一、重载：

\*针对在同一个类中（或者相同作用域中，如果作用域相互嵌套，那么内层会“隐藏”外层），跟继承没关系

\*函数名字相同

\*参数不同

\*virtual可有可无

\*重载只要看参数跟哪个函数匹配调用哪个

被继承下来的函数，如果显式得在子类中重新定义同名函数，要么被覆盖，要么被隐藏。如果只是单纯的继承下来，什么都不做，那么会成为子类的成员函数。

二、覆盖（重写）（常规的多态）：

\*分别位于基类和派生类

\*函数名字相同

\*参数相同

\*基类必须有virtual

此时，基类指针根据指向的对象类型来决定调用基类或者派生类的函数

两个类有了继承关系后，使用基类指针，调用的函数在基类使用virtual声明并且没有被隐藏，满足这三个条件。采用动态联编：在运行阶段确定指针指向对象的类型再调用相应函数。

至于为什么不能在编译阶段完成这项工作：

void fun(Base \* p)

{

p->printf();

}

假如没有动态联编，那么fun函数有可能在指针参数传进来之间就完成了编译。那么就实现不了多态。要实现多态只能在运行阶段（调用fun时）确定了参数指向的对象类型再编译fun()函数。

三、隐藏（重定义）：

1、派生类和基类的函数名相同，参数不同，不论有无virtual

2、派生类和基类的函数名相同，参数相同，基类没有virtual

此时根据指针类型来调用函数，参数要与指针类型的成员函数匹配。并且派生类即使参数与基类函数相同，也不能再使用基类的同名函数（本来根据继承关系是可以的），实现隐藏。并且这种隐藏会随着继承关系延续下去。

采用静态联编。

3、还有一种情况跟继承没有关系，即函数声明在不同的作用域，且作用域相互嵌套，内层作用域声明的同名函数（不管参数是否相同），将隐藏外层作用域函数，在内层作用域，即使参数与外层函数一致，也不能使用外层函数（本来是可以的）。

四、虚函数表

\*虚函数表在编译阶段就产生，只有一张（多重继承和虚继承除外）实例对象通过虚函数表入口指针\_vfptr来控制在虚函数表的哪一部分区域遍历查找。

\*单纯的继承（没有覆盖或者隐藏），虚函数表中对应子类的区域也会有继承来的基类的虚函数地址。此时使用基类指针调用该虚函数，也会采用动态联编。

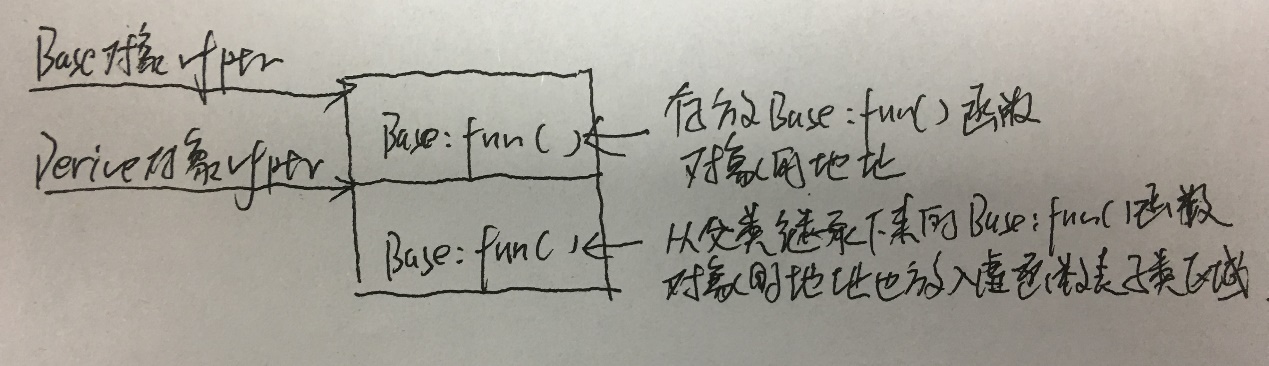
\*如果是覆盖，那么虚函数表中对应子类的区域本来应该存放基类虚函数地址的地方被子类虚函数覆盖。

\*如果是第一种隐藏，假如基类函数用virtual声明，虚函数表中对应子类的区域，同样会有继承来的基类的虚函数地址。但是，此时采用静态联编，虚函数表中的这个函数地址毫无意义。

\*只有在动态联编时才使用虚函数表和\_vfptr来确定调用函数。

\*具体虚函数表和继承、覆盖的关系见图示：

继承：



覆盖：

