《面向对象程序设计》朋辈课程 · 第十辑

# 继承与派生

信息学院 赵家宇

PART 01

# 类的继承

# 类、对象与继承

- 学习了类和对象后，我们已经了解面向对象程序设计的两个重要特征：数据抽象与封装，现在能够设计基于对象的程序；要较好地进行面向对象程序设计，还必须要继续学习继承和多态。
- 通过继承，可以通过重用并扩展已有的类定义，创建新类，体现程序代码的复用性。继承是面向对象程序设计最重要的特征。

# 继承与派生

- 在编程中，我们已经设计了一个类，现在还需要定义第二个类，而二者的内容基本相同或有一部分相同，因此考虑用能否利用原来声明的类为基础，加上新内容，减少重复的工作；
- 继承作为C++语言中类机制的一部分，使类与类之间可以建立一种上下级关系，可以通过提供来自另一个类的操作和数据成员来创建新类，程序员只需在新类中定义已有类中没有的成分即可建立新类；

# 继承与派生

- 继承与派生是一对相对应的概念，继承表示新的类从已有类那里得到已有的特性；派生表示从已存在的类产生一个新的类；
  - 基类（父类）：在继承关系中，被继承的类（或已存在的类）；
  - 派生类（子类）：通过继承关系定义出来的新类（新建立的类）；
- 基类和派生类的关系：派生类通过增加信息将抽象的基类变为某种有用的类型。派生类是基类定义的延续。

# 继承与派生

```
class 派生类名: [继承方式] 基类名
{
    派生类新增加的成员
};
```

- 继承方式：即派生类的访问控制方式，用于控制基类中声明的成员在多大的范围内能被派生类的用户访问。
- 派生类新增加的成员：是指除了从基类继承来的所有成员之外，新增加的数据成员和成员函数。

# 继承与派生

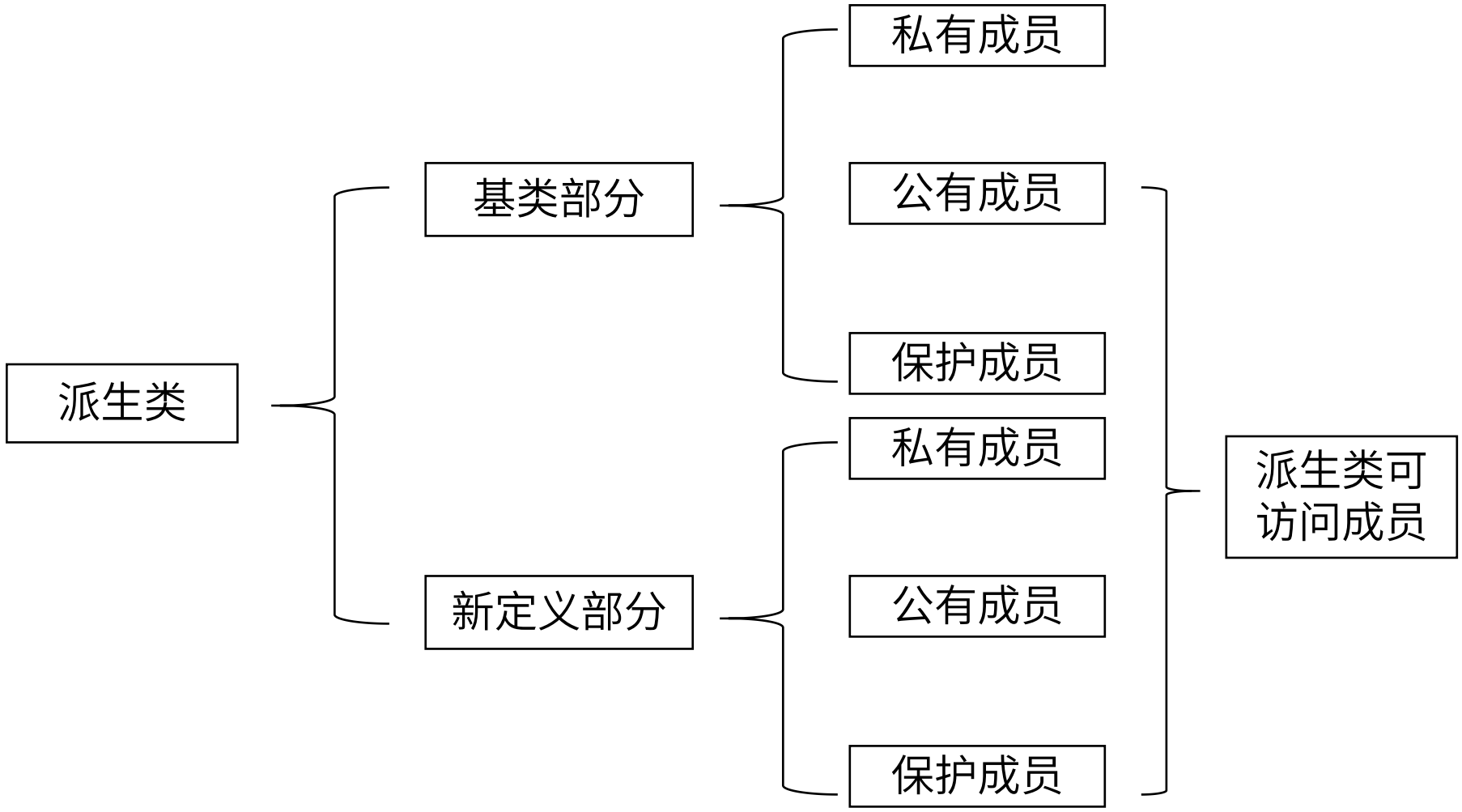
- protected成员用来缓解继承与数据封装的矛盾，与私有成员一样，在基类中，不能被使用类程序员进行公共访问，但可以被类内部的成员函数访问；但能够被派生类的成员函数访问；

```
class A {
   protected:
     int x,y;
   public:
     void f();
}; //A为基类
```

```
class B: public A {
    void h() {
        x=0, y=1; //OK
        f();      //OK
    }
};//B为派生类
```

```
void g() {
   A a;
   a.x=0, a.y=1; //Error
   a.f(); //OK
} //g为一般函数
```

