SYBASE®
An SAP Company

性能和调优系列：物理数据库调优

Adaptive Server® Enterprise

15.7

文档 ID：DC01070-01-1570-01

最后修订日期：2011 年 9 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568。

目录

第 1 章

控制物理数据放置

本章介绍如何通过控制表和索引的位置来提高性能。

要最大限度地优化物理数据库，应了解逻辑设备和物理设备之间的区别：

- 物理磁盘或物理设备是用于存储数据的硬件。
- 数据库设备或逻辑设备可能是整个物理磁盘，也可能是物理磁盘的一部分，为供 Adaptive Server® 使用，该磁盘已初始化（使用 disk init 命令）。数据库设备可以是操作系统文件、整个磁盘或一个磁盘分区。

  请参见 《安装指南》和 《配置指南》，以了解您的平台中对使用磁盘和文件的特定操作系统限制的信息。
- 段是一个供数据库使用的数据库设备的命名集合。构成段的数据库设备可以位于单独的物理设备上。
- 分区是表的子集。分区是可以独立管理的数据库对象。可以对分区后的表进行拆分，以使多个任务可以同时访问表。可以将分区置于特定段上。如果各个分区位于不同的段上，且每个段都有自己的数据库设备，那么访问这些表的查询便会受益于改善的并行度。请参见《参考手册：命令》和 《Transact-SQL 用户指南》中的 create table，了解有关创建和使用分区的详细信息。

使用 sp_helpdevice、 sp_helpsegment 和 sp_helpartition 可获取有关设备、段和分区的详细信息。

通过控制对象放置来提高性能

Adaptive Server 允许您控制数据库、表和索引在物理存储设备内的放置位置，以便可以通过平均分配跨多个设备和控制器的磁盘读写来提高性能。例如，可以：

- 将数据库的数据段放置在一个或多个特定设备上，以便在单独的物理设备上存储数据库日志，从而避免对日志的读写干扰数据访问。
- 跨多个设备分布大的且频繁使用的表。
- 将特定表或非聚簇索引放置在特定设备上。例如，可将表放置在一个跨多个设备的段上，将其非聚簇索引放置在一个单独段上。
- 将表的 text 和 image 页链放在单独设备上，与该表分开。该表存储指向单独数据库结构中的实际数据值的指针，因此每次访问 text 或 image 列都至少需要执行两次 I/O 操作。
- 在单独物理磁盘的分区内平均分配表，以使并行查询的性能达到最佳，并提高 insert 和 update 性能。

对于执行大量磁盘 I/O 操作的多用户系统和多 CPU 系统，要特别注意物理和逻辑设备问题以及跨设备的 I/O 分配：

- 在逻辑和物理设备内均衡划分对象。
- 使用充足的物理设备，包括磁盘控制器，以确保物理带宽。
- 使用更多逻辑设备，以确保最大限度地减少内部 I/O 队列的争用。
- 确定并使用允许进行并行扫描并达到查询性能目标的大量分区。

发现对象放置问题

在以下情况下，若对象放置位置更适当，系统将可以从中受益：

- 单用户性能令人满意，但在 Adaptive Server 执行多进程时响应时间显著增加。
- 访问镜像磁盘所用时间是访问未镜像磁盘的两倍。
- 频繁访问的对象 （“热对象”）降低了使用包含这些对象的表的查询的性能。
- 维护活动的时间较长。
- 如果 tempdb 与其它数据库共享磁盘空间，其性能将受到影响。多数系统过程和应用程序都将 tempdb 用作其工作空间，如果 tempdb 与其它数据库共享同一个磁盘，其性能将受影响。

- 频繁使用的表的 insert 性能较差。
- 由于分区或设备上数据页的不平衡，并行运行的查询执行效果很差，或者由于极不平衡，它们以串行方式运行。

如果您遇到磁盘争用引发的问题和其它有关对象放置的问题，应检查并更正以下问题：

- 随机访问 （数据和索引的 I/O）和串行访问 （日志 I/O）进程使用相同的磁盘。
- 数据库进程和操作系统进程使用相同的磁盘。
- 串行磁盘镜像。
- 在存储数据的同一磁盘上记录数据库日志或进行审计。

更改数据放置位置时使用 *sp_sysmon*

使用 sp_sysmon 可确定跨物理设备放置数据是否会引发性能问题。在调优期间检查整个 sp_sysmon 输出，检验这些更改对所有性能类别有哪些影响。

应特别注意与以下各项有关的输出：

- I/O 设备争用
- 所有页锁定堆表
- 堆上的最后页锁
- 磁盘 I/O 管理

请参见 《使用 sp_sysmon 监控 Adaptive Server》。

提高 I/O 性能

要在 Adaptive Server 中提高 I/O 性能，请尝试以下方法：

- 跨磁盘分布数据以避免 I/O 争用
- 从数据库 I/O 中隔离全服务器范围 I/O
- 将频繁更新的数据库的数据存储和日志存储分开
- 将随机磁盘 I/O 和顺序磁盘 I/O 分开

- 将设备镜像到单独的物理磁盘
- 使用分区在设备之间分配表数据

跨磁盘分布数据以避免 I/O 争用

跨多个磁盘和多个磁盘控制器分布数据存储可避免出现瓶颈。

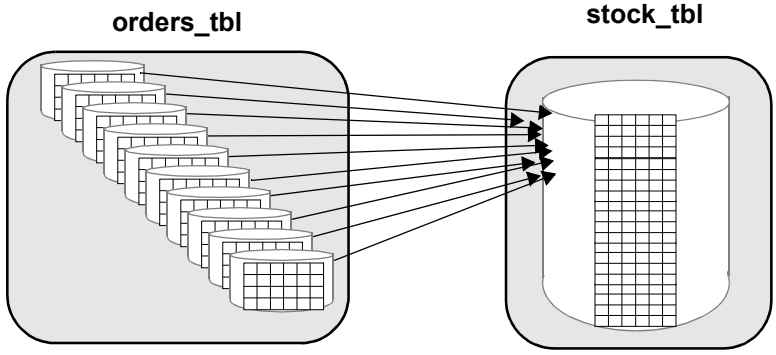
- 将有重要性能要求的数据库放置到单独的设备上。如果可能，还应与其它数据库使用不同的控制器。必要时，对关键表使用段，对并行查询使用分区。
- 将频繁使用和连接的表放置到单独的磁盘上。
- 使用段，将表和索引放置到它们各自的磁盘上。

避免并行连接查询的物理争用

图 1-1 说明了两个分区表 orders_tbl 和 stock_tbl 的连接。其中提供了 10 个可用工作进程；orders_tbl 有 10 个分区，分布于 10 个不同的物理设备上，该分区表在此连接中是外部表；stock_tbl 是未分区的表。这些工作进程在 orders_tbl 上存在访问争用问题，但每个工作进程必须扫描 stock_tbl。如果整个表不装入高速缓存，则可能出现物理 I/O 争用。情况最坏时，10 个工作进程都会尝试访问 stock_tbl 所驻留的物理设备。通过创建包含整个 stock_tbl 表的指定高速缓存，可以避免物理 I/O 争用。

减少或消除物理 I/O 争用的另一种方法是对 orders_tbl 和 stock_tbl 进行分区，并将这些分区分配到不同物理设备上。

图 1-1：连接位于不同物理设备上的表



从数据库 I/O 中隔离全服务器范围 I/O

将有大量 I/O 要求的系统数据库 （例如 tempdb 和 sybsecurity）与应用程序数据库分开放置到不同的物理磁盘和控制器上。

tempdb

这是一个频繁使用的数据库，它会影响服务器上的所有进程，多数系统过程都使用此数据库。它会自动安装到主设备上。如果需要更多空间，可以将 tempdb 扩展到其它设备上。如果希望 tempdb 处于非常活跃的状态，可将其放置到不用于存储其它重要数据库活动的最快速的可用磁盘上。

在某些 UNIX 系统上，操作系统文件的 I/O 要明显快于原始设备的 I/O。由于在关机后将重建而不是恢复 tempdb；因此，将 tempdb 迁移到操作系统文件而不是原始设备上可能会提高性能。可在您自己的系统上对此进行测试。

请参见第 6 章 “临时数据库”，以了解更多 tempdb 的放置建议。

sybsecurity

启用后，审计系统将对 sybsecurity 数据库中的 sysaudits 表执行频繁的 I/O 操作。如果应用程序执行大量审计，应将 sybsecurity 放置到对响应时间要求不高的表所使用的磁盘上。最好将 sybsecurity 放置到其自己的设备上。

使用阈值管理器可监控可用空间，以避免审计数据库填满时发生挂起用户事务的情况。请参见 《系统管理指南，卷 2》中的第 16 章 “使用阈值管理可用空间”，以了解有关确定适当阈值的信息。

将事务日志保存在单独的磁盘上

将事务日志放置到单独段上可以防止这些日志与其它对象争用磁盘空间。将日志放置到单独的物理磁盘上：

- 通过减少 I/O 争用提高性能
- 确保在数据设备上的硬盘发生故障时能够完全恢复
- 加快恢复速度，因为同时异步预取请求可以在日志设备和数据设备上预先读取，而不会发生争用

在可以将事务日志放置到数据所在的设备上之前， create database 和 alter database 都需要使用 with override。

系统更新活动频繁时，其上的日志设备会出现大量 I/O 操作。 Adaptive Server 会在提交事务时将日志页写入磁盘，并可能需要将日志页读取到内存，以将延迟更新替换为延迟操作。

当日志和数据位于同一数据库设备中时，为存储日志页而分配的扩充不是连续的；日志扩充和数据扩充将混合在一起。如果日志位于自己的设备上， Adaptive Server 会按顺序分配扩充，从而减少磁头移动和搜索，并保持较高的 I/O 比率。

Adaptive Server 会将每位用户的日志记录缓冲到用户日志高速缓存中，从而减少为写入日志页而争用内存的现象。如果日志和数据位于相同的设备上，用户日志高速缓存缓冲将被禁用，这会严重影响 SMP 系统的性能。

请参见 《系统管理指南，卷 1》中的第 6 章 “磁盘资源问题概述”。

将设备镜像到单独的磁盘

磁盘镜像是一项具有高可用性的功能，它允许 Adaptive Server 复制整个数据库设备的内容。

请参见 《系统管理指南，卷 2》中的第 2 章 “磁盘镜像”。

若镜像数据，应将镜像放置到单独的物理磁盘上，而不应放置在其镜像的设备上，以尽量减少镜像的性能影响。磁盘硬件故障通常会导致整个物理磁盘丢失或不可用

如果不使用镜像，或使用操作系统镜像，则通过将 disable disk mirroring 配置参数设置为 1，可使性能稍有提高。

由于会在两个磁盘上串行或并行执行写入操作，因此镜像会延长完成磁盘写入所需的时间。磁盘镜像不影响读取数据所需的时间。

镜像设备使用以下两种磁盘写入模式之一：

- 非串行模式 — 完成写入需要的时间可能长于非镜像写入的时间。在非串行模式下，两项写入操作同时开始，Adaptive Server 会等待它们全部完成。完成非串行写入的时间以两次 I/O 中时间较长的为准。
- 串行模式 — 增加写入数据所需的时间，可能比非串行模式时间更久。Adaptive Server 会启动第一项写入并等待其完成，然后再进行第二项写入。所需时间为两次 I/O 时间之和。

使用串行模式

虽然其性能会受到影响，但由于串行模式可以防止写入期间发生故障，因此此模式为缺省模式。

由于在开始第二项写入前，串行模式会等待第一项写入完成，因此任何故障都不会对两个磁盘造成影响。使用非串行模式可以提高性能，但如果发生影响两项写入的故障，您将面临数据丢失风险。

警告！如果必须确保镜像数据库系统绝对可靠，可使用非串行模式。

使用段

段是一个指向一台或多台逻辑设备的标签。使用段时可通过以下方式提高吞吐量：

- 跨磁盘拆分大表，包括为提高并行查询性能而被分区的表
- 将表及其非聚簇索引跨磁盘分开
- 将表分区和索引跨磁盘分开
- 将文本和图像页链与表分开放置在不同的磁盘上，该磁盘存储指向文本值的指针

此外，段还可以用于控制空间使用情况：

- 表或分区不能大于其段分配。可以使用段限制表或分区大小。
- 其它段上的表或分区不能使用分配给另一段上的对象的空间。
- 阈值管理器可监控空间使用情况。

在段上创建对象

每个数据库最多可以使用 32 个段，其中包括创建数据库时由系统（system、 log segment 和 default）创建的 3 个段。

表和索引存储在段上。如果在未指定段的情况下执行 create table 或 create index，对象将存储在数据库的 default 段中。如果在其中任一命令中指定段，将在该段上创建对象。您可以使用 sp_placeobject 系统过程来安排要在指定段上进行的所有后续空间分配，以使表可以跨越多个段。

系统管理员必须使用 disk init 初始化设备，并将该设备分配给数据库。此外，数据库所有者也可以使用 create database 或 alter database 执行此操作。

数据库能使用这些设备后，数据库所有者或对象所有者即可创建段并将对象放置到设备上。

创建用户定义的段时，可使用 create table 或 create index 命令在该段上放置表、索引和分区：

```
create table tableA(...) on seg1
create nonclustered index myix on tableB(...)
      on seg2
```

以下示例创建表 fictionsales，该表根据 date 列中的值按范围分区：

```
create table fictionsales
(store_id int not null,
order_num int not null,
date datetime not null)
partition by range (date)
(q1 values <= ("3/31/2005") on seg1,
q2 values <= ("6/30/2005") on seg2,
q3 values <= ("9/30/2005") on seg3,
q4 values <= ("12/31/2005") on seg4)
```

通过控制关键表的位置，可以对这些表和索引进行安排，使它们跨磁盘分布。

将表和索引分开

可使用段将表放置到一组磁盘上，将非聚簇索引放置到另一组磁盘上。不能将聚簇索引与其数据页分开放置于不同的段上。使用 on *segment_name* 子句创建聚簇索引时，可将整个表移动到指定段，并在该段中建立聚簇索引树。

将非聚簇索引放置在单独的段上，可以提高性能。

跨设备拆分大表

由于段可以跨多个设备，因此可以使用它们在一个或多个磁盘之间分布数据。这有助于平衡繁忙的大型表的 I/O 负载。对于并行查询，创建跨多个设备的段对于基于分区进行扫描期间的 I/O 并行度非常重要。

请参见 《系统管理指南，卷 2》中的第 8 章 “创建和使用段”。

将文本存储移动到单独的设备

如果表包含 text、 image 或 Java 行外数据类型，则此表本身会存储指向数据值的指针。实际数据存储在称为“大对象链” (LOB) 的单独页链接列表中。

写入或读取 LOB 值需要至少访问两次磁盘，一次进行读取或写入指针，另一次进行后续读取或写入数据。如果应用程序频繁读取或写入 LOB 值，可通过将 LOB 链放置到单独的物理设备上来提高性能。将 LOB 链隔离到不忙于处理与其它应用程序相关的表或索引访问的磁盘上。

创建包含 LOB 列的表时，Adaptive Server 会在 sysindexes 和 syspartitions 中为存储 LOB 数据的对象创建一行。 name 列中的值为前缀为 “t”的表名称；indid 始终为 255。如果单个表中有多个 LOB 列，则只有一个用于存储数据的对象。缺省情况下，此对象与表放置在相同的段上。

使用 sp_placeobject 可将对 LOB 列的所有后续分配移动到单独段上。

为提高性能对表进行分区

对表进行分区可以提高几类进程的性能。

- 分区可允许并行查询处理访问表的每个分区。基于分区的扫描中的每个工作进程读取一个单独的分区。
- 分区让您可以与批量复制同时装载表。

  有关并行 bcp 的详细信息，请参见 《实用程序》手册。
- 分区让您可以跨多个数据库设备分配表的 I/O。
- 语义分区 （范围、散列和列表分区表）可以缩短响应时间，因为查询处理器减少了部分分区。
- 分区为堆表提供多个插入点。

选择哪些表进行分区以及选择哪种分区类型取决于出现的性能问题和表查询的性能目标。

请参见 《Transact-SQL 用户指南》一书的第 10 章 “对表和索引进行分区”，以了解有关使用和创建分区的详细信息及示例。

如何Adaptive Server在设备上分配分区

在 15.0 以前的版本中，当您在已映射到多个数据库设备的段上创建多个分区时，Adaptive Server 会在分区和设备之间自动保持关联。但 Adaptive Server 15.0 及以后版本不再提供此功能；将在第一个设备上创建所有分区。要在分区和设备之间实现关联，可采用以下方法：

1 为特定设备创建段。

2 在该段上显式放置分区。

您最多可以创建 29 个用户段，且必须使用 Adaptive Server 15.0 及以后版本中的 alter table 语法来创建这些段，这是因为之前版本中的语法 (alter table t partition 20) 不支持在段上显式放置分区。

让段中的分区数与设备数相匹配可使并行查询获得最佳 I/O 性能。

可对使用 text、 image 或 Java 行外数据类型的表进行分区。但是，列本身并不被分区 — 它们保存在一个单独的页链上。

RAID 设备和分区表

已条带化的独立磁盘冗余阵列 (RAID) 设备可以包含多个物理磁盘，但 Adaptive Server 将此类设备视为单个逻辑设备。您可以在此单个逻辑设备上使用多个分区，并获得良好的并行查询性能。

要确定应用混合的最佳分区数，对分条集中的每个设备均应以一个分区开始。使用操作系统实用程序 （在 UNIX 上为 vmstat、 sar 和 iostat；在 Windows 上为 “性能监视器”）可检查利用率和滞后时间。

要检查最大设备吞吐量，应使用 select count(*)，以便使用 index *table_name* 子句来强制执行表扫描（如果存在非聚簇索引）。这一命令对 CPU 工作的要求最少，而且对其它资源产生的争用也非常少。

分区表的空间计划

在制定分区表计划时，应考虑如何：

- 在实现基于分区的扫描性能和 I/O 并行度的磁盘中保持负载平衡
- 维护聚簇索引，索引所需空间约为表所占空间的 120%，用于删除并重建索引或运行 reorg rebuild

空间计划决策取决于：

- 用于存储表的磁盘资源的可用性
- 应用程序混合与传入数据 （针对语义分区表）的性质

估计维护分区表所需的频率：某些应用程序需要频繁地重建索引以保持平衡，而其它应用程序则需要少量维护。

对于为保证性能而必须频繁进行负载平衡的应用程序来说，应留有空间来重建聚簇索引或运行 reorg rebuild，以提供最快捷和简便的结果。但是，由于创建聚簇索引需要复制数据页，因此段上的可用空间必须等于表所占空间的约 120%。

请参见第 104 页的 “确定维护活动的可用空间”。

以下是对只读、部分读和随机数据修改的说明，简单介绍了对象放置和维护分区表所涉及的问题。

请参见 《Transact-SQL 用户指南》的第 10 章 “对表和索引进行分区”，以了解有关维护期间所需执行的特定任务的信息。

只读表

只读表或极少更改的表可完全填满段上的可用空间，并且不需要进行维护。如果表不需要聚簇索引，可使用并行批量复制 （并行 bcp）来完全填满段上的空间。

如果需要聚簇索引，表的数据页最多可占用段中 80% 的空间。聚簇索引树需要表所占空间的 20%。

此空间要求因键的长度而异。最初，将数据装载到表中并创建聚簇索引需要执行几个步骤，但执行完这些步骤后，所需维护则是最少的。

部分读表

以上只读表的原则同样适用于很少执行插入操作的部分读表。唯一的例外是：

- 如果对表执行插入操作，且聚簇索引键未在各分区间平均分配新空间，则某些分区中的磁盘可能被填满，并且将为其它物理磁盘分配新扩充。这一进程称为扩充挪用。

  在跨许多磁盘分布的大表中，对其它设备的一小部分分配不是问题。通过使用 sp_helpsegment 检查无可用空间的设备，或使用 sp_helpartition 检查页数不成比例的分区，可检测扩充挪用。

  如果分区大小的不平衡导致并行查询的响应时间延长或性能下降，可能需要使用 《Transact-SQL 用户指南》的第 10 章 “对表和索引进行分区”中所述的方法之一平衡分配。

- 如果该表为一个堆表 （循环分区表），则堆表插入操作的随机性应使分区保持平衡。

  对于大批量复制应谨慎操作。但是，如果表是语义分区表，则应考虑使用 alter table... partition by 来更改分区条件，以实现适当的负载平衡。

  可使用并行批量复制（并行 bcp）将行发送到页数最少的分区中，以平衡各分区中的数据。请参见《实用程序指南》的第 4 章“使用 bcp 从 Adaptive Server 传出数据或向 Adaptive Server 传送数据”。

以随机形式进行数据修改的表

对于需要经历许多插入、更新和删除操作的、带有聚簇索引的表来说，随着时间的推移，其数据页的满度会达到大约 70% 到 75%。这样可引起几种方式的性能降低：

- 必须读取更多的页才能访问到指定的行数，这样会需要更多的 I/O，浪费数据高速缓存空间。
- 在所有页锁定表上，由于页链跨多个扩充和分配单元，因此大 I/O 和异步预取的性能将会下降。

  读取所有页后，大 I/O 所占缓冲区可能从高速缓存中被刷新。沿页链前进时，跨分配单元的跳转会减小异步预取的预设空间大小。

  对于使用仅数据锁定的表，由于转移页跨多个扩充和分配单元，因此大 I/O 和异步预取的性能将会下降。

当分段开始降低应用程序性能时，应进行维护，同时请切记，删除和重建聚簇索引需要的空间为表所占空间的 120%。

如果无可用空间，维护工作将变得更为复杂且持续时间更长。一个最好的、通常也最简易的解决方法是，为创建索引增加足够的磁盘容量。

设备填满时添加磁盘

当分区填满时，仅添加磁盘和重建索引可能无法解决负载平衡问题。如果容纳分区的物理设备完全填满，重建索引的数据复制阶段将无法将数据复制到该物理设备。

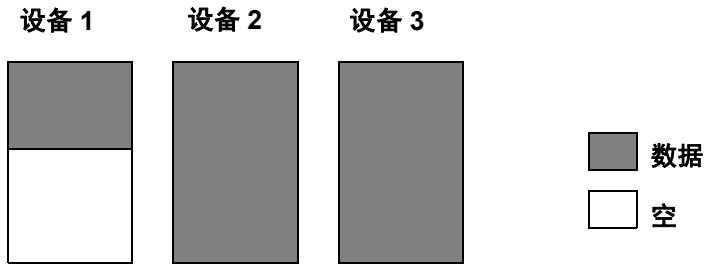
如果物理设备几乎被完全填满，则重建聚簇索引并不总能成功地实现良好的负载平衡。

设备填满时添加磁盘

物理设备完全填满时，创建聚簇索引会在其它物理设备中的一台设备上创建两个分区。

如图 1-2 所示，**设备 2** 和**设备 3** 已完全填满。

图 1-2：3 台设备上具有 3 个分区的表

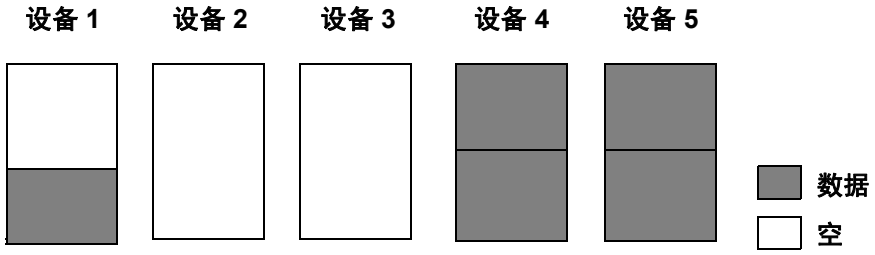


在上述示例中，添加两台设备、将表重新分为五个分区、删除并重建聚簇索引会产生以下结果：

设备 1	一个分区，满度约为 40%。
设备 2 和 3	空。当 create index 启动后，这些设备上已无可用空间，因此无法在设备上创建用于复制索引的分区。
设备 4 和 5	每个设备有两个分区，每个分区满度为 100%。

