第9讲 C++的多态性-运算符重载

目录 CONTENTS

1 运算符重载概述

- 多态的两种方式：
- ✓ 静态多态：函数重载、运算符重载
- ✓ 动态多态：虚函数

除了函数重载外，C++中的运算符也可以重载。

为什么需要运算符函数重载？

- 两个或者两个以上的运算符函数共用一个运算符函数名为运算符函数重载。
- C++系统已经预先实现了将两个基本数据类型的数据相加的运算符函数，可以直接使用它们。

1.求两个整数相加,例如:

```
int i=2+3;
```

2.求两个双精度数相加,例如:

```
double d=2.222+3.333 ;
```

系统默认的运算符的缺点

- C++系统预先编好的运算符函数具有局限性，不能完成将两个对象的关联操作工作，例如两个对象相加。

```
class  Complex {
    ……
   };
```

若要把类Complex的两个对象com1和com2加在一起：

```
int main()
{
  Complex com1(1.1，2.2)，com2(3.3，4.4);
  Complex  total;
  total = com1 + com2;          //错误

  return 0;
}
```

C++系统预先编好的运算符函数不能将两个对象的相加

运算符重载的必要性

- C++认为用户定义的数据类型就像基本数据类型int和char一样有效。
- 运算符（如+、-、*、/）是为基本数据类型定义的，是否允许应用于自定义的类型呢？

怎样进行运算符重载

- 运算符是在C++系统内部定义的，具有特定的语法规则，如参数说明，运算顺序，优先级别等。
- 重载运算符时，要注意该重载运算符的运算顺序和优先级别不变。

运算符重载

- 运算符是函数，除了运算顺序和优先级别不能更改外，参数和返回类型是可以重新说明的，即可以重载。

- 重载的形式是：

返回类型 operator 运算符号（参数说明）；

```
#include <iostream>
using namespace std;
class Complex{          //声明复数类Complex
  private:
     double real;              //复数实部
     double imag;              //复数虚部
  public:
    Complex(double r=0.0,double i=0.0);
    void Display();
};
 Complex::Complex(double r,double i)              //构造函数
  {  real=r;imag=i;  }
 void Complex::Display()       //显示输出复数
  {  cout<<real;
    if(imag>0) cout<<"+";
    if(imag!=0) cout<<imag<<"i\n";
  }
```

运算符函数operator+( )

将类complex的两个对象相加的运算符函数operator+( )：

```
Complex operator+(Complex& co1,Complex& co2)
{
  Complex  temp；
  temp.real=co1.real+co2.real；
  temp.imag=co1.imag+co2.imag；
  return  temp；
}
```

```
#include <iostream>
using namespace std;
class Complex{            //声明复数类Complex
   private:
      double real;              //复数实部
      double imag;              //复数虚部
   public:
     Complex(double r=0.0,double i=0.0);
     friend Complex operator+(Complex& a,Complex& b);
        //声明用友元函数重载运算符"+"
      void Display();
};
```

运算符函数 operator+()

```
int main()
 {
      Complex  A1(2.3,4.6),A2(3.6,2.8),A3;

       A1.display();        //输出复数A1
       A2.display();        //输出复数A2
       A3=A1+A2;            //复数相加
       A3.Display();        //输出复数相加结果A3
       A3=operator+(A1,A2);
       A3.Display();        //输出复数相加结果A4
       return 0;
 }
```

隐式调用运算符函数operator+()

显示调用运算符函数operator+()

2 运算符重载的方式

- 运算符的重载有两种定义方式：类的成员函数、非成员函数（友元函数、普通函数）。

2.1 友元函数运算符重载

- 由于运算符重载多是访问类的成员变量，所以友元函数相比于普通函数更具有优势 。

友元运算符重载函数的语法形式

(1)在类的内部，格式如下:

```
friend  函数类型 operator运算符(形参表)
{
    函数体
}
```

友元运算符重载函数的语法形式

(2) 友元运算符重载函数也可以在类中声明友元函数的原型，在类外定义。

```
class X{
    …
  friend 函数类型 operator运算符(形参表);
    …
};
```

```
函数类型 operator运算符(形参表)
{
    函数体
}
```

```
class Complex{
friend Complex operator+(Complex c1,Complex c2);
      //. . .
};

Complex operator+(Complex c1,Complex c2)
{
      //. . .
}
```

