

Secrets of the Oracle Database

未公开的 Oracle数据库秘密

[德] Norbert Debes 著
蒋海鸥 李浩 金海 译

- 揭秘大量鲜为人知的特性
- Oracle文档系统和Metalink支持平台的完美补充
- 大师秘笈助你内外兼修



人民邮电出版社
POSTS & TELECOM PRESS

未公开的 Oracle数据库秘密

Oracle数据库体系庞大、功能复杂，有些特性用处很大但没有在相关文档中说明，有些特性非常重要却只作了部分说明，这给开发人员造成很大困扰。Oracle数据库的经典著作虽然不在少数，但目前还没有一本专门讲述文档中未说明或未充分说明的特性的书。本书作者Norbert Debes就此进行了出色的尝试和探索，在书中揭示了未在文档中说明的内部算法、PL/SQL包、参数、调试接口等。此外，本书提供了免费扩展SQL跟踪分析器ESQLTRCPROF，它能够帮助开发人员快速诊断和解决性能问题。

本书将是Oracle用户手中的一把利器。

Apress®

图灵网站: www.turingbook.com 热线: (010)51095186转604
反馈/投稿/推荐信箱: contact@turingbook.com
有奖勘误: debug@turingbook.com

分类建议 计算机/数据库/Oracle

人民邮电出版社网址: www.ptpress.com.cn



ISBN 978-7-115-26016-1



9 787115 260161 >

ISBN 978-7-115-26016-1

定价: 69.00元

TURING

图灵程序设计丛书 数据库系列

Secrets of the Oracle Database

未公开的 Oracle数据库秘密

[德] Norbert Debes 著
蒋海鸥 李浩 金海 译

人民邮电出版社
北京

更多资源请访问稀酷客(www.ckook.com)

图书在版编目 (C I P) 数据

未公开的Oracle数据库秘密 / (德) 迪贝斯
(Debes, N.) 著 ; 蒋海鸥, 李浩, 金海译. -- 北京 : 人
民邮电出版社, 2011.8

(图灵程序设计丛书)

书名原文: Secrets of the Oracle Database

ISBN 978-7-115-26016-1

I. ①未… II. ①迪… ②蒋… ③李… ④金… III.

①关系数据库系统: 数据库管理系统, Oracle IV.

①TP311.138

中国版本图书馆CIP数据核字(2011)第141580号

内 容 提 要

本书讲述了 ORACLE 数据库管理系统中大量鲜为人知的特性, 并详细生动地揭示了如何将这些特性集成到 ORACLE DBMS 中。书中主要对初始化参数、数据字典基表、事件、XS 固定表、SQL 语句、提供的 PL/SQL 程序包、应用程序开发、性能、Oracle 网络、实时应用集群、实用工具等进行了深入的探讨, 旨在提出一种解决问题的结构化方法, 引导读者灵活自如地解决 ORACLE DBMS 中遇到的棘手问题。

本书适合具有一定 ORACLE DBMS 管理经验的 ORACLE 数据库管理员阅读。

图灵程序设计丛书

未公开的Oracle数据库秘密

◆ 著 [德] Norbert Debes

译 蒋海鸥 李 浩 金 海

责任编辑 朱 巍

执行编辑 刘美英

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号

邮编 100061 电子邮件 315@ptpress.com.cn

网址 <http://www.ptpress.com.cn>

三河市潮河印业有限公司印刷

◆ 开本: 800×1000 1/16

印张: 27.75

字数: 690千字

2011年8月第1版

印数: 1-3 000册

2011年8月河北第1次印刷

著作权合同登记号 图字: 01-2009-7784号

ISBN 978-7-115-26016-1

定价: 69.00元

读者服务热线: (010)51095186转604 印装质量热线: (010)67129223

反盗版热线: (010)67171154

版 权 声 明

Original English language edition, entitled *Secrets of the Oracle Database* by Norbert Debes, published by Apress, 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705 USA.

Copyright © 2009 by Norbert Debes. Simplified Chinese-language edition copyright © 2011 by Posts & Telecom Press. All rights reserved.

本书中文简体字版由Apress L. P.授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

译者序

ORACLE 是目前世界上最流行的大型关系数据库管理系统之一，它以高级结构化查询语言 (SQL) 为基础。目前，关于 ORACLE 数据库的书籍琳琅满目，但是鲜有专门针对未在相关文档中说明的高级特性的技术书籍。虽然开发者可以通过 Internet 或 ORACLE 的 Metalink 支持平台获得相关信息，但这会造成时间和金钱的大量浪费。本书是对 ORACLE 庞大文档系统和 Metalink 支持平台的有利补充和完善，旨在为大家揭示 ORACLE 数据库大量鲜为人知的特性，提升数据库的性能和功能。

本书是一本技术性较强的书籍，需要读者具有一定的数据库管理经验。本书组织结构极为清晰，每章开头都会介绍某个鲜为人知的特性所能带来的好处，以及与该特性相关的文档说明程度，并包括与该章主题相关的 ORACLE 数据库手册的引用。同时，每章最后都附有源代码文件及功能，这样读者就能及时地实现相关功能并加以巩固。阅读完本书，我们会知道这些没有相关文档说明的特性是如何集成到 ORACLE DBMS 中的，最重要的是我们能够学习如何从扩展的 SQL 跟踪数据得到一个基本的性能诊断方法。通过本书的学习，你也有能力继续揭秘 ORACLE 中更多鲜为人知的有用特性。也就是说，本书能教给读者一种举一反三的结构化方法。

本书的翻译工作得到了华中科技大学“服务计算技术与系统教育部重点实验室”和“集群与网络计算湖北省重点实验室”各位老师和同学的热心帮助，特别感谢金海老师、袁平鹏老师、谢夏老师、蒋文斌老师等实验室各位老师对我们的指导和无微不至的关心。

感谢李浩同学为该书作出的巨大贡献，没有他的辛勤付出就没有该书的出版，与他的合作非常愉快。同时，非常感谢好友潘李敏，感谢她参与了该书部分章节的翻译和审核工作，并提出了许多宝贵的建议；也非常感谢蔡亚男，一些章节因为她的参与和指导性建议变得通俗易懂、更加精彩。

最后，感谢图灵公司给了我们翻译这样一本好书的机会，感谢编辑为本书出版所做的细致入微的工作，特别感谢李松峰老师和傅志红老师为本书翻译提出的宝贵建议。衷心希望该书能让读者更便捷地获取信息、找到所求。

蒋海鸥

2011 年 6 月 23 日于百度大厦

序

如果你对某项技术拥有 10 多年的经验，这听起来会让人印象深刻。然而，如果你有超过 20 年的经验，这听起来就有点可悲了。可悲的是，我从事 Oracle 数据库软件工作将近 21 年了，接触的第一个 Oracle 数据库版本是 5.1。

在我工作的早期，Oracle 文档集只包括很少的手册。万维网（World Wide Web）还是 5 年之后才出现的事物，大部分人甚至都不能访问类似 comp.databases.oracle 的 USENET Internet 组。还没有关于 Oracle 的书出版。除了偶尔打电话给 Oracle 技术支持，你基本上是在孤军奋战。

从一开始，Oracle 就比当时的其他软件（比如 VMS、UNIX 或者笨拙的 MS-DOS）更神秘。Oracle 是如何工作的？这个 SQL 为什么如此慢？我该怎么才能让 Oracle 运行得更快？对 Oracle 秘密的求知欲使那些将 Oracle 软件作为事业基础的人们如痴如醉。

在整个 20 世纪 90 年代，Anjo Kolk、Cary Millsap 等先驱者以及其他人都试图解释 Oracle 的内部工作方式，以及如何提升其性能和功能。Web 出现了，有了它，Oracle 技术人员有了一个能够交换信息的活跃社区。标准 Oracle 文档集随着每个版本的发布都有所增长，到了现在的 Oracle Database 11g，仅数据库集就包括 140 本书。曾经如涓涓细流般出版的技术书在 Oracle5 和 Oracle6 问世期间，如洪水般蜂拥而至，信息的匮乏变成了信息的泛滥。

也许你认为我们无所不知了。然而，Oracle 让我想起了电视剧 *Lost*，在每一集中，问题总会得到解决，但是新的问题又会出现。当你认为已经理解了数据库引擎是如何使用内存进行排序时，Oracle 引进了自动 PGA 内存管理（Automatic PGA Memory Management）；当你认为已经理解了锁（latch）时，Oracle 引进了互斥锁（mutex），等等。想要了解 Oracle，让它工作得更快更好，就不要停止学习的脚步。

那就是我非常喜欢 Norbert 书的原因。有些书总结和阐明了 Oracle 的文档化（documented）行为——Tom Kyte 非凡的 *Expert Oracle Database Architecture* 一书就是非常好的例子。它包含了很多能在 Oracle 文档集和其他地方找到的资料，组织和总结了这些信息。其他一些书，比如 Jonathan Lewis 的杰出著作 *Cost-Based Oracle Fundamentals*，将相关文档信息和原始研究结合起来，成为碰到特殊问题的权威参考。Norbert 则不同，他试图说明 Oracle 软件的一些非常重要但鲜为人知的方面。

在这本书中，Norbert 揭示了目前为止还没有相关文档说明的内部算法、PL/SQL 包、参数、调试接口等。然而，这不是一本琐碎的书，被揭示的每个秘密都有特定的应用场景。阅读本书让我受益颇多，我非常乐意将本书推荐给所有像我一样想去探寻 Oracle 秘密的人。

Guy Harrison, Quest Software 企业发展总监

致 谢

非常感谢以下人提供的意见、建议和给予的鼓励：Helga Debes、Lise Andreasen、Pete Finnigan 和 William Kehoe。特别感谢 Iggy Fernandez 把我介绍给 Jonathan Gennick，也非常感谢 Guy Harrison 为本书作序。同样感谢我所有的客户以及他们对我的信任。

引言

本书汇集了大量未在文档中说明或部分在文档中说明的 ORACLE DBMS (Database Management System, 数据库管理系统) 的特性, 我写本书就是要揭示 ORACLE 数据库服务器的这些隐藏特性。关于这些高级的未说明的主题, 你也许很难在其他书上找到这么多的介绍。当然, Internet 上也有关于某些隐藏特性的文章, 但是, 我给大家提供的是比互联网上更多的背景知识和更深入透彻的实例。本书还包括一系列原创的资料, 如资源配置文件中系统性能诊断的思考时间, 从 RAC 安装向单一实例安装紧急转换的流程, 以及用 SQL 跟踪对 Statspack、自动工作负载信息库 (Active Workload Repository) 和活动会话历史 (Active Session History) 进行集成。

本书是对 Oracle 公司庞大的文档系统和 Metalink 支持平台上所发表文章的补充。极有可能, Oracle 文档因为遗漏了某些特性而被认为有缺陷。许多特性, 特别是那些用于故障处理 (例如, 事件) 和追踪的特性, 仍然有意地未在相关文档中提及, 因为 Oracle 公司有充分的理由认为, 如果对这些特性使用不当或者没有深入理解这些特性的作用将会适得其反。这些特性并不是本书的重点。本书的重点是研究那些在文档中未说明的特性, 而这些特性能在不影响数据库完整性和可用性的前提下带来显著好处。

在本书中, 某个特性在文档中未说明是指在对 Oracle Technology Network® 网站提供的文档进行全文搜索时没有得到任何关于特性的提示, 而那些能被全文搜索到, 但其重点在文档中未说明的 (这样会导致其使用受限) 特性认为是在文档中部分说明的特性。文档不完整使大家不得不借助 Internet 或 Oracle 的 Metalink 支持平台, 通过不断的尝试和失败来领悟那些未在文档中说明的信息, 这会造成时间和金钱的大量浪费。而本书揭秘的大部分未在文档中说明的特性都没有在 Metalink 文章中出现。

本书是一本技术性较强的图书, 我会不遗余力地让读者不需要太多背景知识就能轻松上手, 采用清晰简明的写作风格, 并充分结合实例来描述。偶尔也会用幽默的语言刺激一下你的右脑, 让你具有分析能力的左脑在解决更多技术难题之前得到放松。

虽然本书并不是一本针对 ORACLE DBMS 性能优化的书, 但我的意图是通过分析扩展的 SQL 跟踪数据提供可靠的性能诊断方法。据我所知, 本书是第一本涉及 Oracle10g 和 Oracle11g 扩展 SQL 跟踪文件格式 (这与 Oracle9i 使用的文件格式在很多重要方面有所不同) 的书。我非常希望, 本书免费提供的扩展 SQL 跟踪分析器 ESQLTRCPROF 能够帮助你快速诊断和解决面临的困难性能问

① 参见 <http://otn.oracle.com>。

题。据我所知,ESQLTRCPROF 是唯一将等待事件 SQL*Net message from client 分类为因客户端/服务器通信而造成的不可避免的延迟,以及由客户端处理非相关数据库而造成的思考时间(think time)的分析器。分析器的可配置特性仅仅在需要证明 ORACLE DBMS 不是性能问题的原因时非常有用。由于不能优化思考时间,除非重新编码应用程序或者其他等待的应用程序,对思考时间的适当确认也有助于估计通过调节客户端和 DBMS 实例之间的交互而得到的最大加速比。

我在书中提供了一些操作系统、网络和编程相关的背景资料,以方便 ORACLE 数据库管理员按需查阅。我也用了一些章节来描述那些有助于故障处理和审查的操作系统工具。这会比只关注 ORACLE DBMS 软件而不关心与操作系统的交互更能理解所讨论的特性。我希望你也能认同这一点。

因为存在大量隐藏参数、未在文档中说明的事件和 X\$固定表(X\$ fixed table),我不可能对其一一介绍,否则我将没时间同读者分享见解。我的目的是阐明这些未在文档中说明的特性是怎样集成到 ORACLE DBMS 中的,最重要的是提出一种解决问题的结构化方法。因此,在汲取本书的营养后,你也将有能力揭秘 ORACLE 中更多鲜为人知的有用特性。

ORACLE 数据库服务器版本

当我2007年开始撰写本书第一版时,Oracle9i R2仍然会得到全面的技术支持,而 Oracle10g 也已经被大部分用户所采纳。2007年秋天,Oracle11g 发布了。尽管 Oracle9i R2的错误修正支持已于2007年7月终止,但它仍在被广泛地使用。我认为协调这三个版本最好的方法就是吸收 Oracle11g 的一些新特性,并在 Oracle11g 上重复大部分的测试。一般来说,大部分的新特性都会独立发布。Events、ORADEBUG、ASM、扩展 SQL 跟踪、Statspack 和 AWR 在 Oracle11g 里并没有大的变化。最新版本增加了一些等待和诊断事件,扩展 SQL 跟踪格式在 Oracle11g R1 (11.1.0.6) 里有些细微的变化,Oracle11g R1的 PS1里(11.1.0.7) 同样如此,但是仍然未在文档中说明。当然,新版本中也有不少新的特性配有相关的说明文档,如结果缓存、实时应用测试(Real Application Testing)、更多的分区策略(interval、reference、system、list-list、list-hash、list-range 和 range-range)、PIVOT/UNPIVOT 和 Secure Files 等,本书并未介绍这些特性。

