# “Static”详解

|  |  |  |
| --- | --- | --- |
| 对象 | 例子 | 作用 |
| 对函数 | Static void function(void) | 只对本源文件作用。其它源文件不可见 |
| 对全局变量 | Static int a | 只对本源文件作用。其它源文件不可见。 |
| 对局部变量 | Void function(void)  {  Static int a;  ….  } | 1. 只对本函数作用。其它不可见。 2. 存入静态变量存储区，只被初始化一次，下次读取时保持上一次的值。 |

C语言代码是以文件为单位来组织的，在一个源程序的所有源文件中，一个外部变量（注意不是局部变量）或者函数只能在一个源程序中定义一次，如果有重复定义的话编译器就会报错。伴随着不同源文件变量和函数之间的相互引用以及相互独立的关系，产生了extern和static关键字。

        下面，详细分析一下static关键字在编写程序时有的三大类用法：

        一，static全局变量

           我们知道，一个进程在内存中的布局如图1所示：



      其中.text段保存进程所执行的程序二进制文件，.data段保存进程所有的已初始化的全局变量，.bss段保存进程未初始化的全局变量（其他段中还有很多乱七八糟的段，暂且不表）。在进程的整个生命周期中，.data段和.bss段内的数据时跟整个进程同生共死的，也就是在进程结束之后这些数据才会寿终就寝。

     当一个进程的全局变量被声明为static之后，它的中文名叫**静态全局变量**。静态全局变量和其他的全局变量的存储地点并没有区别，都是在.data段（已初始化）或者.bss段（未初始化）内，但是**它只在定义它的源文件内有效，其他源文件无法访问它**。所以，普通全局变量穿上static外衣后，它就变成了新娘，已心有所属，只能被定义它的源文件（新郎）中的变量或函数访问。

以下是一些示例程序

file1.h如下：

**[cpp]** [view plain](http://blog.csdn.net/keyeagle/article/details/6708077/) [copy](http://blog.csdn.net/keyeagle/article/details/6708077/)

1. #include <stdio.h>
3. **void** printStr();

我们在file1.c中定义一个静态全局变量hello, 供file1.c中的函数printStr访问.

**[cpp]** [view plain](http://blog.csdn.net/keyeagle/article/details/6708077/) [copy](http://blog.csdn.net/keyeagle/article/details/6708077/)

1. #include "file1.h"
3. **static** **char**\* hello = "hello cobing!";
5. **void** printStr()
6. {
7. printf("%s\n", hello);
8. }

file2.c是我们的主程序所在文件，file2.c中如果引用hello会编译出错

**[cpp]** [view plain](http://blog.csdn.net/keyeagle/article/details/6708077/) [copy](http://blog.csdn.net/keyeagle/article/details/6708077/)

1. #include "file1.h"
3. **int** main()
4. {
5. printStr();
6. printf("%s\n", hello);
7. **return** 0;
8. }

报错如下：

[liujx@server235 static]$ gcc -Wall file2.c file1.c -o file2  
file2.c: In function ‘main’:  
file2.c:6: 错误：‘hello’ 未声明 (在此函数内第一次使用)  
file2.c:6: 错误：(即使在一个函数内多次出现，每个未声明的标识符在其  
file2.c:6: 错误：所在的函数内只报告一次。)  
  
如果我们将file2.c改为下面的形式：

**[cpp]** [view plain](http://blog.csdn.net/keyeagle/article/details/6708077/) [copy](http://blog.csdn.net/keyeagle/article/details/6708077/)

1. #include "file1.h"
3. **int** main()
4. {
5. printStr();
6. **return** 0;
7. }

则会顺利编译连接。

运行程序后的结果如下：  
[liujx@server235 static]$ gcc -Wall file2.c file1.c -o file2  
[liujx@server235 static]$ ./file2  
hello cobing!  
  
