

简介

理解 SQL Server 对于内存的管理是对于 SQL Server 问题处理和性能调优的基本，本篇文章讲述 SQL Server 对于内存管理的内存原理。

二级存储(secondary storage)

对于计算机来说，存储体系是分层级的。离 CPU 越近的地方速度愉快，但容量越小(如图 1 所示)。比如：传统的计算机存储体系结构离 CPU 由近到远依次是:CPU 内的寄存器；一级缓存，二级缓存，内存，硬盘。但同时离 CPU 越远的存储系统都会比之前的存储系统大一个数量级。比如硬盘通常要比同时代的内存大一个数量级。

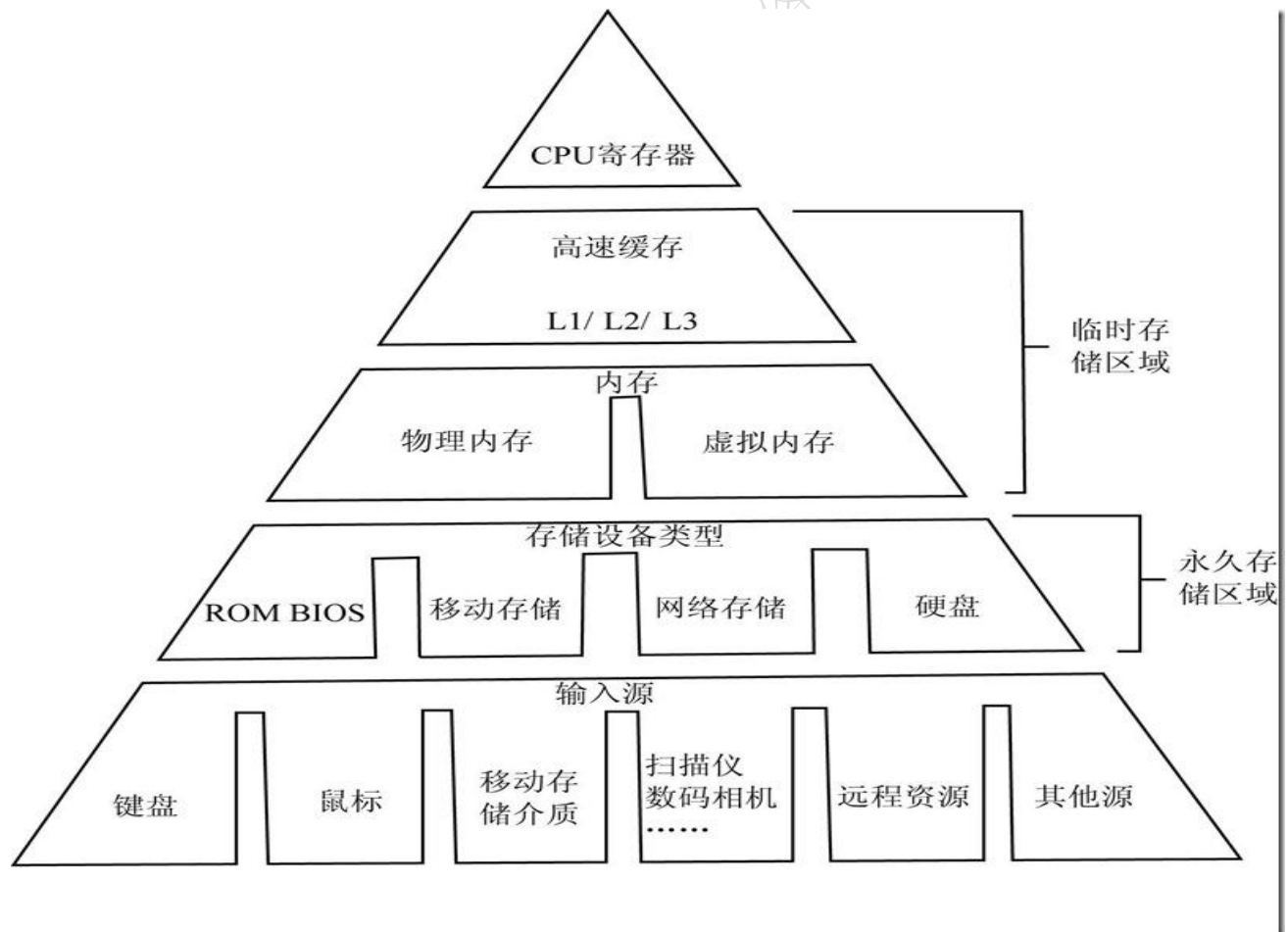


图 1. 计算机存储体系

因此对于 SQL Server 来说, 正常的生产系统所配置的内存通常不能装载所有数据, 因此会涉及到二级存储, 也就是磁盘。磁盘作为现代计算机系统中最后的机械存储部件, 读取数据需要移动磁头 (具体关于磁盘的原理, 可以看我之前写的一篇[文章](#)), 并且由于数据库所访问的数据往往是随机分布在磁盘的各个位置, 因此如果频繁的读取磁盘需要频繁的移动磁头, 这个性能将会十分底下。

由计算机体存储体系结构可以知道, 计算机对于所有硬盘内数据的操作都需要首先读取到内存, 因此利用好内存的缓冲区而减少对磁盘 IO 的访问将会是提升 SQL Server 性能的关键, 这也是本篇文章写作的出发点之一。

SQL Server 引擎, 一个自我调整的引擎

由于 SQL Server 过去一直面向是中小型企业市场的原因, SQL Server 存储引擎被设计成一个不需要太多配置就能使用的产品, 从而减少了部署成本, 但这也是很多人一直诟病的微软开放的配置过少。而对于 SQL Server 如何使用内存, 几乎没有直接可以配置的空间, 仅仅开放的配置只有是否使用 AWE, 以及实例占用的最大或最小内存, 如图 2 所示。