第 28 章提出的 MERITS 性能优化方法,同样适用于这三个版本。我已经在 ESQLTRCPROF 扩展 SQL 跟踪分析器(已升级至 11.1.0.7,见第 27 章)中加入了 Oracle11g SQL 跟踪文件格式的支持。由于 Oracle11g 自带的 TKPROF 仍然不能满足性能分析的需要,所以,不管你所在公司使用的 ORACLE 版本是什么,我都推荐使用 ESQLTRCPROF 代替 TKPROF。

目标读者

本书针对的是已掌握 ORACLE DBMS 多年、功底扎实的高级数据库管理员,并不适合初学者。我想讲明这一点,而不在入门材料上花太多功夫。必要和适当的情况下,对未在文档中说明的特性进行论述时,我会首先对各个特性进行概述,以便引出随后的详述。然而,有了这些概述并不意味着你不需要再读与该特性相关的其他文档,如果你曾经对所讨论特性的文档内容有一定

的实践经验，那就更好不过了。当然，我绝不是想阻止 DBA 新手们阅读本书，毕竟学习本书需要在 ORACLE 数据库方面有一定的基础，只要你愿意学习扩展 Oracle 文档之类的材料而具备了一些预备知识，随时可以阅读本书。

本书结构

本书分为 12 个部分。各部分的主题（例如，初始化参数、实用工具）在 ORACLE DBMS 的文档中都提到过，这样你就能立即把握本书的总体结构。各部分的内容都相对独立，可以不必从头到尾阅读本书。你可以像使用参考手册那样，选择与你的工作相关的章节，帮助你完成当前的工作——性能优化或故障处理。当有些章节之间相互关联时，我会用交叉引用来说明。

每个独立的部分按章组织。每章先介绍某个鲜为人知的特性带来的好处，如果有相关文档，还会陈述该特性在文档中说明的程度。另外还包括与该章主题相关的 ORACLE 数据库手册（如果有的话）的引用。如果你从未接触过该章节要介绍的特性，最好在学习的同时阅读相关的文档。每章的介绍还指出了该章为何值得一读以及这些技能的适用环境。第一部分、第二部分、第三部分、第四部分以及第八部分，因为同一部分中后面的章节一般以前面章节为基础，因此有必要按照书中的顺序阅读；而第五～七部分和第九～十一部分，它们的阅读顺序则没有任何限制。

- 第一部分论述在文档中部分说明和未说明的初始化参数，其中参数 PGA_AGGREGATE_TARGET 使用得最广泛。第 1 章详述了工作区大小的内部工作流程和它所基于的隐藏参数。另外，也介绍了 AUDIT_SYSLOG_LEVEL、EVENT 和 OS_AUTHENT_PREFIX 几个未在文档中说明的参数。第 2 章则把重点放在了 TRACE_FILES_PUBLIC 和 ASM_ALLOW_ONLY_RAW_DISKS 两个隐藏参数上。
- 第二部分讲述的数据字典基表是构成数据字典视图的一部分。第 3 章主要介绍了数据字典基表，第 4 章详述了如何建立一个比内置的 V\$OBJECT_USAGE 更好的视图来查找使用过的索引。
- 第三部分的第 6 章列出了用于性能诊断的事件，第 7 章介绍了如何追踪基于成本的优化器，第 8 章详述了如何转存 Oracle 网络数据包的内容。这一部分还演示了如何在某个版本的 DBMS 中查找它所支持的那些未在文档中的事件。
- 第四部分讨论 X\$ 表，它是 GV_\$ 和 V_\$ 视图的基础。V_\$ 视图以动态性能视图的方式在文档中说明，而 X\$ 表包含一些动态性能视图中没有的信息。这一部分包含 4 章，第 9 章展示了如何查找 X\$ 表中带有描述的隐藏参数，第 10 章介绍了如何获取缓冲区高速缓存和锁存器的其他信息，第 11 章描述了如何使用 V\$SESSION_WAIT 在微秒级而不是厘秒级精度获得等待事件的时间，第 12 章阐明了自动存储管理（ASM）元数据以及数据库文件大小和 ASM 分配单元之间的映射关系。同时，该部分还强调了一种处理那些未在相关文档中说明的特性的结构化方法，例如我们可以利用该方法来生成一个依赖于 V\$ 视图和 X\$ 表的文档，反之亦然。
- 第五部分讲的基本上都是未在文档中说明的 SQL 语句。这些语句按照用途分为 4 章来陈述，第 13 章讲的是在会话或实例级别设置事件的 SQL 语句，第 14 章介绍的语句改变解析模式的标识符，第 15 章展示的是如何利用 SQL 语句来临时修改用户密码，第 16 章详述的 SQL 语句则用于增强并行处理的可扩展性。针对每条语句的使用，我们都提供了相

关的实例。

- 第六部分以 5 个包为中心，其中 3 个包未在文档中说明，另外 2 个包已在文档中说明。第 17 章关于 DBMS_BACKUP_RESTORE 程序包，阐明了在数据灾难发生后，如何用 RMAN (Recovery Manager, 恢复管理器) 备份恢复数据库。在 RMAN 的编目和包含控制文件最新副本的所有数据库文件都丢失后，你不必绝望，因为 RMAN 还包含最近几次备份的目录，我们可以利用这些备份来对数据库进行恢复。当然，本部分还介绍了另外几个 PL/SQL 程序包，包括用于性能诊断和追踪的程序包 (DBMS_SYSTEM)、与任务相关的程序包 (DBMS_IJOB)、Oracle10g 和 Oracle11g 的数据库调度器的未在文档中说明的方面 (DBMS_SCHEDULER)，以及用于数据库对象名称解析的程序包 (DBMS_UTILITY)。
- 第七部分含两章。第 22 章介绍 Perl DBI 和 DBD::Oracle，它是由 Oracle 调用接口 (Oracle Call Interface) 构成的用来访问 Oracle 数据库的 Perl 接口。每个 Oracle10g 和 Oracle11g 的 ORACLE_HOME 包括带 DBI 的 Perl 安装，这在文档中没有说明。Perl DBI 脚本比 UNIX shell 和 SQL*Plus 组合的功能更加强大，而且开发时间更短。Oracle 公司的文档并没有提到 JDBC 端对端度量带来的影响，而这正是第 23 章要介绍的内容。它汇集了跟应用程序插件、端对端追踪 (DBMS_MONITOR)、扩展 SQL 跟踪和 TRCSESS 工具相关的性能诊断和监测的所有信息。
- 第八部分以性能为中心，目的是让读者熟悉一种可靠的性能优化方法，它在很大程度上基于对扩展 SQL 跟踪文件的评估。这部分包括未在文档中说明的扩展 SQL 跟踪文件格式 (第 24 章)，以及如何从 Statspack (第 25 章) 和 AWR (第 26 章) 中获取更多的信息。第 27 章介绍了本书提供的免费扩展 SQL 跟踪分析器 ESQLTRCPROF。第 28 章讲的则是本部分的核心内容——MERITS 性能优化方法。经测试，该方法是诊断和解决性能问题的一个非常行之有效的框架。
- 第九部分讲述未在文档中说明或部分说明的 Oracle 网络参数，它们可用于配置 TNS 监听器和某些 Oracle 网络功能。第 29 章阐述了 IP=FIRST 这个设置，它在 Oracle10g 中引入，但没有在文档中说明。第 30 章介绍了如何利用监听器的完全动态有效节点检查功能，建立一个简单的防火墙来防止 Oracle 实例遭受入侵。第 31 章讨论了 ENABLE=BROKEN 参数，而第 32 章则谈到了关于默认主机名的特性。
- 第十部分介绍 TAF (Transparent Application Failover, 透明应用程序故障切换) 和数据库服务 (DBMS_SERVICE) 未在文档中说明的方面，当一个集群节点发生故障后又重新加入集群时，它们可以用于负载的重新均衡 (第 33 章)。第 34 章介绍了在具有 ASM 特性和没有 ASM 特性的情况下，从 Oracle RAC 安装向 Oracle 单一实例安装转换的快速过程，该过程主要用于硬件故障或软件缺陷导致 DBMS 无法在多实例模式运行的灾难环境 (例如，启用 RAC)。
- 第十一部分包括 OERR 实用工具 (第 35 章)、RMAN 管道接口 (第 36 章) 和 ORADEBUG (第 37 章)。其中，ORADEBUG 大概是最有用的。它可用于控制一个实例程序，启用和禁用 SQL 跟踪，检索 SQL 跟踪文件名，为性能分析生成诊断转储文件，等等。

- 第十二部分包含一个词汇表和参考书目。你可能希望首先阅读词汇表来了解本书的术语。在本书中，用方括号包含的 4 个字母组成的词和出版年份（例如[ShDe 2004]）指的是该书目的来源。这一部分还包含一个附录，列出了启用和禁用的 DBMS 选项，如 RAC 或分区。

源代码库

列表上显示的源代码都可以从 Apress 网页下载到其压缩文件。每个超过 12 行的列表都由唯一的文件名称标识。在大多数章节的结尾处，都有源代码库这一节，它列出了相应章节的所有源代码文件以及它们的功能。要下载整本书的源代码库，请访问 <http://www.apress.com/book/view/1430219521>，然后选择 Source Code 链接。

协议和条款

按我的理解，Oracle 这个名字指的是 Oracle 公司及其国际附属子公司。关于 Oracle 公司提出的 Oracle 和 ORACLE[®] 名称，本人遵守与之相关的协议。Oracle Corporation 指的是 Oracle 公司，而 ORACLE 指的是数据库服务器产品。ORACLE_HOME（UNIX 上的 \$ORACLE_HOME 或 Windows 上的 %ORACLE_HOME% 环境变量）指定了数据库服务器软件的安装目录。和大多数作者不同，我不喜欢把 Oracle 这个名称用在 Oracle 公司的任何软件产品中，而使用 ORACLE DBMS 来表示 Oracle 公司提供的数据库服务器软件。

数据库和实例的对比

Oracle Database Concepts 10g Release 2 手册的 1-8^① 页提到：“Oracle 数据库的物理数据库结构，包括数据文件、重做日志文件（redo log file）和控制文件。”可见，ORACLE 数据库由数据文件（分到不同的表空间中）、重做日志文件和控制文件组成，也就是一个磁盘上的文件集合。该手册的 1-13 页又指出：

每次启动数据库时，就会分配一个系统全局区域（SGA），并启动大量 Oracle 后台进程。这些后台进程和内存缓冲区的组合就称为一个 ORACLE 实例。

我坚决主张描述事情要清楚，使用术语要一致。但是，数据库（database）这一术语却经常使用不当。如果我们把数据库定义为数据文件、重做日志文件和控制文件的组合体，那么这个数据库显然不能启动。通常数据库指的是 ORACLE DBMS 实例（instance），它包括大量进程和 SGA 这样的内存结构，这才是真正有生命的 ORACLE 数据库。本书中，我尽可能地参照表 1 来使用数据库和实例这两个术语，也强烈建议大家参考这种使用方式，以避免混用。

① 请参阅 \$ORACLE_HOME/rdbms/msg/oraus.msg 的第 234 行。

② 本书中提到的 Oracle 数据库手册中的页码为原手册页码，例如 1-8 代表第 1 章第 8 页，后面提及的页码均相似，不再说明。——编者注

表1 实例和数据库

术 语	动 作	SQL*Plus命令
启动ORACLE DBMS实例	实例读取参数文件，启动SMON、PMON等进程，并在内存里分配SGA	STARTUP NOMOUNT
装载数据库	DBMS实例的某些进程打开数据库的控制文件，这时没有其他文件被访问	STARTUP MOUNT
打开数据库	DBMS实例打开在线重做日志和一个或多个表空间（至少打开SYSTEM这个包含数据字典的表空间）	STARTUP或STARTUP OPEN
关闭实例	DBMS实例首先关闭数据文件和在线重做日志（SQL*Plus消息显示 Database closed），然后关闭控制文件（SQL*Plus消息显示 Database dismounted），最后释放SGA并终止所有进程（SQL*Plus消息显示ORACLE实例关闭）。也可使用ALTER DATABASE CLOSE和ALTER DATABASE DISMOUNT这两条SQL语句来完成前两步操作	SHUTDOWN

实例服务名和网络服务名的对比

为了消除服务名（service name）这个术语的歧义，我对 tnsnames.ora 中定义的服务或类似LDAP的目录服务使用网络服务名（Net service name，这里的 Net 是 Oracle Net），而把实例中注册了监听器的服务用实例服务名（instance service name）来区分。Oracle10g 和 Oracle11g 的实例服务名使用 SERVICE_NAMES 参数或 DBMS_SERVICE 程序包来定义。tnsping 命令接受网络服务名，而由 lsnrctl services 命令返回的服务列表中包括实例服务名。像 ndebes/secret@ten. oradbpro.com 这样的连接串，包括网络服务名。一个网络服务名定义的主体，包括任何一个实例服务名或 ORACLE_SID (SID=oracle_sid)。一个在 tnsnames.ora 中定义的网络服务名的格式如下：

```
net_service_name =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = host_name)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = instance_service_name)
    )
  )
```

查看以上格式，注意 DESCRIPTION 内容中的关键字 SERVICE_NAME。SERVICE_NAME 用来设置一个实例服务名，在 Oracle10g 里它存在于 V\$SESSION. SERVICE_NAME 列和 SQL 跟踪文件中。对 Oracle10g 来说，所有配置的实例服务名都在 DBA_SERVICES.NETWORK_NAME 中。为什么是 NETWORK_NAME 呢？因为它们是注册了 Oracle 网络监听器的实例服务名（参数 LOCAL_LISTENER 和 REMOTE_LISTENER）。^①

如果在一个网络服务名的定义里用 SID=oracle_sid 代替 SERVICE_NAME=instance_service_name，那么使用它连接的客户端会话在 V\$SESSION.SERVICE_NAME 中会有 SYS\$USERS 这个服务名，

① 为了用监听器快速注册实例服务名，比如在重启一个监听器之后，可以使用命令 ALTER SYSTEM REGISTER。

对于那些没有指定网络服务名的本地会话同样如此。本地会话使用所谓的 bequeath 协议适配器，通过 ORACLE_SID 环境变量来设置 SID。

排版规则

表 2 总结了本书中使用的排版规则。

表2 排版规则	
规 则	含 义
斜体	斜体表示书名、占位符变量等
Monospace字体	Monospace字体表示操作系统、SQL、SQL*Plus命令或代码片段
...	省略号表示被省略的一行或多行，主要用在日志文件或代码节选中
<占位符>	<占位符>这个表达式主要用于语法描述，代表一个需要用实际值替换的占位符。用尖括号将占位符括起来，其中的字符串必须用实际值替换。举一个例子：CONNECT<username>/<password>，填入实际值后，它可能变成：CONNECT ndebes/secret
\$	UNIX shell输入命令提示符（Bourne Shell或Korn Shell）
C:>	Windows命令解释器输入命令提示符（cmd.exe）
SQL>	在SQL*Plus数据库会话中输入命令的提示符
{value1 ... valueN}	有效值的范围，例如INSTANCE_TYPE={ASM RDBMS}，各个有效值用竖线分隔开