上面的例子中，file1.c中的hello就是一个静态全局变量，它可以被同一文件中的printStr调用，但是不能被不同源文件中的file2.c调用。

      二，static局部变量

      普通的局部变量在栈空间上分配，这个局部变量所在的函数被多次调用时，每次调用这个局部变量在栈上的位置都不一定相同。局部变量也可以在堆上动态分配，但是记得使用完这个堆空间后要释放之。

       static局部变量中文名叫**静态局部变量**。它与普通的局部变量比起来有如下几个区别：

           1）位置：静态局部变量被编译器放在全局存储区.data（注意：不在.bss段内，原因见3）），所以它虽然是局部的，但是在程序的整个生命周期中存在。

           2）访问权限：静态局部变量只能被其作用域内的变量或函数访问。也就是说虽然它会在程序的整个生命周期中存在，由于它是static的，它不能被其他的函数和源文件访问。

           3）值：静态局部变量如果没有被用户初始化，则会被编译器自动赋值为0，以后每次调用静态局部变量的时候都用上次调用后的值。这个比较好理解，每次函数调用静态局部变量的时候都修改它然后离开，下次读的时候从全局存储区读出的静态局部变量就是上次修改后的值。  
以下是一些示例程序：

     file1.h的内容和上例中的相同，file1.c的内容如下：

**[cpp]** [view plain](http://blog.csdn.net/keyeagle/article/details/6708077/) [copy](http://blog.csdn.net/keyeagle/article/details/6708077/)

1. #include "file1.h"
3. **void** printStr()
4. {
5. **int** normal = 0;
6. **static** **int** stat = 0;    //this is a static local var
7. printf("normal = %d ---- stat = %d\n",normal, stat);
8. normal++;
9. stat++;
10. }

为了便于比较，我定义了两个变量：普通局部变量normal和静态局部变量stat，它们都被赋予初值0；

file2.c中调用file1.h：

**[cpp]** [view plain](http://blog.csdn.net/keyeagle/article/details/6708077/) [copy](http://blog.csdn.net/keyeagle/article/details/6708077/)

1. #include "file1.h"
3. **int** main()
4. {
5. printStr();
6. printStr();
7. printStr();
8. printStr();
9. printf("call stat in main: %d\n",stat);
10. **return** 0;
11. }

这个调用会报错，因为file2.c中引用了file1.c中的静态局部变量stat，如下：

[liujx@server235 static]$ gcc -Wall file2.c file1.c -o file2  
file2.c: In function ‘main’:  
file2.c:9: 错误：‘stat’ 未声明 (在此函数内第一次使用)  
file2.c:9: 错误：(即使在一个函数内多次出现，每个未声明的标识符在其  
file2.c:9: 错误：所在的函数内只报告一次。)

编译器说stat未声明，这是因为它看不到file1.c中的stat，下面注掉这一行：

**[cpp]** [view plain](http://blog.csdn.net/keyeagle/article/details/6708077/) [copy](http://blog.csdn.net/keyeagle/article/details/6708077/)

1. #include "file1.h"
3. **int** main()
4. {
5. printStr();
6. printStr();
7. printStr();
8. printStr();
9. //  printf("call stat in main: %d\n",stat);
10. **return** 0;
11. }

[liujx@server235 static]$ gcc -Wall file2.c file1.c -o file2  
[liujx@server235 static]$ ./file2  
normal = 0 ---- stat = 0  
normal = 0 ---- stat = 1  
normal = 0 ---- stat = 2  
normal = 0 ---- stat = 3  
运行如上所示。可以看出，函数每次被调用，普通局部变量都是重新分配，而静态局部变量保持上次调用的值不变。

需要注意的是由于static局部变量的这种特性，使得含静态局部变量的函数变得不可重入，即每次调用可能会产生不同的结果。这在多线程编程时可能会成为一种隐患。需要多加注意。

