KingbaseES V008R006

金 仓 数 据 库 操 作 手 册

北京人大金仓信息技术股份有限公司

约定

以下文本约定适用于本文档：

- 中括号（[和]）表示包含一个或多个可选项。不需要输入中括号本身。
- 花括号（{和}）表示包含两个以上（含两个）的候选，必须在其中选取一个。不需要输入花括号本身。
- | 为分割中括号或者花括号中的两个或两个以上选项。不需要输入“|”本身。
- 点（. . .）表示其之前的元素可以被重复。

日常数据库维护工作

KingbaseES需要定期执行特定的任务来达到最优的性能。这里讨论的任务是必需的，但它们本质上是重复性的并且可以很容易使用cron脚本或Windows的任务计划程序等标准工具来自动进行。建立合适的脚本并检查它们是否成功运行是数据库管理员的职责。

一个显而易见的维护任务是定期创建数据的后备拷贝。如果没有备份，就不能在灾难（磁盘失败、错误地删除一个关键表等）后进行恢复。KingbaseES中的备份和恢复机制在备份和恢复中有详细的介绍。

另一种主要类型的维护任务是周期性地“清理”数据库。该活动在1.日常清理中讨论。与之相关，更新将被查询规划器使用的统计信息的活动将在1.3.更新规划器统计信息中讨论。

另一项需要周期性考虑的任务是日志文件管理。这在3.日志文件维护中讨论。

check_kingbase可用于检测数据库的健康并报告异常情况。check_kingbase与Nagios和MRTG整合在一起，但也可以被单独运行。

1.日常清理

KingbaseES数据库要求周期性的清理维护。在多数情况下，让自动清理守护进程执行清理是足够的，见1.6.自动清理后台进程所述。可能需要调整其中描述的自动清理参数来获得最佳结果。某些数据库管理员会希望使用手动管理的VACUUM命令来对后台进程的活动进行补充或者替换，通常使用cron或任务计划程序脚本来执行。要正确地设置手动管理的清理，最重要的是理解接下来几章节中讨论的问题。依赖自动清理的管理员最好也能略读该内容以帮助他们理解和调整自动清理。

1.1.清理的基础知识

KingbaseES的VACUUM命令出于几个原因必须定期处理每一个表：

1. 恢复或重用被已更新或已删除行所占用的磁盘空间。
2. 更新被KingbaseES查询规划器使用的数据统计信息。
3. 更新可见性映射，它可以加速只用索引的扫描。
4. 保护老、旧数据不会由于事务ID回卷或多事务ID回卷而丢失。

正如后续章节中所解释的，每一个原因都将指示以不同的频率和范围执行VACUUM操作。

有两种VACUUM的变体：标准VACUUM和VACUUM FULL。VACUUM FULL可以收回更多磁盘空间

但是运行起来更慢。另外，标准形式的VACUUM可以和生产数据库操作并行运行（SELECT、INSERT、UPDATE和DELETE等命令将继续正常工作，但在清理期间无法使用ALTER TABLE等命令来更新表的定义）。VACUUM FULL要求在其工作的表上得到一个排他锁，因此无法和对此表的其他使用并行。因此，通常管理员应该努力使用标准VACUUM并且避免VACUUM FULL。

VACUUM会产生大量I/O流量，这将导致其他活动会话性能变差。可以调整一些配置参数来后台清理活动造成的性能冲击 — 参阅服务器配置-4.4.基于代价的清理延迟。

1.2.恢复磁盘空间

在KingbaseES中，一次行的UPDATE或DELETE不会立即移除该行的旧版本。这种方法对于从多版本并发控制（MVCC，见并发控制）获益是必需的：当旧版本仍可能对其他事务可见时，它不能被删除。但是最后，任何事务都不会再对一个过时的或者被删除的行版本感兴趣。它所占用的空间必须被回收来用于新行，这样可避免磁盘空间需求的无限制增长。这通过运行VACUUM完成。