表 3 列出了本书使用的缩略词。

表3 缩略词	
缩 略 词	含 义
ASCII	American Standard Code for Information Interchange（美国信息互换标准代码）
ASH	Active Session History（活动会话历史）
AWR	Active Workload Repository（自动工作负载信息库）
ADDM	Automatic Database Diagnostic Monitor（自动数据库诊断监视器）
DBA	Database Administrator（数据库管理员）
DBMS	Database Management System（数据库管理系统）
GCS	Global Cache Service（全局缓存服务）
GES	Global Enqueue Service（全局队列服务）
I/O	input/output from/to a device（从设备输入/输出到设备）
IP	Internet Protocol（网际协议）
LOB	Large Object（大对象，如BLOB、CLOB、NCLOB）
LUN	Logical unit number（逻辑单元号）
OCI	Oracle Call Interface（Oracle调用接口）
PGA	Program Global Area（程序全局区）
RAC	Real Application Clusters（实时应用集群）

(续)

缩 略 词	含 义
SCN	System Change Number (系统改变号)
SGA	System Global Area (系统全局区)
TCP	Transmission Control Protocol (传输控制协议)
UDP	User Datagram Protocol (用户数据报协议)
n/a	not applicable (不适用)

联系我们

本书的作者和审校者都尽了最大努力对本书内容进行了验证和测试。你在阅读时一旦发现问题，请联系出版商，从而使本书更加完善。你也可以去 Apress 网站提交本书的勘误，网址为 <http://www.apress.com/book/view/1430219521>。

目 录

第一部分 初始化参数

第 1 章 在文档中部分说明的参数	2
1.1 AUDIT_SYSLOG_LEVEL	2
1.1.1 syslog 设备	2
1.1.2 审计简介	3
1.1.3 使用 AUDIT_SYSLOG_LEVEL 参数	4
1.1.4 审计非特权用户	5
1.1.5 小结	6
1.2 PGAAggregate_TARGET	6
1.2.1 自动 PGA 内存管理介绍	7
1.2.2 对 PGAAggregate_TARGET 的 误解	8
1.2.3 研究 PGAAggregate_TARGET	9
1.2.4 使用管道表函数创建一个大表	9
1.2.5 V\$SQL_WORKAREA_ACTIVE	10
1.2.6 PGA_MAX_SIZE	14
1.2.7 SMM_MAX_SIZE	15
1.2.8 SMM_PX_MAX_SIZE	15
1.2.9 共享服务器	15
1.2.10 并行执行	16
1.2.11 小结	17
1.3 EVENT	18
1.3.1 语法	18
1.3.2 在实例级别均衡事件	19
1.3.3 案例研究	19
1.4 OS_AUTHENT_PREFIX	19
1.4.1 OPS\$数据库用户及密码 文件身份认证	20
1.4.2 案例研究	20
1.4.3 小结	23
1.5 源代码库	23

第 2 章 隐藏的初始化参数	24
2.1 跟踪文件权限与 TRACE_FILES_PUBLIC 参数	24
2.2 ASM 测试环境与 ASM_ALLOW_ONLY_RAW_DISKS 参数	26
2.2.1 ASM 隐藏参数	26
2.2.2 为 ASM 配置 Oracle 集群件	27
2.2.3 ASM 实例设置	28
2.2.4 磁盘失效模拟	31
2.3 源代码库	31

第二部分 数据字典基表

第 3 章 数据字典基表介绍	34
第 4 章 IND\$、V\$OBJECT_USAGE 和 索引监控	38
4.1 模式限制	38
4.2 索引使用监控案例研究	39
4.2.1 MONITOR_SCHEMA_INDEXES 函数	40
4.2.2 为 HR 模式启动索引监控	40
4.2.3 小结	44
4.3 源代码库	45

第三部分 事 件

第 5 章 10027 事件和死锁诊断	48
5.1 死锁	48
5.2 10027 事件	49
第 6 章 10046 事件和扩展 SQL 跟踪	52
第 7 章 10053 事件和基于成本的优化器	54
7.1 跟踪文件的内容	57

7.2 案例研究.....57	13.2 ALTER SESSION SET EVENTS.....109
7.2.1 查询块和对象标识符.....58	13.3 ALTER SYSTEM SET EVENTS110
7.2.2 考虑的查询转换.....58	13.4 ALTER SESSION/SYSTEM SET EVENTS 和诊断转储.....111
7.2.3 说明.....60	13.5 立即转储.....112
7.2.4 绑定变量窥视的结果.....61	第 14 章 ALTER SESSION SET CURRENT_SCHEMA.....113
7.2.5 优化器参数.....61	14.1 特权用户与模式用户.....113
7.2.6 系统统计信息.....66	14.2 ALTER SESSION SET CURRENT_SCHEMA 限制.....115
7.2.7 表和索引的对象统计信息.....67	14.2.1 高级队列.....116
7.2.8 单表访问路径和成本.....68	14.2.2 RENAME.....116
7.2.9 联结顺序.....70	14.2.3 私有数据库链接.....117
7.2.10 执行计划.....73	14.2.4 存储概要.....118
7.2.11 谓词信息.....74	第 15 章 ALTER USER IDENTIFIED BY VALUES.....119
7.2.12 提示和查询块名称.....74	15.1 密码游戏.....119
7.3 源代码库.....75	15.2 用 ALTER USER IDENTIFIED BY VALUES 锁定账户.....121
第 8 章 10079 事件和 Oracle 网络数据 包的内容.....76	15.3 ALTER USER 和未加密的密码.....122
第四部分 X\$固定表	第 16 章 SELECT FOR UPDATE SKIP LOCKED.....124
第 9 章 X\$固定表介绍.....80	16.1 高级队列.....124
9.1 X\$固定表与 C 语言编程.....80	16.2 Contention 和 SELECT FOR UPDATE SKIP LOCKED.....126
9.2 分层系统结构.....81	16.3 DBMS_LOCK——题外话.....133
9.3 授权访问 X\$表与 V\$视图.....82	16.4 源代码库.....136
9.4 从 V\$视图深入剖析 X\$固定表.....83	第六部分 提供的 PL/SQL 程序包
9.5 X\$表与 V\$视图之间的关系.....88	第 17 章 DBMS_BACKUP_RESTORE.....138
9.6 源代码库.....89	17.1 恢复管理器.....138
第 10 章 X\$BH 表及门锁争用.....90	17.2 TDPO 灾难恢复案例研究.....142
第 11 章 X\$KSLED 以及增强的会话 等待数据.....96	17.3 源代码库.....144
11.1 深度探讨视图 V\$SESSION_WAIT.....96	第 18 章 DBMS_IJOB.....145
11.2 改进的 V\$SESSION_WAIT 视图.....97	18.1 介绍 DBMS_JOB.....145
11.3 源代码库.....100	18.2 BROKEN 过程.....145
第 12 章 X\$KFFXP 与 ASM 元数据.....101	18.2.1 语法.....145
12.1 固定表 X\$KFFXP.....101	18.2.2 参数.....146
12.2 抢救 SPFILE 文件.....102	18.2.3 使用说明.....146
12.3 映射数据段到 ASM 存储.....104	
第五部分 SQL 语句	
第 13 章 ALTER SESSION/SYSTEM SET EVENTS.....108	
13.1 跟踪你自己的会话.....108	

18.2.4 范例	146	20.3 KSDDDT 过程	168
18.3 FULL_EXPORT 过程	146	20.3.1 语法	168
18.3.1 语法	147	20.3.2 使用说明	168
18.3.2 参数	147	20.3.3 范例	168
18.3.3 范例	147	20.4 KSDFLS 过程	169
18.4 REMOVE 过程	148	20.4.1 语法	169
18.4.1 语法	148	20.4.2 使用说明	169
18.4.2 参数	148	20.4.3 范例	169
18.4.3 范例	148	20.5 KSDIND 过程	169
18.5 RUN 过程	148	20.5.1 语法	169
18.5.1 语法	148	20.5.2 参数	169
18.5.2 参数	148	20.5.3 使用说明	169
18.5.3 使用说明	149	20.5.4 范例	170
18.5.4 范例	149	20.6 KSDWRT 过程	170
18.6 源代码库	150	20.6.1 语法	170
第 19 章 DBMS_SCHEDULER	151	20.6.2 参数	170
19.1 使用数据库调度器运行外部作业	151	20.6.3 使用说明	170
19.1.1 退出代码处理	152	20.6.4 范例	171
19.1.2 标准错误输出	153	20.7 READ_EV 过程	171
19.2 UNIX 系统上的外部作业	155	20.7.1 语法	171
19.2.1 清除环境变量	155	20.7.2 参数	172
19.2.2 命令行处理	157	20.7.3 使用说明	172
19.2.3 外部作业与非特权用户	159	20.7.4 范例	172
19.3 Windows 系统上的外部作业	161	20.8 SET_INT_PARAM_IN_SESSION 过程	172
19.3.1 命令行参数处理	161	20.8.1 语法	173
19.3.2 Windows 环境变量	162	20.8.2 参数	173
19.3.3 外部作业与非特权用户	163	20.8.3 使用说明	173
19.3.4 ORADIM 实用工具创建的 服务	163	20.8.4 示例	173
19.3.5 OracleJobScheduler 服务	163	20.9 SET_BOOL_PARAM_IN_SESSION 过程	174
19.4 源代码库	164	20.9.1 语法	174
第 20 章 DBMS_SYSTEM	165	20.9.2 参数	174
20.1 GET_ENV 过程	165	20.9.3 使用说明	174
20.1.1 语法	165	20.9.4 示例	174
20.1.2 参数	165	20.10 SET_EV 过程	175
20.1.3 使用说明	166	20.10.1 语法	175
20.1.4 范例	166	20.10.2 参数	175
20.2 KCFRMS 过程	166	20.10.3 使用说明	175
20.2.1 语法	166	20.10.4 范例	175
20.2.2 使用说明	166	20.11 SET_SQL_TRACE_IN_SESSION 转储	177
20.2.3 范例	166	20.11.1 语法	177
		20.11.2 参数	177

20.11.3 使用说明	177	第 23 章 应用程序插桩及端到端跟踪	212
20.11.4 范例	177	23.1 插桩简介	212
20.12 WAIT_FOR_EVENT 过程	177	23.2 案例研究	214
20.12.1 语法	177	23.3 程序编译	216
20.12.2 参数	178	23.4 插桩的运行	217
20.12.3 使用说明	178	23.5 TRCSESS 的使用	221
20.12.4 范例	178	23.6 插桩与程序调用栈	226
第 21 章 DBMS_UTILITY	179	23.7 源代码库	227
21.1 NAME_RESOLVE 过程	179		
21.1.1 语法	179	第八部分 性 能	
21.1.2 参数	180	第 24 章 扩展 SQL 跟踪文件格式参考	230
21.1.3 使用说明	181	24.1 扩展 SQL 跟踪文件介绍	230
21.1.4 异常	181	24.2 SQL 和 PL/SQL 语句	231
21.1.5 范例	181	24.3 递归调用深度	231
21.2 对象统计信息的名称解析和提取	183	24.4 数据库调用	232
21.3 源代码库	185	24.4.1 解析	232
第七部分 应用程序开发		24.4.2 PARSING IN CURSOR 条目的 格式	233
第 22 章 Perl DBI 与 DBD::Oracle	188	24.4.3 PARSE 条目的格式	235
22.1 常见的 Perl DBI 陷阱	188	24.4.4 PARSE ERROR 条目的格式	236
22.2 Perl 与 DBI 简史	189	24.4.5 EXEC 条目的格式	236
22.3 为 Perl 与 DBI 设置环境变量	189	24.4.6 FETCH 条目的格式	236
22.3.1 UNIX 环境	189	24.4.7 执行计划散列值	237
22.3.2 Windows 环境	194	24.4.8 计划散列值案例研究	237
22.4 在 UNIX 系统上透明地运行 Perl 程序	196	24.4.9 CLOSE 条目的格式	240
22.5 在 Windows 系统上透明地运行 Perl 程序	197	24.5 COMMIT 与 ROLLBACK	241
22.6 连接到一个 ORACLE DBMS 实例	199	24.6 UNMAP	242
22.6.1 DBI 连接语法	199	24.7 执行计划、统计信息与 STAT 条 目的格式	242
22.6.2 使用 Bequeath 适配器连接	200	24.7.1 Oracle9i 中 STAT 条目的 格式	243
22.6.3 使用 IPC 适配器连接	201	24.7.2 Oracle10g 和 Oracle11g 中 STAT 条目的格式	243
22.6.4 通过 TCP/IP 适配器连接	202	24.8 等待事件	245
22.6.5 简易连接	203	24.8.1 WAIT 条目的格式	245
22.6.6 使用 SYSDBA 或 SYSOPER 特权连接	203	24.8.2 Oracle9i 中的 WAIT	246
22.6.7 使用操作系统认证连接	204	24.8.3 Oracle10g 和 Oracle11g 中 的 WAIT	246
22.6.8 连接属性	205	24.9 绑定变量	247
22.7 完整 Perl DBI 示例程序	206	24.9.1 BINDS 条目的格式	247
22.8 异常处理	210		
22.9 源代码库	211		

24.9.2 语句调优、执行计划以及绑定变量	251	27.3.1 命令行选项	305
24.10 跟踪文件条目其他项	257	27.3.2 ESQLTRCPROF 报告部分	306
24.10.1 会话标识符	257	27.4 小结	314
24.10.2 服务名称 ID	258	27.5 源代码库	315
24.10.3 应用程序插桩	259	第 28 章 MERITS 性能优化方法	316
24.10.4 ERROR 条目的格式	261	28.1 MERITS 方法简介	316
24.10.5 应用程序插桩与并行执行进程	263	28.2 测量	317
第 25 章 Statspack	265	28.3 评估	321
25.1 Statspack 介绍	265	28.4 重现	323
25.1.1 检索捕获到的 SQL 语句文本	267	28.5 改进	323
25.1.2 访问 STATS\$SQLTEXT	270	28.6 推断	324
25.1.3 使用保留格式捕获 SQL 语句	275	28.7 安装	324
25.2 未在文档中说明的 Statspack 报告参数	277	28.8 MERITS 方法案例研究	325
25.3 Statspack 表	278	28.8.1 阶段 1——测量	325
25.4 找出 Statspack 库中代价高的语句	281	28.8.2 阶段 2——评估	325
25.5 识别使用过的索引	281	28.8.3 阶段 3——重现	332
25.6 SQL Trace 捕获语句的执行计划	282	28.8.4 阶段 4——改进	335
25.7 找出高资源利用率的快照	284	28.8.5 阶段 5——推断	339
25.7.1 高 CPU 使用	285	28.8.6 阶段 6——安装	339
25.7.2 高 DB 时间	287	28.8.7 小结	339
25.8 从另一数据库导入 Statspack 数据	290	28.9 源代码库	340
25.9 源代码库	292	第九部分 Oracle Net	
第 26 章 整合扩展 SQL 跟踪和 AWR	294	第 29 章 TNS 监听器 IP 地址绑定与 IP=FIRST	342
26.1 检索执行计划	294	29.1 IP 地址绑定介绍	342
26.2 小结	296	29.2 多宿主系统	344
26.3 源代码库	297	29.3 IP=FIRST 未启用	345
第 27 章 ESQLTRCPROF 扩展 SQL 跟踪分析器	298	29.3.1 主机名	346
27.1 分类等待事件	298	29.3.2 回环适配器	347
27.2 计算响应时间和统计信息	299	29.3.3 引导 IP 地址	348
27.2.1 案例研究	300	29.3.4 服务 IP 地址	348
27.2.2 运行 Perl 程序	301	29.4 IP=FIRST 开启	349
27.2.3 计算统计信息	303	29.5 小结	350
27.2.4 计算响应时间	303	第 30 章 TNS 监听器 TCP/IP 有效结点检验	351
27.3 ESQLTRCPROF 参考	304	30.1 有效结点检验简介	351
		30.2 在运行时打开和修改有效结点检验	353

