C++ 智能指针

在C++中，动态内存的申请和释放主要由new、delete这两个关键字实现的，但动态内存的管理较容易出现以下问题，1、有时会忘记释放内存，造成内存泄漏，2、有时在指针指向的内存在使用的情况下释放内存，造成非法引用指针的情况。为了解决上述问题C++标准模板库引入了智能指针，既能自动释放动态内存，又能像普通指针一样引用和解引用指针指向的对象。常见的智能指针主要有unique\_ptr、shared\_ptr 和 weak\_ptr。

1. 智能指针基本用法

智能指针通过->表示用于调用指针原有的方法，而用.表示调用智能指针本身类的方法。

### shared\_ptr

shared\_ptr强调共享所有权，也就是多个shared\_ptr对象可以拥有同一个原生指针的所有权，内部管理一个引用计数，记录shared\_ptr对象拥有原生指针的个数，当引用计数为0时，会自动释放原生对象。如图1所示，创建一个指向int类型的智能指针p1，引用计数为1个，当p1赋值给p2时，引用计数为2，即引用计数记录当前拥有原生指针的个数。

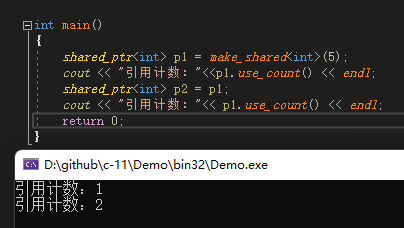


图1 shared\_ptr以及引用计数

常用的函数接口有以下方式，如图2所示、图3所示。

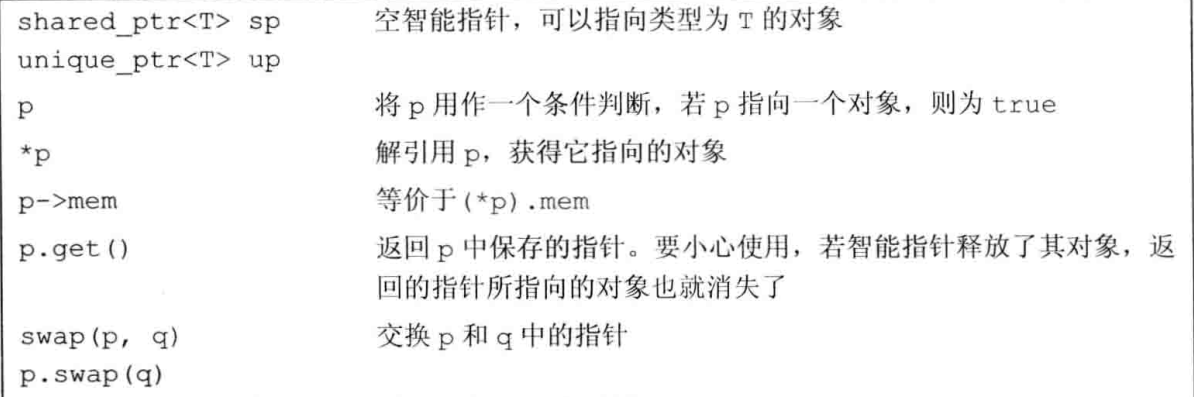


图2 shared\_ptr、unique\_ptr常用接口

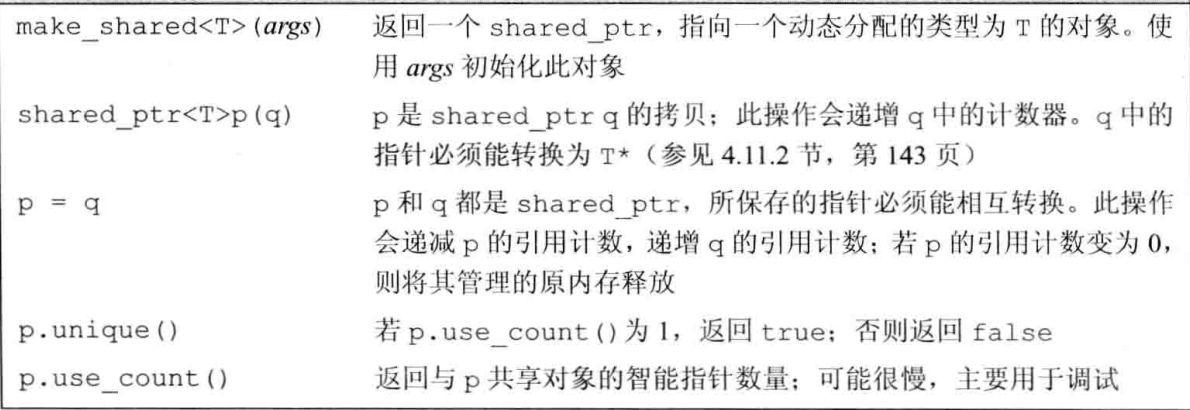


图3 shared\_ptr常用接口

### unique\_ptr

与shared\_ptr不同，某个时刻只能有一个unique\_ptr指向一个给定的对象，它拥有对持有对象的唯一所有权。即两个unique\_ptr不能同时指向同一个对象。也就是unique\_ptr不能被复制到另外一个unique\_ptr，unique\_ptr所持有的对象只能通过转移语义将所有权转移到另外一个unique\_ptr。此外，unique\_ptr与原始指针一样指向一个给定的对象，unique\_ptr效率比shared\_pt效率要高， unique\_ptr常规操作如图3所示。

