

Oracle TimesTen™ 内存数据库

6.0 版推荐编程实践

性能和稳定性最佳的 TimesTen 应用程序的设计方法

ORACLE®

内存数据库

版权© 1996、2005 归 Oracle 公司所有。未经允许不得以任何方式和手段进行复制和使用。

附带的和光盘上的软件和文档（不管是硬拷贝形式还是电子形式）的使用和披露均应依据本许可协议。

被许可方可印刷存储在光盘上的文档，但仅限内部使用。除上面提到的以外，未经 TimesTen 公司事先书面许可，不得以任何电子或机械形式或手段，包括影印、录制或通过任何信息存储和检索系统，复制或传播本文档的任何部分（不管是硬拷贝形式还是电子形式）。

Oracle、JD Edwards、PeopleSoft、Retek、TimesTen、TimesTen 标志、MicroLogging 和 Direct Data Access 是 Oracle 公司和/或其子公司的商标或注册商标。其他名称可能是其各自所有者的商标。

程序（包括软件和文档）包含专有信息；是依据一份包含了使用和披露限制条款的许可协议而提供的，并受版权、专利其他知识产权和工业产权法的保护。除非法律有明确规定，或是为实现与其他独立开发的软件间的互操作性，否则不得对软件程序进行反向工程、反汇编或反编译。

本文所含信息如有变动，恕不另行通知。如果您在本文中发现任何错误，请书面通知我们。我们不保证本文没有错误。除非在您的许可协议中对此有明确许可，否则不得为任何目的，以任何电子或机械形式或手段复制或传播这些程序的任何部分。

2005 年 8 月 19 日
在美国印刷。

目录

1 概述

TimesTen 文档参考索引	1
术语	1
“TimesTen 应用程序”或“直连应用程序”	1
“TimesTen 客户机/服务器应用程序”	2
install_dir	2
C++用户：考虑使用 TTClasses	2

2 实现最佳性能

TimesTen 文档中的性能信息	4
有关最常见问题的（影响性能的因素）建议	4
运行对性能要求高的直连应用程序	4
预先准备好所有 SQL 语句	5
控制磁盘写频率	6
为查询创建合适的索引	6
使用“showplan”验证是否使用了合适的索引来进行查询	8
关闭自动提交(autocommit)并定期提交	9
C/C++ (ODBC)与 Java (JDBC)性能比较	9
通过在加载数据之后创建索引来加速（大批量）数据加载	10
使用 TTClasses，避免使用 OLEDB、ADO 和第三方中间件	10
多 CPU 性能调优	10
使用连接池	11
最大限度地提高数据库的并发性	11
及时关闭只读游标	11
避免大批量的删除语句	11
考虑使用“DELETE FIRST numRows”	12
缩短不必要的长期运行的事务	12

3 最大限度地提高稳定性

最大限度地提高稳定性	13
TimesTen 文档	13
TimesTen 架构和数据库恢复简明指南	13
避免应用程序的意外失败	14
必须断开 TimesTen 应用程序与数据库的连接	14
避免对 TimesTen 应用程序使用“kill -9”命令	14
备份	14
检查点	15
其他好的实践	15
检查所有 ODBC 函数的返回码，然后处理它们	15
处理数据库失效错误	16
处理死锁和锁超时错误	16
从一个已满磁盘恢复	17

4 复制和 XLA

复制.....	18
将 DSN 名用作文件名前缀	18
在进行 - duplicate 操作之前执行两个检查点操作	19
复制配置应（手动）指定端口号	19
监控复制	20
SEQUENCE 与复制和故障恢复的相互影响.....	21
XLA.....	21
使用持久性 XLA	21
始终监控 XLA.....	22
最大限度地提高 XLA 性能	22

索引

概述

本文详细介绍了如何利用 TimesTen 开发具有最佳性能和强健性的应用程序。

TimesTen 文档参考索引

下表列出了本文引用的章节，以及它们在 TimesTen 6.0 文档中的位置。

- TimesTen 内置程序: tt_ref.pdf
- 性能调优: C_dev.pdf; java_dev.pdf
- TimesTen 查询优化器: operations.pdf
- 事务管理与恢复: operations.pdf
- 警告与错误: tt_ref.pdf
- TimesTen 实用程序: tt_ref.pdf
- TTClasses: ttclasses.pdf

术语

下面是本文所使用的一些术语及其含意。

“TimesTen 应用程序”或“直连应用程序”

本术语指一个直接连接到 TimesTen 数据库的应用程序。C 和 C++应用程序显式地连接到下列共享库之一：

- libtten.so (Solaris、Linux)
- libtten.sl (HP-UX)
- tten60.lib (Windows)
- 等等

以“直连”模式使用 TimesTen 的 Java 应用程序以“jdbc:TimesTen:direct:...”的形式连接数据源。

这些应用程序与它们所访问的 TimesTen 数据库驻留在同一台机器上。这样一来，再通过使用“直连”TimesTen 驱动程序，TimesTen 应用程序将可获得

最佳性能。

“TimesTen 客户机/服务器应用程序”

该术语指一个使用 TimesTen 客户机/服务器接口来与数据库连接的应用程序——即显式地连接到共享库中

- libttclient.so (Solaris、Linux)
- libttclient.sl (HP-UX)
- ttcl60.lib (Windows)
- 等等

以客户机/服务器模式使用 TimesTen 的 Java 应用程序以“jdbc:TimesTen:client:...”的形式连接数据源

客户机/服务器应用程序可与 TimesTen 数据库驻留在同一台机器上，也可驻留在其他机器上。客户机/服务器应用程序的运行速度要比直连应用程序慢很多。

无论是直连应用程序还是客户机/服务器应用程序，提供给它们的 API 和功能都是一样的。

install_dir

这是 TimesTen 的安装路径。参考 Oracle 《TimesTen 内存数据库安装指南》了解更多信息。默认路径为：

- /opt/TimesTen/InstanceName (UNIX; 根用户安装的实例)
- \$HOME/TimesTen/InstanceName (UNIX; 非根用户安装的实例)
- C:\TimesTen\tt60 (Windows)