第 31 章	本地命名参数 ENABLE=BROKEN	356
--------	----------------------	-----

第 32 章	Oracle Net 配置中默认的主机名	359
32.1	默认主机名	359
32.2	关闭默认监听器	360

第十部分 实时应用集群

第 33 章	会话断开连接、负载均衡与 TAF	364
33.1	透明应用故障转移介绍	364
33.2	改变系统断开会话设置	365
33.2.1	SELECT 故障转移	366
33.2.2	在事务末的故障转移	369
33.3	会话中断和 DBMS_SERVICE	371
33.3.1	使用 DBMS_SERVICE 创建服务	372
33.3.2	DBMS_SERVICE 和 TAF 下的会话中断	374
33.4	小结	376
33.5	源代码库	376

第 34 章	不重装就移除 RAC 选项	377
34.1	连接 ORACLE 软件	377
34.2	案例研究	378
34.2.1	模拟表决磁盘失效	379
34.2.2	使用 make 工具移除 RAC 可选项	381
34.2.3	转换 CRS 环境为本地环境	383
34.2.4	重启适用于 RAC 的 CRS 环境	385
34.3	小结	386

第十一部分 实用工具

第 35 章	OERR	388
35.1	OERR 脚本介绍	388
35.2	检索未在文档中说明的事件	390
35.3	源代码库	392
第 36 章	数据恢复管理器管道接口	393
36.1	数据恢复管理介绍	393
36.2	DBMS_PIPE 介绍	394
36.3	RMAN_PIPE_IF 包	395
36.4	RMAN_PIPE_IF 包详述	395
36.5	使用 RMAN_PIPE_IF 包	396
36.6	验证备份块	401
36.7	跨节点并行备份与恢复	402
36.8	源码库	403
第 37 章	ORADEBUG SQL*Plus 命令	404
37.1	ORADEBUG 介绍	404
37.2	ORADEBUG 使用步骤	405
37.3	ORADEBUG 命令介绍	405
37.3.1	连接到一个进程	406
37.3.2	ORADEBUG IPC	408
37.3.3	ORADEBUG SHORT_STACK	409
37.3.4	诊断转储	410
37.4	小结	414

第十二部分 附 录

附录 A	启用和禁用 DBMS 可选项	416
附录 B	参考书目	417
附录 C	术语表	419

Part 1

第一部分

初始化参数

本部分内容

- 第 1 章 在文档中部分说明的参数
- 第 2 章 隐藏的初始化参数

打个比方，Oracle 数据库管理系统就像是有无数可以转向的旋钮和开合的开关。Oracle9i R2 有 257 个归档参数，Oracle10g R2 有 258 个，Oracle11g R1 则有 294 个。大概没有哪个 DBA 能记住所有这些参数的含义和允许值。每一个发行版本的 *Oracle Database Reference* 手册都是查看已记录的初始化参数的权威来源。本章详细探讨了文档中只给出部分说明的参数 `AUDIT_SYSLOG_LEVEL`、`PGA_AGGREGATE_TARGET`、`EVENT` 和 `OS_AUTHENT_PREFIX`，并且提供了 *Oracle Database Reference* 手册中未提及的一些信息。`AUDIT_SYSLOG_LEVEL` 和 `OS_AUTHENT_PREFIX` 两个参数与数据库安全相关。`EVENT` 感觉是一个很奇怪的参数，它本身已在文档中说明，但它的允许值却没有说明。其他的参数要么是在错误发生的时候用来收集证据，要么就是通过 Oracle Support Services 收集诊断信息。从性能的角度来看，学会内部处理 `PGA_AGGREGATE_TARGET` 将会显著地帮助 DBA 减少大规模排序操作的响应时间。

1.1 AUDIT_SYSLOG_LEVEL

初始化参数 `AUDIT_SYSLOG_LEVEL` 只在文档中做了部分说明。文档中一些不准确的描述显示该参数实际上不太有用。`SYS` 或数据库管理员或操作员对数据库做出的操作是由 UNIX 操作系统 `root` 用户拥有的 `syslog` 日志守护文件（`daemon log file`）进行审计的。这样就可以防止拥有特权的数据库用户删除包含他们活动日志的审计记录。默认的设置是审计具有 `SYSDBA` 或者 `SYSOPER` 特权的 `CONNECT`、`STARTUP` 和 `SHUTDOWN` 操作，记录到 ORACLE 软件所有者的文件中，而对 `SQL`、`PL/SQL` 语句以及其他操作无论具有何种特权（比如 `DBA`）都不做审计。换句话说，除了上面提到的操作，标准审计（参考 `AUDIT_TRAIL` 参数）和细粒度的审计（参考 `DBMS_FGA` 包）默认都是关闭的。因此，对特权用户执行的很多活动将不会进行跟踪。对于 ORACLE 软件所有者（`AUDIT_TRAIL=OS`）拥有的操作系统文件或者数据库表 `SYS.AUD$`（`AUDIT_TRAIL=DB`）的审计可能会回避，因为 `DBA` 通常可以访问 ORACLE 软件所有者的 UNIX 账户以及数据库表 `SYS.AUD$`，并且可以很容易地删除由他们的操作所生成的审计记录。通过 UNIX 系统的 `syslog` 设备进行审计也可以有效地检测出黑客的侵入和内部人员的恶意操作。

1.1.1 syslog 设备

Oracle10g 一个新特性就是可以在 UNIX 系统中使用 `syslog` 设备写入审计跟踪。该设备包括

一个名为 syslogd (参考 man syslogd) 的守护进程, 它接受来自使用 syslog (参考 man syslog) C 语言库函数的应用程序的日志信息。syslogd 的配置文件通常是位于 /etc/syslog.conf, 日志信息一般是位于 /var/log 或者 /var/adm, 这个取决于 UNIX 的系统变量。日志文件的名称由一个包含设备名称和优先级别的字符串决定, 在设置 AUDIT_SYSLOG_LEVEL 时大多数都会用到。在 /etc/syslog.conf 中的每一个条目都会使用相应的设备和优先级的组合来给日志文件命名。把条目 user.notice /var/log/oracle_dbms 写入 syslog.conf 文件, 然后给 syslogd 发送一个带 kill 命令^①的挂起信号使之重新读取配置文件, 随后在一个 ORACLE 实例中任何被设置了 AUDIT_SYSLOG_LEVEL=user.notice 的日志条目将会记录在 /var/log/oracle_dbms 文件中。

1.1.2 审计简介

在 UNIX 系统上, 使用 SYSDBA 或者 SYSOPER 特权对 ORACLE 实例进行 CONNECT、STARTUP 和 SHUTDOWN 操作, 都将会被无条件地审计, 记录到 \$ORACLE_HOME/rdbms/audit 里以 .aud 作为扩展名的文件或者明确指定参数 AUDIT_FILE_DEST^② 的目录中。Oracle9i 是第一个可以通过设置 AUDIT_SYS_OPERATIONS=TRUE 来审计除了使用 SYSDBA 或者 SYSOPER 特权的 CONNECT、STARTUP 和 SHUTDOWN 之外操作的发行版本。

当 AUDIT_SYSLOG_LEVEL 和 AUDIT_SYS_OPERATIONS 结合时, SYS 用户执行任何的 SQL 和 PL/SQL 都将会使用 syslog 设备对其进行审计。因为 syslog 使用的文件是属于 root 用户的, 并且 DBA 通常不能访问 root 账户, 所以 DBA 并不能删除他们活动的跟踪记录。当然, 这对那些已经成功侵入电脑并获得 ORACLE 软件所有者账户而非 root 账户的非法侵入者同样有效。对于那些破解了特权数据库用户密码并且可以通过 Oracle Net 连接的黑客也是同样有效的。

在 Windows 上并没有实现参数 AUDIT_SYSLOG_LEVEL 和 AUDIT_FILE_DEST, 因为 Windows 事件日志充当了操作系统的审计跟踪 (见图 1-1)。与 UNIX 一样, CONNECT、STARTUP 和 SHUTDOWN 将会被无条件地记录。当设置 AUDIT_SYS_OPERATIONS=TRUE 时, 具有 SYSDBA 或者 SYSOPER 权限的操作也会被写入 Windows 事件日志, 可以通过依次选择“开始”>“控制面板”>“管理工具”>“事件查看器”进行查看。日志类别是应用程序, 来源以 Oracle.ORACLE_SID 命名。对于这一类 DBMS 实例的事件通过选择“查看”>“过滤”进行筛选。

Oracle Database Reference 10g Release 2 手册对 AUDIT_SYSLOG_LEVEL 做出了如下解释 (1-22 页):

如果 AUDIT_TRAIL 参数被设置为 os, AUDIT_SYSLOG_LEVEL 使得操作系统的审计日志可以通过 syslog 工具写入系统中。设备的值可以是以下中的任意一个: USER、LOCAL0~LOCAL7、SYSLOG、DAEMON、KERN、MAIL、AUTH、LPR、NEWS、UUCP 或 CRON。优先级的值可以是以下中的任意一个: NOTICE、INFO、DEBUG、WARNING、ERR、CRIT、ALERT 或 EMERG。

① 在 Red Hat Linux 上使用命令 kill -HUP 'cat/var/run/syslogd.pid'。

② AUDIT_FILE_DEST 在实例启动的时候就会加载。当使用 SYSDBA 或者 SYSOPER 连接时如果实例宕掉了, 就会使用默认的审计文件目标 \$ORACLE_HOME/rdbms/audit。

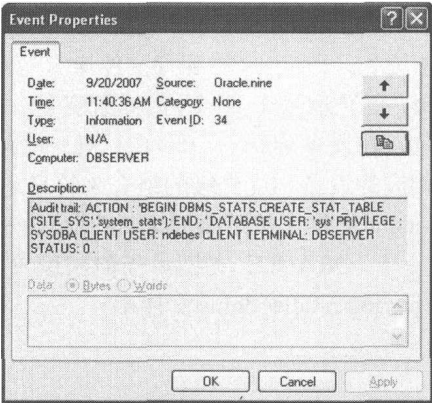


图 1-1 Windows 事件查看器的事件详细信息

在 Solaris 10 和 Red Hat Linux 系统上对新的特性进行测试表明，文档中有如下 3 点不准确的描述。

(1) AUDIT_SYSLOG_LEVEL 与 AUDIT_TRAIL 是无关的。当设置了 AUDIT_SYSLOG_LEVEL 并且 AUDIT_TRAIL 的默认值为 NONE 时，CONNECT、STARTUP 和 SHUTDOWN 操作则通过 syslog 记录日志。

(2) 设置参数 AUDIT_SYSLOG_LEVEL 以及 AUDIT_SYS_OPERATIONS=TRUE 会导致使用 SYSDBA 或者 SYSOPER 特权执行的任何操作（比如 SQL 和 PL/SQL 语句）都会通过 syslog 记录日志，即使是 AUDIT_TRAIL=NONE。

(3) 只有设备和优先级的某些组合会被接受。不被接受的组合会出现 ORA-32028: Syslog facility or level not recognized 错误，并且会阻止 DBMS 实例启动。

如果文档中的描述是准确的，当审计其他用户的操作到数据字典基表 SYS.AUD\$时，审计 SYSDBA 或者 SYSOPER 特权执行的操作到系统日志是不可能的。但是，这种限制并不存在。

1.1.3 使用 AUDIT_SYSLOG_LEVEL 参数

如前所述，分配给 AUDIT_SYSLOG_LEVEL 的字符串必须由设备名称和优先级组成。出人意料的是，当执行 SHOW PARAMETER 命令或对 V\$PARAMETER 执行 SELECT 时，仅仅设备是可见的，点号以及优先级被隐藏了^①。例如，使用 SPFILE 这个带有 *.audit_syslog_level='USER.NOTICE' 条目的文件启动实例，执行 SHOW PARAMETER 就会得到：

```
SQL> SHOW PARAMETER audit_syslog_level
NAME                                TYPE                                VALUE
-----
audit_syslog_level                  string                              USER
SQL> SELECT value FROM v$parameter WHERE name='audit_syslog_level';
VALUE
-----
USER
```

① 基于 ORACLE DBMS 10.2.0.3 版本进行测试。

然而，当执行 CONNECT/AS SYSDBA 时，在 Solaris 上记录到 /var/adm/messages 的设备和优先级是 “user.notice”：

```
Feb 21 11:45:52 dbserver Oracle Audit[27742]: [ID 441842 user.notice]
ACTION : 'CONNECT'
Feb 21 11:45:52 dbserver DATABASE USER: '/'
Feb 21 11:45:52 dbserver PRIVILEGE : SYSDBA
Feb 21 11:45:52 dbserver CLIENT USER: oracle
Feb 21 11:45:52 dbserver CLIENT TERMINAL: pts/3
Feb 21 11:45:52 dbserver STATUS: 0
```

如果使用了 SPFILE，则可以通过查询 V\$SPPARAMETER 得到完整的设置：

```
SQL> SELECT value FROM v$spparameter WHERE name='audit_syslog_level';
VALUE
-----
user.notice
```

1.1.4 审计非特权用户

当然，除了设置 AUDIT_SYSLOG_LEVEL 之外，你还可以设置 AUDIT_TRAIL=OS 来直接审计有关非特权用户的记录到系统日志中去。非特权用户不能删除记录着他们操作的审计跟踪。对审计视图（比如 DBA_AUDIT_STATEMENT 或 DBA_AUDIT_OBJECT）使用查询来寻找作恶者比查找系统日志容易。基于这些原因，优先考虑通过设置 AUDIT_TRAIL=DB 在数据库中保留非特权用户的审计跟踪。通过后一种设置，审计跟踪会写入数据表 SYS.AUD\$，然后可以通过之前提到的数据字典视图进行查询。设置 AUDIT_TRAIL=NONE 可以关闭对非特权用户操作的审计。

启用对非特权用户建立的数据库连接的审计，比如，

```
SQL> AUDIT CONNECT BY appuser /* audit_trail=os set */;
```