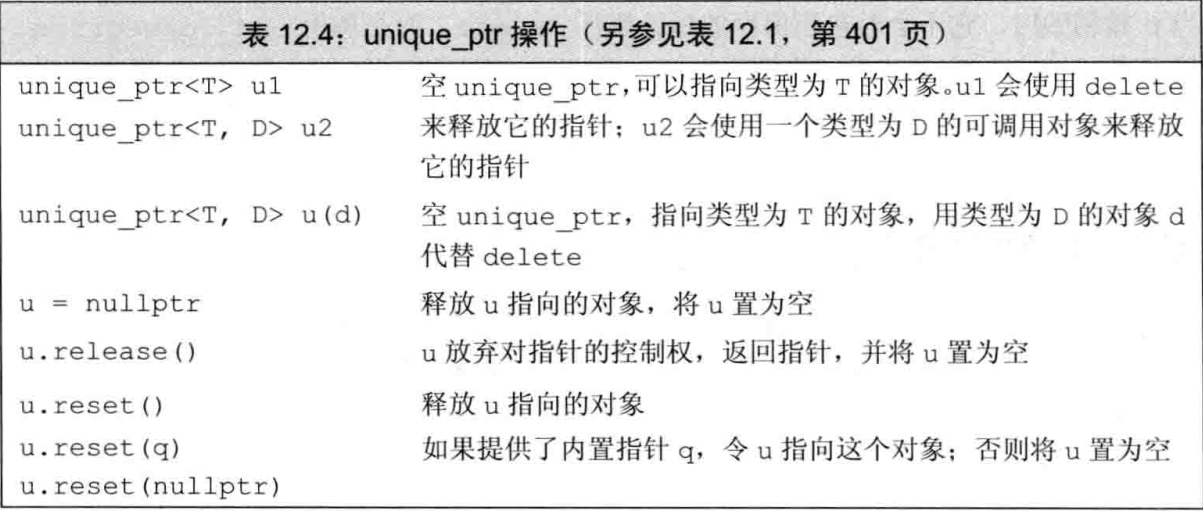


图3 unique\_ptr一些用法

除了上述定义的函数可是否当前的对象，还可以使用右值引用方法转移对象的所有权，而通过std::move()函数可获取绑定在左值上的右值引用，如图4所示。

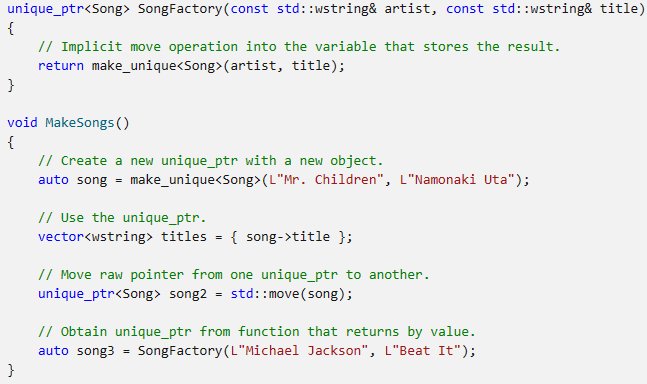


图4 通过右值引用转移对象的所有权

图5演示了两个unique\_ptr 实例通过右值引用转换了对象的所有权。

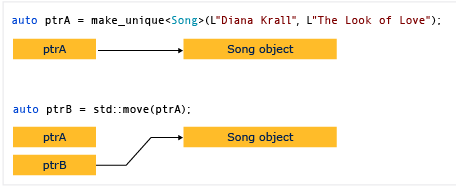


图5对象的所有权转换过程

### 1.3 weak\_ptr

Weak\_ptr是一种不控制所指向对象生存周期的智能指针，它指向由一个shared\_ptr管理的对象。将一个weak\_ptr绑定在一个shared\_ptr不会改变shared\_ptr的引用计数。