查看: 584 | 回复: 48

C 语言面向对象编程 - 封装 [复制链接]

广轻电气091

728 积分 | 28 主题 | 672 帖子

中级会员

发消息

发表于 6 小时前 | 只看该作者 | 只看大图 1楼 电梯直达

大部分使用 C 语言进行开发的工程师，在接触更高级的编程语言之前，都认为 C 语言是面向过程的。
确实，对于一些小规模的应用程序，C 语言一般都被用作面向过程编程。例如：单片机应用程序开发。
但是，如果是使用 C 语言开发一些规模较大的软件时，就必须用面向对象的思想去考虑和设计整个软件框架了。例如：嵌入式Linux操作系统。
嵌入式Linux操作系统虽然是使用 C 语言作为主要的编写语言，但里面的设计大部分都使用了面向对象的编程思想。
很多单片机工程师或者嵌入式Linux驱动初学者，觉得入门困难，很大一部分原因就是，他们还停留在单片机那种面向过程的思维模式上面。
编程语言只是一种工具，编程思想才是用好这个工具的关键。C 语言只是工具，而面向对象是一种编程思想，用来指导我们如何用好 C 语言。
接下来，我们将尝试使用 C 语言进行面向对象程序开发，务求使用 C 语言实现面向对象的一些基本特性。

首先，我们先来说说封装。
封装就是把一个抽象事物的属性和属性的操作函数打包在一起，外界的模块只能通过这个抽象事物对外提供的函数接口，对其属性进行访问。
在C++或其他高级语言中，封装通常被称作“类”，而 C 语言一般使用结构体对事物进行封装。
接下来，我们先看两段代码，这两段代码主要声明和定义了一个坐标类对象，以及其坐标属性，还提供坐标属性的操作函数。

头文件 coordinate.h

```h
#ifndef __COORDINATE_H_
#define __COORDINATE_H_

//创建一个位置类对象，属性为坐标x，y
typedef struct coordinate
{
    short int x;
    short int y;
}COORDINATE_T,*P_COORDINATE_T;

extern P_COORDINATE_T coordinate_create(short int x,short int y);
extern void coordinate_destroy(P_COORDINATE_T p_coordinate);
extern void coordinate_moveby(P_COORDINATE_T p_coordinate,short int dx,short int dy);
extern short int coordinate_get_x(P_COORDINATE_T p_coordinate);
extern short int coordinate_get_y(P_COORDINATE_T p_coordinate);
extern void coordinate_test_function(void);

#endif // !__COORDINATE_H_

```

源文件 coordinate.c

```c
#include "string.h"
#include "inc/coordinate.h"

//创建一个coordinate对象
P_COORDINATE_T coordinate_create(short int x,short int y)
{
    if((x == 0) || (y == 0)){
        printf("coordinate creat error! x or y can not be zero \n");
        return NULL;
    }

    P_COORDINATE_T p_coordiante = NULL;

    p_coordiante = (P_COORDINATE_T)malloc(sizeof(COORDINATE_T));

    if(NULL != p_coordiante){
        p_coordiante->x = x;
        p_coordiante->y = y;

        return p_coordiante;
    }
    else printf("coordinate malloc error! \n");

    return NULL;
}

//销毁一个coordinate对象
void coordinate_destroy(P_COORDINATE_T p_coordiante)
{
    if(NULL != p_coordiante){
        free(p_coordiante);
        p_coordiante = NULL;
    }
}

//修改coordinate的属性值
void coordinate_moveby(P_COORDINATE_T p_coordiante,short int dx,short int dy)
{
    if(NULL != p_coordiante){
        p_coordiante->x += dx;
        p_coordiante->y += dy;
    }
}

//获取coordinate的属性值x
short int coordinate_get_x(P_COORDINATE_T p_coordiante)
{
    return (NULL != p_coordiante) ? p_coordiante->x : -1;
}

//获取coordinate的属性值y
short int coordinate_get_y(P_COORDINATE_T p_coordiante)
{
    return (NULL != p_coordiante) ? p_coordiante->y : -1;
}
```

代码比较简单，在头文件 coordinate.h里面，通过结构体封装了一个coordinate类，里面有两个坐标属性 x 和 y 。
coordinate_create 函数主要用于创建一个 P_COORDINATE_T 类型的对象，并为其分配内存空间，内存分配成功后，设置两个坐标属性的初始值，最后返回申请成功的对象指针。
coordinate_destroy 主要是释放对象之前申请的内存空间，然后把对象指针重置为NULL。
其他的操作函数，主要是对类对象的属性进行操作，比如获取 x 和 y 的属性值，重置坐标的属性值。

以下是测试函数，在主函数中调用，即可测试类coordinate对外提供的接口。

```c
void coordinate_test_function(void)
{
    P_COORDINATE_T p_coordiante_1 = NULL;
    P_COORDINATE_T p_coordiante_2 = NULL;

    p_coordiante_1 = (P_COORDINATE_T)coordinate_create(100,200);
    p_coordiante_2 = (P_COORDINATE_T)coordinate_create(10,20);

    if((NULL == p_coordiante_1) || (NULL == p_coordiante_2)){
        printf("p_coordiante_1 or p_coordiante_2 create error! \n");
    }

    printf("p_coordiante_1 x = %d, y = %d \n",coordinate_get_x(p_coordiante_1), coordinate_get_y(p_coordiante_1));
    printf("p_coordiante_2 x = %d, y = %d \n",coordinate_get_x(p_coordiante_2), coordinate_get_y(p_coordiante_2));

    coordinate_moveby(p_coordiante_1,50,50);
    coordinate_moveby(p_coordiante_2,50,50);

    printf("after moveby p_coordiante_1 x = %d, y = %d \n",coordinate_get_x(p_coordiante_1), coordinate_get_y(p_coordiante_1));
    printf("after moveby p_coordiante_2 x = %d, y = %d \n",coordinate_get_x(p_coordiante_2), coordinate_get_y(p_coordiante_2));

    coordinate_destroy(p_coordiante_1);
    coordinate_destroy(p_coordiante_2);
}
```

测试代码比较简单，主要是创建了两个 P_COORDINATE_T 类型的对象，然后打印其坐标初始值，再通过对外提供的函数修改其坐标值，然后再打印出来。
测试函数运行后，结果如下所示：

```
root@embediot-virtual-machine:/opt/work/my_program_test-master/c_object/c_object_packaging#
p_coordiante_1 x = 100, y = 200
p_coordiante_2 x = 10, y = 20
after moveby p_coordiante_1 x = 150, y = 250
after moveby p_coordiante_2 x = 60, y = 70
```

从上述代码可以看出，使用结构体可以很好地对数据进行封装，并且需要通过指定的操作函数对结构体内的数据进行访问。
每个操作函数的第一个参数是对象本身的指针，通过这个指针去访问其体对象里面的属性。
这是因为在 C 语言中不存在像 C++ 语言那样的 this 指针，所以我们只能显式地通过函数传参的方式，让函数内部可以访问对象实例的其他成员。