



华中科技大学

面向对象程序设计

第六章 单继承类

许向阳

xuxy@hust.edu.cn



- 6.1 单继承类
- 6.2 派生控制
- 6.3 成员访问
- 6.4 派生类的构造函数与析构函数
- 6.5 派生类的存储空间
- 6.6 父类和子类



6.1 单继承类

类的层次性、继承和派生

- 继承是C++类型演化的重要机制
- 通过继承，新类具有原有类型的属性和行为
- 新类只需定义原有类型没有的数据成员和函数成员
- 接受成员的新类称为派生类
- 提供成员的原有类型称为基类
- C++既支持单继承又支持多继承
- 单继承只能获取一个基类的属性和行为
- 多继承可获取多个基类的属性和行为





6.1 单继承类

单继承的定义格式

```
class <派生类名>: 【<派生控制符/继承方式>】 <基类名>
{
    <派生类新定义成员>
    <派生类重定义基类同名的数据和函数成员>
    <派生类声明恢复基类成员访问权限>
};
```

<派生控制符>指明派生类采用什么方式从基类继承成员，

private: 私有继承基类；

protected: 保护继承基类；

public: 公有继承基类。

使用class定义时，缺省的派生控制符为private；





6.1 单继承类

```
class A
{
    protected:
        int x;
    private:
        int y;
    public:
        void setx(int m) { x=m; }
        void sety(int n) { y=n; }
        int getx() const { return x; }
        int gety() const { return y; }
};
```



6.1 单继承类

```
class B : protected A
{
    private:
        int b1,b2;
    public:
        void setBx(int m) { x=m; }
        void setBy(int n) { sety(n); }
        int getBx() const { return x; }
        int getBy() const {
            return gety();
        }
        int getsum() { return x+gety(); }
};

B b;
cout<<" x= "<< b . getBx() <<endl;
```

```
A :
protected : x
private : y
public: setx
        sety
        getx
        gety
```

继承哪些成员？





6.1单继承类

```
A a;  
B b;  
cout<<" x= "<< b . getX() <<endl;
```

监视 1

名称	值
a	{x=-858993460 y=-858993460 }
b	{b1=-858993460 b2=-858993460 }

名称	值
a	{x=-858993460 y=-858993460 }
x	-858993460
y	-858993460
b	{b1=-858993460 b2=-858993460 }
A	{x=-858993460 y=-858993460 }
x	-858993460
y	-858993460
b1	-858993460
b2	-858993460

名称	值
&b.x	0x00c3fb9c
&b.y	0x00c3fba0
&b.b1	0x00c3fba4
&b.b2	0x00c3fba8
&b	0x00c3fb9c {b1=-858993460 b2=-858993460 }





6.2 派生控制

类定义中：

- public 成员： 可被任何类访问，公开的
- protected成员： 可被本类及其派生类访问
- private成员： 只限于本类访问

派生控制： 指明基类的成员在派生类的访问方式





6.2 派生控制

继承类型	基类 成员特性	派生类 成员特性
公有继承	公有	公有
	保护	保护
	私有	不可访问
保护继承	公有	保护
	保护	保护
	私有	不可访问
私有继承	公有	私有
	保护	私有
	私有	不可访问

在派生类中，实际上是继承了所有基类的成员，而只是在写程序时不可访问

相当于将基类的成员定义拷贝到派生类中。





6.2 派生控制

```
class B : protected A
{
public:
    void setBx(int m) { x=m; }
    void setBy(int n) { y=n;
                        sety(n);
    }
    int getBx() const { return x; }
    int getBy() const { // return y;
                        return gety();
    }
    int getsum() { return x+gety(); }
};

B b;
cout<<" x= "<< b . getBx() <<endl;
cout<<" x= "<< b . getx() <<endl;
```

A :

protected : x

private : y

public: setx

 sety

 getx

 gety

继承的成员各有什么访问权限？

哪些语句不正确？

B :

protected :

 x

 setx

 sety

 getx

 gety

public:

 setBx

 setBy

 getBx

 getBy

 getsum





6.2 派生控制

派生类声明恢复基类成员访问权限

```
Class B : private A
```

```
{ public:
```

```
    A::getx;    // getx后面没有括号
```

```
protected:
```

```
    A::x;       // 数据成员
```

```
private:
```

```
    A::y;       // 不可访问 A类的私有成员
```

```
}
```

```
cout<<" x= "<< b . getx() <<endl;
```