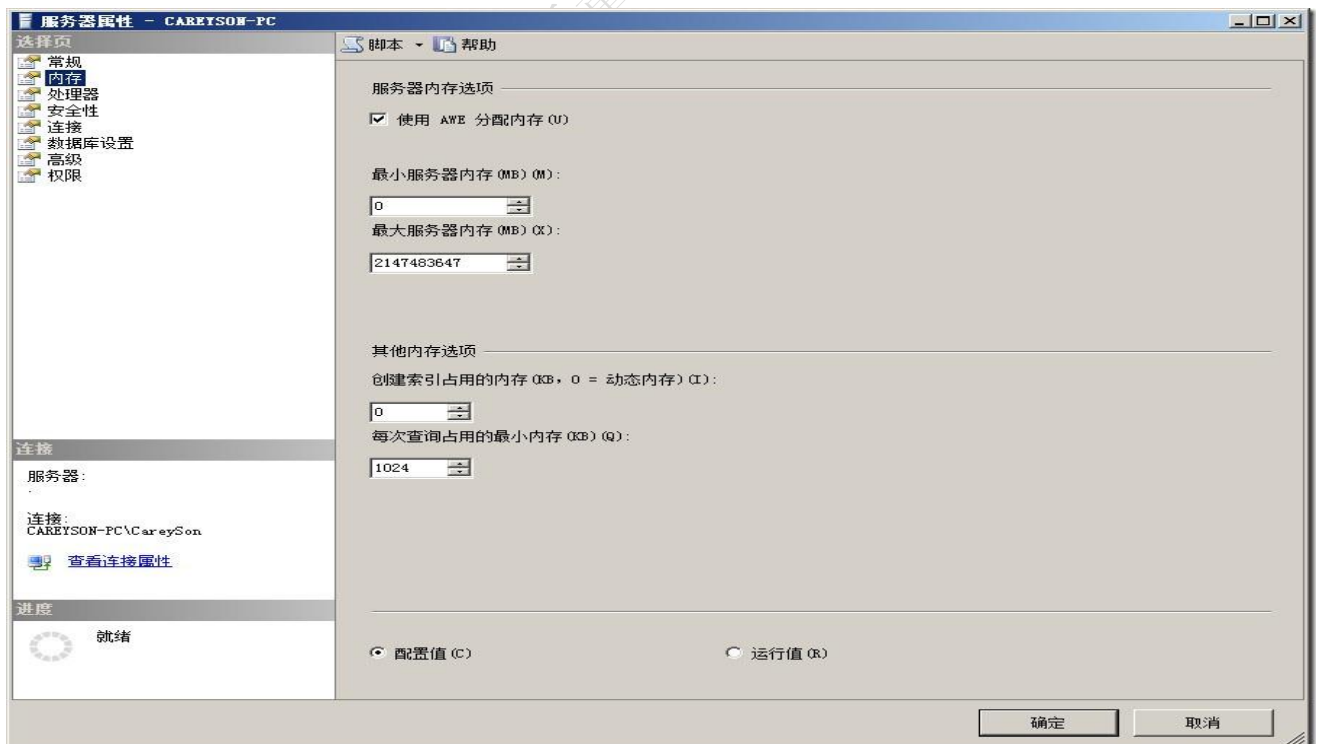


图 2. SQL Server 可控控制内存的选项

而对于具体的 SQL Server 如何使用内存，例如分配给执行计划缓存多少，分配给数据 buffer 多少，这些都无法通过配置进行调控。这也是很多其它技术的开发人员对于使用微软技术的开发人员充满优越感的原因，而在我看来，虽然 SQL Server 提供可控配置的地方很少，但是很多地方都可以在通晓原理的情况下进行“间接”的配置。这也需要了解一些 Windows 的原理。

SQL Server 是如何使用内存的

SQL Server 存储引擎本身是一个 Windows 下的进程，所以 SQL Server 使用内存和其它 Windows 进程一样，都需要向 Windows 申请内存。从 Windows 申请到内存之后，SQL Server 使用内存粗略可以分为两部分：缓冲池内存（数据页和空闲页），非缓冲内存（线程，DLL，链接服务器等）。而缓冲池内存占据了 SQL Server 的大部分内存使用。缓冲池所占内存也就是图 2 最大最小内存所设置的，因此 `sqlservr.exe` 所占的内存有可能会大于图 2 中所设置的最大内存。

还有一点是，SQL Server 使用内存的特点是：有多少用多少，并且用了以后不释放（除非收到 Windows 内存压力的通知）。比如我所在公司的开发服务器，在几乎没有负载的时候来看内存使用，如图 3 所示。



图 3. SQL Server 进程的内存使用

可以看到 CPU 在 0 负载的时候，内存却占据了 13 个 G。这其实是在之前的使用 SQL Server 向 Windows 申请的内存一直没有释放所致。

具体 SQL Server 能够使用多少内存是由以下几个因素决定的：

1. 物理内存的大小
2. 所安装 Windows 版本对于内存的限制(比如 windows server 2008 标准版限制最大内存只能使用 32GB)
3. SQL Server 是 32 位或 64 位
4. 如图 2 所示配置 SQL Server 对于内存的使用量
5. SQL Server 的版本(比如 express 版只能用 1G 内存)

