this指针是一个特殊 的类内指针，它始终指向调用它的那个对象

因此，不难看出，this其实是一个常量指针。

1.它不需要定义，直接就可以使用。

2.this指针指向被调用的成员函数所属的对象

主要作用：解决名称冲突

eg:

```
class Person
{
  public:
    Person(int age)
    {
      age=age;
    }
    int age;
}
```

在这个例子中，三个age会被编译器认为是同一个变量。

因此，我们使用this指针来解决名称冲突的问题：

```
class Person
{
public:
    Person(int age)
    {
        //1、当形参和成员变量同名时，可用this指针来区分
        this->age = age;
    }
    Person& PersonAddPerson(Person &p)
    {
        this->age += p.age;
        //返回对象本身
        return *this;
    }
    int age;
};
```

而再调用时，谁调用，this就指向谁：

```
void test01()
{
    Person p1(10);//this->age指向p1的成员age
    cout << "p1.age = " << p1.age << endl;
```

```
    Person p2(10);//this->age指向p2的成员age
    p2.PersonAddPerson(p1).PersonAddPerson(p1).PersonAddPerson(p1);
    cout << "p2.age = " << p2.age << endl;
}
```

上述代码中，我们可以看到，在对象Peson的行为PersonAddPerson中返回了*this，这就引出了this的下一个用法——返回对象本身。

例如：this是指向p2的指针，则*this指向该对象的本体。

我们再看一下这个函数：

可以看到返回类型为Person&，而return则是*this。

可以理解为：对*this 返回进行引用，而this指向的对象是Person自身，因此调用了自身。

如此一来，很容易就可以理解下面的两行代码：

```
p2.PersonAddPerson(p1).PersonAddPerson(p1).PersonAddPerson(p1);
cout << "p2.age = " << p2.age << endl;
```

相当于执行了三次**p2.PersonAddPerson(p1)**。

输出：40。

这种编程思想也常常被我们称为链式编程。