图 1-3 显示了这些结果。

图 1-3：创建索引后的设备和分区

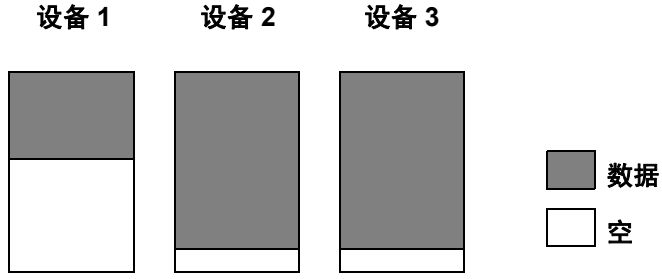


设备完全填满后唯一的解决方法是，将数据批量复制出来、截断表，然后将数据重新复制到表中。

设备接近填满时添加磁盘

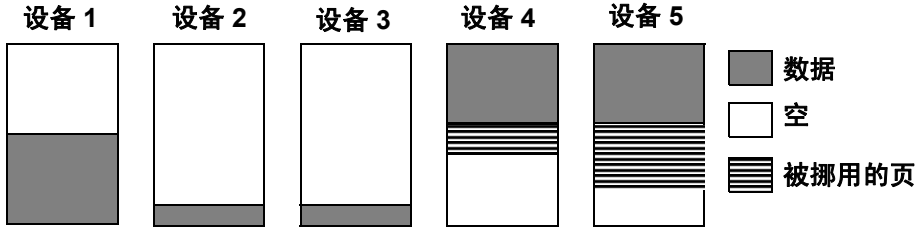
如果设备几乎填满，重建聚簇索引不会在各设备间平衡数据。相反，几乎填满的设备将存储分区的一小部分，而分区的其它空间分配会挪用其它设备上的扩充。图 1-4 显示包含几乎填满数据的设备的表。

图 1-4：分区几乎完全填满设备



添加设备并重建聚簇索引后，结果可能与图 1-5 中所示的结果相似。

图 1-5：扩充挪用和不平衡数据分配



如果**设备 2** 和**设备 3** 上的分区使用较少可用空间，它们将开始从**设备 4** 和**设备 5** 挪用扩充。

再次重建索引可能会使分配更加平衡。但是，如果挪用扩充使其中一台设备几乎填满，则再次重建索引并不能解决问题。

应对这种不平衡的最有效办法是，使用批量复制将数据复制出来，然后再将它们复制回去。

要避免这些情况，必须监控设备空间的使用情况，并提早添加空间。

维护问题和分区表

分区表的维护活动要求取决于对表执行的更新的频率和类型。

需要进行极少量维护工作的分区表有：

- 只读表或很少更新的表。对于较少更新的表，只需定期查看是否需要进行平衡。
- 插入操作在各分区间分配合理的表。对分区堆表的随机插入操作和由于聚簇索引键（在不同分区上放置行）而平均分配的插入操作，不会引发页的倾斜分配情况。

  如果数据修改导致空间分段和数据页被部分填满，则可能需要重建聚簇索引。
- 通过批量复制方式执行插入操作的堆表。可利用批量复制将新数据引导到特定的分区上，以维持负载平衡。

需要频繁监控和维护的分区表包括带聚簇索引的表，这些聚簇索引能将新行引导到分区子集。升序键索引可能需要更频繁的维护。

分区表的常规维护检查

对分区表的例行监控工作除例行的数据库一致性检查外，还应包含以下两类检查：

- 使用 sp_helpartition 可检查分区的平衡情况。如果某些分区明显大于或小于平均值，应重建聚簇索引以重新分配数据。
- 使用 sp_helpsegment 可检查基础磁盘上空间的平衡情况。
- 如果重新创建聚簇索引来重新分配数据，以保证并行查询的性能，应检查是否存在满度接近 50% 的设备。在设备过满前添加空间，能避免出现本章先前提到的复杂过程。
- 可使用 sp_helpsegment 检查各设备上作为可用页提供的可用空间，或使用 sp_helpdb 检查可用千字节数。

您可能需要在分区表上重建聚簇索引，因为：

- 索引键往往将插入操作指派给分区的一个子集。
- 删除活动往往会从分区中的子集中删除数据，引起 I/O 不平衡和基于分区的扫描不平衡。
- 表需经历许多插入、更新和删除活动，从而使许多数据页部分填满。这样会造成磁盘和高速缓存空间的浪费，导致许多查询读取的页数增加，从而增加了 I/O 操作。

第 2 章

数据存储

本章介绍 Adaptive Server® 如何在页中存储数据行，以及如何在无索引的情况下将这些页用于 select 和数据修改语句。本章是了解如何通过创建索引、优化查询和解决对象存储问题来提高 Adaptive Server 性能的基础。

查询优化

通过对访问数据所需的物理 I/O 开销，以及在数据高速缓存中必须读取各页的次数进行估算， Adaptive Server 优化程序可尝试针对查询中各表的数据找到最有效的访问路径。

在大多数数据库应用程序中，数据库中有很多表，每个表有一个或多个索引。依据是否已创建索引以及所创建的索引类型，优化程序的访问方法选项包括：

- 表扫描 — 读取所有表的数据页，有时可读取数以百计或数以千计的页。
- 索引访问 — 用索引只查找所需的数据页，有时总共才读取三或四页。
- 索引覆盖 — 仅使用索引返回数据，而不读取实际的数据行，只需读取表扫描所需的少数几页。

对表使用适当索引可以使多数查询只需读取最少数量的页即可访问所需数据。

查询处理和页读取

查询的大部分执行时间用于从磁盘读取数据页。因此，性能提高多数情况下是源于每次查询所需的磁盘读取数量的减少。

在查询执行表扫描时，由于没有可用索引能够帮助其检索数据，因此 Adaptive Server 会读取表中的所有页。磁盘读取需要一定时间，所以该查询的响应时间可能较长。那些要引起开销昂贵的表扫描的查询也影响服务器上其它查询的性能。

由于表扫描使用系统资源 （如 CPU 时间、磁盘 I/O 和网络容量），因此它们会增加其他用户等待响应的时间。

表扫描会为给定查询使用大量磁盘读取 (I/O)。熟悉访问方法、调优工具、表的大小和结构、应用程序中的查询后，如果存在索引，就应该能够估算出给定连接或选择操作将执行的 I/O 操作的数量。

如果您知道表中有哪些索引列以及表和索引的大小，则通常可以查看查询并预测查询运行情况。对于同一表上的不同查询，可能得出这些结论：

- 此查询返回单行或与 where 子句条件匹配的少量行。

  where 子句中的条件已编制索引；应对此索引执行 2-4 个 I/O，另外再执行一个 I/O 来读取正确的数据页。

- 选择列表中的所有列以及此查询的 where 子句包含在非聚簇索引中。此查询可能将在索引的叶级执行扫描，大约扫描 600 页。

  向选择列表中添加未建索引的列会强制查询扫描表，这需要读取磁盘 5000 次。

- 此查询没有有用的索引；从而将执行表扫描，至少需要读取 5000 次磁盘。

本章介绍如何储存表以及在不使用索引时如何访问数据行。

Performance and Tuning Series: Locking and Concurrency Control （《性能和调优系列：锁定和并发控制》）的第 5 章 “Indexes”（索引）描述了索引的访问方法。第 3 章 “设置空间管理属性”和第 4 章 “表和索引大小”介绍如何确定用于查询、表和索引的大小及查询执行的 I/O 次数的访问方法。这些章节为理解优化程序如何为您的查询建立访问数据的开销模型奠定了基础。

Adaptive Server 页

以下类型的页用于存储数据库对象：

- 数据页 — 存储表的数据行。
- 索引页 — 存储索引所有级的索引行。
- 大对象 (LOB) 页 — 存储 text 和 image 列的数据以及 Java 行外列的数据。

Adaptive Server 12.5 及以后版本不使用 buildmaster 二进制构建主设备。Sybase® 已将 buildmaster 功能整合到 dataserver 二进制中。

您可以使用 dataserver 命令创建逻辑页大小为 2K、4K、8K 或 16K 的主设备和数据库。逻辑页越大，所允许创建的行就越大，这会提高性能，因为 Adaptive Server 在每次读取一页时可访问到更多的数据。例如，16K 页容纳的数据量是 2K 页的 8 倍， 8K 页容纳的数据量是 2K 页的 4 倍，对于所有逻辑页大小，可以依此类推。

逻辑页大小是一种服务器范围的设置；在同一台服务器内，数据库的逻辑页大小必须相同。所有表将自动调整为适当大小，以使行大小不超过服务器的当前页大小。也就是说，一行不能占据多页。

请参见 《实用程序指南》，以了解有关使用 dataserver 命令构建主设备的具体信息。

可能需要使用 Adaptive Server 来为单个查询、 DML 操作或命令处理大量数据。例如，如果使用包含 char(2000) 列的 DOL 锁定表， Adaptive Server 必须分配内存以便在扫描表时执行列复制。在运行查询或命令过程中，增加内存请求意味着可能要减少吞吐量。

Adaptive Server 逻辑页的大小决定服务器的空间分配。每个分配页、对象分配映射 (OAM) 页、数据页、索引页、文本页等都建立在逻辑页上。例如，如果 Adaptive Server 的逻辑页大小为 8K，则这些类型页的每一页大小均为 8K。所有这些页都占用由逻辑页大小所指定的整个大小。对于较大的逻辑页 （如 8K）， OAM 页拥有的 OAM 条目要远多于较小页 (2K) 拥有的 OAM 条目。

页头和页大小

所有页都有一个页头，用来存储相关信息，例如该页所属的分区 ID，以及用于管理页空间的其它信息。表 2-1 显示配置为 2K 页的服务器中数据和索引页的开销及可用空间的字节数。

表 2-1：数据与索引页开销及用户数据空间

锁定方案	开销	用户数据字节数
所有页	32	2016
仅数据	46	2002

页的其它部分可用于存储数据和索引行。

有关如何存储 text、 image 和 Java 列的信息，请参见第 20 页的 “大对象 (LOB) 页”。

数据页和索引页

DOL 锁定表上的数据页和索引页包含一个行偏移表，用于存储指向页中每行起始字节的指针。每个指针占用 2 个字节。

向某页插入数据和索引行时，将以页头之后处为起始插入点，然后连续填充该页。对于 DOL 锁定表上的所有表和索引，行偏移表起始于页上的最后一个字节处，并向上扩展。

每行存储的信息由实际列数据以及行号、行中可变长度列和空列数等信息组成。所有页锁定表的索引页没有行偏移表。

除 text、 image 和 Java 行外列外，行不能跨越页边界。每个数据行至少有 4 个字节的开销；含有可变长度数据的行还要有额外的开销。

有关数据和索引行大小及开销的详细信息，请参见第 4 章 “表和索引大小”。

大对象 (LOB) 页

表的 text、image 和 Java 行外列以单独的数据结构存储，由一组页组成。这些列称为大对象或 LOB 列。包含 text 或 image 列的每个表都具有其中一种结构。即使表有多个 LOB 列，它仍只拥有这些单独数据结构之一。

表本身存储一个指向行中数值第一页的 16 字节指针。该值的其它页通过下一个和前一个指针链接。所有值都存储在各自的单独页链中。第一页存储文本值中的字节数。一个值页链的最后一页以空的下一页指针终止。

读取或写入 LOB 值时至少需要进行两次页读取或写入：

- 一次读/写指针
- 一次读/写文本在文本对象中的实际位置

每个 LOB 页最多存储 1800 个字节。每个非空值至少使用一个整页。

LOB 结构单独列在 sysindexes 中。LOB 结构的 ID 与表 ID 相同。索引 ID 列为 indid，其始终为 255， name 为表名，以字母 “t”作为前缀。

扩充

Adaptive Server 页始终分配给表、索引或 LOB 结构。由 8 页组成的块称为一个**扩充**。扩充的大小取决于服务器所用的页大小。2K 服务器的扩充大小为 16K，8K 服务器为 64K，依次类推。表或索引可占用的最少空间量为 1 个扩充或 8 页。只有在扩充中所有页都为空时，扩充才被释放。

除检查空间使用情况报告时之外，用户在其它时间都可以了解扩充在 Adaptive Server 中的使用情况。例如，来自 sp_spaceused 的报告显示分配的空间 （reserved 列），以及数据和索引使用的空间。 unused 列显示扩充中已分配给对象、但尚未用于存储数据的空间量。

```
sp_spaceused titles
name   rowtotal reserved data    index_size unused
------ -------- -------- ------- ---------- ------
titles 5000     1392 KB 1250 KB 94 KB      48 KB
```

在此报告中， titles 表及其索引在各扩充中保留了 1392KB，其中包括未分配的 48KB （24 个数据页）。

注释　Adaptive Server 通过填满目标分配页中的现有分配扩充 （即使这些扩充已分配给其它分区也是如此）来避免浪费额外空间。其实际结果是，仅当目标分配页中没有可用扩充时才分配扩充

用于管理空间分配的页

除数据、索引和用于存储数据的 LOB 页外，Adaptive Server 还使用其它类型的页管理存储、跟踪空间分配和定位数据库对象。 sysindexes 表还存储在访问数据时使用的指针。

管理空间分配和 sysindexes 指针的页用于：

- 加速查找数据库中对象的进程。
- 加速为对象分配空间和释放空间的进程。
- 为 Adaptive Server 提供了在邻近对象已用空间处再为其分配更多空间的方法。由于减少了磁头移动，因此这有助于提高性能。

这些页跟踪数据库对象所用的磁盘空间：

- 全局分配映射 (GAM) 页包含整个数据库的分配位图。
- 分配页可在 256 页或 0.5MB 的组内跟踪空间使用情况和对象。
- 对象分配映射 (OAM) 页包含用于对象的扩充的有关信息。表和索引的各个分区至少有一个 OAM 页用于跟踪对象页在数据库中的存储位置。
- OAM 页管理分区表的空间分配。

全局分配映射页

每个数据库都有一个 GAM，用于存储数据库的所有分配单元的位图，每个分配单元 1 位。当分配单元没有可用于存储对象的自由扩充时，GAM 中的相应位将设置为 1。

这种机制能加速为对象分配新的空间。用户无法查看 GAM 页；它在系统目录中显示为 sysgams 表。

分配页

创建数据库或向数据库添加空间时，该空间被分成由 256 个数据页组成的分配单元。每个**分配单元**中的第一页为分配页。 0 页和所有 256 的整数倍的页为分配页。

分配页通过记录扩充中存储的对象的分区 ID、对象 ID 和索引 ID，以及已用和可用页数，来跟踪分配单元上每个扩充中的空间。分配页还存储表或索引的相应 OAM 页的页 ID。

对象分配映射页

表、索引和文本链的每个分区都有一个或多个对象分配映射 (OAM) 页，这些页存储在分配给表或索引的页中。如果表有多个 OAM 页，则这些页以链的形式链接。OAM 页存储指向含有该对象页的分配单元的指针。

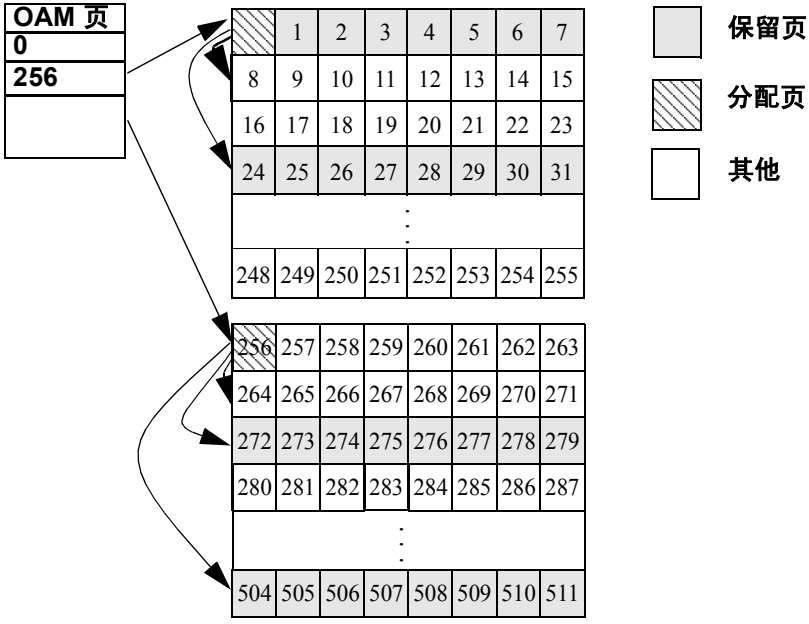
该链中的第一页用于存储分配提示，以指示链中哪一 OAM 页存储拥有可用空间的分配单元的信息。这提供了为对象分配额外空间并使新空间接近对象已使用的页的快捷方式。

OAM 页和分配页管理对象存储的方式

图 2-1 显示了 OAM 页和分配页管理分配单元、扩充和对象的方式。

- 图中显示了两个分配单元，一个起始于 0 页，一个起始于第 256 页。每个单元的第一页为分配页。
- 表存储在四个扩充上，在第一个分配单元上起始于第 1 页和第 24 页，在第二个分配单元上起始于第 272 页和第 504 页。
- 表的第一页为表的 OAM 页。它指向每个分配单元（对象使用其页）的分配页，因此它指向第 0 页和第 256 页。
- 分配页 0 和分配页 256 存储扩充所属的表的对象 ID、索引 ID 和分区 ID。分配页 0 指向表的第 1 页和第 24 页，分配页 256 指向第 272 页和第 504 页。

图 2-1：OAM 页和分配页指针



页分配将对象的页放在一起

Adaptive Server 会尝试将每个对象 （例如表分区、索引或表的文本链或图像链）的页分配放在一起。

通常，当 Adaptive Server 需要新页时：

- 如果对象的当前扩充包含可用页， Adaptive Server 将使用此页。
- 如果当前扩充中没有可用页，但分配给同一分配单元内的对象的另一个扩充中有可用页， Adaptive Server 将使用此页。
- 如果当前分配单元没有包含分配给对象的可用页的扩充，但有可用扩充， Adaptive Server 将分配可用扩充的第一个可用页。
- 如果当前分配单元已满，Adaptive Server 将扫描对象的 OAM 页，以查找其中扩充包含可用页的另一个分配单元，并使用第一个可用页。
- 如果没有 OAM 条目指示存在可用页，Adaptive Server 会将 OAM 条目与全局分配映射页进行比较，以查看任意分配单元是否有可用扩充。 Adaptive Server 将分配第一个可用扩充的第一个可用页。

- 如果所有 OAM 条目都用于已满的分配单元，Adaptive Server 将搜索全局分配映射，以查找至少包含一个可用扩充的分配单元。Adaptive Server 会将该分配单元的 OAM 条目添加到对象的 OAM 中，同时分配可用扩充，并使用该扩充的第一个可用页。

注释　类似 bcp 和 reorg rebuild 之类使用大规模分配的操作不会在已分配扩充中查找可用页，而是会分配已满的可用扩充。大规模分配通常无法使用每个分配单元的第一个扩充。第一个扩充仅有 7 个可用页，因为其第一页用于保存分配页的结构。

使用 *sysindexes* 和 *syspartitions* 的数据访问

sysindexes 表可存储已建索引和未建索引的表的信息。以下信息在 sysindexes 中各占一行：

- 所有页锁定表 — 如果此表没有聚簇索引，indid 列为 0，如果有聚簇索引，则为 1。
- DOL 锁定表 — 对于此表，indid 始终为 0。
- 非聚簇索引 — 以及对 DOL 锁定表创建的每个聚簇索引。
- 包含一个或多个 LOB 列的表 — 对于 LOB 结构，索引 ID 始终为 255。

syspartitions 存储每个表和索引分区的相关信息，其中每个分区各占一行。

在 Adaptive Server 15.0 及后续版本中，syspartitions 的各行存储指向表或索引的指针，以加快访问对象的速度。表 2-2 显示了在访问数据期间如何使用这些指针。

表 2-2：访问数据期间 syspartitions 指针的使用情况

列	用于表访问	用于索引访问
root	如果 indid 为 0，且表是已分区的所有页锁定表，则 root 指向堆的最后一页。	用于查找索引树的根页。
first	指向所有页锁定表页链中的第一个数据页。	指向 DOL 锁定表上非聚簇索引或聚簇索引中的第一个叶级页。
doampg	指向表中的第一个 OAM 页。	
ioampg		指向索引的第一个 OAM 页。

空间开销

Adaptive Server 按扩充为对象 （表、索引、文本页链）分配空间，每个扩充为八个逻辑页，而与逻辑页大小的配置无关。也就是说，如果服务器的逻辑页被配置为 2K，它便为每个对象分配一个 16K 的扩充；如果服务器的逻辑页被配置为 16K，它便为每个对象分配一个 128K 的扩充。

对于系统表也是如此。如果服务器有很多小表，当服务器使用较大的逻辑页时，空间消耗就会很大。

例如，对于逻辑页配置为 2KB 的服务器，systypes（包含约 31 个短行、一个聚簇索引和一个非聚簇索引）将保留 3 个扩充或 48KB 内存。如果将服务器迁移到使用 8KB 页，为 systypes 保留的空间仍为 3 个扩充，但内存为 192KB。

对于逻辑页配置为 16KB 的服务器， systypes 需要 384KB 的磁盘空间。对于小表，如果服务器使用较大逻辑页，则最后一个扩充中的未用空间会非常大。

较大的页大小也会对数据库产生影响。每个数据库包含系统目录及其索引。如果从较小逻辑页大小迁移到较大逻辑页大小，则必须要考虑每个数据库所需的磁盘空间量。

列的数量和大小

表中可创建的列的最大数量为：

- 所有页锁定表 (APL) 和 DOL 锁定表的固定长度列为 1024
- 对于 APL 表中的可变长度列，为 254
- 对于 DOL 锁定表中的可变长度列，为 1024

列的最大大小取决于：

- 表是否包含可变长度列或固定长度列。
- 数据库的逻辑页大小。例如，在 2K 逻辑页的数据库中，APL 表中的最大列大小可与单个行的大小相等，约为 1962 个字节，但要减去行格式的开销。同样，对于 4K 逻辑页的数据库，APL 表中的最大列大小可达 4010 个字节，但要减去行格式的开销。有关详细信息，请参见表 2-3。
- 在尝试创建包含固定长度列的表时，如果该列的大小超出逻辑页大小的限制， create table 将发出错误消息。

表 2-3：行和列的最大长度

锁定方案	页大小	最大行长度	最大列长度
	2K （2048 字节）	1962	1960 字节
	4K （4096 字节）	4010	4008 字节
APL 表	8K （8192 字节）	8106	8104 字节
	16K （16384 字节）	16298	16296 字节
	2K （2048 字节）	1964	1958 字节
	4K （4096 字节）	4012	4006 字节
DOL 锁定表	8K （8192 字节）	8108	8102 字节
	16K （16384 字节）	16300	16294 个字节 如果表不包含任何可变长度列
	16K （16384 字节）	16300 （受限于 varlen 的*最大起始*偏移 = 8191）	8191-6-2 = 8183 字节（如果表至少包含一个可变长度列）。

对于逻辑页大小为 16K 的 DOL 锁定表，固定长度列的最大大小取决于表是否包含可变长度列。可变长度列的最大可能起始偏移为 8191。如果表中有任何可变长度的列，则行的固定长度部分与开销的总和不能超过 8191 字节；而当表包含任何可变长度的列时，所有固定长度列的最大可能大小限制在 8183 字节。

在表 2-3 中，确定最大列长度时需要减掉行开销占用的 6 个字节和行长度字段占用的 2 个字节。

APL 表中的可变长度列

对于包含一个可变长度列 （如 varchar、 varbinary 等）的 APL 表，每行的最小开销如下：

- 初始行开销 2 个字节。
- 行长度 2 个字节。
- 行尾的列偏移表占用 2 个字节。它始终占用 *n+1* 个字节，其中 *n* 代表行中可变长度列的数目。

单列表的最小开销为 6 个字节，再加上额外开销。开销后的最大列大小必须小于或等于：
（列长度） + （额外开销） + （6 字节开销）

表 2-4：APL 表中可变长度列的最大大小

页大小	最大行长度	最大列长度
2K （2048 字节）	1962	1948
4K （4096 字节）	4010	3988
8K （8192 字节）	8096	8068
16K （16384 字节）	16298	16228

超过逻辑页大小的可变长度列

如果表使用 2K 逻辑页，可创建一些行总长度超过 2K 页大小的最大长度限制的可变长度列。这样在创建的表中，某些（并非全部）可变长度列就可包含最大允许大小。但是，当您发出 create table 时，会收到警告消息，指明生成的行大小可能超出最大限值，并可能导致后续 insert 或 update 失败。

