* 我希望你没有刻意为追求一个数字而生活，我希望你找到了真正的价值所在
* 你一定要**坚持做自己**，**静下心**来做自己喜欢的事，然后把自己交给**命运**



@[toc]

# 继承

## 继承的基本语法

* 定义这些类时，下级别的成员除了拥有上一级的共性，还有自己的特性。
* 继承的好处：可以减少重复的代码

class A : public B;

A 类称为子类 或 派生类  
B 类称为父类 或 基类

//公共页面  
class BasePage  
{  
public:  
 void header()  
 {  
 cout << "首页、公开课、登录、注册...（公共头部）" << endl;  
 }  
  
 void footer()  
 {  
 cout << "帮助中心、交流合作、站内地图...(公共底部)" << endl;  
 }  
 void left()  
 {  
 cout << "Java,Python,C++...(公共分类列表)" << endl;  
 }  
  
};  
  
//Java页面  
class Java : public BasePage  
{  
public:  
 void content()  
 {  
 cout << "JAVA学科视频" << endl;  
 }  
};  
//Python页面  
class Python : public BasePage  
{  
public:  
 void content()  
 {  
 cout << "Python学科视频" << endl;  
 }  
};  
//C++页面  
class CPP : public BasePage  
{  
public:  
 void content()  
 {  
 cout << "C++学科视频" << endl;  
 }  
};  
  
void test01()  
{  
 //Java页面  
 cout << "Java下载视频页面如下： " << endl;  
 Java ja;  
 ja.header();  
 ja.footer();  
 ja.left();  
 ja.content();  
 cout << "--------------------" << endl;  
  
 //Python页面  
 cout << "Python下载视频页面如下： " << endl;  
 Python py;  
 py.header();  
 py.footer();  
 py.left();  
 py.content();  
 cout << "--------------------" << endl;  
  
 //C++页面  
 cout << "C++下载视频页面如下： " << endl;  
 CPP cp;  
 cp.header();  
 cp.footer();  
 cp.left();  
 cp.content();  
  
