**C 语言的 static 关键字有三种（具体来说是两种）用途：**

**1. 静态局部变量：**用于函数体内部修饰变量，这种变量的生存期长于该函数。

**[cpp]** [view plain](https://blog.csdn.net/majianfei1023/article/details/45290467) [copy](https://blog.csdn.net/majianfei1023/article/details/45290467)

1. **int** foo(){
2. **static** **int** i = 1; // note:1
3. //int i = 1;  // note:2
4. i += 1;
5. **return** i;
6. }

要明白这个用法，我们首先要了解c/c++的内存分布，以及static所在的区间。  
  
对于一个完整的程序，在内存中的分布情况如下图：　   
1.栈区： 由编译器自动分配释放，像局部变量，函数参数，都是在栈区。会随着作用于退出而释放空间。  
3.堆区：程序员分配并释放的区域，像malloc(c),new(c++)   
3.全局数据区(静态区)：全局变量和静态便令的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。程序结束释放。  
4.代码区  
  
所以上面note:1的static是在全局数据区分配的,那么它存在的意思是什么？又是什么时候初始化的呢？  
  
首先回答第一个问题：它存在的意义就是随着第一次函数的调用而初始化，却不随着函数的调用结束而销毁(如果把以上的note:1换成note:2,那么i就是在栈区分配了，会随着foo的调用结束而释放)。  
那么第二个问题也就浮出水面了，它是在第一次调用进入note:1的时候初始化（当初面试被坑过，我居然说是一开始就初始化了，汗！！）。且只初始化一次，也就是你第二次调用foo(),不会继续初始化，而会直接跳过。  
  
  
那么它跟定义一个全局变量有什么区别呢，同样是初始化一次，连续调用foo()的结果是一样的，但是，使用全局变量的话，变量就不属于函数本身了，不再仅受函数的控制，给程序的维护带来不便。  
　　静态局部变量正好可以解决这个问题。静态局部变量保存在全局数据区，而不是保存在栈中，每次的值保持到下一次调用，直到下次赋新值。  
  
  
那么我们总结一下，静态局部变量的特点（括号内为note:2,也就是局部变量的对比）：  
（1）该变量在全局数据区分配内存(局部变量在栈区分配内存);  
（2）静态局部变量在程序执行到该对象的声明处时被首次初始化，即以后的函数调用不再进行初始化(局部变量每次函数调用都会被初始化);  
（3）静态局部变量一般在声明处初始化，如果没有显式初始化，会被程序自动初始化为0(局部变量不会被初始化);  
（4）它始终驻留在全局数据区，直到程序运行结束。但其作用域为局部作用域，也就是不能在函数体外面使用它(局部变量在栈区，在函数结束后立即释放内存);  
  
  
**2.静态全局变量：**定义在函数体外，用于修饰全局变量，表示该变量只在本文件可见。

**[cpp]** [view plain](https://blog.csdn.net/majianfei1023/article/details/45290467) [copy](https://blog.csdn.net/majianfei1023/article/details/45290467)

1. **static** **int** i = 1;  //note:3
2. //int i = 1;  //note:4

5. **int** foo()
6. {
7. i += 1;
8. **return** i;
9. }

note:3和note:4有什么差异呢？你调用foo(),无论调用几次，他们的结果都是一样的。也就是说在本文件内调用他们是完全相同的。那么他们的区别是什么呢？  
文件隔离！

假设我有一个文件a.c,我们再新建一个b.c,内容如下。

**[cpp]** [view plain](https://blog.csdn.net/majianfei1023/article/details/45290467) [copy](https://blog.csdn.net/majianfei1023/article/details/45290467)

1. //file a.c
3. //static int n = 15;  //note:5
4. **int** n = 15;  //note:6
6. //file b.c
7. #include <stdio.h>
9. **extern** **int** n;
11. **void** fn()
12. {
13. n++;
14. printf("after: %d\n",n);
15. }

