多态

多态是不同继承关系的类对象，去调用同一函数，产生了不同行为。

多态的定义和实现

构成条件

1. 必须**通过基类的指针或者引用**调用虚函数
2. 被调用的函数必须是虚函数，且派生类必须对基类的虚函数进行重写

虚函数

- 虚函数——被virtual修饰的类成员函数
- 虚函数的重写(覆盖)——派生类中有一个跟基类完全相同的虚函数(即派生类虚函数与基类虚函数的**返回值类型、函数名字、参数列表完全相同**)，称子类的虚函数重写了基类的虚函数
- 虚函数重写的两个例外：
    - 协变——派生类重写基类函数时，与基类虚函数返回值类型不同。即基类虚函数返回基类对象的指针或引用，派生类虚函数返回派生类对象的指针或引用
    - 析构函数的重写——基类与派生类析构函数的名字不同，看似违反了重写规则，但是编译器后期会将析构函数名同一处理成destructor

override和final关键字

1. final——修饰虚函数，表示该虚函数不能再被继承

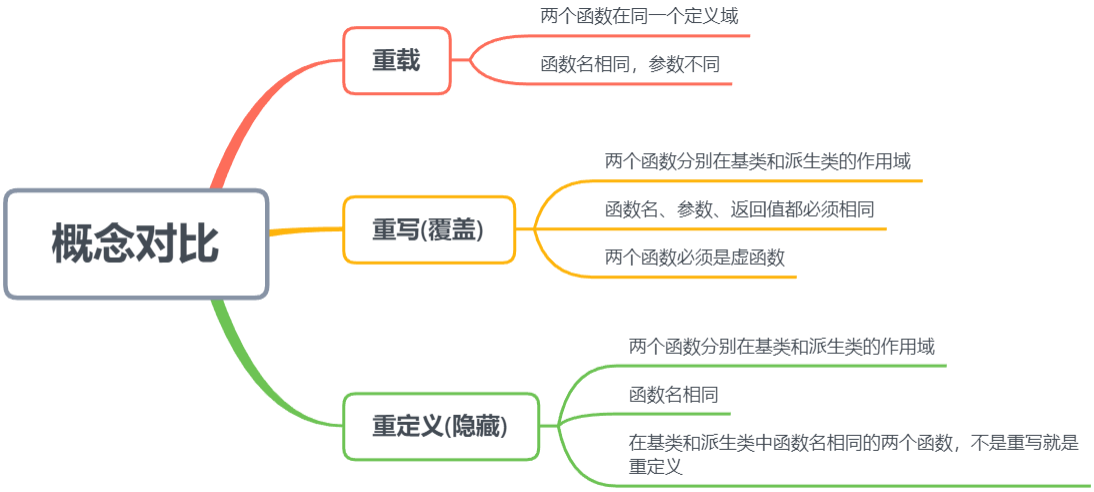
```
class A final//表示该类不能再被继承
{
public:
    virtual void func() final//该虚函数不能再被继承
    {
    }

};
class B: public A
{
piblic:
    virtual void func()
    {
    }
}
```

2. override——检查派生类虚函数是否重写了基类的某个虚函数，如果没有重写编译报错

```
class A
{
public:
    virtual void func() final
    {
    }
```

```
};
class B: public A
{
piblic:
    virtual void func override()
    {
    }
}
```

重载、重写(覆盖)、重定义(隐藏)



抽象类

抽象就是在现实中没有对应的实体。

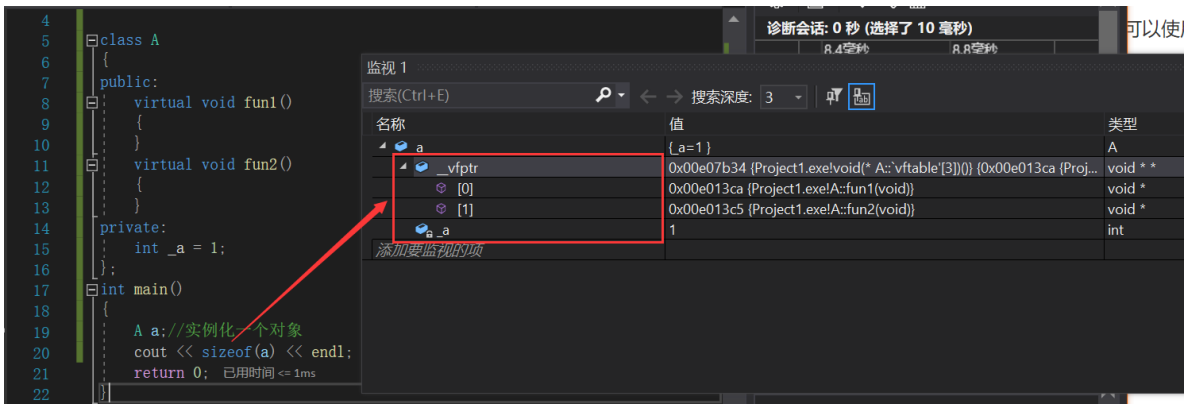
在虚函数的后面写上=0，则这个函数为纯虚函数。**包含纯虚函数的类叫做抽象类(也叫接口类)，抽象类不能实例化出对象。**派生类继承后也不能实例化出对象，只有重写纯虚函数，派生类才能实例化出对象。

```
//抽象类
class A
{
public:
    //纯虚函数
    virtual void fun() = 0;
};
//继承
class B : public A
{
public:
    //void fun()
    virtual void fun()
    {
        //重写虚函数
    }
};
int main()
{
    //A a;//抽象类不能实例化对象
    //B a;//派生类被继承后也不能实例化对象
```