VACUUM的标准形式移除表和索引中的死亡行版本并将该空间标记为可在未来重用。不过，它将不会把该空间交还给操作系统，除非在特殊的情况中表尾部的一个或多个页面变成完全空闲并且能够很容易地得到一个排他表锁。相反，VACUUM FULL通过把死亡空间之外的内容写成一个完整的新版本表文件来主动紧缩表。这将最小化表的尺寸，但是要花较长的时间。它也需要额外的磁盘空间用于表的新副本，直到操作完成。

例行清理的一般目标是多做标准的VACUUM来避免需要VACUUM FULL。自动清理守护进程尝试这样工作，并且实际上永远不会发出VACUUM FULL。在这种方法中，其思想不是让表保持它们的最小尺寸，而是保持磁盘空间使用的稳定状态：每个表占用的空间等于其最小尺寸外加清理之间被用完的空间。尽管VACUUM FULL可被用来把一个表收缩回它的最小尺寸并将该磁盘空间交还给操作系统，但是如果该表将在未来再次增长这样就没什么意义。因此，对于维护频繁被更新的表，适度运行标准VACUUM运行比少量运行VACUUM FULL要更好。

一些管理员更喜欢自己计划清理，例如在晚上负载低时做所有的工作。根据一个固定日程来做清理的难点在于，如果一个表有一次预期之外的更新活动尖峰，它可能膨胀得真正需要VACUUM FULL来回收空间。使用自动清理守护进程可以减轻这个问题，因为守护进程会根据更新活动动态规划清理操作。除非你的负载是完全可以预估的，完全禁用守护进程是不理智的。一种可能的折中方案是设置守护进程的参数，这样它将只对异常的大量更新活动做出反应，因而保证事情不会失控，而在负载正常时采用有计划的VACUUM来做批量工作。

对于那些不使用自动清理的用户，一种典型的方法是计划一个数据库范围的VACUUM，该操作每天在低使用量时段执行一次，并根据需要辅以在重度更新表上的更频繁的清理（一些有着极高更新率的安装会每几分钟清理一次它们的最繁忙的表）。如果你在一个集簇中有多个数据库，别忘记VACUUM每一个，你会用得上vacuumdb程序。

提示

> 当一个表因为大量更新或删除活动而包含大量死亡行版本时，纯粹的VACUUM可能不能达到目的。如果有这样一个表并且需要回收它所占用的过量磁盘空间，需要使用VACUUM FULL、CLUSTER或ALTER TABLE的表重写变体之一。这些命令重写该表的一整个新拷贝并且为它构建新索引。所有这些选项都要求排他锁。注意它们也临时使用大约等于该表尺寸的额外磁盘空间，因为直到新表和索引完成之前旧表和索引都不能被释放。

提示

> 如果有一个表，其所有内容需被周期性删除，考虑使用TRUNCATE而不是先用DELETE再用VACUUM。TRUNCATE将立刻移除该表的整个内容，而不需要后续的VACUUM或VACUUM FULL来回收现在未被使用的磁盘空间。其缺点是会违背严格的 MVCC 语义。

1.3.更新规划器统计信息

KingbaseES查询规划器依赖于有关表内容的统计信息来为查询产生好的计划。这些统计信息由ANALYZE命令收集，它除了直接被调用之外还可以作为VACUUM的一个可选步骤被调用。拥有适度准确的统计信息很重要，否则差的计划可能降低数据库性能。

自动清理守护进程如果被启用，当一个表的内容被改变得足够多时，它将自动发出ANALYZE命令。不过，管理员可能更喜欢依靠手动的ANALYZE操作，特别是如果知道一个表上的更新活动将不会影响“感兴趣的”列的统计信息时。守护进程严格地按照一个被插入或更新行数的函数来计划ANALYZE，它不知道那是否将导致有意义的统计信息改变。

