C++ Tricks 1.1 条件运算符(?:)

从 farseerfc.wordpress.com 导入

1.1 条件运算符(?:)

条件运算符(?:)是C++中唯一的三目运算符(trinary operator)，用于在表达式中作条件判断，通常可以替换if语句，与Visual Basic中的iif函数、Excel中的if函数有同样的作用。语法形式如下：

condition？*true_value*：*false_value*

其中*condition *条件是任何可以转换为bool类型的表达式，包括但不仅限于**bool*、**int**、指针。与**if**和**while**的条件部分稍显不同的是，这里不能定义变量，否则会导致语法错误。

另外，条件语句会切实地控制执行流程，而不仅仅是控制返回值。也

就是说，两个返回值表达式中永远只有一个会被求值，在表达式的执行顺序很重要时，这点尤为值得注意。比如：

```
int *pi=getInt();
int i=pi?*pi:0;
```

这里，只有当pi的值不为0时，它才会被提领(dereference)。这种语义保证了程序的正确性，因为提领一个空指针将导致致命的运行期错误(通常是非法操作的警告)。同时，正因为条件运算符控制运算流程的特点，使得它不能用类似iif的普通函数来模拟：

```
int iif(int con,int t,intf){if(c)return t;return f;}//试图模拟?:
…//in some function
int *pi=getInt();
int i=iif(pi,*pi,0);//Error!
```

这段代码会导致上文提到的致命运行期错误。C/C++标准规定，参数在被传递给函数之前求值，因此无论pi为何值，都会被提领。又因为函数传回一个空指针的情况比较少见，所以这样的错误在调试时很难被发现，一旦发生又势必造成重大灾难。这样的代码在实践中应尽量避免。

有时，条件运算符控制流程的特点会不知不觉影响我们的代码。在C时代，最大值MAX通常用宏实现：

```
#defineMAX(a,b) ((a)>(b)?(a):(b))
```

需要用额外的括号将宏参数和宏本体保护起来，以免运算符优先级扰乱逻辑，这是宏丑陋的特点之一，这里暂且不提。矛盾在于，用具有副作用的表达式调用宏时，会出现问题：

```
int i=5,j=6;//…
int a=MAX(++i,++j);
```

代码的作者原意显然是想先将i,j分别递增，再将其中较大的一个赋给a。执行这段代码，当i=5,j=6时，a=8，知道为什么吗？通过宏展开，赋值语句成这样：

```
int a=(++i)>(++j)?(++i):(++j);//删除了多余括号
```

在判断之前，i、j被分别自增一次，然后舍弃:之前的部分，j又被自增一次。执行之后，i=6,j=8。

MAX的更正确更安全的实现，是利用模板将类型参数化。STL标准算法中就有一个这样的工具级模版函数std::max。

条件运算符是表达式而不是语句，这使得它可以出现在任何需要表达式的地方，这扩大了它的适用范围。在那些语法上只能出现表达式而不能出现语句的地方（比如变量初始化），条件运算符有着不可替代的作用。

条件运算符优于**if**语句的另一个场合是“模板元编程”(TMP, Template MetaProgramming)。在TMP这个古怪奇异的编译期运算编程技术中，一切旧有的技术和法则被全线击破，我们所能仰仗的工具，只有模板特化(Specialization)、**typedef**s、函数声明(无法调用它们)、以及编译期常量运算。已经有人很深入地论证过，仅有以上这些，就已经形成了一个“图灵完善”的计算机语言。我们可以用模板特化技术，来模拟条件分支，循环迭代等一系列复杂的语言结构。由于可以参与编译期常量运算，条件运算符在TMP世界中很自然地扮演起重要角色。

比如，给与类型T的一个变量t，我们想声明一个缓冲区存放t和一个int，缓冲区的大小不小于sizeof(T)也不小于sizeif(int)，我们可以这样写：

```
char buffer[sizeof(T)>sizeof(int)? sizeof(T): sizeof(int)];
```

我们不能用一个if语句替换这个运算：

```
int i;
if(sizeof(T)>sizeof(int))i=sizeof(T);
else i=sizeof(int);
char buffer[i];//语法错误!
```

原因在于数组声明中的下标必须是一个编译期常量，而不是一个运行期的值，条件表达式的运算可以在编译期进行，if语句就只能在执行期执行。