

Copy path

0323df0 8 days ago

48 lines (39 sloc) 2.45 KB

例子1：

```
int main(void)
{
    Array arr1;
    Array arr2 = arr1;
    return 0;
}
```



```
Array arr1;
```

```
Array arr2 = arr1;
```

```
Array(const Array &arr)
{
    m_iCount = arr.m_iCount;
}
```

<https://github.com/YHim/hello-world/blob/master/C%2B%2B%E8%BF%9C%E5%BE%81%E4%B9%8B%E5%B0%81%E8%A3%85%E7%AF%87/...> 1/3

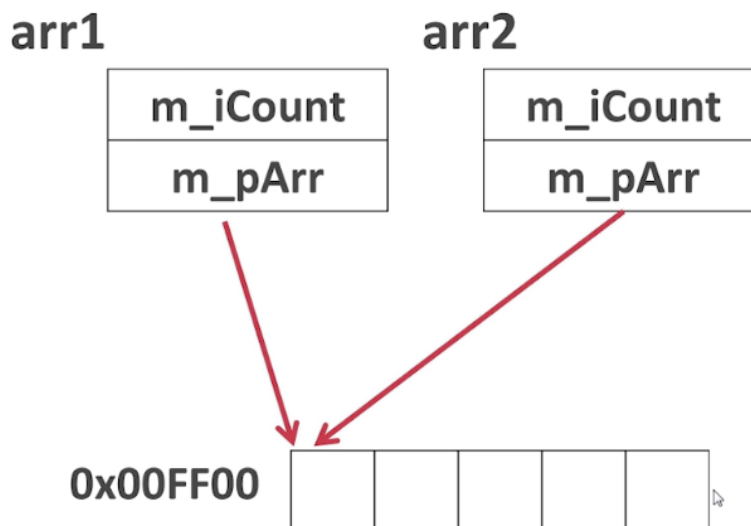
例子2：

```
class Array
{
public:
    Array() { m_iCount = 5; m_pArr = new int[m_iCount];}
    Array(const Array& arr)
    { m_iCount = arr.m_iCount;
      m_pArr = arr.m_pArr;}
private:
    int m_iCount;
    int *m_pArr;
};

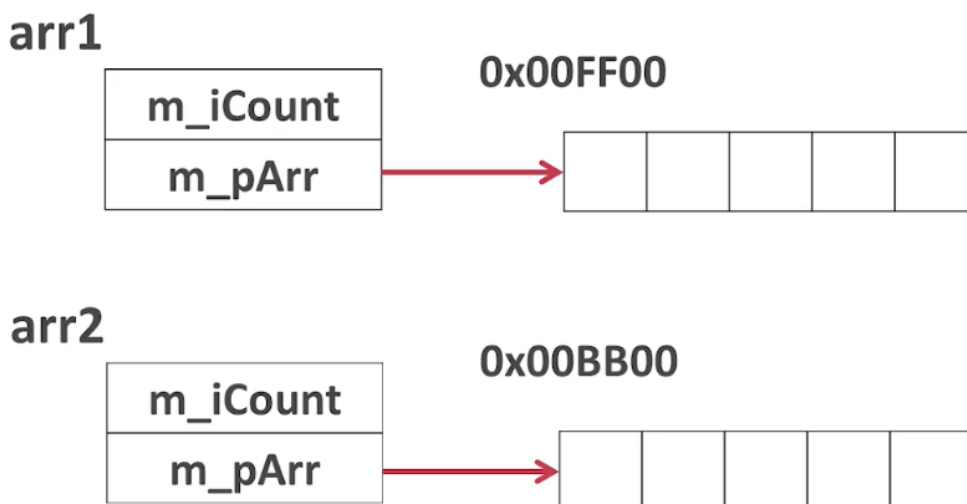
int main(void)
{
    Array arr1;
    Array arr2 = arr1;
    return 0;
}
```

这里的m_pArr指针的构造函数是从堆中申请一段内存，并且指向了堆中的这段内存，内存大小就是m_iCount。使用时，Array arr1; 就会调用构造函数，Array arr2 = arr1; 就会调用arr2的拷贝构造函数。这两个图片中的例子只是将数据成员的值进行了简单的拷贝，这种方式称之为**浅拷贝**。

