第四章 派生类、基类和继承性

继承性是面向对象程序设计的一个最重要概念，理解继承是理解面向对象程序设计所有方面的关键。继承性允许在构成软件系统的层次结构中，可利用已存在的类并扩充它们，以支持新的功能，这使得编程者只需在新类中定义已存在类所没有的成分来建立新类，从而大大提高了软件的可重用性和可维护性。

C++利用派生类机制作为实现继承性的基础，它允许从任何现存的类派生出新的类，这些新类称为“派生类”或“子类”，前者则称为“派生类”的“基类”或“父类”。派生类继承了基类的全部成员(包括数据成员和成员函数)，并且还可以增加基类所没有的数据和成员函数，以满足派生类特殊的应用要求。

4.1继承的概念：什么是继承？继承是客观世界中实体间的一种关系。具有如下关键成份：

- 实体间共有的特征（共有的属性和状态）
- 实体间的区别(个性)
- 层次结构

现实世界是分类分层的客观实在，物质可分为有机和无机，有机体又分为生命体和非生命体，生命体进而可分为动物、植物和微生物等。用继承来描述事物的层次关系，帮助人们更准确地理解事物的本质，一旦搞清了事物的层次结构，也就找到了解决问题的办法。在我们周围处处有继承。例如家庭成员。每个人从自己父母身上继承了相同的基本特征。体魄、外形和举止习惯，而兄弟姐妹间又有不同之处。一个家庭成员的关系是一种层次结构。

如图4.1所示，由父类、子类和孙子类构成了一个三层的类层次结构，那末父类是子类的直接基类，子类是父类的直接派生类但又是孙子类的直接基类，孙子类是子类的直接派生类，而父类对孙子类，以及父类对曾孙子类都称为间接基类，反过来孙子类对父类及曾孙子类对父类都称为间接派生类。以后将直接基类和直接派生类分别简称为基类和派生类。

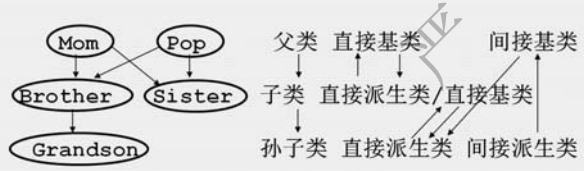


图4.1 家庭内的继承关系

在编程方面，继承类似于家庭成员间的关系。只是它包含类之间的关系。一个类从另一个类中继承了一些属性，用这种技术是为了派生出一个新的类。

用继承建立的类系统称为类层次结构，这与家庭内的继承关系完全类似。类的派生还可以无限继续下去，即派生类又可成为另一派生类的基类。

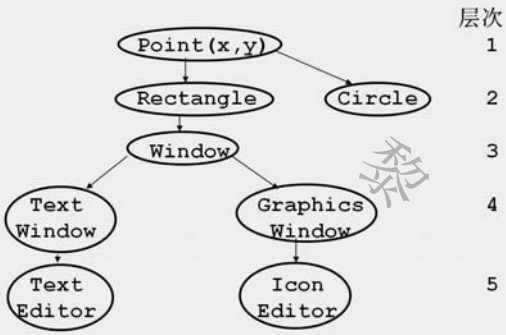


图4.2 窗口的类层次结构

如图4.2所示的类层次结构具有5层，第1层的点Point类派生出第2层的Rectangle类和Circle类，而第3层的窗口Window类是从长方形Rectangle类中派生而得，那末，以窗口类作为基类，又可派生出第4层的文本窗口TextWindow类和图形窗口GraphicsWindow类，再推下去，可派生出第5层的文本编辑TextEditor类和图符编辑IconEditor类。

抽象描述现实世界分类分层的类层次结构具有如下特点：

- 越在上层的类越具有普遍性和共性，越在下层的类越细化、专门化。
- 类层次的继承和多层类层次结构中继承的传递性。即派生类能自动地继承其基类的全部数据结构和操作方法。若类层次结构是多层的话，这种继承还具有传递性，即最下层的派生类能自动继承其上各层类的全部数据结构和操作方法。这是一种垂直的多层共享机制。

