<https://www.cnblogs.com/bdsir/p/5789724.html>

# [c语言回调函数理解](http://www.cnblogs.com/bdsir/p/5789724.html)

开源c项目当中回调函数实在是太常用了，一直不太清楚回调函数和普通函数的区别。找个机会总结下吧！

# 数组指针

C语言的核心之一就是指针，理解指针很关键。先来个有趣的东西。

int \*p1[10] //指针数组

int (\*p1)[10] //数组指针

这个应该不难理解，表达式1中p1和后面[10]结合成为数组的定义，前方的int \* 表示数组的元素为指向int的指针，因此为指针数组。表达式2中 \* 和p1结合成为一个指针的定义，int表示数组中的元素为int，结合起来表示一个指向数组的指针。  
按照一般指针的定义方式，或许我们可以这样定义一个数组指针。  
\*\*int (\*)[10] p2\*\*  
显然上述表达式编译器无法识别，为了方便，编译器将p2提到前面。此处简单介绍一下为后续函数指针作铺垫。

# 函数指针

接触回调函数之前先了解一下函数指针，函数指针就是指向函数的指针。

char \*(\*fun1)(char \*p1, char \*p2);

char \*\*fun1(char \*p1, char \*p2);

char \*fun1(char \*p1, char \*p2);

显然只有表达式1为函数指针，有了数组指针的基础理解这个应该不难了。

# 回调函数

简单来说，如果把一个函数的入口地址传递给一个函数指针，通过函数指针去调用这个函数，那么称这个函数为回调函数。回调函数的一般用法为：A进行结构体的定义（成员中包括回调函数的指针），B进行声明结构体变量，并向A注册回调函数。A在某个特定事件触发的时候，遍历并执行所有注册的回调函数。  
下面代码是一个简单回调函数的例程

#include <stdio.h>

struct operation {

void (\*callback)();

};

void trigger(struct operation\* ops, int ops\_length) {

int i = 0;

while(i++ < ops\_length) {

ops->callback();

ops++;

}

}

void a\_callback() {

printf("I'm A\n");

}

void b\_callback() {

printf("I'm B\n");

}

int main(int argc, char\* argv[]) {

struct operation A,B;

A.callback = &a\_callback;

B.callback = &b\_callback;

struct operation ops[2];

ops[0] = A;

ops[1] = B;

trigger(ops, 2);

return 1;

}

A、B分别向operation注册了各自的回调函数。