Ch-11

继承与派生

主要内容

- 继承与派生的概念
- 派生类的声明方式
- 派生类的构成
- 派生类成员的访问属性
- 类型兼容规则
- 派生类的构造函数和析构函数
- 多重继承
- 基类与派生类的转换
- 继承与组合
- 继承在软件开发中的重要意义

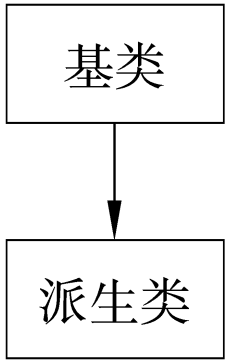
11.1 继承与派生的概念

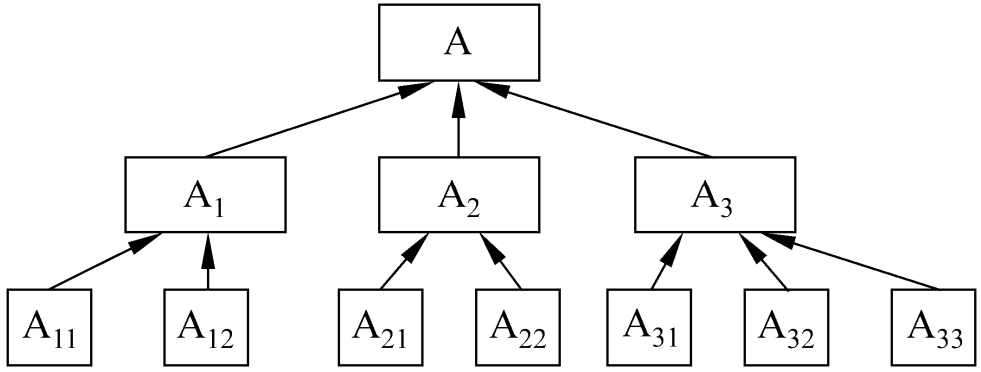
□ 定义

所谓继承是在已存在的类A 的基础上建立一个新类B。

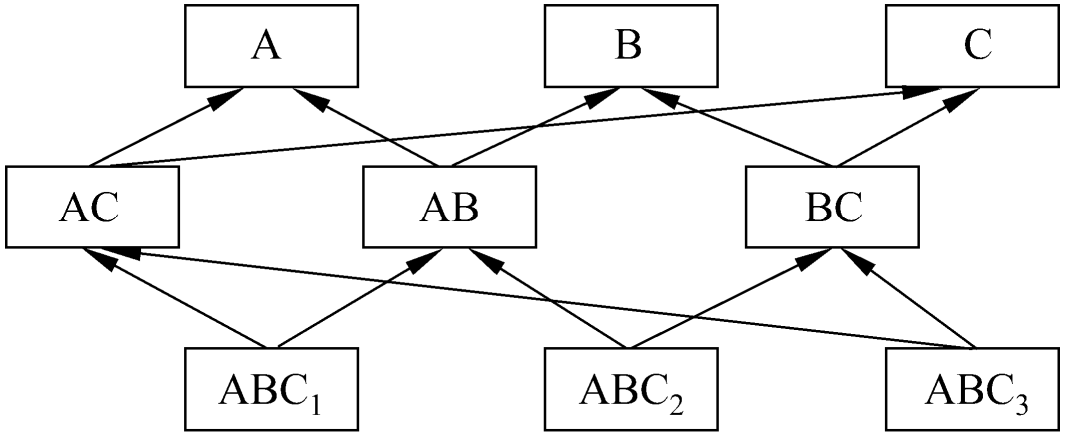
类A称为基类或父类，类B 称作派生类或子类。子类从父类获得其已有的特性，这种现象称作类的继承。从另一个角度看从父类产生子类，称作类的派生。

- 一个基类可以派生出多个派生类，每个派生类又可以作为基类再派生出新的派生类。一个派生类只从一个基类派生，称作单继承。





- **一个派生类也可从多个基类派生，也就是说一个派生类可以有两个或多个基类。一个派生类有两个或多个基类的称作多重继承。**



基类和派生类的关系可以表述为：派生类是基类的扩充，而基类是派生类的抽象。

11.2 派生类的声明方式

口 派生类的声明格式

class 派生类名: [继承方式] 基类名

{ 派生类新增成员声明 };

说明：（1）继承方式包括：public、private、protected；

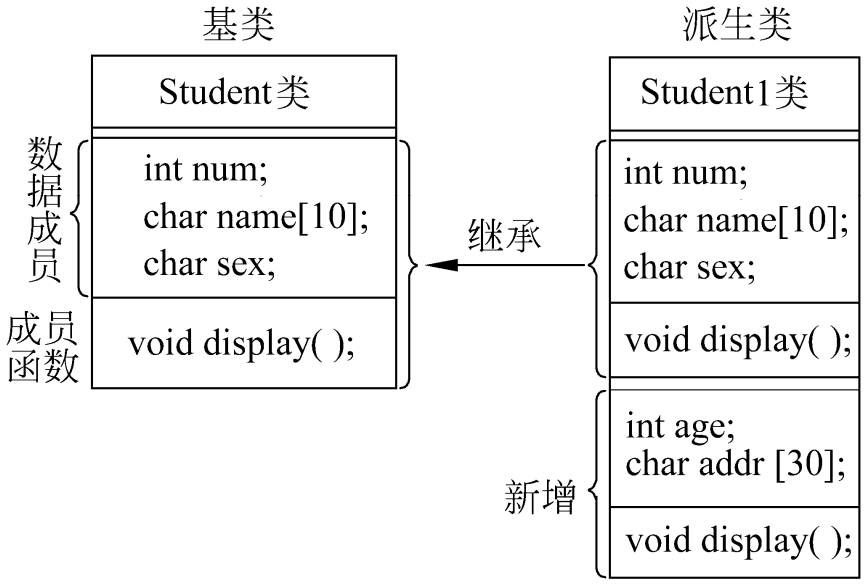
（2）如果省略，系统默认为private。

例如：class Student1: public Student

{ … }

11.3 派生类的构成

派生类中的成员包括从基类继承过来的成员和自己增加的成员。继承基类成员体现了同一基类的派生类都具有的共性，而新增加的成员体现了派生类的个性。



派生类的构造：

（1）从基类接收成员。 派生类将基类除构造函数和析构函数外的所有成员接收过来。

（2）调整从基类接收的成员。 一方面可以通过继承方式改变基类成员在派生类中的访问属性，另一方面可以在派生类中声明一个与基类成员同名的成员屏蔽基类的同名成员，注意如是成员函数不仅要函数名相同，而且函数的参数也要相同，屏蔽的含义是用新成员取代旧成员。