对于例子1来说，使用浅拷贝的方式来实现拷贝构造函数没有问题。但对于例子2来说，是有问题的。经过浅拷贝之后，对象arr1中的指针和对象arr2中的指针势必会指向同一块内存。因为我们将arr1的m_pArr赋值给了arr2的m_pArr。



此时，如果我们先给arr1的m_pArr赋值，然后再给arr2的m_pArr赋值，那么图中的这段内存就会被重写，而不再是之前arr1的m_pArr的值。最严重的情况是，当我们去销毁arr1这个对象的时候，为了避免内存泄漏，肯定会释放掉arr1的m_pArr所指向的这段内存。如果我们已经释放掉了这段内存，再去销毁arr2对象的时候，以同样方式去释放m_pArr这个指针所指向的那段内存。那么，同样的内存被释放了两次，这样计算机就会崩溃。这种崩溃与语法错误是不同的。所以，我们希望拷贝构造函数所完成的工作应该是这样的：



两个对象的指针所指向的应该是两个不同的内存，拷贝的时候不是将指针的地址简单的拷贝过来，而是将指针所指向内存当中的每一个元素依次地拷贝过来。如果想要实现这样的效果，例子2的代码应该写成下面的样子：

```
class Array
{
public:
    Array() { m_iCount = 5; m_pArr = new int[m_iCount];}
    Array(const Array& arr){
        m_iCount = arr.m_iCount;
        m_pArr = new int[m_iCount];
        for(int i = 0; i < m_iCount; i++)
        {m_pArr[i] = arr.m_pArr[i];}
    private:
        int m_iCount;
        int *m_pArr;
};
```



即当进行对象拷贝时，不是简单的做值的拷贝，而是将堆中内存的数据也进行拷贝，这种拷贝模式称之为**深拷贝**。

 YHim 111

0323df0 8 days ago

1 contributor

75 lines (63 sloc) 904 Bytes

例子-浅拷贝

```
/*
*****
*/
示例安排：

1. 定义一个Array类，数据成员为m_iCount，
   成员函数包括数据封装函数、构造函数、拷
   贝构造函数和析构函数，通过此示例体会浅
   拷贝原理。

2. 增加数据成员m_pArr，并增加m_pArr地址
   查看函数，同时改造构造函数、拷贝构造函
   数和析构函数，体会深拷贝的原理和必要性。
*/
*****
```

要求如图：

Array.h

```
class Array
{
public:
    Array();
    Array(const Array &arr);
    ~Array();
    void setCount(int count);
    int getCount();
private:
    int m_iCount;
};
```

Array.cpp

```
#include <iostream>
#include "Array.h"
using namespace std;

Array::Array()
{
    cout << "Array" << endl;
}

Array::Array(const Array &arr)
{
    m_iCount = arr.m_iCount;
    cout << "Array &" << endl;
}

Array::~~Array()
{
    cout << "~Array" << endl;
}

void Array::setCount(int count)
{

```

```
        m_iCount = count;
    }

    int Array::getCount()
    {
        return m_iCount;
    }
}
```