6.3 成员访问

- 访问从基类继承的成员时要注意其访问权限。
- 在派生类和基类有同名成员时，注意区分同名成员。
- 标识符的作用范围越小，被访问到的优先级越高。
要访问作用范围更大的标识符，用“类名::”进行限定。
- 面向对象作用域关于标识符的作用范围从小到大：
 - ① 作用于函数成员内（局部变量）；
 - ② 作用于类或者派生类内；
 - ③ 作用于基类内（基类的成员在派生类也可访问到）；
 - ④ 作用于虚基类内。





6.3 成员访问

```
class A {  
public:    int x;  
    int getx() {cout<< "class A" <<endl;  return x;}  
};  
class B : public A  
{    public:  
    int getx() { cout<< "class B" <<endl;  return x;  }  
};
```

```
B  b;
```

```
Z=b.getx();
```

```
Z=b.A::getx();    // 为何派生控制中要有public, 此处才正确
```

```
A  *p= &b;
```

```
Z= p->getx();    // 显示什么?    class A
```





6.4 派生类的构造函数与析构函数

- ◆ 创建一个对象时，调用相应的构造函数进行初始化
- ◆ 派生类不能继承基类的构造函数、析构函数
- ◆ 派生类有自己的构造函数和析构函数

派生类构造函数的格式：

派生类名(参数表1)：基类名(参数表2)

{ 构造函数体 }

- ◆ 若无“:基类名(参数表2)”，执行基类中无参数的构造函数
- ◆ 基类中的构造函数先于派生构造函数体执行
- ◆ 基类中的构造、析构函数都要有相应的访问权限
不能是 private





6.4 派生类的构造函数与析构函数

```
class Animal {  
public: int age;  
private: int weight;  
public:  
    Animal(int a,int w) {  
        cout<<" Animal with parameter"<<endl;  
        age = a;    weight =w;    }  
  
    Animal() {  
        cout<<" Animal without parameter"<<endl;  
        age = 1; weight =1;    }  
  
    ~Animal() {cout<<"destructing Animal "<<endl; }  
    void setage(int a=0) { age=a; }  
    int  getage( ) { return age; }  
};
```





6.4 派生类的构造函数与析构函数

```
class Dog : public Animal {  
private:
```

```
    char pinzhong[10];
```

```
public:
```

```
    Dog(char *pz,int a,int w) : Animal( a, w )
```

```
    {  
        strcpy(pinzhong,pz);
```

```
        cout<<" constructing Dog "<<endl;
```

```
    }
```

```
    ~Dog() { cout<<"destructing Dog "<<endl;
```

```
    }
```

```
    void set_dogage(int a)
```

```
    { setage(a); }
```

```
};
```

Dog MyDog("警犬",10,100);





6.4 派生类的构造函数与析构函数

◆ 单继承**派生类**只有一个基类或虚基类，根据4个优先级别容易确定构造顺序：

- ① 调用**派生类**虚基类的构造函数；无论列出否，总会被执行；
- ② 调用**派生类**基类的构造函数，无论列出与否，总会被执行；
- ③ 按照**派生类**数据成员声明的顺序，依次初始化数据成员或调用相应构造函数，**对象成员**无论列出与否总会被构造；
- ④ 最后执行**派生类**的构造函数体。

◆ **基类、对象成员不列出时自动调用无参构造函数。**

◆ 若虚基类或基类只定义了带参数的构造函数，
则派生类需要使用带参数的构造函数。





6.4 派生类的构造函数与析构函数

构造函数与析构函数的执行顺序

```
class A{
private:  int a;
public:
    A():a(8) { cout<<a; }
    A(int x):a(x) { cout<<a; }
    ~A(){ cout<<a; }
};

class B: private A{
private:  int b, c;
    const int d;          //有只读成员d, 故B必须定义构造函数
    A x, y, z;
public:
    B(int v):b(v), y(b+2), x(b+1), d(b), A(v) {
        c=v;  cout<<b<<c<<d; cout<<"C";  //b, c可再次赋值
    }
    ~B(){ cout<<"D"; }
};

void main(void){ B z(1); }
// 输出结果: 1238111CD8321
```

C6_init_order





6.4 派生类的构造函数与析构函数

- ◆ 变量 r 引用一个对象 V 时，由对象 V 完成构造和析构。

A oa(...); A &r=oa;

不需要考虑 r 空间的释放，r 指向的 oa 会自动析构。

- ◆ r 引用的对象是用 new 生成的，
应使用 delete &r 析构对象，否则将产生内存泄漏。

A &r = * new A(10); delete &r;

- ◆ 若被 r 指向的对象是用 new 生成的，
用 delete r 析构对象，否则将产生内存泄漏。

A *r = new A(10); delete r;