一旦最后一个指向对象的shared\_ptr被销毁，对象就会被释放。即使有weak\_ptr指向对象，对象也会被释放。常用的weak\_ptr使用方法如图6所示。

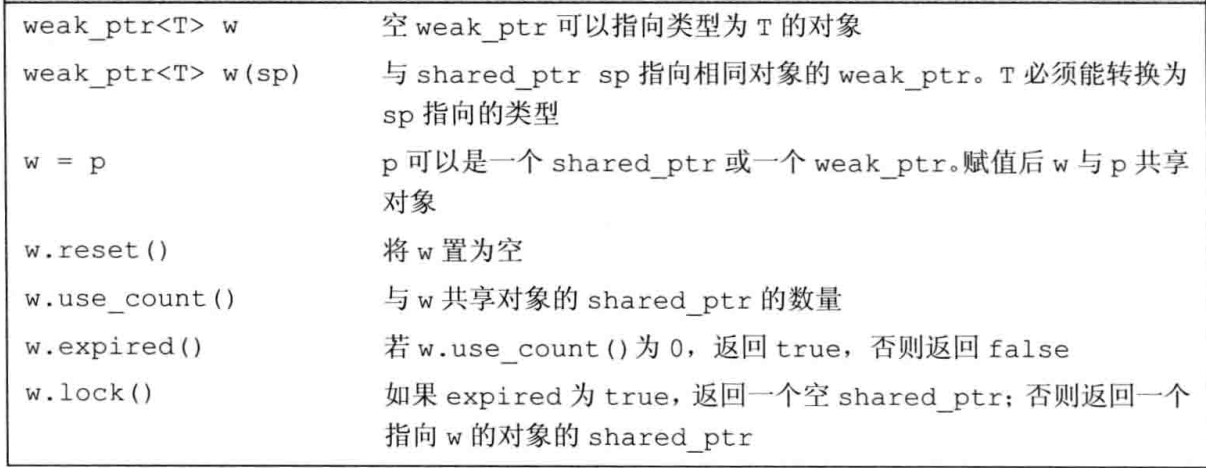


图6 weak\_ptr的使用方法

引入weak\_ptr主要是为了解决shared\_ptr循环引用造成的内存泄露问题,首先运行如下程序，如图7所示。



图7 循环引用

由上图所示，main函数执行完成后，n1、n2对象指针会被回收掉，但未被调用析构函数，造成内存泄露。原因是链表n1中包含了n2指针，n2指针中又包含了n1指针，函数结束后n1、n2的引用计数都不为0，造成资源并未回收。解决上述的问题，可将每个shared\_ptr对象拥有原生指针的个数减少至0，但此与指针指针设计初衷不符，可将上述shared\_ptr改成weak\_ptr，如图8所示，系统调用析构函数。