# 继承与派生



# 继承与派生

- 继承方式分为private/public/protected三种方式；如果省略则默认为private方式；不同的继承方式，导致具有不同访问属性的基类成员在派生类中具有新的访问属性。

# public继承

- 基类的公用成员和保护成员在派生类中保持其公用成员和保护成员的属性；
- 基类的私有成员在派生类中并没有成为派生类的私有成员（派生类中不可见），它仍然是基类的私有成员，只有基类的成员函数可以引用它，而不能被派生类的成员函数引用；
- 通过派生类的对象只能访问基类的public成员；若非如此，则将破坏基类的封装性；

# public继承

```
class A {     // 定义基类
    private: int x;
    public: void setx(int i){ x = i;}
};
class B: public A {
    private: int y;
    public:
        void setxy(int m,int n) {
            x = m; //Error
            setx(m); y = n; //OK
        }//派生类成员函数访问基类公有成员
};
```

```
void main( ){
    A a;      //定义基类对象a
    B b;      // 定义派生类对象b
    a.x = 5; //Error
    b.x = 10; //Error
    b.y = 20; //Error
    a.setx(5); //OK
    b.setx(10); //OK
    b.setxy(10, 20); //OK
}
```

# private继承

- 基类的public和protected成员都以private身份出现在派生类中，而基类的private成员不可直接访问；
- 派生类中的成员函数可以直接访问基类中的public和protected成员，但不能直接访问基类的private成员；
- 通过派生类的对象不能直接访问基类中的任何成员；其作用在于：对于无需再往下继承的类的功能可以采用私有继承方式将其隐蔽起来！

# private继承

```
class A {     // 定义基类
    private: int x;
    public: void setx(int i){ x = i;}
};
class B: private A {
    private: int y;
    public:
        void setxy(int m,int n) {
            x = m; //Error
            setx(m); y = n; //OK
        }//派生类成员函数访问基类公有成员
};
```

```
void main( ){
    A a;      //定义基类对象a
    B b;      // 定义派生类对象b
    a.x = 5; //Error
    b.x = 10; //Error
    b.y = 20; //Error
    a.setx(5); //OK
    b.setx(10); //Error
    b.setxy(10, 20); //OK
}
```