之后，和下面相似的条目就会被写入到 syslog 设备（Solaris 下执行）。

```
Feb 21 11:41:14 dbserver Oracle Audit[27684]: [ID 930208 user.notice]
SESSIONID: "15" ENTRYID: "1" STATEMENT: "1" USERID: "APPUSER"
USERHOST: "dbserver" TERMINAL: "pts/3" ACTION: "100" RETURNCODE: "0"
COMMENT$TEXT: "Authenticated by: DATABASE" OS$USERID: "oracle"
PRIV$USED: 5
```

当一个数据库会话结束时，另一个条目将会添加到 /var/adm/messages 中：

```
Feb 21 11:44:41 dbserver Oracle Audit[27684]: [ID 162490 user.notice]
SESSIONID: "15" ENTRYID: "1" ACTION: "101" RETURNCODE: "0"
LOGOFF$PREAD: "1" LOGOFF$LREAD: "17" LOGOFF$LWRITE: "0" LOGOFF$DEAD:
"0" SESSIONCPU: "2"
```

请注意，动作 LOGON (100) 和 LOGOFF (101) 提供的额外数据同视图 DBA_AUDIT_SESSION 的列相符。将操作的编号对应到操作名称是通过视图 AUDIT_ACTIONS 完成的，如下所示：

```
SQL> SELECT action, name FROM audit_actions WHERE action IN (100,101)
ACTION NAME
-----
```

```
100 LOGON
101 LOGOFF
```

当设置参数 `AUDIT_SYSLOG_LEVEL=AUTH.INFO`、`AUDIT_SYS_OPERATIONS=FALSE` 以及 `AUDIT_TRAIL=NONE` 时，`CONNECT`、`STARTUP` 和 `SHUTDOWN` 将通过 `syslog` 记录日志。基于这些设置，在 Solaris 上关闭一个实例时会将如下类似的条目写入到 `/var/adm/messages` 中：

```
Feb 21 14:40:01 dbserver Oracle Audit[29036]:[ID 63719 auth.info] ACTION:'SHUTDOWN'
Feb 21 14:40:01 dbserver DATABASE USER: '/'
Feb 21 14:40:01 dbserver PRIVILEGE : SYSDBA
Feb 21 14:40:01 dbserver CLIENT USER: oracle
Feb 21 14:40:01 dbserver CLIENT TERMINAL: pts/3
Feb 21 14:40:01 dbserver STATUS: 0
```

当设置参数 `AUDIT_SYSLOG_LEVEL=AUTH.INFO`、`AUDIT_SYS_OPERATIONS=TRUE` 以及 `AUDIT_TRAIL=NONE` 时，`SYSDBA` 或者 `SYSOPER` 特权下执行的 SQL 以及 PL/SQL 语句同样通过 `syslog` 记录日志。通过 `/AS SYSDBA` 连接到数据库之后，删除一个用户将会在 `syslog` 中产生同以下类似的条目：

```
Feb 21 14:46:53 dbserver Oracle Audit[29170]: [ID 853627 auth.info]
ACTION : 'drop user appuser'
Feb 21 14:46:53 dbserver DATABASE USER: '/'
Feb 21 14:46:53 dbserver PRIVILEGE : SYSDBA
Feb 21 14:46:53 dbserver CLIENT USER: oracle
Feb 21 14:46:53 dbserver CLIENT TERMINAL: pts/3
Feb 21 14:46:53 dbserver STATUS: 0
```

1.1.5 小结

尽管设置了 `AUDIT_TRAIL=NONE`，但使用 `SYSDBA` 或者 `SYSOPER` 特权进行的 `CONNECT`、`STARTUP` 以及 `SHUTDOWN` 操作都会默认记录到 `*.aud` 文件中。如果设置了 `AUDIT_SYSLOG_LEVEL`，`SQL*Plus` 的 `STARTUP` 命令会被记录到 `$ORACLE_HOME/rdbms/audit` 中的一个 `*.aud` 文件中，而 `ALTER DATABASE MOUNT` 和随后的命令以及 `SHUTDOWN` 则通过 `syslog` 记录日志，因为使用 `syslog` 设备需要一个正在运行的实例，但是当 `STARTUP` 命令发出时实例还没有运行起来。

设置 `AUDIT_SYSLOG_LEVEL` 以及 `AUDIT_SYS_OPERATIONS=TRUE` 会产生额外的审计跟踪记录，其涵盖了已配置 `syslog` 日志文件中 `SYSDBA` 或者 `SYSOPER` 特权下执行的所有活动，并且不考虑 `AUDIT_TRAIL` 的设置。没能成功破解 UNIX root 用户账户的入侵者将无法删除这些审计跟踪记录。

当然，知道这些特性的入侵者可能会删除 `AUDIT_SYSLOG_LEVEL` 设置，但是如果使用了 `SPFILE`，至少参数的变化会被记录下来，这些变化不会立即生效，因为它是一个静态参数。在维护操作的过程中，比如升级（这必须以用户 `SYS` 运行），你可以寄希望于设置 `AUDIT_SYS_OPERATIONS=FALSE` 以避免产生很多的 `syslog` 日志文件。

1.2 PGA_AGGREGATE_TARGET

在 *Oracle9i Database Performance Tuning Guide and Reference Release 2* 以及 *Oracle Database*

Performance Tuning Guide 10g Release 2 中对初始化参数 PGA_AGGREGATE_TARGET 都进行了说明。上述的 Oracle9i 手册中说明, 用来调整手动 PGA 内存管理的参数 SORT_AREA_SIZE 和 HASH_AREA_SIZE 不应该使用, 除非是在共享服务器 (Shared Server) 的环境下, 因为 Oracle9i 共享服务器不能均衡自动 PGA 内存管理 (见 1-57 页以及 14-50 页)。负责为串行和并行执行任务管理单独工作区分配的算法在文档中并没有说明。

知道了文档中没有说明的限制给工作区分配带来的麻烦, 使得 DBA 可以给 PGA_AGGREGATE_TARGET 设置最适当的值, 从而避免泄漏工作区到磁盘或者允许操作完全在内存中运行这样昂贵的操作, 以实现性能的显著提升。在极少数情况下, 可能需要覆盖受 PGA_AGGREGATE_TARGET 影响的隐藏参数的自动设置。

1.2.1 自动 PGA 内存管理介绍

PGA (Program Global Area, 程序全局区) 是一个私有内存区域, 服务器进程分配各种操作的内存比如 sort、hash join 以及 bitmap merge。因此, PGA 内存区域是有别于 SGA (System Global Area, 系统全局区) 的。甚至还有第三个内存区域, UGA (User Global Area, 用户全局区), 它持有会话以及游标的状态信息。专用服务器进程在 PGA 中分配 UGA 内存, 而共享服务器进程则把 UGA 放置于 SGA 中, 因为它必须得被所有共享服务器访问到。如果 SGA 包含一个大池 (参数 LARGE_POOL_SIZE), 共享服务器进程就会把 UGA 放置到大池中。在 Oracle10g 中, 具有标准数据块大小^①的共享池 (shared pool)、大池 (large pool)、java 池 (java pool)、流池 (stream pool) 以及默认的缓冲池会被 ASMM (Automatic Shared Memory Management, 自动共享内存管理, 参数 SGA_TARGET) 自动地、动态地分配。

在 Oracle9i 之前的版本中, 必须使用几个 *_AREA_SIZE 参数去适应各种 PGA 内存区域的大小。这些参数的典型例子有 SORT_AREA_SIZE 和 HASH_AREA_SIZE。在 UNIX 系统上, ORACLE DBMS 是作为一个多进程架构来实现的, 进行了内存密集型操作之后, PGA 内存不能总是返回到操作系统。它徘徊在服务器进程的虚拟地址空间中, 并可能造成分页。因此像这样已分配了的内存也不能提供给其他服务器进程。同样 PGA 内存区域也没有实例范围的限制。因为每个服务器进程都可以为每个操作分配内存, 甚至可以达到 *_AREA_SIZE 参数设置的范围, 因此当存在数百个服务器进程时, 实例范围内存消耗会变得非常大。还要注意的是 *_AREA_SIZE 参数限制每个操作的范围, 而不是每一个会话的范围。因为一个查询可能同时打开多个游标, 并且每一个游标可能执行包含 ORDER BY 或者 hash join 等代价很高的 SELECT 语句, 这些在旧方法, 也就是现在所谓的手动 PGA 内存管理中是没有总的内存消耗限制的。

为了解决这些缺点, 在 Oracle9i 中引入了自动 PGA 内存管理。在 UNIX 上, 它是基于内存映射这一现代技术的——启用一个进程分配虚拟内存然后将其映射到自身的虚拟地址空间中。一

① Oracle10g 支持多达七个缓冲池: 其中五个缓冲池的数据块大小从 2 KB 到 32 KB 不等, 额外的两个缓冲池为标准数据块大小, 例如, 用 DB_BLOCK_SIZE 参数设置的数据块大小。后两个缓冲池是保存和回收池。ALTER TABLE 或者 ALTER INDEX 语句的数据段可能会被放入保存池、也可能被放入回收池, 这个视情况而定。

且不再需要某一内存，该内存可以被归还给操作系统，通过删除其在虚拟地址空间的映射。在 Solaris 上，UNIX 系统调用为 `mmap` 和 `munmap`。ORACLE 内核对内存映射例程的调用可以使用 `truss` (Solaris) 或者 `strace` (Linux) 进行跟踪^①。另外一个有趣的实用工具是 `pmap` (兼容 Solaris、Linux)，它显示一个进程的地址空间信息，包括使用 `mmap` 的匿名内存映射。早在 32 位计算时代，该工具提供了迁移 SGA 基址所需的准确信息，以便把一个更大的共享内存数据段映射到有限的 32 位应用程序虚拟地址空间中（见 Metalink note 1028623.6）。当一个进程在排序时使用 `pmap`，会揭示出已经映射了多少个匿名内存的区域以及它们总共的大小是多少。

下面是一个例子（29606 是在 `V$PROCESS.SPID` 中找到的服务器进程的 UNIX 进程 ID）。相关列是 “Anon”（匿名映射内存）。

```
$ pmap -x 29606 | grep Kb
      Address      Kbytes      RSS      Anon      Locked Mode      Mapped File
total Kb      934080      888976      63008      806912
```

借助自动 PGA 内存管理，所谓的工作区就被使用在如 `sort` 或者 `hash join` 这样的操作中。所有活跃工作区的目标累积大小通过参数 `PGA_AGGREGATE_TARGET` (PAT) 指定。单个进程可能同时访问多个工作区。自动 PGA 内存管理以及工作区的信息可以通过查询动态性能视图比如 `V$PGASTAT`、`V$PROCESS`、`V$SQL_WORKAREA` 以及 `V$SQL_WORKAREA_ACTIVE` 获得。

1.2.2 对 PGA_AGGREGATE_TARGET 的误解

`PGA_AGGREGATE_TARGET` 参数的名称是经过精心挑选的名称。我想说的是，就如它所听起来的一样，它是一个目标值，而不是一个绝对的限制，仅仅是一个目标值而已。这意味着，在高负荷下实际消耗的内存量是连续的或者至少间歇性地比目标值高。但是，自动 PGA 内存管理的实施是如此的漂亮，以至于进程会在所有可能的情况下释放内存，使得总的内存消耗快速的降到目标值之下。尤其是在 PL/SQL 中会分配大量的内存，例如执行集合，如索引表，则会永久超出目标值。而 `sort` 的内存需求可以通过使用临时数据段降下来，但 PL/SQL 对内存的需求则不行。

当使用 DBCA (Database Configuration Assistant, 数据库建置辅助精灵) 创建一个数据库时，就会有一个内存配置页面用来自定义之前提到的大部分的池，同样也包括 PGA。在这个页面上，DBCA 添加了所有池的大小以及 PGA 的大小，然后在 Total Memory for Oracle 中报告结果数据（见图 1-2）。这使得一些人认为，该内存总数（在截图中为 4956 MB）在 ORACLE 实例启动时会被全部分配。在知道 SGA 是在实例启动的时候分配的前提下，他们就认为 PGA 也同样如此。然而，实际并非如此。PGA 内存是按需分配的。即使是参数 `*_AREA_SIZE` 也不会导致其分配一个指定大小的内存。这些也是按需分配的。

因为文档对工作区分配的细节方面没有详细深入，许多数据库管理员就假定只要单个会话不与其他会话争用内存，如果不考虑 `PGA_AGGREGATE_TARGET`，全部内存对单个会话都是可见的。如果你十分好奇它的内部实现，请继续深入阅读。

① 标题为 *Rosetta Stone for UNIX*，<http://bhami.com/rosetta.html> 网页列出了 UNIX 系统中的系统调用跟踪工具。

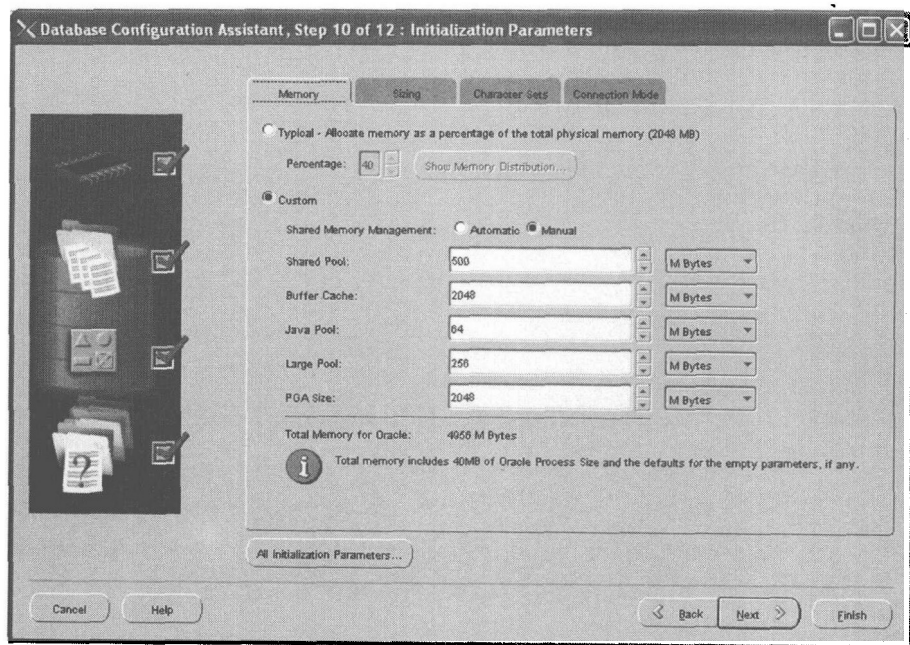


图 1-2 Database Configuration Assistant 中 PGA 分配

1.2.3 研究 PGA_AGGREGATE_TARGET

本节所介绍的研究工作是基于 Oracle10g R2 版本完成的。结果显示 Oracle9i 和 Oracle10g 所使用的算法是不同的。因为空间限制，与 Oracle9i 相关的例子以及证据^①没有被包含进来。

1.2.4 使用管道表函数创建一个大表

首先，我们需要一个表，大到足以导致磁盘在排序操作过程中溢出。接下来的几个段落展示了如何编写一个管道表函数，以返回任意数量的行数据（见源代码库中的文件 row_factory.sql）。这个函数可以被用来与包 DBMS_RANDOM 结合，以创建任意大小的随机数据表。由于管道表函数返回一个集合类型，我们首先创建一个对象类型来接收行编号。

```
SQL> CREATE OR REPLACE TYPE row_nr_type AS OBJECT (row_nr number);
/
```