6.4 派生类的构造函数与析构函数

```
class A{
    int *s;
public:
    A(int x) { s=new int[x];}
    ~A( ) { if(s) { delete s; s=0; } }
};

void main( )
{
    A x(3);
    A &r=x;    // main返回时会自动析构x, r代表对象x
    A &p = *new A(8);
    A *q=new A(9); // 分配一块空间后, 构造对象(s分配一块空间)
    delete    q;    // 释放对象占用空间之前自动调用析构函数释放s
    delete    &p;
}
```





6.5 派生类的存储空间

- 在派生类对象的存储空间中包含了基类的数据成员。
- 在构造派生类对象之前，首先构造或初始化基类对象的存储空间，作为派生类对象存储空间的一部分。
- 在计算派生类对象存储空间时
基类和派生类的**静态数据成员**都不应计算在内。



6.5 派生类的存储空间

```
#include <iostream>
using namespace std;
class A{
private:
    int h, i, j;
    static int k;
};
class B: A{ //等于class B: private A
    int m, n, p;
    static int q;
};
int A::k=0; //静态数据成员必须初始化
int B::q=0;
int main(void){
    cout<<"Size of int="<<sizeof(int)<<"\n";
    cout<<"Size of A="<<sizeof(A)<<"\n";
    cout<<"Size of B="<<sizeof(B)<<"\n";
    return 0;
}
```

输出

Size of int=4
Size of A=12
Size of B=24

int h	A	
int i		
int j		
int m	B	
int n		
int p		



6.6 父类和子类

➤ 如果派生控制为public

派生类称为基类的**子类**，基类则称为派生类的**父类**；

➤ C++允许**父类指针**指向子类对象、**父类引用**子类对象，
无须通过强制类型转换，
编译时按父类说明的权限访问成员；

➤ 通过父类指针调用虚函数时进行晚期绑定；

➤ 根据对象的实际类型绑定到合适的成员函数；

➤ 父类指针实际指向的对象的类型不同；

➤ 虚函数绑定的函数的行为不同，从而产生多态。





6.6 父类和子类

```
class POINT{
    int x, y;
public:
    int getx( ){ return x; }    int gety( ){ return y; }
    void show( ){ cout<<"Show a point\n"; }
    POINT(int x,int y): x(x), y(y){ }
};
class CIRCLE: public POINT{// 公有继承, 基类和派生类构成父子关系
    int r;
public:
    int getr( ){ return r; }
    void show( ){ cout<<"Show a circle\n"; }
    CIRCLE(int x, int y, int r):POINT(x,y){ CIRCLE::r=r; }
} c(3,7,8);
void main(void){
    POINT *p=&c;// 父类指针p指向子类对象, 不用类型转换
    p=(POINT*)&c
    cout<<c.getr( )<<p->getx( );// 不能用p->getr( ),
                                // 编译时无POINT::getr( )
    p->show( );    // p指向子类对象, 但调用的是POINT::show( )
}
```





6.6 父类和子类

◆ 在派生类函数成员内部

基类指针可以直接指向该派生类对象

基类被等同地当作派生类的父类

- ◆ 如果一个函数声明为基类的友元，则该友元内部的基类指针也可以直接指向派生类对象。即对基类的友元而言，基类被等同地当作派生类的父类。
- ◆ 在上述两种情况下，基类引用可直接引用派生类对象。





6.6 父类和子类

```
class VEHICLE{  
    int speed, weight, wheels;  
public:  
    VEHICLE(int spd, int wgt, int whl);  
    friend void main( );  
};  
VEHICLE::VEHICLE(int spd, int wgt, int whl){  
    speed=spd;  weight=wgt;  wheels=whl;  
}
```



6.6 父类和子类

```
class CAR: private VEHICLE{// 非父子关系: private
    int seats;
public:
    VEHICLE *who( );
    CAR(int sd, int wt, int st): VEHICLE(sd, wt, 4) { seats=st; };
};
VEHICLE *CAR::who( ){
    VEHICLE *p=this; //派生类内的基类指针直接指向派生类对象
    VEHICLE &q=*this;//派生类内的基类引用直接引用派生类对象
    return p;
}
//main为基类友元, 基类和派生类构成父子关系
void main(void) {
    CAR c(1, 2, 3);
    VEHICLE *p=&c;
}
```





总结

- 单继承类
- 派生控制、派生类对象可访问的成员
- 成员访问、同名成员的优先级
- 派生类的构造与析构顺序
- 派生类对象的数据存储模式
- 父类对象和子类对象的指针