C 箴言：资源管理类的拷贝行为 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/133/2021_2022_C___E7_AE_B4_E8_A8_80_EF_c97_133678.htm 在上一篇文章中介绍了作为资源管理类支柱的 Resource Acquisition Is Initialization (RAII) 原则，并描述了 auto_ptr 和 tr1::shared_ptr 在基于堆的资源上运用这一原则的表现。并非所有的资源都是基于堆的，然而，对于这样的资源，像 auto_ptr 和 tr1::shared_ptr 这样的智能指针通常就不像 resource handlers（资源管理者）那样合适。在这种情况下，有时，你可能要根据你自己的需要去创建你自己的资源管理类。 例如，假设你使用 C API 提供的 lock 和 unlock 函数去操纵 Mutex 类型的互斥体对象： void lock(Mutex *pm). // lock mutex pointed to by pm void unlock(Mutex *pm). // unlock the mutex 为了确保你从不会忘记解锁一个被你加了锁的 Mutex，你希望创建一个类来管理锁。RAII 原则规定了这样一个类的基本结构，通过构造函数获取资源并通过析构函数释放它： class Lock { public: explicit Lock(Mutex *pm) : mutexPtr(pm) { lock(mutexPtr). } // acquire resource ~Lock() { unlock(mutexPtr). } // release resource private: Mutex *mutexPtr. }. 客户按照 RAII 风格的惯例来使用 Lock： Mutex m. // define the mutex you need to use ... { // create block to define critical section Lock ml(&amp.m). // lock the mutex ... // perform critical section operations } // automatically unlock mutex at end // of block 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