

C++ 程序设计

3.6 友元 (friend)

由于类的私有和保护部分对外是隐藏的，从外部不能直接访问它们，只有通过公有成员函数访问。类的这种封装性和数据隐藏对提高软件的可靠性、可重用性和可维护性起到重要作用。但却增加了程序运行时的函数调用开销。因为每次通过成员函数访问类的私有数据时，对非内联成员函数都需要调用开销。如果访问非常频繁调用开销就非常大，从而导致程序运行效率极低，对内联成员函数也会增加程序代码容量。如前所述，对于一个对象的私有数据，只能通过公有部分的成员函数进行访问，这是一堵不透明的墙。若当两个不同类的对象共享同一函数时，必然带来较大的开销。出于执行效率的考虑，而并非技术上必须这么做，C++ 提供了一些辅助手段，允许外面的类或函数去直接访问一个类的私有数据，相当于在这堵不透明的墙上开了一个传递消息的孔，其方法之一是使用友元，下面用一个例子来说明。

华中科技大学人工智能与自动化学院 面向对象程序设计 黎云

C++ 程序设计

若定义了向量类 `Vector` 和矩阵类 `Matrix`，各类都隐藏了私有数据成员并提供了完整的操作函数。为简单起见，假设向量有3个元素，下标为0、1、2。而矩阵包含3行*3列=9个元素，下标都是0、1、2。若要定义一个矩阵乘向量的函数 `multiply()`，首先想到的方法是将它定义成外部函数，由 `Vector` 类和 `Matrix` 类共享它，此时在 `Vector` 类和 `Matrix` 类定义中，必须分别提供通过下标来访问各自元素的成员函数 `elem()`。

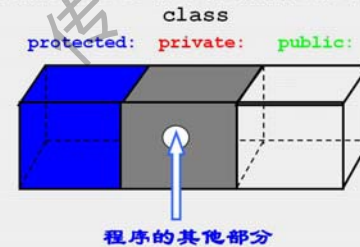


图3.8 用友元在封装墙上开了一个传递消息的孔

华中科技大学人工智能与自动化学院 面向对象程序设计 黎云

C++ 程序设计

```
class Vector{
    double v[3];
public:
    ...           读向量v的第i号元素
    double& elem(int i){return v[i];}
};
class Matrix{
    double m[3][3];
public:
    ...
    double& elem(int i, int j)
    { return m[i][j]; }
}; 读矩阵m的第i行第j列的元素
```

华中科技大学人工智能与自动化学院 面向对象程序设计 黎云

C++ 程序设计

矩阵 `m[3][3]` 乘向量 `v[3]` 的运算法则如下：

$$\begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} m_{00}v_0 + m_{01}v_1 + m_{02}v_2 \\ m_{10}v_0 + m_{11}v_1 + m_{12}v_2 \\ m_{20}v_0 + m_{21}v_1 + m_{22}v_2 \end{pmatrix}$$

其结果矩阵(向量) `r` 的第 `i` 行应为：

$$r[i] = \sum_{j=0}^2 m[i][j] * v[j]; \quad i = 0, 1, 2$$

若将 `multiply()` 定义为外部函数，则可写为：

华中科技大学人工智能与自动化学院 面向对象程序设计 黎云

C++ 程序设计

```
Vector multiply(const Matrix & m,
               const Vector & v)
{ Vector r;
  for(int i = 0; i < 3; i++)
    for(int j = 0; j < 3; j++)
      r.elem(i) += m.elem(i,j) * v.elem(j);
  return r;
}

// 调用Matrix类的elem() n² = 3²
// 调用n² = 3²(Vector类) v.elem(j);
// n² = 3²(Vector类)
```

C++ 程序设计

由于multiply()为非成员函数，不能直接访问Matrix类对象m和Vector类对象v的私有数据成员，只有通过各自的成员函数Vector::elem(int i)和Matrix::elem(int i, int j)访问。每当调用一次multiply()函数，Vector类的elem()要调用 $2n^2 + n = 3 \times (2 \times 3 + 1) = 21$ 次，式中n为向量的元素个数。Matrix类的elem()要调用 $n^2 = 3^2 = 9$ 次，从而产生很大的调用开销，直接影响了程序的运行效率，特别当n增加时，调用次数将大幅度增加。如果把multiply()定义成友元函数，可以有限制地实现对类的私有部分和保护部分的直接访问，不必再定义各自的成员函数elem()。