（3）在声明派生类时增加成员，它体现了派生类对基类功能的扩充。

（4）在声明派生类时，还要自己定义派生类的构造函数和析构函数。

11.4 派生类成员的访问属性

11.4.1 公有继承

在定义派生类时将基类的继承方式指定为public，称为公有继承。

公有基类的成员在派生类中的访问属性

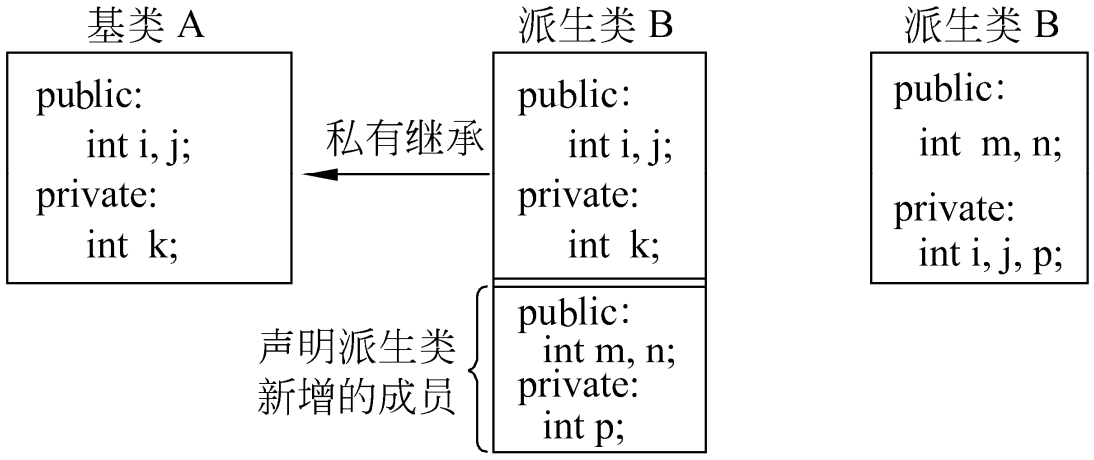
在基类中的访问属性	继承方式	在派生类中的访问属性
private	public	不可访问
public	public	public
protected	public	protected

11.4.2 私有继承

在定义派生类时将基类的继承方式指定为private，称为私有继承。

私有基类的成员在派生类中的访问属性

在基类中的访问属性	继承方式	在派生类中的访问属性
private	private	不可访问
public	private	private
protected	private	private



11.4.3 保护成员和保护继承

保护成员不能被类外访问，这点与私有成员一样，但是保护成员可以被派生类的成员函数引用，而私有成员则不能。

在定义派生类时将基类的继承方式指定为protected，称为保护继承。

保护基类的成员在派生类中的访问属性

在基类中的访问属性	继承方式	在派生类中的访问属性
private	protected	不可访问
public	protected	protected
protected	protected	protected

基类成员在派生类中的访问属性

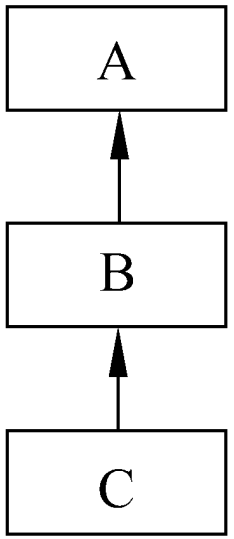
基类成员	在公有派生类中的访问属性	在私有派生类中的访问属性	在保护派生类中的访问属性
私有成员	不可访问	不可访问	不可访问
公有成员	公有	私有	保护
保护成员	保护	私有	保护

派生类成员的访问属性

派生类成员	在派生类中的访问属性	在派生类外的访问属性
私有成员	不可访问	不可访问
公有成员	公有	私有
保护成员	保护	私有

11.4.4 多级派生时的访问属性

如右图所示，类A为基类，类B是类A的派生类，类C是类B的派生类，则：



（1）类C也是类A的派生类，类B是类A的直接派生类，类C是类A的间接派生类；

（2）类A是类B的直接基类，是类C的间接基类。

例11.1 多级派生

```
class A    //基类
{
private:
int k_a;
public:
int i_a;
protected:
void f_a( );
int j_a;
};
```

```
class B: public A    //public继承
{
private:
int m_b;
public:
void f1_b ( );
protected:
void f2_b ( );
};
```

```
class C: protected B    //protected继承
{
private:
int n_c;
public:
void f_c( );
};
```

多级派生结构中，各成员在不同类中的访问属性

	i_a	f_a	j_a	k_a	f1_b	f2_b	m_b	f_c	n_c
类A	公有	保护	保护	私有					
类B (:public A)	公有	保护	保护	不可访问	公有	保护	私有		
类C (:protected B)	保护	保护	保护	不可访问	保护	保护	不可访问	公有	私有

□ 多级派生中的类型兼容规则

一个公有派生类的对象在使用上可以被当作基类的对象，反之则禁止。具体表现在：

- 派生类的对象可以被赋值给基类对象；
- 派生类的对象可以初始化基类的引用；
- 指向基类的指针也可以指向派生类；
- 通过基类对象名、指针只能使用从基类继承的成员。

例11.2 多级派生中的类型兼容

```
class A{   //基类B0声明
   public:    //公有成员函数
      void display() {
         cout<<"A::display()"<<endl;
      }
};
class B: public A {
   public:
      void display() {
         cout<<"B::display()"<<endl;
      }
};
class C: public B {
   public:
      void display() {
         cout<<"C::display()"<<endl;
      }
};
```

```
void print(A *ptr) {
   ptr->display();
}
int  main() {
   A a;//声明A类对象
   B b;//声明B类对象
   C c;//声明C类对象
   A *p;//声明A类指针
   p=&a;  //A类指针指向A类对象
   print(p);
   p=&b;  //A类指针指向B类对象
   print(p);
   p=&c;  //A类指针指向C类对象
   print(p);
   return 0;
}
```

```
A::display()
A::display()
A::display()
```