       三，static函数  
              相信大家还记得C++面向对象编程中的private函数，私有函数只有该类的成员变量或成员函数可以访问。在C语言中，也有“private函数”，它就是接下来要说的static函数，完成面向对象编程中private函数的功能。

            当你的程序中有很多个源文件的时候，你肯定会让某个源文件只提供一些外界需要的接口，其他的函数可能是为了实现这些接口而编写，这些其他的函数你可能并不希望被外界（非本源文件）所看到，这时候就可以用static修饰这些“其他的函数”。

**所以static函数的作用域是本源文件，把它想象为面向对象中的private函数就可以了。**

下面是一些示例：

file1.h如下：

**[cpp]** [view plain](http://blog.csdn.net/keyeagle/article/details/6708077/) [copy](http://blog.csdn.net/keyeagle/article/details/6708077/)

1. #include <stdio.h>
3. **static** **int** called();
4. **void** printStr();

file1.c如下：

**[cpp]** [view plain](http://blog.csdn.net/keyeagle/article/details/6708077/) [copy](http://blog.csdn.net/keyeagle/article/details/6708077/)

1. #include "file1.h"
3. **static** **int** called()
4. {
5. **return** 6;
6. }
7. **void** printStr()
8. {
9. **int** returnVal;
10. returnVal = called();
11. printf("returnVal=%d\n",returnVal);
12. }

file2.c中调用file1.h中声明的两个函数，此处我们故意调用called():

**[cpp]** [view plain](http://blog.csdn.net/keyeagle/article/details/6708077/) [copy](http://blog.csdn.net/keyeagle/article/details/6708077/)

1. #include "file1.h"
3. **int** main()
4. {
5. **int** val;
6. val = called();
7. printStr();
8. **return** 0;
9. }

编译时会报错：

[liujx@server235 static]$ gcc -Wall file2.c file1.c -o file2  
file1.h:3: 警告：‘called’ 使用过但从未定义  
/tmp/ccyLuBZU.o: In function `main':  
file2.c:(.text+0x12): undefined reference to `called'  
collect2: ld 返回 1  
因为引用了file1.h中的static函数，所以file2.c中提示找不到这个函数:undefined reference to 'called'

下面修改file2.c:

**[cpp]** [view plain](http://blog.csdn.net/keyeagle/article/details/6708077/) [copy](http://blog.csdn.net/keyeagle/article/details/6708077/)

1. #include "file1.h"
3. **int** main()
4. {
5. printStr();
6. **return** 0;
7. }

编译运行：

[liujx@server235 static]$ gcc -Wall file2.c file1.c -o file2  
[liujx@server235 static]$ ./file2  
returnVal=6

       static函数可以很好地解决不同原文件中函数同名的问题，因为一个源文件对于其他源文件中的static函数是不可见的。

# 善于使用移位符号“<<”“>>”

例如，“发送事件标志”函数原型为：

Void isr\_evt\_set(u16 event\_flags,OS\_TID task)

参数event\_flags是16位，每一位代表一个事件标志。所以为了方便，可以如下操作：

#define bit\_0 (1<<0)

#define bit\_3 (1<<3)

…

isr\_evt\_set(bit\_0,…);

isr\_evt\_set(bit\_3,…);

# Mesmet

在<sting.h>中声明。

void \*memset(void \*s, int ch, [size\_t](http://baike.baidu.com/item/size_t" \t "_blank) n);

函数解释：将s中当前位置后面的n个字节 （typedef unsigned int size\_t ）用 ch 替换并返回 s 。

memset：作用是在一段内存块中填充某个给定的值，它是对较大的[结构体](http://baike.baidu.com/item/%E7%BB%93%E6%9E%84%E4%BD%93" \t "_blank)或[数组](http://baike.baidu.com/item/%E6%95%B0%E7%BB%84)进行清零操作的一种最快方法。

例如：

Char buffer[4];

Memset(buffer,0,sizeof(buffer));

将buffer[]清0。