计算机二级C 惯用法之RAII计算机二级考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/584/2021_2022__E8_AE_A1_E7_AE_97_E6_9C_BA_E4_c97_584443.htm 2009年下半年全国计算机等级考试你准备好了没?考计算机等级考试的朋友,2009年下半年全国计算机等级考试时间是2009年9月19日至23日。更多优质资料尽在百考试题论坛 百考试题在线题库 C 号称是多范式的通用编程语言， 但是RAII实际上已在C 编程技术中变成不可或缺的核心技术。RAII几乎无处不在的身影不仅仅来自于C 之父的大力提倡， 更来自于这一技术本身的简单，高效和几乎无所不能的适应面。 如果您还没有听说过RAII的话， 那么我在这里再重新叙述一遍， RAII是下列英文短语的首字母缩写： Resource Acquisition IsInitialisation 这句话直译为中文的意思是： 资源获得即初始化。 这只是一个短语， 不能指望靠望文生义来了解字面背后的完整含义，但是短语本身的确反映了重要的论点： 资源是其一， 初始化是其二。 RAII是有关资源的。 资源是一切需要分配的数量有限的资料。 比如， 存储器，文件句柄， 网络套接字端口， 数据库连接，以及线程池等。 基本上， 由于物理的限制， 所有的资料都是有限的。 在某些特殊的情况下，资料由于局部的极大丰富而丧失了资源的意义， 比如沙子， 空气等。但是在大多数情况下， 资料都是有限的， 需要我们善加管理。 资源管理的最基本形式就是善始善终。 申请了资源， 用完了， 就要归还。 在C 程序员生活里最常见的就是内存资源， 资源管理就是内存管理： 申请了内存，不管什么时候逻辑上完成了对这片内存的使用， 内存就要被正确地释放。 注意这里的用词是"不

管什么时候". 在实际应用中， 内存的使用逻辑是如此复杂，使得逻辑上界定某块内存的生命周期会成为非常繁琐非常复杂的任务， 而内存资源就会在人类智力的疏漏中泄漏出去。而即使是简单情况，内存也会在菜鸟程序员手忙脚乱的拙劣中溜走。所以资源管理虽然可以简化为一句"有始有终"， 在实际当中很难得到保证。 有 一类语言， 比如Java，把内存资源接管了， 提供了所谓的自动内存管理， 使用内存分配算法的方式为程序员模拟了一个取之不尽用之不竭的准无穷内存模式。 背后的思想是， 在普通应用中，内存的使用在空间和时间上都是相对集中的， 这就允许用较少的内存来应付时间积累上无限的内存请求。 程序员使用这类语言就不用再考虑内存的释放问题。负担就大大减轻了。 自动内存管理从原理上把内存资源倍增而产生一种资料（准）无限的虚拟环境，从而把程序员从繁重的内存资源管理上解放出来，化更多的精力考虑实际的事务代码， 提高了生产率。 但是它也有自己的局限。 一， 自动内存管理算法比较复杂， 本身的程序就要占一定内存，同时自动内存管理用时间换空间， 还要求实际物理内存至少为应用最大瞬时所需内存的两倍才能较好地发挥作用， 这一要求说明，自动内存管理其实已经不是在管理短缺意义上的"资源"， 而是为不那么浪费地使用丰富的资料提供一种说得过去的代用方案。 其次，由于自动内存管理是与具体的应用分离的， 无法知道最合适的切入点， 所以自动内存管理的介入基本是不可预测的。这限制了自动内存管理在那些对时间响应要求比较严格的程序中的应用。 最后， 自动内存管理仅是对内存资源的管理， 它无法管理其它的资源。 除了内存，程序员往往要和其它的资源打交道。 自动内存