例如，如果创建的表使用 2K 页大小，并包含长度为 1975 字节的可变长度列， Adaptive Server 会创建该表，但要发出警告消息。插入的数据不能超过最大行长度 （1962 字节）。

DOL 表中的可变长度列

对于包含一个可变长度列的 DOL 锁定表，每行的最小开销为：

- 六个字节的初始行开销。
- 两个字节用于存储行长度。
- 行尾的两个字节用于存储列偏移表。每个列偏移条目为两个字节。这样的条目共有 *n* 个，其中 n 代表行中可变长度列的数目。

总开销为 10 个字节。 DOL 行没有调整表。实际的可变长度列大小为：

```
列长度 + 10 字节开销
```

表 2-5：DOL 锁定表中可变长度列的最大大小

页大小	最大行长度	最大列长度
2K （2048 字节）	1964	1954
4K （4096 字节）	4012	4002
8K （8192 字节）	8108	8098
16K （16384 字节）	16300	16290

包含可变长度列的 DOL 锁定表的偏移必须少于 8191 个字节，才能保证所有插入操作都成功执行。例如，以下插入操作将失败，因为列 c2、c3 和 c4 的偏移为 9010，超过了最大限值 8191 字节：

```
create table t1(
    c1 int not null,
    c2 varchar(5000) not null
    c3 varchar(4000) not null
    c4 varchar(10) not null
    ... more fixed length columns)
cvarlen varchar(nnn)) lock datarows
```

宽可变长度行

Adaptive Server 允许宽可变长度 DOL 行的仅数据锁定 (DOL) 列使用的最大行偏移为 32767 字节，前提是其将逻辑页大小配置为 16K。

可使用以下命令为每个数据库启用宽可变长度 DOL 行：

```
sp_dboption database_name, 'allow wide dol rows', true
```

注释　缺省情况下为临时数据库启用 allow wide dol rows。不能为 master 数据库设置 allow wide dol rows。

sp_dboption 'allow wide dol rows' 对逻辑页大小小于 16K 的用户数据库不起作用，因为 Adaptive Server 不能在小于 16384 字节的页上创建宽可变长度 DOL 行。

以下示例将页大小为 16K 的服务器上的 pubs2 数据库配置为使用宽可变长度 DOL 行：

1　为 pubs2 数据库启用宽可变长度行：

```
sp_dboption pubs2, 'allow wide dol rows', true
```

2　创建 book_desc 表，其中包含在 8192 行偏移后开始的宽可变长度 DOL 列：

```
create table book_desc
(title varchar(80) not null,
title_id varchar(6) not null,
author_desc char(8192) not null,
book_desc varchar(5000) not null)
lock datarows
```

批量复制宽数据

必须使用 Adaptive Server 15.7 及以后版本随附的 bcp 版本批量复制包含宽可变长度 DOL 行的数据。必须配置用于接收数据以接受宽可变长度 DOL 行的数据库 （即， bcp 不将宽行复制到尚未为其启用 allow wide dol rows 的数据库中）。

检查降级

请参见所用平台的 《安装指南》，以了解有关如何降级使用宽可变长度行的 Adaptive Server 的详细信息。

转储和装载宽可变长度 DOL 列

数据库和事务日志转储会为要导入装载转储的数据库中的 allow wide dol rows 保留其设置 （如果此数据库没有选项集）。

例如，如果将名为 my_table_log.dmp 的事务日志转储（allow wide dol rows 已设置为 true）装载到数据库 big_database（尚未为其设置 allow wide dol rows）中，那么在装载到 big_database 中后， my_table.log 会为 allow wide dol rows 将其设置保留为 true。

但是，如果数据库或事务日志转储未设置 allow wide dol rows，但用于装载转储的数据库已进行此设置， allow wide dol rows 将保留原有设置。

不能将启用 allow wide dol rows 的数据库的转储装载到 15.7 之前的 Adaptive Server 版本上。

使用包含宽可变长度 DOL 行的代理表

您可以使用包含宽可变长度 DOL 行的代理表。

创建代理表（不考虑表的行长度）时，可在控制 Adaptive Server 上执行 create table 或 create proxy table 命令。 Adaptive Server 可在您连接到的服务器上执行这些命令。但是， Adaptive Server 可在存储数据的服务器上执行数据操作语句 （insert、 delete），并且用户的本地服务器只设置请求格式，然后发送该请求；本地服务器不进行任何控制。

即使数据驻留在远程服务器上， Adaptive Server 仍会按照在本地服务器上的创建方式创建代理表。当 create proxy_table 命令创建包含宽可变长度行的 DOL 锁定表时，仅在用来创建代理表的数据库将 allow wide dol rows 设置为 true 时， 该命令才能成功执行。

注释 Adaptive Server 使用本地服务器的 lock scheme 配置创建代理表；如果将 lock scheme 设置为 datarows 或 datapages，则可创建包含 DOL 行的代理表。

在代理表中插入或更新数据时， Adaptive Server 会忽略本地数据库的 allow wide dol rows 设置。包含数据的服务器决定 insert 或 update 能否成功执行。

转换锁定方案或使用 select into 的限制

无论是使用 alter table 更改锁定方案，还是使用 select into 将数据复制到新表中，下列限制均适用。

对于使用除 16K 页外其它页大小的服务器， APL 表中可变长度列的最大长度比 DOL 锁定表的最大长度小，所以可将包含最大可变长度列的 APL 表的锁定方案转换成 DOL。但不能将至少包含一个最大可变长度列的 DOL 锁定表转换为 APL 表。

在使用 16K 页的服务器上，APL 表可存储的可变长度列的大小要比 DOL 锁定表大得多。您可以将表从 DOL 转换为 APL，但不能反向转换。

仅当源表中的数据超出目标表限制时，才会出现这些锁定方案转换限制。如果出现这种情况，在将行格式从一种锁定方案向另一种转换时，Adaptive Server 会发出错误消息。如果表为空，则不需要进行此类数据转换，锁更改操作将成功执行。但是，在对表执行后续 insert 或 update 操作时，转换后表的目标方案的列或行大小限制可能会引发错误。

按可变长度列的大小组织 DOL 锁定表中的列

对于使用可变长度列的 DOL 锁定表，排列各列时要使最长的列靠近表定义的末尾。与大列出现在表定义开始处的情况相比，前一种方式可创建带更大行的表。例如，在 16K 页服务器中，下列表定义是可接受的：

```
create table t1 (
    c1 int not null,
    c2 varchar(1000) null,
    c3 varchar(4000) null,
    c4 varchar(9000) null) lock datarows
```

但对于将来的插入操作，下列表定义通常是不可接受的。由于之前将 c4 列设置为 9000 字节，因此 c2 列的可能起始偏移将超出上限 8192 字节：

```
create table t2 (
    c1 int not null,
    c4 varchar(9000) null,
    c3 varchar(4000) null,
    c2 varchar(1000) null) lock datarows
```

表创建完毕，但在将来的插入操作可能失败。

每个数据页的行数

DOL 数据页所允许的行数由下列因素决定：

- 页大小
- 用于指定行转移地址的行 ID 的 10 字节开销。

表 2-6 显示 DOL 数据页上可容纳的最大行数：

表 2-6：一个 DOL 数据页的最大数据行数

页大小	最大行数
2K	166
4K	337
8K	678
16K	1361

APL 数据页可拥有的最大行数为 256。由于每页需要有一个字节的行数说明符，因此带短行的大页会产生一些未使用的空间。例如，如果将 Adaptive Server 配置为 8K 逻辑页，行长为 25 字节，则计入行偏移表和页头后，每页将有 1275 字节的未使用空间。

其它对象和大小限制数

存储过程的最大参数数为 2048。请参见 《Transact-SQL 用户指南》的第 16 章 “使用存储过程”。

对 Adaptive Server 12.5 及更高版本可存储的数据的限制与对 12.5 以前版本所存储数据的限制不同。客户端必须能够存储和处理这些新数据限制。如果正在使用较早版本的 Open Client 和 Open Server，则在下列情况下，它们不能处理数据：

- 升级到 Adaptive Server 12.5 及更高版本
- 删除和重新创建包含宽列的表
- 插入宽数据

请参见 Open Client Configuration Guide（《Open Client 配置指南》）的第 2 章 “Basic Configuration for Open Client”（Open Client 的基本配置）。

不含聚簇索引的表

如果在 Adaptive Server 上创建表，但不创建聚簇索引，该表将存储为堆表，这意味着不按任何特定顺序存储数据行。本节介绍当没有 “有用” 索引可帮助检索数据时，如何对堆表执行 select、insert、delete 和 update 操作。

堆表用途很少。多数应用程序在表上存在聚簇索引时会执行得更好。不过，对于仅使用少量页的小表和很少更改的表，堆表则是理想选择。

对于不要求以下项的表，堆表很有用：

- 直接访问单个随机行
- 对结果集进行排序

对于针对必须返回表行子集的多数大表进行的查询，堆表效果不理想。

在频繁执行大批量插入 （此时不接受删除和创建聚簇索引的开销）的应用中，分区的堆表很有用。

顺序磁盘访问效率高，在使用大 I/O 和异步预取时更是如此。但必须始终扫描整个表以查找相关值，这对数据高速缓存和其它查询可能会产生很大影响。

批量插入还可执行有效的顺序 I/O。但如果多个进程尝试同时插入数据，则最后一页中可能出现瓶颈。

有时，索引位于 where 子句中指定的列中，但优化程序确定使用索引会比执行表扫描产生更高开销。

当选择表中所有行时，始终使用表扫描。唯一的例外情况是查询只包括那些在非聚簇索引中为键的列。

有关详细信息，请参见 Performance and Tuning Series: Locking and Concurrency Control （《性能和调优系列：锁定和并发控制》）中的第 5 章 “Indexes”（索引）。

锁定方案

APL 表中的数据页通过每页上的指针链接为页列表。 DOL 锁定表中的页不能链接成页链。

在所有页锁定表中，每页存储一个指向页链中下一页和前一页的指针。在插入新页时，两个相邻页上的指针将更改为指向新页。Adaptive Server 在扫描所有页锁定表时，会跟随这些页指针按顺序读取页。

在所有页锁定表的每个索引级和 DOL 锁定表的索引叶级，页也是双重链接的。如果所有页锁定表已分区，则每个分区有一个页链。

与所有页锁定表不同， DOL 锁定表除了在您刚创建聚簇索引后维护页链，其它时间通常不进行维护。但是，当您第一次在表上发出命令时，此页链将中断。

Adaptive Server 在扫描 DOL 锁定表时使用存储在 OAM 页中的信息。请参见第 23 页的 “对象分配映射页”。

所有页锁定表与 DOL 锁定表间的另一个区别是后者使用固定的行 ID。这意味着在正常查询处理期间，行 ID （页号与页上行号的组合）在 DOL 锁定表中不发生改变。

只有在执行要求复制数据行的某项操作时，行 ID 才发生更改，例如，在 reorg rebuild 期间或在创建聚簇索引时。

有关固定行 ID 如何影响堆操作的信息，请参见第 37 页的 “从 data-only-locked 堆表删除”和第 38 页的 “data-only-locked 堆表”。

对堆表执行选择操作

当对堆表发出 select 查询，并且没有有用的索引时， Adaptive Server 必须扫描表中的每个数据页，以查找满足查询条件的所有行。可能存在一个或多个匹配行，或者没有匹配行。

所有页锁定堆表

对于所有页锁定表， Adaptive Server 会读取该表 syspartitions 中的 firstpage 列，将第一页读入高速缓存中，然后跟随下一页的指针继续操作，直至找到表的最后一页。

Data-only locked 堆表

DOL 锁定表的各页未链接为页链，因此对 DOL 锁定堆表的 select 查询会使用表的 OAM 和分配页来定位表中的所有行。OAM 页指向分配页，这些分配页指向表的扩充和页。

将数据插入到所有页锁定堆表中

将数据插入无聚簇索引的所有页锁定堆表中时，会始终将数据行添加到表的最后一页。如果表中没有聚簇索引，并且表未分区，则堆表的 syspartitions.root 条目将存储指向堆表最后一页的指针，以指示应在其中插入数据的页。

如果最后一页已满，则在当前扩充中分配新页，并将其链接到链上。如果扩充已满，Adaptive Server 将在该表正在使用的其它扩充上查找空页。如果没有可用页，就给表分配新扩充。

对使用所有页锁定的堆表的一项严格的性能限制是，当向某页添加行时，该页必须锁定，且在事务完成前始终保持锁定状态。如果有许多用户同时尝试对所有页锁定堆表执行插入操作，则每项插入操作必须等待前一个事务完成后才能进行。

出现以下情况时堆表的最后一页将出现冲突问题：

- 插入单行
- 使用 select into 或 insert...select，或在批处理中使用多个 insert 语句插入多行
- 批量复制到表中

要消除堆表最后一页中的冲突，可尝试以下方法：

- 切换到数据页或数据行锁定
- 创建聚簇索引，将插入操作引导到其它页上
- 将表分区，为该表创建多个插入点，在所有页锁定表中提供多个“最后页”

对于可能出现锁定冲突的所有事务，还可以：

- 保持事务简短
- 事务获取锁后，尽可能避免网络活动和用户交互

将数据插入到 data-only-locked 堆表中

用户将数据插入 DOL 锁定堆表时，Adaptive Server 会跟踪最近执行插入操作的页码，并保留页码作为一个提示，供以后需要空间的任务使用。随后将对表的插入操作引导到这些页中的一个。如果该页已满， Adaptive Server 会分配一个新页，并用新页码替换原来的提示。

在向 DOL 锁定堆表执行插入操作的过程中，许多用户同时插入数据时发生的阻塞现象要比 APL 表少很多。如果确实发生阻塞， Adaptive Server 会分配少量空页，并使用这些新分配的页作为提示，来将新插入操作引导到这些页。

对于数据行锁定表，只有在写入对数据页的实际更改时才会发生阻塞现象；尽管在事务执行期间事务会持有行锁，但仍可在页上插入其它行。行级锁允许多个事务在页上持有锁。

在 DOL 锁定表上可能会发生少量阻塞，因为 Adaptive Server 允许在刚分配完大量页后发生少量阻塞，这样就可在新分配的页填满后再分配其它页。

对于 DOL 锁定表，向堆表执行插入操作期间的冲突量已大大减少，但仍会发生冲突。如果这些冲突影响到插入速度，请尝试以下方法：

- 切换到数据行锁定 （如果表采用数据页锁定方案）
- 使用聚簇索引分布数据插入
- 对表进行分区，这会提供其它提示，当发生阻塞时允许在每个分区上分配新页

删除堆表上的数据

当从无有用索引的堆表中删除行时， Adaptive Server 将扫描该表中的数据行以查找要删除的行。如果不对每一行进行检查，就无法知道有多少行符合查询条件。

从所有页锁定堆表删除

从所有页锁定表中的某页中删除数据行时，该页上位于此行之后的行将上移，从而使该页上的数据保持连续，避免页内出现分段。

从 data-only-locked 堆表删除

从 data-only-locked 堆表删除行时，如果没有有用的索引，则需要进行表扫描。使用 OAM 和分配页来定位这些页。

页上的空间不会立即恢复。 DOL 锁定表中的行必须保持固定的行 ID，如果事务被回退，则必须将这些行重新插回原来的位置。

删除事务完成后，下列进程之一可将页上的行移位，以使空间的使用保持连续：

- 管家垃圾收集进程
- 需要在页上找到空间的插入
- reorg reclaim_space 命令

删除页上最后一行

如果删除页上的最后一行，该页就被释放。如果表仍使用该扩充上的其它页，当需要页时，表还可再次使用该页。

如果扩充上的所有其它页均为空，则整个扩充被释放。它可被分配给数据库中的其它对象。表或索引的第一个数据页始终不会被释放。

更新堆表上的数据

与对堆表执行的其它操作一样，在对在 where 子句中的各列上没有有用索引的表执行 update 时，需要执行表扫描以查找要更改的行。

所有页锁定堆表

可采用以下几种方法对所有页锁定堆表执行更新：

- 如果行的长度不改变，更新的行就会替换现有行，并且页上不发生数据移动。
- 如果行的长度发生改变，并且页上有足够的可用空间，则该行在页上仍处于原来位置，但其它行向上或向下移动，使这些行在页上保持连续。

  页尾处的行偏移指针被调整为指向变化后的行位置。
- 如果行超出页范围，将从当前页中删除该行，然后再在表的最后一页将其作为新行插入。这种更新可能导致堆表的最后一页上出现冲突。

data-only-locked 堆表

对 DOL 锁定表的要求之一是数据行的行 ID 始终不能改变（欲重建表的情况除外）。因此，只要行不超出页的范围，就可按上述前两个方法对 DOL 锁定表进行更新。

但如果在更新 DOL 锁定表时某行超出了页范围， **row forwarding** 进程将执行下列步骤：

1 该行被插入到另一页，且

2 在该行的初始位置存储一个指向新页上行 ID 的指针。

转移行时不需要修改索引。所有索引仍指向原始的行 ID。

如果必须再次转移行，将更新初始位置，以指向新页 — 转移的行与其初始位置之间不能有一个以上的跳转。

在执行更新操作期间，由于不必更新索引，因此行转移增加了并发性。但这会减慢数据检索速度，因为任务必须在原始位置读取该页，然后读取存储该转移数据的页。

使用 reorg 命令可从表中清除转移的行。

请参见 《性能和调优系列：查询处理和抽象计划》的第 1 章 “了解查询处理”。

Adaptive Server 如何为堆操作执行 I/O

当查询需要数据页时， Adaptive Server 先检查该页是否在数据高速缓存中。如果不在其中，则必须从磁盘读取它。新安装且逻辑页为 2K 的 Adaptive Server 具有一个配置为 2K I/O 的数据高速缓存。每个 I/O 操作均读取或写入一个 Adaptive Server 数据页。系统管理员可以：

- 配置多个高速缓存
- 将表、索引或文本链绑定到高速缓存
- 将数据高速缓存配置成以页大小的倍数 （最多个数据页，即一个扩充）执行 I/O 操作

为了最有效地利用这些高速缓存并减少 I/O 操作， Adaptive Server 优化程序可以执行以下操作：

- 选择每次预取最多八个数据页
- 选择不同的高速缓存策略

顺序预取，或大 I/O

Adaptive Server 数据高速缓存可以配置为允许使用大 I/O。如果高速缓存允许使用大 I/O， Adaptive Server 便可以预取数据页。

高速缓存包含由逻辑页大小决定容量的缓冲池，使 Adaptive Server 能够在单个 I/O 操作中最多读取整个扩充 （八个数据页）。

由于执行 I/O 操作所需的大部分时间都用在了搜索和定位上，因此，以 16K I/O 中读取八页的时间几乎与以 2K I/O 读取单页所需的时间相同。使用八个 2K I/O 读取八页的开销几乎是使用单个 16K I/O 读取八页的开销的八倍。使用大 I/O 时执行表扫描操作更加理想。

当使用一个 I/O 将多个页读入高速缓存时，这些页将被视为一个单元：这些页在高速缓存中一起老化，并且如果单元中的任何一页已在缓冲区位于高速缓存中时发生更改，则所有页将作为一个单元写入磁盘。

请参见 《性能和调优系列：基础知识》的第 5 章 “内存使用和性能”。

注释　对大 I/O 的引用位于一台逻辑页大小为 2K 的服务器上。如果服务器的逻辑页大小为 8K，则用于 I/O 的基本单位为 8K。如果服务器的逻辑页大小为 16K，则用于 I/O 的基本单位为 16K。

维护堆表

随着时间的推移，当存储变得分散时，堆表上的 I/O 效率会变低。删除和更新操作可导致：

- 很多被部分填充的页
- 大 I/O 的效率低下，因为扩充可能包含很多空页
- DOL 锁定表中的转移行

在堆表内回收空间：

- 使用 reorg rebuild 命令 （仅限 DOL 锁定表）
- 创建并删除聚簇索引
- 使用 bcp （批量复制实用程序）和 truncate table

使用 *reorg rebuild* 回收空间

reorg rebuild 可将所有数据行复制到新页，并对堆表重建所有索引。可以只对 DOL 锁定表使用 reorg rebuild。

通过创建聚簇索引回收空间

要创建聚簇索引，数据库中必须至少有相当于表大小 120% 的可用空间。

请参见第 104 页的 “确定维护活动的可用空间”。

使用 *bcp* 回收空间

要使用 bcp 回收空间：

1 使用 bcp 可将表复制到文件。

2 使用 truncate table 可截断表，以回收未使用的空间。

3 再次使用 bcp 将表复制回来。

有关使用分区表的详细信息，请参见 《Transact-SQL 用户指南》的第 10 章 “对表和索引进行分区”。

有关 bcp 的详细信息，请参见 《实用程序指南》。

事务日志：一种特殊的堆表

Adaptive Server事务日志是一种特殊的堆表，用于存储数据库中有关数据修改的信息。事务日志始终为堆表；每个新的事务记录被附加到日志的结尾。事务日志不含索引。

将日志放置在与数据页和索引页不同的物理设备上。由于日志是有顺序的，因此日志设备上的磁头很少需要执行搜索操作，日志可保持高的 I/O 率。

事务日志的写操作经常发生。不要让它们与数据库中其它的 I/O 出现争用，这种争用通常发生在数据页上分散的位置处。

除恢复操作外，以下操作也可读取事务日志：

- 在延迟模式下执行的任何数据修改。
- 包含对插入表和删除表的引用的触发器。当查询表时，从事务日志记录建立这些表。
- 事务回退。

多数情况下，当 Adaptive Server 需要读取这些查询的事务日志页时，这些页仍位于数据高速缓存中，且不需要磁盘 I/O。

有关如何提高事务日志性能的信息，请参见第 5 页的 “将事务日志保存在单独的磁盘上”。

堆表的异步预取和 I/O

必须执行物理 I/O 的所有任务在等待 I/O 完成的过程中都将释放服务器的引擎 (CPU)。如果表扫描必须读取 1000 页，并且其中的任何页都不在高速缓存中，则执行不进行异步预取的 2K I/O 意味着该任务需要在引擎上执行 1000 次循环，然后进入休眠状态等待 I/O。使用 16K I/O 则只需进行 125 次循环。

异步预取有助于提高用于执行表扫描的查询的性能。当任务从属于表的分配单元上读取第一页时，异步预取可请求该分配单元上的所有页。如果 1000 页表驻留在正好 4 个分配单元中，则任务需要很少的执行和休眠循环的周期性循环。

I/O 的类型	循环	每个循环中的步骤
2K I/O 不预取	1000	1　请求一页。 2　休眠直至从磁盘读出该页。 3　请求一页。 4　等待在 Adaptive Server 引擎 (CPU) 上运行的时机。 5　读取页上的行。
16K I/O 不预取	125	1　请求一个扩充。 2　休眠直至从磁盘读出该扩充。 3　等待在 Adaptive Server 引擎 (CPU) 上运行的时机。 4　读取 8 页上的行。
预取	4	1　请求分配单元中的所有页。 2　休眠直至从磁盘读出第一页。 3　等待在 Adaptive Server 引擎 (CPU) 上运行的时机。 4　读取高速缓存中所有页上的所有行。

实际性能取决于高速缓存的大小和数据高速缓存中的其它活动。

请参见 《性能和调优系列：基础知识》的第 6 章 “调优异步预取”。

高速缓存和对象绑定

可将表绑定到特定的高速缓存。如果表未绑定到特定的高速缓存，但其数据库被绑定到某高速缓存，则它的所有 I/O 都发生在该高速缓存中。

否则，表的 I/O 将发生在缺省数据高速缓存中。可以为大 I/O 配置缺省数据高速缓存。使用堆表的应用程序很可能在使用配置为 16K I/O 的高速缓存时性能最佳。

请参见 《系统管理指南，卷 2》的第 4 章 “配置数据高速缓存”。

堆表、 I/O 和高速缓存策略

每个 Adaptive Server 数据高速缓存均作为缓冲区的一个 MRU/LRU （最近使用最多的/最近使用最少的）链来管理。随着缓冲区在高速缓存中的老化，它们会从 MRU 端向 LRU 端移动。

当高速缓存中的已更改页通过 MRU/LRU 链上的 **wash marker** 点时，Adaptive Server 会对位于高速缓存中期间已更改的任何页启动异步写入操作。这有助于保证当这些页到达高速缓存的 LRU 端时，它们是干净的，并可被重新使用。

