# 第十二部分 运算符重载

## ① 先导：函数重载

即同名但参数相异的函数允许存在，
编译器将自动根据返回类型和参数来进行函数调用

详见第七部分 → 函数重载

## ② 运算符重载

为了便利一些自定义类型（结构体，类等）的运算操作，减少程序的复杂程度，增加可读性

⇓

引入了运算符重载机制 [类似函数重载]

语法：

```
Return-Type operator @ (arguement list)
{
    // code
}
```

Return-Type：返回类型　operator：关键字　@：该符号　(arguement list)：参数列表

注：以下符号不允许重载

`.` 成员引用符，`.*` 成员引用指针，`::` 域域，`?:` 三元选择，`sizeof` 大小，`typeid` 数据类型id

基本类型的运算采用默认运算符规则，其他情况重载，

在类内定义重载可看为成员函数

※一元运算符——以++为例

针对在不同位置++有不同效果，应当对重载时也有所区分.

前缀 (++a) ⇒ ReturnType operator ++( ) 无哑元

后缀 (a++) ⇒ ReturnType operator ++(int) 有一个哑元

调用对象

先自增再赋值

先赋值再自增

※赋值运算符的重载

实现对拷贝

内核类似调用拷贝构造函数，实现成员变量的拷贝（人为不定义的话，编译器会缺省提供）→直接拷贝

ReturnType operator = (const Sample& Sample)

一般为 Sample&

只读引用（常左值引用）

当返回类型为void时，无法进行连续赋值 (D=D1=D2)

// copy data

注：在类内包含指针类型的成员变量时，为防止对同一地址的重复清除导致的内存泄漏（析构函数），应当自行重载赋值运算符进行值的copy而非地址的copy

PS：拷贝构造函数与赋值符号重载的不同

用在对新的对象自己初始化中
会使用新内存

用在已存在的两个对象
拷贝中，不使用新内存

※ 运算符的两种重载方式 —— 以+为例.

a. 以成员函数方式重载

此时只需写一个参数 复数类 complex operator +(complex a)

此时 Z=X+Y 正确

Z=X+3 ⇒ Z=X.operator +(3) 正确

编译器自动调用构造函数
且以3为第一个参数

Z=3+X 错误 3不为对象

b. 以类的友元函数[全局函数]方式重载

此时要写两个参数

friend complex operator +(complex a, complex b)

此时 Z=X+Y 正确
Z=X+3 正确
Z=3+X 正确
⇒ 考虑加减乘的交换性质

※补充说明：下标运算符 [ ]
可以应用在类中 private 域的数组的直接调用上

```
int&  operator [ ] (int  i)
{
    return  v[i] ;
}
```

一定要是引用传递，可读可写

※补充说明：函数运算符 ( )
可以应用在函数类中，实现类似函数调用的效果

Overloading function call to realize expression:
f(x, y) = x * y + 5 → 调用对象

```
#include <iostream.h>
class F {
public:
        double operator( ) (double x, double y)
        { return x * y + 5; }
};
int main()
{
        F f;
        cout << f(5.2, 2.5) << endl;
        return 0;
}
```

f.operator( ) (5.2, 2.5);

※补充说明：输出运算符 <<
可以应用在复杂成员类的成员直接输出上

```
friend  ostream&  ( ostream&  os,  const complex&  a )
```

ostream 类型对象 cout　待输出对象
此处务必用引用类型

一定要定义为友员函数　　否则报错[因为拷贝构造函数在类内被禁用]

因为其形式 cout << a << endl;

前参数 cout 为 ostream 类型而非 complex 类型

**补充：什么是 endl**

endl 实际是定义在 iostream 库中的一个函数，

定义在 std 命名空间下 ostream 类内。

其主要作用是 ①插入一个换行符

②刷新输出缓冲区并输出

因此，在 cout << endl 中

实际上是在 ostream 类中重载了运算符 << 使之

可以接受函数指针，并以 endl 作为参数传入

**函数的禁用**

形式：返回类型　函数名(参数) = delete;　（delete：关键字）

PS：同样的，若开发者希望显式的调用默认的函数，可使用

返回类型　函数名(参数) = default　（default 关键字）

**补充：重载类型转换函数**

在 C++ 中，类型转换函数也可以进行重载

支持 基本类型 转换为 类对象 [调用构造函数，部分需要开发者实现]

类对象 转换为 基本类型 [开发者实现]

类对象 互化 [程序实现]

重载类型转换函数
operator Type( ){
//code
}
无需返回类型，已经自行指定

## Conversion Operators

```
#include <iostream>
using namespace std;
class Rational {
public:
  Rational(double x = 0, double y = 1)
  {
     Numerator = x;
     Denominator = y;
  }
  operator double() {
       return Numerator / Denominator;
  }
private:
    double Numerator, Denominator;
};
```

```
int main( )
{
    Rational r(100, 200);
    double d;

    // conversion operator
    d = r;

    cout << d << endl;
    return 0;
}
```

出处
对象r作为double

注意：在日常开发中，为了避免隐式调用类型转换函数所造成的二意性和提升可读性
一般将类型转换函数定为ToXX( )并显式调用
上例中 d=r.Todouble( ); 更好