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

3.4.2.7 类对象作为类成员

C++类中的成员可以是另一个类的对象，我们称该成员为 对象成员

例如：

```
class A{}
class B
{
    A a;
}
```

B类中有对象A作为成员，A为对象成员

那么当创建B对象时，A与B的构造和析构的顺序是谁先谁后？

示例：

```
#include<iostream>
#include<string>
using namespace std;

//类对象作为类成员
//手机类
class Phone
{
public:

    Phone(string pName)
    {
        cout << "Phone的有参构造函数" << endl;
        m_PName = pName;
    }

    ~Phone()
    {
        cout << "Phone的析构函数调用" << endl;
    }
    //手机品牌名称
    string m_PName;



};
```

```
27
28  //人类
29  class Person
30  {
31
32  public:
33
34      // 初始化列表: Phone m_Phone = pName; 隐式转换法
35      Person(string name, string pName): m_Name(name),m_Phone(pName)
36      {
37          cout << "Person的函数构造" << endl;
38
39      }
40      //析构函数
41      ~Person()
42      {
43          cout << "Person的析构函数调用" << endl;
44      }
45
46      //姓名
47      string m_Name;
48      //手机
49      Phone m_Phone;
50  };
51
52  //当其他类对象作为本类成员，构造时候先构造类对象，早构造自身，析构的顺序与构造相反
53  void test01()
54  {
55      Person p("张三", "苹果11");
56      cout << p.m_Name << "拿着: " << p.m_Phone.m_PName << endl;
57
58  }
59
60  int main()
61  {
62
63      test01();
64      system("pause");
65      return 0;
66  }
```

> 结论： 当其他类对象作为本类成员，构造时候先构造类对象，早构造自身，析构的顺序与构造相反

3.4.2.8 静态成员

静态成员就是在成员变量和成员函数前加上关键字static，称为静态成员

静态成员分为:

- 静态成员变量
    - 所有对象共享同一份数据
    - 在编译阶段分配内存

    - 类内声明，类外初始化
- 静态成员函数
    - 所有对象共享同一个函数
    - 静态成员函数只能访问静态成员变量

示例1： 静态成员变量

```
#include<iostream>
#include<string>
using namespace std;

//静态成员函数
//所有对象 共享同一个函数
//静态成员变量只能访问静态成员变量
class Person
{
public:

    //静态成员变量
    static void func()
    {
        m_A = 100; //静态成员函数可以访问 静态成员变量
        //m_B = 200; // 静态成员函数 不可以访问 非静态成员变量，无法区分到底是那个对象的m_B的属性
        cout << "static void func调用" << endl;
    }

    static int m_A; //静态成员变量
    int m_B; //非静态成员变量

    //静态成员函数也是有访问权限的
private:
    static void func2()
    {
        cout << "static void func2调用" << endl;
    }

};

int Person::m_A = 0; //
//有两种访问方式
void test01()
{
    //1、通过对象访问
    Person p;
    p.func();
    //2、通过类名访问
    Person::func();

    //Person::func2();//类外访问不到私有静态成员函数

}


```

```cpp
int main()
{

    test01();
    system("pause");
    return 0;
}
```

3.4.3 C++对象模型和this指针

3.4.3.1 成员变量和成员函数分开存储

在C++中，类内的成员变量和成员函数分开存储

只有非静态成员变量才属于类的对象上

```cpp
#include<iostream>
#include<string>
using namespace std;

//成员变量 和 成员函数  分开存储的
class Person
{
public:
    int m_A; //非静态成员变量,属于这个类上面的
    static int m_B; // 静态成员变量，不属于类对象上
    void func() {} //非静态成员函数，不属于类对象上

    static void fun2() {} //非静态成员函数，不属于类对象上

};

int Person::m_B = 0; //静态成员变量需要初始化


void test01()
{
    Person p; //通过对象访问
    //空对象占用内存空间为：1
    //c++编译器会给每个空对象也分配一个字节空间，是为了区分空对象占内存的位置
    //每个空对象也应该有一个独一无二的内存地址
    cout << "size of p=" <<sizeof(p)<< endl;
}

void test02()
{
    Person p;
    cout << "size of p=" << sizeof(p) << endl;
}