正如用于空间恢复的清理一样，频繁更新统计信息对重度更新的表更加有用。但即使对于一个重度更新的表，如果该数据的统计分布没有很大改变，也没有必要更新统计信息。一个简单的经验法则是考虑表中列的最大和最小值改变了多少。例如，一个包含行被更新时间的timestamp列将在行被增加和更新时有一直增加的最大值；这样一列将可能需要更频繁的统计更新，而一个包含一个网站上被访问页面 URL 的列则不需要。URL 列可以经常被更改，但是其值的统计分布的变化相对很慢。

可以在指定表上运行ANALYZE并且只在表的指定列上运行，因此如果你的应用需要，可以更加频繁地更新某些统计。但实际上，通常只分析整个数据库是最好的，因为它是一种很快的操作。ANALYZE对一个表的行使用一种统计的随机采样，而不是读取每一个单一行。

提示

> 尽管对每列的ANALYZE频度调整可能不是非常富有成效，你可能会发现值得为每列调整被ANALYZE收集统计信息的详细程度。经常在WHERE中被用到的列以及数据分布非常不规则的列可能需要比其他列更细粒度的数据直方图。见ALTER TABLE SET STATISTICS，或者使用default_statistics_target配置参数改变数据库范围的默认值。
>
> 还有，默认情况下关于函数的选择度的可用信息是有限的。但是，如果你创建一个使用函数调用的表达式索引，关于该函数的有用的统计信息将被收集，这些信息能够大大提高使用该表达式索引的查询计划的质量。

提示

> 自动清理守护进程不会为外部表发出ANALYZE命令，因为无法确定一个合适的频度。如果你的查询需要外部表的统计信息来正确地进行规划，比较好的方式是按照一个合适的时间表在那些表上手工运行ANALYZE命令。

1.4.更新可见性映射

清理机制为每一个表维护着一个可见性映射，它被用来跟踪哪些页面只包含对所有活动事务（以及所有未来的事务，直到该页面被再次修改）可见的元组。这样做有两个目的。第一，清理本身可以在下一次运行时跳过这样的页面，因为其中没有什么需要被清除。

第二，这允许KingbaseES回答一些只用索引的查询，而不需要引用底层表。因为KingbaseES的索引不包含元组的可见性信息，一次普通的索引扫描会为每一个匹配的索引项获取堆元组，用来检查它是否能被当前事务所见。另一方面，一次*只用索引的扫描*会首先检查可见性映射。如果它了解到在该页面上的所有元组都是可见的，堆获取就可以被跳过。这对大数据集很有用，因为可见性映射可以防止磁盘访问。可见性映射比堆小很多，因此即使堆非常大，可见性映射也可以很容易地被缓存起来。

1.5.防止事务 ID 回卷失败

KingbaseES的 MVCC 事务语义依赖于能够比较事务 ID（XID）数字：如果一个行版本的插入 XID 大于当前事务的 XID，它就是“属于未来的”并且不应该对当前事务可见。但是因为事务 ID 的尺寸有限（32位），一个长时间（超过 40 亿个事务）运行的集簇会遭受到*事务 ID 回卷*问题：XID 计数器回卷到 0，并且本来属于过去的事务突然间就变成了属于未来 — 这意味着它们的输出变成不可见。简而言之，灾难性的数据丢失（实际上数据仍然在那里，但是如果你不能得到它也无济于事）。为了避免发生这种情况，有必要至少每 20 亿个事务就清理每个数据库中的每个表。

周期性的清理能够解决该问题的原因是，VACUUM会把行标记为 *冻结*，这表示它们是被一个在足够远的过去提交的事务所插入， 这样从 MVCC 的角度来看，效果就是该插入事务对所有当前和

