Linux系统编程之C 游戏程序优化（3） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022_Linux_E7_B3_BB_E7_BB_c103_144825.htm 1.4 预分配和Cache对象 一个游戏一般会有一些类会频繁的分配和释放，比如武器什么的。在C程序中，你会分配一个大数组然后在需要的时候使用。在C 中，经过小小的规划以后，你也可以这样干。这个方法是不要一直构造和析构对象而是请求一个新而把旧的返回给Cache。Cache可以实现成一个模板，它就可以为所有的有一个缺省构造函数的类工作。Cache模板的Sample可以在附带的CD中找到。 你也可以在需要时分配一些对象来填充Cache，或者预先分配好。如果你还要对这些对象维护一个堆栈的话（表示在你删除对象X之前，你先要删除所有在X后面分配的对象），你可以把Cache分配在一个连续的内存块中。 2、内存管理 C 应用程序一般要比C程序更深入到内存管理的细节。在C中，所有的分配都简单的通过malloc和free来进行，而C 则还可以通过构造临时对象和成员变量来隐式的分配内存。很多C 游戏程序需要自己的内存管理程序。 由于C 游戏程序要执行很多的分配，所以要特别小心堆的碎片。一个方法是选择一条复杂的路：要么在游戏开始后根本不分配任何内存，要么维护一个巨大的连续内存块，并按期释放（比如在关卡之间）。在现代机器上，如果你想对你的内存使用很警惕的话，很严格的规则是没必要的。 第一步是重载new和Delete操作符，使用自己实现的操作符来把游戏最经常的内存分配从malloc定向到预先分配好的内存块去，例如，你发现你任何时候最多有10000个4字节的内存分配，你可以先分配

好40000字节，然后在需要时引用出来。为了跟踪哪些块是空的，可以维护一个由每一个空的块指向下一个空的块的列表free list。在分配的时候，把前面的block移掉，在释放的时候，把这个空块再放到前面去。图1描述了这个free list如何在一个连续的内存块中，与一系列的分配和释放协作的情形。你可以很容易的发现一个游戏是有着许多小小的生命短暂的内存分配，你也许希望为很多小块保留空间。为那些现在没有使用到的东西保留大内存块会浪费很多内存。在一定的尺寸上，你应当把内存分配交给一支不同的大内存分配函数或是直接交给malloc()。 3、虚函数 C 游戏程序的批评者总是把矛头对准虚函数，认为它是一个降低效率的神秘特性。概念性的说，虚函数的机制很简单。为了完成一个对象的虚函数调用，编译器访问对象的虚函数表，获得一个成员函数的指针，设置调用环境，然后跳转到该成员函数的地址上。相对于C程序的函数调用，C程序则是设置调用环境，然后跳转到一个既定的地址上。一个虚函数调用的额外负担是虚函数表的间接指向；由于事先并不知道将要跳转的地址，所以也有可能造成处理器不能命中Cache。 所有真正的C 程序都对虚函数有大量的使用，所以主要的手段是防止在那些极其重视效率的地方的虚函数调用。这里有一个典型的例子： Class BaseClass { public: virtual char *GetPointer()=0. }. Class Class1: public BaseClass { virtual char *GetPointer(). }. Class Class2:public BaseClass { virtual char *GetPointer(). }. void Function(BaseClass *pObj) { char *ptr=pObj->GetPointer(). }如果Function()极其重视效率，我们应当把GetPointer从一个虚函数改成内联函数。一种方式是给BaseClass增加一个新的保护的数据成员，在每

一个类中设置该成员的值，在GetPointer这个内联函数中返回该成员给调用者： Class BaseClass { public: inline char GetPointerFast() { return mpPointer. } protected: inline void SetPointer(char *pData) { mpData = pData. } private: char *mpData. }. void Function(BaseClass *pObj) { char *ptr= pObj->GetPointerFast(). }一个更激进的方法是重新规划你的类继承树，如果Class1和Class2只有一点点不同，那么可以把它们捆绑到同一个类中去，而用一个Flag来表明它将象Class1还是象Class2一样工作，同时在BaseClass中把纯虚函数去掉。这样的话，也可以象前面的例子一样把GetPointer写成内联。这种变通看起来不是很高雅，但是在缺少Cache的机器上跑内循环时，你可能会很愿意为了去掉虚函数调用而把事情做得更加难看。 虽然每一个新的虚函数都只给每个类的虚表增加了一个指针的尺寸（通常是可以忽略的代价），第一个虚函数还是在每一个对象上要求了一个指向虚表的指针。这就是说你在很小的、频繁使用的类上使用任何虚函数而造成了额外的负担，这些都是不能接受的。由于继承一般都要用到一个或几个虚函数（至少有一个虚的析构函数），所以你没必要在小而频繁使用的对象上使用任何继承。 