保持C_C 程序代码可伸缩性 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/253/2021_2022__E4_BF_9D_E6_8C_81C_C__c97_253839.htm 在今天，已有许多的32位应用程序感到，在32位平台上可用的虚拟内存受到了一定的限制，对程序开发者来说，即使是开始关注64位平台时，也不得不维护软件的32位版本，这就需要一种方法，以使代码的两个版本都保持相当的可伸缩性。 目前的内存剖析工具能帮助确定，当程序达到峰值内存使用量时，都发生了什么，但是这些工具都过于关注已分配的内存块，而不是已提交的虚拟内存地址空间，而这两种衡量标准没有直接的相关性，如内存泄漏、内存碎片、内存块内的空间浪费、或过度延迟的内存单元重新分配这些因素，都会导致不必要的虚拟内存提交。运行时分析工具如IBM Rational Purify或Parasoft Inuse均可以提供内存泄漏及已用内存的描述，这些信息是非常有用的，但是，一个特殊的内存块也许可能、也许不可能影响到虚拟内存覆盖区，另外，甚至一个有碎片的内存堆中的一个小块，也能直接影响到虚拟内存覆盖区。从另一方面来说，在此范围内的任意内存块--甚至泄漏的块，对虚拟内存覆盖区来说，也不会与之有什么关系，除非每一个此范围内有用的内存块能重新分配到一个更紧凑的范围内，这就有点像Java或托管程序的垃圾回收机制，但对大多数C/C 本地应用程序来说，就绝对不可能了，因为在虚拟内存空间中，它们内存块的位置是不确定的。 至于本地代码，不必要的虚拟内存使用，这个实际的问题，比未清理的内存块这个理论上的问题，更加有实质性。未清理的内存块可能导致虚拟内存的浪费，造

成过多的系统开销，但或者不会；这完全依赖于堆管理器是否提交了更多的虚拟内存，以支撑这种浪费。某些很小的未使用的内存块，不会引起不必要的堆"扩展"。与其让你来猜哪一个或多少已浪费的内存块导致了堆扩展，倒不如学会怎样判定出有意义的浪费是什么。当堆中包含不再使用的内存块时，此时通过加入对未缩减堆的检查，就能确定出与你的程序虚拟内存要求有很大关系的、必须进行的内存块清理。为找出哪一个堆中的内存块需多留意，必须在程序中加入一些额外的代码，以跟踪内存堆范围及已分配的内存块。对额外的代码进行条件编译，生成一个特定的版本，也许是一个不错的办法。 为达到此目的，需编写自定义的内存分配例程，并跟踪每一个内存块，另有一个自定义的释放例程，且跟踪虚拟内存中堆的位置，请参见例1与例2的伪代码算法。可能还需编写自定义的访问函数以标记出访问过的内存块，以便于在适当的时候释放虚拟内存，所有这些并不需要过多的内存开销。另一方面，如果你的程序以堆的形式使用了大量的内存，那么将会极大地降低性能，此处的方法也不是长久之计。 例1：/* 输入参数*/ADDRESS triggerAddrSIZE triggerSizeLIST a list of tracked heap rangesIF (the virtual memory at triggerAddr is tracked on the list as part of a heap range)DOIF (triggerAddr triggerSize > (the tracked upper boundary of that heap range))DO/* 一个现有的堆范围被扩展 */make system call(s) to determine the base and extent of the newly committed range that contains the addresses from triggerAddr to (triggerAddr triggerSize)0update the size of the tracked heap range to indicate its new upper limitENDENDELSE DO/* 在triggerAddr中有一个新的

堆范围 */make system call(s) to determine the base and extent of the newly committed range that contains the addresses from triggerAddr to (triggerAddr triggerSize)track the new committed range in the list of heap rangesEND 