继承有两种类型，单继承和多继承。单继承：每个派生类仅只能有一个基类，正如图4.3所示，基类和派生类构成了树形的类层次结构。

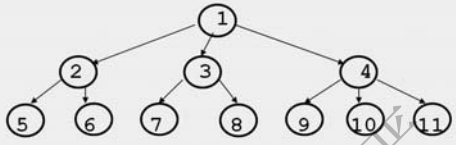


图4.3 单继承构成了树形的类层次结构

多继承：允许派生类同时具有多个基类，因此类和派生类可以构成有向图的层次结构，这在客观世界中确实也大量存在这种关系。如图4.1所示的家庭关系就是一种多继承关系，可以抽象为图4.4所示的有向图结构。

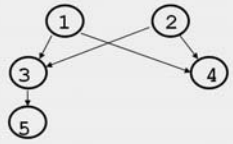


图4.4多继承构成的有向图结构

自然界中，生物通常有父母双亲，这种安排使后代变异的可能性更大，从而具有更大的适应性和生存能力。

在OOP中采用多继承关系来描述，而克隆技术是搞无性繁殖，可用单继承关系来描述。

4.2 单继承的派生类

1. 派生类的概念和定义：

从任何已存在的类，无论是C++标准类库提供的，还是用户自行定义的类，都可以派生出新类，这些新类称为派生类（Derived class），而已存在的用来派生新类的类称为基类（Base class），派生类的定义形式和普通类定义基本一致。所不同的是必须在类头部分指明它的基类，其形式如下：

```
class  派生类名 : <继承方式> 基类名
{
        <派生类新定义的成员>
}
```

其中<继承方式>采用如下三个关键字指定：

用public指定公有继承，其基类称为公有基类，该派生类称为公有派生类，

用private指定私有继承，其基类称为私有基类，该派生类称为私有派生类，

用protected指定保护继承，其基类称为保护基类，该派生类称为保护派生类，

现以例4.1为例来说明派生类的概念和定义格式：


```
#include <iostream>
using namespace std;
class Base {
  float  x, y;
public:
  Base(float x1, float y1)
  { x = x1; y = y1; }
  void Print( )
 {  cout << " x = " << x
         << "\t y = " << y << endl;  }
};

class Derived : public Base {
  float z;
```



```
public:
  Derived(float x1, float y1, float z1)
  : Base(x1, y1)  { z = z1; }
//派生类构造函数中应包含基类的初始化列表
  void Print( );/* 派生类重新定义的同名成员
  函数Derived::Print( )，覆盖了从基类继承的
  Base::Print( )，即在派生类作用域内，写
  "Print( );"语句就是调用Derived::Print( )函
  数。*/
};
void Derived::Print( )
{ Base::Print( );
  /*在派生类作用域内调用从基类继承的同名成员函数时，需用"类名::"指明*/
  cout << " z = " << z << endl ;
}
void main( )
{
  Derived d1(3.0, 4.0, 5.0);
  d1.print( );
}
```

(1) 冒号把派生类与基类名分开，并用它来建立派生类和基类之间的层次结构。

(2) 派生类在类体内也可以定义数据成员和成员函数，由于派生类继承了其基类的所有成员，即基类的成员将自动地成为派生类的成员（共性，可不再重复编写），因此在派生类体中只列出新增的数据成员和成员函数。（个性，只编写新增加的）

例如可以用类Base来表示二维坐标中的点：

```
class Base {
  float  x, y;
public:
  Base(float x1, float y1)
  { x = x1; y = y1; }
  void Print( )
      {  cout << " x = " << x
         << "\t y = " << y << endl;  }
};
```


类Base用浮点数x和y表示点的坐标，并定义了构造函数Base(float x1,float y1)

设置点坐标以及显示点坐标的成员函数Print()，那末从Base类可派生出描述三维坐标系点坐标的Derived类如下所示：

```
class Derived : public Base {
  float z;
public:
  Derived(float x1, float y1, float z1)
 : Base(x1, y1)  { z = z1; }
  void Print( );
};
```


(3) 派生类的构造函数必须提供一个对基类数据成员进行初始化的参数表，写在派生类构造函数定义的第一行，格式为：

```
构造函数名（参数表）:基类名（参数名1，参数名2，…）
{        <函数体>          }
```