  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**派生类中的成员，包含两大部分**：

一类是从基类继承过来的，一类是自己增加的成员。

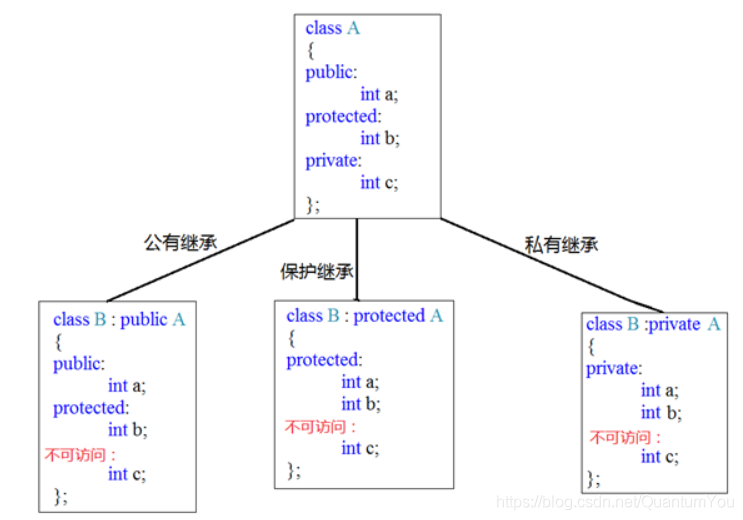
从基类继承过过来的表现其共性，而新增的成员体现了其个性。

## 继承方式

继承的语法：class 子类 : 继承方式 父类

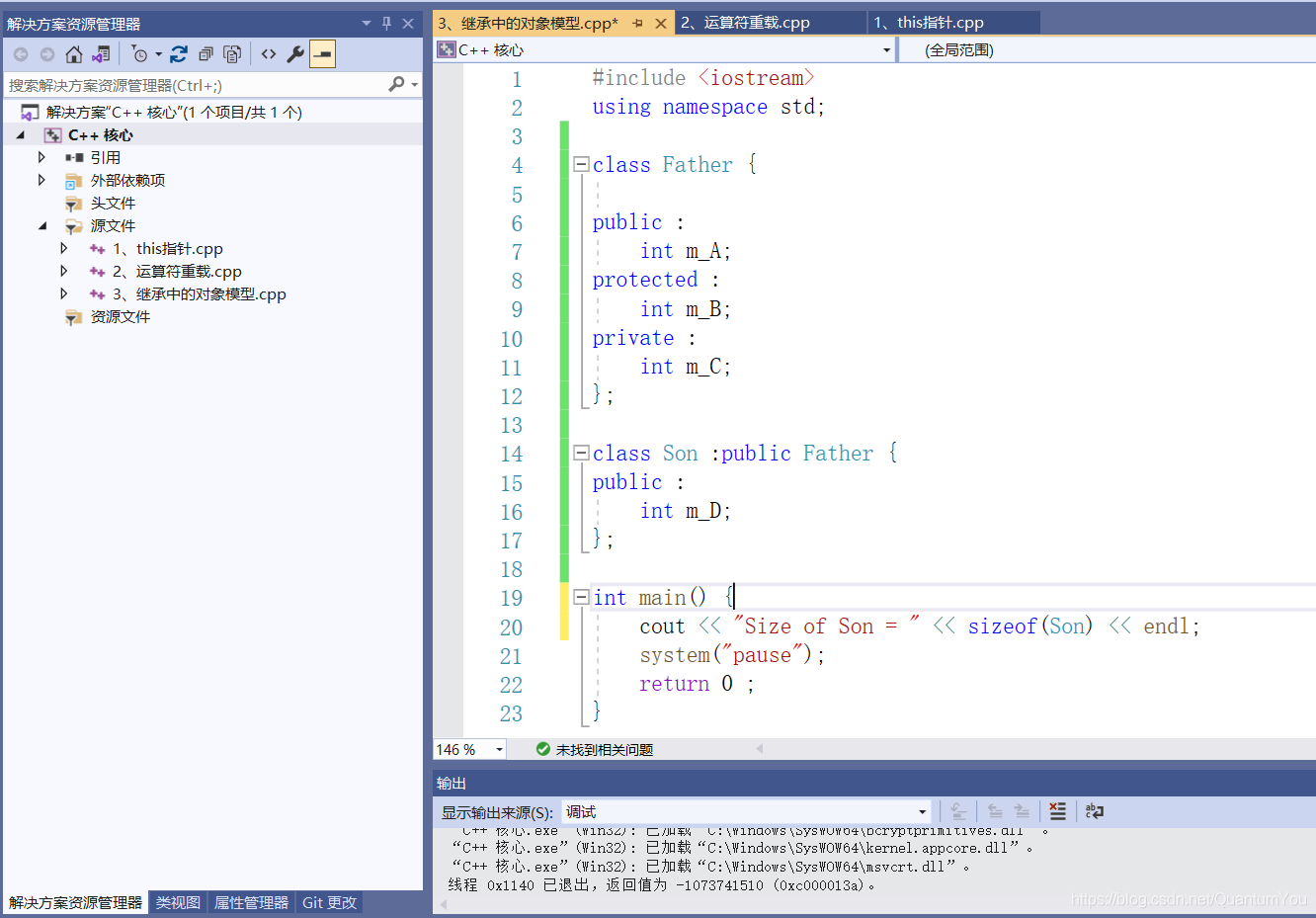
**继承方式一共有三种：**

* 公共继承
* 保护继承
* 私有继承

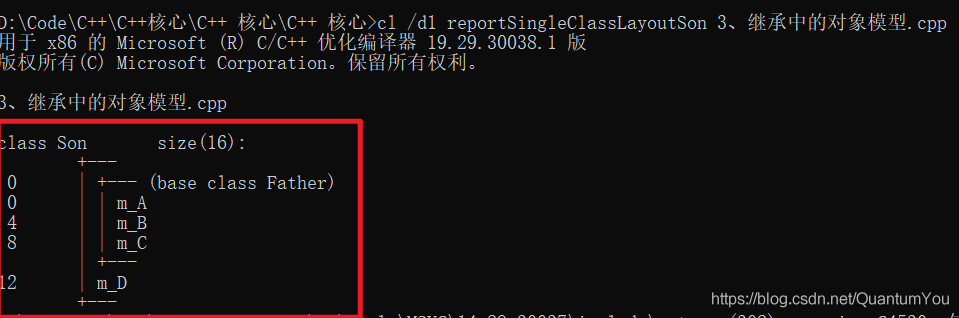
关键为理解好下面的图片：  


class Base1  
{  
public:   
 int m\_A;  
protected:  
 int m\_B;  
private:  
 int m\_C;  
};  
  