在类的内部声明原型

在类外定义友元运算符函数

```
class Complex{                  //声明复数类Complex
 public:
   Complex(double r=0.0,double i=0.0);
                    //声明用友元函数重载运算符“+”
   friend Complex operator+(Complex& a, Complex& b);
                    //声明用友元函数重载运算符“-”
   friend Complex operator-(Complex& a, Complex& b);
                    //声明用友元重载运算符"*"
   friend Complex operator*(Complex& a, Complex& b);
                    //声明用友元函数重载运算符“/”
   friend Complex operator/(Complex& a, Complex& b);
   void Show();
 private:
        double real;                          //复数实部
        double imag;                          //复数虚部
 };
```

友元运算符函数定义

- 若重载的是双目运算符，则参数表中有两个操作数；
- 若重载的是单目运算符，则参数表中只有一个操作数。

双目运算符重载

- 双目运算符有两个操作数，通常在运算符的左右两侧。

  例如:

  3+5, 24>12

- 当用友元函数重载双目运算符时，两个操作数都要传递给运算符重载函数。

双目友元运算符函数

- 若双目友元运算符函数operator@ 所需的两个操作数aa和bb在参数表提供。
- 一般而言，可以采用以下两种方法使用：

```
aa @ bb;                    //隐式（习惯）调用
operator@(aa，bb)；//显式调用
```

如上例中：

```
A3=A1+A2;
A3=operator+(A1,A2);
```

```
Complex operator+(Complex& a,Complex& b)
{
  Complex temp;
  temp.real=a.real+b.real;
  temp.imag=a.imag+b.imag;
  return temp;
}
```

也可以写为

```
Complex operator +(Complex& a,Complex& b)
{
  return Complex(a.real+b.real, a.imag+b.imag);
}
```

单目运算符重载

- 单目运算符只有一个操作数。

  例如:

  -a,　&b,　!c,　++p
- 用友元函数重载单目运算符时，需要一个显式的操作数。

```
#include <iostream>
 using namespace std;
 class Coord{
    private:
        int x,y;
     public:
        Coord(int a=0,int b=0)
        {    x=a;y=b;   }
        friend Coord operator-(Coord obj);
         //声明用友元函数重载单目运算符“-”
        void Show();
};
  void Coord::Show()
   {  cout<<“x="<<x<<“,y="<<y<<endl; }
```

```
Coord operator-(Coord obj) {
        obj.a=-obj.a;
        obj.b=-obj.b;
        return obj;
  }
};
int main()
  {   Coord ob1(50,60),ob2;
      ob1.Show();
      ob2=-ob1;
      ob2.Show();
      return 0;
  }
```

单目友元运算符函数

- 当用友元函数重载单目运算符时，参数表中有一个操作数aa 。
- 一般可以采用以下两种方法来调用：

```
operator@(aa);     //显式调用
@aa;               //隐式(习惯)调用
```

例如:

```
operator ++(ob)
++ob
```

使用友元重载自增运算符。

```
#include <iostream>
 using namespace std;
 class Coord{                    //声明类Coord
     private:
         int x,y;
     public:
         Coord(int i=0,int j=0);
         void Show();
         friend Coord operator++(Coord &op); //前缀方式
                 //声明用友元函数重载单目运算符"++"
         friend Coord operator++(Coord &op, int);
                 //后缀方式，采用引用参数传递操作数
};
```

```
Coord::Coord(int i,int j)
{   x=i;y=j;    }
void Coord::Show()
{  cout<<"x= "<<x<<", y= "<<y<<endl;    }
Coord operator++(Coord &op)
{
    ++op.x; ++op.y;
    return op;
}
```

定义友元运算符函数operator++()(前缀方式)

定义运算符"++"重载友元函数(后缀方式)

```
Coord operator++(Coord &op, int)
  {
    Coord  temp(op);    //先将原对象的值保存
    (op.x)++;
    (op.y)++;           //改变原对象
    return temp;        //返回原对象旧值
  }
```

```
int main()
  {
    Coord ob(11,22);
    ob.Show();
    ++ob;                  //隐式调用友元运算符函数(前缀)
    ob.Show();
    ob++;                  //隐式调用友元运算符函数(后缀)
    ob.Show();
    operator++(ob);     //显式调用友元运算符函数(前缀)
    ob.Show();
    operator++(ob,0);  //显式调用友元运算符函数(后缀)
    ob.Show();
    return 0;
  }
```