未来事务来说当然都 是可见的。KingbaseES保留了一个特殊的 XID（FrozenTransactionId），这个 XID 并不遵循普通 XID 的比较规则 并且总是被认为比任何普通 XID 要老。普通 XID 使用模-2^{32}算 法来比较。这意味着对于每一个普通 XID都有 20 亿个 XID “更老”并且 有 20 亿个“更新”，另一种解释的方法是普通 XID 空间是没有端点的环。 因此，一旦一个行版本创建时被分配了一个特定的普通 XID，该行版本将成为接下 来 20 亿个事务的“过去”（与我们谈论的具体哪个普通 XID 无关）。如 果在 20 亿个事务之后该行版本仍然存在，它将突然变得好像在未来。要阻止这一切 发生，被冻结行版本会被看成其插入 XID 为FrozenTransactionId， 这样它们对所有普通事务来说都是“在过去”，而不管回卷问题。并且这样 的行版本将一直有效直到被删除，不管它有多旧。

注意

> 在KingbaseES V8R3之前的版本中，实际上会通过将一行的插入 XID 替换为FrozenTransactionId来实现冻结，这种FrozenTransactionId在行的 xmin系统列中是可见的。较新的版本只是设置一个标志位， 保留行的原始xmin用于可能发生的鉴别用途。不过， 在9.4之前版本的数据库sys_upgrade中可能仍会找到 xmin等于FrozenTransactionId (2)的行。
>
> 此外，系统目录可能会包含xmin等于BootstrapTransactionId (1) 的行，这表示它们是在initdb的第一个阶段被插入的。 和FrozenTransactionId相似，这个特殊的 XID 被认为比所有正常 XID 的年龄都要老。

vacuum_freeze_min_age控制在其行版本被冻结前一个 XID 值应该有多老。如果被冻结的行将很快会被再次修改，增加这个设置可以避免不必要 的工作。但是减少这个设置会增加在表必须再次被清理之前能够流逝的事务数。

VACUUM通常会跳过不含有任何死亡行版本的页面，但是不会跳过那些含有带旧 XID 值的行版本的页面。要保证所有旧的行版本都已经被冻结，需要对整个表做一次扫描。vacuum_freeze_table_age控制VACUUM什么时候这样做：如果该表经过vacuum_freeze_table_age减去vacuum_freeze_min_age个事务还没有被完全扫描过，则会强制一次全表清扫。将这个参数设置为 0 将强制VACUUM总是扫描所有页面而实际上忽略可见性映射。

一个表能保持不被清理的最长时间是 20 亿个事务减去VACUUM上次扫描全表时的vacuum_freeze_min_age值。如果它超过该时间没有被清理，可能会导致数据丢失。要保证这不会发生，将在任何包含比autovacuum_freeze_max_age配置参数所指定的年龄更老的 XID 的未冻结行的表上调用自动清理（即使自动清理被禁用也会发生）。

这意味着如果一个表没有被清理，大约每autovacuum_freeze_max_age减去vacuum_freeze_min_age事务就会在该表上调用一次自动清理。对那些为了空间回收目的而被正常清理的表，这是无关紧要的。然而，对静态表（包括接收插入但没有更新或删除的表）

就没有为空间回收而清理的需要，因此尝试在非常大的静态表上强制自动清理的间隔最大化会非常有用。显然我们可以通过增加autovacuum_freeze_max_age或减少vacuum_freeze_min_age来实现此目的。

vacuum_freeze_table_age的实际最大值是 0.95 * autovacuum_freeze_max_age，高于它的设置将被上限到最大值。一个高于autovacuum_freeze_max_age的值没有意义，因为不管怎样在那个点上都会触发一次防回卷自动清理，并且 0.95 的乘数为在防回卷自动清理发生之前运行一次手动VACUUM留出了一些空间。作为一种经验法则，vacuum_freeze_table_age应当被设置成一个低于autovacuum_freeze_max_age的值，留出一个足够的空间让一次被正常调度的VACUUM或一次被正常删除和更新活动触发的自动清理可以在这个窗口中被运行。将它设置得太接近可能导致防回卷自动清理，即使该表最近因为回收空间的目的被清理过，而较低的值将导致更频繁的全表扫描。