//公共继承  
class Son1 :public Base1  
{  
public:  
 void func()  
 {  
 m\_A; //可访问 public权限  
 m\_B; //可访问 protected权限  
 //m\_C; //不可访问  
 }  
};  
  
void myClass()  
{  
 Son1 s1;  
 s1.m\_A; //其他类只能访问到公共权限  
}  
  
//保护继承  
class Base2  
{  
public:  
 int m\_A;  
protected:  
 int m\_B;  
private:  
 int m\_C;  
};  
class Son2:protected Base2  
{  
public:  
 void func()  
 {  
 m\_A; //可访问 protected权限  
 m\_B; //可访问 protected权限  
 //m\_C; //不可访问  
 }  
};  
void myClass2()  
{  
 Son2 s;  
 //s.m\_A; //不可访问  
}  
  
//私有继承  
class Base3  
{  
public:  
 int m\_A;  
protected:  
 int m\_B;  
private:  
 int m\_C;  
};  
class Son3:private Base3  
{  
public:  
 void func()  
 {  
 m\_A; //可访问 private权限  
 m\_B; //可访问 private权限  
 //m\_C; //不可访问  
 }  
};  
class GrandSon3 :public Son3  
{  
public:  
 void func()  
 {  
 //Son3是私有继承，所以继承Son3的属性在GrandSon3中都无法访问到  
 //m\_A;  
 //m\_B;  
 //m\_C;  
 }  
};

## 继承中的对象模型

**问题** ： 从父类继承过来的成员，哪些属于子类对象中？  


利用开发工具查看类中属性分布图：  
输入命令 ： cl /d1 reportSingleClassLayout查看的类名 所属文件名



## 继承中构造和析构顺序

* 子类继承父类后，当创建子类对象，也会调用父类的构造函数
* **问题**：父类和子类的构造和析构顺序是谁先谁后？

**示例：**

class Base   
{  
public:  
 Base()  
 {  
 cout << "Base构造函数!" << endl;  
 }  
 ~Base()  
 {  
 cout << "Base析构函数!" << endl;  
 }  
};  
  
class Son : public Base  
{  
public:  
 Son()  
 {  
 cout << "Son构造函数!" << endl;  
 }  
 ~Son()  
 {  
 cout << "Son析构函数!" << endl;  
 }  
  
};  
  
  
void test01()  
{  
 //继承中 先调用父类构造函数，再调用子类构造函数，析构顺序与构造相反  
 Son s;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结**：继承中 先调用父类构造函数，再调用子类构造函数，析构顺序与构造相反

## 关于继承与多个类的书写

* **继承** ：class Son2:protected Base2 语法为：class 子类 : 继承方式 父类
* **多个类** ： Circle :: setR(int r) 这里的语法表示成员函数作用域

## 继承同名成员处理方式

**问题**：当子类与父类出现同名的成员，如何通过子类对象，访问到子类或父类中同名的数据呢？

* 访问子类同名成员 直接访问即可
* 访问父类同名成员 需要加作用域

**示例：**

class Base {  
public:  
 Base()  
 {  
 m\_A = 100;  
 }  
  
 void func()  
 {  
 cout << "Base - func()调用" << endl;  
 }  
  
 void func(int a)  
 {  
 cout << "Base - func(int a)调用" << endl;  
 }  
  
public:  
 int m\_A;  
};  
  
  
class Son : public Base {  
public:  
 Son()  
 {  
 m\_A = 200;  
 }  
  
 //当子类与父类拥有同名的成员函数，子类会隐藏父类中所有版本的同名成员函数  
 //如果想访问父类中被隐藏的同名成员函数，需要加父类的作用域  
 void func()  
 {  
 cout << "Son - func()调用" << endl;  
 }  
public:  
 int m\_A;  
};  
  
void test01()  
{  
 Son s;  
  
 cout << "Son下的m\_A = " << s.m\_A << endl;  
 cout << "Base下的m\_A = " << s.Base::m\_A << endl;  
  
 s.func();  
 s.Base::func();  
 s.Base::func(10);  
  