为高效利用数据高速缓存， Adaptive Server 主要采用两个策略：

- LRU 替换策略
- MRU，或读取和放弃替换策略

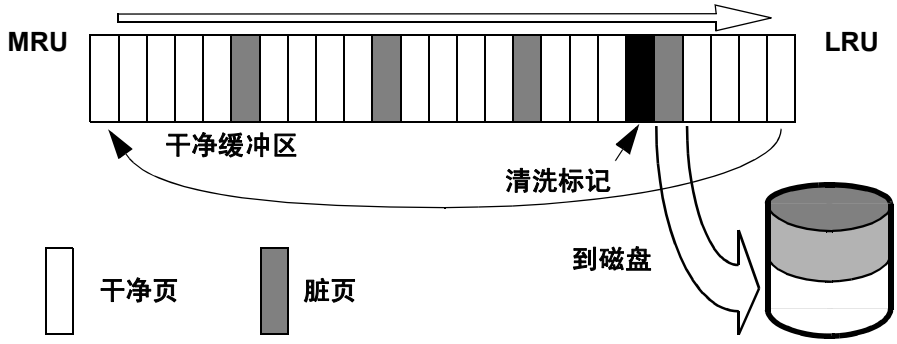
LRU 替换策略

Adaptive Server将 LRU 策略用于：

- 修改页上数据的语句
- 单个查询需要访问多次的页
- OAM 页
- 多数索引页
- 指定了 LRU 策略的任何查询

LRU 替换策略按顺序将数据页读入到高速缓存中，替换 “最近最少使用的”缓冲区。该缓冲区被放置在数据缓冲区链的 MRU 端。随着更多的页被读入高速缓存，它逐渐向 LRU 端移动。

图 2-2：LRU 策略从高速缓存的 LRU 端取出一个干净页



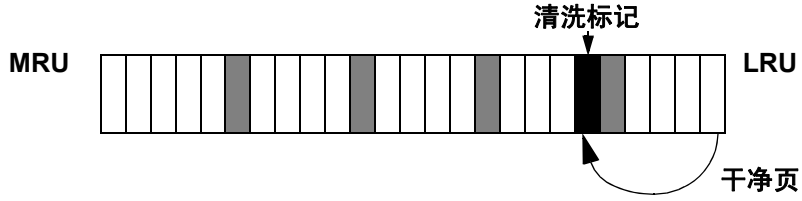
MRU 替换策略

MRU （读取和放弃）最常用于对某页只需访问一次的查询，包括：

- 不使用连接的查询中的多数表扫描
- 连接查询中的一个或多个表

MRU 替换策略用于对堆表进行表扫描。这种策略将页放置在高速缓存中紧挨 （之前）清洗标记的地方 （如图 2-3 所示）。

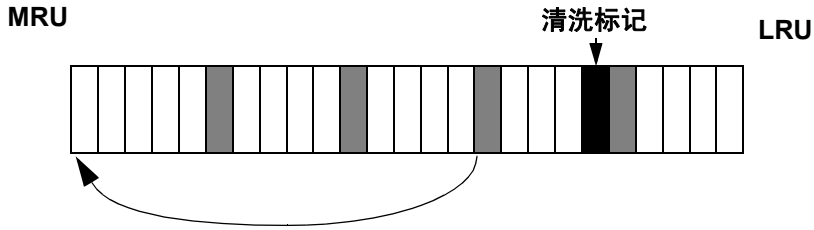
图 2-3：MRU 策略将页放置在紧挨 （之前）清洗标记的地方



将所需页仅在清洗标记处放置一次，这意味着它们不会将其它页推出高速缓存。

读取和放弃策略仅用于查询时从磁盘实际读取出的页。如果某页因表上以前的活动已存在于高速缓存中，该页就会被放置在高速缓存的 MRU 端。

图 2-4：在高速缓存中查找所需的页



选择操作和高速缓存

多数情况下，堆上的单表 select 操作使用：

- 表可用的最大 I/O 和
- 读取和放弃 (MRU) 替换策略

对于堆表，执行大 I/O 的选择操作可能非常高效。 Adaptive Server 可按顺序读取表中的所有扩充。

请参见 《性能和调优系列：查询处理和抽象计划》的第 1 章 “了解查询处理”。

除非正将堆作为嵌套循环连接的内部表进行扫描，否则查询时只需要访问一次数据页，所以 MRU 替换策略会从高速缓存读取和放弃页。

注释 只有在页链未分段时，所有页锁定堆表的大 I/O 才更高效。请参见第 39 页的 “维护堆表”。

数据修改和高速缓存

Adaptive Server 通过将被更改的页保留在高速缓存中来尝试尽可能减少磁盘写入操作。数据页驻留在高速缓存中时，许多用户都可对其进行修改。这些更改记录在事务日志中，但更改数据和索引页不会立即写入磁盘。

堆表的高速缓存和插入

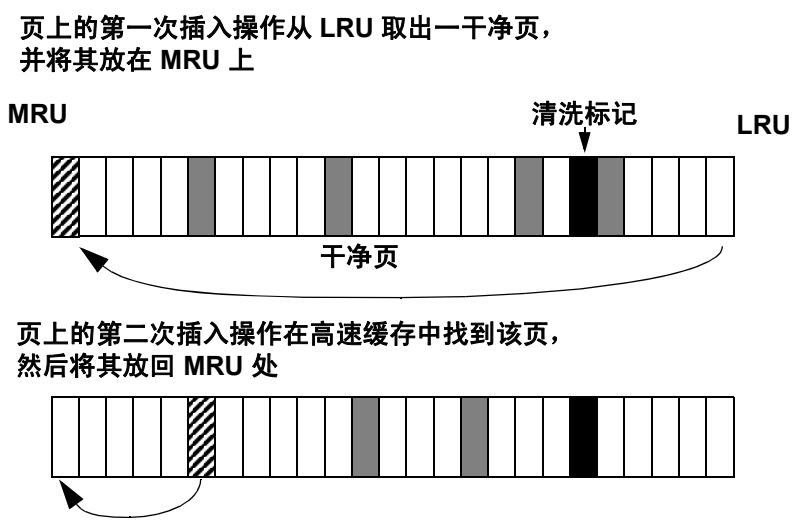
对堆表执行插入操作：

- 所有页锁定表的最后一页上
- DOL 锁定表上最近成功执行了插入操作的页上

如果插入发生在表的新页的第一行上，就会分配干净的数据缓冲区来存储数据页 （如图 2-5 所示）。当其它进程将其它页读入到内存时，此页开始在数据高速缓存中沿 MRU/LRU 链向下移动。

如果在该页仍处于内存中时，发生了另一个插入操作，则该页位于高速缓存中，并移回到 MRU/LRU 链的顶部。

图 2-5：向数据高速缓存中的堆页插入



被更改的数据页保留在高速缓存中，直至到达页链的 LRU 端。当该页处于高速缓存中时，可被修改或引用多次，但只有当发生下列一种情况时，它才被写入磁盘：

- 该页在移动时通过了清洗标记。
- 检查点或管家清洗任务将其写入到磁盘。

请参见 《性能和调优系列：基础知识》的第 5 章 “内存使用和性能”。

对堆表执行高速缓存、更新和删除操作

从堆表更新或删除一行时，对数据高速缓存产生的影响与插入进程类似。如果页已存在于高速缓存中，则行会更改，然后整个缓冲区（可能是单页或多页，具体取决于 I/O 大小）被放置在链的 MRU 端。

如果该页未在高速缓存中，就会从磁盘将其读入到高速缓存中，并对其进行检查以确定该页上的行是否与查询子句匹配。它在 MRU/LRU 链上的放置取决于是否需要更改页上的数据：

- 如果需要更改页上的数据，则缓冲区被放置在 MRU 端。它保留在高速缓存中，在被刷新到磁盘之前，其他用户可对其反复更新或读取。
- 如果不需要更改页上的数据，则缓冲区将被放置在高速缓存中的清洗标记之前，紧邻该标记。

第 3 章

设置空间管理属性

设置空间管理属性有助于减少保持表和索引的高性能所需的维护工作量。

本章介绍以下内容：用于控制空间使用的主空间管理属性，这些属性将如何影响空间使用，以及如何将这些属性应用到不同表和索引。

减少索引维护

利用 create index 命令的 fillfactor 选项，可以指定索引页和聚簇索引的数据页的填充程度。指定除 100% 之外的任意数量的 fillfactor 值时，数据和索引行使用的磁盘空间多于缺省设置情况下需要的空间。

如果要为大小将增加的表创建索引，可使用 fillfactor 选项减少页面拆分对表和索引的影响。

创建索引时将使用 fillfactor，在执行表重组操作（例如，在重建聚簇索引或对表运行 reorg rebuild 时）过程中使用 reorg rebuild 重建索引时将再次用到此命令。 fillfactor 值不保存在 sysindexes 中，且不会随时间的推移对数据或索引页的填充度进行维护。以后对表执行插入或更新操作期间不会不断对 fillfactor 进行维护。

如果最初索引的叶级页只部分充满（由于 fillfactor 值所致），但由于进行后续插入而使用了此可用空间，则叶级页日后将可能拆分。使用 reorg rebuild...index 可构建叶级页，应使用 fillfactor 的指定值创建叶级页，以便将来进行插入时不会引发此类拆分。对整个索引级运行 reorg rebuild，以使 fillfactor 的值可以为整个索引提供更多叶级空间。如果存在本地索引，应在分区级运行 reorg rebuild index，以便仅调整本地索引分区中的叶页，为将来的叶级插入保留更多空间。

注释 Adaptive Server 15.0 及更高版本能够在本地索引分区上运行 reorg rebuild...index。

发出 create index 命令时，作为该命令一部分指定的 fillfactor 值按如下方式应用：

- 聚簇索引：
  - 在所有页锁定表中， fillfactor 应用于数据页。
  - 在 DOL 锁定表中， fillfactor 应用于索引的叶页，数据页将完全充满 （除非已使用 sp_chgattribute 存储该表的 fillfactor）。
- 非聚簇索引 — fillfactor 值应用于索引的叶页。

在表上运行 reorg rebuild 时，也可使用 sp_chgattribute 来存储所用的 fillfactor 值。

请参见第 49 页的 “设置 fillfactor 值”。

使用 *fillfactor* 的优势

将 fillfactor 设置为较小值，可暂时改善性能。数据库插入操作增加了用于数据或索引页的空间，这会使性能降低。

使用较低的 fillfactor 值：

- 可减少叶级索引以及所有页锁定表的数据页上的页面拆分。
- 可提高带有进行过插入操作的聚簇索引的 DOL 锁定表中数据行的聚簇程度。
- 可减少使用页级锁定的表的锁争用，因为它降低了两个进程同时需要相同数据或索引页的可能性。
- 由于较少发生页面拆分，因此可帮助保持数据页和非聚簇索引叶级的高 I/O 效率。这意味着扩充上的八页有可能按顺序排列。

使用 *fillfactor* 的缺点

如果使用 fillfactor（特别是非常小的值），可能会对查询和维护活动产生以下影响：

- 必须为对非聚簇索引进行表扫描或叶级扫描的每个查询读取更多页。

  某些情况下，由于数据级将存在更多页且各索引级也可能存在更多页，因此可以向索引 B 树结构添加一级。

- 索引大小增加会降低索引的空间利用效率。由于无法在页级调优 fillfactor 值，因此具有倾斜数据分配的页面拆分会频繁发生，即使有可用的保留空间也是如此。
- dbcc 命令必须检查更多页，因此需要更长时间。
- 由于必须转储更多页，因此运行 dump database 所需的时间将增加。dump database 复制存储数据的所有页，但不转储尚未使用的页。转储和装载也可能使用更多磁带。
- Fillfactor 值将随时间逐渐减弱。如果使用 fillfactor 来降低页面拆分对性能的影响，那么当页面拆分开始降低性能时，应监控系统并重新创建索引。

设置 *fillfactor* 值

使用 sp_chgattribute 可存储每个索引和表的 fillfactor 百分比。用 sp_chgattribute 设置的 fillfactor 适用于以下情况：

- 对使用任何锁定方案的表运行 reorg rebuild。
- 使用 alter table...partition by 对表进行重新分区。
- 使用 alter table...lock 更改表的锁定方案。或者使用需要复制表的 alter table...add/modify 命令。
- 运行 create clustered index，并为表存储一个值。

请参见 《参考手册：命令》，了解有关这些命令的详细信息。

如果使用缺省的填充因子 0，则在创建新索引时，索引管理进程将在每个索引页上为另外两个行保留空间。将 fillfactor 设置为 100% 后，它将不再为这些行保留空间。仅在计算聚簇索引页数量和非叶页数量时，fillfactor 才会对大小计算产生影响。这两个计算都须从每页行数中减去 2。从这些计算中消除 -2。

fillfactor 的其它值会减少数据页和叶索引页上每页的行数。使用 fillfactor 时若要保证计算出的值正确，应将可用数据页的大小 (2016) 乘以 fillfactor。例如，如果 fillfactor 是 75%，则数据页将可容纳 1471 个字节。计算每页行数时，用此值代替 2016。有关这些计算的详细信息，请参见第 80 页的 “计算数据页的数量”和第 83 页的 “计算索引中叶页的数量”。

当因 create clustered index 命令需要建立非聚簇索引时， Adaptive Server 不会应用存储的 fillfactor：

- 如果使用 create clustered index 指定 fillfactor 值，该值将应用到每个非聚簇索引。
- 如果未使用 create clustered index 指定任何 fillfactor 值，则服务器范围的缺省值（使用 default fillfactor percent 配置参数设置）将应用于全部索引。

fillfactor 示例

以下示例演示 fillfactor 值的应用情况。

无存储的 *fillfactor* 值

如果未在 sysindexes 中存储 fillfactor 值， Adaptive Server 将应用在 create index 中指定的 fillfactor，如表 3-1 所示。

```
create clustered index title_id_ix
on titles (title_id)
with fillfactor = 80
```

表 3-1：没有表级存储值时应用的 fillfactor 值

命令	所有页锁定表	DOL 锁定表
create clustered index	数据页：80	数据页：完全充满 叶页：80
非聚簇索引重建	叶页：80	叶页：80

非聚簇索引使用在 create clustered index 命令中指定的 fillfactor。

如果未在 create clustered index 中指定 fillfactor，则非聚簇索引将始终使用服务器范围的缺省值；而绝不会使用 sysindexes 中的值。

用于 *alter table...lock* 和 *reorg rebuild* 的值

如果未存储 fillfactor 值， alter table...lock 和 reorg rebuild 都会应用 default fillfactor percentage 设置的服务器范围的缺省值。将应用缺省的 fillfactor，如表 3-2 所示。

表 3-2：重建期间应用的 fillfactor 值

命令	所有页锁定表	DOL 锁定表
聚簇索引重建	数据页：缺省值	数据页：完全充满 叶页：缺省值
非聚簇索引重建	叶页：缺省	叶页：缺省

存储的表级或聚簇索引 *fillfactor* 值

以下命令为表存储的 fillfactor 值为 50：

```
sp_chgattribute titles, "fillfactor", 50
```

如果将存储的 fillfactor 的表级值设置为 50，此 create clustered index 命令将应用 fillfactor 值，如表 3-3 所示。

```
create clustered index title_id_ix
on titles (title_id)
with fillfactor = 80
```

表 3-3：为聚簇索引使用存储的 fillfactor 值

命令	所有页锁定表	DOL 锁定表
create clustered index	数据页：80	数据页：50 叶页：80
非聚簇索引重建	叶页：80	叶页：80

注释　运行 create clustered index 时，存储在 sysindexes 中的所有表级 fillfactor 值都将重置为 0。

要在 create clustered index 或 reorg 命令期间指出仅数据锁定数据页已充满，必须首先发出 sp_chgattribute。

存储值后 *alter table...lock* 的影响

当 alter table...lock 命令复制表并重建索引时，将使用 fillfactor。

带聚簇索引的表

在所有页锁定表中，表和聚簇索引共享 sysindexes 行，以便只有一个 fillfactor 值可被存储并用于表和聚簇索引。可以通过提供表名或聚簇索引名为数据页设置 fillfactor 值。以下命令将值保存为 50：

```
sp_chgattribute titles, "fillfactor", 50
```

以下命令将值保存为 80，覆盖由前一个命令设置的值 50：

```
sp_chgattribute "titles.clust_ix", "fillfactor", 80
```

如果在发出上述 sp_chgattribute 命令之后变更 titles 表以使用仅数据锁定，则 fillfactor 存储值 80 将用于数据页和聚簇索引的叶页。

在 DOL 锁定表中，有关聚簇索引的信息存储在 sysindexes 的单个行中。为表指定的 fillfactor 值应用于数据页，为聚簇索引指定的 fillfactor 值应用于聚簇索引的叶级。

当更改 DOL 锁定表以使用所有页锁定时，为表存储的 fillfactor 用于数据页。 Adaptive Server 将忽略为聚簇索引存储的 fillfactor。

表 3-4 显示 fillfactor 值，将使用 alter table...lock 命令 （在运行完上述 sp_chgattribute 命令后执行）对数据和索引页设置这些值。

表 3-4：fillfactor 的存储值在变更表期间的作用

alter table...lock	无聚簇索引	聚簇索引
从所有页锁定到仅数据锁定	数据页：80	数据页：80 叶页：80
从仅数据锁定到所有页锁定	数据页：80	数据页：80

注释 alter table...lock 将表的所有 fillfactor 存储值设置为 0。

为非聚簇索引存储的 *fillfactor* 值

每个非聚簇索引由一个单独的 sysindexes 行表示。这些命令为两个非聚簇索引存储不同的值：

```
sp_chgattribute "titles.ncl_ix", "fillfactor", 90
sp_chgattribute "titles.pubid_ix", "fillfactor", 75
```

表 3-5 显示当上述 sp_chgattribute 命令用于存储 fillfactor 值时， reorg rebuild 命令对 DOL 锁定表的影响。

表 3-5：存储的 fillfactor 值在 reorg rebuild 期间的作用

reorg rebuild	无聚簇索引	聚簇索引	非聚簇索引
DOL 锁定表	数据页：80	数据页：50 叶页：80	ncl_ix 叶页：90 pubid_ix 叶页：75

使用 *sorted_data* 和 *fillfactor* 选项

当要排序的数据已按索引键指定的顺序排列时，使用 create index 的 sorted_data 选项。这可以使 create clustered index 跳过数据排序、重新分配，以及重建表的数据页步骤。

例如，如果批量复制到表中的数据已按聚簇索引键的顺序排列，则使用 sorted_data 选项创建索引而不执行排序。如果数据不需要复制到新页，则不应用 fillfactor。但使用其它 create index 选项可能仍要求复制。

请参见第 95 页的 “对已排序的数据创建索引”。

减少行转移

如果应用程序允许插入包含空值或短的可变长度字符字段的行，并且这些行的长度在后续更新时增加，可为 DOL 锁定表指定所需行宽。设置所需行宽以减少行转移。

例如， pubs2 数据库中的 titles 表有许多 *varchar* 列和允许空值的列。此表的最大行宽是 331 字节，平均行宽 （如 optdiag 报告）是 184 字节，但可插入少于 40 字节的行，因为许多列允许空值。在 DOL 锁定表中插入短行然后将其更新可能导致行转移。

请参见第 38 页的 “data-only-locked 堆表”。

为含有可变长度列的表设置所需行宽，可使用：

- exp_row_size 参数，位于 create table 语句中。
- 现有表的 sp_chgattribute。
- 服务器范围的缺省值，使用配置参数 default exp_row_size percent。此值应用于带可变长度列的所有表，除非使用 create table 或 sp_chgattribute 明确地设置行宽或指示将在数据页上完全填满行。

如果为所有页锁定表指定所需行宽值，该值将存储在 sysindexes 中，但在插入和更新期间不应用该值。如果表稍后转换为仅数据锁定，则在转换进程中以及对所有后继插入和更新， Adaptive Server 都会应用 exp_row_size。 exp_row_size 值适用于整个表。

exp_row_size 的缺省值、最小值和最大值

表 3-6 显示所需行宽的最小值和最大值及特殊值 （0 和 1）的含义。

表 3-6：所需行宽的有效值

exp_row_size 值	最小值、最大值和特殊值
最小值	以下两项中的较大值： • 2 字节 • 所有固定长度列之和
最大值	最大数据行长度
0	使用服务器范围的缺省值
1	完全填满所有页；不为扩展行保留空间

不能为只有固定长度列的表指定所需行宽。根据定义，接受空值的列是可变长度列，因为当它们为空时是零长度。

缺省值

如果创建带可变长度列的 DOL 锁定表时没有指定所需行宽或 0 值，Adaptive Server 将使用由配置参数 default exp_row_size percent 为有可变长度列的所有表指定的空间大小。

有关 default exp_row_size 如何影响数据页空间的信息，请参见第 55 页的“设置服务器范围的缺省所需行宽”。使用 sp_help 可查看表中定义的列长度。

使用 *create table* 指定所需行宽

以下 create table 语句指定所需行宽为 200 字节：

```
create table new_titles (
      title_id    tid,
      title       varchar(80) not null,
      type        char(12),
      pub_id      char(4) null,
      price       money null,
      advance     money null,
      total_sales int null,
      notes       varchar(200) null,
      pubdate     datetime,
      contract    bit                              )
lock datapages
with exp_row_size = 200
```

添加或更改所需行宽

使用 sp_chgattribute 可添加或更改表所需的行宽。例如，要将 new_titles 表的所需行宽设置为 190，请输入：

```
sp_chgattribute new_titles, "exp_row_size", 190
```

要将表行宽从当前的显式值切换为 default exp_row_size percent，请输入：

```
sp_chgattribute new_titles, "exp_row_size", 0
```

要完全填满页，而不为扩展行保留空间，可设置值为 1。

用 sp_chgattribute 改变所需行宽不会立即影响现有数据的存储。应用新值：

- 对表创建聚簇索引或运行 reorg rebuild 时。当行被复制到新的数据页时，应用所需行宽。

  如果增加 exp_row_size 并重新创建聚簇索引或运行 reorg rebuild，表的新副本可能需要更多的存储空间。

- 下次页受到数据修改的影响时。

设置服务器范围的缺省所需行宽

default exp_row_size percent 保留用于扩展更新的页大小百分比。缺省值为 5%，即为所有包括可变长度列的 DOL 锁定表的每个数据页留出百分之五的可用空间。由于 DOL 锁定表的数据页上有 2002 字节可用，则该缺省值为行扩展留出 100 字节。以下命令设置缺省值为 10%：

```
sp_configure "default exp_row_size percent", 10
```

将 default exp_row_size percent 设置为 0 表示没有为任何显式设置（使用 create table 或 sp_chgattribute）所需行宽的表保留用于扩展更新的空间。

如果已使用 create table 或 sp_chgattribute 指定表的所需行宽，该值优先于服务器范围的设置值。

显示表的所需行宽

使用 sp_help 可以显示表的所需行宽：

```
sp_help titles
```

如果值为 0 且表有可空或长度可变的列，则使用 sp_configure 显示服务器范围的缺省值：

```
sp_configure "default exp_row_size percent"
```

下面的查询显示数据库中所有用户表的 exp_rowsize 列的值：

```
select object_name(id), exp_rowsize
from sysindexes
where id > 100 and (indid = 0 or indid = 1)
```

选择表的所需行宽

仅当行在插入表后进行扩展时，设置所需行宽才有助于减少转移的行数目。正确设置所需行宽意味着：

- 应用程序只产生很小比例的转移行。
- 向所需行宽过度分配空间不会浪费数据页上的空间。

使用 *optdiag* 检查转移的行

对于已包含数据的表，使用 optdiag 可显示表的统计信息。“数据行宽”显示平均数据行长度，包括行开销。以下 titles 表的 optdiag 输出样本显示有 12 个转移的行和 184 字节的平均数据行宽：

```
Statistics for table:                "titles"

     Data page count:                655
     Empty data page count:          5
     Data row count:                 4959.000000000
     Forwarded row count:            12.000000000
     Deleted row count:              84.000000000
     Data page CR count:             0.000000000
     OAM + allocation page count:    6
     Pages in allocation extent:     1
     Data row size:                  184.000000000
```