注意：基于安全的考虑，我们推荐，如果可能的话，以非根用户的身份安装 TimesTen。

C++用户：考虑使用 TTClasses

TimesTen C++接口类，即 TTClasses，编写它的目的是提供易用、高性能的 TimesTen 接口。C++类库为大多数常见的 ODBC 功能提供了“封装”，包括 SQL 查询执行、事件通知 (XLA) 和系统目录信息。此外，TTClasses 的设计采用了许多本文推荐的实践。TTClasses 类包含在 TimesTen 内存数据库中。

与 TimesTen 一起提供的还有一些有用的示例应用程序，用于演示如何使用 TTClasses 类。在示例程序中包含如下一些程序：示范如何监视和管理 TimesTen

数据库；如何使用 TimesTen 的 XLA 实施一个事件通知程序；如何估计数据库的大小；以及如何管理 XLA 书签。

实现最佳性能

本章重点讨论如何编写具有最佳性能的 TimesTen 应用程序。

TimesTen 文档中的性能信息

现有的 TimesTen 文档是学习关于如何调优应用程序性能的基础知识的首选资源。在阅读本章其余部分之前，请务必认真阅读 TimesTen 文档“性能调优”这一章。

有关最常见问题的（影响性能的因素）建议

本章的其余部分将讲述那些客户经常遇到的问题。请注意，这些问题在 TimesTen 文档中也有所描述。

运行对性能要求高的直连应用程序

通过使用 TimesTen 的“直连” ODBC 和 JDBC 驱动程序，并在数据库所在的机器上运行应用程序，就很容易实现 TimesTen 的最佳性能。其他机器上的应用程序可通过客户机/服务器模式访问 TimesTen 数据库，但网络流量负载将极大地降低性能。

通过利用 TimesTen 的 Replication 和 Cache Connect to Oracle 技术，通常可以在本地访问数据，而不会产生客户机/服务器模式的流量负载。

注意：TimesTen 采用客户机/服务器架构模式运行虽然不如采用直接内存模式运行得快，但它还是很快的。TimesTen 客户机/服务器应用程序比连接到传统的客户机/服务器数据库的应用程序性能高；尤其是，与传统的数据库相比，TimesTen 具有更高的吞吐能力。

在下面这些情况下，应考虑以客户机/服务器模式连接 TimesTen 数据库。

- 1) 应用程序运行在一台主机上，TimesTen 数据库驻留在另一台主机上。
- 2) 一个 32 位的客户端应用程序必须连接到 64 位的 TimesTen 数据库，并且 32 位的客户端应用程序无法重新编译为 64 位。如果应用程序与 TimesTen 数据库在同一台机器上，或能够被重新部署到该机器上，但是不能被重新编译为 64 位模式，TimesTen 建议采用 TimesTen 共享内存进程间通信（SHMIPC）。

对于客户机/服务器模式的数据库连接而言，与 TCP/IP 连接相比，SHMIPC 大幅提高了性能。

需要有大量 TimesTen 客户机/服务器连接的系统设计应考虑到每一数据库连接均都会消耗服务器主机的操作系统资源。在服务器主机的规模设置和操作系统调优过程中应考虑这一因素。

预先准备好所有 SQL 语句

为实现最佳性能，应预先准备好那些执行不止一次的 SQL 语句。这一准则适用于所有关系数据库；但是对于 TimesTen 及其极高的事务处理速度来说，编译一条语句所耗费的时间实际上要比执行该语句所花的时间多好几倍；因此，未预先准备好语句的应用程序将使 TimesTen 的性能大打折扣。除了要预先准备好语句，那些语句的输入参数和输出列也应预先绑定。

欲了解有关语句准备的更多信息，请参考 TimesTen 文档“性能调优”一章标题为“预先准备语句”部分。

ODBC 用户使用 SQLPrepare 函数预先准备语句。

TTCclasses 用户使用 TTCmd::Prepare()方法预先准备语句。

JDBC 用户使用 PreparedStatement 类预先准备语句。

TimesTen 查询优化器十分擅长选择最优的查询计划；然而，为了选择最优计划，它需要获得复杂查询所涉及到的表的更多信息。

就查询优化器选择最优查询计划来说，数据库统计信息对其是很有帮助的。通过了解表的行数和列值数据分布，优化器更有可能选择出一个高效的表查询计划。

因此，通常一个好的做法是，在准备访问那些表的查询语句之前，先更新数据库中所有表的统计信息。

注意，如果你在表包含行之前更新表的统计信息，那么将会以表没有包含行（或包含很少的行）为假定来对查询进行优化。如果你后来向表中添加了数百万行，然后执行查询，那么在表只包含极少数行的情况下工作良好的查询计划现在可能会变得很慢。

因此，你应该在向表加载数据后，在准备查询语句之前，更新表的统计信息。

欲了解关于更新统计信息的更多信息，请参考 TimesTen 文档“TimesTen 内

置程序”一章对 `ttOptUpdateStats` 和 `ttOptEstimateStats` 的描述。有关其他 `ttOpt*` 内置程序的描述也包含在这一章之中，你可以借此了解更多有用信息。

控制磁盘写频率

多数应用程序都要求某种级别的数据库持久性；也就是说，如果系统崩溃（因某种类型的软硬件故障），应用程序要求最近对数据库所做的更新不会丢失。利用 `DurableCommits` 连接属性可以有多种方法来配置 TimesTen：

- 第一种方法是以连接属性 `DurableCommits=1` 来连接 TimesTen 数据库。在这种连接中，每一次事务提交都会将事务日志写入磁盘；因而，应用程序将拥有充分的持久性。然而，设置 `DurableCommits=1`，应用程序的写性能将大大降低。注意：如果数据库中存在大量用户并发写库的情况，使用 `DurableCommits=1` 仍能够获得很好的整体吞吐能力。
- 第二种方法是，内存事务缓冲区一旦变满（默认方式，即 `DurableCommits=0`），TimesTen 数据库就将缓冲区数据写入磁盘。该方法可实现最佳性能，因为只在确实有必要时，才执行磁盘写操作，事务处理永远不会因写磁盘而受阻；然而，采用这种方法的应用程序将无法确切知道其事务处理的持久性水平（毫无疑问它不会丢失比内存日志缓冲区能容纳的事务更多的数据）。

