引用

一、综述

在之前的章节中，我们已经介绍过，事实上的引用在底层的实现就是指针，但是由于已经经过了非常成熟的封装，他的使用非常漂亮。我们在这里给出一些很好的使用。尤其是在类之中的。

二、使用

1、pass by reference （to const）

```
double func(const int&a);
```

这样可以在一定程度上兼顾效率和安全性。一个引用是特殊的指针，仅仅是 unsigned int 类型，4个字节。传递引用和指针的速度一样。const限定符的目的就是我们不会修改原始数据，特殊的，我们也会去掉const实现对于原始数据的修改。比如在输出操作符的重载中：

```
ostream & operator << (ostream&os, const complex&x);
//注意参数os，是可被修改的
```

2、return by reference(to const)

```
complex& operator += (const complex& r)
```

使用条件是，我们最终的处理结果倒是放在了内存的哪里？

- 函数内开辟的内存空间：显然不行，因为函数结束之后就会被回收。这样外界就访问到了不好的地址。
- 已有的空间地址：可以。这样就少了中间的一次赋值。引用直接被传回去，然后通过地址访问的方式，对于数据进行处理。

  有一个很有意思的东西在于，传递着无需知道接收者是否是以 reference 进行传递。考虑下段代码：

```
//据说是C++STL团队写的complex类的内置实现
inline complex&
    _doapl(complex* ths,const complex&r)
{
    ths->re += r.re;
    ths->im += r.im;
    return *ths;
}
complex&
    operator += (const complex& r)
{
    return _doapl(this,r);
}
```

分析上段代码，很好的地方在于

- 它把加法的功能单独弄了出来，这样就可以在其他的地方使用 _doapl() 函数
- _doapl()函数之中，我们注意到返回值是 *ths，那么，它返回的就应该是一个确切的值。这不是引用，但这其实就是引用的好处，不想指针，需要特殊符号的改变，这个直接返回就好，
- += 的操作符重载中，首先将 this指针（指向左操作数）和r（右操作数的引用）传入，然后计算，得到一个值，返回。这个返回就是 (c1+=c2)的结果，但是是没有人接受的，本题中，是通过 this 指针实现的改变原始值。这个原因是为了避免关于出现这样的情形：`c1 += c2 += c3`