使用 optdiag 可以检查表的转移行的数目，以确定 exp_row_size 的设置是否会减少由应用程序产生的转移行的数目。

请参见《性能和调优系列：利用统计分析改进性能》中的第 2 章“统计表和用 optdiag 显示统计信息”。

查询 *systabstats* 以查看转移的行

systabstats 表的 forwrowcnt 列存储表的转移行的数量。要显示对象 ID 大于 100 的所有用户表中转移行的数目和平均行大小，请使用以下查询：

```
select objectname = object_name(id),
    partitionname = (select name from syspartitions p
      where p.id = t.id and p.indid = t.indid)
  , forwrowcnt, datarowsize
  , exprowsize = (select i.exp_rowsize from sysindexes i
     where i.id = t.id and i.indid = t.indid)
into #temptable
from systabstats t
where id > 100 and indid IN (0,1)

exec sp_autoformat #temptable
```

注释　转移的行数将在内存中更新，管家任务会定期将这些数据刷新到磁盘。

使用 SQL 查询 systabstats 表，首先使用 sp_flushstats 确保最近的统计信息可用。显示值前， optdiag 会将统计信息刷新到磁盘。

max_rows_per_page 到 *exp_row_size* 的转换

如果为所有页锁定表设置了 max_rows_per_page 值，该值将用于在运行 alter table...lock 命令期间计算所需行宽。公式如表 3-7 所示。

表 3-7：max_rows_per_page 到 exp_row_size 的转换

max_rows_per_page 的值	exp_row_size 的值
0	default exp_row_size percent 设置的百分比值
1 - 254	以下两项中的较小值： • 最大行宽 • （逻辑页大小）— （页标题开销） / max_rows_per_page

例如，如果将配置为 2K 页 （最大定义行宽为 300 字节）的服务器上的所有页锁定表的 max_rows_per_page 设置为 10，则在表变更为使用仅数据锁定后， exp_row_size 值将为 200 (2002/10)。

如果将 max_rows_per_page 设置为 10，但定义的最大行宽只有 150，则所需行宽值将设置为 150。

监控和管理使用所需行宽的表

设置表的所需行宽后，使用 optdiag 或 systabstats 查询来确定应用程序生成的转移行数目。如果转移行的数目高到会影响应用程序性能，则运行 reorg forwarded_rows。 reorg forwarded_rows 使用短事务并且影响较小，因此可以在应用程序活动时运行它。

请参见 《系统管理指南，卷 2》中的第 9 章 “使用 reorg 命令”。

可以对各个分区监控转移的行，并在转移行数目较大的分区上运行 reorg forwarded rows。请参见 《参考手册：命令》。

如果应用程序继续生成大量转移的行，应考虑使用 sp_chgattribute 增加表的所需行宽。

您可能要允许转移行占一定百分比。如果运行 reorg 来清除转移行将不会引发应用程序的并发性问题，或者可以在非高峰期运行 reorg，则允许存在少量的转移行将不会导致严重的性能问题。

设置表的所需行宽会增加存储空间量和读取一组行所需的 I/O 数量。如果因增加存储空间使 I/O 数量增加很多，则允许进行行转移或偶尔运行 reorg 可以降低对总体性能的影响。

为已转移行和插入留出空间

设置 reservepagegap 值可减少存储分段，因而也会降低维护活动 （如运行 reorg rebuild 并为某些表重建索引）的频率。 DOL 锁定表的优良性能要求表使用的页、扩充和分配单元上有优良的数据聚簇。

只要附近有空间用于存储转移行和按索引键顺序插入的行，物理存储中的数据和索引页的聚簇程度就将保持很高。当需要分配其它页时，使用 reservepagegap 空间管理属性可保留空白页以进行扩展。

在表上创建聚簇索引或在运行 reorg rebuild 后，行和页的集群比通常为 1.0，或非常接近 1.0。但是，未来的数据修改可能引起行转移，并可能要求分配附加的数据和索引页来存储插入的行。

可以为所有页锁定表和 DOL 锁定表的数据和索引层页设置保留页间距。

扩充分配命令和 *reservepagegap*

扩充分配表示以八的倍数分配页，而不是一次分配一页。这会减少日志记录活动，因为只写入一个日志记录而非八个。

执行扩充分配的命令有：select into、 create index、 reorg rebuild、 bcp、 alter table...lock 以及 alter table...unique 和 primary key 约束选项，因为这些约束可以创建索引。添加、删除或修改列，或更改表分区方案的 alter table 命令有时也需要执行表复制操作。缺省情况下，上述所有命令都使用扩充分配。

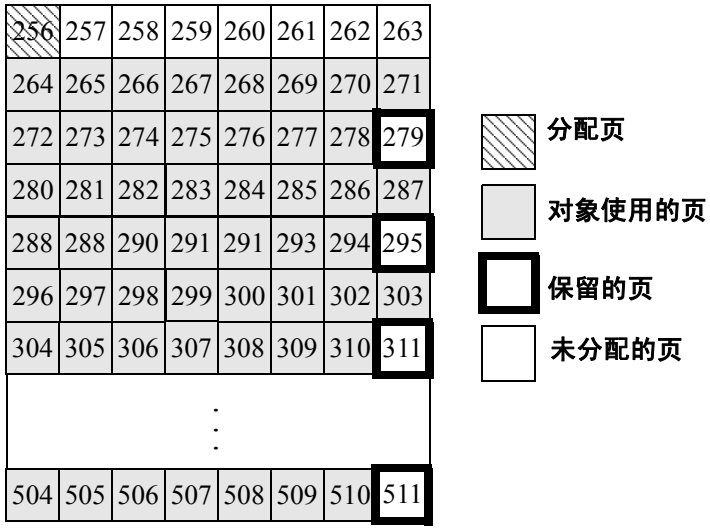
指定页的 reservepagegap 值，该值指示空白页与填充页的比率。例如，如果指定 reservepagegap 值为 8，则使用扩充分配的操作将填充七页，第八页将保留为空。

扩充分配操作不使用每个分配单元的第一页，因为此页用于存储分配页。例如，如果对一个大表创建聚簇索引且不指定保留页间距，则每个分配单元有七个未分配的空白页， 248 个已使用的页和一个分配页。Adaptive Server 可以将这七个空白页用于行转移和插入表操作，这有助于将带聚簇索引的转移行和插入保留在同一分配单元上。使用 reservepagegap 在每个分配单元上保留额外的空白页。

请参见 《Transact-SQL 用户指南》中的第 12 章 “创建表的索引”，了解有关何时使用 reservepagegap 的信息。

图 3-1 说明使用 reservepagegap 值 16 对表创建聚簇索引后分配单元的外观。不使用与分配单元共享第一个扩充的页，也不会将这些页分配给表。279、 295 和 311 页是分配到表的扩充上的未使用的页。

图 3-1：创建聚簇索引后保留的页



使用 *create table* 指定保留页间距

以下 create table 命令指定 reservepagegap 的值为 16：

```
create table more_titles (
      title_id    tid,
      title       varchar(80) not null,
      type        char(12),
      pub_id      char(4) null,
      price       money null,
      advance     money null,
      total_sales int null,
      notes       varchar(200) null,
      pubdate     datetime,
      contract    bit
    )
lock datarows
with reservepagegap = 16
```

对 more_titles 表执行扩充分配的任何操作要为每 15 个填充页留出 1 个空白页。在分区表中， reservepagegap 值适用于所有分区。

reservepagegap 的缺省值为 0，表示未保留空间。

使用 *create index* 指定保留页间距

以下命令将非聚簇索引页的 reservepagegap 指定为 10：

```
create index type_price_ix
on more_titles(type, price)
with reservepagegap = 10
```

可使用创建索引的 alter table...constraint 选项 （primary key 和 unique）指定 reservepagegap 值。分区表中本地索引的 reservepagegap 值适用于所有本地索引分区。

以下示例创建唯一约束：

```
alter table more_titles
add constraint uniq_id unique (title_id)
with reservepagegap = 20
```

更改 *reservepagegap*

要将 titles 表的保留页间距更改为 20，请输入：

```
sp_chgattribute more_titles, "reservepagegap", 20
```

以下命令将索引 title_ix 的保留页间距设置为 10：

```
sp_chgattribute "titles.title_ix",
    "reservepagegap", 10
```

sp_chgattribute 只改变系统表中的值；运行此过程不会使数据在数据页中移动。改变表的 reservepagegap 会影响将来的存储，具体如下：

- 当数据被批量复制到表中时，保留页间距被应用到所有最新的分配空间，但不影响现有页的存储。
- 复制表数据以创建新版表的任何命令都会在该操作的数据复制阶段应用保留页间距。例如，使用 reorg rebuild 或 alter table 更改表的锁定或分区方案，或对需要复制数据的方案进行任何更改，二者均适用于保留页间距。
- 当创建聚簇索引时，为表存储的保留页间距值将应用于数据页。

如下命令期间，对索引页应用保留页间距：

- alter table...lock，将重建索引。
- 当使用 alter table 更改表的锁定或分区方案，或更改任何需要复制数据的方案时，在 reorg rebuild 期间的索引重建阶段。
- create clustered index 和 alter table 命令，它们在重建非聚簇索引时，创建聚簇索引。

reservepagegap 示例

以下示例说明在运行 alter table 和 reorg rebuild 命令期间如何应用 reservepagegap。

仅为表指定的 *reservepagegap*

以下命令为表指定一个 reservepagegap，但不指定 create index 命令中的值：

```
sp_chgattribute titles, "reservepagegap", 16
create clustered index title_ix on titles(title_id)
create index type_price on titles(type, price)
```

表 3-8 显示运行 reorg rebuild 或删除并创建聚簇索引时应用的值。

表 3-8：有表级存储值时应用的 reservepagegap 值

命令	所有页锁定表	DOL 锁定表
create clustered index 或聚簇索引重建 （由于 reorg rebuild）	数据页和索引页：16	数据页：16 索引页：0 （填充扩充）
非聚簇索引重建	索引页：0 （填充扩充）	索引页：0 （填充扩充）

对于带聚簇索引的所有页锁定表， reservepagegap 对数据页和索引页均适用。对于 DOL 锁定表， reservepagegap 适用于数据页，但不适用于聚簇索引页。

为聚簇索引指定的 *reservepagegap*

如下命令为表和聚簇索引指定不同的 reservepagegap 值，并为非聚簇 type_price 索引指定一个值：

```
sp_chgattribute titles, "reservepagegap", 16
create clustered index title_ix on titles(title)
    with reservepagegap = 20
create index type_price on titles(type, price)
    with reservepagegap = 24
```

表 3-9 显示此序列命令的作用。

表 3-9：应用于索引页的 reservepagegap 值

命令	所有页锁定表	DOL 锁定表
create clustered index 或聚簇索引重建（由于 reorg rebuild）	数据页和索引页：20	数据页：16 索引页：20
非聚簇索引重建	索引页：24	索引页：24

对于所有页锁定表，使用 create clustered index 指定的 reservepagegap 对于数据页和索引页均适用。对于 DOL 锁定表，使用 create clustered index 指定的 reservepagegap 仅适用于索引页。如果表具有存储的 reservepagegap 值，此值适用于数据页。

为 *reservepagegap* 选择值

根据以下条件选择 reservepagegap 的值：

- 表是否有聚簇索引，
- 插入表的比率，
- 出现在表中的转移行的行数，以及
- 重新创建聚簇索引或运行 reorg rebuild 命令的频率。

正确配置 reservepagegap 后，可留出足够的页用于分配表和索引的新页，以便在常规索引维护任务的时间间隔期间，表、聚簇索引和非聚簇叶级页的集群比保持较高值。

监控 *reservepagegap* 设置

使用 optdiag 可以检查表的集群比和转移行的数目。集群比下降可能还表明运行 reorg 命令可改善性能：

- 如果聚簇索引的数据页集群比很低，则运行 reorg rebuild 或删除并重新创建聚簇索引。
- 如果索引页集群比很低，则删除并重新创建非聚簇索引。

要降低使用其运行 reorg 命令以保持集群比的频率，请在运行 reorg rebuild 前略微增加 reservepagegap。

请参见《性能和调优系列：利用统计分析改进性能》中的第 2 章“统计表和用 optdiag 显示统计信息”。

reservepagegap 和 *sorted_data* 选项

按索引键顺序对已存储在数据页上的表创建聚簇索引时， sorted_data 选项禁止为未分区表按键顺序复制数据页。可在 create clustered index 命令中指定 reservepagegap 选项，以在表使用的扩充上留出空白页，为以后的扩展留出空间。有规则确定哪一个选项生效。不能使用 sp_chgattribute 来更改 reservepagegap 值，且这两个选项的优势无法兼得。

如果使用 create clustered index 指定两个选项：

- 在未分区的所有页锁定表中，如果使用 create clustered index 指定的 reservepagegap 值与 sysindexes 中存储的值相匹配，则优先选择 sorted_data 选项。因为没有复制数据页，所以不应用 reservepagegap。如果在 create clustered index 命令中指定的 reservepagegap 值与 sysindexes 中存储的值不同，将复制数据页，且在该命令中指定的 reservepagegap 值适用于这些复制的页。
- 在 DOL 锁定表中，使用 create clustered index 值指定的 reservepagegap 值仅适用于索引页。不复制数据页。

除 reservepagegap 外， create clustered index 的其它选项可能要求排序，这会导致 sorted_data 选项被忽略。有关详细信息，请参见第 95 页的“对已排序的数据创建索引”。

特别要指出的是，以下注释与 reservepagegap 的使用有关：

- 在分区表中，需要复制数据页的任何 create clustered index 命令均执行并行排序，然后按排序顺序复制数据页，当这些页复制到新扩充时应用 reservepagegap 值。
- 只要 sorted_data 选项没有被其它 create clustered index 选项代替，将扫描表以确定数据是否按键的顺序存储。索引在扫描期间建立，并不执行排序。

表 3-10 显示这些规则如何应用。

表 3-10：reservepagegap 和 sorted_data 选项

	分区表	未分区表
所有页锁定表		
create index with sorted_data 和匹配的 reservepagegap 值	不复制数据页；在扫描页时建立索引。	不复制数据页；在扫描页时建立索引。
create index with sorted_data 和其它 reservepagegap 值	执行并行排序，当页按排序顺序存储在新位置时应用 reservepagegap。	复制数据页，在复制页时应用 reservepagegap 并建立索引；不执行任何排序。
DOL 锁定表		
create index with sorted_data 和任何 reservepagegap 值	reservepagegap 仅适用于索引页；不复制数据页。	reservepagegap 仅适用于索引页；不复制数据页。

匹配选项和目标

要重新分配表的数据页，为以后的扩展留出空间，请使用以下方法：

- 对于所有页锁定表，删除并重新创建聚簇索引，而不使用 sorted_data 选项。如果存储在 sysindexes 中的值不符合需要，请使用 create clustered index 指定所需的 reservepagegap。
- 对于 DOL 锁定表，使用 sp_chgattribute 将表的 reservepagegap 设置为所需的值，然后删除并重新创建聚簇索引，而不使用 sorted_data 选项。为表存储的 reservepagegap 应用于数据页。如果在 create clustered index 命令中指定 reservepagegap，则它只应用于索引页。

要创建聚簇索引而不复制数据页：

- 对于所有页锁定表，使用 sorted_data 选项指定 reservepagegap，而不使用 create clustered index。或者，指定一个与 sysindexes 中存储的值相匹配的值。
- 对于 DOL 锁定表，使用 sorted_data 选项。如果在 create clustered index 命令中指定 reservepagegap 值，则它只应用于索引页且不会引起数据页复制。

要在执行批量复制操作、运行 select into 命令或另一个使用扩充分配的命令之后使用 sorted_data 选项，请在复制数据之前为数据页设置所需的 reservepagegap 值，或在 select into 命令中加以指定。因为不需要复制数据页，一旦数据页被分配并填充，以下命令只将 reservepagegap 应用到索引页：

```
create clustered index title_ix
on titles(title_id)
with sorted_data, reservepagegap = 32
```

在所有页锁定表上使用 *max_rows_per_page*

设置每页的最大行数可减少对所有页锁定表和索引的争用。多数情况下，转换表以使用仅数据锁定方案更为可取。如果因为某种原因无法更改锁定方案并且所有页锁定表或索引上存在争用问题，则设置 max_rows_per_page 值可能有助于提高性能。

索引或数据页上行较少时，锁争用的机会便会降低。由于键分布在更多页，所需页不是另外程序所需的页面的可能性更大。要更改每页的行数，请调整表和索引的 fillfactor 或 max_rows_per_page 值。

fillfactor （由 sp_configure 或 create index 定义）确定 Adaptive Server 在对现有数据创建新索引时，对每一数据页的填充程度。由于 fillfactor 有助于减少页面拆分，索引上的排它锁也最大可能地减少，从而提高了性能。但是， fillfactor 值不是通过对数据的后续更改来保持的。max_rows_per_page （由 sp_chgattribute、 create index, create table 或 alter table 定义）与 fillfactor 相似，但 Adaptive Server 在数据更改时保持 max_rows_per_page 值。

使用 fillfactor 或 max_rows_per_page 减少每页行数的相关开销包括读取相同数目数据页需要更多 I/O、从数据高速缓存获得相同性能需要更多内存，以及需要更多锁。此外，在将数据插入表中时，将表的 max_rows_per_page 设置为较小的值可以增加页面拆分数。

减少锁争用

在 create table、create index 或 alter table 命令中指定的 max_rows_per_page 值限制数据页、聚簇索引的叶页或非聚簇索引的叶页允许的行数。这将减少对锁的争用，并将改善被经常访问的表的并发性。

max_rows_per_page 适用于堆表的数据页或索引的叶页。与 fillfactor 不同（在创建表或索引后不会保留后者），Adaptive Server 在添加或删除行时会保留 max_rows_per_page 值。

如下命令创建 sales 表并将每页最大行数限制为四：

```
create table sales
      (stor_id          char(4)       not null,
      ord_num           varchar(20)   not null,
      date              datetime      not null)
      with max_rows_per_page = 4
```

如果使用 max_rows_per_page 值创建表，然后对此表创建聚簇索引而不指定 max_rows_per_page，聚簇索引将从 create table 语句继承 max_rows_per_page 值。使用 max_rows_per_page 创建聚簇索引将更改表数据页的值。

索引与 *max_rows_per_page*

max_rows_per_page 的缺省值为 0，可对完整数据页创建聚簇索引，对完整叶页创建非聚簇索引，并在聚簇索引和非聚簇索引的 B 树索引内保留适当空间量。

对于堆表和聚簇索引， max_rows_per_page 范围为 0 - 256。

对于非聚簇索引， max_rows_per_page 的最大值为与叶页相适的索引行数 （不超过 256）。要确定最大值，请从页大小减去 32 （页头的大小），然后再将差额除以索引键的大小。以下语句计算非聚簇索引 max_rows_per_page 的最大值：

```
select (@@pagesize - 32)/minlen
    from sysindexes
    where name = "indexname"
```

select into 和 *max_rows_per_page*

缺省情况下， select into 不结转基表的 max_rows_per_page 值，但使用 max_rows_per_page 值 0 创建新表。不过，您可以将 max_rows_per_page 选项添加到 select into，以此来指定非零值。

将 *max_rows_per_page* 应用到现有数据

有多种方法可将 max_rows_per_page 值应用到现有数据：

- 如果表有聚簇索引，则删除并使用其它 max_rows_per_page 值重新创建索引。
- 使用 sp_chgattribute 更改 max_rows_per_page 值，然后使用 reorg rebuild 重建整个表及其索引。例如，若要将 authors 表的 max_rows_per_page 值更改为 1，请输入：

  ```
  sp_chgattribute authors, "max_rows_per_page", 1
  go
  reorg rebuild authors
  go
  ```

- 使用 bcp 重新填充表并执行以下操作：
  a 复制出表数据。
  b 截断表。
  c 使用 sp_chgattribute 设置 max_rows_per_page 值。
  d 将数据复制回去。

第 4 章

表和索引大小

本章阐述如何确定表和索引的当前大小及如何为空间计划估计表大小。

了解表和索引的大小对理解查询和系统行为很重要。在调优工作的几个阶段，需要用到描述大小的数据：

- 了解特定查询计划的 statistics io 报告。《性能和调优系列：利用统计分析改进性能》中的第 1 章 “使用 set statistics 命令”介绍了如何使用 statistics io 检查执行的 I/O。
- 理解优化程序对查询计划的选择。 Adaptive Server 基于成本的优化程序可对每个可能的访问方法所需的物理和逻辑 I/O 做出估计，然后选择最经济的方法。如果认为某查询计划反常，可使用 dbcc traceon(302) 确定优化程序作出这一决策的原因。此输出包括页数估计。
- 根据数据库对象的大小和对象上预期的 I/O 模式确定对象的放置。可通过在各物理设备数据之间分配数据库对象来实现对磁盘读写的平均分配，从而改善性能。对象放置在第 1 章 “控制物理数据放置”中进行了说明。
- 理解性能更改。如果对象增长，它们的性能特性会发生更改。例如，某个表的使用率很高，且通常被 100% 高速缓存。如果该表就其高速缓存而言增长得过大，就能使访问该表的查询性能急剧下降。这对于需要多次扫描的连接更是如此。
- 执行容量规划。不论是设计新系统还是规划扩展现有系统，都必须要了解空间要求，以便规划所需的物理磁盘和内存。
- 理解 Adaptive Server Monitor 的输出和有关物理 I/O 的 sp_sysmon 报告的输出。

确定表和索引的大小

Adaptive Server 带有多个工具，用于提供表或索引当前大小的信息，并可预测其未来大小：

- optdiag 显示表和索引的大小及许多其它统计信息。请参见 《性能和调优系列：利用统计分析改进性能》中的第 2 章“统计表和用 optdiag 显示统计信息”
- sp_spaceused 报告现有表和索引的当前大小。
- sp_estspace 可在给定行数参数的情况下预测表及其索引的大小。

也可使用本章中提供的公式计算表和索引的大小。sp_spaceused 和 optdiag 报告空间的实际使用情况。本章中提供的其它方法可用于估计大小。

对于分区表， sp_helpartition 报告存储在表的每个分区上的页数。请参见《Transact-SQL 用户指南》中的第 10 章 “对表和索引进行分区”。

数据修改对对象大小的影响

一段时间后，随机分配的数据修改对一组表的影响是产生平均满度约为 75% 的数据页和索引页。其主要因素有：

- 当插入行时，如果该行将被放置在一个有聚簇索引的所有页锁定表的页上，且该页无法容纳插入的行，则该页将被拆分，拆分成的两页的填满程度约为 50%。
- 从堆或有聚簇索引的表中删除行时，页上使用的空间将减少。有的页的行数非常少，有的甚至只有一行。
- 进行了一些删除或页面拆分后，向有聚簇索引的表中插入行可能会填满被拆分或行被删除的页。

当需要将行插入完整的索引页时也会发生页面拆分，因此除非定期删除并重新创建索引页，否则它们的平均满度也往往为 75% 左右。

使用 *optdiag* 显示对象大小

optdiag 命令显示表、索引和列的统计信息，包括表和索引的大小。如果要执行查询调优， optdiag 可提供查看所需的所有统计信息的最佳工具。以下是 pubtune 数据库中 titles 表的样本报告：

```
Table owner:                          "dbo"
Statistics for table:                 "titles"
     Data page count:                 662
     Empty data page count:           10
     Data row count:                  4986.0000000000000000
     Forwarded row count:             18.0000000000000000
     Deleted row count:               87.0000000000000000
     Data page CR count:              86.0000000000000000
     OAM + allocation page count:     5
     First extent data pages:         3
Data row size:                        238.8634175691937287
```

请参见《性能和调优系列：利用统计分析改进性能》中的第 2 章“统计表和用 optdiag 显示统计信息”。

optdiag 的优点

optdiag 的优点：

- 可显示数据库中所有表或单个表的统计信息。
- optdiag 的输出包含有助于了解查询开销（如索引高度和平均行长度）的附加信息。
- 经常用于其它调优任务，所以这些报告可能随手可用。

optdiag 的缺点

optdiag 的主要缺点是它会产生大量输出。如果只需要一条信息（如表的页数），使用其它方法会更迅速，系统开销也更低。

使用 *sp_spaceused* 显示对象大小

系统过程 sp_spaceused 可读取存储在对象的 OAM 页上的值，以提供对象使用空间的快速报告。

```
sp_spaceused titles
name         rowtotal reserved   data      index_size  unused
------------ -------- ---------- --------- ----------- --------
titles       5000        1756 KB   1242 KB     440 KB     74 KB
```

rowtotal 值有时可能不准确，因为并不是所有 Adaptive Server 进程都更新 OAM 页上的这个值。命令 update statistics、 dbcc checktable 和 dbcc checkdb 可更正 OAM 页上的 rowtotal 值。表 4-1 说明 sp_spaceused 输出的标题。