管道表函数将会返回一个由 row_nr_types 组成的集合类型。

```
SQL> CREATE OR REPLACE TYPE row_nr_type_tab AS TABLE OF row_nr_type;
/
```

函数 row_factory 返回任意数量的行数据——当然，是在数据类型 ORACLE NUMBER 的限制内。它包含有两个参数 first_nr 和 last_nr，通过它们控制返回多少行数据。

^① 对于 Oracle9i 的研究，请参阅 Jonathan Lewis 的文章 <http://www.jlcomp.demon.co.uk/untested.html>。

```

CREATE OR REPLACE FUNCTION row_factory(first_nr number, last_nr number)
RETURN row_nr_type_tab PIPELINED
AS
    row_nr row_nr_type:=NEW row_nr_type(0);
BEGIN
    FOR i IN first_nr .. last_nr LOOP
        row_nr.row_nr:=i;
        PIPE ROW(row_nr);
    END LOOP;
    return;
END;
/

```

当 last_nr 比 first_nr 大, row_factory 将返回 last_nr - first_nr + 1 行数据。其返回的结果与 SELECT ROWNUM FROM table 的结果非常相似, 除了参数值以及不能控制从一个表中返回多少行数据之外。下面是一个例子:

```

SQL> SELECT * FROM TABLE(row_factory(1,2));
   ROW_NR
-----
        1
        2

```

对于生成一张大表的经典方法包括从一张真实表中筛选数据, 可能使用笛卡尔连接得到数目非常大的行数据。除了需要对 CREATE TABLE 语句较少的编码外, 这种新方法使用了管道表函数, 它具有在一个数据段上不造成任何一致性或者物理读取的额外好处。通过调用 row_factory 并且设置参数 first_nr 为 1 和 last_nr 为 1 000 000, 我们现在可以创建一个百万数据行的表了。

```

SQL> CREATE TABLE random_strings AS
SELECT dbms_random.string('a', 128) AS random_string
FROM TABLE(row_factory(1,1000000))
NOLOGGING;

```

第一个参数 (opt) 告诉 DBMS_RANDOM 随机生成以不同的大小写混合字母组成的字符串, 第二个参数 (len) 控制随机字符串的长度。请注意, Oracle11g 之前的发行版本, 不能直接从 SQL 中向 PL/SQL 函数传递参数^①。

在测试数据库中我设置 db_block_size=8192, 之前的 CTAS (create table as select) 将会导致一个数据段的大小大约为 150 MB。DBMS_RANDOM 也能够生成由小写、大写或者大小写混合的字母与数字或者纯数字组成的随机字符串^②。

1.2.5 V\$SQL_WORKAREA_ACTIVE

在会话级别监测 PGA 内存管理的一个好方法就是查询动态性能视图 V\$SQL_WORKAREA_

① 在 Oracle11g 中, SELECT * FROM TABLE(row_factory(first_nr => 1, last_nr => 3))在语法上是正确的。在之前的版本中该语句将会导致 ORA-00907 错误。

② 文档 A Security Checklist for Oracle9i 在 package 清单中列出的 DBMS_RANDOM 可能会被滥用, 并且推荐取消 PUBLIC 对 DBMS_RANDOM 的执行权限 (参见 http://www.oracle.com/technology/deploy/security/oracle9i/pdf/9i_checklist.pdf)。

ACTIVE, 其包含如下的列:

```
SQL> DESC v$sql_workarea_active
Name                                         Null?    Type
-----
WORKAREA_ADDRESS                           RAW(4)
OPERATION_TYPE                             VARCHAR2(20)
OPERATION_ID                               NUMBER
POLICY                                     VARCHAR2(6)
SID                                         NUMBER
QCINST_ID                                 NUMBER
QCSID                                     NUMBER
ACTIVE_TIME                               NUMBER
WORK_AREA_SIZE                             NUMBER
EXPECTED_SIZE                             NUMBER
ACTUAL_MEM_USED                           NUMBER
MAX_MEM_USED                              NUMBER
NUMBER_PASSES                             NUMBER
TEMPSEG_SIZE                              NUMBER
TABLESPACE                                VARCHAR2(31)
SEGRFNO#                                  NUMBER
SEGBLK#                                  NUMBER
```

我写了一个小的 Perl DBI 程序使之密切监测 PGA 工作区的使用情况。该 Perl 程序在 V\$SQL_WORKAREA_ACTIVE 上每秒执行一个 SELECT 并将结果打印到屏幕上。除了会话标识符 (对应于 V\$SESSION.SID)、当前和最大的工作区大小、临时数据段的大小之外, 该查询同样也检索时间戳。所有的大小都是以 MB 作为单位的。Perl 程序使用的 SELECT 语句如下:

```
SELECT sid, to_char(sysdate,'mi:ss') time,
round(work_area_size/1048576, 1) work_area_size_mb,
round(max_mem_used/1048576, 1) max_mem_used_mb, number_passes, nvl(tempseg_size/
1048576, 0) tempseg_size_mb
FROM v$sql_workarea_active
ORDER BY sid;
```

现在我们有了一个大表和监测工具, 因此我们就可以运行一些实际的测试了。因为我是唯一使用该实例的测试人员, 我可以假设不管是否设置了 PGA_AGGREGATE_TARGET, 全部内存对我是可见的。如前所述, 表的数据段大小约为 150 MB, 这样设置 PGA_AGGREGATE_TARGET 为 256 MB 对于一个在内存中的排序是绰绰有余的。因此, 这将是我们将使用的值:

```
SQL> ALTER SYSTEM SET pga_aggregate_target=256m;
System altered.
```

要启动监测, 我们需要设置 ORACLE_SID 以及 DBI 环境变量 (在第 22 章进一步讨论), 然后运行 sql_workarea_active.pl。下面是来自 Windows 的例子。在 UNIX 上, 使用 export 设置环境变量。

```
C:> set ORACLE_SID=ORCL
C:> set DBI_USER=ndebes
C:> set DBI_PASS=secret
C:> set DBI_DSN=DBI:Oracle:
```

```
C:> sql_workarea_active.pl
SID TIME WORK_AREA_SIZE MAX_MEM_USED PASSES TEMPSEG_SIZE
```

该 Perl 程序直到分配了一个或多个工作区之后才会显示数据。我将使用脚本 `sort_random_strings.sql` 在 SQL*Plus 中运行 `SELECT...ORDER BY`。以下是脚本的内容：

```
set timing on
set autotrace traceonly statistics
SELECT * FROM random_strings ORDER BY 1;
exit
```

SQL*Plus 中带有选项 `TRACEONLY` 和 `STATISTICS` 的命令 `SET AUTOTRACE` 在该上下文中是非常有用的，因为它执行语句时不会打印结果集到屏幕。此外，它收集并显示来自 `V$SESSTAT` 的执行统计结果。在一个运行脚本 `sql_workarea_active.pl` 的单独的窗口中，在 SQL*Plus 中执行脚本 `sort_random_strings.sql` 如下所示：

```
C:> sqlplus ndeb/secret @sort_random_strings.sql
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
1000000 rows selected.
Elapsed: 00:00:18.73
Statistics
-----
      133 recursive calls
       7 db block gets
    18879 consistent gets
   16952 physical reads
       0 redo size
138667083 bytes sent via SQL*Net to client
  733707 bytes received via SQL*Net from client
   66668 SQL*Net roundtrips to/from client
       0 sorts (memory)
       1 sorts (disk)
  1000000 rows processed
```

出人意料的是，出现了可用内存不足并且排序溢出到磁盘的情况。以下是脚本 `sql_workarea_active.pl` 的输出，显示该会话执行了一次 `one-pass` 排序，因为它只获得了 51.2 MB 的工作区大小：

```
SID TIME WORK_AREA_SIZE MAX_MEM_USED PASSES TEMPSEG_SIZE
148 31:38          51.2          51.2      0          16
148 31:39          51.2          51.2      0          48
148 31:40          51.2          51.2      1          73
148 31:41          21.4          51.2      1         100
148 31:42          25.8          51.2      1         130
...
148 31:56           2.9          51.2      1         133
```

通过时间戳可以确认该语句是在 18 秒内完成的。临时数据段增长到 133 MB，比该表的数据段大小稍微小一点。显然，不管是否设置了 `PAT`，整个内存对单个会话是不可见的。相应地，未在文档中说明的额外限制应该补充进去。在互联网上搜索 `pga_aggregate_target tuning undocumented`，你就会很快意识到有一些隐藏参数在影响自动 PGA 内存管理。这些隐藏参数分

别是PGA_MAX_SIZE、SMM_MAX_SIZE和SMM_PX_MAX_SIZE。其中，PGA_MAX_SIZE是以字节为单位的，其余两个是以千字节为单位的。对这些参数的说明及其目前的值可以通过查询X\$固定表X\$KSPPPI以及X\$KSPPCV获得（见第4部分）。脚本auto_pga_parameters.sql如下所示，该脚本查询这些X\$固定表然后标准化4个相关参数为KB单位：

```
SELECT x.kspinm name,
CASE WHEN x.kspinm like '%pga%' THEN to_number(y.kspstvl)/1024
ELSE to_number(y.kspstvl)
END AS value,
x.kspdesc description
FROM x$ksppi x, x$ksppcv y
WHERE x.inst_id = userenv('Instance')
AND y.inst_id = userenv('Instance')
AND x.indx = y.indx
AND x.kspinm IN ('pga_aggregate_target', '_pga_max_size',
'_smm_max_size', '_smm_px_max_size');
```

根据当前设置，脚本执行结果如下：

```
C:> sqlplus -s / as sysdba @auto_pga_parameters
NAME                                Value (KB) DESCRIPTION
-----
pga_aggregate_target                262144 Target size for the aggregate PGA memory
                                     consumed by the instance
_pga_max_size                        204800 Maximum size of the PGA memory for one
                                     process
_smm_max_size                        52428 maximum work area size in auto mode (serial)
_smm_px_max_size                     131072 maximum work area size in auto mode (global)
```

我编写了脚本pga_aggregate_target_iterator.sql，在PGAAggregateTarget的最小值10MB和最大值32GB之间改变并且为每个设置调用auto_pga_parameters.sql，来探讨如何改变PAT的值影响这些隐藏参数。关闭并且重新启动DBMS是没有必要的，因为PGA_MAX_SIZE、SMM_MAX_SIZE和SMM_PX_MAX_SIZE这三个参数会在一定限制范围内动态地进行重新计算。结果见表1-1，超过8GB的结果被省略，因为超过该值后没有任何变化。

表1-1 PGAAggregateTarget（PAT）及依赖的隐藏参数

PAT	_pga_max_size（占PAT百分比）	_smm_max size（占_pga_max_size百分比）	_smm_max_size（占PAT百分比）	_smm_px_max_size（占PAT百分比）
10 MB	200 MB (2000%)	2 MB (1%)	20%	5 MB (50%)
32 MB	200 MB (625%)	6.4 MB (3.2%)	20%	16 MB (50%)
64 MB	200 MB (320%)	12.8 MB (6.4%)	20%	32 MB (50%)
128 MB	200 MB (156%)	25 MB (12%)	20%	64 MB (50%)
256 MB	200 MB (78%)	51 MB (25%)	20%	128 MB (50%)
512 MB	200 MB (39%)	100 MB (50%)	19.5%	256 MB (50%)
1 GB	204 MB (20%)	102 MB (50%)	10%	512 MB (50%)
2 GB	410 MB (20%)	205 MB (50%)	10%	1 GB (50%)

(续)

PAT	_pga_max_size (占PAT 百分比)	_smm_max_size (占 _pga_max_size百分比)	_smm_max_size (占 PAT百分比)	_smm_px_max_size (占 PAT百分比)
3 GB	416 MB (13.5%)	208 MB (50%)	6.8%	1536 MB (50%)
4 GB	480 MB (11.7%)	240 MB (50%)	5.8%	2 GB (50%)
8 GB	480 MB (5.8%)	240 MB (50%)	2.9%	4 GB (50%)

在表 1-1 中，出现了几种模式，它们将会在接下来的几节中进行处理。请注意，当通过写入 `_SMM_MAX_SIZE` 或者 `_SMM_PX_MAX_SIZE` 到参数文件覆盖设置时，这些将不再是动态调整了，因为 `PGA_AGGREGATE_TARGET` 已被修改。因为这两个参数都是静态的，即使是在运行时间间接地对其进行改变也是不可能的。

1.2.6 _PGA_MAX_SIZE

参数 `_PGA_MAX_SIZE` 限制了单个进程下所有工作区的最大大小。

- 当 PAT 值低于 1 GB 时，`_PGA_MAX_SIZE` 为 200 MB。
- 当 PAT 值在 1 GB 和 2 GB 之间时，`_PGA_MAX_SIZE` 大小为 PAT 的 20%。
- 当超过 2 GB 时，随着 PAT 的增加 `_PGA_MAX_SIZE` 也会不断增长，但是会以一个较低的比例增长，这样 `_PGA_MAX_SIZE` 就会低于 PAT 值的 20%。
- 限制 `_PGA_MAX_SIZE` 在 480 MB 会对 4 GB 的 PAT 值产生影响。
- 当超过 4 GB 时，增加 PAT 的值将不会使得 `_PGA_MAX_SIZE` 的值高于 480 MB。
- 在 Oracle9i 中，`_PGA_MAX_SIZE` 限制在 200 MB 内。

就像 `PGA_AGGREGATE_TARGET`，`_PGA_MAX_SIZE` 是一个动态参数，可以用 `ALTER SYSTEM` 进行修改。通过改变 `_PGA_MAX_SIZE` 增加 `_SMM_MAX_SIZE`，和修改 `PGA_AGGREGATE_TARGET` 的方式类似。然而，`_SMM_MAX_SIZE` 为 `_PGA_MAX_SIZE` 的 50% 这一规则对手动修改 `_PGA_MAX_SIZE` 是不成立的。下面是一个例子，通过修改 `PGA_AGGREGATE_TARGET` 可以使 `_PGA_MAX_SIZE` 增大到超过 480 MB 的限制：

```
SQL> @auto_pga_parameters
NAME                                Value (KB) DESCRIPTION
-----
pga_aggregate_target                1048576 Target size for the aggregate PGA memory
                                         consumed by the instance
_pga_max_size                        209700 Maximum size of the PGA memory for one
                                         process
_smm_max_size                       104850 maximum work area size in auto mode (serial)
_smm_px_max_size                     524288 maximum work area size in auto mode (global)
SQL> ALTER SYSTEM SET "_pga_max_size"=500m;
System altered.
SQL> @auto_pga_parameters
NAME                                Value (KB) DESCRIPTION
-----
pga_aggregate_target                1048576 Target size for the aggregate PGA memory
                                         consumed by the instance
_pga_max_size                        512000 Maximum size of the PGA memory for one
```

```

process
_smm_max_size      209715 maximum work area size in auto mode (serial)
_smm_px_max_size    524288 maximum work area size in auto mode (global)

