

深度剖析 C++ 对象池自动回收技术实现

4 回复 139 查看

 (https://www.shiyanlou.com/user/8490) 实验楼管理员  (https://www.shiyanlou.com/vip) 2015-11-30 15:37

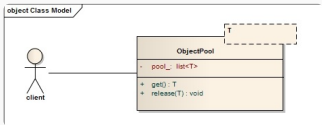
来自： C++ 经典项目实战 (https://www.shiyanlou.com/questions/courses/454)

技术分享 (https://www.shiyanlou.com/questions/?tag=技术分享)

对象池可以显著提高性能，如果一个对象的创建非常耗时或非常昂贵，频繁去创建的话会非常低效。

对象池通过对象复用的方式来避免重复创建对象，它会事先创建一定数量的对象放到池中，当用户需要创建对象的时候，直接从对象池中获取即可，用完对象之后再放回到对象池中，以便复用。


这种方式避免了重复创建耗时或耗资源的大对象，大幅提高了程序性能。本文将探讨对象池的技术特性以及源码实现。





对象池类图

ObjectPool：管理对象实例的pool。

Client：使用者。

 分享到微博

全部回答

 实验楼管理员 (https://www.shiyanlou.com/user/8490)  (https://www.shiyanlou.com/vip)
(https://www.shiyanlou.com/user/8490)

适用性：

- 类的实例可重用。
- 类的实例化过程开销较大。
- 类的实例化的频率较高。

效果：

- 节省了创建类实例的开销。
- 节省了创建类实例的时间。
- 存储空间随着对象的增多而增大。

问题

目前纵观主流语言的实现方式无外乎3个步骤：

- 初始创建一定数量的对象池（也允许从外面添加对象）。
- 从对象池中取对象来使用。

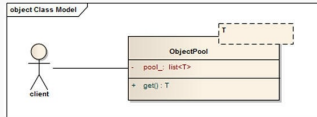
- 用完之后返回对象池。

一般情况下这样是OK的，可能存在的问题是在第三步，有两个问题：

- 不方便，每次都需要显式回收对象。
- 忘记将对象放回对象池，造成资源浪费。

改进动机

解决显式回收的问题，实现自动回收，省心省力。改进之后的对象池无须提供 `release` 方法，对象会自动回收，改进之后的类图如下。



2015-11-30 15:38



实验楼管理员 (<https://www.shiyanlou.com/user/8490>) 💛 (<https://www.shiyanlou.com/vip>)

(<https://www.shiyanlou.com/user/8490>)

技术内幕

借助C++11智能指针，因为智能指针可以自定义删除器，在智能指针释放的时候会调用删除器，在删除器中我们将用完的对象重新放回对象池。

思路比较简单，但实现的时候需要考虑两个问题：

- 什么时候定义删除器？
- 用 `shared_ptr` 还是 `unique_ptr`？

1、什么时候定义删除器

自定义删除器只做一件事，就是将对象重新放入对象池。

如果对象池初始化的时候就自定义删除器的话，删除器中的逻辑是将对象放回对象池，放回的时候无法再定义一个这样的删除器，所以这种做法行不通。需要注意，回收的对象只能是默认删除器的。

除了前述原因之外，另外一个原因是对象池释放的时候需要释放所有的智能指针，释放的时候如果存在自定义删除器将会导致对象无法删除。

只有在 `get` 的时候定义删除器才行，但是初始创建或加入的智能指针是默认删除器，所以我们需要把智能指针的默认删除器改为自定义删除器。

2、用 `shared_ptr` 还是 `unique_ptr`

因为我们需要把智能指针的默认删除器改为自定义删除器，用 `shared_ptr` 会很不方便，因为你无法直接将 `shared_ptr` 的删除器修改为自定义删除器，虽然你可以通过重新创建一个新对象，把原对象拷贝过来的做法来实现，但是这样做效率比较低。

而 `unique_ptr` 由于是独占语义，提供了一种简便的方法方法可以实现修改删除器，所以用 `unique_ptr` 是最适合的。

2015-11-30 15:38



实验楼管理员 (<https://www.shiyanlou.com/user/8490>) 💛 (<https://www.shiyanlou.com/vip>)

