我们首先来看一个类的定义，这是关于商店书的销售情况的简单类：

class Sales\_item{

public:

double avg\_price() const; //求售出书的平均价格

bool same\_isbn(const Sales\_item &rhs) const //判断两本书的isbn号是否相等

{return isbn==rhs.isbn;}

private:

std::string isbn;

unsigned units\_sold; //记录售出书的数量

double revenue; //记录售出所有书的价格

};

double Sales\_item::avg\_price() const

{

if(unit\_sold)

return revenue/units\_sold;

else

return 0;

}

每个成员函数（static成员函数除外）都有一个额外的，隐含的形参this。在调用成员函数时，形参this都将初始化为调用该成员函数的类的对象的地址。成员函数不能定义this形参，而是由编译器隐含地定义。考虑下面的语句：

total.same\_isbn(trans); //total 和trans是Sales\_item类的两个对象

隐含的形参this的值就是Sales\_item类对象total的地址。

现在我们来看看Sales\_item成员函数声明的形参表后面的const的作用： const改变了隐含的this形参的类型。在调用total.same\_isbn(trans)时，隐含的形参this将是一个指向total对象的const Sales\_item\*类型的指针，也就是说，this指针指向的对象total是const类型。就像如下编写same\_isbn的函数体一样：

bool Sales\_item::same\_isbn(const Sales\_item \*const this, const Sales\_item &rhs) const

{return (this->isbn==rhs.isbn);}

用这种方式使用const的函数称为常量成员函数。由于this是指向const对象的指针，const成员函数不能修改调用该函数的类的对象。total.same\_isbn(trans);对这条语句来说，也就是same\_isbn函数不能修改调用该函数的类对象total。

对于成员函数，函数声明必须与其定义一致。如果函数被声明为const成员函数，那么函数定义时形参表后面也必须有const.如本例中double Sales\_item::avg\_price() const 函数的声明。

总结：在普通的非const成员函数中，this的类型是一个指向类类型的const 指针。可以改变this所指向的值，但不能改变this所保存的地址。在const成员函数中，this的类型是一个指向const类类型对象的const指针，既不能改变this所指向的对象，也不能改变this所保存的地址。