派生类构造函数的参数表，必须包含从基类继承的数据成员和新增数据成员的初始化参数。如图4.5所示，对Derived类除了具有类体内自身定义的数据成员z和成员函数Derived::Print()外，还继承了Base类中的所有数据成员x和y以及成员函数Base::Print()。

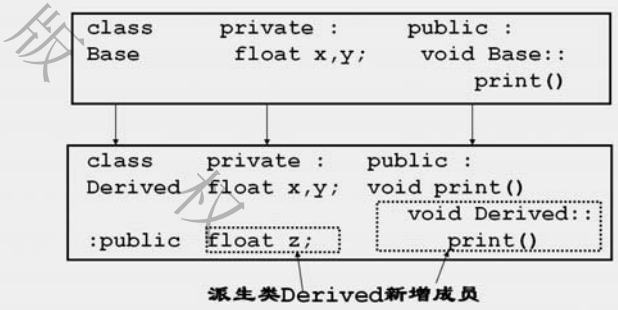


图4.5 基类Base和派生类Derived

在main( )函数内，Derived类的对象d1在生成时，调用派生类构造函数Derived()初始化所有数据成员x、y和z的值，可写为：

```
Derivde d1(3.0,4.0,5.0);
```

2.公有继承和私有继承

(1)继承方式的种类：前面讨论了派生类对基类的继承关系，在派生类的作用域内，基类成员作为派生类的成员，对于程序其它部分（例如所生成的对象）的访问权限限制是怎样的呢？派生类的对象是否可以直接访问基类的公有成员或私有成员呢？

这是由在定义派生类时，对继承方式所用的关键字是public、private还是protected来决定的。如前所述，有公有继承、私有继承和保护继承，它们继承而来的派生类分别称为公有、私有和保护派生类，如表4.1所示。

表4.1 直接基类和直接派生类的访问权限

继承类型	基类中的访问权限	基类成员在派生类中的访问权限
public 公有继承	public private protected	public 不可见 protected
private 私有继承	public private protected	private 不可见 private
protected 保护继承	public private protected	protected 不可见 protected

由表4.1可知，使用public关键字的继承称为公有继承，基类成员的访问权限在(直接)派生类中保持不变，换句话说，基类中的public成员在直接派生类中仍是public成员，protected成员仍是protected，而基类的私有成员则变为不可见。

使用private关键字的继承称为私有继承，对于私有继承，基类成员的访问权限在(直接)派生类中都变成私有的，换句话说，基类中的public成员和protected成员都是(直接)派生类的private成员。

使用protected关键字的继承称为保护继承，基类的所有公有成员和保护成员都作为(直接)派生类的保护成员，基类的私有成员仍然是私有的。由于保护继承用得很少，所以只讨论公有继承和私有继承。

(2)基类的私有成员在(直接)派生类的作用域内不可见：不管是哪种继承方式，派生类虽然继承了基类的所有成员，但派生类的成员函数却不能直接访问它们，例如派生类Derived中的Print()成员函数不能直接访问Base类中的私有数据成员x和y。

```
void Derived::Print( )
{   cout << " x = " << x
         << "\t y = " << y << endl;
       //error, 不能直接访问Base类中
       //的私有数据成员x和y。
    cout << " z = " << z << endl ;
}
```

因此派生类的成员函数也只有通过基类公有部分提供的成员函数访问基类的私有成员，所以正确处理Derived类中的Print()函数应是调用基类Base中的公有成员函数Base::Print()来访问基类的私有数据成员x和y.其函数体的定义如下：

```
void Derived::Print( )
{
  Base::Print( );
  cout << " z = " << z << endl ;
}
```

这里用Base::来标识Print()是必须的。它告诉编译系统这里是调用基类Base的Print()。否则将引起Derivded::Print()的无约束条件限制的递归调用，形成无穷循环导致死机。

(3)保护部分（protected）

如前所述，一个派生类继承了基类的全部成员，但基类的所有私有成员不能被派生类直接访问，只有调用基类中的公有成员函数才能直接访问。上例中的派生类Derived的成员函数Print()不能访问基类Base中的私有数据成员float x, float y,