对于使用 `DurableCommits=0` 的应用程序，TimesTen 内置程序 `ttDurableCommit` 使应用程序开发人员能够精确确定 TimesTen 内存日志缓冲何时被写入磁盘，从而可将持久性和性能控制到合适水平。应用程序能够确定哪些事务会导致同步的磁盘写操作，哪些不会。通过这种方式，应用程序能够精细地平衡性能和数据持久性。对于某些应用程序，按设定的时间间隔调用 `ttDurableCommit` 是很合理的，这样能保证一次仅会损失极少的，例如约 100 毫秒的未写入磁盘的日志缓冲区数据；对于其他应用程序，某些提交操作必须使数据库处于持久状态（例如，账户间的资金转移）。

不管 `DurableCommit` 如何设置，TimeTen 复制和/或从 TimesTen 到 Oracle 的更新传播可被用来进一步提高事务持久性。

为查询创建合适的索引

想要知道创建多少索引才能实现最佳数据库性能颇有难度。如果你创建的索引太少，那么某些经常性的数据库操作会比平常慢；如果你创建的索引太多，那么插入/更新/删除操作则会耗用太多的时间，因为需要额外的时间来更新索引。

在设计 TimesTen 数据库表和索引模式时，有以下几个需要考虑的问题。

- 有两种类型的索引：哈希索引和 T 树索引。对于精确匹配查询，调优的哈希

索引要快于 T 树索引，但是哈希索引不能用于范围查询（T 树索引既可用于精确匹配查询，也可用于范围查询，以及分类查询，如涉及 ORDER BY、GROUP BY 或 DISTINCT 子句的 SQL 查询）。

- 使用哈希索引为每个表的主键建立索引。
- 需要对哈希索引进行调优。应始终使用“CREATE TABLE”语句的“pages=”选项来指定表的期望大小（用 256 除以表的行数，即可得到你需要为哈希索引指定的页数）。为索引指定的页数太多将浪费不必要的空间；而指定的页数太少则会降低哈希索引的性能，因为此时哈希桶（buckets）会溢出。如果需要，可在其创建后使用 ALTER TABLE 语句来更改主键索引的大小。
- 通过引入一个 T 树索引（任何 T 树索引），可改进一个未包含 T 树索引的表的全表扫描性能，即使对该表的扫描没有引用该索引的列。这虽然不太直观，却很容易证实。因此，你应该为应用程序的全表扫描所引用的每一个表建立至少一个 T 树索引。
- 一个哈希三列索引只能被用于三列全部精确匹配的查询。T 树索引列的任何前导前缀可被用于一个精确匹配查询。
- 在两者都可以使用的情况下，对于等式谓词，哈希索引的性能要优于 T 树索引。然而，哈希索引需要的空间比 T 树索引大。

最后一条有些复杂，因此我们举例说明。考虑这样一条查询
 SELECT ... FROM T1 WHERE COL1 = ? AND COL2 = ?

<p>情景 #1:</p>	<p>T1 表有两个索引:</p> <ul style="list-style-type: none"> ● 哈希索引: (COL1, COL2) ● T 树索引: (COL1, COL2, COL3) <p>在这种情况下，两种索引都能用于返回这一查询。</p> <p>此处可以使用哈希索引，因为 WHERE 子句的各列与哈希索引的各列完全匹配；T 树索引也可用于返回该查询，因为 WHERE 子句的列是该索引列的前导前缀（前两个）。</p> <p>TimesTen 优化器将选择哈希索引（因为它速度更快）。</p>
<p>情景 #2:</p>	<p>T1 表有两个索引:</p> <ul style="list-style-type: none"> ● 哈希索引: (COL1, COL2, COL3) ● T 树索引: (COL1, COL2) <p>在这种情况下，只有 T 树索引能被用于返回这一查询。</p> <p>此处不能使用哈希索引，因为 WHERE 查询子句中没有指定哈希索引列</p>

	<p>(COL3)。而 T 树索引的列与 WHERE 查询子句中的列完全匹配。</p> <p>TimesTen 优化器将选择 T 树索引。</p>
情景 #3:	<p>T1 表有两个索引:</p> <ul style="list-style-type: none"> ● 哈希索引: (COL1) ● T 树索引: (COL3, COL1, COL2) <p>在这种情况下, 只有哈希索引能被用于返回这一查询。</p> <p>可使用哈希索引, 因为其所有的列均包含在 WHERE 查询子句之中; 注意由于 WHERE 查询子句中有另外的列, 所以在返回查询结果之前, 从哈希索引读取的每一行都会应用 “COL2 = ?” 谓词。</p> <p>不能使用 T 树索引, 因为 T 树首先使用 COL3 分类, 而查询未涉及 COL3。</p> <p>TimesTen 优化器将选择哈希索引。</p>
情景 #4:	<p>T1 表有两个索引:</p> <ul style="list-style-type: none"> ● 哈希索引: (COL1, COL2, COL3) ● T 树索引: (COL3, COL1, COL2)

在这种情况下, 两种索引都无法有效地返回查询结果 (尽管 T 树索引可用于全表扫描)。

出于上述情景#2 中同样的原因, 哈希索引无法使用。不能使用 T 树索引, 因为查询的列 (COL1, COL2) 不是索引的前导前缀。

由于两种索引都不能使用, 数据库将不得不执行一次表扫描以返回查询结果 (或可能建立一个临时索引), 这样查询性能将十分糟糕。

从这四个例子你不难看出, 你必须基于计划查询数据库的最频繁的查询来审慎选择你所创建的索引。

欲了解创建合适索引的更多信息, 参见 TimesTen 文档 “性能调优” 一章标题为 “调优语句和使用索引” 和 “适当选择哈希索引或 T 树索引” 的这两部分。