说明

运算符重载函数operator @可以返回任何类型，甚至可以是void类型，但通常返回类型与它所操作的类的类型相同，这样可以使重载运算符用在复杂的表达式中。

例如，可以将几个复数连续进行加、减、乘、除的运算。

A4 = A3 + A2 + A1

2.2 成员运算符重载函数

- 在C++中，可以把运算符重载函数定义成某个类的成员函数，称为成员运算符重载函数。

运算符重载函数的语法形式

(1) 在类的内部,定义成员运算符重载函数的格式如下:

```
函数类型 operator 运算符(形参表)
{
    函数体
}
```

运算符重载函数的语法形式

(2)成员运算符重载函数也可以在类中声明成员函数的原型,在类外定义。:

```
class X{
  …
  函数类型 operator运算符(形参表);
  …
};
```

```
函数类型 X::operator运算符(形参表)
{
   函数体
}
```

成员运算符重载函数的形参表

在成员运算符重载函数的形参表中,

- 若运算符是单目的,则参数表为空;
- 若运算符是双目的,则参数表中有一个操作数。

说明

- 因为成员运算符函数，参数中隐含的有一个this指针，所以对于双目算符，类成员运算符函数有一个参数。
- 当成员函数重载双目算符时，有一个参数没有被显式地传递给成员运算符函数,该参数是通过this指针隐含地传递给函数的。

双目运算符重载

- 例如：

```
class Complex {
    //…
  Complex operator+(Complex c);
    //...
};
```

参数表中的参数----右操作数
当前对象----左操作数,它是通过this指针隐含地传递给函数的。

- 对双目运算符而言，成员运算符函数的参数表中仅有一个参数:

```
Complex Complex::operator+(Complex c)
{
  Complex  temp;
  temp.real= this->real + c.real;
  temp.imag= this->imag + c.imag;
  return temp;
}
```

```
int main( )
{
   Complex  A1(2.3,4.6),A2(3.6,2.8),A3;
        //定义3个复数类对象
    A3=A1.operator+(A2);
    A3.display();
    A3=A1+A2;      //复数相加,隐式调用
    A3.display();    //输出复数相加的结果A3
    return 0;
}
```

左操作数

右操作数

显式调用

隐式调用

调用双目运算符的两种形式

- 采用成员函数重载双目运算符@后，一般可以用以下两种方法来调用：

  aa @ bb;　　//隐式(习惯)调用

  aa.operator@ (bb);　//显式调用，或

  bb.operator@(aa);

- 如上例中：

  A3=A1+A2;

  A3=A2.operator+(A1);

  A3=A1.operator+(A2);

单选题 2分

设置

此题未设置答案，请点击右侧设置按钮

假定要为类AB定义成员函数重载“+”号运算符，实现两个AB类对象的加法，并返回相加结果，则该成员函数的声明语句为

A AB operator+(AB &a, AB &b)

B AB operator+(AB &a)

C operator+(AB a)

D AB & operator+( )

提交

运算符重载的说明

C++语言对运算符重载制定了以下一些规则:

(1) C++中绝大部分的运算符允许重载,不能重载的运算符只有以下几个:

.	成员访问运算符
.*	成员指针访问运算符
::	作用域运算符
sizeof	长度运算符
?:	条件运算符

运算符重载的说明

(2) C++语言中只能对已有的C++运算符进行重载,不允许用户自己定义新的运算符。

(3) 在重载运算符时，运算符函数所作的操作没有要求保持C++中该运算符原有的含义，但不推荐改变。

例如，可以把加运算符重载成减操作，但这样容易造成混乱。所以保持原含义,容易被接受,也符合人们的习惯。

运算符重载的说明

(4) 重载不能改变运算符的操作对象(即操作数)的个数。

(5) 重载不能改变运算符原有的优先级。

例如，C++语言规定，先乘除，后加减，因此表达式:

x=y-a*b;

等价于 x=y-(a*b);

运算符重载的说明

(6) 重载不能改变运算符原有的结合性。

例如，在C++语言中乘、除法运算符“*”和“/”都是左结合的,

因此表达式: x=a/b*c;

等价于 x=(a/b)*c;

而不等价于 x=a/(b*c);