C++ 程序设计

```
#include <iostream>
using namespace std;
const int ROWNUM = 3;
const int VOLUMENUM = 3;
class Matrix;
class Vector{
    double v[VOLUMENUM];
public:
    Vector();
    Vector(double x, double y, double z);
    void InputV();
    void DispV();
    friend Vector Multiply
        (const Matrix&, const Vector&);
};
```

C++ 程序设计

```
class Matrix{
    double m[ROWNUM][VOLUMENUM];
public:
    Matrix();
    void InputM();
    void DispM();
    friend Vector Multiply
        (const Matrix&, const Vector&);
};

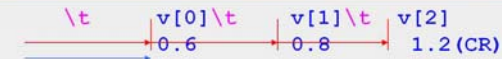
Vector::Vector()
{ for(int i = 0; i < VOLUMENUM; i++)
    v[i] = 0;
}

Vector::Vector(double x, double y,
               double z)
{ v[0] = x; v[1] = y; v[2] = z; }
```

C++ 程序设计

```
void Vector::DispV( ) //显示格式:
{
    cout << " V(";
    for(int i = 0; i < VOLUMENUM; i++) {
        cout << v[i];
        if(i != VOLUMENUM - 1)
            cout << " , ";
        else
            cout << ")\n\n";
    }
} // Vector类的键盘输入格式:
```

C++ 程序设计

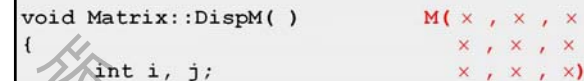


```
void Vector::InputV( )
{ cout << "\n每输入完一个元素
  按一次'TAB'键移动光标, ";
  cout << "\n再输入下一个元素,
  全部输入完后按'CR'键 !\n\n";
  for(int i = 0; i < VOLUMENUM; i++) {
      cout << "\tv[" << i << " ]";
      if(i == VOLUMENUM - 1)
          cout << "\n\t";
      else
          cout << " ";
  }
  for(i = 0; i < VOLUMENUM; i++)
      cin >> v[i];
}
```

C++ 程序设计

```
Matrix::Matrix( )
{ int i, j;
  for(i = 0; i < ROWNUM; i++)
      for(j = 0; j < VOLUMENUM; j++)
          m[i][j] = 0;
} //显示格式:
```

C++ 程序设计



```
void Matrix::DispM( )
{
    int i, j;
    cout << " M(";
    for(i = 0; i < ROWNUM; i++)
        for(j = 0; j < VOLUMENUM; j++) {
            cout << m[i][j];
            if(j < VOLUMENUM - 1)
                cout << " , ";
            else if(j == VOLUMENUM - 1 &&
                    i == ROWNUM - 1)
                cout << ")\n";
            else cout << "\n ";
        }
    cout << "\n\n";
} // Matrix类的键盘输入格式:
```

C++ 程序设计

```

\t      mv[0]\t mv[1]\t mv[2]
row { mr[0] 1.6      8.8      3.6 (CR)
    mr[1]
    mr[2]

```

C++ 程序设计

```

void Matrix::InputM( )
{ cout << "\n每输入完一个元素按一次'TAB'键移动光标, ";
  cout << "\n再输入下一个元素,
    一行输入完后按'CR'键 !\n\n";
  for(int i = 0; i < VOLUMENUM; i++) {
    cout << "\tmv[" << i << "]\t";
    if(i == VOLUMENUM - 1)
      cout << "\n";
    else
      cout << " ";
  }
  for(i = 0; i < ROWNUM; i++) {
    cout << "mr[" << i << "]\t";
    for(int j = 0; j < VOLUMENUM; j++)
      cin >> m[i][j];
  }
  cout << "\n";
}

```

C++ 程序设计

```

Vector Multiply(const Matrix & mt,
               const Vector & vc)
{
  Vector r;
  for(int i = 0; i < ROWNUM; i++)
    for(int j = 0; j < VOLUMENUM; j++)
      r.v[i] += mt.m[i][j] * vc.v[j];
  //直接访问Vector类和Matrix类的私有数据成员
  return r;
  //输出显示结果
}

```

C++ 程序设计

```

void main( )
{
  Vector vt, re;

  Matrix ma;

  vt.InputV( );
  ma.InputM( );
  ma.DispM( );
  cout << "
  vt.DispV( );
  re = Multiply(ma, vt);
  cout << "
  re.DispV( );
}

