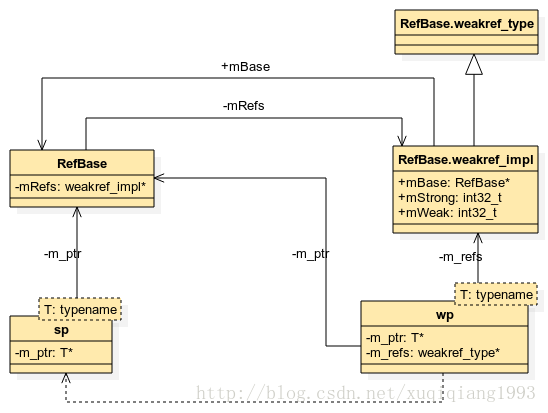
1.smart point 继承关系：



使用sp、wp的对象都要继承于RefBase；

2.smart point的使用：

代码路径：

\system\core\libutils

RefBase.h\RefBase.cpp

\system\core\include\utils\StrongPointer.h

3.类结构：

RefBase：

Public:

void incStrong(const void\* id) const;

void decStrong(const void\* id) const;

void forceIncStrong(const void\* id) const;

weakref\_type\* createWeak(const void\* id) const;

weakref\_type\* getWeakRefs() const;

protected:

virtual void onFirstRef();

virtual void onLastStrongRef(const void\* id);

virtual bool onIncStrongAttempted(uint32\_t flags, const void\* id);

virtual void onLastWeakRef(const void\* id);

Private:

weakref\_impl\* const mRefs;

自定义的类继承于RefBase，在sp、wp这些引用中使用自定义类父类RefBase进行引用计数的操作，最终实现引用计数是在weakref\_impl中。

weakref\_impl：

std::atomic<int32\_t> mStrong;

std::atomic<int32\_t> mWeak;

RefBase\* const mBase;

std::atomic<int32\_t> mFlags;

SP强指针对象的创建：

SP的构造函数和拷贝构造函数：

template<typename T>

sp<T>::sp(T\* other)

: m\_ptr(other) {

if (other)

other->incStrong(this);

}

template<typename T>

sp<T>::sp(const sp<T>& other)

: m\_ptr(other.m\_ptr) {

if (m\_ptr)

m\_ptr->incStrong(this);

}

调用到RefBase类的incStrong方法：

void RefBase::incStrong(const void\* id) const

{

weakref\_impl\* const refs = mRefs;

//为weak引用计数加1

refs->incWeak(id);

//release 版本可以忽略

refs->addStrongRef(id);

//为strong 引用计数加1，返回加1之前的值

const int32\_t c = refs->mStrong.fetch\_add(1, std::memory\_order\_relaxed);

//不是第一次引用直接return

if (c != INITIAL\_STRONG\_VALUE) {

return;

}

//第一次引用将引用计数减去INITIAL\_STRONG\_VALUE，使其变为1

int32\_t old = refs->mStrong.fetch\_sub(INITIAL\_STRONG\_VALUE,

std::memory\_order\_relaxed);

//给自定义对象在第一次引用时候做一些初始化的操作

refs->mBase->onFirstRef();

}

SP的析构函数：

当sp对象在某个作用域使用完毕以后，会调用其析构函数做清理的动作。

template<typename T>

sp<T>::~sp() {

if (m\_ptr)

m\_ptr->decStrong(this);

}

最终调用到自定义对象父类RefBase的decStrong函数：

weakref\_impl 的mFlags：

定义对象持有的mode，自定义子类通过extendObjectLifetime函数来设置.

enum {

OBJECT\_LIFETIME\_STRONG = 0x0000,

OBJECT\_LIFETIME\_WEAK = 0x0001,

OBJECT\_LIFETIME\_MASK = 0x0001

};

void RefBase::decStrong(const void\* id) const

{

weakref\_impl\* const refs = mRefs;

//release 版本不做任何事

refs->removeStrongRef(id);

//将强引用计数器减1，返回减1之前的值。

const int32\_t c = refs->mStrong.fetch\_sub(1, std::memory\_order\_release);

//如果减1之前的值为1说明没有sp引用该对象

if (c == 1) {

std::atomic\_thread\_fence(std::memory\_order\_acquire);

//给自定义对象析构前的最后一次调用进行清理的动作

refs->mBase->onLastStrongRef(id);

int32\_t flags = refs->mFlags.load(std::memory\_order\_relaxed);

//如果对象 life time mode为strong（默认为该值），直接删除该对象

if ((flags&OBJECT\_LIFETIME\_MASK) == OBJECT\_LIFETIME\_STRONG) {

delete this;

}

}

//将弱引用计数减1

refs->decWeak(id);

wp对象的构造：

template<typename T>

wp<T>::wp(T\* other)

: m\_ptr(other)

{

if (other) m\_refs = other->createWeak(this);

}

template<typename T>

wp<T>::wp(const wp<T>& other)

: m\_ptr(other.m\_ptr), m\_refs(other.m\_refs)

{

if (m\_ptr) m\_refs->incWeak(this);

}

void RefBase::weakref\_type::incWeak(const void\* id)

{

weakref\_impl\* const impl = static\_cast<weakref\_impl\*>(this);

impl->addWeakRef(id);

const int32\_t c \_\_unused = impl->mWeak.fetch\_add(1,

std::memory\_order\_relaxed);

ALOG\_ASSERT(c >= 0, "incWeak called on %p after last weak ref", this);

}

wp对象的析构：

template<typename T>

wp<T>::~wp()

{

if (m\_ptr) m\_refs->decWeak(this);

}

void RefBase::weakref\_type::decWeak(const void\* id)

{

weakref\_impl\* const impl = static\_cast<weakref\_impl\*>(this);

impl->removeWeakRef(id);

const int32\_t c = impl->mWeak.fetch\_sub(1, std::memory\_order\_release);

if (c != 1) return;

//没有WP引用该对象

atomic\_thread\_fence(std::memory\_order\_acquire);

int32\_t flags = impl->mFlags.load(std::memory\_order\_relaxed);

if ((flags&OBJECT\_LIFETIME\_MASK) == OBJECT\_LIFETIME\_STRONG) {

if (impl->mStrong.load(std::memory\_order\_relaxed)

== INITIAL\_STRONG\_VALUE) {

//没有SP引用该对象，WP引用计数为0删除该对象

//在~RefBase的析构函数中删除weakref\_impl对象

delete impl->mBase;

} else {

//有SP引用该对象，弱引用计数为0只负责删除weakref\_impl对象

//weakref\_impl对象的生命周期可能大于继承refbase的自定义对象。

delete impl;

}

} else {

//弱引用模式，只有SP和WP引用计数都为0删除定义的对象

impl->mBase->onLastWeakRef(id);

delete impl->mBase;

}

}

总结：

OBJECT\_LIFETIME\_STRONG：

有sp引用对象时，对象的生命完全由sp的引用计数控制，当只有wp引用对象时，对象的生命周期完全由wp来控制。都有的时候由sp的引用计数控制。

OBJECT\_LIFETIME\_WEAK：

对象不受SP引用计数控制，只有WP引用计数为0的时候才会删除该对象。