const 的用法

const修饰的函数不仅能限制函数去修改成员变量，同时也能实现函数的重载。要想调用const修饰的重载函数，需要用const对象去调用。

另外要注意的是，如果一个函数用const修饰了，但是这个函数没有实现重载，那么非const对象和const对象都能调用这个函数。

特别注意的是，const修饰的对象只能调用const修饰的函数，比如，testC.fun()是错误的，因为fun()是非const的，而testC是const的。

- const修饰数据成员

修饰数据变量，保证数据变量只能在定义的时候进行初始化，并且在程序的执行过程中不能被再次赋值。

1、修饰类中的数据成员变量,可以直接在声明的时候进行初始化（const int num = 100;）,不过最好还是在构造函数的初始化列表中进行初始化（必须用初始化列表进行初始化）。

```
class A
{
public:
          A(int val):num(val){ }
private:
          const int num;//const成员变量
};
```

2、也可以在程序中定义const变量。const int num = 100;//比如在定义全局const变量num=100;

3、const int num = 100;等价于int const num = 100;

也就是说**const int 和int const**意思相同。

- **const修饰成员函数**

1、const修饰成员函数的返回类型

```
class A
{
public:
          A(int val):num(val){ }
          const void fun(int val){
               cout << "num = " << num << endl;
               age = val;
          }
private:
          const int num;//const成员变量
          int age;
};
```

此时const的位置有下面两种等价的情况：

const voidfun ()；

void constfun ();

2、const修饰成员函数的形参变量

此时const的位置有下面两种等价的情况：

void fun(const int val);

void fun(int const val);

3、const直接修饰成员函数

const放在函数声明之后，函数实现之前。void fun(int val)const;

const此时**保证不会修改该对象的数据成员。**

下面的例子是错误的，因为fun是const成员函数，它不能修改对象的成员变量age;

```
void fun(int val)const{
     cout << "num = " << num << endl;
     age = val;//此时是错误的。
}
```

(1):const成员函数此时**能访问const和非const数据成员。但不能修改非const数据成员。**(都能访问，但是不能改变量的值)

(2):const成员函数中只能调用其它const的成员函数。**（const函数只能调用const函数）**

(3):void fun(int val)const;和void fun(int val);两个函数构成重载函数。**（加const函数，相当于重载。const对象只能调用const函数。 如果只有const函数，非const对象也能调用它）**

const对象只能调用const成员函数。

非const对象会优先调用非const成员函数，但是如果fun函数只有const函数，那么非const对象也会调用const成员函数。（但是该非const成员函数中不能修改对象成员变量）

- **const修饰类对象**

定义一个const对象a：const A a(100);

const修饰函数，是从函数的层面，不修改数据。

const修饰类对象，是从对象的层面，不修改数据，只能调用const成员函数。

const对象只能调用const成员函数。

非const对象会优先调用非const成员函数，但是如果fun函数只有const函数，那么非const对象也会调用const成员函数。（但是该非const成员函数中不能修改对象成员变量）

一、用const修饰函数参数

1、修饰指针，可以防止指针被修改

```
void test(const ClassA* a)
{
    ClassA** b = &a; //编译错误,不能对const指针取地址
    ClassA* c = a; //编译错误，不能将const指针转普通指针
    (*b) = new ClassA();
}
void test2(ClassA* a)
{
    ClassA** b = &a;
    (*b) = new ClassA();
}1234567891011
```

2、修饰普通类型，说明这个参数不应该被修改

```
void test(const int a)
{
        a++; //编译错误
        int* c= &a; //编译错误，不能取地址，否则就具备了改a的能力
        int b = a;  //不用强转也可以编译通过，但还是没能力改a的值
}123456
```

3、修饰引用类型，参数的值不能被修改， 也就失去了引用类型的效果，但传递对象时，可以起到不copy对象的目的。

```
void test(const int& a)
{
    a = 2; //编译错误，不能修改const引用类型的值
}
void test(const ClassA& a) //传递的时候，不需要copy一个新的ClassA,又能保护a
{

}12345678
```

二、用const修饰局部或全局变量，功能类似函数参数

三、用const修饰函数返回值，说明函数的返回类型是const的，功能类似于函数参数

```
const int test()
{
    int a = 1;
    return a;
}12345
```

四、用const修饰函数，说明函数不会修改成员变量的值

```
class ClassA
{
public:
    int b;
    int test() const
    {
        b = 3; //编译错误，const修饰的函数不能修改类成员变量的值
        return b;
    }
}
```

为什么要用const?

将参数声明为常量数据的引用原因有3条

1.使用const 可以避免无意中修改数据的错误编程。

2.使用const 使函数能够处理const 和非const 实参，否则将只能接受非const 数据。

3.使用const 引用使函数能够正确的生成并使用临时变量。

引用和继承派生之间的关系：

基类引用可以指向派生类对象，而无需进行强制类型转换。

调用函数的时候，可以将基类对象作为参数，也可以将派生类对象作为参数。