对象与函数

范 懿

目录

```
1  // 把对象传入函数
2  #include <iostream>
3  using namespace std;
4  class MyClass {
5          int i;
6      public:
7          MyClass(int n);
8          ~MyClass();
9          void set_i(int n) { i = n; }
10         int get_i() { return i; }
11 };
12
13 MyClass::MyClass(int n)
14 {
15     i = n;
16     cout << "Constructing " << i << "\n";
17 }
18
19 Myclass::~Myclass()
20 {
21     cout << "Destroying " << i << "\n";
```

```
22 }
23
24 void f(MyClass ob);
25 int main()
26 {
27     MyClass o(1);
28     f(o);
29     cout << "This is i in main: ";
30     cout << o.get_i() << "\n";
31     return 0;
32 }
33 void f(MyClass ob)
34 {
35     ob.set_i(2);
36     cout << "This is local i: " << ob.get_i();
37     cout << "\n";
38 }
39 /*示例输出
40 Constructing 1
41 This is local i: 2
42 Destroying 2
```

```
This is i in main: 1
Destroying 1*/
/*构造函数执行1次，析构函数执行2次。*/
```

拷贝构造函数

函数机制

- 当一个对象被传入函数时，函数内部会创建它的一个副本
- 函数结束时，这个副本会被销毁

- 当这个副本被创建的时候，是否会启用构造函数?
- 当这个副本被销毁的时候，是否会启用析构函数?

☞ 当函数调用需要创建一个副本时，构造函数不会被启用。

实际上，拷贝构造函数（Copy Constructor）会被启用。

拷贝构造函数说明了如何生成对象的一个副本。

如果拷贝构造函数没有被定义，函数将按位一模一样进行复制。

构造函数只用于初始化

因为构造函数是用于**初始化**对象的，它不能被用于**复制**对象。

如果启用构造函数，那将可能修改对象。

我们传入对象时，是希望使用它**现在的**状态，而非**初始的**状态。

当对象离开生存域（Scope）而被销毁时，我们需调用析构函数。

☞ `f()` 中的对象首先失效，然后 `main()` 中的对象才失效。

总结前面的程序

- 在函数创建传入值的副本时，通常的构造函数不会被调用
- 而默认的拷贝构造函数会简单地逐个逐个位进行简单复制。

当一个副本因为离开生存域而被销毁时，析构函数会被启用。

因为默认的拷贝构造函数会一模一样复制，这可能会导致麻烦。

即使以传值方式进行，用于产生传入值的对象依然可能被破坏。

例

如果一个对象被销毁时回收内存，那么，它在函数中的副本将会回收同一片内存——从而破坏了函数外的对象。

```cpp
// 函数返回对象
#include <iostream>
using namespace std;
class MyClass {
        int i;
    public:
        void set_i(int n) { i = n; }
        int get_i() { return i; }
};
MyClass f(); // 返回MyClass类型的对象
int main()
{
    MyClass o;
    o = f(); // f()返回的对象，在赋值给o之后，将被销毁
    cout << o.get_i() << "\n";
    return 0;
}
MyClass f()
{
    MyClass x;
```

```
21      x.set_i(1);
22      return x;
23  }
```

机制

- 当一个对象被返回的时候，它的一个临时副本被产生。
- 实际上函数返回的是这个副本。
- 当这个副本完成任务（比如赋值）后，它将被销毁。
- 这个临时副本在销毁中可能导致问题。

例

如果临时返回的副本的析构函数会释放内存，那么，

- 相应的内存会被释放，
- 即使接收这个副本的对象依然在使用相应空间。

两种解决方案

- 重载赋值运算符
- 使用拷贝构造函数

```cpp
// 对象赋值
#include <iostream>
using namespace std;
class MyClass {
        int i;
    public:
        void set_i(int n) { i=n; }
        int get_i() { return i; }
};
int main()
{
    MyClass ob1, ob2;
    ob1.set_i(99);
    ob2 = ob1; // 把ob1的数据赋给ob2
    cout << "This is ob2's i: " << ob2.get_i();
    return 0;
}
```

所有 ob1 的数据会原样赋值到 ob2，不过，也可以重载赋值符，自定义操作。