|  |  |
| --- | --- |
| 1. 虚函数 2. 对虚函数的调用可能在运行时才被解析 3. 派生类中的虚函数 4. final和override说明符 5. 虚函数与默认实参 6. 回避虚函数的机制 | OPP  Final override  Void f1(int) const override;  Void f1(int) const final;  Basep->Quote::net\_price(42); |
| 1. 虚函数 2. 使用基类的引用或指针调用一个虚成员函数会执行动态绑定 3. 对虚函数的调用可能在运行时才被解析 4. 虚函数被调用是与绑定到指针或引用上的对象的动态类型相匹配的那个 5. 具有普通类型的表达式调用虚函数，在编译时就确定了版本，可以改变base表示的对象的值(内容)，但是不会改变该对象的类型   关键概念：C++多态性  1) 把具有继承关系的多个类型称为多态类型  2) 基类的引用或指针调用基类定义的一个函数时  虚函数：运行时才决定哪个版本  非虚函数：编译时就决定与静态类型一致版本   1. 用对象进行调用的函数   虚函数与非虚函数在编译时与静态类型一致   1. 派生类中的虚函数 2. 一个函数为虚函数，在所有派生类中都是虚函数，可以再写virtual显示表示 3. 当派生类重新覆盖了某个继承而来的虚函数，形参必须与覆盖的基类函数完全一致   返回类型也必须匹配，例外的是当基类中返回类本身的指针或引用时.   1. 如：基类的虚函数返回B\*,派生类的虚函数返回D\*,前提是D到B的类型转换是可访问的 2. final和override说明符 3. override，说明派生类中的虚函数覆盖了基类的虚函数，若没有则会显示提示报错   只有虚函数才可以被覆盖   1. final，把函数指定为final，则之后任何尝试覆盖该函数的操作都将引发错误 2. override 与final出现在形参列表 以及尾置类型之后 这两个顺序无所谓 3. 虚函数与默认实参 4. 虚函数也可以有默认实参 5. 默认实参值由本次调用的静态类型决定。即使派生类重写这个虚函数并且改变了默认值，也是基类的默认参数值。（在发生动态绑定时） 6. 所以派生类的虚函数的默认实参应该与基类的默认实参保持一致 7. 回避虚函数的机制 8. 希望对虚函数的调用不要执行动态绑定，而是强迫执行虚函数的某个特定版本。使用作用域运算符, 在编译时完成解析   note：通常，只有成员函数（或友元）中的代码才需要使用作用域运算符来回避虚函数的机制   1. 什么时候用：   一般在派生类的虚函数中调用基类虚函数的虚函数完成共同任务，  注意要用作用域，不然就会自身调用自身 | |