平台：Android4.4

头文件路径：*system/core/include/utils/RefBase.h*

实现文件：*system/core/libutils/RefBase.cpp*

1. 使用强弱指针sp/wp来管理类对象，那么该类必须继承于基类RefBase。
2. sp是个模板类，使用方法：

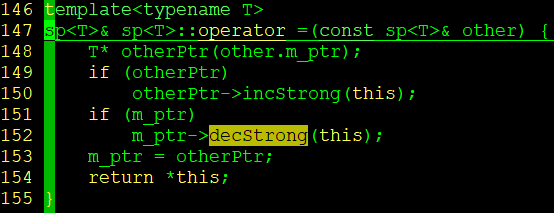
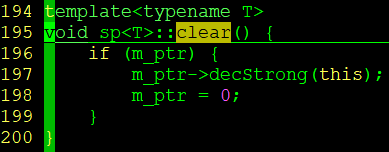
*sp<SomeClass> sp\_sc = new SomeClass();*

其中，sp类声明头文件路径：*system/core/include/utils/StrongPointer.h*

1. sp的clear()方法和=NULL结果一样，可以释放所管理的对象。

即sp\_sc.clear()等价于sp\_sc=NULL;其中，后者是对=的运算符方法重载。

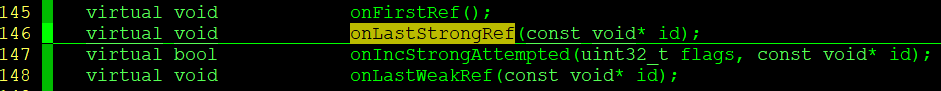
但二者有些许差异之处，operator=还可以针对参数不为NULL的。但当参数为NULL时二者目的是完全一样的。因为：

1. 在一个函数内使用sp的stack栈对象来管理heap堆对象，好处是可以有OS来接管heap对象声明周期处理，因为stack栈对象声明周期是其作用域，当跳出函数时由OS回收其堆栈资源。
2. sp类的实现重载了一些运算符，来访问/调用其真正的类的成员方法。如->,\*，等价于sp的方法get(),来获取内部管理的真正的对象指针：m\_ptr。
3. 使用incStrong()后，一定要显式使用decStrong()，否则容易资源泄漏。

因为，使用incStrong()后，真正的对象至少引用值为2，而clear或=NULL，只会将其计数值减一，但还不会释放。当sp的堆栈对象回收时，因其内部的m\_ptr已经为0了，所以不会释放真正的资源。

1. 派生类可以实现onFirstRef()，用于构造时进行额外初始化工作。第二次及以后引用时不会调用该函数。但是onLastStrongRef()好像不工作->原因找到了。



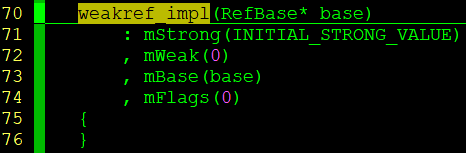
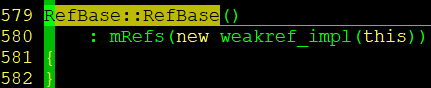
注意参数！！！

1. IncStrong(const void\* id)中有参数，这个参数是什么作用？

使用时将参数设置为NULL时，也会引用递增，但具体参数和NULL，二者有哪些区别？

id其实是这个堆栈对象的指针——this，在正式版本中都可以写成NULL，内部并没有使用该参数。但为了通过编译系统，需要带参数NULL。

1. RefBase的派生类在new后，而不被sp引用时，其RefCounter为初始值：INITIAL\_STRONG\_VALUE（1<28）。
2. RefBase在构造时，内部会构造另外一个类来管理对象的生命周期和计数值(mStrong,mWeak)：

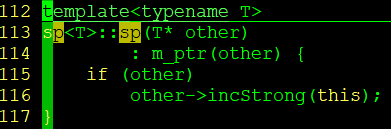
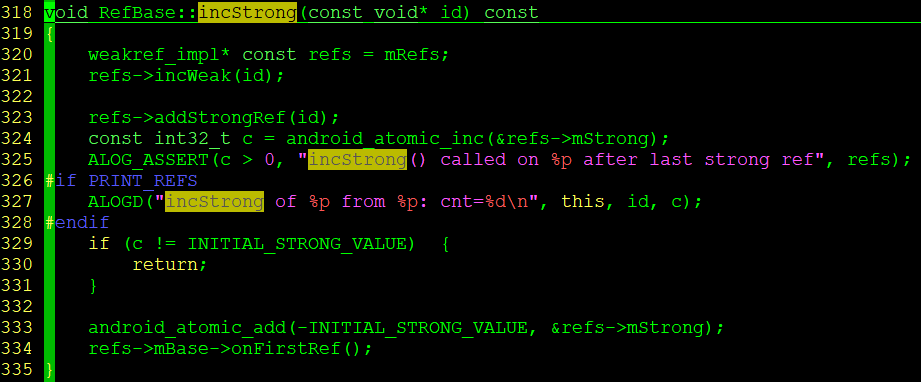