□ 基类与派生类的对应关系

- **单继承**

  派生类只从一个基类派生。

- **多继承**

  派生类从多个基类派生。

- **多重派生**

  由一个基类派生出多个不同的派生类。

- **多层派生**

  派生类又作为基类，继续派生新的类。

11.5 派生类的构造函数和析构函数

□ 继承与派生的目的

- 继承的目的：实现代码重用。
- 派生的目的：当新的问题出现，原有程序无法解决（或不能完全解决）时，需要对原有程序进行改造。

□ 派生类的构造函数

- 基类的构造函数不被继承，派生类中需要声明自己的构造函数。
- 声明构造函数时，只需要对本类中新增成员进行初始化，而对继承来的基类成员调用基类构造函数进行初始化。
- 派生类的构造函数需要给基类的构造函数传递参数。

11.5.1 简单的派生类的构造函数

简单派生类只有一个基类，而且只有一级派生，在派生类的数据成员中不包含基类的对象（即子对象）。

在定义派生类的构造函数时除了对自己的数据成员进行初始化外，还必须调用基类的构造函数初始化基类的数据成员。构造函数格式如下：

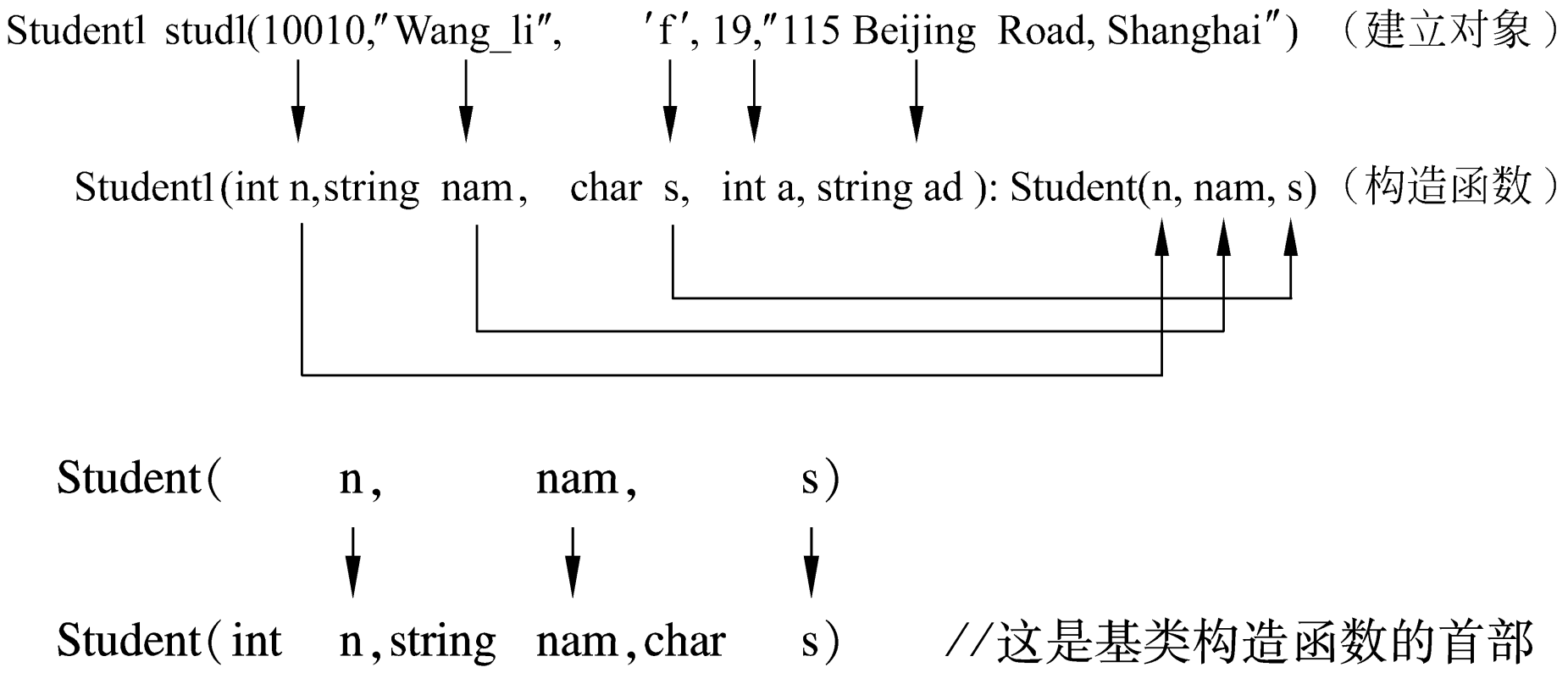
派生类构造函数名(总参数表): 基类构造函数名(基类参数表)

{ 本类新增成员数据赋值语句; };

说明：（1）派生类构造函数名后的总参数表包括基类和派生类的构造函数的形参（类型+形参变量名）；（2）基类参数表是传递给基类构造函数的实参，是用派生类构造函数总参数表中的基类形参变量做基类构造函数的实参。

例11.3 简单派生类的构造函数

代码见11.3.cpp



口 简单派生类的构造函数和析构函数的执行顺序：

- 在建立一个对象时，执行构造函数的顺序是：① 派生类构造函数先调用基类构造函数；② 再执行派生类构造函数本身（即派生类构造函数的函数体）。按上面的例子说，先初始化num、name和sex，然后再初始化age和addr。
- 释放派生类对象时，先执行派生类析构函数，再执行其基类的析构函数。

11.5.2 有子对象的派生类的构造函数

类的成员数据除了是内置类型或系统提供的类类型如string类，还可以是自定义的类类型。

系统在建立派生类对象时调用派生类构造函数对子对象进行初始化，但不能在声明派生类时对子对象初始化。

□ 派生类构造函数的任务包括：

- 对基类的成员数据初始化；
- 对子对象的成员数据初始化；
- 对派生类的成员数据初始化。

- 派生类构造函数的一般形式：

  派生类构造函数名(总参数表): 基类构造函数名(参数表), 子对象名(参数表)

