C/C++中static关键字作用总结

1.先来介绍它的第一条也是最重要的一条：隐藏。（static函数，static变量均可）

当同时编译多个文件时，所有未加static前缀的全局变量和函数都具有全局可见性。

举例来说明。同时编译两个源文件，一个是a.c，另一个是main.c。

//a.c

char a = 'A'; // global variable

void msg()

{

printf("Hello\n");

}

复制代码

//main.c

int main()

{

extern char a; // extern variable must be declared before use

printf("%c ", a);

(void)msg();

return 0;

}

复制代码

程序的运行结果是：

A Hello

为什么在a.c中定义的全局变量a和函数msg能在main.c中使用？前面说过，所有未加static前缀的全局变量和函数都具有全局可见性，其它的源文件也能访问。此例中，a是全局变量，msg是函数，并且都没有加static前缀，因此对于另外的源文件main.c是可见的。

如果加了static，就会对其它源文件隐藏。例如在a和msg的定义前加上static，main.c就看不到它们了。利用这一特性可以在不同的文件中定义同名函数和同名变量，而不必担心命名冲突。static可以用作函数和变量的前缀，对于函数来讲，static的作用仅限于隐藏.

2.static的第二个作用是保持变量内容的持久。（static变量中的记忆功能和全局生存期）

存储在静态数据区的变量会在程序刚开始运行时就完成初始化，也是唯一的一次初始化。共有两种变量存储在静态存储区：全局变量和static变量，只不过和全局变量比起来，static可以控制变量的可见范围，说到底static还是用来隐藏的。虽然这种用法不常见

PS：如果作为static局部变量在函数内定义，它的生存期为整个源程序，但是其作用域仍与自动变量相同，只能在定义该变量的函数内使用该变量。退出该函数后， 尽管该变量还继续存在，但不能使用它。

程序举例：

复制代码

＃include <stdio.h>

int fun(){

static int count = 10; //在第一次进入这个函数的时候，变量a被初始化为10！并接着自减1，以后每次进入该函数，a

return count--; //就不会被再次初始化了，仅进行自减1的操作；在static发明前，要达到同样的功能，则只能使用全局变量：

}

int count = 1;

int main(void)

{

printf("global\t\tlocal static\n");

for(; count <= 10; ++count)

printf("%d\t\t%d\n", count, fun());

return 0;

}

复制代码

程序的运行结果是：

global local static

1 10

2 9

3 8

4 7

5 6

6 5

7 4

8 3

9 2

10 1

---基于以上两点可以得出一个结论：把局部变量改变为静态变量后是改变了它的存储方式即改变了它的生存期。把全局变量改变为静态变量后是改变了它的作用域， 限制了它的使用范围。因此static 这个说明符在不同的地方所起的作用是不同的。

3.static的第三个作用是默认初始化为0（static变量）

其实全局变量也具备这一属性，因为全局变量也存储在静态数据区。在静态数据区，内存中所有的字节默认值都是0x00，某些时候这一特点可以减少程序员的工作量。比如初始化一个稀疏矩阵，我们可以一个一个地把所有元素都置0，然后把不是0的几个元素赋值。如果定义成静态的，就省去了一开始置0的操作。再比如要把一个字符数组当字符串来用，但又觉得每次在字符数组末尾加‘\0’;太麻烦。如果把字符串定义成静态的，就省去了这个麻烦，因为那里本来就是‘\0’;不妨做个小实验验证一下。

复制代码

＃include <stdio.h>

int a;

int main()

{

int i;

static char str[10];

printf("integer: %d; string: (begin)%s(end)", a, str);

return 0;

}

复制代码

程序的运行结果是：

integer: 0; string: (begin) (end)

最后对static的三条作用做一句话总结。首先static的最主要功能是隐藏，其次因为static变量存放在静态存储区，所以它具备持久性和默认值0.

4.static的第四个作用：C++中的类成员声明static（有些地方与以上作用重叠）

在类中声明static变量或者函数时，初始化时使用作用域运算符来标明它所属类，因此，静态数据成员是类的成员，而不是对象的成员，这样就出现以下作用：

(1)类的静态成员函数是属于整个类而非类的对象，所以它没有this指针，这就导致 了它仅能访问类的静态数据和静态成员函数。

(2)不能将静态成员函数定义为虚函数。

(3)由于静态成员声明于类中，操作于其外，所以对其取地址操作，就多少有些特殊 ，变量地址是指向其数据类型的指针 ，函数地址类型是一个“nonmember函数指针”。

(4)由于静态成员函数没有this指针，所以就差不多等同于nonmember函数，结果就 产生了一个意想不到的好处：成为一个callback函数，使得我们得以将C++和C-based X W indow系统结合，同时也成功的应用于线程函数身上。 （这条没遇见过）

(5)static并没有增加程序的时空开销，相反她还缩短了子类对父类静态成员的访问 时间，节省了子类的内存空间。

(6)静态数据成员在<定义或说明>时前面加关键字static。

(7)静态数据成员是静态存储的，所以必须对它进行初始化。 （程序员手动初始化，否则编译时一般不会报错，但是在Link时会报错误）

(8)静态成员初始化与一般数据成员初始化不同:

初始化在类体外进行，而前面不加static，以免与一般静态变量或对象相混淆；

初始化时不加该成员的访问权限控制符private，public等；

初始化时使用作用域运算符来标明它所属类；

所以我们得出静态数据成员初始化的格式：

<数据类型><类名>::<静态数据成员名>=<值>

(9)为了防止父类的影响，可以在子类定义一个与父类相同的静态变量，以屏蔽父类的影响。这里有一点需要注意：我们说静态成员为父类和子类共享，但我们有重复定义了静态成员，这会不会引起错误呢？不会，我们的编译器采用了一种绝妙的手法：name-mangling 用以生成唯一的标志