于压栈顺序。鉴于执行效率和旧代码兼容性等细节问题，C++标准在压栈顺序上给编译器的开发者留有很大自主性。标准的说辞是，编译器可能以任何它觉得方便的顺序将参数压栈，从左到右，从右到左，甚至从中间到两边，在这一点上我们不能安全地做任何假设。在上面的例子中，编译器生成的代码可能先计算b1再计算b2，也可能是相反的顺序。再看看编译器的实际情况，在我试过的所有基于X86体系结构的编译器中，参数都是以逆向压栈，即从右到左，有悖于大多数人的阅读习惯和直觉(别说你是来自伊斯兰的……)。

在C时代使用函数调用时，压栈顺序并不是什么大问题，毕竟大多数人会在函数调用的边界稍稍小心一些。但是到了C++中，事情变得有些复杂，因为简单如a+b的使用，就有可能被运算符重载机制替换为函数调用。更何况有模板参与之后，我们写代码时不能确定对象的真实类型，也就无法预知一个运算符是否真的被重载过，唯一稳妥的方法是，假定任何有可能被重载的运算符的使用都是函数调用。

<p style="margin:0;">

回到上文的示例中，由于,和&&都被替换为函数调用，程序的执行顺序将成为压栈顺序，在X86上很有可能是从右到左，与标准定义的运算符的顺序正好相反。逗号运算符原本就含有“先…后…”的语义，这种颠倒的执行顺序势必造成程序和程序员的混乱。以我的经验而言，含有**operator**,的类，完全没有办法和STL或者iostream相互协作，反而会导致巨量的错误报告(什么叫巨量的错误报告有概念么？如果没有，那说明你还没玩过范式编程(GP, Generic Programming)。去玩玩GP吧，看看你的编译器对巨量的定义。在我手头，针对3.5KB的代码文件倾泻出3.8**MB**的错误信息的编译器不在少数……)。有鉴于此，我的结论是，除非你有充足的依据支持你这么做(比如你的粗暴上司的键盘上只剩下逗号能用)，并且你清楚的了解这么做的后果的严重性(比如至少要看过此文)，否则我奉劝你，永远不要碰**operator**,、**operator**&&以及**operator**||！