一种内存管理设计模式

许冬海,黄明和,许生模
(江西师范大学,江西 南昌 330022)

摘 要:主要阐述一种内存管理器的设计模式,包括内存管理器的创建,在内存管理器中创建一对象,释放一对象,查找一对象,在实际设计中利用这种设计模式,可以方便构建自己的内存管理器,同时也能提高整个应用程序的运行效率,屏蔽底层操作系统的内存管理差异。采用本设计模式创建的内存管理器已经在一平台无关的动画播放器库函数中使用,播放器中的一些常用图片,如图标等皆放置管理器中。播放器移植时无需改动播放器本身代码,只需利用本设计模式重新实现本内存管理器。在同一款手机上测试时发现,用本内存管理器能部分提高性能。本模式同样可以应用到整个系统中框架中,它可以有效减少系统设计的复杂度和提高整个系统的效率。

关键词:Hash散列;设计模式;共享内存
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1673-629X(2008)09-0072-04

A Memory Management Design Pattern

XU Dong-hai, HUANG Ming-he, XU Sheng-mo
(Jiangxi Normal University, Nanchang 330022, China)

Abstract: Mainly discuss a memory management design pattern, including creating the memory management, memory management create an object, release an object and search an object. In development using the design pattern that can facilitate the creation for the memory management, can enhance the application efficiency, and it also can shield the difference between operating system memory management. A memory management designed with this theory is applied in a mobile animation player. Icons, pictures in the player are cached in the memory management. Through experiments found the player's performance has some improvements. The design pattern also can apply to framework in mobile or PDA system.

Key words: Hash; design pattern; shared memory

0 引 言

随着计算机技术的发展,计算机软硬件已经应用到各行各业,针对某个具体应用设计一套操作系统,从而造成文件管理、内存管理等等方式各不相同,提供的应用编程接口也差异较大,从而造成应用软件无法在这些操作系统间移植。应用软件要在操作系统之间移植,一般有两种方式:一是操作系统设计时尽量与标准一致;另一种方式是在操作系统与应用软件之间设计中间层,使应用软件与操作系统之间的耦合度降低。目前,市场上的操作系统种类繁多,且各具特色,加上标准制定过晚,所以很多操作系统都只能尽量逼近标准,对外部提供的应用编程接口也不尽相同,这导致应用移植时只能采用第二种方式(软件框架见图1)。

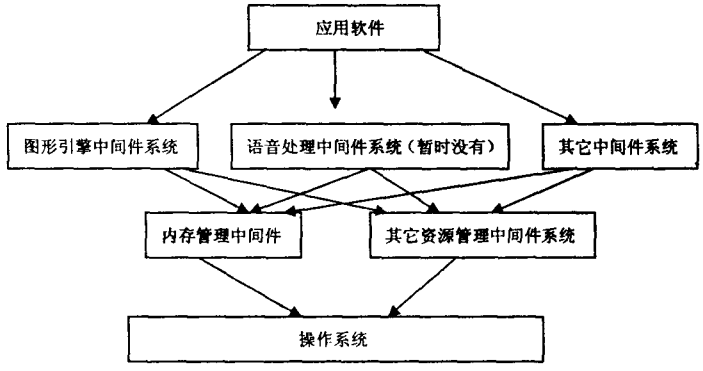

图1 采用中间件的软件框架

由于中间件向应用软件提供接口,对操作系统进一步封装,所以中间件的设计时必须考虑到通用性以及良好的性能,这使得中间件的设计非常复杂[1]。

在参考了一些内存设计管理方式的基础上[2],提

收稿日期:2007-12-01
基金项目:国家自然科学基金(60363002);教育部科技重点项目(206078)
作者简介:许冬海(1983-),男,江西宁都人,硕士研究生,主要从事嵌入式系统研究;黄明和,教授,硕士生导师,研究方向为算法设计与面向对象技术。

出一种内存管理中间件设计模式,它能有效地屏蔽操作系统内存管理的差异,使应用软件可以更方便地在不同操作系统之间移植,同时能方便应用开发中的设计[3,4];另外,对于应用中频繁使用的较大对象,频繁地在操作系统中申请和回收,将导致系统整体性能下降[5],所以内存管理中间件一次性向操作系统申请一块合适大小的内存自行管理,不但可以提高应用软件的运行效率,也有利于应用软件在不同操作系统间进行移植。尤其是一些图形引擎库,一次性向操作系统申请一块较大内存自己管理能显著提高运行效率。

1　内存管理器的设计思想

内存管理器为应用软件访问内存提供统一的接口,由于本设计的通用性,所以把本设计抽象为模式[6],为了提供系统的整体效率,管理器提供两套应用编程接口,它的大致框架见图2。