  { 派生类新增成员数据的初始化语句; }

- 执行派生类构造函数的顺序是：

  ① 调用基类构造函数，初始化基类成员数据；

  ② 调用子对象构造函数，初始化子对象成员数据；

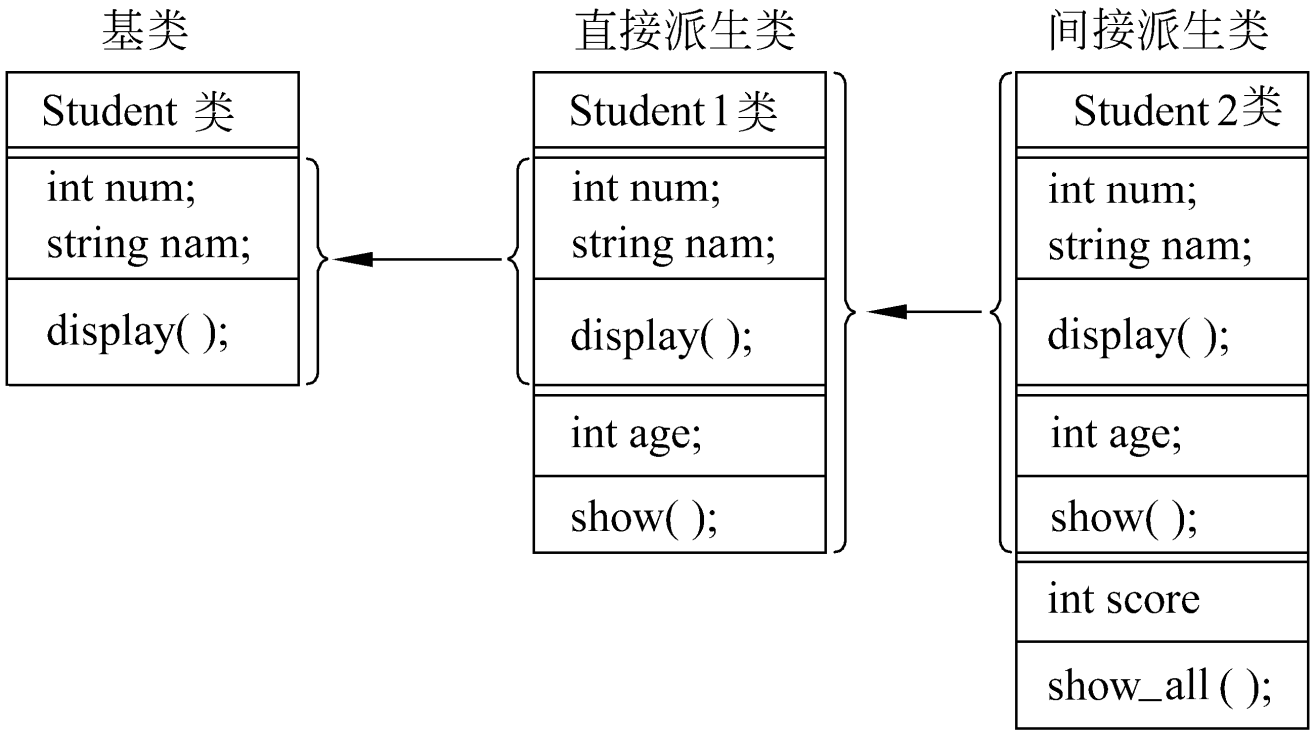
  ③ 执行派生类构造函数，初始化派生类成员数据。

  编译系统会根据参数名（而不是参数的顺序）决定各参数表中参数之间的传递关系。如有多个子对象，要逐个列出子对象及其参数表。

```
class Student1: public Student {
public:
Student1(int n, string nam, char s, int n1, string nam1, char s1, int a, string ad)
: Student(n, nam, s), monitor(n1, nam1, s1) {
age=a;
addr=ad;
}
void show();
private:
int age;
string addr;
Student monitor;    //子对象monitor
};
```

11.5.3 多层派生时的构造函数

如图所示的多层派生类：



按照前面派生类构造函数的规则逐层写出各个派生类的构造函数：

- **基类的构造函数首部：**

  Student(int n, string nam);

- **派生类Student1的构造函数首部：**

  Student1(int n, string nam, int a): Student(n, nam);

- **派生类Student2的构造函数首部：**

  Student2(int n, string nam, int a, int s): Student1(n, nam, a);

1、写派生类构造函数的规则是：只须调用其直接基类的构造函数即可，不要列出每一层派生类的构造函数。

2、在声明Student2类对象时，调用Student2构造函数，在执行Student2构造函数时，先调用Student1构造函数，在执行Student1构造函数时，先调用基类Student构造函数。初始化的顺序是：①先初始化基类的数据成员num和name；②再初始化Student1的数据成员age；③最后初始化Student2的数据成员score。

11.5.4 派生类构造函数的特殊形式

1、当不需要对派生类新增成员进行初始化时，派生类构造函数的函数体可以为空。

2、如果在基类里没有定义构造函数，或定义了没有参数的构造函数，在定义派生类构造函数时可以不写基类构造函数。因为此时派生类构造函数没有向基类构造函数传递参数的任务。在调用派生类构造函数时，系统地自动首先调用基类的默认构造函数。