}  
int main() {  
  
 test01();  
  
 system("pause");  
 return EXIT\_SUCCESS;  
}

总结：

1. 子类对象可以直接访问到子类中同名成员
2. 子类对象加作用域可以访问到父类同名成员
3. 当子类与父类拥有同名的成员函数，子类会隐藏父类中同名成员函数，加作用域可以访问到父类中同名函数

**注意** :

* 如果子类中出现和父类同名的成员函数，子类的同名成员会隐藏掉父类中所有同名成员函数
* 如果想访问到父类中被隐藏的同名成员函数，需要加作用域

## 继承同名静态成员处理方式

**问题**：继承中同名的*静态成员*在子类对象上如何进行访问？

静态成员和非静态成员出现同名，处理方式一致

* 访问子类同名成员 直接访问即可
* 访问父类同名成员 需要加作用域

**示例：**

class Base {  
public:  
 static void func()  
 {  
 cout << "Base - static void func()" << endl;  
 }  
 static void func(int a)  
 {  
 cout << "Base - static void func(int a)" << endl;  
 }  
  
 static int m\_A;  
};  
  
int Base::m\_A = 100;  
  
class Son : public Base {  
public:  
 static void func()  
 {  
 cout << "Son - static void func()" << endl;  
 }  
 static int m\_A;  
};  
  
int Son::m\_A = 200;  
  
//同名成员属性  
void test01()  
{  
 //通过对象访问  
 cout << "通过对象访问： " << endl;  
 Son s;  
 cout << "Son 下 m\_A = " << s.m\_A << endl;  
 cout << "Base 下 m\_A = " << s.Base::m\_A << endl;  
  
 //通过类名访问  
 cout << "通过类名访问： " << endl;  
 cout << "Son 下 m\_A = " << Son::m\_A << endl;  
 cout << "Base 下 m\_A = " << Son::Base::m\_A << endl;  
}  
  
//同名成员函数  
void test02()  
{  
 //通过对象访问  
 cout << "通过对象访问： " << endl;  
 Son s;  
 s.func();  
 s.Base::func();  
  
 cout << "通过类名访问： " << endl;  
 Son::func();  
 Son::Base::func();  
 //出现同名，子类会隐藏掉父类中所有同名成员函数，需要加作作用域访问  
 Son::Base::func(100);  
}  
int main() {  
  
 //test01();  
 test02();  
  
 system("pause");  
  
 return 0;  
}

总结：同名静态成员处理方式和非静态处理方式一样，只不过有两种访问的方式（通过对象 和 通过类名）

注意：

* **静态成员变量** ： 类内声明 ， 类外初始化
* 即为**静态成员变量的访问方式**有两种 ： ① 通过对象 Son s ; s.m\_A ; ② 通过类名 Son :: m\_A ;
* **静态成员变量特点** ： 所有对象共享同一份数据，编译阶段分配内存、类内声明、类外初始化
* **静态成员函数特点** ：所有对象共享同一份数据，只能访问静态成员变量
* 子类出现和父类同名静态成员函数，也会隐藏父类中所有同名成员函数 , 如果想访问父类中被隐藏同名成员，需要加作用域

## 多继承语法

* C++允许**一个类继承多个类** 简言之，一个类可以有多个爸爸

语法：class 子类 ：继承方式 父类1 ， 继承方式 父类2...

多继承可能会引发父类中有同名成员出现，需要加作用域区，因此，**C++实际开发中不建议用多继承**

**示例：**

class Base1 {  
public:  
 Base1()  
 {  
 m\_A = 100;  
 }  
public:  
 int m\_A;  
};  
  
class Base2 {  
public:  
 Base2()  
 {  
 m\_A = 200; //开始是m\_B 不会出问题，但是改为mA就会出现不明确  
 }  
public:  
 int m\_A;  
};  
  
//语法：class 子类：继承方式 父类1 ，继承方式 父类2   
class Son : public Base2, public Base1   
{  
public:  
 Son()  
 {  
 m\_C = 300;  
 m\_D = 400;  
 }  
public:  
 int m\_C;  
 int m\_D;  
};  
  
