<http://blog.csdn.net/morixinguan/article/details/65494239>

版权声明：本文为博主原创文章，如有需要，请注明转载地址:http://blog.csdn.net/morixinguan。若是侵权用于商业用途，请联系博主，否则将追究责任 http://blog.csdn.net/morixinguan/article/details/65494239

关于静态库和动态库的使用和制作方法。

http://blog.csdn.net/morixinguan/article/details/52451612

今天我们要搞明白的一个概念叫回调函数。

什么是回调函数？

百度的权威解释如下：

回调函数就是一个通过[函数指针](http://baike.baidu.com/view/1604730.htm)调用的函数。如果你把函数的[指针](http://baike.baidu.com/view/159417.htm)（地址）作为[参数传递](http://baike.baidu.com/view/2691131.htm)给另一个函数，当这个指针被用来调用其所指向的函数时，我们就说这是回调函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另外的一方调用的，用于对该事件或条件进行**响应**。

那么我们可以来看一个例子:

**[cpp]** [view plain](http://blog.csdn.net/morixinguan/article/details/65494239) [copy](http://blog.csdn.net/morixinguan/article/details/65494239)

1. #include <stdio.h>
2. **void** print();
3. **int** main(**void**)
4. {
5. **void** (\*fuc)();
6. fuc = print ;
7. fuc();
8. }
9. **void** print()
10. {
11. printf("hello world!\n");
12. }

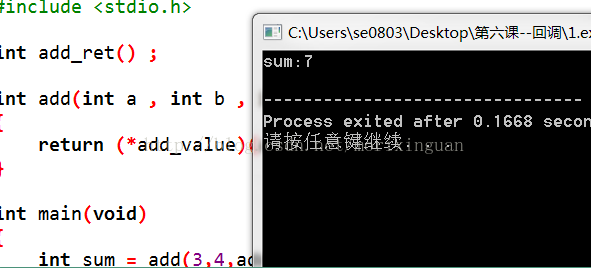


从这个例子可以看到，我们首先定义了一个函数指针fuc ,这个函数指针的返回值为void型,然后我们给函数指针赋值，赋值为print,也就是print函数的首地址，此时fuc获得了print的地址，fuc的地址等于print的地址，所以最终调用fuc();也就相当于调用了print();

那 么我写的这个例子明显和百度解释的不符合啊？定义是如果你把函数的[指针](http://baike.baidu.com/view/159417.htm)（地址）作为[参数传递](http://baike.baidu.com/view/2691131.htm)给另一个函数，当这个指针被用来调用其所指向的函数时，我们就说这是回调函数，确实，有所不同，但道理是一样的，我们接下来再来看一个例子。

**[cpp]** [view plain](http://blog.csdn.net/morixinguan/article/details/65494239) [copy](http://blog.csdn.net/morixinguan/article/details/65494239)

1. #include <stdio.h>
3. **int** add\_ret() ;
5. **int** add(**int** a , **int** b , **int** (\*add\_value)())
6. {
7. **return** (\*add\_value)(a,b);
8. }
10. **int** main(**void**)
11. {
12. **int** sum = add(3,4,add\_ret);
13. printf("sum:%d\n",sum);
14. **return** 0 ;
15. }
17. **int** add\_ret(**int** a , **int** b)
18. {
19. **return** a+b ;
20. }



从这个例子里，我们看到:

这样子不就符合我们的定义了嘛？我们把函数的指针(地址),这里也就是add\_ret,作为参数int add(int a , int b , int (\*add\_value)()) ， 这里的参数就是int(\*add\_value)() , 这个名字可以随便取，但是要符合C语言的命名规范。当这个指针被用来调用其所指向的函数时，我们就说这是回调函数。我们看到add函数内部，return (\*add\_value)(a,b) ; 这个(\*add\_value)(a,b)相当于对指针进行了简引用，我们在main函数中，传入具体要实现功能的函数，add\_ret,这个函数很简单，就是实现两数相加并返回,这里刚刚好，简引用，相当于取出指针返回地址里的值，这个值就是return a+b，也就是我们传入a和b两数相加的结果。