3、如果在基类和子对象的类中都没有定义带参数的构造函数，也不需要对派生类自己的数据成员进行初始化，可以不定义派生类构造函数。

4、如果在基类或子对象的类声明里定义了带参数的构造函数，就必须定义派生类构造函数，并在派生类构造函数中写出基类或子对象类的构造函数及其参数表。

5、如果在基类既定义了无参数的构造函数也定义了有参数的构造函数，在定义派生类构造函数时，既可以包含基类构造函数及其参数，也可以不包含基类构造函数。

11.5.5 派生类的析构函数

类派生时，派生类不能继承基类的析构函数。

撤销派生类对象时，需要派生类的析构函数去调用基类的析构函数：

（1）首先执行派生类自己的析构函数，清理派生类新增加的成员：

（2）然后调用子对象类的析构函数清理子对象；

（3）最后调用基类析构函数清理基类的成员。

例11.4 单一继承时的构造函数、析构函数

```
3   class B
4   {
5       private:
6           int b;
7       public:
8           B();
9           B(int i);
10          ~B();
11          void Print() const;
12  };
13  B::B()
14  {
15      b=0;
16      cout<<"调用B的默认构造函数."<<endl;
17  }
18  B::B(int i)
19  {
20      b=i;
21      cout<<"调用B的构造函数." <<endl;
22  }
23  B::~B()
24  {
25      cout<<"调用B的析构函数."<<endl;
26  }
27  void B::Print() const
28  {
29      cout<<"B's Print().  b="<<b<<endl;
30  }
```

```
31  class C:public B
32  {
33      private:
34          int c;
35      public:
36          C();
37          C(int i,int j);
38          ~C();
39          void Print() const;
40  };
41  C::C()
42  {
43      c=0;
44      cout<<"调用C的默认构造函数."<<endl;
45  }
46  C::C(int i,int j):B(i)
47  {
48      c=j;
49      cout<<"调用C的构造函数."<<endl;
50  }
51  C::~C()
52  {
53      cout<<"调用C的析构函数."<<endl;
54  }
55  void C::Print() const
56  {
57      B::Print();
58      cout<<"C's Print().  c="<<c<<endl;
59  }
```

```
60  int main()
61  {
62      C obj(5,6);
63      obj.Print();
64      return 0;
65  }
```

```
调用B的构造函数.
调用C的构造函数.
B's Print().  b=5
C's Print().  c=6
调用C的析构函数.
调用B的析构函数.
```

11.6 多重继承

11.6.1 声明多重继承的方法

```
class 派生类名: 继承方式1 基类名1,
                继承方式2 基类名2,
                ……
{ 派生类类体 };
```

例如：假定已声明了类A、类B和类C，则由它们派生出的类D的声明形式是：

```
Class D: public A, private B, protected C
{ D类新增的成员声明 }
```

11.6.2 多重继承派生类的构造函数

□ 多重继承派生类的构造函数的一般形式

派生类构造函数名(总参数表): 基类1构造函数(参数表),
基类2构造函数(参数表),
……
子对象名1(参数表)
……
{ 派生类新增成员赋值语句 }

说明：（1）派生类构造函数负责所有基类构造函数的调用。（2）派生类构造函数执行顺序：执行所有基类的构造函数-->执行所有子对象的构造函数-->执行派生类构造函数体。
（3）处于同一层次的各基类构造函数的执行顺序取决于定义派生类时所指定的各基类顺序，与派生类构造函数中所定义的成员初始化列表中的各项顺序无关；

11.6.3 多重继承引起的二义性问题

□ 二义性产生的原因

1、在多重继承时，基类与派生类之间，或基类之间出现同名成员时，将出现访问时的二义性——采用虚函数或同名隐藏规则来解决。

- 问题（代码在右侧）：

  如果定义C c1; 则c1.f(); 会产生二义性，因为不能识别调用的是类A的f函数还是类B的f函数。而 c1.g(); 无二义性（同名覆盖）。

- 解决办法：

  a.用作用域限定符区别出是类A还是类B的f函数；

  c1.A::f(); 或 c1.B::f();

  b.在类C中定义同名函数f()。

```
class A
{
      public:
            void  f();
};
class B
{
      public:
            void f();
 void g();
};
class C: public A, piblic B
{
      public:
 void g();
 void h();
};
```

2、当派生类从多个基类派生，而这些基类又有一个共同的基类，则在访问该共同基类中的成员时，将产生二义性——采用虚基类和同名覆盖来解决。

- 问题（代码在右侧）：

  如果定义C c1; 则c1.a与c1.A::a将产生二义性，因为不能识别是通过类B1还是类B2调用的类 A中的a。

- 解决办法：

  a.用作用域限定符区别出是通过类B1还是类B2调用的类A中的a；

  c1.B1::a; 或 c1.B2::a;

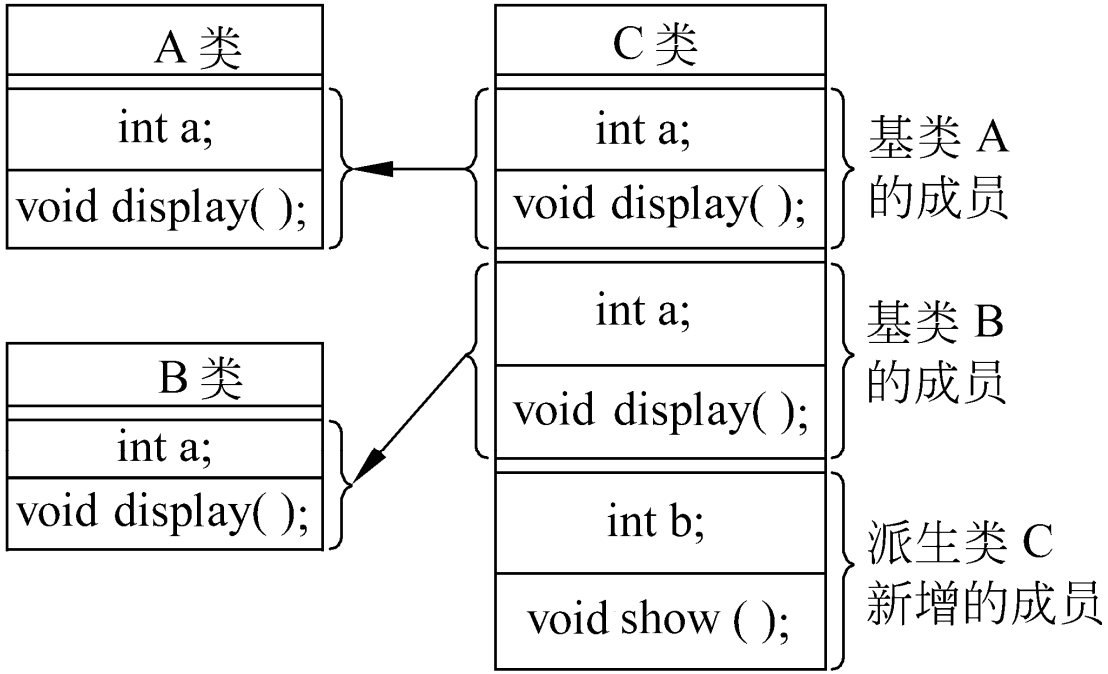
  b.虚基类。

```
class A
{
      public:
   int a;
};
class B1:public A
{
      private:
            int b1;
};
class B2:public A
{
      private:
            int b2;
}
class C:public B1, public B2
{
      public:
            int f();
      private:
            int c;
}
```

