基本概念

重载运算符都是以关键字operator和其后要定义的运算符共同组成。重载的运算符也包含返回类型、参数列表以及函数体。

1. 重载运算符的参数数量和该运算符作用的运算对象数量一样多。
    - 一元运算符，如++,--等有一个参数。
    - 二元，如+,-,*,/等有两个
2. 重载运算符（所有非static函数）第一个参数都是隐含的this指针
3. 内置类型的运算符不可以重定义，如

```
int operator+(int,int); // 错误，不能为int重定义内置的运算符
```

4. 不可以重载不存在的运算符（因为重载运算符基本上都是基于内置的运算符定义实现的），如

```
Foo &opeartor**();  // 不存在**运算符
```

5. 重载的运算符，其优先级和结合律与内置的一致。

```
x==y+z;
// 永远等价于
x==(y+z);
```

6. 不可以重载::,.*,.,?:等运算符

调用重载运算符

调用重载运算符有两种方式：

1. 重载的运算符函数为成员函数

```
Foo f1,f2;
f1+f2;  // 正确
f1.operator+(f2);    // 等价的调用
```

2. 重载的运算符函数为非成员函数

```
Foo f1,f2;
f1+f2;
operator+(f1,f2);
```

这两种方式都是完全等价的。

某些运算符不应该（不是不可以，只是不应该）被重载

因为使用重载的运算符本质上是一次函数调用，所有有些运算对象的求值顺序无法保留下来或一些其他和内置运算符习惯不一致的运算符都不建议重载。

选择作为成员或非成员

我们先看下，两者有何区别，然后看一下如何区分哪些运算符重载最好定义为成员，哪些定义为非成员。

```
// 以下三种都是合法的调用
string s="world";
string t=s+"!";       // 等价于 s.operator+("!");
string u="hi"+s;     // 若是成员，等价于"hi".operator+(s);
                     // 不是成员，等价于opreator+("hi",s);
                     // 因为+是重载的非成员函数，如果是成员函数，
                     // 那么"hi"不是一个string对象，因此无法调用+运算符
                     // 只能通过 string u=string("hi")+s;
```

"hi"是const char *类型，内置类型，根本没有成员函数。

因此，一些具有对称性的运算符可能转换任意一端的运算对象，如算术、相等性、关系和位运算符因此，它们通常应该是普通的非成员函数。

其他的大多数建议为成员。

重载输入、输出运算符

由于输入、输出运算符的第一个形参是istream或ostream的非常量的引用。而成员函数的第一个参数（隐含的）往往都是this指针。因此**一般情况下**，输入、输出必须是非成员函数。否则，调用起来，将会是类似下面的形式，

```
Foo f;
f<<cout;
f>>cin;
```

> 如果想让输入、输出的行为习惯和输入输出一样，也可以重载输入、输出运算符，将其定义成员函数，但是不是自定义类的成员函数，而且修改标准库中的代码，将其添加为ostream或istream的成员函数，这样其行为就一致了。

重载输入、输出运算符为成员函数

如果想让其成为成员函数，并且调用方式一致

```
Foo f;
cin>>f;
cout<<f;    // 等价于os.operator<<(f);
```

那么，应将其定义为ostream或istream的成员函数

重载递增和递减运算符

自定义类应同时定义递增和递减运算符的前置版本和后置版本。

1. 前置版本

```
class Foo{
public:
    Foo & operator++(); // 前置运算符
    Foo & operator--();
};
```

2. 后置版本（为了区分，接受一个不被使用的int类型的形参），该形参会被编译器提供一个值为0的实参，尽管从语法上来说，可以使用该形参，但是实际中，并不这样做。该形参的唯一作用就是区分。

```
class Foo{
public:
    Foo & operator++(int);  //后置
};
```

显示地调用后置运算符

```
Foo f;
f.operator++(0);    // 调用后置版本
f.operator++();     // 调用前置版本
```