1. sp<SomeClass> sp\_sc = new SomeClass()

该操作的具体动作：

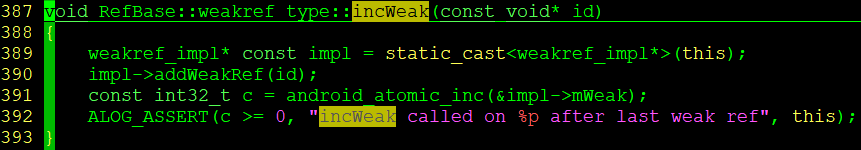
A.SomeClass的构造；

B.sp<T>模板类对象的构造，在其构造过程中，又涉及真正对象引用计数自增。

从中看到参数void\* id为sp<T>对象本身，因为RefBase类为更底层，其不知道sp类的定义，因而使用void\*类型来规避编译错误问题。主要包括以下五个步骤：

*①refs->incWeak(id);//内置的weakref\_impl类型成员（mRefs）的弱引用计数自增（mRefs->mWeak）*



*②refs->addStrongRef(id);//空操作*



*③c = android\_atomic\_inc(&refs->mStrong); //计数自增，变为INITIAL\_STRONG\_VALUE+1*

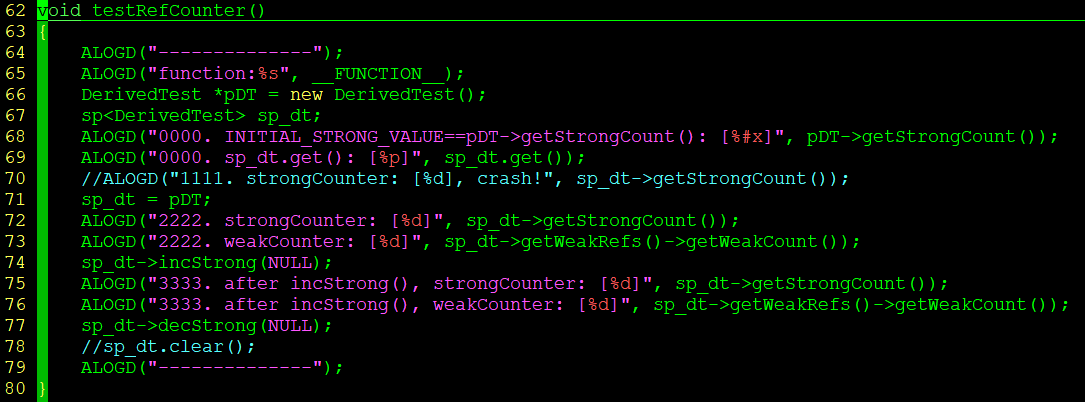
*④android\_atomic\_add(-INITIAL\_STRONG\_VALUE, &refs->mStrong);//又修正为1*

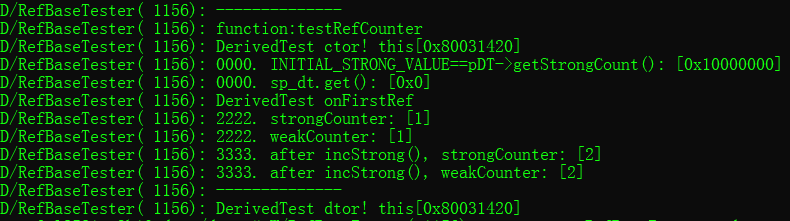
*⑤refs->mBase->onFirstRef();//虚函数，当子类T中没有实现时，使用基类实现的空操作：*



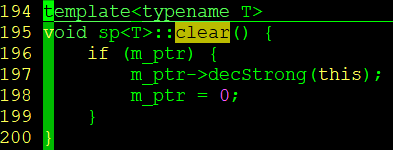
1. 强引用计数值永远大于等于弱引用计数值，因为由StrongRef控制生命周期。

