

三、C++学习笔记—核心编程

本阶段，将对C++面向对象编程技术做详细学习，深入C++中的核心和精髓

3.4.5.3 递增运算符重载

作用：通过重载递增运算符，实现自己的整型数据

```
1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //递增运算符重载
6
7  //自定义整型
8  class MyInteger
9  {
10     friend ostream & operator<<(ostream & cout, MyInteger myint);
11 public:
12
13     MyInteger()
14     {
15         m_Num = 0;
16     }
17
18     //重载前置++运算符 返回引用是为了一直对一个数据进行递增操作
19     MyInteger& operator++()
20     {
21         //先进行++运算
22         m_Num++;
23         //再将自身做一个返回
24         return *this;
25     }
26
27     //重载后置++运算符
28     //void operator++(int) int代表占位参数，可以用于区分前置和后置递增
29     MyInteger operator++(int)
30     {
31         //先 记录当时结果
32         MyInteger temp = *this;
33         //后 递增
34         m_Num++;
35         //最后 将记录结果做返回
36         return temp;
37     }
38
39
40 private:
```

```

41
42     int m_Num;
43
44 };
45
46
47 //重载左移运算符
48 ostream & operator<<(ostream & cout, MyInteger myint)
49 {
50     cout << myint.m_Num;
51     return cout;
52 }
53
54
55 void test01()
56 {
57     MyInteger myint;
58     cout << ++(++myint) << endl;
59     cout << myint << endl;
60
61 }
62
63 void test02()
64 {
65     MyInteger myint;
66
67     cout << myint++ << endl;
68     cout << myint << endl;
69 }
70
71 int main()
72 {
73
74     //test01();
75     test02();
76     //int a = 0;
77     //cout << a << endl;
78     system("pause");
79     return 0;
80 }

```

前置递减和后置递减

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //递减运算符重载
6
7  class MyInter
8  {
9      friend ostream & operator<<(ostream & cout, MyInter myint);
10 public:

```

```

11     MyInter()
12     {
13         m_Num = 5;
14     }
15
16     //重载前置--运算符 返回引用是为了一直对一个数据进行递减操作
17     MyInter & operator--()
18     {
19         //先--运算
20         m_Num--;
21         return *this;
22     }
23
24     //重载后置--运算符
25     //void operator--(int) int代表占位参数, 可以用于区分前置和后置递减
26     MyInter operator--(int)
27     {
28         //先 记录当前结果
29         MyInter temp = *this;
30         //后 递减
31         m_Num--;
32         //最后 将记录结构做返回
33         return temp;
34     }
35
36
37 private:
38     int m_Num;
39 };
40
41 //重载左移运算符
42 ostream & operator<<(ostream & cout, MyInter myint)
43 {
44     cout << myint.m_Num;
45     return cout;
46 }
47
48 void test01()
49 {
50     MyInter myint;
51     cout << myint-- << endl;
52     cout << myint << endl;
53
54 }
55
56 void test02()
57 {
58     MyInter myint;
59     cout << --(--myint) << endl;
60     cout << myint << endl;
61
62 }
63

```

```

64
65 int main()
66 {
67
68     //test01();
69     test02();
70     //int a = 0;
71     //cout << a << endl;
72     system("pause");
73     return 0;
74 }

```

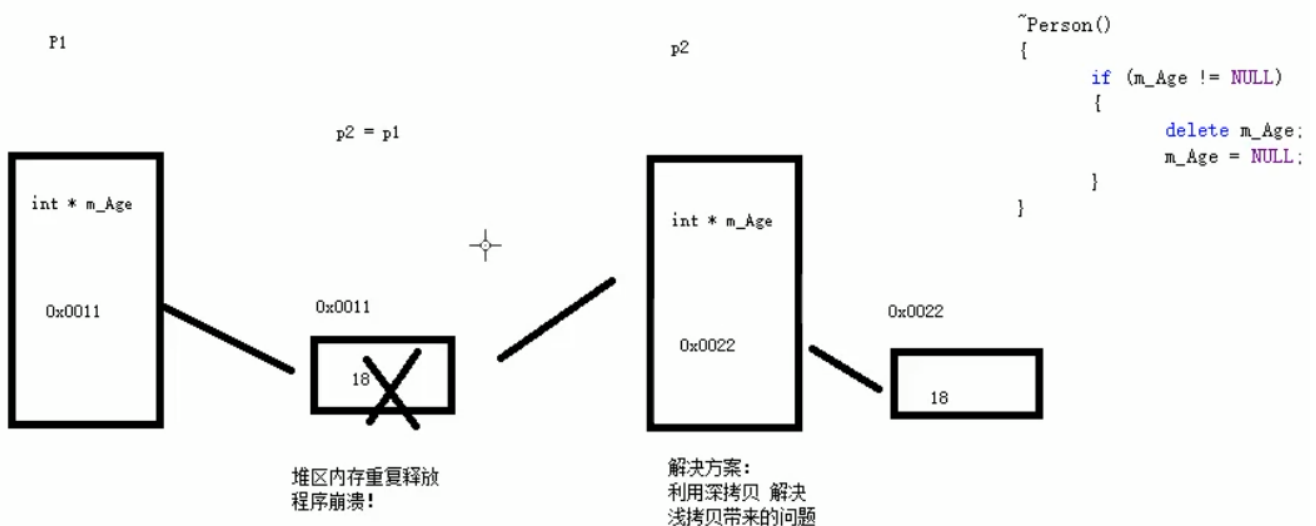
总结：前置递增返回引用，后置递增返回值

3.4.5.4 赋值运算符重载

c++编译器至少给一个类添加4个函数

1. 默认构造函数(无参，函数体为空)
2. 默认析构函数(无参，函数体为空)
3. 默认拷贝构造函数，对属性进行值拷贝
4. 赋值运算符 operator=, 对属性进行值拷贝

如果类中有属性指向堆区，做赋值操作时也会出现深浅拷贝问题



示例：

```

1 #include<iostream>
2 #include<string>
3 using namespace std;
4
5 //赋值运算符重载

```

```

6  class Person
7  {
8  public:
9      Person(int age)
10     {
11         m_Age = new int(age); //年龄创建了一个堆区
12     }
13
14     ~Person() //析构函数-配合堆区
15     {
16         if (m_Age != NULL)
17         {
18             delete m_Age;
19             m_Age = NULL; //以防野指针
20         }
21     }
22
23     //重载 赋值运算符
24     Person& operator=(Person &p)
25     {
26         //编译器是提供的浅拷贝
27         //m_Age = p.m_Age;
28
29         //应该先判断是否有属性在堆区，如果有先释放赶紧，然后在深拷贝
30         if (m_Age!=NULL)
31         {
32             delete m_Age;
33             m_Age = NULL;
34         }
35
36         //深拷贝
37         m_Age = new int(*p.m_Age);
38         //返回对象本身
39         return *this;
40     }
41
42     int *m_Age;
43 };
44
45 void test01()
46 {
47     Person p1(18);
48     Person p2(20);
49     Person p3(30);
50     p3 = p2 = p1; //赋值操作
51
52     cout << "p1的年龄为: " << *p1.m_Age << endl;
53     cout << "p2的年龄为: " << *p2.m_Age << endl;
54     cout << "p3的年龄为: " << *p3.m_Age << endl;
55
56 }
57
58 int main()

```

```

59 {
60
61     test01();
62     //test02();
63     //int a = 0;
64     //cout << a << endl;
65     system("pause");
66     return 0;
67 }

```

3.4.5.5 关系运算符重载

作用：重载关系运算符，可以让两个自定义类型对象进行对比操作

示例：

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4
5  //关系运算符重载
6  class Person
7  {
8
9
10 public:
11
12     Person(string name, int age) //有参构造函数
13     {
14         m_Name = name;
15         m_Age = age;
16     }
17
18     //重载==号
19     bool operator==(Person &p)
20     {
21         if (this->m_Name == p.m_Name && this->m_Age == p.m_Age)
22         {
23             return true;
24         }
25         return false;
26     }
27
28     //重载!=号
29     bool operator!=(Person &p)
30     {
31         if (this->m_Name == p.m_Name && this->m_Age == p.m_Age)
32         {
33             return false;
34         }
35         return true;

```

```

36     }
37
38     string m_Name;
39     int m_Age;
40 };
41
42 void test01()
43 {
44     Person p1("张珊", 23);
45     Person p2("张三", 23);
46     if (p1 == p2)
47     {
48         cout << "p1和p2是相等的" << endl;
49     }
50     else
51     {
52         cout << "p1和p2是不相等的" << endl;
53     }
54
55     if (p1 != p2)
56     {
57         cout << "p1和p2是不相等的" << endl;
58     }
59     else
60     {
61         cout << "p1和p2是不相等的" << endl;
62     }
63
64 }
65
66 int main()
67 {
68
69     test01();
70
71     system("pause");
72     return 0;
73 }