         那么，回调函数究竟有什么作用呢？

说到这里，就有了用户和开发者之间的概念，比方说，刚刚说的add()这个函数，假设一下，用户是实现add这个函数，而开发者是实现add\_value这个函数，用户做的工作不多，就是想要通过开发者提供实现两个数相加的这么一个接口，然后在函数中通过调用开发者实现的这个接口的返回值，然后来实现我们的功能。这个开发者角色就很多了，可以是自己公司的核心开发人物，也可以是别的工作的外包商的人物，这时候，他作为一个开发者的角色完完全全可以将add\_value实现的add\_ret这个函数封装起来并且加密，然后扔一个.so或者.a给用户,那么用户就看不到具体add\_ret的实现内容，用户只需要开发者给他提供一个.h和.so即可，这样，作为开发者，他就将他实现的函数功能给保密了。

 接下来，我们用linux来演示下这个结果:

         我们在目录下创建三个文件，main.c,vendor.c,vendor.h

         Main.c是用户开发的

         Vendor.c和vendor.h是开发者实现的。

在main.c中，代码如下：

**[cpp]** [view plain](http://blog.csdn.net/morixinguan/article/details/65494239) [copy](http://blog.csdn.net/morixinguan/article/details/65494239)

1. #include <stdio.h>
2. #include "vendor.h"
4. **int** add(**int** a , **int** b , **int** (\*add\_value)())
5. {
6. **return** (\*add\_value)(a,b);
7. }
9. **int** main(**void**)
10. {
11. **int** sum = add(3,4,add\_ret);
12. printf("sum:%d\n",sum);
13. **return** 0 ;
14. }

vendor.c，代码如下：

**[cpp]** [view plain](http://blog.csdn.net/morixinguan/article/details/65494239) [copy](http://blog.csdn.net/morixinguan/article/details/65494239)

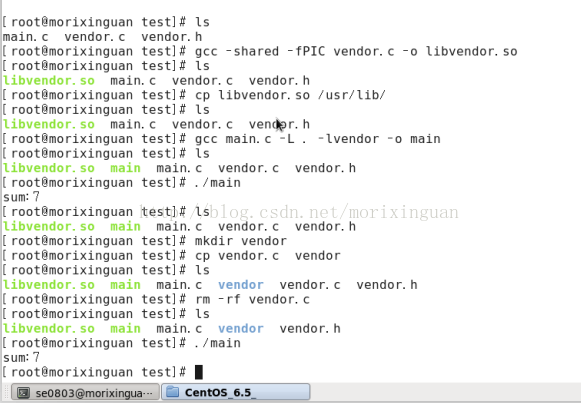
1. #include "vendor.h"
2. **int** add\_ret(**int** a , **int** b)
3. {
4. **return** a+b ;
5. }

vendor.h，代码如下：

**[cpp]** [view plain](http://blog.csdn.net/morixinguan/article/details/65494239) [copy](http://blog.csdn.net/morixinguan/article/details/65494239)

1. #ifndef \_\_VENDOR\_H
2. #define \_\_VENDOR\_H
4. **int** add\_ret(**int** a, **int** b) ;
6. #endif

接下来，我们制作一个动态链接库，最终开发者把vendor.c的内容封起来，把vendor.h提供给用户使用。



在linux下制作动态链接库，将vendor.c和vendor.h打包成一个动态链接库

先明白以下几个命令是什么意思：

生成动态库:

gcc -shared -fPIC dvendor.c -o libvendor.so

-shared : 生成动态库;

-fPIC  : 生成与位置无关代码;

-o               :指定生成的目标文件;

使用动态库:

gcc main.c -L . –lvendor -o main

-L : 指定库的路径(编译时); 不指定就使用默认路径(/usr/lib/lib)

-lvendor : 指定需要动态链接的库是谁;

代码运行时需要加载动态库:

./main 加载动态库 (默认加载路径:/usr/lib /lib ./ ...)

./main

我们将编译动态库生成的libvendor.so拷贝到/usr/lib后，现在就不需要vendor.c了，此时我们将vendor.c移除，也可以正常的编译并且执行main函数的结果，这就是回调函数的作用之一。