如果我们希望派生类Print()能够访问它们，可将x，y定义在保护部分。

既可以达到基类的隐藏和保护，又可以被派生类直接访问。是公有和私有之间的一种中间状态。


```
#include <iostream>
using namespace std;
class Base {
protected:
  float  x, y;
public:
  Base(float x1, float y1)
  { x = x1; y = y1; }
  void Print( )
  { cout << " x = " << x
         << "\t y = " << y << endl;
  }
};
class Derived : public Base {
  float z;
public:
  Derived(float x1, float y1, float z1)
        : Base(x1, y1)  { z = z1; }
    void Print( );
};
```



```
void Derived::Print( )
{
  cout << " x = " << x   << "\t y = "
    << y << "\t z = " << z << endl;
}
void main( )
{
      Derived d1(3.0, 4.0, 5.0);
      d1.Print( );
}
```

因此基类保护部分的成员（数据成员和成员函数）可以被派生类的成员函数直接访问，但不能被其它类的成员函数直接访问。

(4) 基类对象和派生类对象：不同的继承方式基类对象和派生类对象对它们各种成员的访问限制是有区别的。①对于公有继承方式，如图4.6所示。

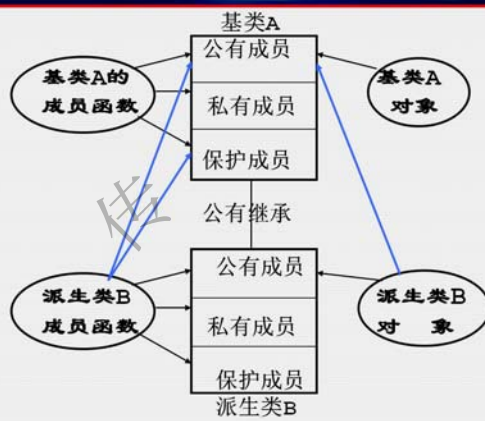


图4.6 公有继承方式的访问限制

- 基类对象只能访问基类的公有成员，而不能访问基类的私有成员和保护成员

```
#include <iostream>
using namespace std;
class A {
    int a, b;
public :
    A(int i, int j) { a = i; b = j; }
void Move(int x, int y)
{ a += x; b += y; }
void Show( )
{ cout << " A类的对象 : (" << a
   << " , " << b << ")" << endl; }
};
```



```
class B : public A {
    int x, y;
public :
    B(int i, int j, int k, int l) :
          A(i, j), x(k), y(l) { }
    void Show( )
    { cout << " B类的对象 : (" << x
      << " , " << y << ")" << endl; }
    void fun( ) { Move(3, 5); }
    /*调用基类公有成员函数Move( )，因派生类中没有
    同名的成员函数Move( )，则可缺省A::*/
};
```

```
void main( )
{
  A e(1, 2);
  /* 基类对象e不能直接访问本类的私有数据成员a和
     b，只有通过本类的公有成员函数Show( )来访问
     它们。*/
  e.Show( );
  B d(3, 4, 5, 6);
  d.fun( );
  d.A::Show( );
  d.B::Show( );
  /* 在公有继承方式下，派生类B的对象d不仅可以访
     问本类的公有成员函数，即“d.B::Show( )”，还
     可以直接访问基类A的公有成员函数，即
     “d.A::Show( )”*/
}
```


该程序的输出结果：

A类的对象 ： (1 , 2)
A类的对象 ： (6 , 9)
B类的对象 ： (5 , 6)

•派生类对象除了能访问本身类的公有成员，而不能访问本身类的私有成员和保护成员。可以访问基类的公有成员，但不能访问基类的私有成员和保护成员。

②对私有继承方式。

•与公有继承方式一样，基类对象只能访问基类的公有成员，而不能访问基类的私有成员和保护成员。

•派生类对象只能访问本身类的公有成员，不能访问本身类的私有成员和保护成员，且与公有继承方式不同，它再不能访问基类的公有成员了因为它们已经变成私有的，同样也不能访问基类的私有成员和保护成员。