```

3.4.5.6 函数调用运算符重载

- 函数调用运算符 () 也可以重载
- 由于重载后使用的方式非常像函数的调用，因此称为仿函数
- 仿函数没有固定写法，非常灵活

示例：

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4

```

```

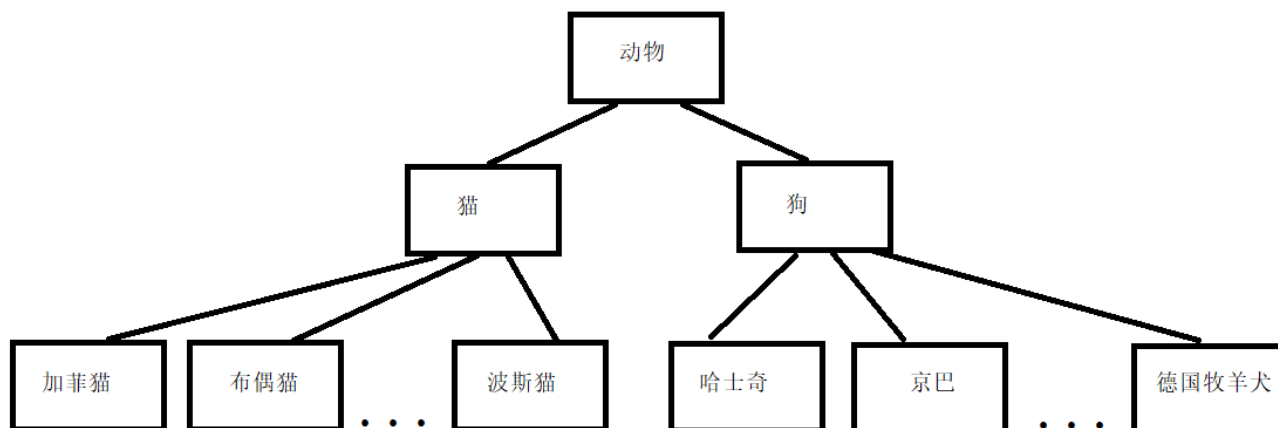
5 //函数调用运算符重载
6
7 //打印输出类
8 class Myprint
9 {
10 public:
11
12     //重载函数调用运算符
13     void operator()(string test)
14     {
15         cout << test << endl;
16     }
17
18 };
19
20 //仿函数非常灵活，没有一个固定的写法
21 //加法类
22 class MyAdd
23 {
24
25 public:
26     int operator()(int num1, int num2)
27     {
28         return num1 + num2;
29     }
30 };
31
32
33 void test02()
34 {
35     MyAdd myadd;
36     int ret = myadd(100, 100);
37     cout << "ret=" << ret << endl;
38
39     //匿名函数对象
40     cout << MyAdd()(100, 100) << endl;
41 }
42
43
44 void test01()
45 {
46     Myprint myPrint;
47     myPrint("hello world"); //由于使用起来非常类似于函数调用，因此称为仿函数
48
49 }
50
51 int main()
52 {
53     test01();
54     test02();
55     system("pause");
56     return 0;
57 }

```


3.4.6 继承

继承是面向对象三大特性之一

有些类与类之间存在特殊的关系，例如下图中：



我们发现，定义这些类时，下级别的成员除了拥有上一级的共性，还有自己的特性。

这个时候我们就可以考虑利用继承的技术，减少重复代码

3.4.6.1 继承的基本语法

例如我们看到很多网站中，都有公共的头部，公共的底部，甚至公共的左侧列表，只有中心内容不同。

接下来我们分别利用普通写法和继承的写法来实现网页中的内容，看一下继承存在的意义以及好处

普通实现：

```
1 //Java页面
2 class Java
3 {
4     public:
5         void header()
6         {
7             cout << "首页、公开课、登录、注册... (公共头部)" << endl;
8         }
9         void footer()
10        {
11            cout << "帮助中心、交流合作、站内地图... (公共底部)" << endl;
12        }
13        void left()
14        {
15            cout << "Java,Python,C++... (公共分类列表)" << endl;
16        }
17        void content()
18        {
```

```

19     cout << "JAVA学科视频" << endl;
20 }
21 };
22 //Python页面
23 class Python
24 {
25 public:
26     void header()
27     {
28         cout << "首页、公开课、登录、注册... (公共头部) " << endl;
29     }
30     void footer()
31     {
32         cout << "帮助中心、交流合作、站内地图...(公共底部)" << endl;
33     }
34     void left()
35     {
36         cout << "Java,Python,C++...(公共分类列表)" << endl;
37     }
38     void content()
39     {
40         cout << "Python学科视频" << endl;
41     }
42 };
43 //C++页面
44 class CPP
45 {
46 public:
47     void header()
48     {
49         cout << "首页、公开课、登录、注册... (公共头部) " << endl;
50     }
51     void footer()
52     {
53         cout << "帮助中心、交流合作、站内地图...(公共底部)" << endl;
54     }
55     void left()
56     {
57         cout << "Java,Python,C++...(公共分类列表)" << endl;
58     }
59     void content()
60     {
61         cout << "C++学科视频" << endl;
62     }
63 };
64
65 void test01()
66 {
67     //Java页面
68     cout << "Java下载视频页面如下: " << endl;
69     Java ja;
70     ja.header();
71     ja.footer();

```