表 4-1：sp_spaceused 输出

列	含义
rowtotal	报告估计的行数。该值从 OAM 页上读取。尽管不总是很精确，但与 select count(*) 相比，这种估计的速度要快得多，且引发的争用也较少。
reserved	报告供表及其索引使用的保留页。它包括分配给对象的扩充中已使用的和未使用的页。它是 data、index_size 和 unused 的合计值。
data	报告表所使用的页的千字节数。
index_size	报告索引使用的页的总千字节数。
unused	报告分配给对象的扩充中未使用的页的千字节数，包括对象索引中未使用的页的千字节数。

要分别报告索引大小，应使用：

```
sp_spaceused titles, 1
 index_name         size       reserved   unused
 ------------------ ---------- ---------- ----------
 title_id_cix       14 KB      1294 KB    38 KB
 title_ix           256 KB     272 KB     16 KB
 type_price_ix      170 KB     190 KB     20 KB

name         rowtotal reserved   data      index_size  unused
------------ -------- ---------- --------- ----------- --------
titles       5000        1756 KB   1242 KB     440 KB     74 KB
```

对于所有页锁定表上的聚簇索引， size 值代表根索引页和中间索引页使用的空间。 reserved 值包括索引大小以及未使用和已使用的数据页。

sp_spaceused 语法中的 “1” 指示应输出详细的索引信息。它与索引 ID 或其它信息无关。

sp_spaceused 的优点

sp_spaceused 的优点：

- 因为它只使用表和索引 OAM 页中的值返回结果，所以无需过多的 I/O 和锁定就可提供快速报告。
- 它显示保留的用于对象扩展、而当前又尚未用于存储数据的空间的大小。
- 它提供有关索引、 text、 image 和 Java 行外列存储大小的详细报告。

注释　使用 sp_helpartition 可报告各分区中的页数。 sp_helpartition 的报告没有 sp_spaceused 详细，但可以大致提供分区使用的空间量。在 Adaptive Server 15.0.2 及更高版本中， sp_spaceusage 提供各种主题的详细信息，包括表在索引级和分区级使用的空间，以及分段使用的空间。

请参见 《Adaptive Server 参考手册：过程》，了解所有系统过程的详细信息。

sp_spaceused 的缺点

sp_spaceused 的缺点有：

- 它可能会报告不准确的总行数和空间使用情况。
- 输出只以千字节为单位，而绝大多数查询调优活动以页数为计量单位。不过，您可以使用 sp_spaceusage 报告指定的任何单位的信息。

使用 *sp_estspace* 估计对象大小

sp_spaceused 和 optdiag 报告空间的实际使用情况。sp_estspace 可帮助规划表和索引以后的扩展。此过程使用系统表 （sysobjects、 syscolumns 和 sysindexes）中的信息来确定数据和索引行的长度。您只要提供表名和希望在表中出现的行数， sp_estspace 即可估计出表及任何现有索引的大小。它并不查看表中数据的实际大小。

可使用 sp_estspace：

- 创建表 （如果表不存在）。
- 创建表上的任何索引。
- 执行过程，估计表将能容纳的行数。

其输出可报告表及索引每一级的页数和字节数。

下例中估计了 titles 表的大小，该表有 500,000 行、一个聚簇索引和两个非聚簇索引：

```
   sp_estspace titles, 500000
name                 type         idx_level Pages    Kbytes
-------------------- ------------ --------- -------- --------
titles               data                 0    50002   100004
title_id_cix         clustered            0      302      604
title_id_cix         clustered            1        3        6
title_id_cix         clustered            2        1        2
title_ix             nonclustered         0    13890    27780
title_ix             nonclustered         1      410      819
title_ix             nonclustered         2       13       26
title_ix             nonclustered         3        1        2
type_price_ix        nonclustered         0     6099    12197
type_price_ix        nonclustered         1       88      176
type_price_ix        nonclustered         2        2        5
type_price_ix        nonclustered         3        1        2

Total_Mbytes
-----------------
  138.30

name                 type         total_pages  time_mins
-------------------- ------------ ------------ ------------
title_id_cix         clustered          50308          250
title_ix             nonclustered       14314           91
type_price_ix        nonclustered        6190           55
```

sp_estspace 还允许指定填充因子、可变长度字段和文本字段的平均大小及 I/O 速度。有关详细信息，请参见 《参考手册：过程》。

注释 sp_estspace 输出的索引创建次数不会影响并行排序的结果。

sp_estspace 的优点

sp_estspace 的优点如下：

- 为规划初始容量及表和索引的扩展提供有效方法。
- 可帮助估计索引级的数目。
- 可帮助估计未来的磁盘空间、高速缓存空间和内存需求。

sp_estspace 的缺点

sp_estspace 的缺点如下：

- 返回的大小只是估计值，可能因填充因子、页面拆分、可变长度字段的实际大小及其它因素而与实际大小有出入。
- 依据磁盘速度、扩充 I/O 缓冲区的使用和系统负载情况，索引创建次数可有很大不同。

使用公式估计对象大小

此处讨论的公式可帮助您估计数据库中表和索引的未来大小。包含可变长度字段的表和索引中每一行的开销量比只包含固定长度字段的表中每一行的开销量大，因此需要有两套公式。

这一过程包括计算每一行的数据和开销的字节数，然后用该数除以某数据页上可用的字节数。每一页都需要一些开销，这就限制了数据可用的字节数：

- 对于所有页锁定表，页开销为 32 字节，在 2K 字节的页上可留出 2016 字节供数据使用。
- 对于 DOL 锁定表，页开销为 46 字节，可留出 2002 字节供数据使用。

要进行最准确的估计，可用 round down 除法计算每页的行数 （不跨页拆分行），用 round up 除法计算页数。

可影响存储大小的因素

使用空间管理属性可增加表或索引所需的空间。请参见第 89 页的 “空间管理属性的影响”和第 90 页的 “max_rows_per_page”。

如果表中包含 text 或 image 数据类型或 Java 行外列，请在计算中使用 16 作为列的大小（存储在行中的文本指针的大小）。然后参见第 90 页的 “LOB 页”，了解如何计算实际 text 或 image 数据所需的存储空间。

受以下两个因素影响， DOL 锁定表上的索引数可能要比公式预测的少：

- 重复键只存储一次，继之以该键的行 ID 列表。
- 非叶级上的键压缩；仅存储足以区别于相邻键的键。这对于在使用长字符键时减少大小特别有效。

如果配置参数 page utilization percent 设置为小于 100， Adaptive Server 可能会在填充已分配扩充上的所有页前分配新扩充。这样做不会改变对象使用的页数，但会在分配给对象的扩充中留出空页。

数据类型的存储大小

数据类型的存储大小如表 4-2所示：

表 4-2：Adaptive Server 数据类型的存储大小

数据类型	大小
char	定义的大小
nchar	定义的大小 * *@@ncharsize*
unichar	n**@@unicharsize (@@unicharsize equals 2)*
univarchar	实际字符数 **@@unicharsize*
varchar	实际字符数
nvarchar	实际字符数 * *@@ncharsize*
binary	定义的大小
varbinary	数据大小
int	4
smallint	2
tinyint	1
float	4 或 8，取决于精度
double precision	8
real	4
numeric	2 - 17，取决于精度和标度
decimal	2 - 17，取决于精度和标度
money	8
smallmoney	4
datetime	8
smalldatetime	4
bit	1
text	16 字节 + 2K *使用的页数
image	16 字节 + 2K *使用的页数
timestamp	8

numeric 或 decimal 列的存储大小取决于它的精度。 1 或 2 位数列的最小存储要求为 2 个字节。精度每增加 2 位，存储大小增加 1 字节，最大可达到 17 字节。

任何定义为 NULL 的列都被视为可变长度列，因为它们涉及与可变长度列有关的开销。

下例中的所有计算都基于 varchar、univarchar、nvarchar 和 varbinary 数据的最大大小 — 定义的列大小。同时假设列被定义为 “非空”。

公式中使用的表和索引

本例说明对一个有 9,000,000 行的表的计算：

- 固定长度列大小合计为 100 字节。
- 可变长度列大小合计为 50 字节；共有 2 个可变长度列。

该表有两个索引：

- 一个固定长度列上的聚簇索引，大小为 4 字节
- 一个包含以下列的组合非聚簇索引：
  - 一个固定长度列，大小为 4 字节
  - 一个可变长度列，大小为 20 字节

所有页锁定表和 DOL 锁定表需要使用不同的公式，因为它们的页及每一行的开销量不同：

- 有关所有页锁定表使用的公式，请参见第 78 页的 “为所有页锁定表计算表及聚簇索引大小”。
- 有关 DOL 锁定表使用的公式，请参见第 84 页的 “计算 DOL 锁定表的大小”。

为所有页锁定表计算表及聚簇索引大小

所有页锁定表的公式和示例分为几套步骤在下面列出。 1 至 6 步概述了有聚簇索引的所有页锁定表的计算，并提供了表大小和索引树大小。 7 至 12 步概述了对非聚簇索引所需空间的计算。所有的公式均使用可变长度字段的最大大小。步骤如下：

1 第 79 页的 “计算数据行的大小”

2 第 80 页的 “计算数据页的数量”

3 第 80 页的 “计算聚簇索引行的行宽”

4 第 81 页的 “计算聚簇索引页的数目”

5 第 81 页的 “计算索引页总数”

6 第 81 页的 “计算分配开销和总页数”

7 第 82 页的 “计算叶索引行行宽”

8 第 83 页的 “计算索引中叶页的数量”

9 第 83 页的 “计算非叶行行宽”

10　第 83 页的 “计算非叶页页数”

11　第 84 页的 “计算非叶索引页的总数”

12　第 84 页的 “计算分配开销和总页数”

这些公式显示了计算表及聚簇索引大小的方法。如果表中没有聚簇索引，可跳过第 3、 4 和 5 步。在第 2 步中计算数据页页数时，跳至第 6 步与 OAM 页数相加。

optdiag 输出包括数据行和索引行的平均长度。如果想使用平均长度而不使用最大长度，这些值可用于数据和索引行长度。

计算数据行的大小

存储可变长度数据的行比只存储有固定长度数据的行要求的开销更多，因此计算数据行的行宽有两个不同的公式。

仅固定长度列

如果表只包含固定长度列，且都定义为 NOT NULL，请使用：

公式	
4	（开销）
+	所有固定长度列的字节之和
	= 数据行大小

某些可变长度列

如果表包含任何可变长度列或允许空值的列，请使用此公式。

由于示例中的表包含可变长度列，因此算式显示在右列。

公式		示例
4	（开销）	4
+	所有固定长度列的字节之和	+ 100
+	所有可变长度列的字节之和	+ 50
	= 小计	154
+	（小计/ 256）+ 1 （开销）	1
+	可变长度列的列数 + 1	3
+ 2	（开销）	2
	= 数据行大小	160

计算数据页的数量

公式

2016 /数据行行宽 = 每页数据行行数

行数/每页行数 = 所需数据页页数

示例

2016 / 160	=	每页 12 个数据行
9,000,000 / 12	=	750,000 个数据页

计算聚簇索引行的行宽

包含可变长度列的索引行比只包含固定长度值的索引行要求的开销更多。如果所有键的长度固定，使用第一个公式。如果键包含可变长度列或允许空值，则使用第二个公式。

仅固定长度列

示例中的聚簇索引只有固定长度键。

公式

	5	（开销）
+		固定长度索引键字节之和
		= 聚簇行行宽

示例

	5	
+	4	
	9	

某些可变长度列

	5	（开销）
+		固定长度索引键字节之和
+		可变长度索引键字节之和
		= 小计

+		（小计/ 256） + 1 （开销）
+		可变长度列的列数 + 1
+	2	（开销）
		= 聚簇索引行行宽

除法运算 （小计/ 256）的结果向下舍入。

计算聚簇索引页的数目

公式			示例		
（2016 /聚簇行行宽） - 2	=	每页聚簇索引行行数	(2016 / 9) - 2	=	222
行数/每页聚簇索引行行数	=	下一级索引页页数	750,000 / 222	=	3379

如果 “下一级索引页页数”的结果大于 1，重复下面的除法步骤，将商作为下一次的被除数，直到商等于 1 为止，则表明已到达索引的根级：

公式

上一级索引页页数　/　每页聚簇索引行行数　=　下一级索引页页数

示例

3379 / 222	=	16 个索引页 （级别 1）
16 / 222	=	1 个索引页 （级别 2）

计算索引页总数

将每级的页数相加，以确定索引中的总页数：

公式			示例	
索引级	页		页	行
2			1	16
1	+		+ 16	3379
0	+		+ 3379	750000
		索引页总数	3396	

计算分配开销和总页数

每个表和表中的每个索引都有一个对象分配映射 (OAM)。单个 OAM 页可容纳 2,000 到 63,750 个数据页或索引页的分配映射。在绝大多数情况下，所需的 OAM 页数接近最小值。要计算表的 OAM 页数，应使用：

公式			示例		
保留的数据页数/ 63,750	=	最少 OAM 页数	750,000 / 63,750	=	12
保留的数据页数/ 2000	=	最多 OAM 页数	750,000 / 2000	=	376

要计算索引的 OAM 页数，应使用：

公式			示例		
保留的索引页页数/ 63,750	=	最少 OAM 页数	3396/ 63,750	=	1
保留的索引页页数/ 2000	=	最多 OAM 页数	3396 / 2000	=	2

所需总页数

最后，将 OAM 页数与之前的总计值相加，以确定所需的总页数：

公式			示例	
	最小	最大值	最小	最大值
聚簇索引页			3396	3396
OAM 页	+	+	1	2
数据页	+	+	750000	750000
OAM 页	+	+	12	376
总计			753409	753773

计算叶索引行行宽

包含可变长度列的索引行比只包含固定长度值的索引行要求的开销更多。

仅固定长度键

如果索引只包含固定长度键，且都定义为 NOT NULL，请使用：

公式	
7	（开销）
+	固定长度键之和
	= 叶索引行行宽

某些可变长度键

如果索引包含任何可变长度键或定义为 NULL 的列，请使用：

公式		示例	
9	（开销）		9
+	固定长度键的长度之和	+	4
+	可变长度键的长度之和	+	20
+	可变长度键数 + 1	+	2
	= 小计		35
+	（小计/ 256）+ 1 （开销）	+	1
	= 叶索引行行宽		36

计算索引中叶页的数量

公式			示例		
（2016 /叶行行宽）	=	每页叶索引行行数	2016 / 36	=	56
表的行数/每页叶行行数	=	下一级索引页页数	9,000,000 / 56	=	160,715

计算非叶行行宽

公式		示例	
	叶索引行行宽		36
+ 4	开销	+	4
	= 非叶行行宽		40

计算非叶页页数

公式			示例
（2016 /非叶行行宽） - 2	=	每页非叶索引行行数	(2016 / 40) - 2 = 48

如果第 8 步中的叶页页数大于 1，请重复下面的除法步骤，将商作为下一次的被除数，直到商等于 1 为止，则表明已到达索引的根级：

公式				
上一级索引页页数	/	每页非叶索引行行数	=	下一级索引页页数

示例	
160715 / 48 = 3349	索引页，级别 1
3349 / 48 = 70	索引页，级别 2
70 / 48 = 2	索引页，级别 3
2 / 48 = 1	索引页，级别 4 （根级）

计算非叶索引页的总数

将每级的页数相加，以确定索引中的总页数：

索引级	页			页	行
4				1	2
3	+		+	2	70
2	+		+	70	3348
1	+		+	3349	160715
0	+		+	160715	9000000
		使用的 2K 数据页总数		164137	

计算分配开销和总页数

公式			示例		
索引页页数/ 63,750	=	最少 OAM 页数	164137 / 63,750	=	3
索引页页数/ 2000	=	最多 OAM 页数	164137 / 2000	=	83

所需总页数

将 OAM 页数与第 11 步的总计值相加，以确定索引页的总页数：

公式			示例	
	最小	最大值	最小	最大值
非聚簇索引页			164137	164137
OAM 页	+	+	3	83
总计			164140	164220

计算 DOL 锁定表的大小

下面的公式和示例显示如何计算表和索引的大小。以下示例使用与前一个示例相同的列大小和索引。所有的公式均使用可变长度字段的最大大小。有关说明，请参见第 78 页的 “公式中使用的表和索引”。

DOL 锁定表的公式分为两套步骤：

- 步骤 1 — 3 概述 DOL 锁定表的计算。紧接步骤 3 的示例说明对一个有 9,000,000 行的表的计算。
- 步骤 4 — 8 概述索引所需的空间计算，紧接的示例使用了有 9,000,000 个行的表。

optdiag 输出包括数据行和索引行的平均长度。如果想使用平均长度而不使用最大长度，这些值可用于数据和索引行长度。

步骤如下：

1　第 85 页的 “计算数据行的大小”

2　第 86 页的 “计算数据页的数量”

3　第 86 页的 “计算分配开销和总页数”

4　第 86 页的 “计算索引行行宽”

5　第 87 页的 “计算索引中叶页的数量”

6　第 87 页的 “计算索引中非叶页的数量”

7　第 88 页的 “计算非叶索引页的总数”

8　第 88 页的 “计算分配开销和总页数”

计算数据行的大小

存储可变长度数据的行比只存储有固定长度数据的行要求的开销更多，因此计算数据行的行宽有两个不同的公式。

仅固定长度列

如果表只包含定义为 NOT NULL 的固定长度列，请使用：

	6	（开销）
+		所有固定长度列的字节之和
		数据行大小

注释　DOL 锁定表必须为每个行留有空间才能存储 6 字节的转移行 ID。如果 DOL 锁定表的行不足 10 字节，在插入每行后会将该行填充为 10 字节。这种改变只影响数据页，而不会影响索引，也不会影响所有页锁定表。

某些可变长度列

如果表包含可变长度列或允许空值的列，请使用：

公式			示例	
	8	（开销）		8
+		所有固定长度列的字节之和	+	100
+		所有可变长度列的字节之和	+	50
+		可变长度列列数* 2	+	4
		数据行大小		162

计算数据页的数量

公式

2002 /数据行行宽 = 每页数据行行数
行数/每页行数 = 所需数据页页数

在此步骤的开始部分，每页的行数向下舍入：

示例

2002 / 162	=	每页 12 个数据行
9,000,000 / 12	=	750,000 个数据页

计算分配开销和总页数

分配开销

每个表和表中的每个索引都有一个对象分配映射 (OAM)。 OAM 存储在分配给表或索引的页上。单个 OAM 页可容纳 2,000 到 63,750 个数据页或索引页的分配映射。在绝大多数情况下，所需的 OAM 页数接近最小值。要计算表的 OAM 页数，应使用：

公式			示例		
保留的数据页数/ 63,750	=	最少 OAM 页数	750,000 / 63,750	=	12
保留的数据页数/ 2000	=	最多 OAM 页数	750,000 / 2000	=	375

所需总页数

将 OAM 页数与之前的总计值相加，以确定所需的总页数：

公式			示例	
	最小	最大值	最小	最大值
数据页	+	+	750000	750000
OAM 页	+	+	12	375
总计			750012	750375

计算索引行行宽

对于 DOL 锁定表的聚簇索引和非聚簇索引，应使用这些公式。

包含可变长度列的索引行比只包含固定长度值的索引行要求的开销更多。

仅固定长度键

如果索引只包含定义为 NOT NULL 的固定长度键，请使用：

9	（开销）
+	固定长度键之和
	索引行行宽

某些可变长度键

如果索引包含任何可变长度键或允许空值的列，请使用：

公式		示例
9	（开销）	9
+	固定长度键的长度之和	+ 4
+	可变长度键的长度之和	+ 20
+	可变长度键的数目* 2	+ 2
	索引行行宽	35

计算索引中叶页的数量

公式

2002 /索引行行宽 = 每页的行数
表中行数/每页行数 = 叶页页数

示例

2002 / 35 = 57 （每页非聚簇索引行行数）
9,000,000 / 57 = 157,895 个叶页

计算索引中非叶页的数量

公式

叶页页数　/ 每页索引行行数　= 下一级页数

如果上面的下一级索引页页数大于 1，重复下面的除法步骤，将商作为下一次的被除数，直到商等于 1 为止，则表明已到达索引的根级：

公式

上一级索引页页数　/ 每页非叶索引行行数　= 下一级索引页页数

示例

157895/57 = 2771	索引页，级别 1
2770 / 57 = 49	索引页，级别 2
48 / 57 =1	索引页，级别 3

计算非叶索引页的总数

将每级的页数相加，以确定索引中的总页数：

公式			**示例**	
索引级	页		页	行
3			1	49
2	+		49	2771
1	+		2771	157895
0	+		157895	9000000
		使用的 2K 页总数	160716	

计算分配开销和总页数

公式

索引页页数/ 63,750 = 最小 OAM 页数
索引页页数/ 2000 = 最多 OAM 页数

示例

160713 / 63,750 = 3 （最小值）
160713 / 2000 = 81 （最大值）

所需总页数

将 OAM 页数与第 8 步的总计值相加，以确定索引页的总页数：

公式			**示例**	
	最小	最大值	最小	最大值
非聚簇索引页			160716	160716
OAM 页	+	+	3	81
总计			160719	160797

影响对象大小的其它因素

除随时间推移数据修改施加的影响外，其它可影响对象大小和大小估计的因素有：

- 空间管理属性
- 计算使用的是平均行宽还是最大行宽
- 文本行非常短
- 使用 text 和 image 数据

空间管理属性的影响

fillfactor、 exp_row_size、 reservepagegap 和 max_rows_per_page 的值可影响对象的大小。

填充因子

创建索引时，应用为 create index 指定的 fillfactor。向表执行插入操作期间并不维护 fillfactor。如果已使用 sp_chgattribute 为索引存储了 fillfactor，在用 alter table 命令和 reorg rebuild 重新创建索引时应使用这个值。 fillfactor 的主要功能是在索引页上留出空间，以减少页面拆分。fillfactor 值如果非常小则可能导致表或索引所需的存储空间显著增大。

请参见第 47 页的 “减少索引维护”，了解有关设置 fillfactor 值的详细信息。

exp_row_size

设置表的预期行宽可增加所需的存储空间。如果表中有许多行的行宽比预期的行宽短，设置此值并运行 reorg rebuild 或更改锁定方案可增加表所需的存储空间。但之前使用 max_rows_per_page 的表的空间使用情况应大致保持不变。

请参见第 53 页的 “减少行转移”，了解有关设置 exp_row_size 值的详细信息。

reservepagegap

为表或索引设置 reservepagegap 会于执行扩充分配命令时在分配给对象的扩充上留下空页。将 reservepagegap 设置为小值将增加空页的数目，并将数据扩展到更多的扩充上，因此在诸如 create index 或 reorg rebuild 的命令刚执行完时，所需的额外空间最大。行转移和向表中插入将填充这些保留页。

请参见第 58 页的 “为已转移行和插入留出空间”。

max_rows_per_page

max_rows_per_page 值 （由 create index、 create table、 alter table 或 sp_chgattribute 指定）会限制数据页上的行数。

要想在使用 max_rows_per_page 时计算出正确的值，应使用 max_rows_per_page 值，或使用在第 80 页的 “计算数据页的数量”和第 83 页的 “计算索引中叶页的数量”中计算出的每页的数据行行数，以较小者为准。

请参见第 66 页的 “在所有页锁定表上使用 max_rows_per_page”。

很短的行

对于所有页锁定表， Adaptive Server 无法在页上存储 256 个以上的数据行或索引行。即使行极短，数据页的最少页数也为：

行数/ 256 = 所需数据页页数

LOB 页

每个 text 或 image 或 Java 行外列可在数据类型为 varbinary(16) 的数据行中存储一个 16 字节的指针。已初始化的每列需要至少 2K（一个数据页）的存储空间。

列存储隐式空值，即数据行中的文本指针仍为 NULL，且没有为该值初始化文本页，从而节约了 2K 的存储空间。

如果某一 LOB 列定义为允许空值，且该行是用包括该列的 NULL 的 insert 语句创建的，则不会初始化该列，也不会分配存储空间。

如果用 update 对 LOB 列进行了任何更改，则分配文本页。向列中放置实际数据的插入或更新操作会初始化该页。如果随后将该列设置为 NULL，则单个页将保持已分配状态。