```

通过增加 PGA_MAX_SIZE，工作区有效大小也可以增加，而不用扩展整个实例的可用内存。内存有限可能会避免一些分页活动。只要很少的会话同时请求大工作区，即对 PGA 内存的争用较低，这可能会使得涉及大数据集排序操作的响应时间更短。通过修改 PGA_MAX_SIZE，_SMM_MAX_SIZE 的值可动态设置得比正常限制范围 240 MB 大。

1.2.7 _SMM_MAX_SIZE

参数 _SMM_MAX_SIZE 限制了单个进程下一个单独工作区的最大大小。

- 当 PAT 值低于 512 MB 时，_SMM_MAX_SIZE 为 PGA_AGGREGATE_TARGET 的 20%。
- 当 PAT 的值大于等于 512 MB 时，_SMM_MAX_SIZE 总是 PGA_MAX_SIZE 值的 50%。
- 在 Oracle9i 中，_SMM_MAX_SIZE 限制在 100 MB 内。以下为当给定参数有效时，一个会话同时有两个活跃工作区的例子：

NAME	Value (KB)
-----	-----
pga_aggregate_target	1536000
_pga_max_size	307200
_smm_max_size	153600
_smm_px_max_size	768000

```

C:> sql_workarea_active_hash.pl
SID  TIME HASH_VALUE      TYPE WORK_AREA_SIZE MAX_MEM_USED PASSES TMP_SIZE
159  57:46 1705656915  SORT (v2)          133.7         133.7      0         0
159  57:46 3957124346  HASH-JOIN           16.2          15.7      0        105
...
159  57:52 1705656915  SORT (v2)          133.7         133.7      0         0
159  57:52 3957124346  HASH-JOIN          108.3          96.1      1        138

```

Perl 脚本 sql_workarea_active_hash.pl 输出包括来自 V\$SQL_WORKAREA_ACTIVE 的列 HASH_VALUE 和 TYPE。两个工作区相加超过了 _SMM_MAX_SIZE 但是没有超过 PGA_MAX_SIZE。

1.2.8 _SMM_PX_MAX_SIZE

_SMM_PX_MAX_SIZE 始终设置为 PGA_AGGREGATE_TARGET 的 50%。对于 _SMM_PX_MAX_SIZE 没有任何限制（至少不在 PGA_AGGREGATE_TARGET 的 10 MB 到 32 GB 这个测试范围内）。在 Oracle9i 中，_SMM_PX_MAX_SIZE 为 PGA_AGGREGATE_TARGET 的 30%。

1.2.9 共享服务器

在 Oracle10g 中，共享服务器 (shared server) 使用自动 PGA 内存管理。Oracle9i 共享服务器使用 *_AREA_SIZE 参数，也就是说，它看起来就像是执行了 ALTER SESSION SET WORKAREA_SIZE_POLICY=MANUAL。因此将 SORT_AREA_SIZE 留在 Oracle9i 的 PFILE 或者 SPFILE 中是有效的，然后给它设置一个更有用的值，比如 1 048 576 而不是默认的 65 536。当然在 Oracle10g 中给 SORT_AREA_

SIZE、HASH_AREA_SIZE 等设置有意义的值也是有效的，因为会话有可能使用手动工作区分配 (WORKAREA_SIZE_POLICY= MANUAL) 来运行。

1.2.10 并行执行

隐藏参数 `_SMM_PX_MAX_SIZE` 适用于并行执行，但是确切地如何应用还需要进一步的测试。关于 PX (Parallel Execution, 并行执行)，重要的是要记住，一个 n 度并行全表扫描是在 n 个并行执行进程中切分工作，这样每个进程处理的数据量大约相当于全部数据量的 n 分之一。数字 n 通常称作 DOP (并行度)。

每个并行执行进程分配各自的工作区。由于每个进程仅处理数据的一小部分，在并行模式下单独进程所需的工作区比在串行模式下一个单独的工作区要小。

原来，`_SMM_PX_MAX_SIZE` 对最大工作区大小作出了额外的限制，它被用于并行执行进程中。每个 PX 进程都不可能使用超出 `_SMM_PX_MAX_SIZE/DOP` 的内存。`_SMM_MAX_SIZE` 对每个进程的限制还是对 PX 有效的，这样可用内存就是 `_SMM_MAX_SIZE` 和 `_SMM_PX_MAX_SIZE/DOP` 中较小的一个。完全在内存中进行排序，必须满足以下两个条件。

- 每个 PX 进程的数据量必须低于 `_SMM_MAX_SIZE`。
- 每个 PX 进程的数据量必须低于 `_SMM_PX_MAX_SIZE/DOP`。

让我们运行一些例子。之前测试的结果显示，从测试表执行 SELECT 就有大概 133 MB 的数据量。因此，当 DOP 为 4 时，每个 PX 进程需要的工作区大小约为 133 MB，对于一个最优排序来说，除以 4 即大约 34 MB。稍微进位则为 40 MB，这使得其在 PX 进程间允许一定的数据量波动，我们将设置 `_SMM_MAX_SIZE=40960`，因为 `_SMM_MAX_SIZE` 的单位为 KB。为了避免 PGAAggregateTarget 或 `_SMM_PX_MAX_SIZE` 成为限制因素，我们也设置了这两参数为 `_SMM_MAX_SIZE` 的 DOP 倍或者 160 MB。要设置这些参数，将以下 3 行写到一个参数文件然后使用 STARTUP PFILE 重启实例：

```
pga_aggregate_target=160m
_smm_px_max_size=163840 # in KB
_smm_max_size=40960 # in KB
```

使用脚本 `auto_pga_parameters.sql` 验证该设置得到如下结果：

NAME	Value (KB)	DESCRIPTION
pga_aggregate_target	163840	Target size for the aggregate PGA memory consumed by the instance
_pga_max_size	204800	Maximum size of the PGA memory for one process
_smm_max_size	40960	maximum work area size in auto mode (serial)
_smm_px_max_size	163840	maximum work area size in auto mode (global)

接下来，需要在 SELECT 语句中加入 FULL 和 PARALLEL 提示以进行并行操作。

```
SQL> SELECT /*+ FULL(r) PARALLEL(r, 4) */ * FROM random_strings r ORDER BY 1;
```

在 DOP 为 4 的情况下运行并行查询，通过 `sql_workarea_active.pl` 进行监视，得到的结果如下所示：

SID	TIME	WORK_AREA_SIZE	MAX_MEM_USED	PASSES	TEMPSEG_SIZE
143	06:36	1.7	1.6	0	0
144	06:36	1.7	1.6	0	0
145	06:36	2.6	2.1	0	0
146	06:36	1.7	1.6	0	0
...					
145	06:43	32.3	39.6	0	0
...					
146	06:46	31.6	31.6	0	0
...					
144	06:48	31.4	31.4	0	0
...					
143	06:50	31.2	31.2	0	0

正如所料，执行了一个最优排序，响应时间为 14 秒。DOP 减半导致只有两个进程共享工作负载，以下是测量结果：

SID	TIME	WORK_AREA_SIZE	MAX_MEM_USED	PASSES	TEMPSEG_SIZE
140	23:48	3.1	2.7	0	0
147	23:48	3.8	3.1	0	0
...					
147	24:03	1.2	40	1	71
...					
140	24:08	1.2	40	1	63

在这里，_SMM_MAX_SIZE使得响应时间降低到约 20 秒，因为当DOP为 2 时每个进程需要约 75 MB的工作区大小，但只有 40 MB可用，导致了one-pass排序并且溢出到磁盘。现在让我们回到开始时DOP为 4 的情况——_SMM_PX_MAX_SIZE降到数据量除以DOP之后的值同样会导致溢出到磁盘。以下是这些设置在DOP为 4 时的结果：

```
pga_aggregate_target=160m
_smm_px_max_size=122880 # in KB
_smm_max_size=40960 # in KB
```

这次，_SMM_PX_MAX_SIZE 是限制因素。

SID	TIME	WORK_AREA_SIZE	MAX_MEM_USED	PASSES	TEMPSEG_SIZE
143	33:27	1.7	1.7	0	0
...					
145	33:41	1.2	30	1	40
...					
146	33:44	1.2	30	1	32
...					
144	33:46	1.2	30	1	32
...					
143	33:49	1.2	30	1	31

所有 slave 将它们的工作区溢出到磁盘，因为工作区被限制在 120 MB/DOP=40 MB 内，并且在 22 秒内完成了查询。

1.2.11 小结

当使用自动 PGA 内存管理，三个隐藏参数_PGA_MAX_SIZE、_SMM_MAX_SIZE 和 _SMM_PX_MAX_SIZE

就在幕后工作以加强对内存消耗的限制。参数 `_PGA_MAX_SIZE` 限制单个进程使用的所有工作区的大小。参数 `_SMM_MAX_SIZE` 限制无论是串行还是并行执行情况下一个单独工作区的大小。当使用并行执行时，就会对进程涉及使用的所有工作区总的大小设置一个额外限制，该限制使用参数 `_SMM_PX_MAX_SIZE` 进行控制。在这些限制范围内，由于修改了 PAT，所以这三个参数都在运行时重新计算。这三个参数可以手动设置以覆盖该计算结果。

1.3 EVENT

初始化参数 `EVENT` 在 *Oracle Database Reference* 手册中在文档中仅有部分说明。该参数的语法以及可以设置在手册中的事件并没有在文档中说明。手册指出该参数不允许被使用，除非是在 Oracle Support Services 的指导下。

参数 `EVENT` 可以用来设置一个或多个实例级的事件。通过这种方式设置的事件在一个实例的整个生命周期都有效。所有其他设置事件的方法（比如 `DBMS_SYSTEM`）并不会覆盖到一个实例的整个生命周期。该参数适用于其他手段设置事件不可行或者事件刚好必须在一个进程启动时设置的情况。向 Oracle Support Services 请求技术支持的处理过程大概也就涉及设置这些事件。熟悉参数 `EVENT` 的 DBA 可以降低对 Oracle Support 的依赖，并可能找到解决办法或收集诊断数据而无需求助于技术支持。

1.3.1 语法

通过参数 `EVENT` 设置的事件与使用其他方法（比如 `ALTER SESSION`、`ALTER SYSTEM` 和 `ORADEBUG`）设置的事件可能是相同的事件。更进一步的共性，上述方法与参数 `EVENT` 中事件规范语法是相同的。可通过输入一些由分号分隔的事件规范设置多个事件。语法为：

```
event='event_specification1[:event_specificationN]*'
```

括号表示一个元素是可选的。星号显示，前面的元素可能会重复。对于个别事件规范的语法如下：

```
event_number trace name context forever, level event_level
```

占位符 `event_number` 和 `event_level` 都是整数。大多数事件编号都在范围 10 000 与 10 999 之间^①。在 UNIX 系统上，这些事件在文件 `$ORACLE_HOME/rdbms/msg/oraus.msg` 列出来了，并且还有说明。大多数事件支持的事件级别在文件中没有具体说明，因此在 Oracle Support 的参与下去确定正确的级别是有必要的。OERR 工具可以用来检索某一事件的描述信息。以下是一个事件关闭 CBO（Cost-Based Optimizer，基于成本的优化器）访问路径的例子：

```
$ oerr ora 10196
10196, 00000, "CBO disable index skip scan"
// *Cause:
// *Action:
```

① 事件 14532（修复了 10.2.0.3 中在已分区对象 DDL 对共享池内存过度使用的错误）和 38068（CBO 使其能覆盖对索引选择的猜测影响）是此规则的例外示例。

它同样可以请求一个诊断转储，当出现一个 ORA-*nnnnn* 错误时。它的语法必须和 ALTER SESSION/SYSTEM SET EVENTS 一起使用的语法是相同的（第 13 章介绍）。事件的详细资料见第三部分。

1.3.2 在实例级别均衡事件

以下几种情况下需要在实例级别设置事件。

- 启用或禁用错误修复。
- 启用或禁用功能，如基于成本的优化器访问路径。
- 跟踪一个实例中所有进程的某些代码路径或功能。
- 启用或禁用某些检查。
- 在任何一个数据库会话中无论何时发生 ORACLE 错误 (ORA-*nnnnn*) 都会写入诊断转储。

考虑参数 EVENT，无论何时事件必须刚好在一个实例的一个或所有进程启动时设置。虽然获取多个进程的 SQL 跟踪文件是完全可行的，例如通过使用参数 EVENT 设置事件 10046 获取那些来源于一个连接池的 SQL 跟踪文件，但我强烈反对使用该过程，因为还有更成熟的方法，比如使用 logon 触发器或 DBMS_MONITOR。使用参数 EVENT 在第 8 或 12 级设置事件 10046 比在实例级别设置 SQL_TRACE=TRUE 更好，因为等待事件和绑定可能被包含在里面；然而，二者跟踪实例的每一个进程会带来不必要的开销。我当然不会愿意通过筛选数十甚至数百个跟踪文件来寻找少数有关的文件，当其他功能可以仅跟踪感兴趣的进程时。

1.3.3 案例研究

RMAN (Recovery Manager，恢复管理器) 支持将备份写入到文件系统以及第三方媒体管理器中。将备份写入到文件系统可以立即生效并且不会带来额外的开销。因为写入到本地的文件系统并不能保障数据库服务器硬件整体不会失效，RMAN 同样支持向远程文件系统或者通过 NFS 向 NAS (Network-Attached Storage，网络连接存储) 阵列写入。如果没有使用挂载选项 hard、rsz=32768、wsz=32768，RMAN 将拒绝写入到一个 NFS 文件系统。然而，某些 RMAN 的发行版本在遇到这种需求仍将抛出错误。在这种情况下，Oracle Support 建议在级别 32 设置事件 10298 作为一个临时替代办法，直到基本问题得以解决。

这是在实例级别使用参数 EVENT 设置一个事件的情况。使用其他方法，比如 ORADEBUG 或 DBMS_SYSTEM，则不可能为 RMAN 产生的多个进程及时设置事件。此外，每个实例使用 ALTER SYSTEM SET EVENTS 启动后设置事件则会过于繁琐。

1.4 OS_AUTHENT_PREFIX

初始化参数 OS_AUTHENT_PREFIX 在 *Oracle Database Reference* 手册中有相应的文档说明。在通过网络进行连接时，以字符串 OPS\$ 为前缀的数据库用户名允许通过操作系统和密码文件身份认证进行本地身份认证，这并没有在文档中说明。

出于安全考虑应该设置 REMOTE_OS_AUTHENT=FALSE，那么就不可能使用外部验证用户通过网络（即使用 Oracle Net 以及 TCP/IP 协议适配器）连接到一个实例。创建一个使用密码身份认证

的 OPS\$ 用户便于在使用 Oracle Net 遗留的适配器进行本地连接时省略用户名和密码，同时能够使用密码文件身份认证方式进行网络连接。

1.4.1 OPS\$ 数据库用户及密码文件身份认证

操作系统身份认证旨在用于本地连接。Oracle Database SQL Reference 10g Release 2 手册对于外部验证用户进行了如下叙述：

EXTERNALLY 子句

指定 EXTERNALLY 来创建一个外部用户。这样的用户必须通过一个外部服务进行身份认证，如操作系统或第三方服务。在这种情况下，Oracle 数据库依赖操作系统或第三方服务的身份认证来确保某个特定的外部用户可以访问特定的数据库用户。