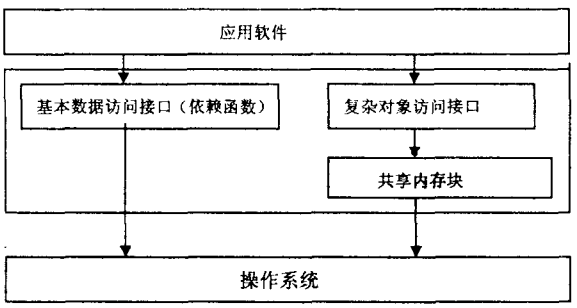

图2　抽象模式软件框架

对于简单数据,管理器只对操作系统的API进行简单封装,把封装后的函数称为依赖函数,对于复杂对象(比如图片对象)管理器采用共享内存方式。

管理器采用共享内存方式时,首先向操作系统申请一块较大内存块。管理器对上层应用软件提供相应编程接口,其中包括在共享内存中创建一对象,查找一对象,删除一对象等。在目前主流操作系统中,由于内存大都采用段页式管理,而对于小于一页大小的内存块往往难以利用,从而造成内存碎片。在本内存管理器中,由于对象大小可以不一,所以内存也无须定长分配从而不会造成碎片现象。在设计中把整个管理器管理的共享内存块组成一个链表形式(称为NODE链),管理器刚创建时链表中只有一个接点(NODE链中的接点称为NODE接点),NODE接点中包括三个指针(具体设计中可以用相对偏移):preNode, nextNode, nextFreeNode, 其中preNode指向前一个NODE接点,nextNode指向下一个NODE接点,而nextFreeNode指向下一个空的接点。这样整个全局共享空间便形成了一条NODE接点链。为了方便申请和释放一对象,在NODE链中同时包括一个空闲NODE接点链表,即NODE接点中nextFreeNode指针串在一块形成了一条链表(称为Free链)。管理器对外提供服务时对这两个链表进行操作,如果是申请或者释放一对象,则必须对这两个链表进行调整。

为了使管理器更方便查找到指定的一对象,管理器内置了一个Hash数组,当某个对象需要创建申请一个NODE接点时,对这个对象的描述(如果是图片,可以是图片名)进行散列(假设散列到K),把这个NODE接点的地址放到数组第K个元素的value中。

当应用程序删除一对象时,对象所对应的NODE接点并不会立即被删除,因为可能被其他对象引用到,所以对其引用数减一,当减为零时,这个对象可以被删除,但当外部应用程序申请内存的请求还能满足时,管理器不会删除引用为零的对象,而是把引用为零的对象通过一个叫DEL的数组管理起来,这样当外部应用下次需要再次创建该对象时,会发现有该对象已经在管理器中,当上层应用程序申请内存失败时才释放DEL数组指向的对象。

当应用程序为对象申请一空间时,首先通过Hash散列检查此对象是否存在内存管理器中,如果此对象在内存管理器中已经存在则无须再打开而直接返回。

管理器主要是通过对Free链、NODE链、DEL数组、Hash数组进行操作,各数据结构具体说明如下:

(1)NODE链:共享内存是一块连续的空间,当对象创建时,在共享内存中创建一个NODE接点,对象的数据拷贝到接点中;NODE接点提供三个指针:preNode、nextNode、nextFreeNode, preNode指向前一个NODE接点,nextNode指向下一个NODE接点,而nextFreeNode指向下一个空的NODE接点。这样整个全局共享空间便形成了一条NODE链。

(2)Free链:Free链实际上是包含在NODE链中的,Free链是单向链表,链表中的每个接点都是可用(即为空闲)的NODE接点。当需要分配空间时在本链中找一个大小合适的接点返回,然后调整NODE链和Free链。

除了上面两个链外,在内存管理器中还包含着Hash数组、待释放图片数组(DEL数组)。

(3)Hash数组:用来方便对象查找的数组,数组项的值是对象的地址,可以统一为偏移,数组下标是图像名进行Hash散列得到的值。当需要查找图片时,根据输入的图片名字进行散列,得到图片的地址,从而可以提高查找效率。

(4)DEL数组:即待释放数组,数组项的值同样是NODE接点的地址,当某个对象的引用为零时,它的地址放到此数组中。当此数组满或外部应用申请一片内

存时再统一释放对象。

2 内存管理器设计方案

2.1 管理器的创建

在实际设计时,可把管理器抽象成一组类,由于内存管理器可能会同时被多个进程使用,所以当管理器初始化时可以向操作系统申请一块共享内存,各个进程可以把这块空间影射到自己的进程地址空间,对于每个进程,管理器必须提供一个接口返回自己的一个引用,而且必须保证此引用的对象在整个系统中的唯一性,设计时采用 SINGLETON 设计模式,全局共享内存在管理器初始化时创建,系统提供了统一的管理器的创建方法,假设为 get - MemManager(),它能保证在进程中只有一个管理器的对象。在管理器类实例化时,先调用操作系统 API 获得一块大小为 SIZE(根据需要定)的空间,然后通过操作系统 API 把这块空间影射到进程的地址空间。管理器设计时可采用对象组合的方式,一种可能设计方案见图 3。

