

Branch: master hello-world / C++远征之封装篇 / 第四章-对象指针 / this指针 / this指针.md

Find file Copy path

YHim 111

0323df0 19 days ago

1 contributor

48 lines (30 sloc) 2.53 KB

this指针

例子：

```
class Array
{
public:
    Array(int _len) {len = _len;}
    int getLen() { return len; }
    void setLen(int _len) { len = _len; }
private:
    int len;
};
```



这是一个数组的类，通过观察可以发现参数和数据成员都不同(即_len和len)。

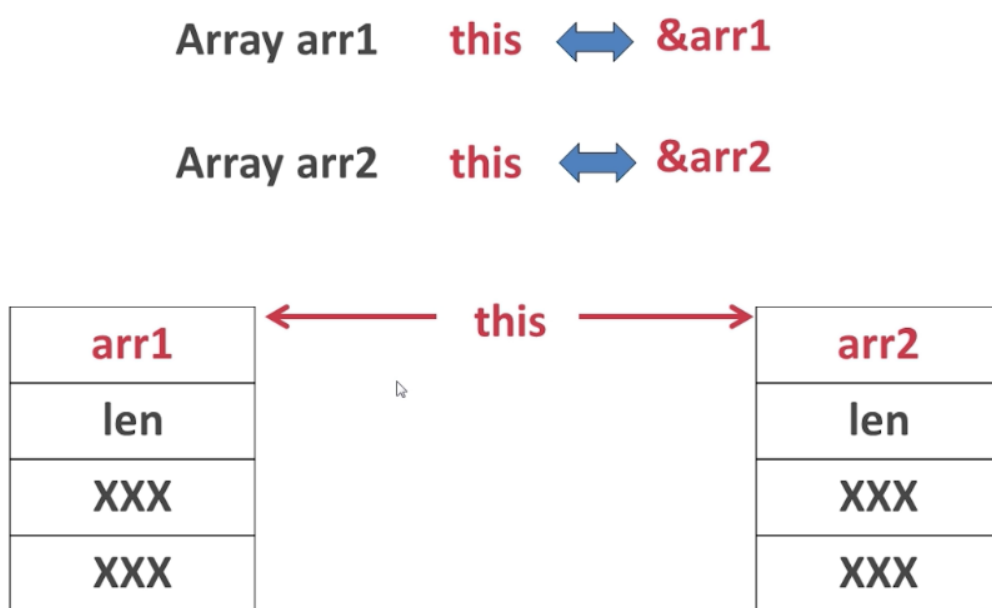
如果参数和数据成员同名会怎样呢？(见下面的例子)

```
class Array
{
public:
    Array(int len) {len = len;}
    int getLen() { return len; }
    void setLen(int len) { len = len; }
private:
    int len;
};
```



在这个例子中，对于构造函数和setLen()，编译器无法分辨哪个是作为参数的len，哪个又是作为数据成员的len，所以认为这是错误的赋值方式。

this指针就是指向对象自身数据的指针。



在刚才的例子中，如果使用Array这个类实例化一个对象arr1，那么this指针就相当于给arr1取地址，也就是说this就是&arr1；如果再实例化一个对象arr2，那么this指针此时就是arr2的地址。如果画图来说，就可以按照上图的下半部分来表示。如果实例化arr1，那么在内存当中就有这么一段空间。在arr1的空间当中，有len这个数据成员，未来也许还会有其他的数据成员xxx。如果实例化另外一个对象arr2，那么arr2中也有自己的len，当然，未来也可能有其他的数据成员。this如果写在arr1里面，就是arr1的地址，写在arr2里面，就是arr2的地址。可见，this要表达什么意思，取决于它在什么位置。通过this指针，可以访问到它表达的对象自身的任何数据。

将this指针应用到代码当中，可以写成这样：

```
class Array
{
public:
    Array(int len) {this->len = len;}
    int getLen() { return len; }
    void setLen(int len) { this->len = len; }
private:
    int len;
};
```



可以发现，如果想用与数据成员同名的参数，那么就可以在数据成员的前面加上"this->"，来表达数据成员的len，然后将参数的len赋值给数据成员的len。这样，计算机就分得清，代码也就能编译的过去。