例：多重继承的同名问题

如果类A和类B都有成员函数display和成员数据a,类c是类A和类B的直接派生类。下面分别讨论3种情况：

（1）两个基类有同名成员。如图所示。



```
Class  A {
       public:
              int a;
              void display( );
};
Class  B {
       public:
              int a;
              void display( );
};
Class  C : public A, public B {
       public:
             int b;
             void show( );
};
```

如果在main函数中定义C类对象c1，并写出如下语句：

```
C c1;
c1.a=3;
c1.display();
```

由于基类A 和基类B都有数据成员a和成员函数display，系统无法辨别要访问的是A类的还是B类的，程序编译报错。

解决这个问题可以用基类名限定，如：

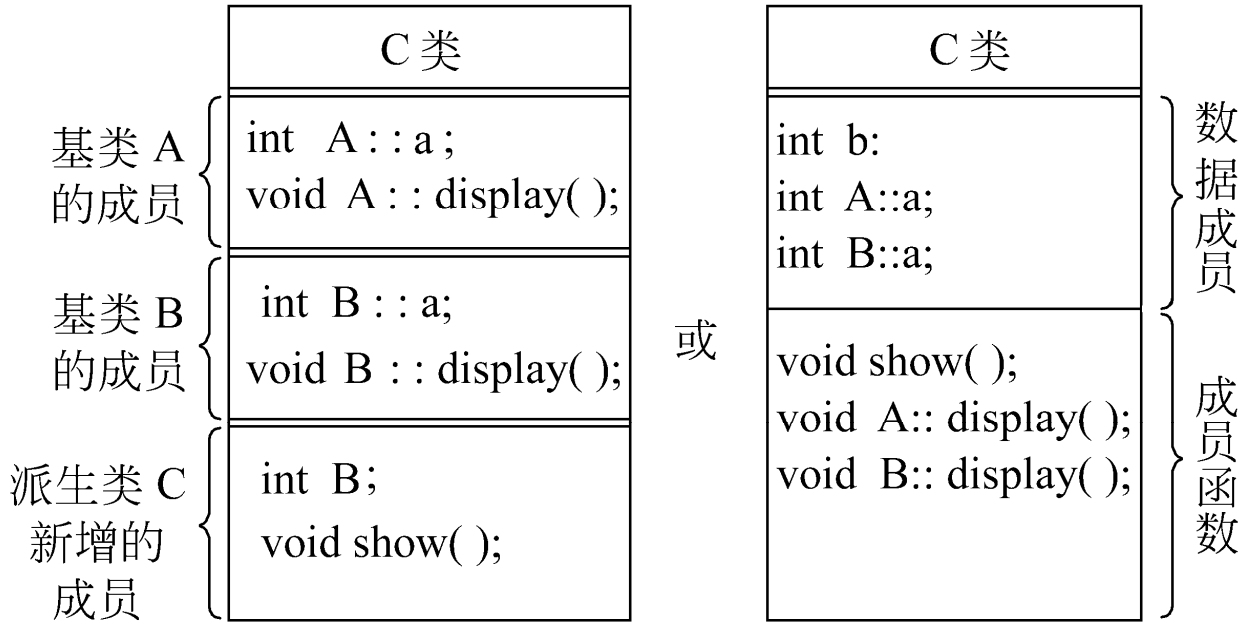
```
c1. A::a=3;
c1. A::display();
```

如果在派生类C中通过派生类成员函数show访问基类A的display和a，可以这样写：

A::a=3;

A::display();

为了清楚起见，前图应改用下图的形式表示。



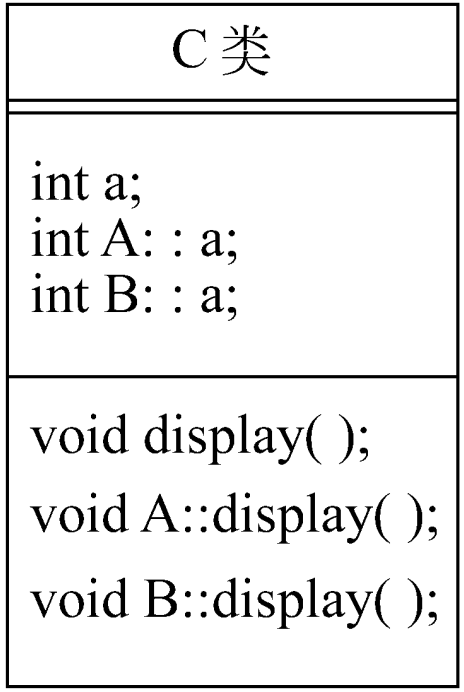
（2）两个基类和派生类三者都有同名成员

如果在main函数中定义C类对象c1：

```
C c1;
c1.a= 3;
c1.display( );
```

程序可以通过编译，也能正常运行，在语法上认为这样访问的是派生类C的成员数据a和成员函数display，规则是：派生类屏蔽基类的同名成员，对于带参数的成员函数除了要函数名相同外，参数的个数和类型都要相同才符合屏蔽条件。如果在派生类外访问基类A的成员，要指明作用域A：

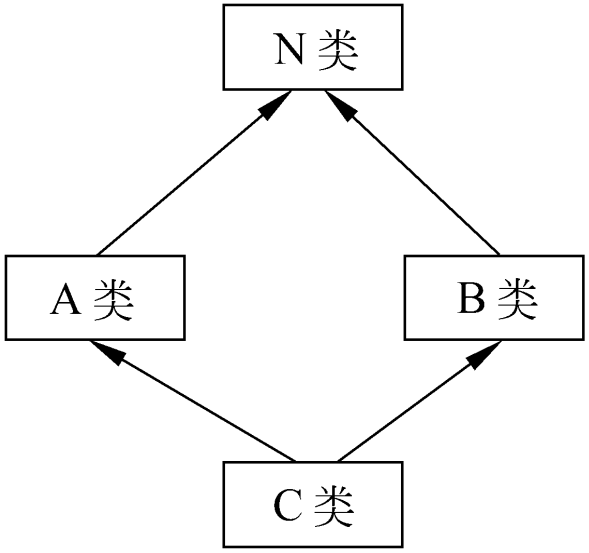
```
c1. A::a= 3;
c1. A::display( );
```




```
Class  A {
      public:
            int a;
            void display( );
};
Class  B {
      public:
            int a;
            void display( );
};
Class  C : public A, public B {
      public:
            int a;
            void display( );
};
```