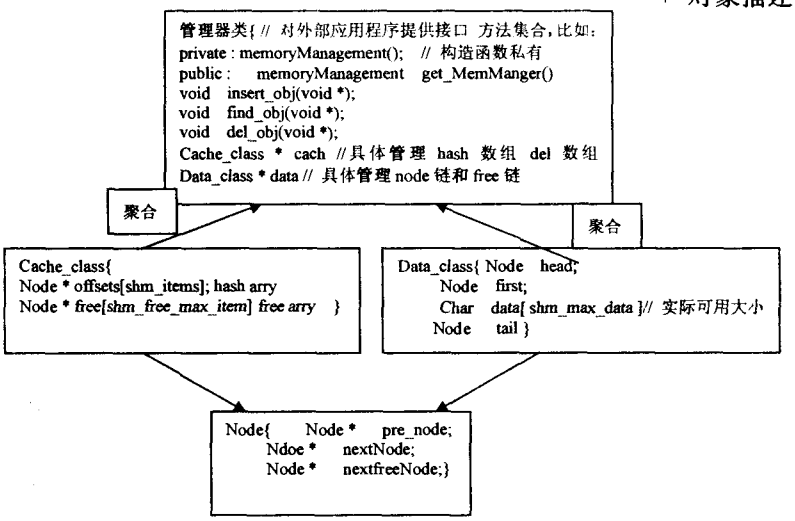

图 3 对象组合方式设计

构造函数结束后管理器中内存情况见图 4。

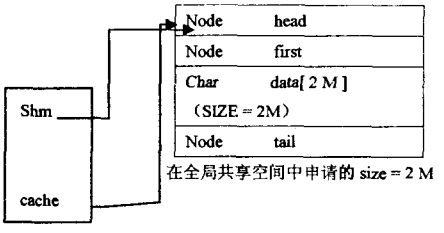

图 4 管理器内存情况

在内存中管理器管理的空间如图 4,管理方式通过下面链表示式进行(见图 5)。

管理器初始化的同时也完成了共享内存的创建,并且使成员变量初始化,即初始化了 NODE 链、free 链、hash 数组、待释放图片数组。

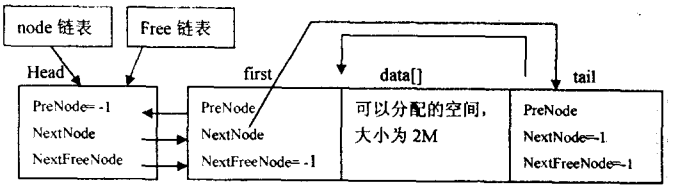

图 5 管理方式

2.2 创建一对象

内存管理器类为对象的创建提供了很好的接口,接口实现算法如下:

输入:要加入的对象以及此对象的描述(比如对象为图片,则可以是图片名)。

输出:对全局共享内存的更改,即对 Free 链、NODE 链、Hash 数组的更改。

算法:

(1)通过对象描述散列到 Hash 数组中进行查找,若找到则返回这个对象,若没找到往下执行。

(2)计算此对象需要的大小(size = 对象实际大小 + 对象描述信息大小);

(3)分配一块大小为 size 的空间并把对象与对象相关的信息拷贝到此空间,在设计时,对象以及对象相关的信息可用对象组合的方式,把它们作为管理器类的成员变量。再对对象描述信息进行 Hash(散列)得出一个序列码 K,然后把 K 当作 Hash 数组的下标,使这个数组项的值字段等于此次分配的内存地址。如果已经没有空间分配,则自动释放没有被对象引用到的图片。

管理器创建对象的过程实际上较复杂,通过 head 访问 freeNODE 链,找到一块大于或等于 size 的空间,如果大于则要把这个空间拆分成两个 NODE 接点,修改 freeNODE 链和 NODE 链,如果恰好等于则直接修改 freeNODE 链。

在实际设计中,可以创建两个类,一个负责从 Hash 数组访问 NODE 链,另一个从头接点访问 NODE 链和 freeNODE 链。这两个类同样以组合的方式出现在管理器类中。插入一对象后的示意图见图 6。

2.3 释放一对象

管理器同样提供了释放一对象的过程,目的是删除一个保存对象的 NODE 接点,但 NODE 接点是包括对象与其相关描述信息的整体,所以必定是多个对象的聚合,释放过程遵循先内后外,最后释放一个 NODE 接点。释放时需要修改 Hash 数组、待释放数组,然后

