13拷贝控制

1.拷贝构造函数 2.拷贝赋值运算符 3.移动构造函数 4.移动赋值运算符 5.析构函数

13.1 拷贝赋值与销毁

13.1.1 拷贝构造函数

一个构造函数第一个参数是自身类类型的引用，且任何额外参数都有默认值，则称其为拷贝构造函数。

```
class Foo{
    public:
    Foo(): //默认构造函数
    Foo(const Foo& ); //拷贝构造函数
}
```

合成拷贝构造函数

如果我们没有为一个类定义拷贝构造函数，编译器会为我们定义一个，即合成拷贝构造函数。 会将其参数的成员逐个拷贝到正在创建的对象中。

拷贝初始化

拷贝初始化不仅在我们用=定义变量，在下列情况也会发生：

- 将一个对象作为实参传递给一个非引用类型的形参
- 从一个 返回类型为非引用类型的函数返回对象
- 用花括号列表初始化一个数组中的元素或一个聚合类中的成员(P266)

13.1.2 拷贝赋值运算符

重载赋值运算符

重载运算符本质是函数，名字由operator关键字后接 运算符号组成 如果一个运算符是一个成员函数，其左侧运算对象就绑定到隐式的this参数。

拷贝赋值运算符接受一个与其所在类相同类型的参数：其通常返回一个指向其左侧运算对象的引用。

```
class Foo{
    public:
    Foo& operator=(const Foo&);
}
```

13.1.3 析构函数

析构函数执行与构造函数相反的操作，构造函数初始化对象的非static数据成员，析构函数释放对象使用的资源，并销毁非static数据成员。

```
class Foo{
    public:
    ~Foo();//析构函数
}
```

调用析构函数：

- 变量离开作用域被销毁
- 当一个对象被销毁，其成员被销毁
- 容器（无论是标准库容器还是数组）被销毁时，其元素被销毁
- 动态分配的对象，对指向它的指针应用delete运算符时被销毁
- 对于临时对象，当创建它的完整表达式结束而被销毁。 **当指向一个对象的引用或指针离开作用域时，析构函数不会执行**

合成析构函数

析构函数体自身不直接销毁成员

13.1.4 三/五法则

需要析构函数的类也需要拷贝和赋值操作 如果只定义了析构函数，未定义拷贝构造函数和拷贝赋值运算符，会引发错误

13.1.5 使用=default

我们可以通过将拷贝控制成员定义为=default来显示地要求编译器生成合成的版本

```
class Sales_data{
    Sales_data() = default;
}
```

13.1.6 阻止拷贝

可以通过将拷贝构造函数和拷贝赋值运算符定义为**删除的函数**来阻止拷贝。

```
struct NoCopy{
    NoCopy() = default;
    NoCopy(const NoCopy& ) = delete;//阻止拷贝
}
```

析构函数不能是删除的成员

合成的拷贝控制成员可能是删除的

private 拷贝控制

```
class PrivateCopy{
    PrivateCopy(const PrivateCopy & );
    PrivateCopy & operator=(const PrivateCopy&);
public:
    PrivateCopy() = default;
    ~PrivateCopy();//用户可以定义此类型的对象，但是无法拷贝它们;
}
```

拷贝构造函数和拷贝赋值运算符是private的，用户代码不能拷贝对象，但是友元和成员函数依旧可以拷贝对象；

最好还是用=delete来祖师拷贝

13.2拷贝控制和资源管理

确定此类型对象的**拷贝语义**，有两种：

- 类的行为看起来像一个值

> 它有自己的状态，当我们拷贝一个像值的对象时，副本和原对象完全独立，改变副本不会改变原对象；

- 类的行为看起来像一个指针

> 即共享状态，副本和原对象使用相同的底层数据，改变一个会改变另一个；

13.2.1 行为像值的类

对于类管理的资源，每个对象都该拥有一份自己的拷贝

赋值运算符

- 如果将一个对象赋予其自身，赋值运算符必须能正确工作
- 大多数赋值运算符组合了析构函数和拷贝构造函数的工作
- **好的模式是 将右侧对象拷贝到一个局部临时对象**