使用 “showplan” 验证是否使用了合适的索引来进行查询

如果你发现某一特定查询进行得比预期的慢, 可能是 TimesTen 优化器没有选择最优的查询计划来返回该查询。如何了解你的查询计划? 在 TimesTen 文档的 “TimesTen 查询优化器” 一章中的 “审视查询计划” 部分对这个问题有深入的阐述。

关闭自动提交(autocommit)并定期提交

在 ODBC 和 JDBC 中，默认情况下，与数据库的所有连接都会设为自动提交开启状态。这意味着每一独立的 SQL 语句都在其自己的事务中被提交。通过关闭自动提交，并在一次事务处理中明确提交多个 SQL 语句，将会极大地提高应用程序的性能。这对大批量操作尤其有意义，如大批量插入或大批量删除。（TTClasses 用户会注意到自动提交在默认情况下是关闭状态。）

然而，这可能会在一次事务处理中集中了过多操作（这会降低系统整体并发性，因为锁定的时间过长）。一般来说，大批量插入/更新/删除操作在你每次提交几千行时的工作效率是最高的。

C/C++ (ODBC)与 Java (JDBC)性能比较

TimesTen 的本机 API 是 ODBC。TimesTen Java JDBC 驱动程序是构建在 TimesTen ODBC 驱动程序之上的 II 型驱动程序。因此，使用 TimesTen ODBC 驱动程序的 C/C++ 应用程序的性能要稍优于使用 TimesTen JDBC 驱动程序的 Java 应用程序。对性能要求极高的应用程序应使用 TimesTen ODBC 驱动程序进行开发。

注意：TimesTen JDBC 驱动程序速度非常快。TimesTen JDBC 性能大大优于其他数据库的 JDBC 性能。TimesTen JDBC 驱动程序仅比 TimesTen ODBC 驱动程序慢一点点；而且差别是可测的。

ODBC 与 JDBC 间预计的性能差异受执行的查询类型(SELECT 与 UPDATE/INSERT/DELETE)和事务组合(读到写)的影响。影响性能的因素包括：

- 1) 为预先准备的语句绑定的参数的数量——增加绑定参数的数量会降低 Java 应用程序的性能。
- 2) SELECT 语句返回的列的数量——增加 SELECT 语句返回的列的数量会降低 Java 应用程序的性能。
- 3) 参数和列的类型——在 JDBC 中，包含大量字节的数据类型通常会比较慢。

基本上，随着在应用程序和数据库间传输的数据量的增加，Java 应用程序的性能将会降低。欲了解 ODBC 与 JDBC 间预计的性能差异的详细信息，请与 TimesTen 技术支持或你的 TimesTen 客户团队联系。

Java 固有的一些其他问题也会影响 Java 应用程序的性能：

- 1) JVM 垃圾回收——在 JVM 垃圾回收期间，Java 应用程序的响应将会极慢。对高性能的“实时”应用程序来说，这种垃圾回收将难以保证一致的性能。精心调优 JVM 垃圾回收对最大程度地降低或消除这些性能骤降是很有必要的。

-
- 2) Java 运行时内存管理——Java 运行时内存管理子系统通常会导致隐性数据拷贝，对于字符串对象情况更是如此。
 - 3) 高度多线程化的 Java 应用程序的扩展性——就 TimesTen 而言，我们的经验是：高度多线程化的 Java 应用程序的扩展性没有预期的那样好。这部分是由于 JVM 线程调度程序与操作系统的线程调度程序的设计差异。在多数情况下，使用多个 JVM 执行多个线程的整体性能要好于只使用单个 JVM 的情况。

我们还发现某些 JVM 要比另外一些 JVM 速度更快。请审慎评估你的选择。

通过在加载数据之后创建索引来加速（大批量）数据加载

如果应用程序的设计允许，你可以通过在加载数据后创建 T 树索引来最大限度地缩短加载数据所需的时间。在这种情况下，操作顺序为：

- 1) 数据加载到表（在表加载操作进行期间，禁用日志和/或使用数据库级锁将可提供更优的性能）
- 2) 创建 T 树索引
- 3) 更新统计信息
- 4) 准备查询语句

注意这只适用于极大批量的数据加载操作（例如，数百万行）。

使用 TTClasses，避免使用 OLEDB、ADO 和第三方中间件

许多第三方数据库接口软件包会降低数据库的性能。如果性能对某一特定应用程序不是特别重要，则可以使用这些接口程序，但用户应该明白此时性能会有所降低。

由于这些第三方软件包速度较慢，TimesTen 开发了自己的名为 TTClasses 的 C++ ODBC 封装类，它已包含在 TimesTen 数据库中。欲了解关于 TTClasses 的更多信息，请参见第 1 页的“TimesTen 文档参考索引”。

多 CPU 性能调优

要实现最佳的 SMP（对称多处理）性能，还有许多其他事情要对应用程序正确加以处理。这一部分专门讨论在多 CPU 机器上实现最佳性能还需要做哪些事情。

使用连接池

如果你的 TimesTen 应用程序是多线程的，并且打开了到同一数据库的多个连接，你一定要谨慎管理你的这些连接。一般来说，如果同时连接到数据库的激活连接数多于 CPU 数目时将难以获得最佳性能；事实上，多个 TimesTen 并发连接（连接到同一数据库）会极大地降低数据库的整体吞吐能力。

避免这一问题的一个方法就是使用连接池。连接池的一个很好的例子就是 TTClasses 中的 TTConnectionPool 类。欲了解关于 TTClasses 的更多信息，请参见第 1 页的“TimesTen 文档参考索引”。

最大限度地提高数据库的并发性

对任何数据库系统来说，如果一些用户占用了太多的系统资源，就会影响到其他用户的使用性能。下面是一些常见的例子和简单的解决方案。

及时关闭只读游标

与 Oracle 类似，TimesTen 只读事务无须提交。然而，为了释放一个只读 SQL 查询所占用的所有资源（如分类所占用的临时空间），及时关闭只读游标就显得非常重要了。