成员函数如何访问到对应的数据成员？再看一个例子：

```
class Array
{
public:
    Array(T *this , int _len) {this->len = _len;}
    int getLen(T *this) { return this->len; }
    void setLen(T *this, int _len) { this->len = _len; }
private:
    int len;
};
```



```
int main(void)
{
    Array arr1(this, 10); → len = 10 → this->len = 10
    arr1. getLen(this); → return len → return this->len
    Array arr2(this, 20); → len = 20 → this->len = 20
    arr2. setLen(this, 30); → len = 30 → this->len = 30
    return 0;
}
```



当实例化对象并使用成员函数时，this指针就代表着这个对象本身的地址。Array arr1(this,10);当调用len = 10的时候，就相当于调用this->len = 10。用this去指向len的时候，其实就是指向arr1中的len，也就不会给其他的对象赋错值了。其他语句同理。

回到之前的例子：

```
class Array
{
public:
    Array(int _len) {len = _len;}
    int getLen() { return len; }
    void setLen(int _len) { len = _len; }
private:
    int len;
};
```



其实在编译的时候，编译器自动地为每个成员函数的参数列表都加了一个this指针，因此我们自定义的时候，就不必要加this指针这个参数，使用的时候也可以当作没这回事。那么系统自动添加的this指针，添加在参数列表的什么位置呢？详见this指针例子。

Branch: master hello-world / C++远征之封装篇 / 第四章-对象指针 / this指针 / this指针-例子.md

Find fileCopy path

YHim 12392f84ee 11 days ago

1 contributor

449 lines (366 sloc)6.53 KB

this指针例子

要求如下：

```
/* *****
/* 示例要求：
/*  定义一个Array类。
/*  数据成员：m_iLen 表示数组长度
/*  成员函数：
/*    构造函数
/*    析构函数
/*    len的封装函数
/*    信息输出函数 printInfo
/* ***** */
```

简单的this指针用法：Array.h

```
class Array
{
public:
    Array(int len);
    ~Array();
    void setLen(int len);
    int getLen();
    void printInfo();
private:
    int len;
};
```

Array.cpp

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

Array::Array(int len)
{
    this->len = len;
}

Array::~Array()
{
}

void Array::setLen(int len)
{
    this->len = len;
}

int Array::getLen()
{
    return len;
}

void Array::printInfo()
```

```
{  
}
```

demo.cpp

```
#include "Array.h"  
#include <iostream>  
#include <stdlib.h>  
using namespace std;  
  
int main()  
{  
    Array arr1(10);  
    cout << arr1.getLen() << endl;  
  
    system("pause");  
    return 0;  
}
```

this指针比较特殊的用法：Array.h

```
class Array  
{  
public:  
    Array(int len);  
    ~Array();  
    void setLen(int len);  
    int getLen();  
    Array printInfo(); //将返回值改为Array  
private:  
    int len;  
};
```

Array.cpp

```
#include <iostream>  
#include "Array.h"  
using namespace std;  
  
