|  |  |
| --- | --- |
| 1. 派生类的拷贝控制成员 2. 定义派生类的拷贝或移动构造函数 3. 派生类赋值运算符 4. 派生类析构函数 5. 在构造函数和析构函数中调用虚函数 | D(const D&d):Base(d)  D(D&& d):Base(std::move(d));  Base::operator=(const Base&)  D &D::operator=(const D &RHS)  {  Base::operator=(rhs);  Return \*this;  } |
| 1. 派生类的拷贝控制成员 2. 拷贝控制成员，操作了派生类也要操作基类部分的成员 3. 不同的是析构函数只负责销毁派生类自己分配的资源。对象的成员是被隐式销毁的，类似的，派生类对象的基类部分也是自动销毁的   note：当派生类定义了拷贝或移动操作时，该操作负责拷贝或移动包括基类部分成员在内的整个对象   1. 定义派生类的拷贝或移动构造函数 2. 需要使用对应的基类构造函数初始化对象的基类部分 3. D(const D& d):B(d){}   将一个D对象传递给基类构造函数。道理上说，base可以有一个参数类型为D的构造函数，但通常不那么做   1. 一般会匹配Base的拷贝构造函数，将参数d上的基类部分拷给要创建的对象 2. 假如没有提供基类的初始值的话，基类的部分被默认初始化，很可能未定义的行为   note：基类默认构造函数初始化派生类对象的基类部分，若想拷贝或移动基类部分，必须在派生类的造函数初始化列表中显示地使用基类的拷贝或移动构造函数   1. 派生类赋值运算符 2. 也必须显示地为其基类部分赋值 3. 显示的：那么无论基类的构造函数或赋值运算符是自定义的版本还是合成版本，派生类的对应操作都能使用它们。 4. 派生类析构函数 5. 析构函数体执行完成后，对象的成员会被隐式销毁。 6. 对象的基类部分也是隐式销毁的。 7. 和构造函数及赋值运算符不同的是，派生类析构函数只负责销毁由派生类自己分配的资源：   Base::~Base被自动调用执行  对象销毁的顺序正好与其创建的顺序相反：派生类析构函数首先执行，然后是基类的析构函数，沿着继承体系的反方向直至最后   1. 在构造函数和析构函数中调用虚函数 2. 基类部分首先被构建，基类构造函数运行时，派生类的部分是未被初始化状态 3. 销毁派生类对象时，基类的析构函数运行时，派生类的部分已经被销毁掉了 4. 则在构造函数与析构函数中执行基类的虚函数时，该对象处于未完成的状态，因为派生类并不完整未被初始化或销毁掉了 5. 虚函数的调用绑定是把对象的类和构造函数的类看成同一个 6. 为了能处理上述状态： 7. 编译器认为对象的类型在构造或析构的过程中仿佛发生了改变一样，在我们构建一个对象时，需要把对象的类和构造函数的类看做是同一个 8. 这样则基类构造函数运行时，对象的类为基类，那么执行虚函数则是基类版本的虚函数。 9. 直接调用虚函数 10. 间接调用：并且通过构造函数或析构函数调用另一个函数也是可以 11. 假设上述行为没有执行，在基类的构造函数执行虚函数的派生类版本时，派生类成员未初始化，程序可能会崩溃。 12. note：如果构造函数或析构函数调用了某个虚函数，应该与我们执行构造函数或析构函数所属类型相对应得虚函数版本 | |