删除 NODE 接点,同时调整 NODE 链和 Free 链。

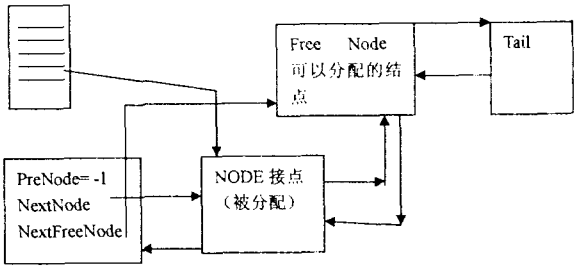

图6　管理器的链式表示状态

大致算法如下:

输入:一个对象指针。

输出:对 NODE 链、Free 链的修改。

算法:

(1)把这个对象指针对应的 NODE 接点设置为 Free。

(2)判断此 NODE 的前驱接点是否为 Free,是则合并这两个 NODE 接点。

(3)判断此 NODE 的后继接点是否为 Free,如果是合并这两个接点。

(4)向下调整 freeNode 链,从此接点开始依 NODE 链向下(即每次按 nextNode 往下查)查找第一个 Free 接点,找到则把它设置为此接点的后继 Free 接点。

(5)向上调整 freeNode 链,从 pre 接点开始依 NODE 链向上(即每次按 preNode 往上查)找第一个 Free 接点,找到则把此接点设置为它的后继 Free 接点。

(6)调整 Hash 数组和 Free 数组。

2.4　管理器中查找一对象

在管理器中查找一对象相对较简单,因为不存在对 NODE 链表和 freeNode 链表调整,也不用对 Hash 数组和 DEL 数组进行修改。

输入:对象描述信息。

输出:指向此对象的指针,否则为 NULL。

算法:

(1)对对象描述信息进行散列计算出输入值在 Hash 数组中对应的下标,根据这个下标找到图片的存放位置,然后找到一个对象。

(2)如果此对象为空则没找到,否则比较这个对象描述信息与输入的对象描述信息是否相等。

(3)如相等则找到了,返回这个对象,且对这个对象的引用加一。

(4)如果没找到继续执行 Hash(),计算出新的数组下标,重复上面的步骤直到找到要找的图片的地址或返回一个空的。

3　结束语

本内存管理器设计模式已经在一平台无关的播放器库函数中使用,播放器中的一些常用图片,如图标等皆放置管理器中。播放器移植时无需改动播放器本身代码,只需利用本设计模式重新实现本内存管理器。在同一款手机上测试时发现,用本内存管理器能部分提高性能。本模式可以应用到整个系统中,它可以有效的减少系统设计的复杂度和提高整个系统的效率。

文中创新点:提出一种内存管理中间件设计模式,能有效地屏蔽操作系统内存管理的差异,使应用软件可以更方便地在不同操作系统之间移植,同时能方便应用开发中的设计。

参考文献:

[1] Brooks F P. No Silver Bullet: Essence and Accidents of Software Engineering[J]. IEEE Computer, 1987, 20(4): 10-19.

[2] 杨　雷,吴　珏,陈汶滨.实时系统中动静结合的内存管理实现[J].微计算机信息,2005(2):15-16.

[3] Johnson R E. Documenting Frameworks using Patterns[C]// Proceedings OOPSLA '92, ACM SIGPLAN Notices. [s.l.]: [s.n.], 1992:63-76.

[4] Gruijs D. A framework of Concepts for Representing Object-Oriented Design and Design Patterns[R]. [s.l.]: Utrecht University, 1997.

[5] Barr M. C/C++嵌入式系统编程[M].于志宏译.北京:中国电力出版社,2001.

[6] Shull F, Melo W L, Basili V R. An Inductive Method for Discovering Design Patterns from Object-Oriented Software Systems[R]. [s.l.]: University of Maryland, 1996.

(上接第71页)

参考文献:

[1] 金　纯,许光辰,孙　睿.蓝牙技术[M].北京:电子工业出版社,2001.

[2] 吴永忠,韩江洪.蓝牙技术综述[J].微型机与应用,2001(6):60-63.

[3] 吴永忠,韩江洪.蓝牙基带层包格式简析及相关芯片简介[J].计算机应用,2001(10):68-70.

[4] 宋　军,顾冠群.多媒体同步[J].中国图象图形学报,1997(6):52-55.

[5] 宣善立,吴永忠,韩江洪.蓝牙技术安全研究[J].计算机工程,2002(9):42-45.

[6] 吴永忠,韩江洪.蓝牙基带层数据包类型简析[C]//全国第十四届计算机科学及其在仪器仪表中的应用学术交流会论文集.合肥:中国科学技术大学出版社,2001.