Array::Array(int len)  
{  
    this->len = len;  
}  
  
Array::~Array()  
{  
}  
  
void Array::setLen(int len)  
{  
    this->len = len;  
}  
  
int Array::getLen()  
{  
    return len;  
}  
  
Array Array::printInfo()    //与之前代码相比，把返回值改为了Array  
{  
    cout << "len = " << len << endl;  
    return *this;    //this本身是一个指针，*this就是一个对象，而这里要求的返回值就是返回一个Array对象。  
}
```

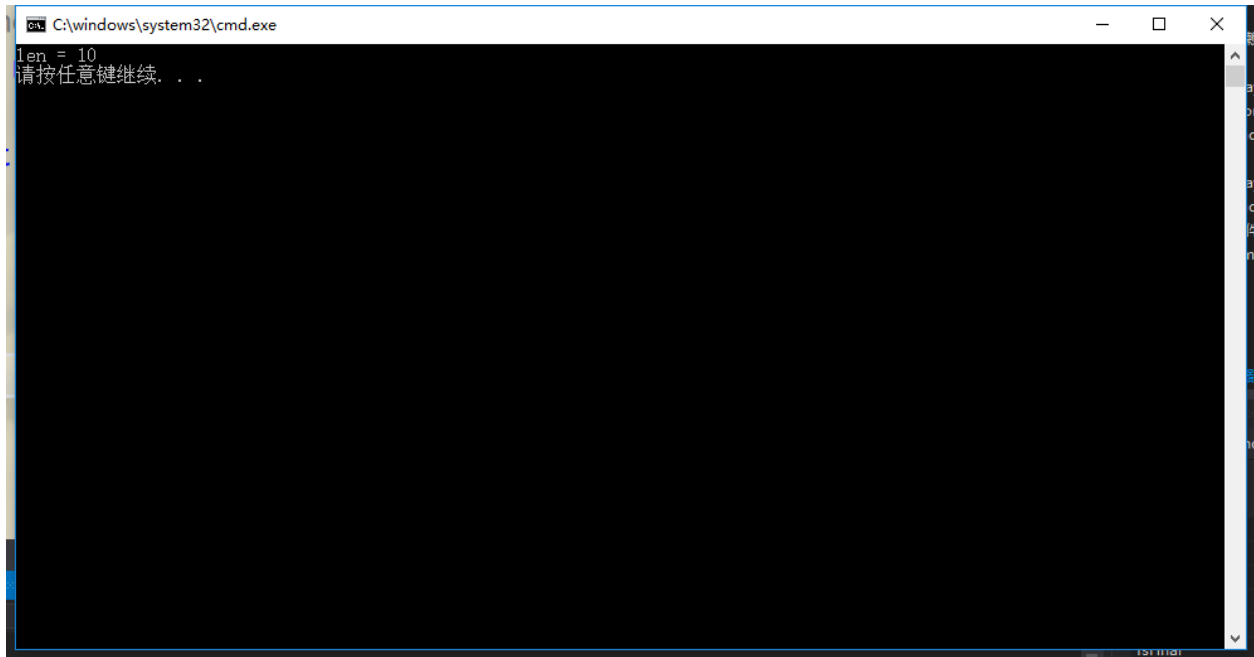
demo.cpp

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

int main()
{
    Array arr1(10);
    arr1.printInfo();

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

运行结果为：



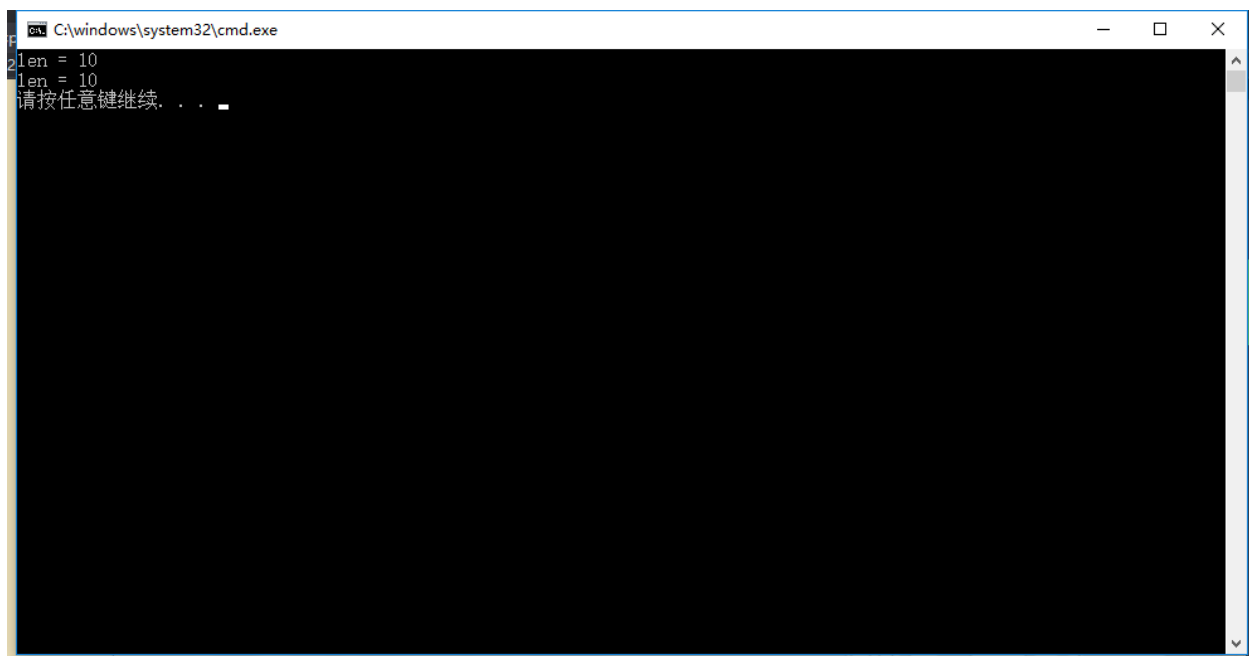
this指针的返回价值：代码中this指针返回回来之后就意味着调用printInfo()这个函数之后，可以在后面用“.”来访问更多的内容。比如说上面代码中的arr1.printInfo().setLen(5); 那么，这里的setLen(5)是不是改变的是arr1中的值呢？将demo.cpp中的代码改成下面的：

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

int main()
{
    Array arr1(10);
    arr1.printInfo().setLen(5);
    cout << "len = " << arr1.getLen() << endl;

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

运行结果为：



可见，在这里使用setLen并没有改变arr1的值。这是因为：返回的*this返回出来变成了另一个Array对象，这个对象是一个临时的对象，它并不是arr1。如果想要它返回的是arr1，就要将Array.cpp中的