18. **void** main()
19. {
20. printf("before: %d\n",n);
21. fn();
22. }

我们先使用note:6,也就是非静态全局变量，发现输出为:  
before: 15  
after: 16  
  
  
也就是我们的b.c通过extern使用了a.c定义的全局变量。  
那么我们改成使用note:5,也就是使用静态全局变量呢？

*gcc a.c b.c -o output.out*

会出现类似undeference to "n"的报错，它是找不到n的，因为static进行了文件隔离，你是没办法访问a.c定义的静态全局变量的，当然你用 #include "a.c",那就不一样了。

以上我们就可以得出静态全局变量的特点：

静态全局变量不能被其它文件所用(全局变量可以);  
其它文件中可以定义相同名字的变量，不会发生冲突(自然了，因为static隔离了文件，其它文件使用相同的名字的变量，也跟它没关系了);  
  
  
**3.静态函数：**准确的说，静态函数跟静态全局变量的作用类似：

**[cpp]** [view plain](https://blog.csdn.net/majianfei1023/article/details/45290467) [copy](https://blog.csdn.net/majianfei1023/article/details/45290467)

1. //file a.c
2. #include <stdio.h>

5. **void** fn()
6. {
7. printf("this is non-static func in a");
8. }

11. //file b.c
12. #include <stdio.h>

15. **extern** **void** fn();  //我们用extern声明其他文件的fn(),供本文件使用。

18. **void** main()
19. {
20. fn();
21. }

可以正常输出：this is non-static func in a。  
当给void fn()加上static的关键字之后呢？ undefined reference to "fn".  
  
所以，静态函数的好处跟静态全局变量的好处就类似了：  
1.静态函数不能被其它文件所用;  
2.其它文件中可以定义相同名字的函数，不会发生冲突;  
  
上面一共说了三种用法，为什么说准确来说是两种呢？  
1.一种是修饰变量，一种是修饰函数，所以说是两种（这种解释不多）。  
2.静态全局变量和修饰静态函数的作用是一样的，一般合并为一种。（这是比较多的分法）。  
  
  
  
**C++ 语言的 static 关键字有二种用途：**  
当然以上的几种，也可以用在c++中。还有额外的两种用法：  
  
**1.静态数据成员**：用于修饰 class 的数据成员，即所谓“静态成员”。这种数据成员的生存期大于 class 的对象（实体 instance）。静态数据成员是每个 class 有一份，普通数据成员是每个 instance 有一份，因此静态数据成员也叫做类变量，而普通数据成员也叫做实例变量。

**[cpp]** [view plain](https://blog.csdn.net/majianfei1023/article/details/45290467) [copy](https://blog.csdn.net/majianfei1023/article/details/45290467)

1. #include<iostream>

4. **using** **namespace** std;

7. **class** Rectangle
8. {
9. **private**:
10. **int** m\_w,m\_h;
11. **static** **int** s\_sum;
13. **public**:
14. Rectangle(**int** w,**int** h)
15. {
16. **this**->m\_w = w;
17. **this**->m\_h = h;
18. s\_sum += (**this**->m\_w \* **this**->m\_h);
19. }

22. **void** GetSum()
23. {
24. cout<<"sum = "<<s\_sum<<endl;
25. }

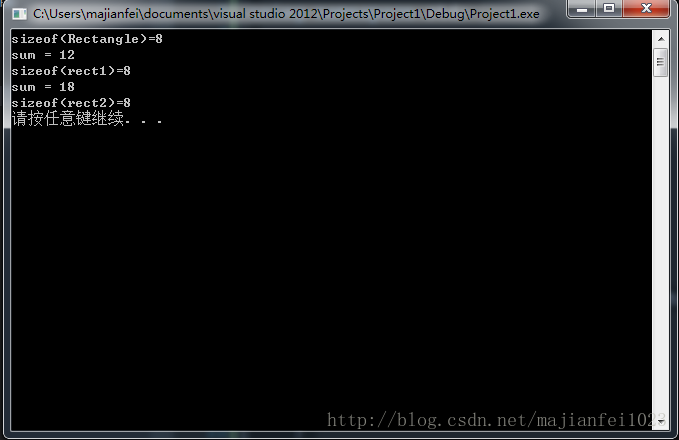
28. };