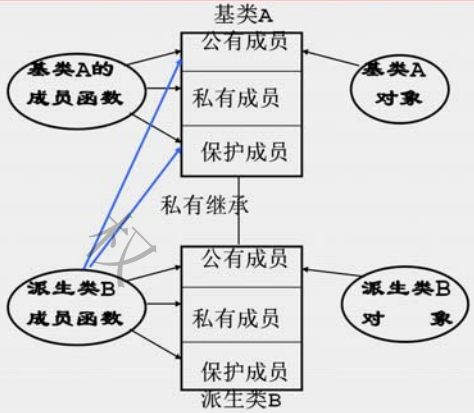


图4.7 私有继承方式的访问限制

```
#include <iostream>
using namespace std;
class A {
    int a, b;
public :
    A(int i, int j) { a = i; b = j; }
    void Move(int x, int y)
    { a += x; b += y; }
    void Show( )
    { cout << " A类的对象 : (" << a
      << " , " << b << ")" << endl; }
};
```



```
class B : private A {
    int x, y;
public :
B(int i, int j, int k, int l)
      : A(i, j), x(k), y(l) { }
    void Show( )
    { cout << " B类的对象 : (" << x
      << " , " << y << ")" << endl; }
    void fun( ) { Move(3, 5); }
    void f1( ) { A::Show( ); }
};
```



```
void main( )
{
  A e(1, 2);
  /* 在派生类B的公有部分内，编写公有成员函数fun( )
     和f1( )，去访问基类的私有数据成员a和 b。*/
  e.Show( );
  B d(3, 4, 5, 6);
  d.fun( );
  d.Show( );
  d.f1( );
}
```


该程序的输出结果：

```
A类的对象 : (1 , 2)
B类的对象 : (5 , 6)
A类的对象 : (6 , 9)
```

(5) 基类和派生类的成员函数：

①不管是公有、还是私有继承方式，基类和派生类的成员函数都可以直接访问各自类的公有、私有和保护成员（同作用域）。

②不管是公有、还是私有继承方式，派生类的成员函数可以直接访问基类的公有和保护成员，而不能直接访问基类的私有成员。

③综合应用上述两个性质，可以在派生类的公有部分内，设计成员函数去访问基类的私有成员，从而打通了派生类对象访问基类私有成员的消息通路。

④对私有继承方式，由于基类的公有成员在派生类中都变成私有的，特别是成员函数变为私有后，切断了派生类对象访问基类私有成员的消息通路。为此C++还提供了一种“访问声明”的调整机制，其作用是在派生类作用域内以恢复它们原来在基类中的访问权限，即原来在基类中是公有的，在派生类中重新恢复成公有的，其格式为：

```
class 派生类名 : private 基类名 {
                  。。。
public :
    基类名::基类公有数据成员名;
    基类名::基类公有成员函数名;
            。。。
};
```

由于这是一种“访问声明”的调整机制，不是重载成员函数，所以成员函数名后面不能再写圆括号，只能对直接派生类实现调整，而间接派生类采用接力式调整，即逐级在每个直接派生类中进行调整声明。


```
#include <iostream>
using namespace std;
class A {
public :
      void f(int i) { cout << i << endl; }
      void g( ) { cout << "Call g().\n"; }
};
class B : A {
public :
      void h( ) { cout << "Call h().\n"; }
      A::f;
      A::g;
};
void main( )
{     B d1;
      d1.f(6);
      d1.g( );
      d1.h( );
}
```

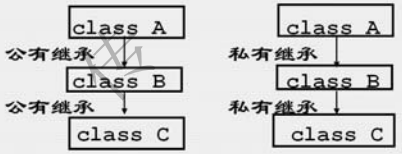

(6) C++中struct是一种特殊的类，可以有成员函数(其中包含构造函数和析构函数)，还可以有派生类。在派生类定义时，类头部分冒号后的访问限制符public和private可以缺省，如果派生类是class类型，private可缺省，如果派生类是struct类型，缺省时为public。例如，可写出两个strucrt分别表示点和圆，而表示圆的Circle类是从表示点的Point类公有继承而来，两个类体内的成员都是公有的。

```
struct Point {
int  x , y;
void setlocn(int x1, int y1)
{ x = x1;    y = y1;  }
};
struct Circle : Point {
int radius;
void setsize(int r){radius = r;}
};
```

对于那些不需要隐藏数据的场合，使用结构体是非常方便的，因为它具有与class一样的管理机制，但将所有成员都指定为公有的，便于对象进行访问。

3.派生类对基类的继承具有传递性。例如：

```
class A {…};
class B : public A {…};
class C : public B {…};
```