SQL Server OS 的三层内存分配

SQL Server OS 对于内存的分配分为三个层级,依赖关系如图 4 所示。

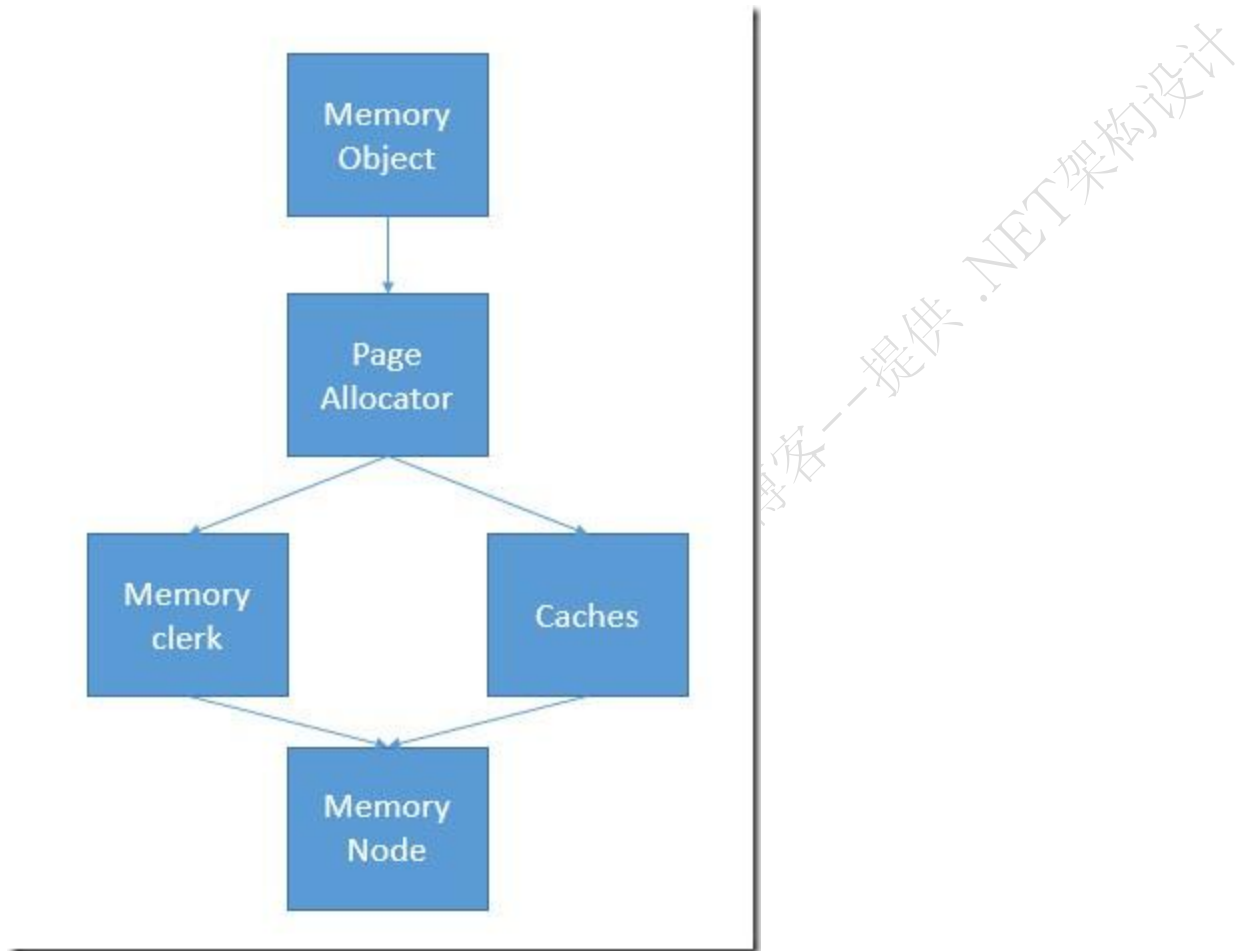


图 4. SQL Server OS 内存依赖关系

Memory Node

首先最底层的是 *Memory Node*, *Memory Node* 的作用是使得分配内存由 Windows 移交到 SQL Server OS 层面执行。每个 SQL Server 实例通常都只拥有一个 *Memory Node*, *Memory Node* 的多寡只取决于 NUMA 构架的硬件配置。我们通过 *DBCC MEMORYSTATUS* 可以看到 *Memory Node* 的一些信息,如图 5 所示。

	Memory Manager	KB
1	VM Reserved	16922008
2	VM Committed	14008136
3	AWE Allocated	0
4	Reserved Memory	1024
5	Reserved Memory In Use	0

	Memory node Id = 0	KB
1	VM Reserved	16916248
2	VM Committed	14002464
3	AWE Allocated	0
4	MultiPage Allocator	27960
5	SinglePage Alloca...	3330768

图 5. 查看 Memory Node 信息

我们可以看出，按照申请内存大小分类，可以分为两部分

1. 申请小于等于 8KB 为一个单位的内存，这些内存被用于缓存。(图 5 中的 SinglePage Allocator)
2. 申请大于 8KB 为一个单位的内存，这些内存称为 Multi-Page (或 MemToLeave) (图 5 中的 MultiPage Allocator)