31. **int** Rectangle::s\_sum = 0;  //初始化



36. **int** main()
37. {
38. cout<<"sizeof(Rectangle)="<<**sizeof**(Rectangle)<<endl;
39. Rectangle \*rect1 = **new** Rectangle(3,4);
40. rect1->GetSum();
41. cout<<"sizeof(rect1)="<<**sizeof**(\*rect1)<<endl;
42. Rectangle rect2(2,3);
43. rect2.GetSum();
44. cout<<"sizeof(rect2)="<<**sizeof**(rect2)<<endl;
46. system("pause");
47. **return** 0;
48. }

结果如下：



由图可知：sizeof(Rectangle)=8bytes=sizeof(m\_w)+sizeof(m\_h)。也就是说 static 并不占用Rectangle的内存空间。  
那么static在哪里分配内存的呢？是的，全局数据区(静态区)。  
再看看GetSum()，第一次12=3\*4，第二次18=12+2\*3。由此可得，static只会被初始化一次，于实例无关。

结论：

对于非静态数据成员，每个类对象(实例)都有自己的拷贝。而静态数据成员被当作是类的成员，由该类型的所有对象共享访问,对该类的多个对象来说，静态数据成员只分配一次内存。  
静态数据成员存储在全局数据区。静态数据成员定义时要分配空间，所以不能在类声明中定义。  
  
也就是说，你每new一个Rectangle，并不会为static int s\_sum的构建一份内存拷贝，它是不管你new了多少Rectangle的实例，因为它只与类Rectangle挂钩，而跟你每一个Rectangle的对象没关系。

**2、静态成员函数：**用于修饰 class 的成员函数。  
我们对上面的例子稍加改动：

**[cpp]** [view plain](https://blog.csdn.net/majianfei1023/article/details/45290467) [copy](https://blog.csdn.net/majianfei1023/article/details/45290467)

1. #include<iostream>

4. **using** **namespace** std;

7. **class** Rectangle
8. {
9. **private**:
10. **int** m\_w,m\_h;
11. **static** **int** s\_sum;
13. **public**:
14. Rectangle(**int** w,**int** h)
15. {
16. **this**->m\_w = w;
17. **this**->m\_h = h;
18. s\_sum += (**this**->m\_w \* **this**->m\_h);
19. }

22. **static** **void** GetSum()  //这里加上static
23. {
24. cout<<"sum = "<<s\_sum<<endl;
25. }

28. };

31. **int** Rectangle::s\_sum = 0;  //初始化



36. **int** main()
37. {
38. cout<<"sizeof(Rectangle)="<<**sizeof**(Rectangle)<<endl;
39. Rectangle \*rect1 = **new** Rectangle(3,4);
40. rect1->GetSum();
41. cout<<"sizeof(rect1)="<<**sizeof**(\*rect1)<<endl;
42. Rectangle rect2(2,3);
43. rect2.GetSum();  //可以用对象名.函数名访问
44. cout<<"sizeof(rect2)="<<**sizeof**(rect2)<<endl;
45. Rectangle::GetSum();  //也可以可以用类名::函数名访问

48. system("pause");
49. **return** 0;
50. }

上面注释可见:对GetSum()加上static，使它变成一个静态成员函数，可以用类名::函数名进行访问。  
那么静态成员函数有特点呢？  
1.静态成员之间可以相互访问，包括静态成员函数访问静态数据成员和访问静态成员函数;  
2.非静态成员函数可以任意地访问静态成员函数和静态数据成员;  
3.静态成员函数不能访问非静态成员函数和非静态数据成员;  
4.调用静态成员函数，可以用成员访问操作符(.)和(->)为一个类的对象或指向类对象的指针调用静态成员函数,也可以用类名::函数名调用(因为他本来就是属于类的，用类名调用很正常)  
  
  
前三点其实是一点：静态成员函数不能访问非静态(包括成员函数和数据成员)，但是非静态可以访问静态，有点晕吗？没关系，我给你个解释，  
因为静态是属于类的，它是不知道你创建了10个还是100个对象，所以它对你对象的函数或者数据是一无所知的，所以它没办法调用，而反过来，你创建的对象是对类一清二楚的(不然你怎么从它那里实例化呢)，所以你是可以调用类函数和类成员的，就像不管GetSum是不是static，都可以调用static的s\_sum一样。

https://blog.csdn.net/majianfei1023/article/details/45290467