```
Array Array::printInfo()
```

更改为：(加上一个引用符号)

```
Array& Array::printInfo()
```

Array.h里面相应的地方也要进行更改。即再将Array.cpp和Array.h代码更改成下面的： Array.h

```
class Array
{
public:
    Array(int len);
    ~Array();
    void setLen(int len);
    int getLen();
    Array& printInfo();
private:
    int len;
};
```

Array.cpp

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

Array::Array(int len)
{
    this->len = len;
}

Array::~Array()
{
}

void Array::setLen(int len)
{
    this->len = len;
}

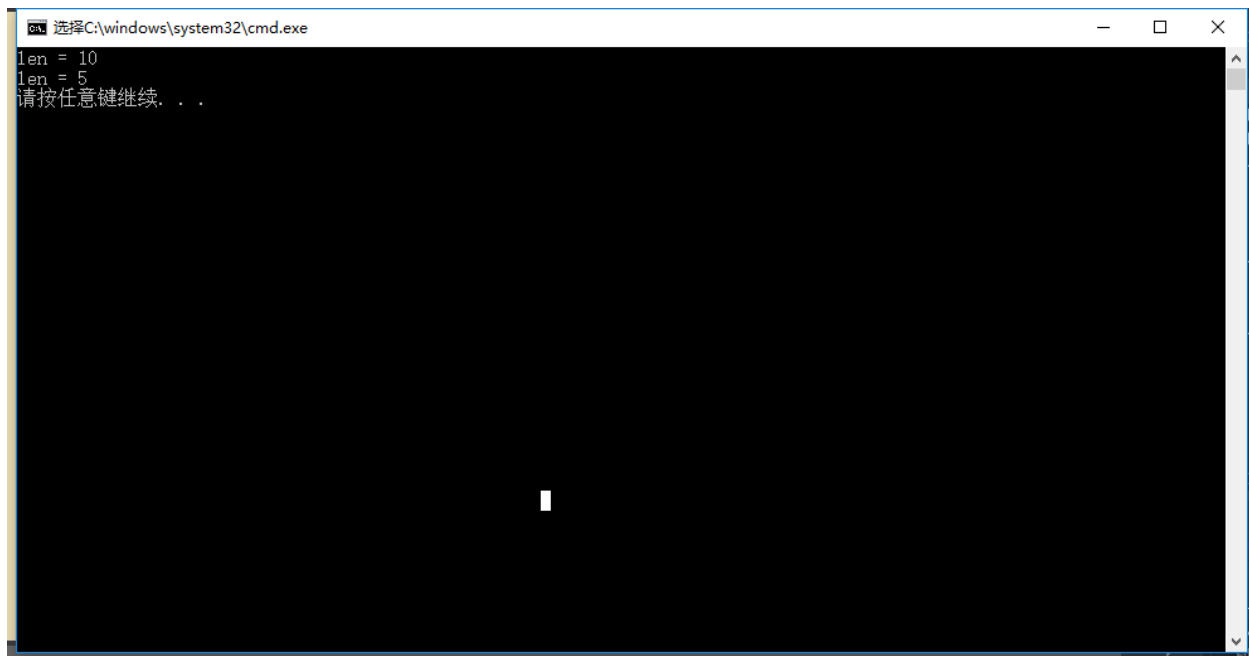
int Array::getLen()
{
}
```



```
        return len;
    }

    Array& Array::printInfo()
    {
        cout << "len = " << len << endl;
        return *this;
    }
}
```