```


C++ 程序设计

1. 在类体内**声明**一个**普通函数**，在原型前加上关键字**friend**就成了该类的友元，它虽然**不是成员函数**，但可以访问该类的**所有成员**。友元函数的定义则在类体外，因为它不是成员函数，不需要用“类名::”指定它属于哪个类。其作用是提高了程序运行效率，相当于在类的封装墙上开了一个传递消息的孔。如例程中，将**multiply()**函数声明为**Matrix**和**Vector**两个类的友元，就使得**multiply()**函数既能访问**Matrix**的私有数据成员，又可访问**Vector**类的私有数据成员，达到了“两个不同类的对象共享同一函数”的目的。

2. **友元函数没有this指针**，当它访问类的私有成员时，必须传递给它一个指向该类对象的指针或该类对象的引用，作为友元函数的一个参数替代**this**指针，在友元函数体内通过对象指针或对象引用才能访问私有成员。

17

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

```
#include <iostream>
using namespace std;
class Point {
    int x, y;
public:
    Point(int xi, int yi)
    { x = xi; y = yi; }

    friend int Compare(Point & a,
                       Point & b);
};
```

18

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

替代**this**指针

通过对象访问数据成员

```
int Compare(Point & a, Point & b)
{ return a.x * a.x + a.y * a.y
        - b.x * b.x - b.y * b.y; }
```

```
void main( )
{ Point p(14, 17), q(25, 66);
  //可直接调用友元函数，不能通过对象。
  if(Compare(p, q) > 0)
    cout << "q is closer to origin\n";
    else if(Compare(p, q) <= 0)
      cout << "p is closer to origin\n";
}
```

19

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

3. 友元函数可放在类的公有部分、保护或私有部分，但不管它放在哪个部分，它始终是开放的，像普通函数一样不用通过对象，直接调用。若将**compare()**函数不定义成友元，而定义为成员函数可写为：

```
#include <iostream>
using namespace std;
class Point {
    int x, y;
public:
    Point(int xi, int yi)
    { x = xi; y = yi; }
    int Compare(Point & b); //成员函数
};
```

20

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

```
//只需一个参数, 另一参数由this指针传递
int Point::Compare(Point & b)
{ return x * x + y * y
  b.x * b.x - b.y * b.y; }

void main( )
{ Point p(14, 17), q(25, 66);
  //必须通过对象p访问成员函数compare()
  if (p.Compare(q) > 0 )
    cout << "q is closer to origin\n";
  else if (p.Compare(q) < 0)
    cout << "p is closer to origin\n";
}
```

//此时两个比较的只能是相同类对象了(同作用域),如果是前面的数组和矩阵,则形参无法直接访问私有数据成员,必须在非成员函数的类里面声明为友元函数。

C++ 程序设计

4. 可将一个类的成员函数说明为另一个类的友元, 该成员函数就可访问另一个类的所有成员, 甚至可将整个类说明为另一个类的友元, 简称为“友类”, 该类的每个成员函数都可访问另一个类中的所有成员。

```
#include <iostream>
using namespace std;
class X {
  friend class Y;
public:
  void Set(int i) { x = i; }
  void Display( )
  { cout << "x = " << x << ", ";
    cout << "y = " << y << endl; }
private:
  int x;
  static int y;
};
```

C++ 程序设计

```
class Y {
public:
  Y(int i, int j);
  void Display( );
private:
  X a; //对象成员
}; //友类Y的成员函数可访问X类的私有成员。
int X::y = 1;
Y::Y(int i, int j)
{ a.x = i; X::y = j; }
void Y::Display( )
{ cout << "x = " << a.x << ", ";
  cout << "y = " << X::y << endl;
} //友类Y的成员函数可直接访问X类的私有静态成员。
```

C++ 程序设计

```
void main( )
{ X b;
  b.Set(5);
  cout << "(1)";
  b.Display( );
  Y c(6,9);
  cout << "(2)";
  c.Display( );
  cout << "(3)";
  b.Display( );
} 该程序的输出结果:
```

```
(1) x = 5, y = 1
(2) x = 6, y = 9
(3) x = 5, y = 9
```

C++ 程序设计

说明：

(1) 该程序把Y类说明为X类的“友类”，所以在Y类的成员函数中才能直接访问X类的私有数据成员a.x。a是X类的对象，定义为Y类的私有对象成员。

(2) 在X类中又定义了一个静态数据成员y，必须对它进行初始化。从例程中可知，在Y类的成员函数中也能直接访问X类的私有静态数据成员y。因此可以直接访问X类的一切成员。