（3）类A和类B又是从同一个基类派生的

```
Class N {
        public :
           int a;
           void display()
           { cout<<“N::a=“<<a; }
};
```




```
class  A : public N {
        public:
                int a1;
};
class  B:public N {
        public:
                int a2;
};
class  C : public A, public B {
        public:
                int a3;
                void show( )
                { cout<<“a3=“<<a3; }
};
int main() {  C c1; … }
```

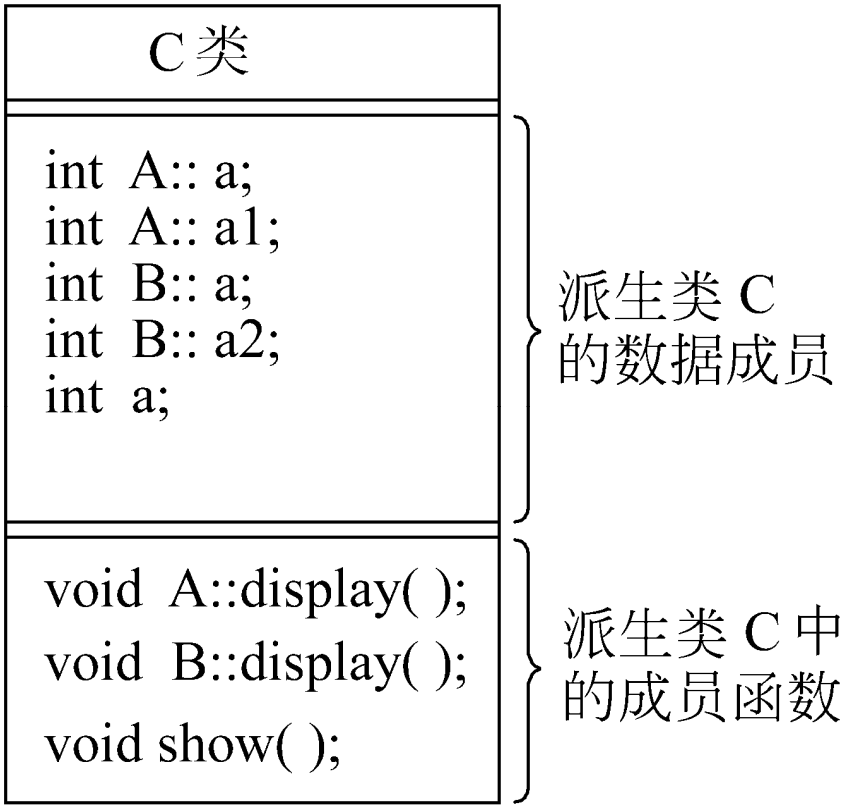
类A和类B分别从类N继承了数据成员a和成员函数display，这样在类A和类B中就存在同名成员数据a和同名成员函数display。

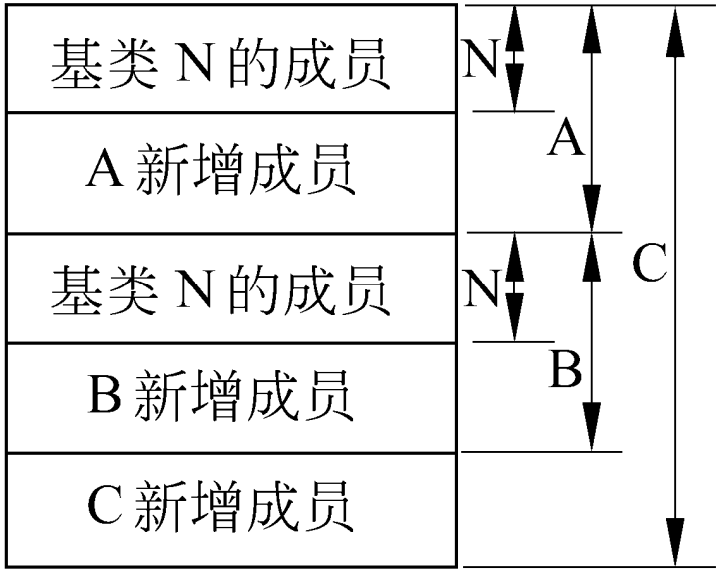
但类A和类B中的同名成员数据a分别为不同的内存单元，在程序中类A和类B的构造函数调用基类N的构造函数分别对类A和类B的成员数据a初始化。