(<https://www.shiyanlou.com/user/8490>)

3、实现源码

```

#pragma once
#include <memory>
#include <vector>
#include <functional>

template <class T>
class SimpleObjectPool
{
public:
    using DeleterType = std::function<void(T*)>;

    void add(std::unique_ptr<T> t)
    {
        pool_.push_back(std::move(t));
    }

    std::unique_ptr<T, DeleterType> get()
    {
        if (pool_.empty())
        {
            throw std::logic_error("no more object");
        }

        //every time add custom deleter for default unique_ptr
        std::unique_ptr<T, DeleterType> ptr(pool_.back().release(), [this](T* t)
        {
            pool_.push_back(std::unique_ptr<T>(t));
        }));

        pool_.pop_back();
        return std::move(ptr);
    }

    bool empty() const
    {
        return pool_.empty();
    }

    size_t size() const
    {
        return pool_.size();
    }

private:
    std::vector<std::unique_ptr<T>> pool_;
};

//test code
void test_object_pool()
{
    SimpleObjectPool<A> p;
    p.add(std::unique_ptr<A>(new A()));
    p.add(std::unique_ptr<A>(new A()));
    {
        auto t = p.get();
        p.get();
    }

    {
        p.get();
        p.get();
    }

    std::cout << p.size() << std::endl;
}

```

如果你坚持用 `shared_ptr`，那么回收的时候你需要这样写：

```

std::shared_ptr<T> get()
{
    if (pool_.empty())
    {
        throw std::logic_error("no more object");
    }

    std::shared_ptr<T> ptr = pool_.back();
    auto p = std::shared_ptr<T>(new T(std::move(*ptr.get()))), [this](T* t)
    {
        pool_.push_back(std::shared_ptr<T>(t));
    });

    //std::unique_ptr<T, DeleterType> ptr(pool_.back().release(), [this](T* t)
    //{
    //    pool_.push_back(std::unique_ptr<T>(t));
    //});

    pool_.pop_back();
    return p;
}

```

2015-11-30 15:38



实验楼管理员 (<https://www.shiyanlou.com/user/8490>) [👑 \(https://www.shiyanlou.com/vip\)](https://www.shiyanlou.com/vip)

(<https://www.shiyanlou.com/user/8490>) 这种方式需要每次都创建一个新对象，并且拷贝原来的对象，是一种比较低效的做法。代码仅仅是为了展示如何实现自动回收对象，没有考虑线程安全、对象池扩容策略等细节，源码链接：[object_pool \(https://github.com/qicosmos/cosmos/tree/master/object_pool\)](https://github.com/qicosmos/cosmos/tree/master/object_pool)

总结凡是需要自动回收的场景下都可以使用这种方式：在获取对象的时候将默认删除器改为自定义删除器，确保它可以回收。

注意，回收的智能指针使用的是默认删除器，可以确保对象池释放时能正常释放对象。同时也将获取对象和释放对象时，对象的控制权完全分离。

其他的一些应用场景：多例模式，无需手动释放，自动回收。

文章作者：祁宇

文章地址：<http://www.csdn.net/article/2015-11-27/2826344-C++>

2015-11-30 15:39

登录后才能回答问题哟~

我要提问

标签

Linux (<https://www.shiyanlou.com/questions/?tag=Linux>) Python (<https://www.shiyanlou.com/questions/?tag=Python>)

C/C++ (<https://www.shiyanlou.com/questions/?tag=C/C++>) 实验环境 (<https://www.shiyanlou.com/questions/?tag=实验环境>)

技术分享 (<https://www.shiyanlou.com/questions/?tag=技术分享>) 功能建议 (<https://www.shiyanlou.com/questions/?tag=功能建议>)

课程需求 (<https://www.shiyanlou.com/questions/?tag=课程需求>) Java (<https://www.shiyanlou.com/questions/?tag=Java>)

其他 (<https://www.shiyanlou.com/questions/?tag=其他>) SQL (<https://www.shiyanlou.com/questions/?tag=SQL>)

[NodeJS \(https://www.shiyanlou.com/questions/?tag=NodeJS\)](https://www.shiyanlou.com/questions/?tag=NodeJS) [Hadoop \(https://www.shiyanlou.com/questions/?tag=Hadoop\)](https://www.shiyanlou.com/questions/?tag=Hadoop)