(3) 由于X类的y是静态数据成员，通过Y类的对象c把它的值从初值1变成9后，在X类的对象b中，y成员的值仍是9，由此可见Y类对象和X类对象共享静态数据成员y。

(4) 点一下嵌套类中的友元关系

25

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

§ 3.7 标识符的作用域和可见性：

程序中“标识符”包括函数名、常量名、变量名、类名、对象名、成员名、语句标号名等。它们由编程者启用，是具有任意长度的字符数字序列，第一个字符必须是字母和下划线。标识符的作用域是能使用该标识符的程序部分（程序段）。它与标识符的可见性密切相关，ANSI C++对此有如下规定：

1. 标识符的可见性是可以对该标识符进行访问（或称存取Access）操作的为可见的，否则为不可见。

2. 标识符作用域按其范围的大小可分为程序级、文件级、函数级、类级和块（Block，即复合语句）级等五种。程序级范围最大，块级最小。

26

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

(1) 程序级作用域包含组成该程序的所有源文件，如外部（外部连接）函数和外部变量属于程序级作用域，它们在某个源文件中定义，经声明后在该程序的所有其他源文件中都是可见的，所以它是全局的，包含一个源程序中的所有其他作用域（文件级、函数级、类级和块级等）。

(2) 在所有块、函数和类以外定义的标识符具有文件级作用域。其作用域从定义点开始到该源文件结束为止，例如静态（内部连接）函数和外部静态变量，以及用#define语句定义的宏指令和符号常量等。在头文件中定义的标识符，其作用域可以扩展到包括头文件的任意源文件。所以它也是全局的，包含一个源文件中的所有其他作用域（函数级、类级和块级等）。

27

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

(3) 类的作用域为类体，即类体的一对大括号所包围的程序部分，还包含该类的所有成员函数定义范围。在该范围内，一个类的所有成员函数都能访问同一类的任一其他成员。

① 类的成员（数据成员和成员函数）为类作用域，而类名则是文件作用域。

② 友元函数名不是类作用域，而是文件作用域，它可象普通函数那样在整个源文件中使用。

28

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

(4) 块级作用域：在一个复合语句内说明的标识符具有块级作用域，其作用域从说明点开始到复合语句的右大括号结束。如自动变量、内部静态变量、寄存器变量和函数的形参等。

①如果块内还有一个嵌套块，那末外部块中的标识符作用域包括内部块。例如：

```
for(int i = 0; i < 6; i++){
    int x;    内部块
    if(i) { x = 4; }
}            外部块
```

②由于函数体在句法上看成一个复合语句，所以函数中的绝大多数标识符是块级作用域，其中包括函数的形式参数。

29

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

(5) 函数级作用域：在函数体内说明的语句标号是函数级作用域，语句标号是唯一具有函数级作用域的标识符。

①只要语句标号在该函数体内作了说明，那末语句标号就可以在函数体内的任何地方使用，不必先说明后使用。例如：...

```
for(int i = 0; i < 3; i++)
    for(int j = 0; j < 3; j++)
        if(num[i][j] < 0) goto found;
...
found:    ... ;
```

30

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

②语句标号在一个函数体内必须是唯一的，不管它放在哪个嵌套块中。例如：

```
void func(int a)
{
    x : cout << "At first label.\n";
    {
        //A nested block
        x : //error:Duplicate label!
        cout<<"At second label.\n";
    }
    if(--a > 0) goto x; //which x:?
}
```

③在不同的函数体内可使用相同的语句标号。在函数间不能使用goto语句转移。

31

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

3. 在C++中，通常标识符的可见性和存在期是基本一致的。

只有在嵌套块的结构中，内层块的标识符覆盖了所有外层块的同名标识符，即在内层块的作用域范围内外层块的同名标识符不可见，但它们仍然存在，从而导致标识符的可见性和存在期不一致。若将main()编为Block #1，依此类推有Block #2、Block #3。先在Block #1内定义了三个int型变量a, b, c，又在Block #2中定义了同名变量b和c，那末在Block #2的作用域内int型变量a仍然可见，但是在Block #1内原来定义的int型变量b和c却被隐藏起来变为不可见的，即分别被Block #2内定义的同名int型变量b和float型变量c所覆盖。进而在Block #3内又定义了同名int型变量c，Block #2内定义的同名float型变量c又被隐藏起来，且它覆盖了所有外层的同名变量。当退出Block #3时，其内的变量c消失了，Block #2内定义的同名float