下面是关闭 SQL 游标的方法：

- [ODBC] SQLFreeStmt(SQL_CLOSE)
- [TTClasses] TTCmd::Close()

[JDBC] PreparedStatement.close()

避免大批量的删除语句

避免使用 “DELETE FROM <table>” 语句

如果一个表有许多行（10 万行或更多），试图在一个单一 SQL 语句（在一次事务）中删除它们是有问题的。首先，这种方法速度较慢。TimesTen 将删除的每一行都记入日志，以防万一操作需要回滚；写入所有这些日志记录会非常耗时（因为要对磁盘进行写操作）。

这种大批量的删除操作还存在另一个问题，那就是在发生写密集型删除事务时，其他数据库操作将会变慢。在单次事务中删除数百万行需要好几分钟才能完成。

当对表进行复制时，一次删除数百万行就会遇到第三个问题。由于复制仅传输已提交的事务，传输一个单一的几百 MB（或 GB）的事务会使复制代理的速度变慢。TimesTen 的复制被优化为许多小型事务，在一次事务中删除数百万行将会使其速度变慢。

首选“TRUNCATE TABLE”语句

考虑采用 TRUNCATE TABLE 语句，而不采用删除表中的所有行的方法。执行该语句的最终效果与执行不带 WHERE 子句的 DELETE 语句相同，这极大地减少了日志记录。此外，在使用复制时，TRUNCATE 操作是作为一个单一操作被复制给任何其他数据库的，而不是每删除一行就视为发生一个操作。

考虑使用“DELETE FIRST NumRows”

当必须删除大量的行而 TRUNCATE 语句又不适用时，考虑使用“FIRST NumRows”子句从一个表中批量删除行。“FIRST NumRows”句法使你能够将“DELETE FROM TableName WHERE ...”语句变成一系列“DELETE FIRST 10000 FROM TableName WHERE ...”操作。

通过将大量删除操作分割成一批小型操作，行删除速度会快得多，系统的整体并发性和复制性能也不会受影响。

缩短不必要长期运行的事务

如第 18 页“处理死锁和锁超时错误”所述，一个长期运行的事务会使其他数据库操作因锁超时错误而失败。一般而言，长期运行的事务会降低系统的整体并发性，这是由于资源在很长的时间内保持锁定状态。因此，长期运行的事务会影响系统整体稳定性和整体性能。

最大限度地提高稳定性

最大限度地提高稳定性

本章重点讨论如何编写强健性和稳定性俱佳的应用程序，也就是编写尽可能长时间与数据库保持连接的应用程序。

TimesTen 文档

TimesTen 现有的文档含有关于开发容错性强的应用程序的大量信息。感兴趣的读者可以参考 TimesTen 文档的“事务管理和恢复”、“TimesTen 内置程序”和“诊断、错误和警告”这三章。

TimesTen 架构和数据库恢复简明指南

TimesTen 使用一个共享的内存段来在内存中存储 TimesTen 数据库的部分内容。当一个应用程序连接到 TimesTen 数据库时，它会将该数据库的共享内存段映射到它的地址空间。共享内存的使用使多个进程连接到一个单一的数据库映像。

在使用共享内存的传统应用程序中，每一个连接到共享内存的应用程序进程的稳定性会极易影响共享内存的结构。一个不稳定的应用程序常常会导致共享内存错误和应用程序失败。当应用程序正在更新共享内存时必然被（例如，由“kill -9”命令）终止，就会发生这种情况。这会使共享内存处于一种不一致的状态，数据结构仅得到部分更新。

TimesTen 通过使用 Micrologging™ 技术来防止某个应用程序的意外失败对其他应用程序的损害，从而避免此类问题的发生，即使是在使用共享内存的情况下。如果某个应用程序在修改 TimesTen 共享内存时失败，TimesTen 能够检测到这种情况并能够使共享内存回滚到一致、正确的状态。

尽管 TimesTen 能够从应用程序的意外失败中恢复，然而这么做会占用本来能够更好地用于运行应用程序的资源。避免这种开销其实很容易，只需在应用程序开发过程中遵循以下几个简单步骤。

避免应用程序的意外失败

以下建议对设计 TimesTen 应用程序会很有帮助。

必须断开 TimesTen 应用程序与数据库的连接

有许多方法可以用来终止应用程序。一种方法是调用 `exit()` 函数或 `abort()` 函数或到达 `main()` 函数的结尾处；另一种方法是收到一个信号。如果进程接收到一个不能捕获的信号，它就会终止，而且可能使数据库失效。

在终止之前，应用程序应回滚任何正在处理中的事务，并断开与 TimesTen 数据库的连接。

因此，使用 TimesTen 的 C 和 C++ 应用程序应实施一个信号处理器，该信号处理器捕捉其他进程可能发给该应用程序的所有信号。最好的方法就是通过一个特定的信号处理线程来做这件事。

某些应用程序把信号用于进程间通信；这样接收到一个信号未必就意味着要终止进程。

对于 Java 应用程序，可使用“shutdown hooks”来达到同样的目的。

一般来说，与 TimesTen 一起提供的示例说明了如何恰当地使用信号。可以看一下“ttCGmonitor” (`monitor.cpp`)，它是一个很好的例子。对于 Java 应用程序，“Tutorial”目录下的示例说明了如何恰当地使用“shutdown hooks”。

避免对 TimesTen 应用程序使用“kill -9”命令

SIGKILL 信号（这相当于在 Windows 平台上使用任务管理器中的“结束进程”）不能被进程捕获。“kill -9”强行终止目标进程，在此之前目标进程无法做任何反应。

因此，我们不主张对 TimesTen 应用程序执行“kill -9”命令。

你应该这样设计你的应用程序和系统，那就是在生产环境中永远没有必要使用“kill -9”命令。这是一个很好的主意。

备份

对于任何数据库而言，定期进行备份对保证数据完整性是很有必要的。备份数据库并将这些备份存储在可移动介质或另一组永久性挂载介质上，将降低

由于数据库硬件（如，磁盘驱动器）故障导致数据丢失的可能性。

TimesTen 的 ttBackup 程序使数据库备份非常容易进行。备份可在后台进行，而同时可继续执行正常的数据库操作。备份不会锁定数据库，因此这是一个高效的过程。