demo.cpp

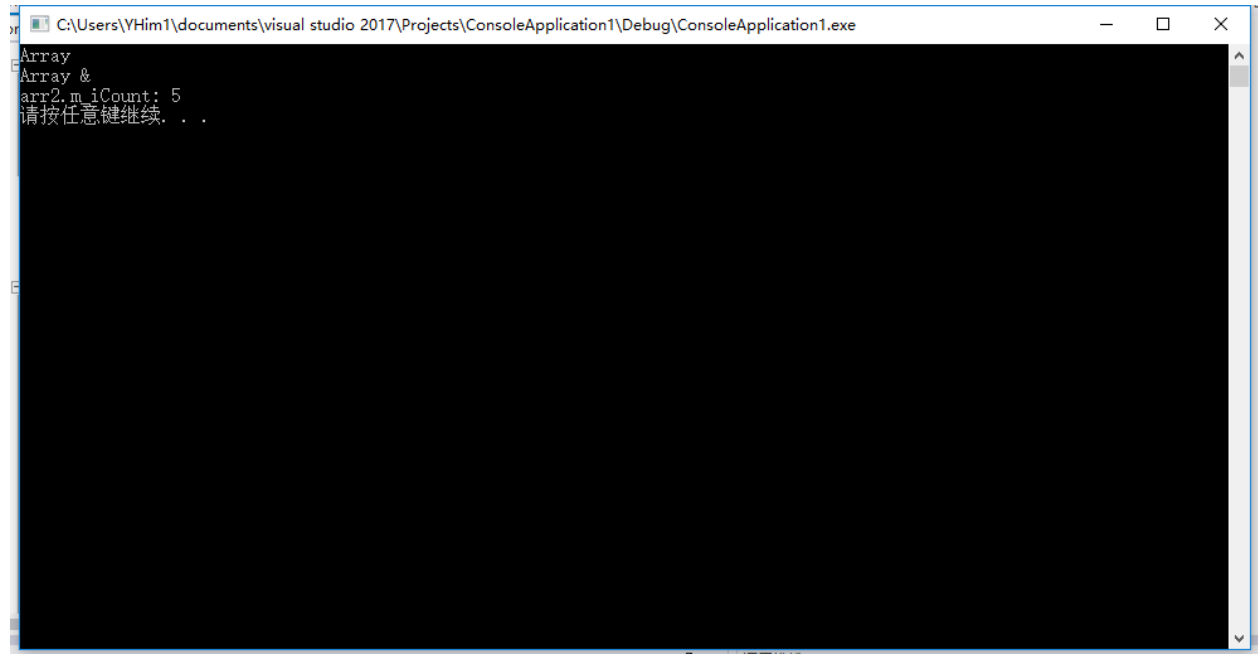
```
#include <iostream>
#include <stdlib.h>
#include "Array.h"
using namespace std;

int main(void)
{
    Array arr1;
    arr1.setCount(5);

    Array arr2(arr1);
    cout << "arr2.m_iCount: " << arr2.getCount() << endl;

    system("pause");
    return 0;
}
```

运行结果为：



```
C:\Users\YHim1\documents\visual studio 2017\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
Array
Array &
arr2.m_iCount: 5
请按任意键继续. . .
```

例子-深拷贝

```
/*
*****
示例安排：

1. 定义一个Array类，数据成员为m_iCount，
   成员函数包括数据封装函数、构造函数、拷
   贝构造函数和析构函数，通过此示例体会浅
   拷贝原理。

2. 增加数据成员m_pArr，并增加m_pArr地址
   查看函数，同时改造构造函数、拷贝构造函
   数和析构函数，体会深拷贝的原理和必要性。
*/
*****
```

要求如图：

Array.h

```
class Array
{
public:
    Array(int count);
    Array(const Array &arr);
    ~Array();
    void setCount(int count);
    int getCount();
    void printAddr();
private:
    int m_iCount;
    int *m_pArr;
};
```

Array.cpp

```
#include <iostream>
#include "Array.h"
using namespace std;

Array::Array(int count)
{
    m_iCount = count;
    m_pArr = new int[m_iCount];
    cout << "Array" << endl;
}

Array::Array(const Array &arr)
{
    m_pArr = arr.m_pArr;    //浅拷贝
    m_iCount = arr.m_iCount;
    cout << "Array &" << endl;
}

Array::~~Array()
{
}
```

```
delete []m_pArr;    /*这里释放内存会出错，但运行起来程序没出
                    错的原因是demo.cpp的main函数中写了
                    system("pause"),当代码运行完后就会出错，
                    即按下任意键后程序会出错。 */

m_pArr = NULL;
cout << "~Array" << endl;
}

void Array::setCount(int count)
{
    m_iCount = count;
}

int Array::getCount()
{
    return m_iCount;
}

void Array::printAddr()
{
    cout << "m_pArr的值是: " << m_pArr << endl;
}
```