# 不同继承方式的基类特性和派生类特性

- 公有继承时，派生类的对象可以访问基类中的公有成员，派生类的成员函数可以访问基类中的公有成员和保护成员。
- 私有继承时，基类的所有成员不能被派生类的对象访问，而派生类的成员函数可以访问基类中的公有成员和保护成员。

# protected继承

- 基类的public和protected成员都以protected身份出现在派生类中，但基类的private成员不可直接访问；
- 派生类中的成员函数可以直接访问基类中的public和protected成员，但不能直接访问基类的private成员；
- 通过派生类的对象不能直接访问基类中的任何成员；

# 不同继承方式的基类特性和派生类特性

| 继承方式 | 基类特性 | 派生类特性 |
| --- | --- | --- |
| 公有继承 | public | public |
| | protected | protected |
| | private | 不可访问 |
| 私有继承 | public | private |
| | protected | private |
| | private | 不可访问 |
| 保护继承 | public | protected |
| | protected | protected |
| | private | 不可访问 |

PART 02

# 派生类的实现

# 恢复访问控制方式

- 若希望基类非隐藏的成员的访问控制方式通过继承在派生类中的身份不要改变，即保持在基类中声明的访问控制方式，这可通过恢复访问控制方式的声明来实现。

```
访问控制方式:
        基类名::基类成员名;
```

- 其中，访问控制方式是protected、public之一，基类成员名是要恢复访问控制权限的基类成员名，只写成员名即这种声明是在派生类中针对基类的某个成员进行的。

# 恢复访问控制方式

```
class A {
   public:
      void f() { cout<<”f”<<endl; }
      void g() { cout<<”g”<<endl; }
};
class B: private A {
   public:
      void h() { cout<<”h”<<endl; }
      A::f;
      //恢复从基类继承的f()的访问控制权限
};
```

```
void main( ) {
      B b;
      a.f(); //OK
      a.g(); //OK
      b.f(); //OK
      b.g(); //error
      b.h(); //OK
}
```

# 继承成员的重命名和重定义

- C++的类继承中派生类默认将基类的成员全盘接收，这样派生类就包含了除了构造函数和析构函数之外的基类所有成员。在程序中有时需要对基类成员进行改造或调整。在派生类中对继承成员可以重命名（rename）或重定义（override）。
- 当派生类与基类中有相同成员时，若未明确指明，则通过派生类对象使用的是派生类中的同名成员；如要通过派生类对象访问基类中被覆盖的同名成员，应使用基类名限定。

# 继承成员的重命名和重定义

```
class Point {
    int x, y;
    public:
        void move(int m, int n) { x+=m; y+=n; }
        int area(){ return 0; }
};
class Rectangle: public Point{
    int w, h;
    public:
       void shift(int i,j) { Point::move(i,j); }
       //继承的公有成员move()重命名为shift()
       int area() { return w*h; }
       //对继承的公有成员area()重定义
};
```

```
Rectangle r(2, 3, 20, 10);
r.shift(3, 2);
cout<<r.Getx()<<r.Gety( )
<<r.Getw( )<<r.Geth( );
cout<<r.area( )
<<r.Point::area();
//结果:
        5,5,20,10
        200 0
```

# 公有继承下的赋值兼容规则

- C++中以public方式继承的派生类可以看成基类的子类型，所谓赋值兼容规则指的是不同类型的对象间允许相互赋值的规定；
- 在公有派生的情况下，允许将派生类的对象赋值给基类的对象，但反过来却不行，即不允许将基类的对象赋值给派生类的对象。这是因为一个派生类对象的存储空间总是大于它的基类对象的存储空间。若将基类对象赋值给派生类对象，这个派生类对象中将会出现一些未赋值的不确定成员。