检查点

周期性地进行检查点操作对 TimesTen 数据库的运行非常重要。调整检查点频率会对某些应用程序有益。

TimesTen 在磁盘上维护了每个数据库的内容的两个完整的镜像。这些镜像称为“检查点”。TimesTen 定期自动更新磁盘上的这些检查点，使它们与当前的数据库内容保持同步。这些同步操作是在后台异步进行的，因此不会涉及到应用程序。这些周期性的检查点操作之所以重要，有两个主要原因。

- (1) 检查点操作使 TimesTen 能够释放事务日志的空间。通过删除旧的事务日志文件，磁盘就不会占满，并保持 TimesTen 继续顺畅运行（参见第 19 页“从一个已满的磁盘恢复”，了解监视磁盘空间的重要性）。
- (2) 删除日志文件还可减少在系统失败时恢复数据库的时间。这是因为 TimesTen 将数据库加载到内存所需的时间与磁盘上日志文件的数量成正比。

默认情况下，TimesTen 每隔 600 秒（10 分钟）或每当写满 64MB 日志数据时，对数据库进行一次检查点操作。上述任一种条件满足时，TimesTen 就进行这种操作。如果需要，可以使用 CkptFrequency 和 CkptLogVolume 数据库属性来改变这种行为方式。

其他好的实践

检查所有 ODBC 函数的返回码，然后处理它们

在使用 ODBC 时，有必要检查所有 ODBC 函数的返回码。如果检测到警报或错误，你的应用程序应恰当地处理它们。接下来的两部分将描述将会出现的一些最重要的错误。

想要查看全部错误码和 TimesTen 用于描述这些错误码的助记符，请参见文件：`install_dir/include/tt_errCode.h`。

关于应用程序通常如何处理错误的简单描述，请参考“事务管理和恢复”一章标题为“处理应用程序中的 TimesTen 错误和恢复”的部分。

当使用 JDBC (Java)和 TTClasses (C++)时,如果发生了与数据库有关的错误,在大多数情况下异常会被“抛出”。这些错误必须使用适当的 try/catch 块来加以处理。

处理数据库失效错误

同任何数据库一样, TimesTen 数据库也有可能失效。当应用程序所连接的数据库意外“宕机”时,应用程序必须处理这种情况。

有许多原因会导致数据库失效。可能是管理员终止了 TimesTen 实例。也可能是一个重大的硬件错误影响了数据库。或者, TimesTen 无法使用 Micro Logging 回滚一个失败的应用程序提交的改变,这种情况比较少见。如果发生这种情况, TimesTen 将使数据库“失效”,并要求所有应用程序停止使用数据库。

有两个 TimesTen 错误码用于指示应用程序连接到了一个失效的数据库: 846 (tt_ErrBadConnect) 和 994 (tt_ErrDrtyByte)。当应用程序接收到这两个错误码中的任何一个时,应用程序必须回滚它们已经开始处理的所有事务,在继续处理之前必须断开与数据库的连接并重新进行连接。

一个简单有效的处理数据库失效错误的方法可在 ttMonitor 的 C++源码中找到(文件名: monitor.cpp),该文件与 TimesTen 一起提供。

处理死锁和锁超时错误

通常采用同样的方法处理死锁和锁超时,尽管它们通常是由不同的原因引起的。

当数据库中两个并发的任务争用相同的资源,并且每个任务都占用了另一任务继续进行所必需的资源时,就会发生死锁错误。死锁发生后,要使任务处理得以继续,唯一的方法就是由数据库选择一个“牺牲品”,并向该任务发送死锁错误。TimesTen 死锁错误码为 6002 (tt_ErrDeadlockVictim)。

当数据库中两个并发的任务争用相同的资源,而其中一个任务未能在给定的时间内获得该资源时,就会发生锁超时。通常这种情况的发生是由于一个任务长期运行而不进行提交(从而长时间占用锁),进而妨碍其他任务锁定长期运行任务已锁定的资源。无法获得锁的任务将收到 TimesTen 错误码 6003 (tt_ErrTimeoutVictim)。

设计糟糕的应用程序很容易陷入死锁状态。例如,如果多个执行线程总试图同时修改相同的行集,就很容易发生死锁。

TimesTen 实用程序 ttXactAdmin 是诊断数据库并发问题的一个良好起点。欲

了解有关 ttXactAdmin 的信息，请参阅 TimesTen 文档“TimesTen 实用程序”一章中的相关描述。

TimesTen 提供了一些很好的工具，应用程序开发人员可利用它们来减少数据库的并发问题。这些工具包括：

- TimesTen READ_COMMITTED 隔离级别，实现了“非阻止读取”操作；即读取者与写入者互不阻碍。
- SQL 语句 SELECT ... FOR UPDATE，当应用程序在执行 SELECT 后，希望立即对所提取的行执行 UPDATE 时，该语句可实现良好的并发性。

欲了解有关这些 SQL 语句的更多信息，请参见《TimesTen API 和 SQL 参考指南》中的“SELECT”和“UPDATE”部分。

从一个已满磁盘恢复

从一个已满磁盘恢复是个棘手的问题。当包含 TimesTen 事务日志的磁盘已满时（由于检查点操作间隔过长，或复制或 XLA 落后），由于已没有任何可写入后续日志文件的空间，因此数据库将失效（请参见上文），连接也将无法自动重新连接。

一般情况下，当 TimesTen 事务日志文件写满一张磁盘时，必须释放一些磁盘空间（通过某种方式），然后连接到数据库并执行两个快速检查点操作（以删除累积起来的多余日志文件）。