每个 LOB 页大约可存储 1800 个字节的数据。要估计一个特定条目将使用的页数，应使用此公式：

数据长度/ 1800 = 2K 页页数

在任何情况下，结果都应向上舍入；即 1801 字节的数据长度需要两个 2K 的页。

由于一些 LOB 页为列中其它页链存储指针信息，数据所需空间的总计值可能会比计算出的值稍大。在访问 LOB 列时，Adaptive Server 利用此指针信息进行随机访问和预取数据。存储指针信息所需的额外空间取决于存储在列中的数据的大小总计值和数据类型。使用表 4-3 估计为 LOB 列中的数据存储指针信息所需的额外页数。

表 4-3：估计的存储 LOB 列中指针信息所需的额外页数

数据大小和类型	存储指针信息所需的额外页数
400K image	0 到 1 页
700K image	0 到 2 页
5MB image	1 到 11 页
400K 多字节 text	1 到 2 页
700K 多字节 text	1 到 3 页
5MB 多字节 text	2 到 22 页

使用公式估计对象大小的优点

使用公式的优点有：

- 可了解数据和索引存储内部的更多细节。
- 使用公式可灵活地为字符或二进制列指定平均大小。
- 在计算索引大小时，可了解每个索引的级数，这样有助于估计性能。

使用公式估计对象大小的缺点

使用公式的缺点有：

- 估计的精度只能达到对可变长度列的平均大小的估计精度。
- 多步计算很繁琐，遗漏步骤可能会导致错误。
- 对象的实际大小可能与计算的值不同，具体取决于使用情况。

第 5 章

数据库维护

本章介绍维护活动如何影响其它 Adaptive Server 活动的性能，以及如何改善维护任务的性能。

维护活动包括多项任务，例如删除和重新创建索引、执行 dbcc 检查，以及更新表和索引统计信息。所有这些活动都可由服务器上的其它处理工作来完成。

应尽可能在 Adaptive Server 使用率较低时执行维护任务。本章可帮您确定这些活动对单个应用程序的性能以及 Adaptive Server 整体性能的影响。

对表和索引运行 *reorg*

reorg 命令可通过改进表和索引的空间利用率来改善 DOL 锁定表的性能。 reorg 子命令及其使用方法：

- reclaim_space — 在更新缩短的数据行的长度时，清除提交的删除内容和其余空间。
- forwarded_rows — 将转移的行返回到主页。
- compact — 可执行上述两项操作。
- rebuild — 重建整个表或索引。可对所有页锁定表和 DOL 锁定表使用 reorg rebuild。

在对表运行 reorg rebuild 时，会在重建表及其索引的整个过程中锁定该表。当用户不需要访问该表时，可对表安排执行 reorg rebuild 命令。

所有其它 reorg 命令 （包括对索引执行的 reorg rebuild）将一次锁定少量页面，并使用短期的独立事务来执行其工作。可随时运行这些命令。唯一的负面影响可能是对 I/O 非常密集的系统的影响。

有关运行 reorg 命令的详细信息，请参见《系统管理指南，卷 2》的第 9 章 “使用 reorg 命令”。

创建和维护索引

在用户创建索引时，将对所有其他用户锁定表。锁类型取决于索引类型：

- 创建聚簇索引需要一排它表锁来锁定所有表活动。由于聚簇索引中的行按索引键的顺序排列，因此 create clustered index 将对数据页重新排序。
- 创建非聚簇索引需要一共享表锁来锁定更新活动。

配置 Adaptive Server 以加速排序

使用 number of sort buffers 配置参数可以设置高速缓存中可用于容纳输入表页面的缓冲区的数量。另外，并行排序可从执行排序的缓存中的大 I/O 中获益。

请参见 《性能和调优系列：查询处理和抽象计划》的第 5 章 “并行查询处理”。

创建索引后转储数据库

创建索引时， Adaptive Server 会将 create index 事务和页分配写入事务日志，但不会记录对数据和索引页的实际更改。要恢复因创建索引而未转储的数据库，可在装载事务日志转储时，再次执行整个 create index 进程。

如果定期重新创建索引 （例如，在索引中维护 fillfactor），则可能需要在进行常规数据库转储之前安排这些操作短暂运行。

对已排序的数据创建索引

要重新创建聚簇索引，或创建按索引键顺序批量复制到服务器中的数据的聚簇索引，可使用 sorted_data 选项来 create index，以缩短索引创建时间。

由于必须按键顺序排列数据行以创建聚簇索引，所以在不使用 sorted_data 创建聚簇索引时，需要将数据行重新写入一组全新的数据页中。在某些情况下， Adaptive Server 可以跳过排序和复制表的数据行的操作：因素包括表分区和 create index 语句中使用的 on 子句。

在未分区表上创建索引时， sorted_data 和使用下列任何子句都需要复制数据，但不需要排序：

- ignore_dup_row
- fillfactor
- on *segment_name* 子句，可指定与表数据所在段不同的段
- 此 max_rows_per_page 子句，可指定一个与表相关联的值不同的值。

当这些选项和 sorted_data 包括在分区表的 create index 中时，将执行排序步骤、复制数据，并在表的分区上平均分配数据页。

表 5-1：使用选项来创建聚簇索引

选项	分区表	未分区表
未指定选项	并行排序；复制数据，在分区上均匀分配；创建索引树。	并行或非并行排序；复制数据，创建索引树。
仅有 with sorted_data 或者 with sorted_data on *same_segment*	只创建索引树。不执行排序或复制数据。不并行运行。	只创建索引树。不执行排序或复制数据。不并行运行。
with sorted_data 和 ignore_dup_row 或 fillfactor 或 on *other_segment* 或 max_rows_per_page	并行排序；复制数据，在分区上均匀分配；创建索引树。	复制数据并创建索引树。不执行排序。不并行运行。

在最简单的情况下，使用 sorted_data 且未分区表内无其它选项时将检查表行的顺序，并在此单次扫描期间建立索引树。

如果必须复制数据行但不需要执行排序，则单个表扫描检查行的顺序、建立索引树并将数据页复制到单个表扫描中的新位置上。

对于需要大量传递来建立索引的大型表，明显缩短排序时间可降低 I/O 和 CPU 的利用率。

创建用于复制数据行的聚簇索引时，可用空间必须约为表大小的 120%，才能复制数据和存储索引页。

维护索引和列统计信息

索引的直方图和密度值不能如同添加和删除的数据行那样进行维护。数据库所有者必须发出 update statistics 命令来确保统计信息最新。在执行以下操作后运行 update statistics：

- 删除或插入更改索引中键值的倾斜的行。
- 将行添加到之前使用 truncate table 删除其行的表中。
- 更新索引列中的值。
- 插入到包含 IDENTITY 列的任何索引，或任何递增键值。日期列通常具有规则的递增键。

如果 IDENTITY 列或其它递增键是索引中的前导列，则对这些类型的索引运行 update statistics 尤其重要。创建索引时，在表中最后一个键后插入了许多行之后，优化程序只能通知该搜索值位于分布页中的最后一行以外。无法准确地确定有多少行匹配给定值。

注释 更新统计信息失败会严重降低性能。

请参见 《性能和调优系列：利用统计分析改进性能》。

重建索引

重建索引将回收二叉树 （树中的所有叶页距索引的根页的距离都相同）中的空间。由于页被拆分、行被删除，因此索引可能包含很多页，每页中只有很少的几行。此外，如果应用程序对覆盖非聚簇索引和大 I/O 执行扫描，则重建非聚簇索引可通过减少分段来维护大 I/O 的效率。

可通过删除并重新创建索引来重建索引。

出现以下情况时需重建索引：

- 数据和使用模式发生了重大变化。
- 需要大量插入期间，或刚刚完成。
- 排序顺序更改。

- 使用大 I/O 的查询需要进行多于预期的磁盘读取，或 optdiag 报告的集群比较之平常更低。
- 空间使用超出了估计数量，因为大量的数据修改保留了使很多数据和索引页局部已满。
- 空间管理属性 （填充因子、期望行宽和保留页间距）提供的扩展空间已经被插入和更新导致的页面拆分、转移行和分段所填充。
- dbcc 已识别出索引中的错误。

如果重新创建聚簇索引或在 DOL 锁定表或所有页锁定表上运行 reorg rebuild，则会重新创建所有非聚簇索引，这是由于创建聚簇索引会将行移动到其它页中。

当系统活动频率较低时：

- 删除所有索引，允许更有效的批量插入。
- 创建新的索引组，以帮助生成一组报告。

创建或变更数据库

创建或变更数据库是 I/O 密集型操作；因此，可能妨碍其它 I/O 密集型操作。创建数据库时， Adaptive Server 会将 model 数据库复制到新数据库中，然后初始化所有分配页并清除数据库页。

要加快数据库创建速度或将其对其它进程的影响降至最小，请使用以下方法：

- 如果正在恢复数据库；即正准备发出 load database 命令，请使用 create database...for load 选项。

  如果不使用 for load 创建数据库， Adaptive Server 将复制 model，然后初始化所有分配单元。

  使用 for load 时， Adaptive Server 会初始化分配单元，直到完成装载。然后，只初始化未改动的分配单元。如果要装载非常大的数据库转储，这样可节约大量时间。
- 如果可能，应在非高峰期间创建数据库。

create database 和 alter database 在清除数据库页时执行并发并行 I/O。设备数量由 number of large i/o buffers 配置参数限定。此参数的缺省值为 6，允许一次在 6 个设备上执行并行 I/O。

单个 create database 和 alter database 命令一次最多可使用 32 个此类缓冲区。load database、磁盘镜像和部分 dbcc 命令也使用这些缓冲区。

使用缺省值 6，如果指定多于 6 个设备，则立即开始写入前 6 个设备。当每个设备的 I/O 都已完成， 16K 缓存区将用于在命令中列出的其它设备。下列示例命名了 10 个单独的设备：

```
create database hugedb
      on dev1 = 100,
      dev2 = 100,
      dev3 = 100,
      dev4 = 100,
      dev5 = 100,
      dev6 = 100,
      dev7 = 100,
      dev8 = 100
log on logdev1 = 100,
      logdev2 = 100
```

使用这些缓冲区的操作期间，缓冲区数量超出范围时，会向日志发送一条消息。对于上述 create database 命令，此信息指示 create database 在清除其它设备上的页之前，开始使用所有大 I/O 缓冲区清除前 6 个磁盘上的设备，然后等待其完成：

```
CREATE DATABASE: allocating 51200 pages on disk 'dev1'
CREATE DATABASE: allocating 51200 pages on disk 'dev2'
CREATE DATABASE: allocating 51200 pages on disk 'dev3'
CREATE DATABASE: allocating 51200 pages on disk 'dev4'
CREATE DATABASE: allocating 51200 pages on disk 'dev5'
CREATE DATABASE: allocating 51200 pages on disk 'dev6'
01:00000:00013:1999/07/26 15:36:17.54 server  No disk i/o buffers are
available for this operation. The total number of buffers is
controlled by the configuration parameter 'number of large i/o
buffers'.
CREATE DATABASE: allocating 51200 pages on disk 'dev7'
CREATE DATABASE: allocating 51200 pages on disk 'dev8'
CREATE DATABASE: allocating 51200  pages on disk 'logdev1'
CREATE DATABASE: allocating 51200  pages on disk 'logdev2'
```

注释 在 Adaptive Server 12.5.0.3 及以后版本中， create database、 alter database、 load database 和 dbcc checkalloc 所用的大 I/O 缓冲区的大小是一个分配 (256 页)，而不是较早版本中的一个扩充 (8 页)。因此服务器需要对大缓冲区分配更多内存。例如，早期版本中需要 8 页的内存的磁盘缓冲区现在需要 256 页内存。

备份和恢复

所有 Adaptive Server 备份都由 Backup Server 执行。备份体系结构使用客户端 / 服务器范例，将 Adaptive Servers 作为 Backup Server 的客户端。

本地备份

Adaptive Server 通过远程过程调用发送本地 Backup Server 指令，并告知 Backup Server 要转储或装载的页面、要使用的备份设备及其它选项。Backup Server 执行所有磁盘 I/O。

Adaptive Server 不读取或发送转储和装载数据，它只发送指令。

远程备份

Backup Server 也支持备份至远程计算机。对于远程转储和装载，本地 Backup Server 执行与数据库设备相关的磁盘 I/O，并通过网络向将数据存储到转储设备上的远程 Backup Server 发送数据。

联机备份

可在数据库处于活动状态时执行备份。显然，这种处理影响其它事务，但不应因此而不愿经常备份关键数据库，否则将无法满足系统的可靠性要求。

请参见 《系统管理指南，卷 2》，以了解备份和恢复策略的完整讨论。

使用阈值来防止日志空间用完

如果数据库限制了日志空间，您偶尔会达到*最后机会阈值*，则安装另一个可提供充足时间来执行事务日志转储的阈值。用尽日志空间会严重影响性能。日志空间被释放之前，用户不能执行任何数据修改命令。

尽量缩短恢复时间

可通过更改 recovery interval 配置参数来尽量缩短恢复时间。每个数据库 5 分钟的缺省值适用于大多数安装。仅在功能性需求指示需要更快速的恢复期间时，才可减少此值。减少此值会增加所需的 I/O 量。

请参见 《性能和调优系列：基础知识》的第 5 章 “内存使用和性能”。

恢复速度也可能受到 housekeeper free write percent 配置参数的值的影响。此参数的缺省值允许服务器的管家清洗任务在服务器的空闲周期中，将脏缓冲区写入磁盘，只要磁盘 I/O 增量未超过 20%。

恢复顺序

恢复期间，系统数据库最先恢复。然后，用户数据库按数据库 ID 的顺序恢复。

批量复制

如果表无聚簇索引且已启用 select into/ bulkcopy，批量复制到 Adaptive Server 上的表时运行速度将达到最快。如果未启用此选项， slow bcp 将用于包含任何索引或活动触发器的表。

fast bcp 仅记录无索引的表的页分配。由于 fast bcp 不更新每个数据插入的索引，且不记录对索引页的更改，因此可以节省时间。但如果对含索引的表使用 fast bcp，它将记录索引更新。

fast bcp 会自动用于带触发器的表。要使用 slow bcp，可在执行复制时禁用 select into/bulk copy 数据库选项。

使用快速批量复制：

1 使用 sp_dboption 设置 select into/bulkcopy/pllsort 选项。记住在批量复制操作完成后禁用此选项。

2 删除所有聚簇索引。完成批量复制后重新创建它们。

注释 复制期间无需停用触发器。

进行快速批量复制期间，不强制执行规则，但必须使用缺省值。

由于未记录对数据进行的更改，因此会在执行快速批量复制操作后即执行 dump database。由于无法从事务日志转储中恢复未记录的数据更改，因此在数据库中执行快速批量复制会阻止使用 dump transaction。

并行批量复制

为获得最佳性能，可使用快速批量复制将数据复制到分区表中。为每个批量复制会话指定要在其中驻留数据的分区。

如果输入文件已按顺序排序，则可按顺序将数据批量复制到分区中，并避免在创建聚簇索引时执行排序步骤。

请参见 《Transact-SQL 用户指南》的第 10 章 “对表和索引进行分区”，以了解分步过程。

批处理和批量复制

如果在快速批量复制期间指定批处理大小，由于在快速批量复制期间只记录页分配而不记录数据更改，因此每次执行新的批处理时必须从新数据页开始。复制 1000 行，批处理大小为 1，在事务日志中需要 1000 个数据页和 1000 个分配记录。

如果使用较小的批处理大小来帮助检测输入文件中的错误，则可能需要选择与数据页适合的行数相当的批处理大小。

慢速批量复制

缺省情况下，如果表带有聚族索引、索引或触发器，且已启用 select into/bulk copy， Adaptive Server 将缺省使用 slow bcp。

对于慢速批量复制：

- 不必设置 select into/bulkcopy。
- 不强制使用规则且不引发触发器，但要使用缺省值。
- 所有数据更改同页分配一样都将被记录。
- 行被复制进来时将更新索引并记录索引更改。

改善批量复制性能

提高批量复制性能的其它方法：

- 设置 trunc log on chkpt 选项以防止塞满事务日志。如果数据库拥有在日志填满时可自动转储日志的阈值过程，则可缩短事务转储时间

  每个批处理都是一个单独的事务，因此，如果不指定批处理大小，则设置 trunc log on chkpt 将不会提高性能。

- 如果执行的大批量复制较多，请将 number of pre-allocated extents 配置参数设置为较高值。

  请参见 《系统管理指南，卷 1》的第 5 章 “设置配置参数”。

- 查找最佳网络包大小。

  请参见 《性能和调优系列：基础知识》的第 2 章 “网络和性能”。

在大表中替换数据

如果替换大表中的所有数据，应使用可执行较少日志记录操作的 truncate table，而不是 delete。只记录页解除分配。

1 截断表。

2 删除表上的所有索引。

3 装载数据。

4 重建索引。

请参见 《参考手册：命令》。

向表中添加大量数据

向大表中添加其容量的 10%-20% 或更多数据时，可删除非聚簇索引、装载数据，然后重新创建非聚簇索引。

对于超大表，由于空间限制可能需要保留聚簇索引。 Adaptive Server 在创建聚簇索引时必须生成表副本。在很多情况下，如果表非常大，则使用保留索引进行慢速批量复制所需的时间将少于执行快速批量复制和重建聚簇索引所需的时间。

使用分区和多个批量复制进程

如果要将数据装载到无索引的表中，则可在该表上创建分区并对每个分区使用 bcp 会话。

请参见《实用程序指南》的第 4 章“使用 bcp 从 Adaptive Server 传出数据或向 Adaptive Server 传送数据”。

对其他用户的影响

向内或向外批量复制大表可能影响其他用户的响应时间。如果可能：

- 安排在非高峰期间执行批量复制操作。
- 使用快速批量复制，因为只进行少量记录且只需少量 I/O。

数据库一致性检查程序

可定期使用 dbcc 运行数据库一致性检查。如果备份损坏的数据库，则该备份无用。由于 dbcc 必须获取其检查的对象的锁，因此， dbcc 会影响性能。

请参见《系统管理指南，卷 2》的第 10 章“检查数据库一致性”，了解有关 dbcc 和锁定的信息，以及有关如何尽量降低 dbcc 对用户应用程序的影响的更多信息。

使用 *dbcc tune (cleanup)*

Adaptive Server 在处理每项任务后，都会执行冗余内存清除检查（作为最终的完整性检查）。在高吞吐量环境中，跳过此清除错误检查会使性能稍有提高。要关闭错误检查，输入：

```
dbcc tune(cleanup,1)
```

最终清除可释放任务可能占用的所有内存。如果关闭错误检查，但发生内存错误，可通过输入以下命令重新启用该检查：

```
dbcc tune(cleanup,0)
```

对螺旋锁使用 *dbcc tune*

当您发现因螺旋锁争用而导致的缩放问题时，可以使用 des_bind 来提高为热对象保留对象描述符的服务器的可伸缩性。绑定对象的描述符永不释放。即使只绑定几个常用对象的描述符，也可以减少总体元数据螺旋锁争用情况并提高性能。

```
dbcc tune(des_bind, <dbid>, <objname>)
```

若要删除绑定，请使用：

```
dbcc tune(des_unbind, <dbid>, <objname>)
```

注释 若要从数据库解除对象绑定，该数据库必须处于“单用户模式”。

请勿在以下项目中使用 des_bind：

- 系统数据库 （如 master 和 tempdb）中的对象上
- 在系统表中

des_bind 不具持久性，因此，每次重启服务器时都必须重新发出所有绑定命令。

确定维护活动的可用空间

几个维护操作需要空间来复制表的数据页：

- create clustered index
- alter table...lock
- 一些用于添加或修改列的 alter table 命令
- alter table...partition by
- 表中的 reorg rebuild

多数情况下，这些命令也需要空间来重新创建索引，所以您必须确定：

- 表及其索引的大小
- 存储表的段上的可用空间量
- 表及其索引的空间管理属性设置

空间要求概述

复制表行的命令也会对表重新创建全部索引。因此需要足够的可用空间来容纳表的完整副本和全部索引的副本。

这些命令不能估计需要多少空间。如果命令耗尽表或表索引使用的段空间，则该命令将停止，并发出错误消息。对于大表，启动该命令后数分钟甚至数小时内都可能出现这种情况。

在表及其索引所使用的段上需要可用空间，具体如下：

- 表使用的段上的可用空间必须至少等于：
  - 表的大小，加上
  - 约为表的 20% 的大小（如果表有聚簇索引，并且您正从所有页锁定更改为仅数据锁定）。
  - 非聚簇索引使用的段上的可用空间必须至少等于索引的大小。

DOL 锁定表的聚簇索引在数据页上有一叶级。如果将带有聚簇索引的表从所有页锁定更改为仅数据锁定，则生成的聚簇索引将需要更多空间。所需的额外空间取决于索引键的大小。

检查空间使用情况和可用空间

一个简单原则是，复制表及其索引所需空间等于表及其索引使用的当前空间，再额外加上约 20% 的空间。然而：

- 如果数据修改创建了大量部分填满的页，则复制表所需的空间可能要小于当前空间大小。
- 如果表的空间管理属性已更改，或者 fillfactor 或 reservepagegap 所需的空间已被数据修改内容填满，那么复制表所需的空间可能更大。
- 对较大的数据类型添加列或修改列，需要为复制提供更大的空间。

还需要日志空间。由于 Adaptive Server 将 reorg rebuild 作为单个事务来处理，因此所需的日志空间量可能较大，尤其当正在重建的表包含多个非聚簇索引时更是如此。每个非聚簇索引都需要日志空间，因此，必须拥有足够的日志空间才能创建全部索引。

检查用于表和索引的空间

要查看表及其索引的大小，可使用：

```
sp_spaceused titles, 1
```

有关估计聚簇索引大小的信息，请参见第 84 页的“计算 DOL 锁定表的大小”。

检查段的空间

始终会将表复制到其当前所在段的可用空间内，并且也会在索引当前所在段中重建这些索引。创建聚簇索引的命令可指定一个段。表和聚簇索引的副本在目标段上创建。

若要确定段上可用页的数量，请使用 sp_helpsegment。 sp_helpsegment 的最后一行显示可用于段上的可用页总数。

以下命令输出 default 段 （当未显式指定段时，会在其中存储对象）的段信息：

```
sp_helpsegment "default"
```

sp_helpsegment 报告段上索引的名称。如果不知道表的段名，则使用 sp_help 和表名。索引的段名也由 sp_help 报告。

检查空间管理属性的空间要求

如果对空间管理属性值进行重大改变，则表副本可能会比原来的表大很多或小很多。空间管理属性的设置存储在 sysindexes 表中，并可通过 sp_help 和 sp_helpindex 显示。此输出显示 titles 表的空间管理属性：

```
exp_row_size reservepagegap fillfactor max_rows_per_page
------------ -------------- ---------- -----------------
         190             16         90                 0
```

sp_helpindex 会输出以下报告：

```
index_name           index_description
    index_keys
    index_max_rows_per_page index_fillfactor index_reservepagegap
    ----------------------- ---------------- --------------------
title_id_ix          nonclustered located on default
    title_id
                         0               75                    0
title_ix             nonclustered located on default
    title
                         0               80                   16
```

```
type_price            nonclustered located on default
    type, price
                          0                90                    0
```

应用于表的空间管理属性

在复制步骤中，表的空间管理属性使用如下：

- 如果为表指定了期望行宽值，且将锁定方案从 allpages 锁改变为仅数据锁定，则期望行宽将在复制数据行时被应用到这些数据行。

  如果未设置所需行宽，但表具有 max_rows_per_page 值，则会计算出所需行宽并使用该值。

  否则，为表分配的所有页都将使用通过配置参数 default exp_row_size percent 所指定的缺省值。
- reservepagegap 则作为扩充分配给表。
- 如果 sp_chgattribute 已被用于保存表的 fillfactor 值，则在复制行时会将其应用到新数据页。

应用于索引的空间管理属性

重建索引时，将应用索引的空间管理属性，如下所示：

- 如果 sp_chgattribute 已用于保存索引的 fillfactor 值，则在重新创建索引时将应用这些值。
- 如果为索引设置 reservepagegap 值，则重新创建索引时将应用这些值。