对象c1怎样访问类A从基类N继承来的成员呢？访问的格式如下：

```
c1.A::a=3;
c1.A::display();
```

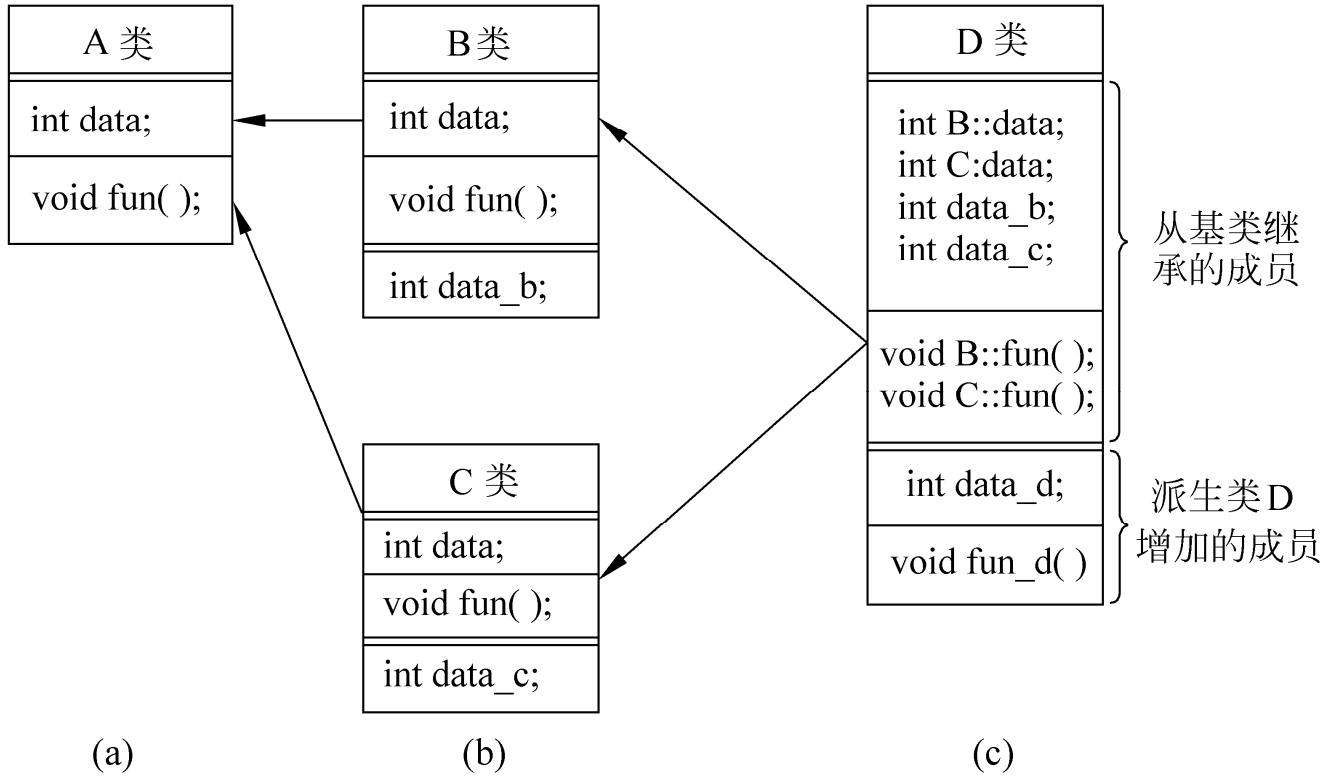
11.6.4 虚基类


C 类
int A:: a;
int A:: a1;
int B:: a;
int B:: a2;
int a;
派生类 C
的数据成员
void A::display( );
void B::display( );
void show( );
派生类 C 中
的成员函数


基类 N 的成员
A 新增成员
基类 N 的成员
B 新增成员
C 新增成员
N
A
N
B
C

11.6.4.1 虚基类的作用

如果不希望在派生类中保留间接共同基类的多个同名成员，C++ 提供了虚基类的方法，使派生类在继承间接共同基类时只保留一份成员。



虚基类主要用来解决多层继承时可能发生的对同一基类继承多次而产生的二义性问题，使用虚基类为最低层的派生类提供唯一的基类成员，而不重复产生多次拷贝。

口 虚基类的声明格式：

class 派生类名：virtual 继承方式 基类名

说明：（1）当基类通过多条派生路径被一个派生类继承时，派生类只继承该基类一次；（2）在第一级继承时就要将共同基类设计为虚基类；（3）虚基类是在声明派生类时，指定继承方式时声明的。因为一个基类可以作为一个派生类的虚基类同时也可以作为另一个派生类的非虚基类；（4）关键字virtual与关键字public或private的相对位置无关，但必须位于虚基类名之前，且virtual只对紧随其后的基类名起作用；

例如：class D:virtual public A, private B, virutal public C

其中，类A和类C是虚基类，而类B是非虚基类。

11.6.4.2 虚基类的初始化

如果在虚基类中定义了带参数的构造函数，而且没定义默认构造函数，则要求在它的所有派生类（直接和间接）中，通过构造函数的初始化列表对虚基类进行初始化。

```
class N {
    private:
        int n;
    public :
        N(int n1): num(n1) {}
};
class A : virtual public N {
    private:
        int a;
    public:
        A(int n1, int a1):N(n1), a(a1) {}
};
```

```
class B: virtual public N {
    private:
        int b;
    public:
        B(int n1, int b1):N(n1), b(b1) {}
};
class C : public A, public B {
    private:
        int c;
    public:
        C(int n1, int a1, int b1 , int c1):N(n1) ), A(a1) , B(b1), c(c1) {}
        void show( ) { cout<<“a3=“<<a3; }
};
int main() { C c_object; return 0; }
```

说明：

- **派生类中只有一份虚基类成员；**
- **虚基类构造函数必须只被调用一次，目的是要保证虚基类成员只被初始化一次；**
- **最后派生类：继承结构中建立对象时所指定的类；**
- **虚基类成员由最后派生类的构造函数通过调用虚基类的构造函数进行初始化；**
- **最后派生类的构造函数的成员初始化列表中必须给出对虚基类构造函数的调用，如果未列出，则相应的虚基类必须有缺省构造函数；**
- **多重继承中的构造函数的调用顺序：**

  （1）按继承虚基类的顺序调用虚基类的构造函数；

  （2）按继承非虚基类的顺序调用非虚基类的构造函数；

  （3）按声明成员对象的顺序调用其构造函数；

  （4）调用派生类自己的构造函数。

11.7 基类与派生类的转换

公有派生类实际上具备了基类的所有功能，凡是基类能解决的问题，公有派生类都可以解决。在需要基类对象的任何地方都可以使用公有派生类的对象来替代。

口 基类对象与公有派生类对象之间的赋值兼容（赋值兼容即不同类型间的自动转换和赋值）：

- 公有派生类对象可以向基类对象赋值，反之则不可以；
- 公有派生类对象可以替代基类对象向基类对象的引用进行赋值或初始化，这样该引用只是派生类对象中基类成员部分的别名；
- 如果函数的形参是基类对象或基类对象的引用，则实参可以用公有派生类的对象；
- 公有派生类对象的地址可以赋给指向基类对象的指针变量，也就是说，指向基类对象的指针变量也可以指向派生类对象，但通过这样的基类对象指针只能访问派生类中的基类成员。

11.8 继承与组合

在一个类中以另一个类的对象作为成员数据，称为类的组合。

- **通过继承建立了派生类与基类的关系，它们之间是一种“是”的关系，派生类是基类的具体化实现；**
- **通过组合建立了成员类与组合类的关系，它们之间是一种“有”的关系。**

```
class Teacher { … };
class Birthdate { … };
class Professor: public Teacher {
    public:
        …
    private:
        Birthdate birthday;
}
```

Professor类通过继承，从Teacher类中得到num，name，age，sex等成员数据；通过组合，从Birthdate类得到了year，month，day等成员数据。

CH11 End