运行结果为：



可以看到，更改了arr1的值。

比如要想在setLen()后面继续用"."来访问更多的内容，要将代码更改如下：Array.h

```
class Array
{
public:
    Array(int len);
    ~Array();
    Array& setLen(int len); // 做出更改
    int getLen();
    Array& printInfo();
private:
    int len;
};
```

Array.cpp

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

Array::Array(int len)
{
    this->len = len;
}

Array::~Array()
{
}

Array& Array::setLen(int len) // 做出更改
{
    this->len = len;
```

```

        return *this;//做出更改
    }

    int Array::getLen()
    {
        return len;
    }

    Array& Array::printInfo()
    {
        cout << "len = " << len << endl;
        return *this;
    }

```

demo.cpp

```

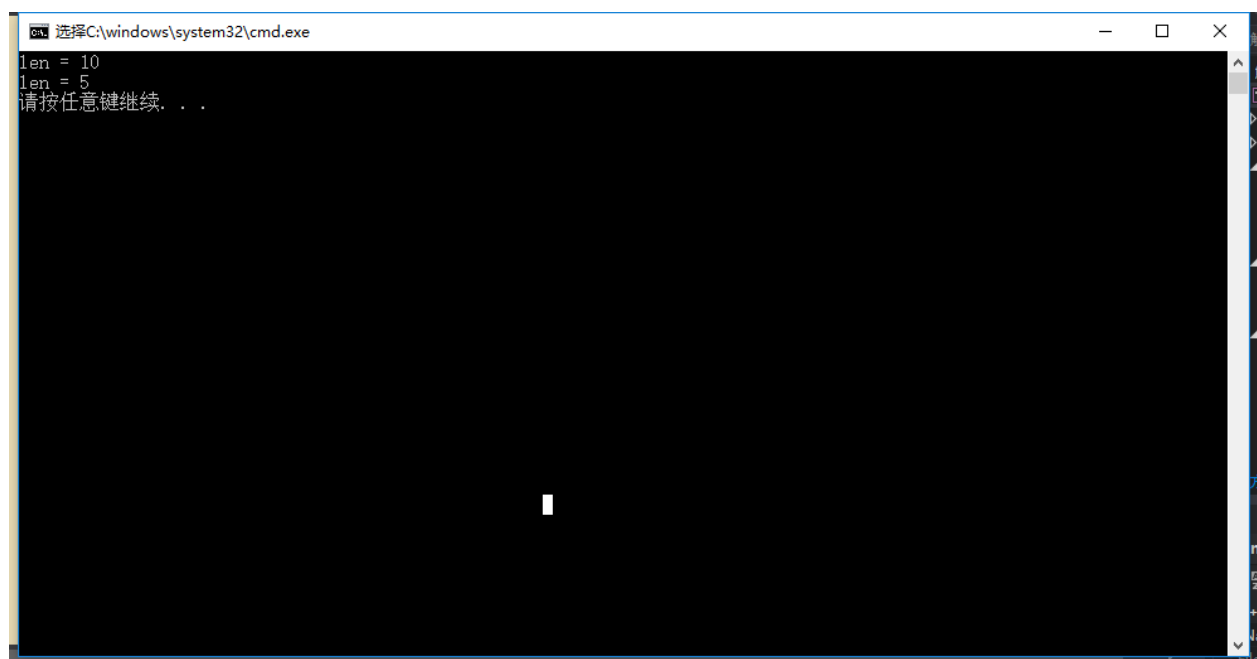
#include "Array.h"
#include <iostream>
#include <stdlib.h>
using namespace std;

int main()
{
    Array arr1(10);
    arr1.printInfo().setLen(5).printInfo();//做出更改

    system("pause");
    return 0;
}

```

运行结果为：



```

len = 10
len = 5
请按任意键继续. . .

```

arr1.printInfo().setLen(5).printInfo()这里首先用printInfo()打印出arr1中的值，然后通过setLen(5)将arr1中的值设置为5，最后再用printInfo()打印出arr1中的值。因为每次都使用的是Array的引用（即Array&），所以实际就是在操作arr1这个对象。

如果return的不是引用，而是指针，那又应该如何去写呢？ Array.h

