|  |  |
| --- | --- |
| 1. 继承中的类作用域 2. 在编译时进行名字查找 3. 名字冲突与继承 4. 通过作用域运算符来使用隐藏的成员 5. 一如往常，名字查找先于类型检查 6. 虚函数与作用域 7. 通过基类调用隐藏的虚函数 8. 覆盖重载的函数 | Return Base::mem; |
| 1. 继承中的类作用域 2. 每个类定义自己的作用域，作用域中定义类的成员。 3. 继承关系时：派生类的作用域嵌套在其基类的作用域之内 4. 如果在派生类的作用域无法找到成员，则会在外层的基类作用域查找 5. 在编译时进行名字查找 6. 一个对象、引用或指针的静态类型决定了该对象的哪些成员是可见的。 7. 即使静态类型与动态类型可能不一致。 8. 总：由静态类型决定哪些成员可见可使用,即若静态类型是基类，找不到派生类的成员   即：向上转型时若静态类型与动态类型不一致，从静态类型（父类）查找，则会忽略动态类型（子类）的成员   1. 名字冲突与继承 2. 派生类的成员名字会隐藏定义在基类的成员名字,同名 3. 通过作用域运算符来使用隐藏的成员 4. 可以使用作用域运算符来使用一个被隐藏的基类成员 5. note：除了覆盖继承而来的虚函数之外，最好不要定义同名名字   关键概念：名字查找与继承  p->mem();4个步骤  1. 确定p的静态类型。必然是类类型  2. p的静态类型对应的类中查找mem，找不到再找基类直至顶，若没有则报错  3. 若找到了mem，常规的类型检查，是否合法  4. 合法：是否是虚函数产生不同的代码：  1) 若mem是虚函数且我们是通过引用或指针进行的调用，则编译器产生的代码将在运行时确定到底运行该虚函数的哪个版本，根据动态类型  2) 若不是虚函数或者通过对象（而非引用或指针）进行的调用，则编译器将产生一个常规函数调用  5. 一如往常，名字查找先于类型检查   1. 内层作用域的函数不会重载外层作用域的函数 2. 定义派生类中的函数也不会重载基类的成员。 3. 若派生类中的成员与基类的某个成员同名，则派生类将在其作用域内隐藏该基类成员，即使形参列表不一致 4. 虚函数与作用域 5. 所以基类与派生类的虚函数接受的实参必须相同 6. 通过基类调用隐藏的虚函数 7. 就是根据动态类型查找，不会调用隐藏的虚函数，会向上查找 8. 覆盖重载的函数 9. 引出：和其它函数一样，成员函数无论是否是虚函数都能被重载 10. 派生类可以覆盖重载函数的0个或多个实例。 11. 派生类希望所有的重载版本对于它来说都是可见，则需要覆盖所有的版本，或者一个也不覆盖 12. 问题是：若派生类需要使用基类的重载成员，并且自身需要覆盖部分基类的重载成员，则不得不需要覆盖基类的每个版本，显然麻烦 13. 解决方案是：使用using   用using声明基类中重载函数的名称，再重新定义向覆盖的重载函数版本则可以解决。  对派生类没有重新定义的重载版本的访问实际上是对using声明点的访问 | |