demo.cpp

```
#include <iostream>
#include <stdlib.h>
#include "Array.h"
using namespace std;

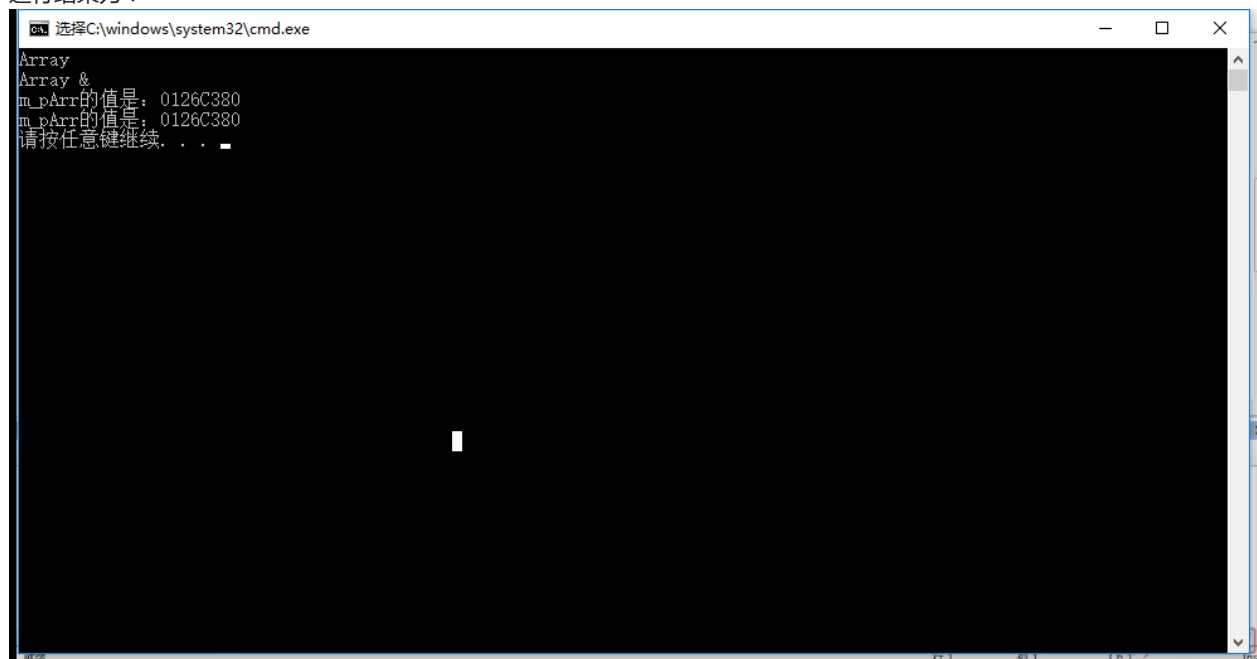
int main(void)
{
    Array arr1(5);

    Array arr2(arr1);

    arr1.printAddr();
    arr2.printAddr();

    system("pause");
    return 0;
}
```

运行结果为：



```
选择C:\windows\system32\cmd.exe
Array
Array &
m_pArr的值是: 0126C380
m_pArr的值是: 0126C380
请按任意键继续. . .
```

解决以上程序的问题就要使用深拷贝。将代码更改如下：

Array.h

```

class Array
{
public:
    Array(int count);
    Array(const Array &arr);
    ~Array();
    void setCount(int count);
    int getCount();
    void printAddr();
    void printArr();
private:
    int m_iCount;
    int *m_pArr;
};

```

Array.cpp

```

#include <iostream>
#include "Array.h"
using namespace std;

Array::Array(int count)
{
    m_iCount = count;
    m_pArr = new int[m_iCount];
    for (int i = 0; i < m_iCount; i++)
    {
        m_pArr[i] = i;    //为了更清楚的理解，将m_pArr每个元素赋值
    }
    cout << "Array" << endl;
}

Array::Array(const Array &arr)
{
    m_iCount = arr.m_iCount;
    m_pArr = new int[m_iCount]; //先申请一段内存
    for (int i = 0; i < m_iCount; i++) /*然后将传入进来的arr的这个
                                        对象对应位置的内存拷贝到
                                        申请的那段内存中去*/
    {
        m_pArr[i] = arr.m_pArr[i];
    }
    cout << "Array &" << endl;
}

Array::~Array()
{
    delete []m_pArr;
    m_pArr = NULL;
    cout << "~Array" << endl;
}

void Array::setCount(int count)
{
    m_iCount = count;
}

int Array::getCount()
{
    return m_iCount;
}

void Array::printAddr()
{
    cout << "m_pArr的值是: " << m_pArr << endl;
}

void Array::printArr()
{
    for (int i = 0; i < m_iCount; i++)
    {
        cout << m_pArr[i] << endl;
    }
}