(a) 公有继承　　(b) 私有继承

图4.8　两种继承方式的传递

如图4.8所示，A类派生出B类，B类又派生出C类，如前所述，A类是B类的直接基类，B类是C类的直接基类，而A类称为C类的间接基类，但底层的派生类C，将继承所有上层基类A和B的成员。对于公有和私有继承的多层继承传递关系如表4.2所示。前面已讨论了直接基类和派生类间访问权限的变化情况，现从表4.2可知间接基类和派生类间访问权限的变化情况。

表4.2　公有继承和私有继承的传递关系

继承方式	class A	class B: :public A的直接派生类	class C :public A的间接派生类
公有继承	private protected public	不可见 protected public	不可见 protected public
私有继承	class A private protected public	class B :private 不可见 private private	class C :private 不可见 不可见 不可见

(1)对公有继承方式，基类A和间接派生类C间的变化情况与基类A和直接派生类B间的变化完全一样，即基类A的私有成员在派生类C中不可见，而公有、保护成员没有变化。

```
#include <iostream>
using namespace std;
class A {
   int  value;
public :
   A(int v) { value = v; }
   int ReadValue( ) { return value; }
};
class B : public A {
   int  total;
public :
   B(int v, int t) : A(v)
   { total = t; }
   int ReadTotal( ) { return total; }
};
```



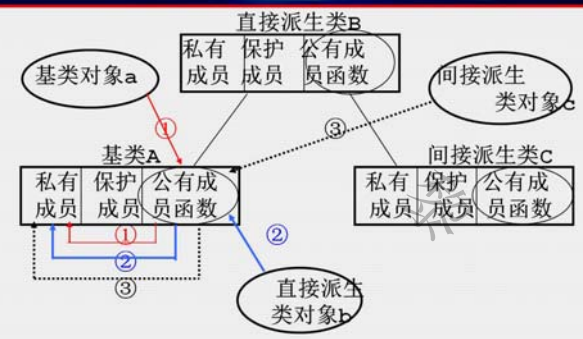
```
class C : public B
{
      int  count;
public :
      C(int v, int t, int c) : B(v , t)
      { count = c; }

       int ReadCount( ) { return count; }
};
```



```
void main( )
{
      A a(2);  B b(4, 6);  C c(8, 10, 12);
      cout << " value of a = "
           << a.ReadValue( ) << endl;①
      cout << " value of b = "
           << b.ReadValue( ) << endl;②
      cout << " total of b = "
            << b.ReadTotal( ) << endl;
      cout << " value of c = "
           << c.ReadValue( ) << endl;③
      cout << " total of c = "
            << c.ReadTotal( ) << endl;
      cout << " count of c = "
            << c.ReadCount( ) << endl;
}
```


该程序的输出结果：

```
value of a = 2
value of b = 4
total of b = 6
value of c = 8
total of c = 10
count of c = 12
```



①基类对象a访问本类私有成员的消息通路。
②直接派生类对象b访问基类私有成员的消息通路。
③间接派生类对象c访问基类私有成员的消息通路

图4.9　公有继承方式派生类对象访问基类成员

①基类对象a通过调用基类A中的公有成员函数ReadValue( )，去访问它的私有数据成员value。这是公有继承方式下，通过对象访问基类A中的私有数据成员value的第1条消息通路。

②直接派生类对象b可直接调用基类A中的公有成员函数ReadValue( )，去访问基类A中的私有数据成员value，这是公有继承方式下，通过对象访问基类A中的私有数据成员value的第2条消息通路。

③间接派生类对象c可直接调用基类A中的公有成员函数ReadValue( )，去访问基类A中的私有数据成员value，这是公有继承方式下，通过对象访问基类A中的私有数据成员value的第3条消息通路。