增加autovacuum_freeze_max_age（以及和它一起的vacuum_freeze_table_age）的唯一不足是数据库集簇的sys_xact和sys_commit_ts子目录将占据更多空间，因为它必须存储所有向后autovacuum_freeze_max_age范围内的所有事务的提交状态和（如果启用了track_commit_timestamp）时间戳。提交状态为每个事务使用两个二进制位，因此如果autovacuum_freeze_max_age被设置为它的最大允许值 20 亿，sys_xact将会增长到大约 0.5 吉字节，sys_commit_ts大约20GB。如果这对于你的总数据库尺寸是微小的，我们推荐设置autovacuum_freeze_max_age为它的最大允许值。否则，基于你想要允许sys_xact和sys_commit_ts使用的存储空间大小来设置它（默认情况下 2 亿个事务大约等于sys_xact的 50 MB存储空间，sys_commit_ts的2GB的存储空间）。

减小vacuum_freeze_min_age的一个不足之处是它可能导致VACUUM做无用的工作：如果该行在被替换成FrozenXID之后很快就被修改（导致该行获得一个新的 XID），那么冻结一个行版本就是浪费时间。因此该设置应该足够大，这样直到行不再可能被修改之前，它们都不会被冻结。

为了跟踪一个数据库中最老的未冻结 XID 的年龄，VACUUM在系统表sys_class和sys_database中存储 XID 的统计信息。特别地，一个表的sys_class行的relfrozenxid列包含被该表的上一次全表VACUUM所用的冻结截止 XID。该表中所有被有比这个截断 XID 老的普通 XID 的事务插入的行 都确保被冻结。相似地，一个数据库的sys_database行的datfrozenxid列是出现在该数据库中的未冻结 XID 的下界 — 它只是数据库中每一个表的relfrozenxid值的最小值。一种检查这些信息的方便方法是执行这样的查询：

```
SELECT c.oid::regclass as table_name,
       greatest(age(c.relfrozenxid),age(t.relfrozenxid)) as age
FROM sys_class c
LEFT JOIN sys_class t ON c.reltoastrelid = t.oid
WHERE c.relkind IN ('r', 'm');

SELECT datname, age(datfrozenxid) FROM sys_database;
```

age列度量从该截断 XID 到当前事务 XID 的事务数。

VACUUM通常只扫描从上次清理后备修改过的页面，但是只有当全表被扫描时relfrozenxid才能被推进。当relfrozenxid比vacuum_freeze_table_age个事务还老时、当VACUUM的FREEZE选项被使用时或当所有页面正好要求清理来移除死亡行版本时，全表将被扫描。当VACUUM扫描全表时，在它被完成后，age(relfrozenxid)应该比被使用的vacuum_freeze_min_age设置略大（比在VACUUM开始后开始的事务数多）。如果在autovacuum_freeze_max_age被达到之前没有全表扫描VACUUM在该表上被发出，将很快为该表强制一次自动清理。

如果出于某种原因自动清理无法从一个表中清除旧的 XID，当数据库的最旧 XID 和回卷点之间达到 1 千万个事务时，系统将开始发出这样的警告消息：

```
WARNING:  database "mydb" must be vacuumed within 177009986 transactions
HINT:  To avoid a database shutdown, execute a database-wide VACUUM in "mydb".
```

（如该示意所建议的，一次手动的VACUUM应该会修复该问题；但是注意该次VACUUM必须由一个超级用户来执行，否则它将无法处理系统目录并且因而不能推进数据库的datfrozenxid）。如果这些警告被忽略，一旦距离回卷点只剩下 1 百万个事务时，该系统将会关闭并且拒绝开始任何新的事务：

```
ERROR:  database is not accepting commands to avoid wraparound data loss in
database "mydb"
HINT:  Stop the kingbase and vacuum that database in single-user mode.
```