对于为什么叫 MemToLeave, 被称为 MemToLeave 的原因是由于 SQL Server 虽然大部分内存被用于缓冲区，但还需要一些连续的内存用于 SQL CLR, linked server, backup buffer 等操作，32 位 SQL Server 在启动实例时会保留一部分连续的虚拟地址 (VAS) 用于进行 MultiPage Allocator。具体保留多少可以用如下公式计算：

保留地址 = ((CPU 核数量 - 4) + 256) * 0.5MB + 256MB, 通常在 384MB 左右。

Memory Clerk

让我们再来看 Memory Clerk, Memory Clerk 用于分配内存，用于将 Allocate 出去的内存进行分类，可以简单的进行如下语句, 如图 6 所示。

```
--
33 SELECT SUM(single_pages_kb) AS memory_consume, type
34 FROM sys.dm_os_memory_clerks
35 GROUP BY type
36 ORDER BY memory_consume desc
```

	memory_consume	type
1	3085456	CACHESTORE_SQLCP
2	87168	OBJECTSTORE_LOCK_MANAGER
3	60440	MEMORYCLERK_SQLGENERAL
4	23608	CACHESTORE_OBJCP
5	22000	CACHESTORE_PHDR
6	21976	MEMORYCLERK_SOSNODE
7	8272	USERSTORE_DBMETADATA
8	6736	USERSTORE_SCHEMAMGR
9	3432	MEMORYCLERK_SQLSTORENG
10	2504	MEMORYCLERK_SQLCONNECTIONPOOL
11	1992	CACHESTORE_SYSTEMROWSET
12	1656	OBJECTSTORE_SNI_PACKET
13	1088	USERSTORE_OBJPERM