  
//多继承容易产生成员同名的情况  
//通过使用类名作用域可以区分调用哪一个基类的成员  
void test01()  
{  
 Son s;  
 cout << "sizeof Son = " << sizeof(s) << endl;  
 cout << s.Base1::m\_A << endl;  
 cout << s.Base2::m\_A << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结： 多继承中如果父类中出现了同名情况，子类使用时候要加作用域

## 菱形继承

**菱形继承概念：**

* 两个派生类继承同一个基类
* 又有某个类同时继承者两个派生类
* 这种继承被称为菱形继承，或者钻石继承

\_\_\_\_\_\_\_\_  
 | A 类 |  
 |\_\_\_\_\_\_|  
 / \  
 / \  
 \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_  
 | B 类 | | C 类 |  
 |\_\_\_\_\_\_| |\_\_\_\_\_\_|  
 \ /  
 \ \_\_\_\_\_\_\_\_ /  
 | D 类 |   
 |\_\_\_\_\_\_\_\_|

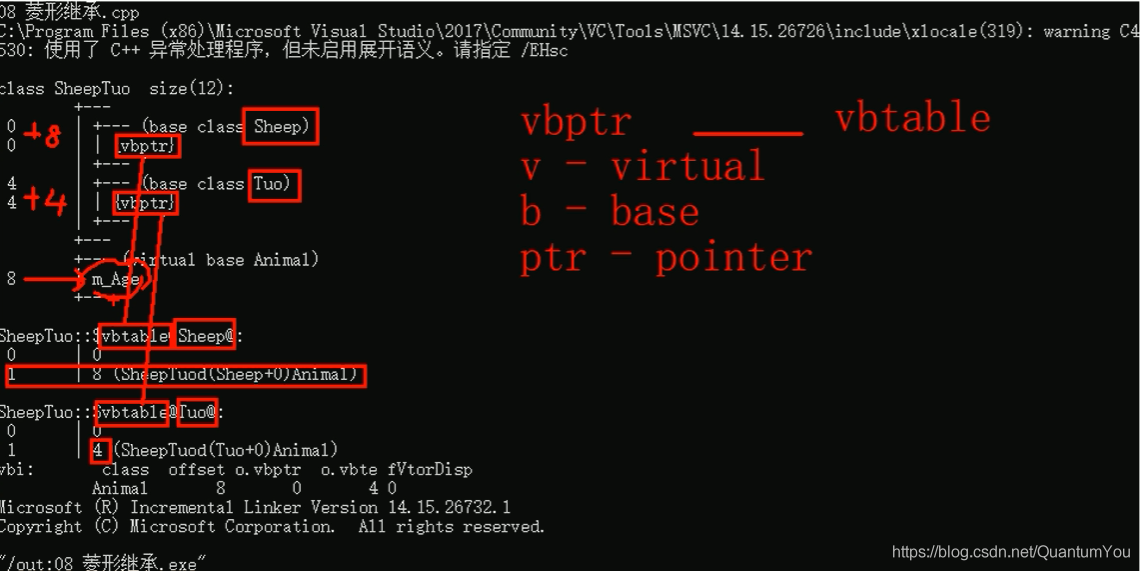
**语法** ： class Sheep : virtual public Animal {};

class Animal  
{  
public:  
 int m\_Age;  
};  
  
//继承前加virtual关键字后，变为虚继承  
//此时公共的父类Animal称为虚基类  
class Sheep : virtual public Animal {};  
class Tuo : virtual public Animal {};  
class SheepTuo : public Sheep, public Tuo {};  
  
void test01()  
{  
 SheepTuo st;  
 st.Sheep::m\_Age = 100;  
 st.Tuo::m\_Age = 200;  
  
 cout << "st.Sheep::m\_Age = " << st.Sheep::m\_Age << endl;  
 cout << "st.Tuo::m\_Age = " << st.Tuo::m\_Age << endl;  
 cout << "st.m\_Age = " << st.m\_Age << endl;  
}  
  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 菱形继承带来的主要问题是子类继承两份相同的数据，导致资源浪费以及毫无意义
* 利用虚继承可以解决菱形继承问题

**内部底层实现原理**：

* 指向内部虚基类表（表内记录偏移量数据）（最后两者指向的是同一个数据）  
  

[参考链接 16:20](https://www.bilibili.com/video/BV1et411b73Z?p=134&spm_id_from=pageDriver)