也可根据前述原则，找出对象b、c访问各自的total和count的消息通路。

(2)对私有继承方式，由表4.2可知，基类A的公有、保护成员在直接派生类B中变成为私有的，其私有成员则不可见，继而在间接派生类C中基类A的所有成员都成为不可见，封装的很严实，因此必须设法打通派生类对象访问基类A私有成员的消息通路。

```
#include <iostream>
using namespace std;
class A {
   int  value;
public :
   A(int v) { value = v; }
   int ReadValue( ) { return value; }
};
```



```
class B : private A {
   int  total;
public :
   B(int v, int t) : A(v)
   { total = t; }
   int ReadValue( )
   { return A::ReadValue( ); }
   int ReadTotal( ) { return total; }
};
class C : private B {
   int  count;
public :
   C(int v, int t, int c) : B(v , t)
   { count = c; }
   int ReadValue( )
   { return B::ReadValue( ); }
   int ReadTotal( )
   { return B::ReadTotal( ); }
   int ReadCount( ) { return count; }
};
```

```
void main( )
{
   A a(2);  B b(4, 6);  C c(8, 10, 12);
   cout << " value of a = "
        << a.ReadValue( ) << endl;      ①
   cout << " value of b = "
        << b.ReadValue( ) << endl;      ②
   cout << " total of b = "
         << b.ReadTotal( ) << endl;
   cout << " value of c = "
        << c.ReadValue( ) << endl;      ③
   cout << " total of c = "
         << c.ReadTotal( ) << endl;
   cout << " count of c = "
         << c.ReadCount( ) << endl;
}
```

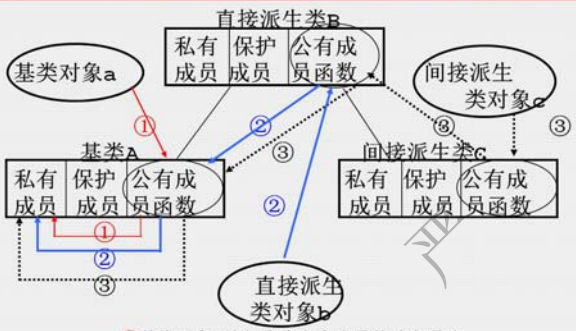

其中：private是可缺省的派生类型符，这与类封装中的成员访问限制符是一致的。

①基类对象a通过调用基类A中的的公有成员函数ReadValue( )，去访问它的私有数据成员value。这是私有继承方式下，通过对象访问基类A中的私有数据成员value的第1条消息通路。

②在私有继承方式下，直接派生类对象b不能直接调用基类A中的公有成员函数ReadValue( )，因它已变成私有的(见表4.2)。应在直接派生类B的公有部分内重新编写成员函数，在该成员函数体内，编写调用基类A公有成员函数Readvalue()的语句，其格式为：

基类名::成员函数名(参数表);

例中为"Return A::ReadValue();"语句，去访问基类A中的私有数据成员value，这是在私有继承方式下，通过对象访问基类A中的私有数据成员value的第2条消息通路。



①基类对象a访问本类私有成员的消息通路。
②直接派生类对象b访问基类私有成员的消息通路。
③间接派生类对象c访问基类私有成员的消息通路

图4.10 私有继承方式派生类对象访问基类成员

③在私有继承方式下，间接派生类对象c不能直接访问B类的公有成员，更不能直接访问基类A中的公有成员，应在C类的公有部分内，设计成员函数进行接力式访问，间接派生类对象c，访问基类A中的私有数据成员value的消息通路如下：(main()函数中)c.ReadValue();

- (C类体内)B::ReadValue();
- (B类体内)A::ReadValue();
- (A类体内)ReadValue( ){return value;}

才能访问到间接派生类对象c中的value值。这是在私有继承方式下，通过对象访问基类A中的私有数据成员value的第3条消息通路。

C++ 程序设计

④从例程inh.cpp可知，C++允许派生类重新定义从基类继承的成员，即派生类定义了与基类同名的成员，称为派生类的成员覆盖了基类的同名成员，如果要在派生类中使用基类的同名成员，应使用如下格式：

```
基类名::数据成员名;
基类名::成员函数名（参数表）;
```

上例中：由于是私有继承，派生类B中由基类A继承的成员函数ReadValue()。已变成私有成员函数，对象b不能访问，只有在派生类B的公有部分内再定义一个同名的成员函数B::ReadValue()，覆盖继承的私有成员函数A::ReadValue()。由于B::ReadValue()是公有的，B的对象b可以访问它。然后在B::ReadValue()的函数体内使用基类A的同名成员函数A::ReadValue()；访问A类的私有数据成员value。顺便指出，A::ReadValue()和B::ReadValue()这两个成员函数虽然有相同的函数名，因它们的作用域不同而不会发生重载。