这一百万个事务的富余是为了让管理员能通过手动执行所要求的VACUUM命令进行恢复而不丢失数据。但是，由于一旦系统进入到安全关闭模式，它将不会执行命令。做这个操作的唯一方法是停止服务器并且以单一用户启动服务器来执行VACUUM。单一用户模式中不会强制该关闭模式。关于使用单一用户模式的细节请见kingbase参考页。

1.5.1.多事务和回卷

*Multixact ID*被用来支持被多个事务锁定的行。由于在一个元组头部 只有有限的空间可以用来存储锁信息，所以只要有多于一个事务并发地锁住一个行， 锁信息将使用一个“多个事务 ID”（或简称多事务 ID）来编码。任何特定 多事务 ID 中包括的事务 ID 的信息被独立地存储

在sys_multixact子目 录中，并且只有多事务 ID 出现在元组头部的xmax域中。和事务 ID 类似，多事务 ID 也是用一个 32 位计数器实现，并且也采用了相似的存储，这些都要 求仔细的年龄管理、存储清除和回卷处理。在每个多事务中都有一个独立的存储区域 保存成员列表，它也使用一个 32 位计数器并且也应被管理。

在一次VACUUM表扫描（部分或者全部）期间，任何比 vacuum_multixact_freeze_min_age 要老的多事务 ID 会被替换为一个不同的值，该值可以是零值、 一个单一事务 ID 或者一个更新的多事务 ID。 对于每一个表，sys_class.relminmxid 存储了在该表任意元组中仍然存在的最老可能多事务 ID。如果这个值比 vacuum_multixact_freeze_table_age老， 将强制一次全表扫描。可以在 sys_class.relminmxid 上使用mxid_age()来找到它的年龄。

全表VACUUM扫描（不管是什么导致它们）将为表推进该值。 最后，当所有数据库中的所有表被扫描并且它们的最老多事务值被推进， 较老的多事务的磁盘存储可以被移除。

作为一种安全设备，对任何多事务年龄超过 autovacuum_multixact_freeze_max_age的表， 都将发生一次全表清理扫描。如果已用的成员存储空间超过总量的 50%，全表清理扫描 也将逐步在所有表上进行，这会从那些具有最老多事务年龄的表开始。即使自动清理被 在名义上被禁用，这两中类型的全表扫描都将会发生。

1.6.自动清理后台进程

KingbaseES有一个可选的但是被高度推荐的特性*autovacuum*，它的目的是自动执行VACUUM和ANALYZE 命令。当它被启用时，自动清理会检查被大量插入、更新或删除元组的表。这些检查会利用统计信息收集功能，因此除非track_counts被设置为true，自动清理不能被使用。在默认配置下，自动清理是被启用的并且相关配置参数已被正确配置。

“自动清理后台进程”实际上由多个进程组成。有一个称为 *自动清理启动器*的常驻后台进程， 它负责为所有数据库启动*自动清理工作者*进程。 启动器将把工作散布在一段时间上，它每隔autovacuum_naptime秒尝试在每个数据库中启动一个工作者 （因此，如果安装中有*N*个数据库，则每 autovacuum_naptime/*N*秒将启动一个新的工作者）。 在同一时间只允许最多autovacuum_max_workers 个工作者进程运行。如果有超过autovacuum_max_workers 个数据库需要被处理，下一个数据库将在第一个工作者结束后马上被处理。 每一个工作者进程将检查其数据库中的每一个表并且在需要时执行 VACUUM和/或ANALYZE。 可以设置log_autovacuum_min_duration 来监控自动清理工作者的活动。

如果在一小段时间内多个大型表都变得可以被清理，所有的自动清理工作者可能都会被占用来在一段长的时间内清理这些表。这将会造成其他的表和数据库无法被清理，直到一个工作者变得可用。对于一个数据库中的工作者数量并没有限制，但是工作者确实会试图避免重复已经被其他工作者完成的工作。注意运行着的工作者的数量不会被计入max_connections或superuser_reserved_connections限制。

relfrozenxid值比autovacuum_freeze_max_age事务年龄更大的表总是会被清理（这页表示这些表的冻结最大年龄被通过表的存储参数修改过，参见后文）。否则，如果从上次VACUUM以来失效的元组数超过“清理阈值”，表也会被清理。清理阈值定义为：

```
清理阈值 = 清理基本阈值 + 清理缩放系数 * 元组数
```

