智能指针：#include <memory>

C++不支持垃圾自动回收机制，程序员必须手动释放动态申请的空间，否则会发生内存泄漏；智能指针就可以保证我们申请的资源，最后忘记释放的问题，防止内存泄露。

智能指针是一个**类模板，对普通指针进行封装，模板参数为指针指向的类型**，使用时智能指针申请在栈空间，在函数结束时，栈上的变量会释放空间，调用智能指针的析构函数，而这个智能指针析构函数的内部实现了对传入指针的释放操作，从而实现了内存的智能释放。

auto_ptr: auto_ptr<int> p(new myclass);

1、不能用于数组

2、支持所有权概念，当一个 auto_ptr 对象被用于另一个对象初始化或者赋值时，左边对象获得所有权，右边对象不在拥有所有权。(指针独占问题)

3、auto_ptr 对象不能用于非 new 动态分配对象

4、两个 auto_ptr 不能指向同一个对象

当 auto_ptr 指针作为函数的值传递时，主调函数内部的智能指针所有权被转移，智能指针失效，只能用引用传递。

unique_ptr:最接近 auto_ptr 的指针，在此基础上提高了犯错误的成本，例如值传递和拷贝构造私有，在写的时候就报错了

1.unique_ptr 持有对对象的独有权—两个 unique_ptr 不能指向一个对象，不能进行复制操作只能进行移动操作

2.提供删除器释放对象，允许用户自定义删除器

3.添加了对象数组的偏特化实现，new[]，delete[]

4.使用 C++ 11 的右值引用特性，实现所有权转移 std::move()

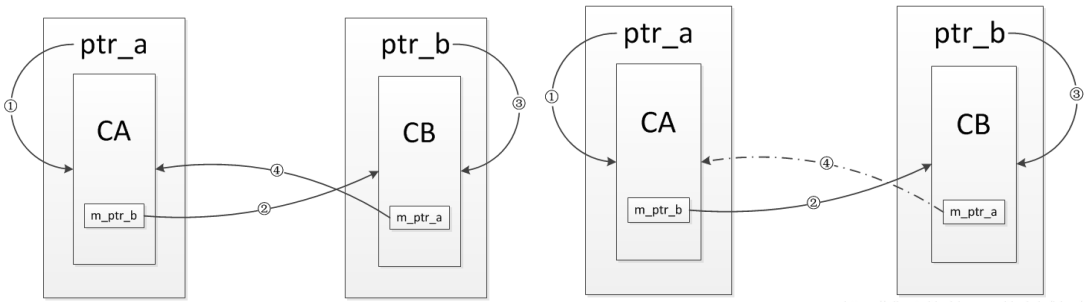
5.线程不安全，但是效率高，一般线程安全用 share_ptr，单线程首选 unique_ptr

share_ptr:

1.通过引用计数解决了 auto_ptr 的指针独占问题，当引用计数为 0 时，析构对象。

2.会存在一个引用成环的问题。当 A 类有一个成员变量是 B 类的智能指针引用，B 类有一个也有成员变量是 A 类的智能指针引用。当声明两个 share_ptr 智能指针分别指向 A 类和 B 类时，同时将 A 的成员变量指向 B，B 的指向 A。当释放 A,B 类的智能指针时，引用计数并不等于 0，导致无法释放，内存泄露。

3.解决办法则是引入 weak_ptr 指正，将 A B 类内部的成员变量引用其中一个改为 weak_ptr，打破这个环路。



weak_ptr:

1. weak_ptr 虽然是一个模板类，但是不能用来直接定义指向原始指针的对象。

2. weak_ptr 接受 shared_ptr 类型的变量赋值，但是反过来是行不通的，需要使用 lock 函数。

3. weak_ptr 设计之初就是为了服务于 shared_ptr 的，所以不增加引用计数就是它的核心功能。

4. 由于不知道什么之后 weak_ptr 所指向的对象就会被析构掉，所以使用之前请先使用 expired 函数检测一下。