图 6.按照 Memory Clerk 的类别进行分类

注意：由图 4 可以看到，Memory Clerk 只是分配内存的一部分，另一部分是数据缓存（Buffer Pool）

Buffer Pool

在开始讲述 Buffer Pool 之前，首先想讲一下虚拟内存。

在 Windows 中每个进程都有一个虚拟内存(Virtual Address Space VAS)，32 位系统是 2 的 32 次方，也就是 4G，这 4G 被 Windows 划为两部分，一部分是 Windows 使用，另一部分才是应用程序使用。虚拟内存并不是实际的物理内存，而是对于物理内存的映射，当物理内存不存在虚拟内存指向的内容时，产生缺页中断，将一部分页面置换出内存，然后将需要的部分从硬盘读到内存，关于这块，可以读我之前写的一篇文章：[浅谈操作系统对内存的管理](#)。

因此 Buffer Pool 的作用时缓冲数据页，使得未来读取数据时减少对磁盘的访问。

这个 *Buffer Pool* 这部分就是图 2 中设置最大最小服务器内存所占用的空间。这个最小值并不意味着 *SQL Server* 启动时就能占用这么多内存，而是 *SQL Server Buffer Pool* 的使用一旦超过这个值，就不会再进行释放了。

在 *DBCC MEMORYSTATUS* 其中有一部分我们可以看到 *Buffer Pool* 的信息，如图 7 所示。

	MEMORYCLERK_SQLBUFFERPOOL (Total)	KB
1	VM Reserved	16809984
2	VM Committed	13952568
3	AWE Allocated	0
4	SM Reserved	0
5	SM Committed	0
6	SinglePage Allocator	0
7	MultiPage Allocator	136

图 7. *Buffer Pool* 的相关信息

在 *SQL Server* 实例启动时，*Buffer Pool* 所保留的 VAS 地址空间取决于多个因素：包括实际的物理内存和 *SQL Server* 是 32 位或是 64 位（这个限制 32 位是 4G，还要划一半给 Windows 和减去 *MemToLeave* 空间），而对于实际上 *SQL Server* 所使用的物理内存，可以通过如下语句查看，如图 8 所示。