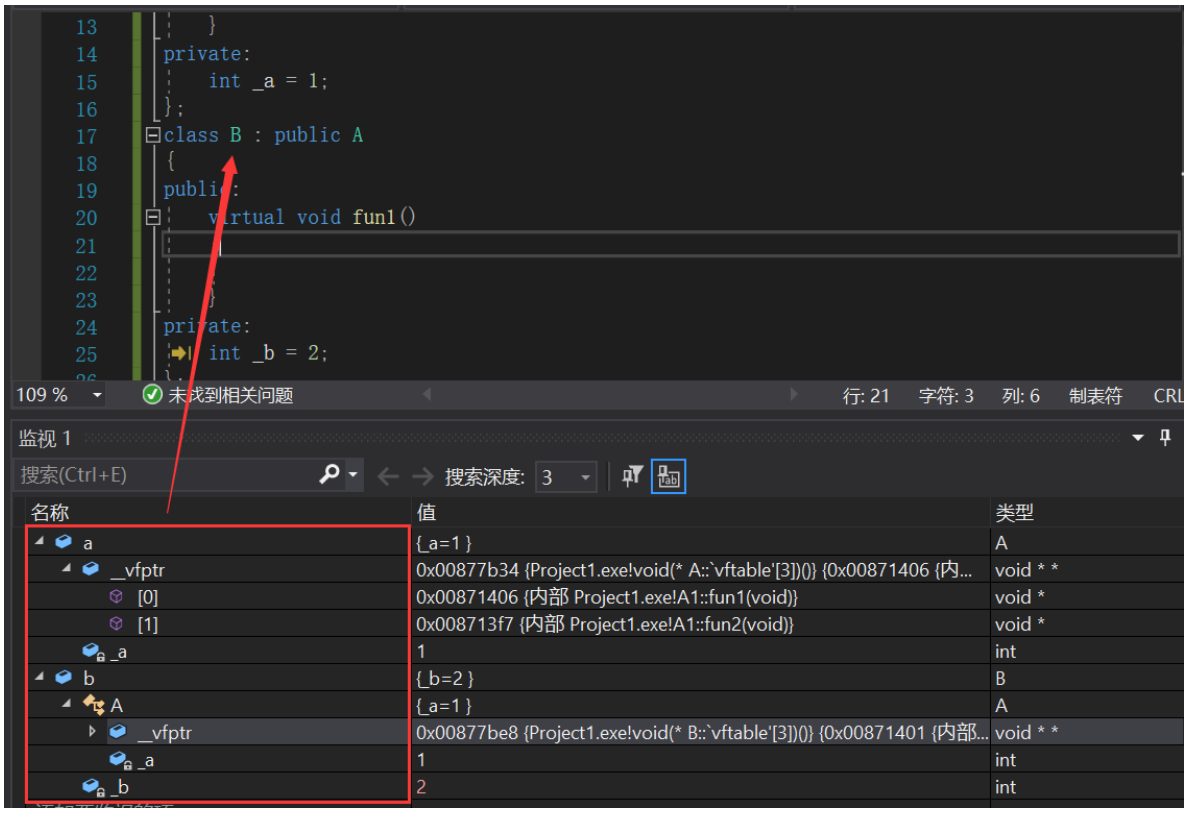
```
    B a;//重写纯虚函数，派生类可以实例化对象
    return 0;
}
```

- 纯虚函数不需要实现，声明即可。
- 纯虚函数的作用，就是强制子类去完成重写。
- 普通函数的继承是一种实现继承，派生类继承了基类函数，可以使用函数，继承的是函数的实现。
- 虚函数的继承是一种接口继承，派生类继承的是基类虚函数的接口，目的是为了重写，达成多态，继承的是接口。

多态的原理



- 实例化的对象a，除了_a成员之外，还多了一个_vfptr指针，这个指针我们叫做虚函数表指针(v——virtual，f——function)。
- 一个含有虚函数的类中至少有一个虚函数表指针，虚函数的地址要被放到虚函数表中。



- 派生类的虚表生成：
  - 先将基类中的虚表内容拷贝到派生类的虚表中；
  - 如果派生类重写了基类中的某个函数，用派生类自己的虚函数覆盖虚表中基类的虚函数；
  - 派生类自己新增加的虚函数依次增加到派生类虚表的后面；
- 虚函数存在哪？虚表存在哪？

    - 虚表的本质是一个存虚函数指针的指针数组。虚表存的是虚函数指针，不是虚函数，虚表存在代码段；
    - 虚函数和普通函数一样，都是存在代码段的，只是它的指针存在虚表中；
- 满足多态的函数调用，是运行起来以后到对象中寻找的，不满足多态的函数调用是编译时确认好的；
- **多继承派生类的未重写的虚函数放在第一个继承基类部分的虚函数表中；**

动态绑定与静态绑定

1. 静态绑定——在程序编译阶段确定程序的行为，也称为静态多态。
2. 动态绑定——在程序运行期间，根据拿到的类型确定程序的具体行为，调用具体的函数，也称为动态多态。