管理模式无法应用到其它类型的资源管理。C 提供了RAII作为一个真正意义上的资源管理实用方案。这也是C 语言在资源管理这一意义更加广泛的问题上作出的贡献。虽然其实用意义如此重大，但是其做法却很简单，就是用类来表示资源，在类的构造函数里分配资源，在类的析构函数里释放资源。比如，class Resource { public： Resource（const char *name）：_resource（alloc_resource（name））{……} ~Resource（）{ release_resource（_resource）；}}；资源类的使用也很简单，按泛围使用。比如，有一个事务处理，使用到了某种资源。如果这一事务可以用一个函数来表示，那么，可以简单地用一个在函数入口处分配的资源变量来表示资源分配。例如：void transaction1（const char*res_name）{ Resource res（res_name）；//后面是使用资源res } 不管程序体内资源res的使用逻辑如何复杂，退出路径如何繁多，C 语言保证了在退出函数范围的时候，资源对象必定得到析构，资源必定得到释放。这一保证甚至包括底层函数抛出异常的情况。所有这些都是免费的，程序员要做的，就是用一个RAII语义的类来表达一类资源，然后用一对标识代码范围的大括号来构勒资源的每一个生命周期范围。如果该事务逻辑过于复杂，无法有效地在单一函数里表达，那么可以用一个类来表达该事务，这个类可以简单地把用到的RAII资源作为成员包含，在表达逻辑上达到了资源和事务逻辑共生死的地步。你会说，这太不够用了，资源的生命周期可能是动态的，无法静态决定。有时候甚至是外部用户决定的。遇上这种情况，智能指针类（其本身就有RAII语义）的引用计数基本上可以解决百分之九十以上的问题。例如在一个很实际的应用中，一个事

务可能由用户通过界面发起执行，发起后可能由于外部资源失败而中止，可能由于用户命令而中止，也有可能是自然执行完毕而中止。一个比较自然的表达是线程池和事务函数。接到用户命令，主程序选择可用的闲置线程，载入该事务函数运行，一旦事务函数因为故障或者自然原因返回，线程重新回到闲置状态。事务函数和主程序用数据同步通讯的方式来实现事务运行状态的控制。在这种方式中，资源成为函数的一部分，其生死已基本不是我们关心的问题，我们只要关心，何时调用事务函数。这已经是比较大比较接近事务逻辑的问题了。如果不能使用线程，或者认为多线程管理要比资源管理还要微妙还要邪恶，那么就要写所谓的异步过程。整个的逻辑就是要在一个线程里通过轮询和动作切割的方式来实现事务和事件的多道并行处理，这仍然可以通过写一个异步事务类来表达，资源将作为成员附着在该事务类上，而所有的异步事务类将作为资源被轮询循环所在的函数自动管理（这是必然的，主循环需要轮流执行当前正在执行的事务）。可见，通过RAII程序员成功地把资源管理问题弱化，转为如何表达事务逻辑上。而这正是程序员的主要任务。也就是说，一旦资源被用RAII的形式封装起来，程序员就不再考虑资源泄漏问题，而考虑如何表达事务逻辑的问题，这个代价并不算大。当然，程序员要坚持只用资源类的对象形式而不是显式动态分配的形式（也就是函数里的普通变量或者事务类里的普通成员，而不是任何new出来的对象形式），否则所有的努力都白费了。这算是一点点代码要求。并不难做到。和自动内存管理比较起来，RAII仅需少量的管理代码（对类不对对象），能普遍适用于各种资源对象的使用，时间上

可以控制和预测。 能为资源管理提供一个统一的模式。 RAII是自由的，它更多是靠程序员对规范的简单遵守（坚持使用对象而不是指针）来达到目的。 我认为， 程序员是需要遵守纪律的， 特别是那些好的纪律。 使用RAII应该成为C 程序员的基本习惯， 这正如书写无错高效代码应该成为每个C 程序员的追求。 特别推荐： C/C 误区一:voidmain() C/C 误区二:fflush(stdin) C/C 误区三:强制转换malloc()的返回值 C/C 误区四:charc=getchar(). C/C 误区五:检查new的返回值 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