```

class Array
{
public:
    Array(int len);
    ~Array();
    Array* setLen(int len);//做出更改
    int getLen();
    Array* printInfo(); //做出更改
private:

```

```
    int len;  
};
```

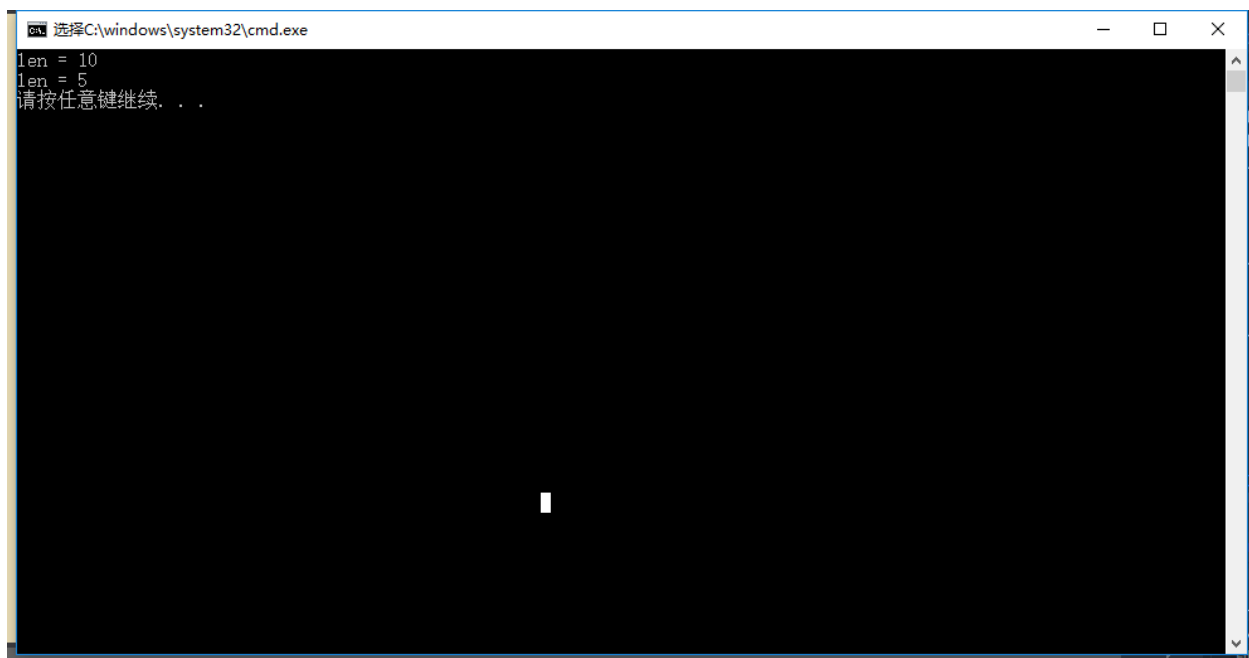
Array.cpp

```
#include <iostream>  
#include "Array.h"  
using namespace std;  
  
Array::Array(int len)  
{  
    this->len = len;  
}  
  
Array::~Array()  
{  
  
}  
  
Array* Array::setLen(int len)//做出更改  
{  
    this->len = len;  
    return this;//做出更改(因为要求的就是返回指针，this是指针，*this是对象)  
}  
  
int Array::getLen()  
{  
    return len;  
}  
  
Array* Array::printInfo()    //做出更改  
{  
    cout << "len = " << len << endl;  
    return this;//做出更改  
}
```

demo.cpp