```

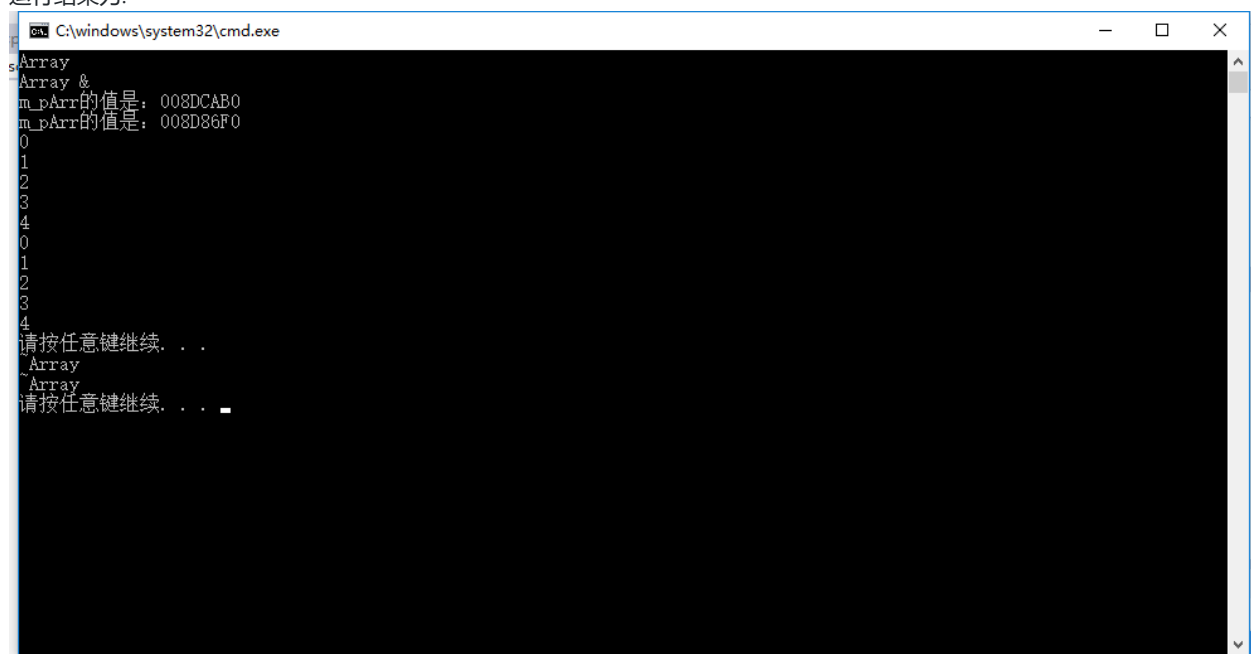


```
}  
}
```

demo.cpp

```
#include <iostream>  
#include <stdlib.h>  
#include "Array.h"  
using namespace std;  
  
int main(void)  
{  
    Array arr1(5);  
  
    Array arr2(arr1);  
  
    arr1.printAddr();  
    arr2.printAddr();  
    arr1.printArr();  
    arr2.printArr();  
  
    system("pause");  
    return 0;  
}
```

运行结果为:



```
C:\windows\system32\cmd.exe  
Array  
Array &  
m_pArr的值是: 008DCAB0  
m_pArr的值是: 008D86F0  
0  
1  
2  
3  
4  
0  
1  
2  
3  
4  
请按任意键继续. . .  
Array  
Array  
请按任意键继续. . .
```

可以看到指向的不再是同一块内存了,按了任意键之后程序也没有崩溃掉。第一遍打印出来的0、1、2、3、4是arr1中的,第二遍是arr2中的。