```

```
int main()
{

    test01();
    test02();
    system("pause");
    return 0;
}
```

3.4.3.2 this指针概念

通过3.4.3.1我们知道在C++中成员变量和成员函数是分开存储的

每一个非静态成员函数只会诞生一份函数实例，也就是说多个同类型的对象会共用一块代码

那么问题是：这一块代码是如何区分那个对象调用自己的呢？

---

c++通过提供特殊的对象指针，this指针，解决上述问题。**this指针指向被调用的成员函数所属的对象**

this指针是隐含每一个非静态成员函数内的一种指针

this指针不需要定义，直接使用即可

---

this指针的用途：

- 当形参和成员变量同名时，可用this指针来区分
- 在类的非静态成员函数中返回对象本身，可使用return *this

```
#include<iostream>
#include<string>
using namespace std;


class Person
{
public:

    //类有参函数构造
    Person(int age)
    {
        //this指针指向的是 被调用的成员函数(p1) 所属的对象
        this->age = age;

    }

    Person& PersonAddAge(Person &p) //引用方式传入
    {
        this->age += p.age;
        //this 指向p2的指针，而*this指向的就是p2这个对象本体
        return *this; //返回引用 Person&, 返回值 Person
```

```
    }

public:
    //int m_Aage; //m代表member成员的意思
    int age;

};

//1、解决名称冲突
void test01()
{
    Person p1(18); //类对象调用
    cout << "p1的年龄: " << p1.age << endl;
}

//2、返回对象本身用*this
void test02()
{
    Person p1(10);

    Person p2(10);

    //链式编程思想
    p2.PersonAddAge(p1).PersonAddAge(p1).PersonAddAge(p1);

    cout << "p2的年龄为: " << p2.age << endl;
}

int main()
{

    test01();
    test02();
    system("pause");
    return 0;
}
```

3.4.3.3 空指针访问成员函数

C++中空指针也是可以调用成员函数的，但是也要注意有没有用到this指针

如果用到this指针，需要加以判断保证代码的健壮性

示例：

```
#include<iostream>
#include<string>
using namespace std;


//空指针调用成员函数
class Person
```

```
{

public:
    void showClassName()
    {
        cout << "this is Person Class" << endl;
    }

    void showPersonAge()
    {
        //报错因为传入的指针为NULL
        if (this==NULL)
        {
            return;
        }
        cout << "age=" << m_Age << endl;
    }

public:
    int m_Age;
};

void test01()
{
    Person *p=NULL; //指针指向空
    p->showClassName();
    p->showPersonAge();
}

int main()
{

    test01();
    //test02();
    system("pause");
    return 0;
}
```

3.4.3.4 const修饰成员函数

常函数：

- 成员函数后加const后我们称为这个函数为**常函数**
- 常函数内不可以修改成员属性
- 成员属性声明时加关键字mutable后，在常函数中依然可以修改

常对象：

- 声明对象前加const称该对象为常对象
- 常对象只能调用常函数

示例：

```cpp
#include<iostream>
#include<string>
using namespace std;


//常函数
class Person
{
public:

    //this指针的本质  是指针常量，  指针的指向是不可以修改的
    //const Person * const this;
    //在成员函数后面加const，  修饰的是this指向，让指针指向的值也不可以修改
    void showPerson() const  //常函数
    {
        this->m_B = 100;
        //this->m_Age = 100;
        //this = NULL; // this指针是不可以修改指针指向的
    }

    void func()
    {
        m_Age = 100;

    }

    int m_Age;
    mutable int m_B; //特殊变量，即使在常函数中，也可以修改这个值,加关键字mutable
};

//常对象
void test01()
{
    const Person p; // 在对象前面const，变为常对象
    //p.m_Age = 100;
    p.m_B = 100; //m_B是特殊值，在常对象下也可以修改

    //常对象只能调用常函数
    p.showPerson();
    //p.func(); //常对象  不可以调用普通成员函数，因为普通成员函数可以修改属性

}

void test01()
{
    Person p;
    p.showPerson();
}
int main()
{

    test01();
    //test02();
```

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