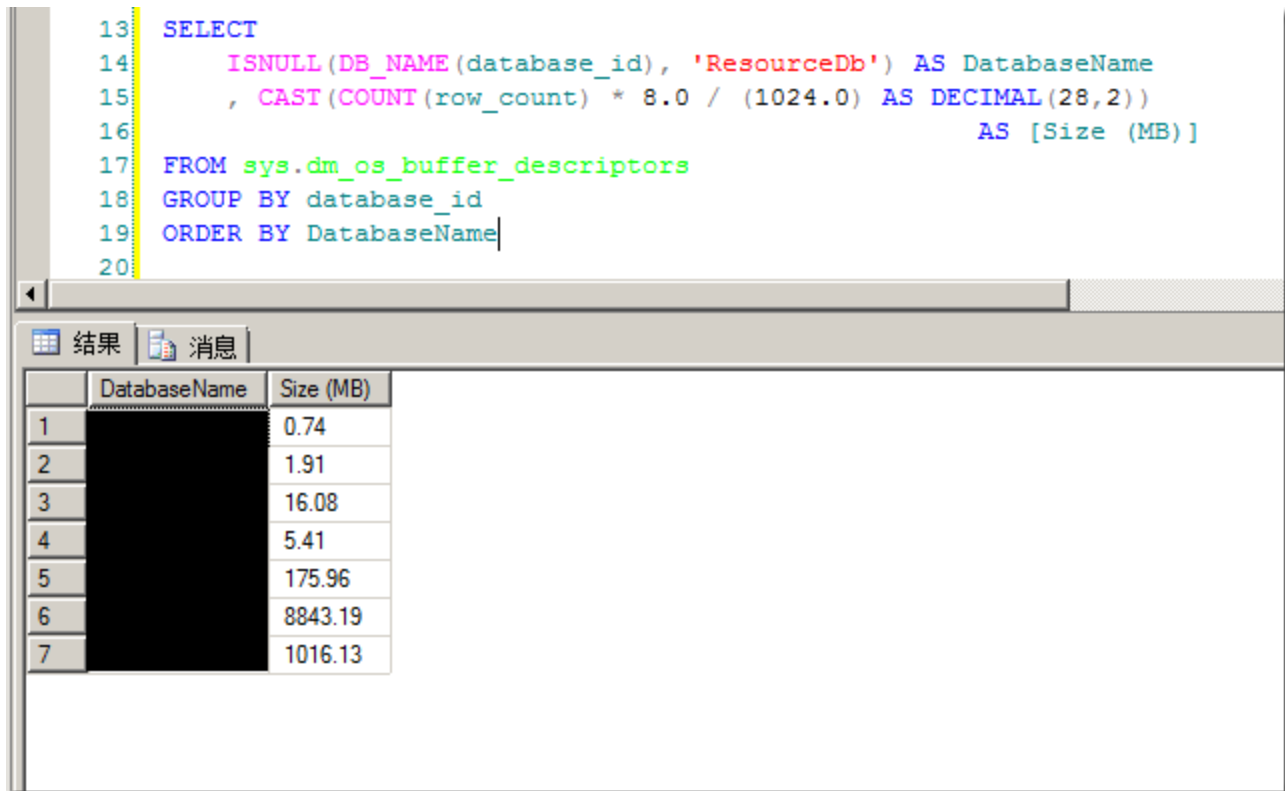


图 8. 查看 Buffer Pool 所使用物理内存

Buffer Pool 会按照需要不断的提出内存申请。Buffer Pool 如果需要, Buffer Pool 会不断消耗内存, 直到 Windows 通知 SQL Server 内存过低时, Buffer Pool 才有可能释放内存, 否则 Buffer Pool 占据了内存不会释放。

另外值得注意的一点是, Buffer Pool 所分配的页面和 SQL Server OS 页面大小是一致的, 也就是 8192 字节, 当 SQL Server 其它部分需要向" Buffer Pool"借内存时,也只能按照 8k 为单位借, 并且这部分内存存在物理内存中是不连续的, 这听上去像是 Buffer Pool 内存管理自成体系😊, 可以这么理解, 因为 Buffer Pool 不使用任何 SQL Server 的 page allocator,而直接使用 virtual 或 AWE SQLOS's 的接口。

所以 SQL Server 所占用的内存可以用这个公式粗略估算出来: buffer pool 占用的内存+从 buffer pool 借的页占得内存+multiPageAllocator 分配的非 buffer pool 内存, 如图 9 所示。



图 9.可以近似的估算出 sql server 所占的内存

Memory Object

memory object 本质上是一个堆,由 Page Allocator 进行分配,可以通过 sys.dm_os_memory_objects 这个 DMV 进行查看,这个 DMV 可以看到有一列 Page_Allocator_Address 列,这列就是 Memory Clerk 的标识,表明这个 Memory Object 是由哪个 Memory Clerk 进行分配的。

32 位 SQL Server 的内存瓶颈

由文章前面所述的一些基本原理可以看出,由于 32 位的 SQL Server 使用的是 VAS 进行地址分配,因此寻址空间被限制在 4GB,这 4GB 还要有一半分给 Windows,使得 Buffer Pool 最多只能用到 2G 的内存,这使得 32 位 SQL Server 即使有多余的物理内存,也无法使用。

解决办法之一是通过减少 Windows 默认占用的 2G 到 1G,使得 SQL Server 可以使用的内存变为 3G。这个可以通过在 Windows Server 2008 中的命令行键入 BCDEdit /set 设置 increaseuserva

选项，设置值为 3072MB，对于 Windows Server 2003 来说，需要在 boot.ini 中加上 /3gb 启动参数。

另一种办法是使用 AWE (Address Window Extension) 分配内存。AWE 通过计算机物理地址扩展 (Physical Address Extension PAE)，增加 4 位，使得 32 位的 CPU 寻址范围增加到 2 的 36 次方，也就是 64GB。基本解决了寻址范围不够的问题。