```
#include "Array.h"  
#include <iostream>  
#include <stdlib.h>  
using namespace std;  
  
int main()  
{  
    Array arr1(10);  
    arr1.printInfo()->setLen(5)->printInfo();//做出更改(因为是指针，所以用“->”来访问)  
  
    system("pause");  
    return 0;  
}
```

运行结果为：



这也证明了，无论是通过指针还是引用，都可以改变实际的值。

通过代码来说明this本身的值(它就相当于它所在的对象的地址) Array.h

```
class Array
{
public:
    Array(int len);
    ~Array();
    Array* setLen(int len);
    int getLen();
    Array* printInfo();
private:
    int len;
};
```

Array.cpp

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

Array::Array(int len)
{
    this->len = len;
}

Array::~Array()
{
}

Array* Array::setLen(int len)
{
    this->len = len;
    return this;
}

int Array::getLen()
{
    return len;
}

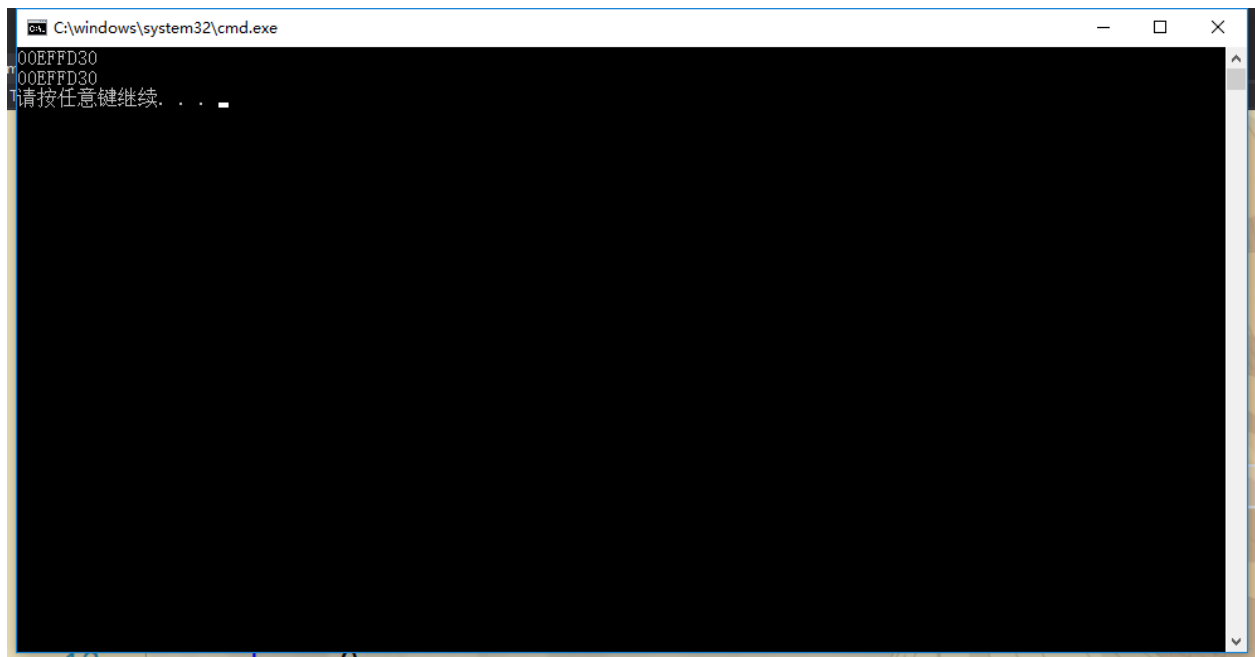
Array* Array::printInfo()
{
    cout << this << endl; //做出更改(打印this的地址)
```

```
        return this;  
    }  
}
```

demo.cpp

```
#include "Array.h"  
#include <iostream>  
#include <stdlib.h>  
using namespace std;  
  
int main()  
{  
    Array arr1(10);  
    arr1.printInfo();//做出更改, 打印this的值  
    cout << &arr1 << endl;//打印arr1的地址  
  
    system("pause");  
    return 0;  
}
```

运行结果为：



```
C:\windows\system32\cmd.exe  
00EFFF30  
00EFFF30  
请按任意键继续. . .
```