[常见问题 \(https://www.shiyanlou.com/questions/?tag=常见问题\)](https://www.shiyanlou.com/questions/?tag=常见问题) [Web \(https://www.shiyanlou.com/questions/?tag=Web\)](https://www.shiyanlou.com/questions/?tag=Web)

[Shell \(https://www.shiyanlou.com/questions/?tag=Shell\)](https://www.shiyanlou.com/questions/?tag=Shell) [PHP \(https://www.shiyanlou.com/questions/?tag=PHP\)](https://www.shiyanlou.com/questions/?tag=PHP)

[Git \(https://www.shiyanlou.com/questions/?tag=Git\)](https://www.shiyanlou.com/questions/?tag=Git) [HTML \(https://www.shiyanlou.com/questions/?tag=HTML\)](https://www.shiyanlou.com/questions/?tag=HTML)

[HTML5 \(https://www.shiyanlou.com/questions/?tag=HTML5\)](https://www.shiyanlou.com/questions/?tag=HTML5) [信息安全 \(https://www.shiyanlou.com/questions/?tag=信息安全\)](https://www.shiyanlou.com/questions/?tag=信息安全)

[网络 \(https://www.shiyanlou.com/questions/?tag=网络\)](https://www.shiyanlou.com/questions/?tag=网络) [GO \(https://www.shiyanlou.com/questions/?tag=GO\)](https://www.shiyanlou.com/questions/?tag=GO)

[NoSQL \(https://www.shiyanlou.com/questions/?tag=NoSQL\)](https://www.shiyanlou.com/questions/?tag=NoSQL) [训练营 \(https://www.shiyanlou.com/questions/?tag=训练营\)](https://www.shiyanlou.com/questions/?tag=训练营)

[Android \(https://www.shiyanlou.com/questions/?tag=Android\)](https://www.shiyanlou.com/questions/?tag=Android) [Ruby \(https://www.shiyanlou.com/questions/?tag=Ruby\)](https://www.shiyanlou.com/questions/?tag=Ruby)

[Perl \(https://www.shiyanlou.com/questions/?tag=Perl\)](https://www.shiyanlou.com/questions/?tag=Perl)

相关问题

[谈Runtime机制和使用的整体化梳理 \(https://www.shiyanlou.com/questions/3010\)](https://www.shiyanlou.com/questions/3010)

[JavaScript：彻底理解同步、异步和事件循环\(Event Loop\) \(https://www.shiyanlou.com/questions/3009\)](https://www.shiyanlou.com/questions/3009)

[Github上的十大深度学习项目 \(https://www.shiyanlou.com/questions/3000\)](https://www.shiyanlou.com/questions/3000)

[git基础知识整理 \(https://www.shiyanlou.com/questions/2999\)](https://www.shiyanlou.com/questions/2999)

[Linux编程之内存映射 \(https://www.shiyanlou.com/questions/2992\)](https://www.shiyanlou.com/questions/2992)

动手做实验，轻松学IT。

实验楼-通过动手实践的方式学会IT技术。

[公司简介 \(https://www.shiyanlou.com/aboutus\)](https://www.shiyanlou.com/aboutus) [联系我们 \(https://www.shiyanlou.com/contact\)](https://www.shiyanlou.com/contact) [常见问题 \(https://www.shiyanlou.com/faq#howtostart\)](https://www.shiyanlou.com/faq#howtostart)
[我要开课 \(https://www.shiyanlou.com/labs\)](https://www.shiyanlou.com/labs) [隐私协议 \(https://www.shiyanlou.com/privacy\)](https://www.shiyanlou.com/privacy) [会员条款 \(https://www.shiyanlou.com/terms\)](https://www.shiyanlou.com/terms)
[友情链接 \(https://www.shiyanlou.com/friends\)](https://www.shiyanlou.com/friends)

站长统计 (http://www.cnzz.com/stat/website.php?web_id=5902315) 蜀ICP备13019762号 (<http://www.miibeian.gov.cn/>)



QQ群



微信



微博

(<http://weibo.com/shiyanlou2013>)