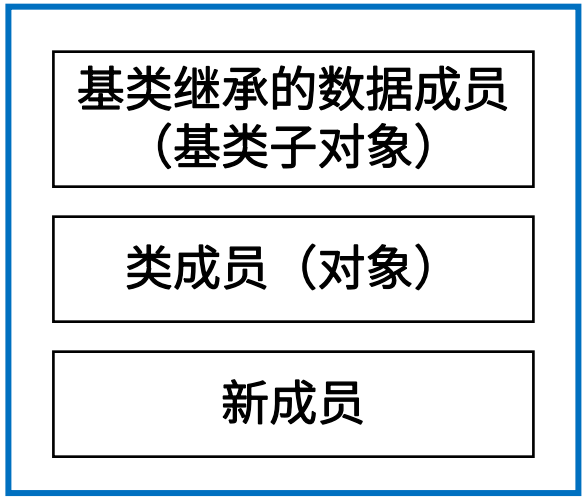
# 公有继承下的赋值兼容规则

- 将派生类对象赋值给基类对象有三种方法：
- 直接将派生类对象赋值给基类对象；
- 定义派生类对象的基类引用；
- 用基类对象指针指向它的派生类对象；
- 当派生类对象赋值给基类或基类指针后，只能通过基类名或基类指针访问派生类中从基类继承来的成员，不能访问派生类中的其它成员。

```
//假设Derived是Base的派生类
Base objB;
Derived objD;
ObjB=objD;   //OK
ObjD=objB;   //Error
Base &b=objD; //OK
Base *pb=&objD; //OK
```

# 派生类的构造

- 派生类对象由基类中继承的数据成员和派生类新定义的数据成员共同构成。
- 在派生类对象中由基类继承的数据成员和成员函数所构成的封装体称为基类子对象。
- 由于构造函数不能被继承，派生类的构造函数除了对派生类新增数据成员进行初始化外，还需要承担为基类子对象初始化的任务，即为（调用的）基类的构造函数提供参数。
- 另外，对于含有其它类对象成员的派生类，还要负责这些对象成员的初始化提供参数，以完成派生类对象的整个初始化工作。



# 派生类的构造

- 派生类构造函数的定义格式如下（采用成员初始化列表形式）

```
派生类名::派生类名(<参数表>)：基类构造函数(<参数表>)，其它成员(<参数>) {
        …;      //派生类新增数据成员的初始化
}
```

- 派生类构造函数执行的一般顺序是：
  - 基类构造函数，
  - 派生类对象成员类的构造函数（如果有的话），
  - 派生类构造函数体中的内容。

# 派生类的构造

```
class A {     //定义基类
    private: int a;
    public: A(int x) {
        a=x; cout<<"A construct "; }
};
class B {     //定义成员类
    private: int b ;
    public: B(int x) {
        b=x; cout<<"B construct "; }
};
```

```
class C: public A{ //定义派生类
    private: B obj_b ;
    public: C(int x, y):A(x), obj_b(y){
        cout<<"C's construct ";
    }// 派生类构造函数
};
C c(1,2);
//输出:
  A construct B construct C construct
```

# 派生类的构造

- 派生类构造函数提供了将参数传递给基类构造函数的途径，以保证在基类进行初始化时能够获得必要的数据。因此，如果基类的构造函数定义了一个或多个参数时，派生类必须定义构造函数。
- 如果新增成员中包括成员对象，成员对象的处理情况与基类相同。
- 如果基类中定义了默认构造函数时，在派生类构造函数的定义中可以省略对基类构造函数的调用。

# 派生类的析构

- 派生类析构函数的功能是在该类对象释放之前进行一些必要的清理工作，其定义与一般类的析构函数的定义完全相同，只要在函数体中完成派生类新增的非对象成员的清理工作。
- 派生类析构函数的执行顺序（与构造函数时的顺序相反）：
  - 先执行派生类的析构函数；
  - 再执行对象成员类的析构函数（如果派生类有）；
  - 最后执行基类的析构函数。

# 派生类的析构

```
class X {
    private: int x1, x2;
    public:
    X (int i,int j) { x1 = i; x2 = j; }
    void print() {
       cout<< x1 << ”,” << x2 ;}
    ~X() {cout << ”X destruct.”<<endl; }
};
```

```
class Y:public X{
    private: int y;
    public:
    Y(int i, j, k): X(i, j) { y = k; }
    void print( ) {
       X::print(); cout<<”,”<<y<<endl; }
    ~Y() { cout<< ”Y destruct. ”; }
};
```

```
Y y1(5,6,7), y2(2,3,4);
y1.print();
y2.print();
```

```
//输出:
5,6,7 2,3,4
Y destruct. X destruct.
Y destruct. X destruct.
```

# 感谢倾听

给个好评！