VirtualAlloc 和 AllocateUserPhysicalPages

VirtualAlloc 和 AllocateUserPhysicalPages 是 SQL Server 向 Windows 申请内存所使用的方法。在默认情况下，SQL Server 所需要的所有内存都会使用 VirtualAlloc 去 Windows 申请内存，这种申请是操作系统层面的，也就是直接对应的虚拟内存。这导致一个问题，所有通过 VirtualAlloc 分配的内存都可以在 Windows 面临内存压力时被置换到虚拟内存中。这会造成 IO 占用问题。

而使用 AllocateUserPhysicalPages 所申请的内存，直接和更底层的页表(Page Table)进行匹配，因此使用这个方法申请的内存不会被置换出内存。在 32 位 SQL Server 的情况下，通过开启 AWE 分配内存，buffer pool 中的 data cache 部分将会使用这个函数，而 MemToLeave 部分和 Buffer Pool 中的另一部分内存（主要是执行计划缓存）依然通过 VirtualAlloc 进行内存分配。

因此在开启通过 AWE 分配内存之前，SQL Server 首先需要对应的权限，否则就会在日志中报错，如图 10 所示。

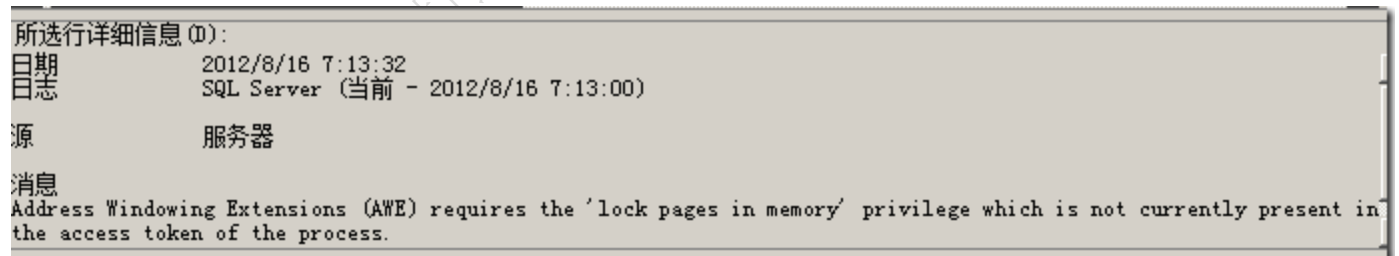


图 10. 开启 AWE 却没有开启对应权限报错

我们可以在组策略里设置启动 SQL Server 的账户拥有这个权限，如图 11 所示。

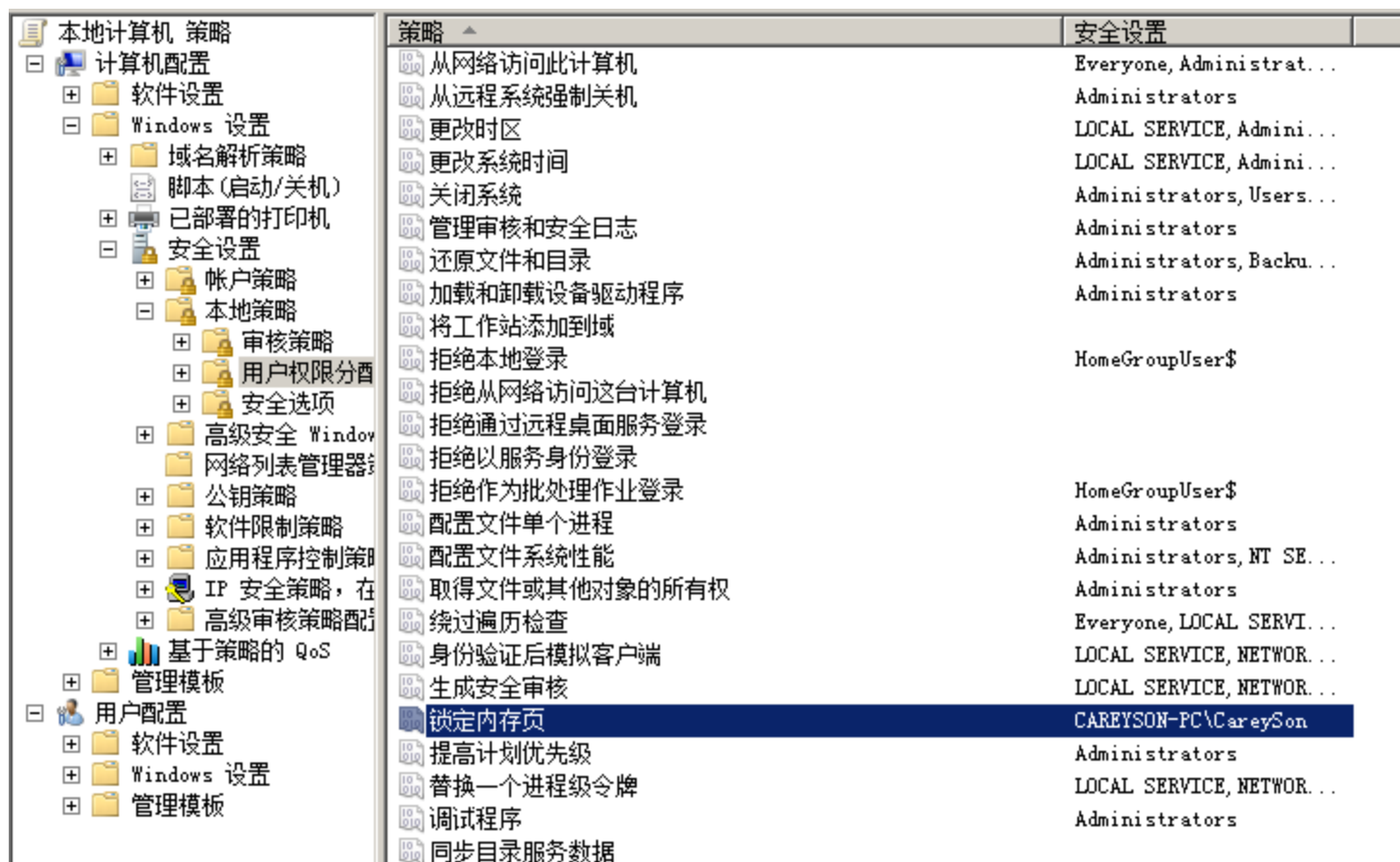


图 11. 锁定内存页(Lock Page In Memory)

64 位 SQL Server 的问题

64 位 Windows 基本已经不存在上述的内存问题，但是依然要注意，在默认情况下，64 位的 SQL Server 使用的依然是 VirtualAlloc 进行内存分配，这意味着所有分配的内存都会在 Windows 面临压力时将页置换出去，这很可能造成抖动 (Buffer Pool Churn)，这种情况也就是 SQL Server Buffer Pool 中的页不断的被交换进硬盘，造成大量的 