# [extern用法总结](https://www.cnblogs.com/Lunais/p/5991135.html)

在C语言中，修饰符extern用在变量或者函数的声明前，用来说明“此变量/函数是在别处定义的，要在此处引用”。  
  
**1. extern修饰变量的声明。**

如果文件a.c需要引用b.c中变量int v，就可以在a.c中声明extern int v，然后就可以引用变量v。

这里需要注意的是，被引用的变量v的**链接属性**必须是**外链接（external）的**，也就是说a.c要引用到v，不只是取决于在a.c中声明extern int v，还取决于变量v**本身是能够被引用到**的。

这涉及到c语言的另外一个话题－－变量的作用域。能够被其他模块以extern修饰符引用到的变量通常是**全局变量**。

**还有很重要的一点是，extern int v可以放在a.c中的任何地方**，比如你可以在a.c中的函数fun定义的开头处声明extern int v，然后就可以引用到变量v了，只不过这样只能在函数fun作用域中引用v罢了，这还是变量作用域的问题。对于这一点来说，很多人使用的时候都心存顾虑。好像extern声明只能用于文件作用域似的。  
  
**2. extern修饰函数声明。**

从本质上来讲，变量和函数没有区别。函数名是指向函数二进制块开头处的指针。

如果文件a.c需要引用b.c中的函数，比如在b.c中原型是int fun(int mu)，那么就可以在a.c中声明extern int fun（int mu），然后就能使用fun来做任何事情。

就像变量的声明一样，extern int fun（int mu）可以放在a.c中任何地方，而不一定非要放在a.c的文件作用域的范围中。

对其他模块中函数的引用，最常用的方法是包含这些函数声明的头文件。**使用extern和包含头文件来引用函数有什么区别呢？**extern的引用方式比包含头文件要简洁得多！extern的使用方法是直接了当的，想引用哪个函数就用extern声明哪个函数。

这样做的一个明显的好处是，会加速程序的编译（确切的说是预处理）的过程，节省时间。在大型C程序编译过程中，这种差异是非常明显的。  
  
**3. 此外，extern修饰符可用于指示C或者C＋＋函数的调用规范。**

比如在C＋＋中调用C库函数，就需要在C＋＋程序中用extern “C”声明要引用的函数。

这是给链接器用的，告诉链接器在链接的时候用C函数规范来链接。主要原因是C＋＋和C程序编译完成后在目标代码中命名规则不同。

链接属性

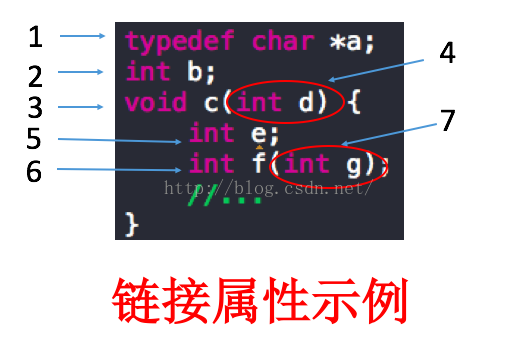
C语言中链接属性决定如何处理在不同文件中出现的标示符。标示符的作用域与它的链接属性有关，但这两个属性并不相同。

链接属性有3种： external(外部)，internal(内部) 和 none(无)。

1. none: 没有链接的标示符，总是被当做单独的个体，也就是说改标示符的多个声明被当做不同的实体。

2. internal: 在同一个源文件内的所有声明中都指同一个实体，但位于不同源文件的多个声明则分属不同的实体。

3. external: 标示符不论声明多少次，位于几个文件都表示同一个实体。



1. 在缺省情况下，标示符b,c,f 的链接属性为external， 其余标示符的链接属性则为none。因此，另一个源文件也包含了标示符b的类似声明并调用函数c，他们实际上访问的是这个源文件所定义的实体。f的链接属性之所以是external，是因为它是函数名。这个源文件中调用f，实际上将链接到其他源文件所定义的函数，甚至这个函数的定义可能出现在某个函数库。

2. 关键字extern和static用于在声明中修改标示符的连接属性。如果某个声明在正常情况下具有external链接属性，在它前面加上static关键字可以使它的链接属性变为internal。例如，将b声明为

static int b;

那么变量b就将为这个源文件所私有。在其他源文件中，如果也链接到一个叫做b的变量，那么它所引用的是另一个不同的变量。

3. static只对缺省链接属性为external的声明才有改变链接属性的效果。例如，你尽管可以在变量e前面加上static关键字，但它的效果完全不一样，因为e的缺省链接属性不是external。

## extern不可以定义局部变量

定义的时候，extern表示外部文件可以使用它，当然它至少得是全局的；全局变量默认是加上了extern的，与之相对的是static，它定义的全局变量，外部文件不能使用。  
  
声明的时候，它起一个扩展作用域的作用，它至少还是全局的，同一文件中，存在一个后面定义的全局变量，在文件开头添加一个extern的声明，前面的代码也可以使用这个变量了；其他的文件，在添加extern声明后，也可以使用它了。

//===============a.h===============

#ifndef A\_H

#define A\_H

#ifdef \_\_cplusplus

**extern**"C"

{

#endif

**int** a(**void**);

#ifdef \_\_cplusplus

}

#endif

#endif

//==============a.cpp================

#include<iostream>

#include"a.h"

#include"b.h"

**int** a(**void**)

{

    b();

**return** 0;

}

//===============b.h===

#ifndef B\_H

#define B\_H

#ifdef \_\_cplusplus

**extern**"C"

{

#endif

**int** b(**void**);

#ifdef \_\_cplusplus

}

#endif

#endif

//================b.c======

#include "b.h"

#include<stdio.h>

#include<stdlib.h>

//#define MAX 6

**int** b(**void**)

{

**int** d = 0;

**return** 0;

}

//==============fw.c=======

#include<stdio.h>

#include<stdlib.h>

#include"a.h"

#include"b.h"

**int** main()

{

**printf**("fa\n");

    a();

    //intf("%d\n",Gg[0]);

    //intf("%d\n",FK[0]);

    //b();

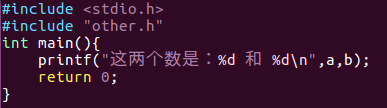
**return** 0;

}

**extern与include “\*.h”的区别**

（1）include的方法

main.c



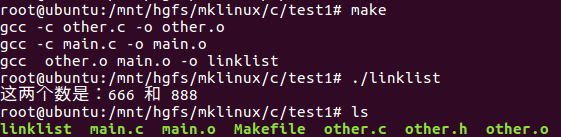
other.c



other.h

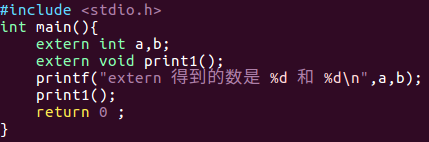


make后

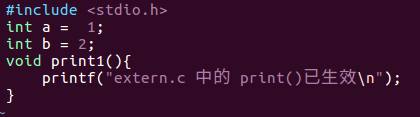


（2）extern 的方法

main.c



extern.c





1. 可以看出extern的方法比方法1 的步骤少了：

创建“\*.h”文件，

在main.c中#include “\*.h”

相比之下extern在编辑阶段比较方便。Extern的功能也可以理解为先不要报错，先让c文件编译成o文件，最后extern部分的变量链接起来就可以用了。