谈谈C 内存越界问题及解决方法 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/470/2021_2022__E8_B0_88_E8_B0_88C___E5_c67_470694.htm 与内存泄露相比，C 最令人头痛的问题是内存越界，而内存越界很多情况下是由于悬挂指针引起的。 假设一个指针变量： Object * ptr. 使用ptr时，我们除了要判断ptr是否为0以外，还要怀疑它指向的对象是否有效，是不是已经在别的地方被销毁了。我们希望当它指向的对象被销毁时，ptr被自动置为0。 显然，C 没有这种机制，但是，可以借助于boost::weak_ptr做到这一点。 inline void null_0deleter(void const *) { } class X { private: shared_ptr this_. int i_. public: explicit X(int i): this_(this, amp. rhs): this_(this, amp. operator=(X const &amp. rhs) { i_ = rhs.i_. } weak_ptr weak_this() const { return this_. } }. 定义变量： weak_ptr ptr = x.weak_this(). // x为一个X 对象 则当 x 被销毁时，ptr 被自动置为无效。使用方法如下： if (shard_ptr safePtr = ptr.lock()) safePtr->do_something(). 这种办法用于单线程中，因为 x 对象可能是基于栈分配的。如果需要在多线程中访问X对象，那么最好的办法还是使用shared_ptr 来管理对象的生命期。这样的话，对于safePtr, 可以保证在 safePtr 的生命期内，它所指向的对象不会被其它线程删除。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com