运算符重载的说明

(7) 运算符重载函数的参数至少应有一个是类对象(或类对象的引用)，数不能全部是C++的基本数据类型。

以下定义运算符重载函数的方法是错误的:

```
int operate+(int x，double y)
{ return x+y;}
```

这项规定的目的是,防止用户修改用于基本数据类型的数据的运算符性质。

运算符重载的说明

(8) 一般而言,用于类对象的运算符必须重载，但是赋值运算符“=”例外，不必用户进行重载。

因为C++系统已为每一个新声明的类重载了一个赋值运算符函数。只能实现两个对象的成员变量完全相同

如果类的构造函数中存在诸如new等操作时，需要重新定义赋值运算符。

单选题 2分

设置

此题未设置答案，请点击右侧设置按钮

通过运算符重载，可以

A 改变运算符的原有优先级

B 改变运算符的原有操作数的个数

C 实现自定义类的运算，如复数类的相加

D 创建新的运算符

提交

单目运算符重载

- 对单目运算符而言，成员运算符函数的参数表中没有参数，此时当前对象作为运算符的一个操作数。

自增运算符++重载

【分析】前缀方式和后缀方式的区别在于运算当中，是“先变”还是“先用”。

使用时： ++ ob　　//前缀方式

定义时： operator++( );　　//成员运算符函数

使用时： ob ++　　//后缀方式

定义时： operator++( );　　//成员运算符函数

为了区分开，在后缀方式的定义中加入一个参数。

自增运算符++重载

在C++中，编译器通过在运算符函数参数表中是否插入关键字int来区分++是前缀方式还是后缀方式。

```
++ ob                          //前缀方式
operator++();                  //成员运算符函数

ob ++                          //后缀方式
operator ++ ( int );           //成员运算符函数
```

调用时，参数int一般被传值0。

```
#include<iostream.h>
class Coord {
    int x,y;
 public:
    Coord(int i=0,int j=0);
    void Show( );
    Coord operator++( );
    Coord operator++(int);
};
Coord::Coord(int i,int j)
{ x=i; y=j; }
void Coord:Show( )
{ cout<<"x="<<x<<",y="<<y<<endl;   }
```

参数表中没有参数，当前对象作为运算符的一个操作数

```
Coord Coord::operator++( )
{
    ++x;   ++y;
    return  *this;
}
```

想一想，a=++i，之后a的值和i的值分别有什么变化？

想一想，a=i++，之后a的值和i的值分别有什么变化？

```
Coord  Coord::operator++(int)
{
    Coord  temp(*this);        //先将原对象的值保存
    x++;    y++;               //改变原对象
    return temp;               //返回原对象旧值
}
```

```
int main()
{   Coord ob(11,22);
    ob.Show();
    ++ob;                          //隐式调用前缀方式
    ob.Show();
    ob++;                          //隐式调用后缀方式
    ob.Show();
    ob.operator ++();              //显式调用前缀方式
    ob.Show();
    ob.operator ++(0);             //显式调用后缀方式
    ob.Show();
    return 0;
}
```

运行结果如下：

```
x=11 , y=22
x= 12 , y= 23
x= 13 , y=24
x= 14 , y=25
x= 15 , y=26
```

使用重载的单目运算符

- 采用成员函数重载单目运算符时，一般可以采用以下两种方法来使用：

```
aa.operator@( );    //显式调用
@aa;                //隐式(习惯)调用
```

如上例中：

```
ob.operator ++( )
++ob
```

两个操作数不都是类对象，如何解决？

例如：A2 = A1 + 10;

A2 = 10 + A1;

(1)友元函数

```
using namespace std;
class Coord{                          //声明类Coord
    private:
          int x,y;
    public:
          Coord(int a=0,int b=0);
          void Show();
          friend Coord operator+(Coord ob,int a);
          friend Coord operator+(int a, Coord ob);
};
Coord::Coord(int a,int b)
  {      x=a;y=b;   }
void Coord::Show()
  {  cout<<“x="<<x<<“,y="<<y<<"\n";}
```

```
Coord operator+(Coord ob,int a)
  {
        Coord temp;
         temp.x=ob.x+a;
         temp.y=ob.y+a;
         return temp;
  }
Coord operator+(int a,Coord ob)
  {
       Coord temp;
       temp.x=a+ob.x;
       temp.y=a+ob.y;
       return temp;
  }
```

```
int main()
  {
      Coord ob1(50,60),ob2;
      ob2=ob1+20;
      ob2.Show();
      ob2=40+ob1;
      ob2.Show();
      return 0;
  }
```