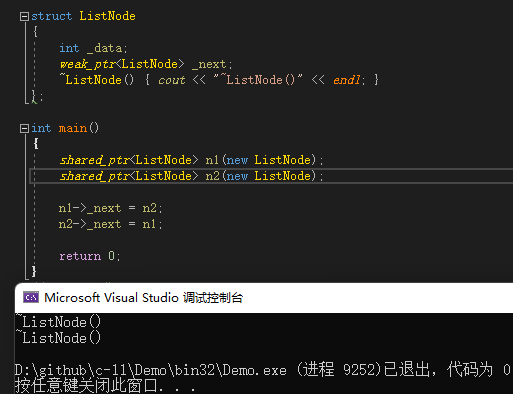


图8 weak\_ptr解决循环引用

二．智能指针的错误用法

### 2.1 使用智能指针，尽量不要用原生指针

智能指针内部会维护原始指针指向的资源，虽然可从智能指针的方法中获取到原生指针，但原生指针异常操作都会影响到智能指针。如原生指针的提前释放、多次引用到其他智能指针中。所以将原生指针交予智能指针管理后，尽量不要出现原生指针。如图9所示，图9中，delete操作会释放内存，但智能指针中的引用计数在为0时，会再次释放内存，造成二次释放内存。

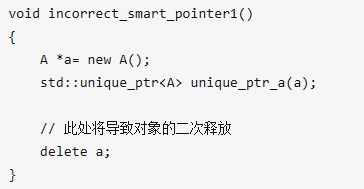


图9 对对象造成二次释放

图10为原生指针交给多个指针指针来管理，当其中一个指针因指定原因对资源进行回收后，其他智能指针或因资源提前释放造成悬空指针对对象的二次释放。

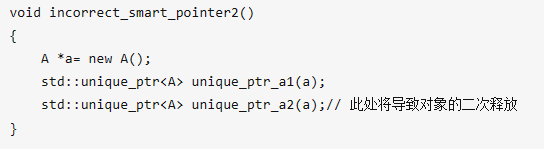


图10 原生指针给多个智能指针管理

### 2.2 将this指针交予智能指针管理

This作用域是在类内部，当在类的非静态成员函数中访问类的非静态成员的时候，编译器会自动将对象本身的地址作为一个隐含参数传递给函数。当this指针指向的对象由智能智能在类的外部管理，而在类的内部，又由this指针又有智能指针管理，造成一个对象由多个指针指针管理，会造成内存的二次释放问题，如图11所示。

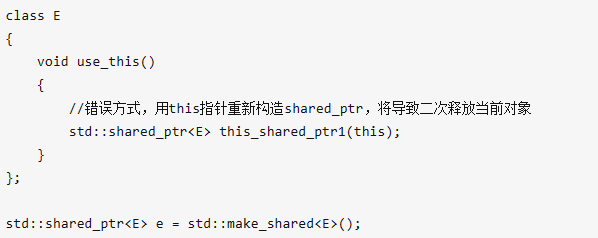


图11 this指针交予智能指针管理

### 2.3 智能指针只能管理动态内存

与原生指针一样，智能指针只能管理动态内存，如将栈内存地址赋值给指针指针，当函数执行完毕后，栈内存自动回收资源，智能指针上的原生指针将成为悬空指针。如图12所示。

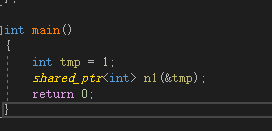


图12 智能指针只能管理动态内存