```
//一：std::shared_ptr使用场景
//myfunc(12); //如果这块不用shared_ptr变量来接收myfunc返回的结果，那么从myfunc返回的shared_ptr就会被销毁，所
//auto p11 = myfunc(12); //我门用了一个变量来接myfunc的返回值，那么myfunc返回的shared_ptr就不会被销毁，它所指

//二：std::shared_ptr使用陷阱分析:一旦用错，也是致命的
//（2.1）慎用裸指针
//int *p = new int(100); //裸指针
////proc(p); //语法错，int*p不能转换成shared_ptr<int>
//shared_ptr<int> p2(p);
//proc(p2);
////proc(shared_ptr<int>(p)); //参数是个临时的shared_ptr，用一个裸指针显式的构造；
////*p = 45; //潜在的不可预料的问题； 因为p指向的内存已经被释放了；
//*p2 = 45;
//把一个普通裸指针绑定到了一个shared_ptr上之后，那内存管理的责任就交给这个shared_ptr了，这个时候你就不应该再
//用裸指针（内置指针）来访问shared_ptr所指向的内存了；
```

```
//绝对要记住，不要用裸指针初始化多个shared_ptr
//int *p = new int(100); //裸指针
//shared_ptr<int> p1(p);
//shared_ptr<int> p2(p); //p1和p2无关联了(p1和p2每个强引用计数都为1了)，会导致p1和p2所指向的内存被释放两次，产生异常
//
shared_ptr<int> p1(new int);
shared_ptr<int> p2(p1); //这种写法就是OK的，p1和p2指向同一个内存地址并且两者是互通(用的是同一个控制块)
```

```
//（2.2）慎用get()返回的指针
//返回智能指针指向的对象所对应的裸指针（有些函数接口可能只能使用裸指针）
//get返回的指针不能delete，否则会异常
/*shared_ptr<int> myp(new int(100));
int *p = myp.get();*/
//delete p; //不可以删除，会导致 异常
```

```
//不能将其他智能指针绑到get返回的指针上
shared_ptr<int> myp(new int(100));
int *p = myp.get(); //这个指针千万不能随意释放，否则myp就没办法正常管理该指针了
{
    //shared_ptr<int> myp2(p); //现在myp和myp2引用计数都为1，但一旦跳出这个程序块；
    //shared_ptr<int> myp2;
    //myp2 = shared_ptr<int>(p);
    shared_ptr<int> myp2(myp);
}
//离开上边这个myp2的范围，导致myp指向的内存被释放了；
*myp = 65; //该内存已经被释放，这样赋值会导致不可预料的后果；
//结论：永远不要用get得到的指针来初始化另外一个智能指针或者给另外一个智能指针赋值。
```

```
class CT : public enable_shared_from_this<CT>
{
public:
    shared_ptr<CT> getself()
    {
        //return shared_ptr<CT>(this); //用裸指针初始化了多个shared_ptr的感觉；
        return shared_from_this(); //这个就是enable_shared_from_this类中的方法，要通过此方法返回智能指针
    }
};
```

```
//（2.3）不要把类对象指针(this)作为shared_ptr返回，改用enable_shared_from_this
shared_ptr<CT> pct1(new CT);
//shared_ptr<CT> pct2 = pct1; //这是两个强引用；
shared_ptr<CT> pct2 = pct1->getself(); //问题出现；
//用到c++标准库里边的类模板：enable_shared_from_this;
//现在，在外面创建CT对象的智能指针以及通过CT对象返回的this智能指针都是安全的；
//这个enable_shared_from_this中有一个弱指针weak_ptr，这个弱指针能够监视this，
//在我们调用shared_from_this()这个方法时，这个方法内部实际上是调用了这个weak_ptr的lock()方法；
//大家都知道lock()方法会让shared_ptr指针计数+1，同时返回这个shared_ptr，这个就是工作原理；
```

```
//（2.4）避免循环引用：能够导致内存泄漏
//妖异的代码；
```