对象+整数x

整数x+对象

(2)成员函数

```
class Coord{                          //声明类coord
    private:
          int x,y;
    public:
          cCoord(int a=0,int b=0);
          void Show();
          Coord operator+(int x);
};
Coord::Coord(int a,int b)
 {      x=a;y=b;   }
void Coord::show()
{
    cout<<“x="<<x<<“,y="<<y<<"\n";
}
```

```
Coord operator+(int a)
  {
      Coord temp;
      temp.x=a+x;
      temp.y=a+y;
      return temp;
  }
int main()
  {    Coord ob1(50,60),ob2;
       ob2=ob1+20;
       ob2.Show();
       ob2=40+ob1; //错误
       return 0;
  }
```

成员，还是友元？

- 大部分既可说明为运算符成员运算符函数，又可说明为友元运算符函数。
- 主要取决于实际情况和程序员的习惯。一般而言：
  - 对于单目运算符，建议选择成员函数;
  - 对于运算符“=、()、[]、->”只能作为成员函数;
  - 对于运算符“+=、-=、/=、*=、&=、!= 、~=、%=、<<=、>>=”，建议重载为成员函数;
  - 对于其他运算符，建议重载为友元函数。

补充（1）：赋值运算符

```
class CComplex
{
        int real;
        int imag;
public:
        CComplex(void);
        CComplex(int r, int i);
        virtual ~CComplex(void);
        CComplex& operator= (const CComplex& a);
};
```

补充（1）：赋值运算符

```
CComplex& CComplex::operator= (const CComplex& a)
{
        this->real = a.imag;
        this->imag = a.real;
        return *this;
}
```

- 赋值运算符只能是成员函数，不能是友元函数。

补充（2）：<<运算符

```
class CComplex
{
        int real;
        int imag;
public:
        CComplex(void);
        CComplex(int r, int i);
        virtual ~CComplex(void);
        CComplex& operator= (const CComplex& a);
        friend ostream& operator<<(ostream& os, const CComplex& other);

};
```

补充：<<运算符

```
ostream& operator<<(ostream& os, const CComplex& other)
{
        os<<"real:"<<other.real<<", imag:"<<other.imag<<endl;
        return os;
}
```

- <<运算符只能是友元函数，不能是成员函数。

补充（3）：运算符[]重载

```
class Point
{
private:
    int x[5];
    int y[5];
public:
    Point()
    {
        for (int i=0;i<5;i++)
        {           x[i]=i;       }
       for (int i=0;i<5;i++)
        {           y[i]=10+i;   }
    }
    int& operator[](int n);
};
```

补充（3）：运算符[]重载

```
int& Point::operator[](int n)
{
    static int t=0;
    if (n<5)
    {       return x[n];
    }
    else if(n>=10 && n<15)
   {      return y[n-10];
   }
    else
    {       cout<<"下标出界";
            return t;
    }
}
```

补充（3）：运算符[]重载

```
void main()
{
	Point a;
	a[3]=10;//修改数组的值
	for (int i=0;i<20;i++)
	{
		cout<<a[i]<<endl;
	}
}
```

```
0
1
2
10
4
下标出界0
下标出界0
下标出界0
下标出界0
下标出界0
10
11
12
13
14
下标出界0
下标出界0
下标出界0
下标出界0
下标出界0
请按任意键继续. . .
```

补充（4）：类型转换运算符

- 对于基本数据类型的数据可以通过强制类型转换操作符来将数据转换成需要的类型。

```
double d = 3.14;
int i = (int)(d);
Int j = static_cast<int>(d);
```

- 对于自定义的类来说，在很多情况下也需要支持此操作来实现自定义类与基本数据类型之间的转换。
- C++提供了类型转换函数，也称为类型转换运算符重载函数。

补充（4）：类型转换运算符

- 重载类型转换运算符的格式

  operator 基本类型名(); //类对象－>基本类型

  - 通过类型转换函数可以将自定义类型对象转换为基本类型数据。
  - 类型转换函数被重载的是类型名。在函数名前不能指定函数类型，函数也没有参数。返回值类型是由函数名中指定的类型名来确定的（如果类型名为double， 则将类型数据转换为double类型返回）
  - 因为函数转换的主体是本类的对象，类型转换运算符重载函数只能作为类的成员函数。

补充（4）：类型转换运算符

```
class A
{
        int m_i;
public:
        A(int i) {
                m_i = i;
        }
        operator char() {
                return static_cast<char>(m_i);
        }
};
```

```
int main()
{
        A a(65);
        char c = a;
        cout<<c;
        system("pause");
        return 0;
}
```