估计空间管理属性的影响

表 5-2 显示如何估计设置空间管理属性的影响。

表 5-2：空间管理属性对空间使用的影响

属性	公式	示例
fillfactor	如果页面当前被完全填满，要求 (100/fillfactor) * *num_pages*	值为 75 的 fillfactor 需要的页数是当前页数的 1.33 倍；如果表有 1,000 页，则将增加到 1,333 页。
reservepagegap	如果当前填充了扩充，则要使用 1/reservepagegap 增加空间	reservepagegap 为 10 将增加 10% 的空间； 1,000 页的表将增加到 1,100 页。
max_rows_per_page	转换到仅数据锁定时，将转换为 exp_row_size	请参见第 108 页的表 5-3。

属性	公式	示例
exp_row_size	按照行宽小于 exp_rowsize 的行的数目，以及这些行的平均长度增加	如果 exp_row_size 为 100，并且有 1,000 行长度为 60，则增加的空间为： (100 - 60) * 1000 或 40,000 字节；大约 20 个额外的页。

请参见第 3 章 “设置空间管理属性”。

如果表有 max_rows_per_page 设置，且表由所有页锁定转换为仅数据锁定，则在 alter table...lock 命令将表复制到新位置之前，此值将转换为 exp_row_size 值。

复制期间将强制使用 exp_row_size。表 5-3 显示了这些值的转换方式。

表 5-3：将 max_rows_per_page 转换为 exp_row_size

如果将 *max_rows_per_page* 设置为	将 *exp_row_size* 设置为
0	default exp_row_size percent 设置的百分比值
1 - 254	以下两项中的较小值： • 最大行宽 • 2K 逻辑页 — 2002/max_rows_per_page 值 4K 逻辑页 — 4050/max_rows_per_page 值 8K 逻辑页 — 8146/max_rows_per_page 值 16K 逻辑页 — 16338/max_rows_per_page 值

如果没有足够空间

如果没有足够空间来复制表和重建全部索引，则应确定删除表的非聚簇索引是否会留出足够空间创建表的副本。没有任何非聚簇索引，复制操作只要求表和聚簇索引占用的空间。

不要删除聚簇索引，因为聚簇索引用于排序复制的行，而且在稍后重建此索引时可能需要空间来创建表副本。命令完成后请重新创建非聚簇索引。

第 6 章

临时数据库

本章讨论了与临时数据库有关的性能问题。临时数据库是一种全服务器范围的资源，主要用于处理排序、创建工作表、重新格式化以及存储用户创建的临时表和索引。任何人都可以在临时数据库中创建对象。许多进程在使用这些数据库时都不会有任何提示。

许多应用程序都使用在临时数据库中创建表的存储过程来加快复杂连接的速度，或执行其它不易在单步中执行的复杂数据分析。

临时数据库管理如何影响性能

有效管理临时数据库对 Adaptive Server 的整体性能至关重要。不过，临时表可能增加对 tempdb 大小的要求。使用临时表会极大地影响 Adaptive Server 和应用程序的性能。您不能忽视对临时数据库的管理或将其保留为缺省状态。在许多服务器中，tempdb 是最具动态性的数据库。

通过预先制定计划并考虑以下这些问题，可以避免与临时数据库相关的大多数性能问题：

- 临时数据库经常处于填满状态，导致向用户发出错误消息，而用户必须等到空间可用时才能重新提交查询。
- 临时数据库排序缓慢，且数据库查询性能较不稳定。
- 因锁定系统表而导致经常短暂阻止用户查询创建临时表。
- 大量使用临时数据库中的对象导致从数据高速缓存中清除其它页。

可通过以下方法解决这些问题：

- 配置数目足够多的用户临时数据库。
- 为所有 Adaptive Server 活动适当调整临时数据库的大小。
- 以最佳方式放置临时数据库以减少争用。
- 尽可能减少临时数据库中的资源锁定。
- 将临时数据库绑定到其自身的数据高速缓存。
- 正确配置临时数据库组。
- 将登录名和应用程序绑定到相应的临时数据库或临时数据库组。

使用临时表

在临时数据库中创建的表称为临时表。使用临时数据库可创建不同类型的临时表。临时表的类型包括：

- 散列 (#) 临时表
- 常规用户表
- 工作表

散列 (#) 临时表

散列临时表：

- 仅存在于用户会话期间或临时表创建过程范围内，可以在会话或创建过程结束时手动或自动删除。
- 不能在用户连接间共享
- 是在为会话分配的临时数据库中创建的。

可通过将井号 (“#”) 作为表名的第一个字符来创建散列临时表：

```
create table #temptable (...)
```

或：

```
select select_list
into #temptable ...
```

对临时表创建索引时，这些索引存储所在的会话与分配给散列表所在的临时数据库的会话相同：

```
create index littletableix on #littletable(col1)
```

常规用户表

要在临时表中创建常规用户表，请在 create table 命令中指定数据库名称：

```
create table tempdb..temptable (...)
```

临时数据库中的常规用户表：

- 在会话间可以保持
- 可用于批量复制 (bcp) 操作
- 通过对其授予权限即可共享
- 必须由所有者显式删除，或在 Adaptive Server 重新启动时自动删除

或：

```
select select_list
into tempdb..temptable
```

可以对在临时数据库中创建的常规用户表创建索引：

```
create index tempix on tempdb..temptable(col1)
```

工作表

Adaptive Server 可为分配了 tempdb 的会话创建内部临时表，以用于合并、排序、连接等。这些临时表称为工作表，具有以下特点：

- 始终不能共享
- 命令完成后立即消失

临时数据库

为避免因使用单个临时数据库导致的性能问题，您可以创建多个临时数据库。

Adaptive Server 包含一个由系统创建的临时数据库，名为 tempdb，此数据库是在您安装 Adaptive Server 时在主设备上创建的。

除 tempdb 外， Adaptive Server 还允许用户创建多个临时数据库。用户创建的临时数据库类似于系统 tempdb：这些数据库主要用于创建临时对象，在启动期间会重新创建而不是恢复它们。与 tempdb 不同的是，可以删除用户创建的临时数据库。

多个临时数据库：

- 可减少对系统 tempdb 中系统目录和日志文件的争用
- 可在快速访问设备上进行创建
- 可根据需要进行创建或删除

分配了临时数据库的会话

当客户端建立连接后， Adaptive Server 会向其会话分配临时数据库。Adaptive Server 将此分配了临时数据库的会话作为一个缺省空间，用于为客户端执行的工作创建临时对象 （包括散列临时表和工作表）。系统会始终为会话分配临时数据库，直至该会话连接到客户端。

Adaptive Server 根据以下规则为会话选择临时数据库：

- 如果登录名已存在绑定，则使用该绑定。
- 如果已指定应用程序名且其已存在绑定，则使用该绑定。
- 如果 Adaptive Server 未发现绑定，则使用循环方案从缺省组中分配一个临时数据库。

可指定 Adaptive Server 在某个特定临时数据库中创建对象。例如：

```
create procedure inv_amounts as
     select stor_id, "Total Due" = sum(amount)
     from #tempstores
     group by stor_id
```

使用多个临时数据库

本节讨论如何创建、配置、绑定和选择临时数据库。

创建用户临时数据库

使用 create database 语法中的 temporary database 关键字可创建多个临时数据库：

create temporary database *temporary_database_name* on *device_name*=size log on *device_name*=size

例如，要在 tempdb_device 上创建名为 tempdb_1 的用户临时数据库，请输入：

```
create temporary database tempdb_1 on tempdb_device = 3
log on log_device = 1
```

配置缺省 tempdb 组

Adaptive Server 包含一组名为缺省组的临时数据库。Adaptive Server 在启动会话时会使用循环技术从缺省组（所有临时数据库活动都在该组中执行）中选择一个临时数据库。 Adaptive Server 将此临时数据库分配给会话。 sp_who 会在 tempdbname 列中显示此临时数据库。由于缺省组中的单个临时数据库不执行所有活动，因此 Adaptive Server 可利用循环方案在该组的所有临时数据库之间平均分配负载。

开始时，缺省组仅包含 tempdb。但用户可以在该组中添加多个用户数据库。可使用 sp_tempdb 向缺省组中添加用户数据库。例如，要将 tempdb_1 添加到缺省组中，请使用：

```
sp_temodb "add'", "tempdb_1" , "default"
```

要从缺省组中删除 tempdb_1，请使用：

```
sp_tempdb "drop", "tempdb_1" , "default"
```

请参见 《参考手册：过程》以了解完整的 sp_tempdb 语法。

绑定到组和 tempdb

sp_tempdb. . . 'bind'...'unbind' 系统过程允许您将应用程序或登录名绑定到特定的临时数据库或 tempdb 组，或者将其解除绑定。创建绑定后，当应用程序或登录名连接到服务器时， Adaptive Server 会分配与其绑定的指定临时数据库或临时数据库组。通过绑定，您可以控制针对特定应用程序或登录名的临时数据库分配。

以下示例将登录名 “sa”绑定到缺省组：

```
sp_tempdb 'bind', 'lg', 'sa', 'GR', 'default'
```

以下示例解除对登录名 “sa”的绑定：

```
sp_tempdb 'unbind', 'lg', 'sa'
```

请参见 《参考手册：过程》以了解完整的 sp_tempdb 语法。

将应用程序和登录名绑定到临时数据库

确认临时数据库对应用程序和登录名的要求。通过将这些应用程序和登录名绑定到不同的数据库或缺省组，可在可用的临时数据库之间平均分配负载，从而避免出现目录争用。如果绑定不适当，那么即使有足够数量的临时数据库也无法解决目录争用问题，因为在这种情况下，Adaptive Server 不能在临时数据库之间平均分配负载。请参见第 114 页的 “绑定到组和 tempdb”。

调优系统临时数据库以获得最佳性能

本节讨论与临时数据库相关的配置问题。

放置系统 *tempdb*

在确定 tempdb 的放置位置时：

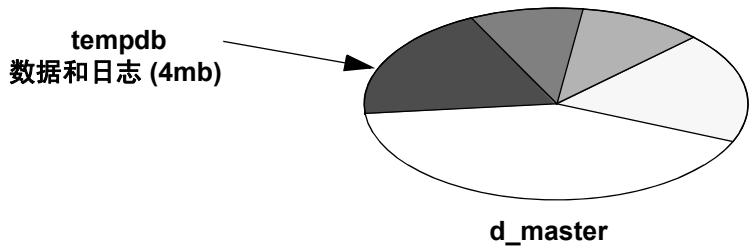
- 应将 tempdb 和重要的应用程序数据库分开放在不同的物理磁盘上。
- 使用可用的最快磁盘。如果您的平台支持固态设备，并且您的应用程序不适合使用 tempdb，则可以使用这些设备。
- 将 tempdb 扩展到其它设备后，从 system、 default 和 logsegment 段中删除主设备。

尽管可以将 tempdb 扩展到 master 数据库所在的设备， Sybase 仍建议您使用单独的设备。而且要记住，使用 Adaptive Server 镜像功能镜像的是逻辑设备而不是数据库。如果镜像主设备，将创建驻留在主设备上的数据库的所有部分的一个镜像。如果镜像使用 serial 进行写入，则频繁使用 tempdb 数据库会对性能产生严重影响。

系统 *tempdb* 的初始分配

安装 Adaptive Server 时，tempdb 的大小为 4MB，且全部位于主设备上，如图 6-1 所示。 tempdb 通常是系统管理需要增大的第一个数据库。服务器上的用户越多，它就需要越大。根据需要，可能希望将 tempdb 分条，使其位于多个设备上。

图 6-1：tempdb 缺省分配



使用 sp_helpdb 来查看 tempdb 的大小和状态。以下示例显示了 tempdb 在安装时的缺省值：

```
                sp_helpdb tempdb
name       db_size  owner   dbid     created                 status
---------- -------- ------ ------ -----------  --------------------
tempdb       2.0 MB      sa      2 May 22, 1999    select into/bulkcopy

device_frag      size        usage     free kbytes
------------ -------- ------------        --------
master          2.0 MB   data and log             1248
```

从 *tempdb* 段中删除主设备

缺省情况下，tempdb 的 system、default 和 logsegment 段在主设备上的分配大小为 4MB。在将新设备分配给 tempdb 时，除非将这些设备添加为专用数据或日志，否则它们会自动划分到这三个段中。为 tempdb 分配第二台设备后，可以将主设备从 default、 system 和 logsegment 段中删除。这样可以确保 tempdb 中的工作表和其它临时表不与其它操作争用主设备资源。

从各段中删除主设备：

1 如果尚未进行此类操作，则将 tempdb 变更到其它设备上。例如：

```
alter database tempdb on tune3 = 20
```

2 发出 use tempdb 命令，然后从各段中删除主设备：

```
sp_dropsegment "default", tempdb, master
sp_dropsegment "system", tempdb, master
sp_dropsegment "logsegment", tempdb, master
```

3 要检验这些段是否已不再包括主设备，可对 master 数据库发出以下命令：

```
select dbid, name, segmap
from sysusages, sysdevices
where sysdevices.vdevno= sysusages.vdevno
and dbid = 2
and (status&2=2 or status&3=3))
```

segmap 列对主设备上的任何分配都应报告 “0”，以表明不存在段分配：

```
 dbid   name            segmap
 ------ --------------- -----------
      2          master           0
      2           tune3           7
```

或者发出以下命令：

```
use tempdb
sp_helpdb 'tempdb'
device_fragments     size      usage             created   free kbytes
-----------------   ------   ----------  -----------------  ----------
master              4.0 MB   data only     Feb 7 2008 2:18AM      2376
tune3               20.0 MB  data and log May 16 2008 1:55PM     16212

device    segment
--------- ------------------------------
master    -- unused by any segments --
tune3                        default
tune3                     logsegment
tune3                          system
```

配置用户创建的临时数据库

应用程序对临时数据库具有各自的资源和空间要求。除非您了解应用程序要求，并维护符合这些数据库要求的应用程序与数据库或组的绑定，否则请将所有临时数据库设为同样大小。如果所有临时数据库具有相同大小，则无论为应用程序或会话分配了哪个数据库，应用程序都不会耗尽资源或空间。

高速缓存用户临时数据库

通常，可采用相同的方式在组内的临时数据库中配置高速缓存。查询处理器可以根据这些缓存特性选择查询计划，而且，如果使用具有其它配置的高速缓存执行该计划，则会降低性能。

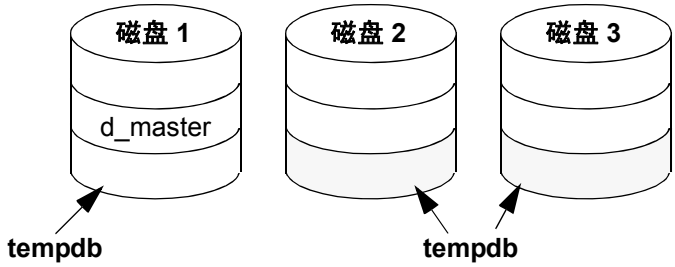
一般准则

本节介绍配置临时数据库的一般准则，该准则适用于系统和用户临时数据库。

使用多个磁盘以提高并行查询性能

如果临时数据库跨多台设备 （如图 6-2 所示），则可利用某些临时表或工作表的并行查询性能。

图 6-2：tempdb 跨越磁盘



将 *tempdb* 绑定到其自己的高速缓存

在正常使用 Adaptive Server 的情况下，在创建、填充和删除临时表时，临时数据库将大量使用数据高速缓存。

将临时数据库分配给其自己的数据高速缓存：

- 防止临时对象上的活动将其它对象刷新出缺省数据高速缓存
- 帮助在多个高速缓存间分布 I/O

用于高速缓存绑定的命令

使用 sp_cacheconfig 和 sp_poolconfig 可创建指定数据高速缓存，并为大 I/O 配置规定大小的池。仅系统管理员可以配置高速缓存和池。

注释 对大 I/O 的引用位于一台逻辑页大小为 2K 的服务器上。如果服务器的逻辑页大小为 8K，则用于 I/O 的基本单位为 8K。如果服务器的逻辑页大小为 16K，则用于 I/O 的基本单位为 16K。

有关配置指定高速缓存和池的说明，请参见 《系统管理指南，卷 2》中的第 4 章 “配置数据高速缓存”。

配置了高速缓存且服务器已重新启动后，即可将 tempdb 绑定到新的高速缓存：

```
sp_bindcache "tempdb_cache", tempdb
```

确定临时数据库的大小

为临时数据库分配足够空间可为各个并发 Adaptive Server 用户处理以下进程：

- 用于合并连接的工作表
- 为 distinct、group by 和 order by， reformatting， or 策略以及实现某些视图和子查询而创建的工作表
- 散列临时表 （创建这些表时使用 “#”作为表名的第一个字符）
- 临时表的索引
- 临时数据库中的常规用户表
- 由动态 SQL 创建的过程

如果使用临时表来拆分复合连接，某些应用程序可能会更好地执行。此策略常用于：

- 优化程序不能为连接多于四个表的查询选择良好查询计划的情况
- 连接大量表的查询
- 非常复杂的查询
- 需要将过滤数据作为中间步骤的应用程序

您还可以使用临时数据库执行以下操作：

- 非规范化一些表并放入几个临时表中
- 规范化一个非规范划化的表进行集合处理

根据使用情况确定临时数据库的大小。对于大多数应用程序，应将临时数据库的大小设置为用户数据库大小的 20-25%，以便提供足够空间供其使用。

尽量减少临时数据库中的日志记录

即使在临时数据库中启用 trunc log on checkpoint 数据库选项， Adaptive Server 仍会将对临时数据库的更改写入事务日志。可通过以下方法减少临时数据库中的日志活动：

- 使用 select into，而不使用 create table 和 insert
- 仅将所需的列选择到临时表

使用 *select into*

在创建和填充临时数据库中的临时表时，应尽可能使用 select into 命令，而不使用 create table 和 insert...select。select into/bulkcopy 数据库选项在临时数据库中缺省为启用状态，以便执行上述操作。

select into 操作只记录最小量的日志，因此运行速度更快。跟踪的只是数据页的分配，而不是每个数据行的实际更改。系统会完整记录 insert...select 查询中的每个数据插入操作，从而导致开销增加。

使用较短的行

如果在临时数据库中创建表的应用程序只使用少量表列，则可以通过以下方法最大程度地减少日志记录的数量和大小：

- 只选择应用程序所需的列，而不在将数据插入表的查询中使用 select *
- 限制所选的行恰好满足应用程序要求

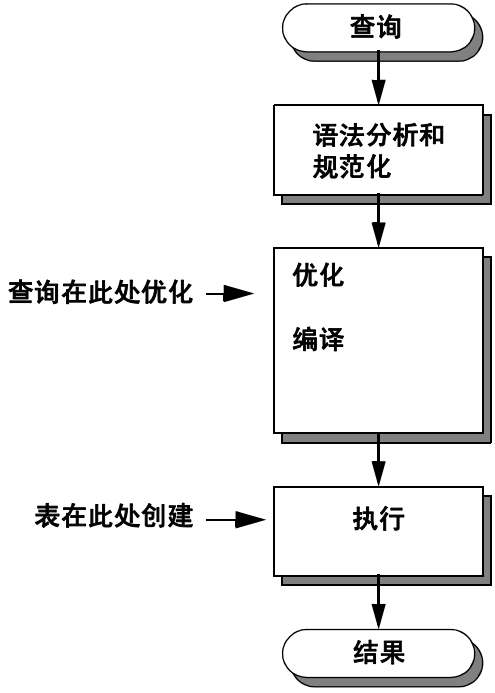
上述建议还可减小表本身的大小。

优化临时表

临时表的许多用法都简单明了，几乎不需要优化。但是，如果应用程序需要多次访问临时数据库中的表，则需要对其进行检查，以尽可能优化策略。通常，它包括使用多个过程或批处理对访问它的表的创建和索引进行拆分。

在使用表的存储过程或批处理中创建表时，查询优化程序无法确定表的大小，因为在优化查询时尚未创建该表，如图 6-3 所示。这适用于临时表和常规用户表。

图 6-3：优化和创建临时表



优化程序假定任何此类表都有 10 个数据页和 100 行。如果实际上表很大，此假定可导致优化程序选择一个次优查询计划。

这两种技术可改善临时表的优化：

- 创建临时表的索引
- 将临时表复杂的使用拆分为多个批处理或过程，以便为优化程序提供信息

创建临时表的索引

您可定义临时表的索引。多数情况下，这些索引可提高使用临时数据库的查询的性能。优化程序像使用普通用户表的索引一样使用这些索引。只是要求：

- 创建索引时表中必须包含数据。如果创建临时表并在空表上创建索引，Adaptive Server 不会创建列统计信息（如直方图和密度）。如果创建索引后插入数据行，优化程序所拥有的统计信息将是不完整的。
- 当查询使用索引优化时，该索引必须存在。不能创建一个索引然后在同一个批处理或过程中的查询中使用它们。查询处理器使用在存储过程内运行的查询中的存储过程中创建的索引。
- 如果自索引创建以来或自运行 update statistics 以来，已经添加或删除了行，优化程序可能选择次优计划。

为优化程序提供一个索引可显著提高性能，特别是在创建临时表并对其执行大量操作的复杂过程中。

使用临时表创建嵌套过程

您需要采取额外步骤来创建上述过程。如果 select_proc 不存在，则无法创建 base_proc；而没有临时表，也无法创建 select_proc。

1　在过程外创建临时表。此表可以是空的；但它必须存在，且包含与 select_proc 兼容的列：

```
select * into #huge_result from ... where 1 = 2
```

2　按以上所示创建过程 select_proc。

3　删除 #huge_result。

4　创建过程 base_proc。

将 *tempdb* 的使用拆分为多个过程

例如，此查询在使用 #huge_result 时会导致优化问题：

```
create proc base_proc
as
    select *
        into #huge_result
        from ...
    select *
        from tab,
        #huge_result where ...
```

通过使用两个过程可以获得更好的性能。当 base_proc 过程调用 select_proc 过程时，优化程序可确定表的大小：

```
create proc select_proc
as
    select *
        from tab, #huge_result where ...
create proc base_proc
as
    select *
        into #huge_result
        from ...
    exec select_proc
```

如果处理 #huge_result 时需进行多次访问、连接或需要其它进程（如使用 while 循环），则对 #huge_result 创建索引可以提高性能。可在 base_proc 中创建该索引，以便在优化 select_proc 时使用它。

优化临时数据库的日志记录

Adaptive Server 在关闭或出现故障时不恢复临时数据库，但会在您重新启动服务器时创建临时数据库。由于不需要恢复临时数据库，因此 Adaptive Server 将通过以下方法优化临时数据库的日志记录机制以提高性能：

- 单个日志记录 — 强制 Adaptive Server 在记录日志后立即将 syslogs 刷新到磁盘。 Adaptive Server 会在修改 OAM 页或分配页时 （在配置为在同一台设备上使用混合日志和数据的数据库中）创建单个日志记录。 Adaptive Server 必须刷新 syslogs，以避免在固定缓冲区期间创建的未检测到的死锁。由于 Adaptive Server 不为临时数据库固定缓冲区，因此它在写入单个日志记录时不需要刷新临时数据库的 syslogs 数据，这可以减少日志信号争用。
- 将脏页刷新到磁盘 — 对于需要恢复的数据库， Adaptive Server 会在检查点将脏页刷新到磁盘，以确保在 Adaptive Server 出现故障时，能够将所有提交的数据保存到磁盘。对于临时数据库， Adaptive Server 支持运行时回退，但不支持故障恢复，从而使其可以避免在检查点刷新脏数据页。
- 避免预写日志记录 — 预写日志记录可确保， Adaptive Server 可以通过重新发出日志中列出的事务，并撤消中止或回退事务执行的更改，来恢复已提交事务的相关数据。在不需要恢复的数据库中，Adaptive Server 不支持预写日志记录。由于 Adaptive Server 不恢复临时数据库，因此不固定临时数据库的缓冲区，这使 Adaptive Server 能够在使用临时数据库提交事务时跳过刷新临时数据库日志的过程。

用户日志高速缓存 (ULC)

Adaptive Server 为分配给会话的临时数据库提供一个单独的用户日志高速缓存 (ULC)。当用户在用户数据库和会话的临时数据库之间进行切换或者满足以下所有条件时， Adaptive Server 可以利用 ULC 避免刷新日志：

- Adaptive Server 当前正在提交事务。
- 所有日志记录都位于 ULC 中。
- 没有提交后日志记录。

可以使用配置选项 session tempdb log cache size 配置 ULC 的大小，并帮助确定它所需的刷新频率。请参见 《系统管理指南，卷 1》中的第 5 章 “设置配置参数”。

索引

英文

B

C

D

F

G

H

M

P

Q

R

S

T

W

X

Y

Z