其中清理基本阈值为autovacuum_vacuum_threshold， 清理缩放系数为autovacuum_vacuum_scale_factor， 元组数为sys_class.reltuples。 失效元组的数量从统计信息收集器获得，它是一个由每个UPDATE和DELETE命令更新的半准确的计数（它只是半准确，是因为在高负载的情况下某些信息可能会丢失）。如果表的relfrozenxid值比vacuum_freeze_table_age事务年龄更大，整个表将被扫描以冻结旧元组并增长relfrozenxid，否则只有从上次清理以来被修改的页面会被扫描。

对于分析，也使用了一个相似的阈值：

```
分析阈值 = 分析基本阈值 + 分析缩放系数 * 元组数
```

该阈值将与自从上次ANALYZE以来被插入、更新或删除的元组数进行比较。

临时表不能被自动清理访问。因此，临时表的清理和分析操作必须通过会话期间的SQL命令来执行。

默认的阈值和缩放系数都取自于kingbase.conf， 但是可以为每一个表重写它们(和许多其他自动清理控制参数)， 详情参见存储参数。 如果一个设置已经通过一个表的存储参数修改，那么在处理该表时使用该值， 否则使用全局设置。全局设置请参阅服务器配置-10.自动清理。

当多个工作者运行时，在所有运行着的工作者之间自动清理代价延迟参数 (参阅服务器配置-4.4.基于代价的清理延迟)是 “平衡的”，这样不管实际运行的工作者数量是多少， 对于系统的总体 I/O 影响总是相同的。不过，任何正在处理已经设置了每表 autovacuum_vacuum_cost_delay或 autovacuum_vacuum_cost_limit 存储参数的表的工作者不会被考虑在均衡算法中。

自动清理工作者通常不会屏蔽其他指令。如果一个进程试图获取一个与自动清理持有的共享更新独占锁冲突的锁，那么获取锁将中断自动清理。有关冲突锁模式，请参阅表3-1。但是，如果自动清理正在运行，为防止事务ID封装，自动清理不会自动中断。

警告

> 定期运行获取与SHARE UPDATE EXCLUSIVE 锁(例如，ANALYZE)相冲突的锁的命令，可以有效地防止自动清理完成。

2.日常重建索引

在某些情况下值得周期性地使用REINDEX命令或一系列独立重构步骤来重建索引。

已经完全变成空的B树索引页面被收回重用。但是，还是有一种低效的空间利用的可能性：如果一个页面上除少量索引键之外的全部键被删除，该页面仍然被分配。因此，在这种每个范围中大部分但不是全部键最终被删除的使用模式中，可以看到空间的使用是很差的。对于这样的使用模式，推荐使用定期重索引。

对于非B树索引可能的膨胀还没有很好地定量分析。在使用非B树索引时定期监控索引的物理尺寸是个好主意。

还有，对于B树索引，一个新建立的索引比更新了多次的索引访问起来要略快， 因为在新建立的索引上，逻辑上相邻的页面通常物理上也相邻（这样的考虑目前并不适用于非B树索引）。仅仅为了提高访问速度也值得定期重索引。

REINDEX在所有情况下都可以安全和容易地使用。 此命令默认需要一个ACCESS EXCLUSIVE锁，因此通常更可取的做法是使用它的并发选项，这只需要一个SHARE UPDATE EXCLUSIVE锁。

3.日志文件维护

把数据库服务器的日志输出保存在一个地方是个好主意， 而不是仅仅通过/dev/null丢弃它们。 在进行问题诊断的时候，日志输出是非常宝贵的。不过，日志输出可能很庞大（特别是在比较高的调试级别上）， 因此你不会希望无休止地保存它们。你需要*轮转*日志文件， 这样在一段合理的时间后会开始新的日志文件并且移除旧的。