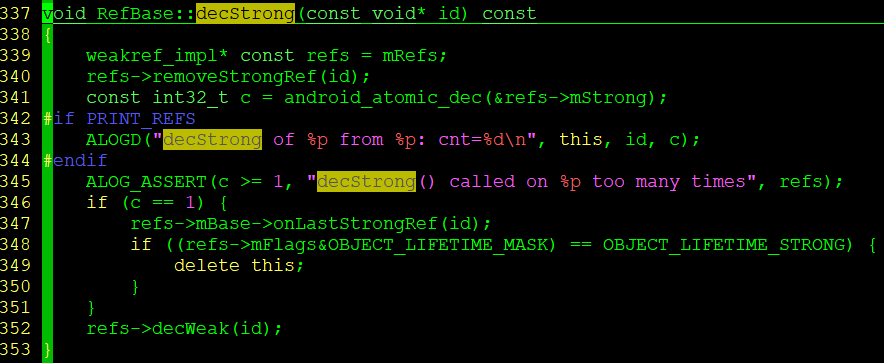
sp\_dt->getStrongCount() >= getWeakRefs()->getWeakCount()





1. 对象释放逻辑操作



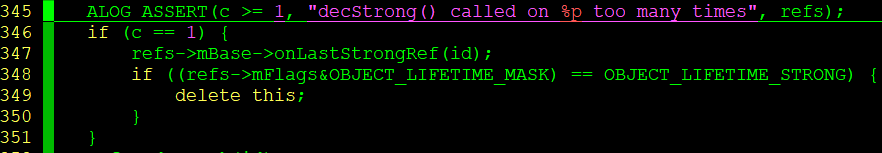


几步核心操作：

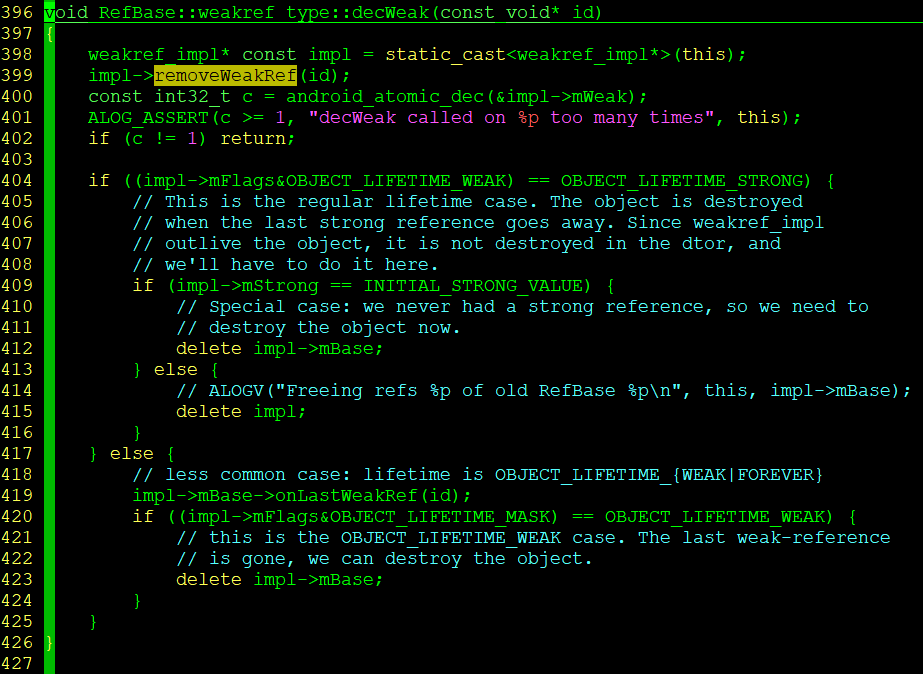
①refs->removeStrongRef(id);//空操作

②c = android\_atomic\_dec(&refs->mStrong);//拿到递减前的强引用值

③如果c==1，表示需要释放资源了：

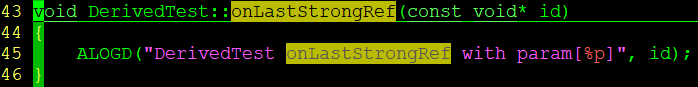


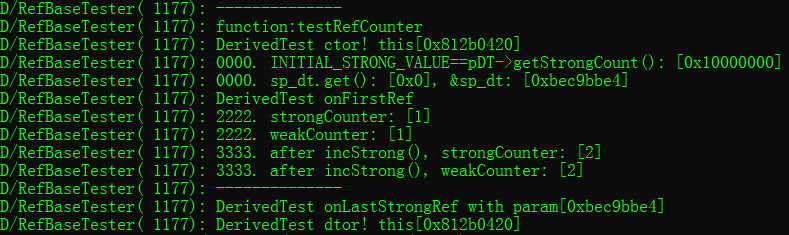
④refs->decWeak(id);//将弱引用计数减一。当弱引用值为1时，走415行将管理器delete掉。



其他注意点：m\_ptr->decStrong(this);中的this指栈对象地址（0xbec9bbe4），从如下可以log看出：







完整调试版本执行情况:

