|  |  |
| --- | --- |
| 1. 合成拷贝控制与继承 2. 派生类中删除的拷贝控制与基类的关系 3. 移动操作与继承 |  |
| 1. 合成拷贝控制与继承 2. 合成拷贝控制成员：它们对类本身的成员依次进行初始化、赋值或销毁的操作， 3. 此外，这些成员还负责使用直接基类中对应的操作对一个对象的直接基类部分进行初始化、赋值或销毁的操作 4. 如：   bulk\_quote默认构造函数运行disc\_quote的默认构造函数，后者又运行Quote的默认构造函数,Quote的默认构造函数完成后，继续执行disc\_quote的构造函数，执行后  继续执行bulk\_quote的构造函数  其它拷贝控制成员类似的。   1. 注意的是：无论基类成员是合成的版本还是自定义的版本都没有太大影响。唯一的要求是相应的成员应该可访问并且不是一个被删除的函数 2. Quote继承体系。所有类都使用合成的析构函数，派生类隐式的使用而基类通过=default显示的使用。 3. 对于派生类的析构函数来说，它除了销毁派生类自己的成员外，还负责销毁派生类的直接基类，该直接基类又销毁它自己的直接基类，以此推至。 4. Quote因为析构函数而不能有合成的移动操作，因此移动Quote对象实际使用的是合成的拷贝操作，Quote没有移动操作意味派生类也没有 5. 派生类中删除的拷贝控制与基类的关系   基类或派生类可将拷贝控制成员定义被删除的，并且定义基类的方式也可能导致有的派生类成员成为被删除的函数   1. 若基类的默认 拷贝构造函数，拷贝赋值运算符，或析构函数是删除的或不可访问，派生类对应的成员也是被删除的   原因是：派生类无法使用基类成员来对派生类基类部分进行构造，赋值，销毁   1. 若基类中有一个不可访问或删除掉的析构函数则派生类中合成的默认和拷贝构造函数将是被删除的，因为派生类无法销毁它 2. 编译器不会合成一个删除掉的移动操作。若基类的移动操作是删除或不可访问，则派生类的也是删除的，原因：无法对基类部分移动 3. 若基类的析构函数是删除的或不可访问的，则派生类的移动构造函数也将是被删除的。 4. 若想派生类能移动和拷贝，则派生类需要自定义相应版本的构造函数，当然需要考虑如何移动或拷贝其基类部分的成员。 5. 如果在基类中没有默认、拷贝或移动构造函数，一般情况下派生类也不会定义相应的操作 6. 移动操作与继承 7. 因为基类的虚析构函数，通常基类不会有合成的移动操作，而且在它的派生类中也没有合成的移动操作 8. 基类缺少，则会阻止派生类，当我们确实需要执行移动操作时应在基类中定义。一旦基类定义了移动，则必须同时又显示的定义拷贝操作(P476) 9. 通过对基类的显示进行拷贝、移动 赋值和销毁。除非quote的派生类中含有排斥移动的成员，否则它将自动获得合成的移动操作   代码：  不管析构函数有没有被声明， 如果显示定义了拷贝移动操作，则执行拷贝移动操作，若没定义移动操作，默认执行拷贝构造  class B {  public:  B() = default;  B(const B&) {  cout << "B(const B&);" << endl;  }  B(B&&){  cout << "B(B&&);" << endl;  }  //virtual ~B() = default;  };  class D :public B {  };  int main()  {  int a;  D d;  D d3(std::move(d));// 错误，隐式的使用D的被删除的拷贝构造函数,  cin >> a;  return 0;  } | |