如果你简单地把kingbase的stderr定向到一个文件中，你会得到日志输出， 但是截断该日志文件的唯一方法是停止并重起服务器。这样做对于开发环境中使用的KingbaseES可能是可接受的，但是你肯定不想在生产环境上这么干。

一个更好的办法是把服务器的stderr输出发送到某种日志轮转程序里。 我们有一个内建的日志轮转程序，你可以通过在 kingbase.conf里设置配置参数logging_collector为true的办法启用它。该程序的控制参数在 服务器配置-8.1.在哪里做日志里描述。你也可以使用这种方法把日志数据捕捉成机器可读的CSV（逗号分隔值）格式。

另外，如果在你已经使用的其他服务器软件中有一个外部日志轮转程序，你可能更喜欢使用它。比如，包含在Apache发布里的 rotatelogs工具就可以用于KingbaseES。 这么做的一种方法是把服务器的stderr用管道重定向到要用的程序。 如果你用sys_ctl启动服务器，那么stderr已经 重定向到stdout， 因此你只需要一个管道命令，比如：

```
sys_ctl start | rotatelogs /var/log/kingbase_log 86400
```

您可以通过设置logrotate来收集由KingbaseES内置日志收集器产生的日志文件，从而组合这些方法。 在这种情况下，日志收集器定义日志文件的名称和位置，而logrotate定期归档这些文件。

当启动日志循环时，logrotate必须确保应用程序将进一步的输出发送到新文件。 这通常通过postrotate脚本完成，该脚本向应用程序发送SIGHUP信号，然后应用程序重新打开日志文件。 在KingbaseES中，您可以使用logrotate选项来运行sys_ctl。 当服务器接收到此命令时，服务器要么切换到新的日志文件，要么根据日志配置重新打开现有文件(见服务器配置-8.1.在哪里做日志)。

注意

> 当使用静态日志文件名时，如果达到最大打开文件限制或出现文件表溢出，服务器可能无法重新打开日志文件。在这种情况下，日志消息被发送到旧的日志文件，直到日志循环成功为止。如果logrotate被配置为压缩日志文件并删除它，服务器可能会丢失在此时间段内记录的消息。为了避免这个问题，可以配置日志收集器来动态分配日志文件名，并使用prerotate脚本来忽略打开的日志文件。

另外一种生产级的管理日志输出的方法就是把它们发送给syslog，让syslog处理文件轮转。利用此工具，需要设置kingbase.conf里的log_destination配置参数设置为syslog（记录syslog日志）。然后在你想强制syslog守护进程开始写入一个新日志文件的时候，就可以发送一个 SIGHUP信号给它。 如果你想自动进行日志轮转，可以配置logrotate程序处理 来自syslog的日志文件。

但是，在很多系统上，syslog不是非常可靠，特别是在面对大量日志消息的情况下； 它可能在你最需要那些消息的时候截断或者丢弃它们。另外，Linux，syslog会把每个消息刷写到磁盘上， 这将导致性能变差（建议在syslog配置文件里面的文件名开头使用“-”来禁用此行为）。

请注意上面描述的所有解决方案关注的是在可配置的间隔上开始一个新的日志文件， 但它们并没有处理对旧的、不再需要的日志文件的删除。你可能还需要设置一个批处理任务来定期地删除旧日志文件。 另一种可能的方法是配置日志轮转程序，让它循环地覆盖旧的日志文件。

版权申明

联系我们

欢迎您针对此手册提出宝贵意见和建议，您的意见和建议将成为完善此手册的重要部分。

如果您发现了手册中的错误，或有更好的意见和建议，可通过以下方式发送给我们。

电子邮箱：support@kingbase.com.cn

传真：86-10-59111032

服务热线：4006011188

通信地址：北京市朝阳区容达路7号E座2层