第9章-拷贝构造

author：岳石磊 copyright：科林明伦 内部资料禁止外泄

1. 转换构造函数

一个类中构造函数可以重载，允许多个存在。如果构造函数只有一个参数且非当前类对象时，可以将其他类型自动转换为当前类类型，这个过程为隐式类型转换。如下示例中的三个构造函数都可以发生隐式类型转换，在使用等号初始化或赋值时自动转换。
这样的可以发生隐式类型转换的构造函数可以称之为：**转换构造函数**。

```
class CTest{
    CTest(int a){}              //转换构造函数
    /*
    CTest(int a,int b =0){}     //转换构造函数
    CTest(int a=0,int b =0){}  //转换构造函数
    explicit CTest(int a=0,int b =0){}//禁止发生隐式类型转换
    CTest(int a,int b ){}  //不是转换构造函数
    */
};

CTest tst(10);  //调用带参数的构造
CTest tst2 = 20;  //合法操作 发生隐式类型转换,将int类型转换为CTest类型
tst2 = 30;        //合法操作 发生隐式类型转换
```

注意：如果是多个参数且无默认值时，则不能自动隐式类型转换。如果想要避免隐式类型转换，在构造函数前加上 关键字：explicit。

2. 拷贝构造函数

它也是编译器默认给提供的一个特殊的构造函数，在空类中它与默认的无参构造并存，拷贝构造函数是众多构造函数中的一种
参数为当前类对象的引用。与默认无参构造不同，其函数体代码一般不为空，操作为：参数中对象成员依次给this对象成员进行初始化。

```
class CTest {
    int m_a;
    //默认的拷贝构造函数一般长这个样子，对于初始化代码下面两种写法都可以
    CTest(const CTest & tst):m_a(tst.m_a) {
        //this->m_a = tst.m_a;

```

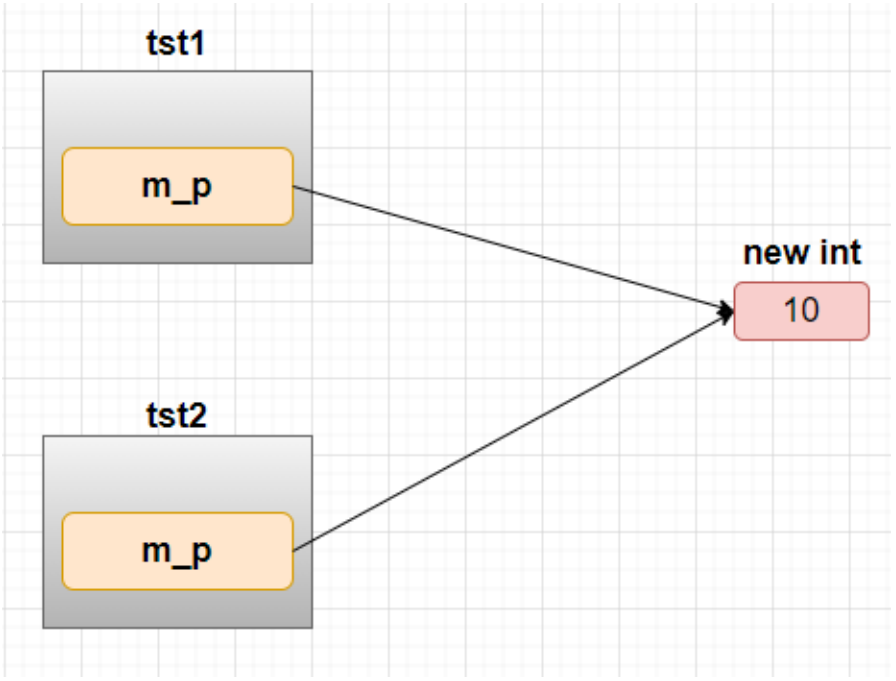
```
    }
};
```

当我们手动重构拷贝构造函数时，编译器就不会提供默认的拷贝构造函数了，当然也不会存在默认的无参构造了。
当用一个类对象给类的另一个对象初始化时，会调用拷贝构造函数。

```
class CTest{
    CTest();
    CTest(const CTest &tst);
};
CTest tst1;         //调用无参构造
CTest tst2(tst1);   //调用拷贝构造
```

默认拷贝构造函数是一个浅拷贝，当类中存在指针成员且指向了一个new出来的具体空间，拷贝构造函数只是将两个指针里存储的地址进行拷贝，并不会处理指针指向的空间。这样就导致了多个对象里的指针指向了同一个空间，那么会导致以下两个问题：

1. 当其中一个对象通过指针修改其指向空间的值，那么其他对象再使用就是修改之后的值了，这样的情况多数不是我们预期的。
2. 如果是new出来的空间，那么对导致多个对象回收同一块内存空间，引起非法操作错误。



解决办法： **深拷贝**，它并不是一个固定的写法，而是一个解决的办法：即在拷贝构造时，如果参数对象中的指针成员指向了一个内存空间，那么在重构拷贝构造时，需要为当前this对象中指针成员额外开辟新的内存空间，并初始化对应的值。

```
class CTest {
    int *m_p;
    CTest(const CTest & tst) {
```

```
        //深拷贝
        if (tst.m_p)
            m_p = new int(*tst.m_p);
        else
            m_p = nullptr;
    }
};
```

在某些情况下，可以使用**指针或引用**可以避免对象的值传递，也避免了浅拷贝问题。

```
void fun(CTest tst);  //避免值传递
void fun(CTest & tst);// or void fun(CTest *ptst);
```

3. 默认operator=

空类中编译器也会默认提供一个operator=函数，参数和 返回值 为当前类对象的引用，如果我们手动重构了编译器也就没必要为我们提供了。
当用一个类对象给类的另一个对象赋值时，会调用默认的operator=函数。

```
class CTest {
public:
    int m_a;
    CTest& operator=(const CTest& tst){
        this->m_a = tst.m_a;
    }
};

CTest tst1;
CTest tst2;
tst2 = tst1;  //匹配operator =
```

默认的operator=函数的函数体代码不为空，参数中对象成员依次给this对象成员进行赋值。

拷贝构造函数一样也是一个浅拷贝，解决方法**深拷贝**。如下：

```
...

int * m_p;
CTest& operator=(const CTest& tst){
    if(this != &tst){  //判断是否自己给自己赋值
        this->m_a = tst.m_a;
        if (tst.m_p) {
```

```
            if (this->m_p) {
                *this->m_p = *tst.m_p;
            }
            else {
                this->m_p = new int(*tst.m_p);
            }
        }
        else {
            if (this->m_p) {
                delete this->m_p;
            }
            this->m_p = nullptr;
        }
    }
}

...
```

4. 总结

空类中存在的默认的函数4个：

1. 默认无参数构造
2. 默认的拷贝构造
3. 默认的operator=
4. 默认析构函数