注意，如果是由于复制和/或 XLA 导致事务日志文件累积而占满磁盘，则需要先清除复制和/或 XLA 书签，然后通过检查点操作成功清除事务日志文件。

复制和 XLA

TimesTen 的复制功能十分强大、灵活，使系统设计人员能开发出可用性高、容错性强的应用系统。

TimesTen XLA 用于监控对 TimesTen 数据库范围内的一个或多个表所做的更改。

本章首先介绍复制功能，然后再介绍 XLA。

复制

本章主要介绍某些在设计使用 TimesTen 复制和 XLA 功能的系统时需要了解的重要问题。

有关复制功能的信息的另一个资料来源是《在线升级指南》，该文档可通过 TimesTen 专业服务获得。

将 DSN 名用作文件名前缀

这不是必需的，但却是避免不必要混淆的好方法。

在进一步阐述之前，我们需要定义“文件名前缀”术语。

“文件名前缀”的定义

在 UNIX 平台下，DSN myDSN 如下所示：

```
[myDSN]
Driver=/opt/TimesTen/tt60/lib/libtten.so
Datastore=/directory/ds_file_prefix
```

myDSN 的文件名前缀为“ds_file_prefix”。

在 Windows 平台下，在 ODBC 控制面板中，情况很相似；如果将 myDSN 配置到以下路径，那么在标为“数据库路径和名称”的框中有：

C:\directory\ds_file_prefix

同样，myDSN 的文件名前缀为“ds_file_prefix”。

TimesTen 复制配置使用文件名前缀

复制配置命令引用数据库的文件前缀名。在基于 SQL 的复制配置命令中，使用方法如下：

```
CREATE REPLICATION rep.name ELEMENT e TABLE rep.t1 MASTER <一些数据库的文件名前缀> ON <计算机名称> ...
```

当 ttRepAdmin 命令行实用程序需要引用数据库的“名称”时也使用数据库的文件名前缀：

```
ttRepAdmin -duplicate -from <其他数据库的文件名前缀> -host <计算机名称>  
> dsn=myDSN
```

此外，在 TimesTen 复制中，<文件名前缀,主机名>对必须是唯一的，即如果两个数据库具有相同的文件名前缀，便不能在同一台计算机上对它们进行复制。

结论：始终将文件名前缀设置为与 DSN 相同！

由于以上原因，始终将数据库的文件名前缀设置为与其 DSN 名称相同是很方便的：

- 如果无需不断跟踪 DSN 和文件前缀名两者，那么所有复制配置命令的编写都会变得更加简单。
- <文件名前缀,主机名>对必须唯一不再是问题，因为 ODBC 已确保了<DSN,主机名>的唯一性。

在进行 - duplicate 操作之前执行两个检查点操作

TimesTen 建议在执行 ttRepAdmin - duplicate 操作前，先在主数据库进行两个显式的检查点操作。使用 ttCkpt 内置程序强制执行检查点。

在主数据库执行检查点操作可清除不必要的事务日志文件，这相应减少了复制操作过程中需要传输的数据量，从而加快了 -duplicate 操作的速度。

复制配置应（手动）指定端口号

如果你在线升级你的 TimesTen 应用程序，只是 TimesTen 本身的版本得到了

升级，复制配置命令/脚本还必须手动指定复制端口号。

这是因为 TimesTen 有个有趣的缺陷/特性，即它会阻止在线升级：如果不手动指定复制要使用的端口号（即让复制动态分配一个端口号），不同 TimesTen 版本之间的复制将无法成功进行通信。

欲了解有关在线升级的更多信息，请咨询 TimesTen 专业服务，获取名为《在线升级》的文档。

监控复制

监控 TimesTen 复制以确保高可用性很重要。在复制过程中，应用系统需监控以下几方面：

- 首先，如果复制发送方与接收方之间的连接出现故障（网络连接中断或接收方节点失效等），那么发送方需确定在某一点停止向该接收方复制。这一点的确定完全取决于应用系统及其要求。一般情况下，如果复制计划中的两个节点长期不同步，那么通过删除接收方的数据库并利用“`ttRepAdmin -duplicate`”操作重新初始化来重建接收方节点，复制会更快一些。
- 其次，如果在复制计划的接收方存在长时间、高处理速率的数据库更新/插入/删除操作，那么 TimesTen 复制操作将会落在后面。由于网络通信自身的性质，因此很容易出现复制无法跟上以极高的插入/更新/删除速率运行的 TimesTen 的状况。TimesTen 复制会努力以尽可能快的速度追赶，但是如果处理速率一直不减慢，复制操作将会越来越落后。

某些类型的应用程序可能会在系统初始化时执行数量极大的数据库操作，这样处理速率就有可能降到一个较低的水平。这种情况下，在接收方节点执行“`ttRepAdmin -duplicate`”并仅在发送方节点完全填充后启动复制可能是更明智的做法。

这两种情况（与复制接收方之间的通信出现故障、复制落后）都要求复制发送方有额外的磁盘空间来存储那些包含等待复制的数据的事务日志。在这种情况下，检查点操作将不会删除所有的日志文件，而是仅删除最近的日志文件，因此应用程序可能会面临磁盘空间用完的危险。

复制用于保留其所需的日志文件，以避免它们被检查点操作删除的机制称为复制书签。要在命令行下查看某个数据库的复制书签，请执行“`ttRepAdmin -bookmark`”命令。应用程序应定期检查未复制数据的累积情况，并在复制落后于多个日志文件时采取适当的措施。

可考虑为复制配置一个“阈值”，以便在接收方过于落后时自动停止保存日志。

欲了解有关 ttRepAdmin 的更多信息，请参阅 TimesTen 文档“TimesTen 实用程序”一章中关于 ttRepAdmin 的描述。有关 ttBookmark 的信息可在 TimesTen 文档“TimesTen 内嵌程序”一章中找到。

SEQUENCE 与复制和故障恢复的相互影响

每个节点的 SEQUENCE 都是独立的，它们不会重复。因此，在数据库 db1 中，一个已命名的序列可能被定义为从 1 到 100000；在数据库 db2 中，一个已命名的序列可能被定义为从 100001 到 200000。因此，举例来说，每个节点的序列可用于填充一个表的主键列，这样每个节点就可以在互不冲突的情况下执行表插入操作。