32

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

```
#include <iostream>
using namespace std;
void main()
{
    int a(5), b(7), c(10);
    cout<<"Begin In Block #1 a = "<<a
    <<"", b = "<<b<<"", c = "<<c<<endl;

    {
        int b(8);
        float c(8.8);
        cout<<"Begin In Block #2 a = "<<a
        <<"", b = "<<b<<"", c = "<<c<< endl;
        a = b;
        {
            int c;
            c = b;
            cout<<"In Block #3 a = "<<a
            <<"", b = "<<b<<"", c = "<<c<< endl;
        }
        cout<<"End In Block #2 a = "<<a
        <<"", b = "<<b<<"", c = "<<c<<endl;
    }
    cout<<"End In Block #1 a = "<<a
    <<"", b = "<<b<<"", c = "<<c<<endl;
}
```

37

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

该程序的输出结果:

```
Begin In Block #1 a = 5, b = 7, c = 10
Begin In Block #2 a = 5, b = 8, c = 8.8
In Block #3 a = 8, b = 8, c = 8
End In Block #2 a = 8, b = 8, c = 8.8
End In Block #1 a = 8, b = 7, c = 10
```

//2010测试通过

变量c恢复为可见的,且其值仍然为8.8,依此类推。。。,这里只有变量a在Block #2和Block #3中没有同名变量将其覆盖,所以在这些块的作用域内都是可见的。变量a在Block #2中改变了数值(5 → 8),并保持改变后的值进入到Block #3以及后面的程序段。综上所述,如图3.6所示的例程,标明了各标识符应属于那级作用域:

38

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

```
class buffer {
public:
    char *str;
    int add(char *s, int n);
};

int buffer::add(char *s, int n)
{
    int i;
    loop: if (n > 0 && *s) {
        *str++ = *s++;
        i++; n--;
        goto loop;
    }
}

buffer b;
```

35

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

4. 头文件:

实用的程序经常采用工程文件,一个工程文件程序通常由多个源文件组成,每个源文件是一个可编译的程序单位。编程者在将程序分解成多个源文件时,必须规划好每个源文件中哪些信息在其他源文件内是可见的,哪些是不可见的。C++提供了在文件间开放和隐藏信息的方法,即指定变量或函数具有外部(extern)或静态(static)存储类型或都不具有。

(1) 同一标识符的声明可以多次,具有外部存储类型的声明如"extern int a;",可以在多个源文件中使用。最好的方法是放在头文件中,头文件起着源文件之间的桥梁作用。

(2) 规划哪些可以,哪些不可以放在头文件中的经验规则(所谓“经验规则”是对#include语句运行机制使用的一个合理建议,并非C++基本语法非要这么做不可)如下:

36

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

★一般可放入头文件的程序部分有：

- 类型定义如typedef unsigned char BYTE;
和 enum COLORS{BLACK, . . . , WHITE};

- 函数声明，如[extern] int fun(int v);

- 内联函数定义，如
inline int add(int a, int b)
{ return a + b; }

- 常量定义，如const int arraysize = 100;

37

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

- 全局变量和数组的声明，如
extern int a;和extern char s[81];

- 预处理语句，如 #include <iostream.h>
和#define max(a , b) ((a > b) ? a : b)

- 注释，如/*check for End of File*/
或//check for End of String

38

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

★不适于放在头文件内的程序部分有：

- 普通函数定义，如
int add(int a,int b){return a + b;}

- 全局变量和数组的定义，如int k;和int
a[6];

- 常量数组的定义，如
const int a[] = {1, 2, 3};

39

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

例如有一个编辑名为area.prj的工程文件，它由如下三个源文件组成：

```
#include "area.h" //circle.cpp
//定义计算圆面积的函数
const double pi = 3.1415926;
double circle(double radius)
{ return pi * radius * radius; }

#include "area.h" //rect.cpp
//定义计算矩形面积的函数
double rect(double width,
            double length)
{ return width * length; }
```

40

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云

C++ 程序设计

```
#include <iostream>
using namespace std;
#include "area.h"
void main() //调用不同功能的函数计算面积
{ double w , l;
  cout << "Please enter a width : ";
  cin >> w;
  cout << "Please enter a length : ";
  cin >> l;
  cout << "Area of the rectangle is "
        << rect(w, l) << endl;

  double r;
  cout << "Please enter a radius : ";
  cin >> r;
  cout << "Area of the circle is "
        << circle(r) << endl;
}
```

C++ 程序设计

这三个源文件中，都包含了自行定义的头文件如，

```
//area.h
double circle(double radius);
double rect(double width,
            double length);
```

头文件area.h只包含了各源文件内所定义的外部函数声明语句，使得它们可以在任何源文件中使用。凡是要调用这些外部函数的源文件，如area.cpp，只需写上#include "area.h"语句即可，达到了信息共享的目的。