切换导航 Farseerfc的小窝

文

繁體

<u>简体</u>

English

日本語

- <u>▲ About</u>
- <u> Links</u>
- ≜ About
- <u>Links</u>
- □ Import
- □ Life
- <u>□ Tech</u>

- □ Import
- □<u>Life</u>
- <u>Tech</u>
- Q

搜索

- 搜索
- *y*

C++ Tricks 3.1 左值右值与常量性(Ivalue, rvalue & constant) <u>辽</u>

3.1 左值右值与常量性(lvalue, rvalue & constant)

首先要搞清楚的是,什么是左值,什么是右值。这 里给出左值右值的定义:

- 1、左值是可以出现在等号(=)左边的值,右值是只能 出现在等号右边的值。
- 2、左值是可读可写的值,右值是只读的值。
- 3、左值有地址,右值没有地址。

根据左值右值的第二定义,值的左右性就是值的常量性——常量是右值,非常量是左值。比如:

1=1;//Error

这个复制操作在C++中是语法错误,MSVC给出的错误提示为"error C2106: '=': left operand must be l-value",就是说'='的左操作数必须是一个左值,

而字面常数1是一个右值。可见,严格的区分左值右 值可以从语法分析的角度找出程序的逻辑错误。

根据第二定义,一个左值也是一个右值,因为左值 也可读,而一个右值不是一个左值,因为右值不可 写。

通常情况下,声明的变量是一个左值,除非你指定 const将它变成一个右值:

int lv=1;

const int rv=lv;

由于右值的值在程序执行期间不能改变,所以必须用另一个右值初始化它。

一个普通变量只能用右值初始化,如果你想传递左 值,必须声明一个引用或一个指针:

int & ref=lv;//用引用传递左值

int * plv=&lv;//传递指针以间接传递左值

必须用左值初始化引用,然而,可以用右值初始化 常量引用: int & r1=1; //Error!

const int & r2=1; //OK

这实际上相当于:

int _r2=1;

const int & $r2=_r2$;

这样的写法在函数体内没什么作用,但是在传递函数参数时,它可以避免潜在的(传递左值时的)复制操作,同时又可以接受右值。

通常情况下,函数的参数和返回值都只传回右值, 除非你明确的通过引用传递左值。

明确了左值与右值的区别,有助于我们写函数时确 定什么时候应该有const,什么时候不该有。比如, 我们写了一个代表数学中复数的类Complex:

class Complex;

然后,我们写针对Complex的运算符重载: operator+和operator=。问题在于,参数和返回值 应该是什么类型,可选类型有四种: Complex、 const Complex、Complex&、const Complex&。

对于operator+,我们不会改变参数的值,所以可以通过const Complex&传递参数。至于返回值类型,由于int类型的加法返回右值,所以根据Do as the ints do的原则,返回值类型为const Complex:

const Complex operator+(const Complex&,const
Complex&);

对于operator=,同样要思考这些问题。我们写入第一个参数,所以第一个参数为Complex&,我们只读取第二个参数,所以第二个参数为const Complex&。至于返回值,还是Do as the ints do。int的赋值返回左值,不信你可以试一试:

int i;

$$(i=1)=2;$$

虽然比较傻,先将i赋为1,再将其改为2,但是这是被C++语法支持的做法,我们就理应遵守。所以返回第一个参数的左值:

Complex& operator=(Complex&,const

Complex&);

const是C++引入的语言特性、也被ANSI C99借鉴、 在经典版本的C语言中是没有的。关于const的历 史,有几点值得玩味。最初Bjarne Stroustrup引入 const时,可写性是和可读性分开的。那时使用关键 字readonly和writeonly。这个特点被首先提交到C 的ANSI标准化委员会(当时还没有C++标准化的计 划),但是ANSI C标准只接受了readonly的概念,并 将其命名为const。随后,有人发现在多线程同步的 环境下,有些变量的值会在编译器的预料之外改 变,为了防止过度优化破坏这些变量,C++又引入关 键字violate。从语义特点来看,violate是const的 反义词,因为const表示不会变的量,而violate表示 会不按照预期自行变化的量。从语法特点而言, violate与const是极为相似的,适用于const的一切 语法规则同样适用于violate。

值的常量性可以被划分为两种:编译期常量和运行期常量。C++语法并没有严格区分这两种常量,导致了少许混乱:

const int i=5;const int * pi=&i;

const_cast<int&>i=1;//对于运行期常量,在需要时可以去除它的常量性

int a[i];//对于编译期常量,可以用它来指定数组大 小

cout<<i<<sizeof(a)/sizeof(a[0])<<*pi;</pre>

这种将编译期与运行期常量的特性混用的方法,势必导致语义的混乱。数组a的大小最终是5,因为采用了i的编译期值,而不管i在运行期是否被改变了值。最后一句代码将(有可能)输出551,第一个i的值作为一种优化在编译期绑定,第二个值标明了a的大小,第三个值通过指针显示地输出i的运行期真实值。

在C++的近亲C#的语法中,这两种常量被严格地区分开:编译期常量由const指定,只能是内建类型变量;运行期常量由readonly指定,可以是任何类型。永远不会改变的常量,如圆周率pi的值,应该用const声明;而其它有可能改变的常量,皆由readonly声明。

C++中的const的特点更倾向于C#中的readonly,虽

然语法上允许使用const的编译期常量性,但正如上 文所展示的,这容易造成混乱。为了得到C#中const 的语义,在C++中,我们不必回归恶魔#define的怀 抱,可以使用所谓"匿名enum技巧"。当匿名声明一 个enum类型时,其中的枚举值就是一个int类型的 编译期常量,比如:

enum{Size=5;};

int a[Size];

这种使用匿名enum来声明编译期常量的做法,被广 泛应用于STL、boost等模板库的实现代码中。