IO 占用(可以通过 sys.dm_exec_query_memory_grants 这个 DMV 查看等待内存的查询)，因此 64 位 SQL Server 将 Buffer Pool 中的 Data Page 通过 AllocateUserPhysicalPages 来进行内存分配就能避免这个问题。与 32 位 SQL Server 不同的是，64 位 SQL Server 并不需要开启 AWE,只需开启如图 11 所示的“Lock Page In Memory”就行了。

但这又暴露出了另一个问题，因为 SQL Server 锁定了内存页，当 Windows 内存告急时，SQL Server 就不能对 Windows 的内存告急做出响应（当然了 Buffer Pool 中的非 data cache 和 MemToLeave 部分依然可以，但往往不够，因为这部分内存相比 Data Cache 消耗很小），因为 SQL Server 的特性是内存有多少用多少，因此很有可能在无法做出对 Windows 低内存的响应时造成 Windows 的不稳定甚至崩溃。因此开启了“Lock Page In Memory”之后，要限制 SQL Server Buffer Pool 的内存使用，前面图 2 中已经说了，这里就不再细说了。

还有一个问题是当 Buffer Pool 通过 AllocateUserPhysicalPages 分配内存时，我们在任务管理器中看到的 sqlservr.exe 占用的内存就仅仅包含 Buffer Pool 中非 Data Cache 部分和 MemToLeave 部分，而不包含 Data Cache 部分，因此看起来有可能造成 sqlservr.exe 只占用了几百兆内存而内存的使用是几十 G。这时我们就需要在 Perfmon.exe 中查看 SQL Server:Memory Manager\Total Server Memory 计数器去找到 SQL Server 真实占用的内存。

总结

本文讲述了 SQL Server 对内存管理的基本原理和 SQL Server 对内存使用所分的部分，对于 SQL Server 性能调优来说，理解内存的使用是非常关键的一部分，很多 IO 问题都有可能是内存所引起的。