出现连接故障后，如果允许复制“正常”恢复（即通过“追赶”或重放已保存的日志文件），则不需要执行任何特殊操作。然而，如果故障节点出现故障“过久”，恢复操作则需要使用“ttRepAdmin -duplicate”将未出现故障的节点的数据重新填充到故障节点的数据库中。（如果事务处理队列过长，这是必要的操作。）出现这种状况时，“ttRepAdmin -duplicate”命令将把序列定义从一个节点复制到另一个节点，在我们例子中，最终结果是分配给两个节点的序列值都位于 100001 到 200000 之间，而这正是我们试图避免的。

为了避免出现这种情况，执行完“ttRepAdmin -duplicate”命令但在启动应用程序之前，故障恢复脚本必须删除并重新创建所有序列号，以便其取值范围再次处于不重叠的状态。

借助这个简单的技巧，可将 SEQUENCE 用于复制设置来确保键值的唯一性（但不是密度和时间的单调递增）。

XLA

使用持久性 XLA

TimesTen 支持两种类型的 XLA：持久性 XLA（通过 ttXlaPersistOpen 函数连接）和非持久性 XLA（通过 ttXlaOpenTimesTen 函数连接）。

持久性 XLA 几乎在各方面都优于非持久性 XLA。

注意：正如本节所讨论的，应当避免使用非持久性 XLA (ttXlaOpenTimesTen)，而应使用持久性 XLA (ttXlaPersistOpen)。本章所讨论的有关 XLA 的所有内容都指持久性 XLA。

始终监控 XLA

和复制一样，XLA 也需要谨慎地监控。

监控 XLA 进程的重要性

每个 XLA 连接在事务日志中都保留有一个书签。如果这些 XLA 进程中的一个进程终止，其书签会继续保留在日志中，除非前移或删除该书签，否则检查点操作将无法删除该日志文件（以及后来的所有日志文件）。

这样，终止的持久性 XLA 进程将导致日志文件的增加，进而导致磁盘空间被占满，因此，当无法再写入日志文件时，这将导致数据库失效。19 页的“从一个已满磁盘恢复”描述了如何从一个已满磁盘恢复。

TTClasses 提供一个有用的命令行实用程序，称为 `ttXlaAdmin`，该实用程序列出数据库中的 XLA 书签（显示每个书签占用的日志文件的数量），并允许用户在需要时删除它们。欲了解有关 TTClasses 的更多信息，请参见第 1 页中的“TimesTen 文档参考索引”。

确认 XLA 更新（使用 `ttXlaAcknowledge`）的重要性

这是对前文所述内容的总结。

XLA 的持久性是通过在事务日志中对某个特定的 XLA 读取器已读取（和未读取）的记录保留书签来实现的。但请注意，当 XLA 读取器接收来自一组 XLA 读取器的记录时，该书签不会被前移。这样做是为了防止出现 XLA 读取器还未对这些记录进行处理便终止的情况。

为了前移书签，XLA 读取器必须显式地“确认”一批 XLA 记录。这通过 XLA 函数 `ttXlaAcknowledge` 来实现。如果用户程序不定期调用该函数，XLA 书签将永远不会前移，并会出现前文所描述的磁盘空间用尽的问题。

TTClasses 方法 `TTXlaPersistConnection::ackUpdates()` 是 `ttXlaAcknowledge` 函数的封装器。有关使用 `ackUpdates` 的示范说明，请参见 TTClasses 示例程序 `xlaSimple`。欲了解有关 TTClasses 的更多信息，请参见第 1 页中的“TimesTen 文档参考索引”。

但是，对该函数调用不能过于频繁，下面的一节对 XLA 性能进行了阐述。

最大限度地提高 XLA 性能

XLA 是一个灵活、功能强大的框架，用于接收关于对数据库中的一个或多个表所做的更改的通知。但是，如果不考虑以下问题，XLA 应用程序的性能将

会受到影响。

使用 `ttXlaAcknowledge`，但不要过于频繁

如前文所述，XLA 函数 `ttXlaAcknowledge` 是所有 XLA 应用程序的基本组成部分：它前移 XLA 书签，使事务日志能通过检查点操作清除。

但 `ttXlaAcknowledge` 耗费的资源相对较多，因此不要 `ttXlaNextUpdate`（或 `ttXlaNextUpdateWait`）每返回一批 XLA 记录就调用一次该函数。

该函数的调用频率取决于应用程序业务逻辑对恢复和性能要求的权衡取舍。对于吞吐量要求相对较低的应用程序来说，`ttXlaNextUpdate[Wait]`每返回 10 次记录调用一次 `ttXlaAcknowledge` 即可；但在需要保证极高的吞吐量时，建议这两个函数间的比例为 1:100。

Index

Symbols

"file name prefix" definition 21

"TimesTen application" definition 1

"TimesTen Client/Server application" definition 2

C

checkpointing

importance 19

connection pooling 12

D

deadlock errors 18

disk full errors, recovery 19

I

indexes

creating the right ones 8

speeding up data loads 12

install_dir 2

invalidation

causes 5, 15, 16

error codes 18

how to avoid 16

importance of signal handlers 16

J

JDBC

performance 10

K

kill -9, cause of data store invalidation 16

L

living document 1

lock timeout, setting the interval 19

long transactions 14

M

monitoring replication, importance of 19

N

notation 1

P

preparation

importance of 6

Q

query plans, viewing 10

R

replication

importance of monitoring 19

use DSN as file name prefix 22

use manual port number 22

S

sequence, interaction with replication 24

short transactions 14

showplan 10

signal handlers 16

signal handling, importance of 16

statement preparation

importance of 6

T

timeout errors 18

transactions 14

TTClasses 12

TTConnectionPool 12

ttLockWait 19

ttXlaAdmin 25

U

updating statistics 6

W

whitepaper on datastore recovery 5, 15

X

XLAacknowledging updates, importance of 25

always monitor it 25

limit to one reader 26

maximizing performance 25

multi-user 25

non-persistent 25

persistent 25

single-user 25

ttXlaAcknowledge 25, 26