```

72     ja.left();
73     ja.content();
74     cout << "-----" << endl;
75
76     //Python页面
77     cout << "Python下载视频页面如下: " << endl;
78     Python py;
79     py.header();
80     py.footer();
81     py.left();
82     py.content();
83     cout << "-----" << endl;
84
85     //C++页面
86     cout << "C++下载视频页面如下: " << endl;
87     CPP cp;
88     cp.header();
89     cp.footer();
90     cp.left();
91     cp.content();
92
93 }
94
95 int main() {
96
97     test01();
98
99     system("pause");
100
101     return 0;
102 }

```

继承实现:

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4  //继承实现页面
5  //公共页面类
6
7  class BasePage
8  {
9  public:
10     void header()
11     {
12         cout << "首页, 公共课, 登录, 注册... (公共头部) " << endl;
13     }
14     void footer()
15     {
16         cout << "帮助中心, 交流合作, 站内地图... (公共地步) " << endl;
17     }
18     void left()
19     {

```

```

20         cout << "java,python,c++... (公共分类列表) " << endl;
21     }
22
23 };
24
25
26 //继承的好处: 减少重复代码
27 //语法: class 子类: 继承方式 父类
28 //子类: 也称为 派生类
29 //父类: 也称为 基类
30 //java页面
31 class Java : public BasePage
32 {
33 public:
34     void content()
35     {
36         cout << "java学科视频" << endl;
37     }
38
39 };
40
41 //python页面
42 class Python : public BasePage
43 {
44 public:
45     void content()
46     {
47         cout << "PYTHON学科视频" << endl;
48     }
49 };
50
51 //c++页面
52 class CPP : public BasePage
53 {
54 public:
55     void content()
56     {
57         cout << "CPP学科视频" << endl;
58     }
59 };
60
61 2
62 void test01()
63 {
64     cout << "java下载视频页面如下: " << endl;
65     Java ja;
66     ja.header();
67     ja.footer();
68     ja.left();
69     ja.content();
70
71     cout << "-----" << endl;
72     cout << "Python下载视频页面如下: " << endl;

```

```

73     Python py;
74     py.header();
75     py.footer();
76     py.left();
77     py.content();
78
79     cout << "-----" << endl;
80     cout << "C++下载视频页面如下: " << endl;
81     CPP cpp;
82     cpp.header();
83     cpp.footer();
84     cpp.left();
85     cpp.content();
86
87 }
88
89 int main()
90 {
91     test01();
92     system("pause");
93     return 0;
94 }

```

总结:

继承的好处: 可以减少重复的代码

class A : public B;

A 类称为子类 或 派生类

B 类称为父类 或 基类

派生类中的成员, 包含两大部分:

一类是从基类继承过来的, 一类是自己增加的成员。

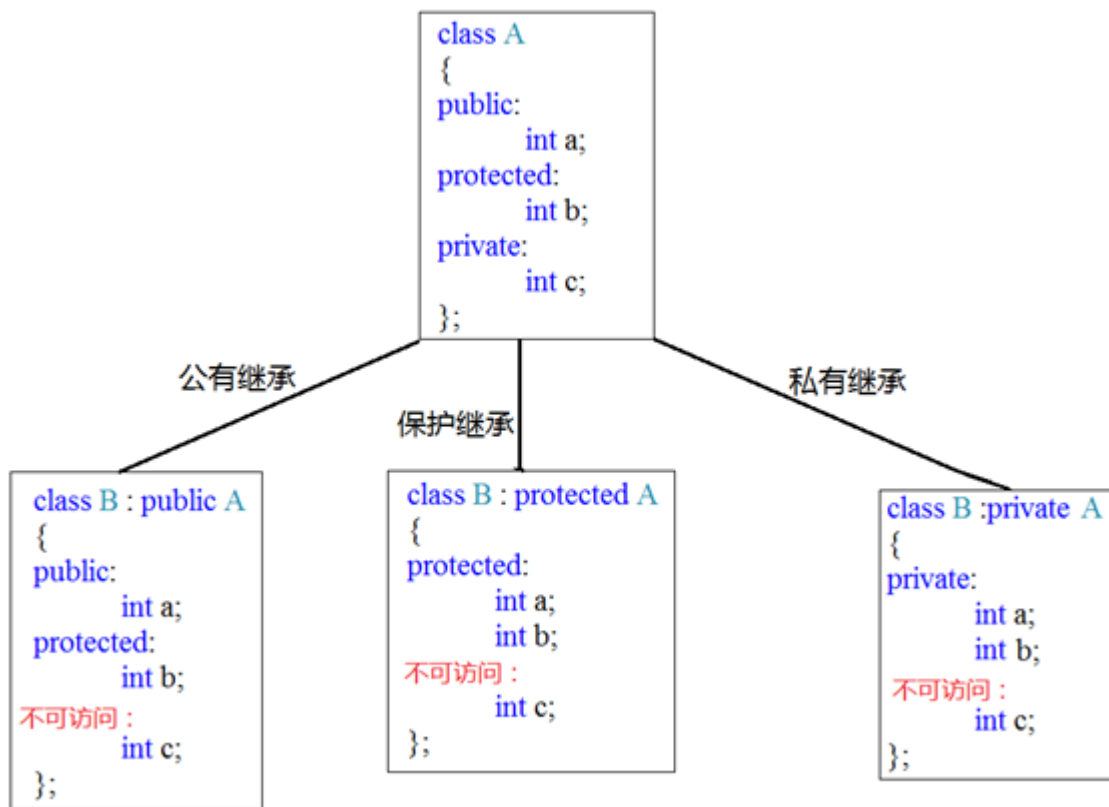
从基类继承过来的表现其共性, 而新增的成员体现了其个性。

3.4.6.2 继承方式

继承的语法: class 子类 : 继承方式 父类

继承方式一共有三种:

- 公共继承
- 保护继承
- 私有继承



示例:

```

1  #include<iostream>
2  #include<string>
3  using namespace std;
4  //继承方式
5
6  //公共继承
7  class Base1
8  {
9  public:
10     int m_A;
11 protected:
12     int m_B;
13 private:
14     int m_C;
15
16 };
17
18 class Son1 : public Base1
19 {
20 public:
21     void func()
22     {
23         m_A = 10; //父类中的公共权限成员 到子类中依然是公共权限
24         m_B = 10; //父类中的包含权限成员 到子类中依然是保护权限
25         //m_C = 10; //父类中的私有权限成员 子类访问不到
26     }

```

```
27
28 };
29
30
31
32 //测试案例
33 void test01()
34 {
35     Son1 s1; //子类对象
36     s1.m_A = 100;
37     //s1.m_B = 100; //到Son1中, m_B是保护权限, 类外是访问不到的
38
39 }
40
41 //保护继承
42 class Base2
43 {
44 public:
45     int m_A;
46 protected:
47     int m_B;
48 private:
49     int m_C;
50
51 };
52
53 class Son2 : protected Base2
54 {
55 public:
56     void func()
57     {
58         m_A = 100; // 父类中的公共成员, 到子类中变为保护权限
59         m_B = 100; // 父类中的保护成员, 到子类中变为保护权限
60         //m_C = 100; //父类中私有成员, 子类是访问不到
61     }
62 };
63
64 void test02()
65 {
66     Son2 s1;
67     //s1.m_A = 1000; //在Son2中, m_A变为保护权限, 因此类外访问不到
68     //s1.m_B = 1000; //在Son2中, m_B变为保护权限, 因此类外访问不到
69 }
70
71 //私有继承
72 class Base3
73 {
74 public:
75     int m_A;
76 protected:
77     int m_B;
78 private:
79     int m_C;
```

```
80
81 };
82
83 class Son3 : private Base3
84 {
85 public:
86     void func()
87     {
88         m_A = 100; //父类中的公共成员，到子类变成私有成员
89         m_B = 100; //父类中保护成员，到子类变成私有成员
90         //m_C = 100; //父类中私有成员，子类访问不到
91     }
92 };
93
94 class GrandSon3 : public Son3
95 {
96 public:
97     void func()
98     {
99         //m_A = 1000; //到了Son3中 m_A变成私有，即时是儿子，也是访问不到
100        //m_B = 1000;
101    }
102
103 };
104
105 void test03()
106 {
107     Son3 s1;
108     //s1.m_A = 1000; // 到Son3中，变成 私有成员类外访问不到
109     //s1.m_B = 1000; // 到Son3中，变成 私有成员类外访问不到
110 }
111
112
113 int main()
114 {
115
116     system("pause");
117     return 0;
118 }
```