以同样的方式，属于 DBA 用户组的用户（通常是 UNIX 用户组 DBA）可以使用命令 CONNECT/AS SYSDBA 以 SYSDBA 特权而无需输入密码连接，一个外部验证用户可以使用命令 CONNECT/连接。当核实一个外部验证用户的资格时，会向操作系统用户名前置 ORACLE 初始化参数 OS_AUTHENT_PREFIX 的值。如果生成的用户名已经在数据字典中存在并且该用户的 DBA_USERS.PASSWORD=EXTERNAL，那么该用户可以连接而无需输入密码。创建一个外部验证用户的语法如下：

```
CREATE USER <os_authent_prefix><os_user_name> IDENTIFIED EXTERNALLY;
```

在文档中没有说明的是，操作系统身份认证对于使用密码文件身份认证创建的用户同样有效，只要 OPS\$ 的 OS_AUTHENT_PREFIX 仍然为其默认设置 OPS\$。也就是说，使用语法 CREATE USER ops\$os_user_name IDENTIFIED BY password 创建的用户，只要 OS_AUTHENT_PREFIX=ops\$，就可以进行本地连接而无需输入密码。从某种意义上说，这种方法很好地连接了数据库世界和现实世界。交互式数据库会话输入密码以及为本地运行批量作业保存密码已经没有必要了，而且可以通过相同的用户名来进行网络连接。

1.4.2 案例研究

该案例研究是在 UNIX 系统环境下进行的，并且 DBA 用户组的名称为 dba，OPER 用户组的名称为 oper，ORACLE 软件拥有者的组为 oinstall。此外，我们还用了密码文件。在这里，你将会看到一个不属于上述任何三个用户组的用户怎样被授予 SYSOPER 特权，使得它可以启动和停止一个实例，而不改变参数或者修改 ORACLE 软件的安装。这是一个使用前一节讨论的未在文档中说明的方法而实现的额外方法。

首先，我们验证参数 OS_AUTHENT_PREFIX 的默认值为 ops\$。

```
SQL> SHOW PARAMETER os_authent_prefix
```

NAME	TYPE	VALUE
os_authent_prefix	string	ops\$

接下来，我们创建一个数据库用户，它的名字通过在操作系统用户名前置字符串 ops\$ 组成，

在本案例中为 ndebes，并授予新用户特权 CONNECT 和 SYSOPER。

```
SQL> CREATE USER ops$ndebes IDENTIFIED BY secret;
User created.
SQL> GRANT CONNECT, SYSOPER TO ops$ndebes;
Grant succeeded.
SQL> SELECT * FROM v$pwfile_users;
USERNAME                                SYSDBA SYSOPER
-----
SYS                                     TRUE  TRUE
OPS$NDEBES                             FALSE TRUE
```

从图 1-3 可以看出，数据库用户 OPS\$NDEBES 可以从 Windows 系统通过 Oracle Net TCP/IP 适配器连接。密码文件身份认证是必要的，因为设置了 REMOTE_OS_AUTHENT=FALSE。

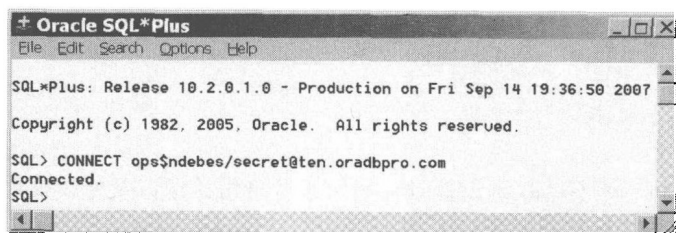


图 1-3 通过 Oracle Net TCP/IP 适配器的 SQL*Plus 会话

回到 UNIX 系统上，操作系统用户 ndebes 可以无需输入密码进行连接。

```
$ id
uid=500(ndebes) gid=100(users) groups=100(users)
$ sqlplus /
SQL*Plus: Release 10.2.0.3.0 - Production on Wed Sep 5 08:02:33 2007
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production
SQL> SHOW USER
USER is "OPS$NDEBES"
```

由于有了密码文件身份认证和一个密码文件，使用 AS SYSOPER 连接也可以奏效。

```
SQL> CONNECT ops$ndebes/secret AS SYSOPER
Connected.
SQL> SHOW USER
USER is "PUBLIC"
SQL> SELECT * FROM session_privs;
PRIVILEGE
-----
CREATE SESSION
RESTRICTED SESSION
SYSOPER
```

由于是 SYSOPER 特权，数据库用户 OPS\$NDEBES 可以停止和重启实例。

```
SQL> SHUTDOWN IMMEDIATE
Database closed.
Database dismounted.
```

```

ORACLE instance shut down.
SQL> STARTUP
ORACLE instance started.
Database mounted.
Database opened.

```

与 SYSDBA 相反, SYSOPER 特权并不包括访问数据字典视图或表, 但是允许使用 ARCHIVE LOG LIST 进行监测。SYSOPER 特权只能访问对 PUBLIC 用户可访问的数据库对象。

```

SQL> SELECT startup_time FROM v$instance;
SELECT startup_time FROM v$instance
      *
ERROR at line 1:
ORA-00942: table or view does not exist
SQL> ARCHIVE LOG LIST
Database log mode                No Archive Mode
Automatic archival               Disabled
Archive destination              /opt/oracle/product/db10.2/dbs/arch
Oldest online log sequence       18
Current log sequence             19

```

当 OS_AUTHENT_PREFIX 不是默认设置时, 操作系统身份认证以及密码文件身份认证的组合优势就不可用了。SYSDBA 特权仅能被授予采用密码文件身份认证创建的数据库用户, 但显然这些用户在连接时必须输入正确的密码。问题就在于尽管有一个指定的密码, 但是当 OS_AUTHENT_PREFIX 不是默认值时, 对操作系统身份认证的检查(文档中未说明该检查)并没有执行。

```

SQL> ALTER SYSTEM SET os_authent_prefix='' SCOPE=SPFILE;
System altered.

```

由于 OS_AUTHENT_PREFIX 现在是一个长度为 0 的字符串, 操作系统用户名与数据库用户名是相同的。

```

SQL> CREATE USER ndebes IDENTIFIED BY secret;
User created.
SQL> GRANT CONNECT, SYSOPER TO ndebes;
Grant succeeded.

```

为了使 OS_AUTHENT_PREFIX 更改的值生效, 必须重新启动该实例。显然, 如果不输入密码 secret, 操作系统用户 ndebes 将不能以数据库用户 ndebes 进行连接。

```

$ sqlplus -s /
ERROR:
ORA-01017: invalid username/password; logon denied

```

当设置用户的身份认证方式为操作系统身份认证时, 字符串“EXTERNAL”将会被存储在 DBA_USERS.PASSWORD 中, 而不是存储密码的散列值。

```

SQL> ALTER USER ndebes IDENTIFIED externally;
User altered.
SQL> SELECT password FROM dba_users WHERE username='NDEBES';
PASSWORD
-----
EXTERNAL

```

现在操作系统用户 ndebes 能够连接而无需输入密码了。

```
$ id
uid=500(ndebes) gid=100(users) groups=100(users)
$ sqlplus /
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production
SQL> CONNECT ndebes/secret as SYSOPER
ERROR:
ORA-01031: insufficient privileges
```

然而，现在的外部验证数据库用户则丧失了使用存储在密码文件中的密码并以 SYSOPER 身份连接的能力。对 SYSDBA 同样有效。

1.4.3 小结

有一个未在文档中说明的代码路径，它使得以 OPS\$为用户名开头的数据库用户可以执行操作系统身份认证，即使这些用户是采用密码身份认证的方式创建的。这就结合了操作系统身份认证以及密码身份认证两种不同并相互排斥的方法的优势。为了利用未在文档中说明的特性，初始化参数 OS_AUTHENT_PREFIX 必须为默认值 ops\$。该特性还可以用来建立一个具有 SYSDBA 或 SYSOPER 特权的用户，而其并不属于操作系统用户组 DBA 或 OPER，但是可以进行本地连接而不需要输入密码。像这样的用户仅需要在通过网络进行连接或者需要一个具有 SYSDBA 或 SYSOPER 特权的会话时输入密码。独立数据库用户需要不具有未在文档中说明的特性，或者 OS_AUTHENT_PREFIX 的非默认设置有效。如果你正在处理一个安全敏感的环境并且需要确认一个入侵者不会利用该特性，你应通过给参数 OS_AUTHENT_PREFIX 指定一个非默认值以禁用该特性。

1.5 源代码库

表 1-2 列出了本章的源代码文件以及它们的功能。

表1-2 在文档中部分说明的参数源代码库

文 件 名	功 能
auto_pga_parameters.sql	检索所有无论在文档中有无记录都会影响SQL工作区分配的参数
pga_aggregate_target_iterator.sql	该脚本使得PGA_AGGREGATE_TARGET在10 MB和32 GB间改变，并且对每一次迭代都调用一次auto_pga_parameters.sql
row_factory.sql	创建一个返回任意行数据的管道表函数
sort_random_strings.sql	该脚本启用SQL*Plus的AUTOTRACE，并且从测试表RANDOM_STRINGS中选择行数据
sql_workarea_active.pl	该Perl脚本通过查询动态性能视图V\$SQL_WORKAREA_ACTIVE监测工作区
sql_workarea_active_hash.pl	该Perl脚本通过查询动态性能视图V\$SQL_WORKAREA_ACTIVE监测工作区。输出中包含SQL语句的散列值

通过视图 V\$PARAMETER 能够访问在文档中说明的初始化参数。此视图基于 X\$固定表 X\$KSPPI 和 X\$KSPPCV (详见第四部分)。所有的隐藏参数均以一個或者两个下划线开头。视图 V\$PARAMETER 就通过这种方式仅列出在文档中说明的参数。在 Oracle9i R2 中有 540 个未在文档中说明的参数, Oracle10g R2 中有 1 124 个, Oracle11g 中有 1 627 个。这些参数的名字和简短的描述可以通过以下查询语句进行检索 (见文件 hidden_parameters.sql):

```
SQL> SELECT ksppinm name,  
ksppstvl value,  
ksppdesc description  
FROM x$ksppi x, x$ksppcv y  
WHERE (x.indx = y.indx)  
AND x.inst_id=userenv('instance')  
AND x.inst_id=y.inst_id  
AND ksppinm LIKE '\_%' ESCAPE '\'  
ORDER BY name;
```

源代码库中包含了 Oracle9i、Oracle10g 以及 Oracle11g 各发行版中未在文档中说明的参数的完整列表。由于未在文档中说明的参数数目过于庞大, 要想调研清楚这些参数的作用并且在文档中加以记录说明是不可能的。所以我选择了 _TRACE_FILES_PUBLIC 和 _ASM_ALLOW_ONLY_RAW_DISKS 这两个有用的隐藏参数作为例子并在本章进行详细探讨。当然还有其他很多在某些情况下很有用的参数。有些可能是被用来解决 bugs (例如 _DISABLE_RECOVERABLE_RECOVERY), 而其他可能对性能有影响 (例如 _ROW_CACHE_CURSORS, _ASM_AUSIZE)。还有其他一些则可用来从数据库中挽救数据, 当文档中记录的恢复过程因为无效的备份过程或者丢失目前在线的 redo 日志而失败时 (_ALLOW_RESETLOGS_CORRUPTION、_OFFLINE_ROLLBACK_SEGMENTS)。正常情况下隐藏参数只应该在 Oracle Support Services 的帮助下进行设置。

2.1 跟踪文件权限与 _TRACE_FILES_PUBLIC 参数

跟踪文件要么根据需求创建 (例如使用 ALTER SYSTEM SET SQL_TRACE=TRUE), 要么在发生内部错误时创建。跟踪文件的前台进程会被定向到设置有 USER_DUMP_DEST 参数的目录, 而跟踪文件的后台进程则使用设置有初始化参数 BACKGROUND_DUMP_DEST 的目录。无论在哪种情况下, 文件名

的后缀都是.trc。跟踪文件默认只能被 Oracle 安装用户（通常是 oracle）或者安装用户组（通常是 oinstall）的成员读取。如果数据库管理员不属于安装用户组，无论是谁都是不能读取跟踪文件的。

由于跟踪文件可能包含敏感信息，无论是作为绑定变量值还是常量，恰当的做法是限制默认权限。在一个测试系统中，开发人员启用 SQL 跟踪然后需要使用 TKPROF 分析输出，所以最好允许任何可登录系统的用户去读取跟踪文件。一个隐藏参数 _TRACE_FILES_PUBLIC 就可以用来使得任何人都能读取新生成的跟踪文件。运行脚本 hidden_parameter_value.sql 的结果如下所示，默认的静态参数设置为 FALSE：

```
$ cat hidden_parameter_value.sql
col name format a33
col value format a36
set verify off
SELECT x.ksppinm name, y.ksppstvl value
FROM x$ksppi x, x$ksppcv y
WHERE x.inst_id = userenv('Instance')
AND y.inst_id = userenv('Instance')
AND x.indx = y.indx
AND x.ksppinm='&hidden_parameter_name';
$ sqlplus -s / as sysdba @hidden_parameter_value.sql
Enter value for hidden_parameter_name: _trace_files_public
NAME                                VALUE
-----
_trace_files_public                  FALSE
```

让我们看一下用户转储目标文件的权限：

```
SQL> SHOW PARAMETER user_dump_dest
NAME                                TYPE                                VALUE
-----
user_dump_dest                      string                             /opt/oracle/obase/admin/TEN/udump
SQL> !cd /opt/oracle/obase/admin/TEN/udump; ls -l
total 68
-rw-r----- 1 oracle oinstall 1024 Jul 21 21:26 ten1_ora_11685.trc
-rw-r----- 1 oracle oinstall 874 Jul 24 02:56 ten1_ora_13035.trc
-rw-r----- 1 oracle oinstall 737 Jul 24 02:56 ten1_ora_13318.trc
```

正如所料，读取权限仅授予了文件所有者和 oinstall 用户组^①。如果使用了服务器参数文件（SPFILE），_TRACE_FILES_PUBLIC 必须使用 ALERT SYSTEM 命令进行修改。参数名称必须添加双引号，因为它以下划线（_）开头。

```
SQL> ALTER SYSTEM SET "_trace_files_public" = TRUE SCOPE=SPFILE;
```

当使用文本参数文件（PFILE）时参数不需要双引号。由于参数是静态的，实例必须关闭然后重启以使新的设置生效。使用 ORADEBUG（参看第 37 章），我们可以快速地验证其他用户对新创

① UNIX 命令 ls 显示权限的格式是 {r|-}{w|-}{x|-}。这个字符序列重复三次。左边部分显示文件的所有者，中间部分为用户组权限，右边部分显示对任何用户的权限（也称为公众权限）。减号意味着在字符串中相应的位置所代表的权限不被批准。例如，rwxr-xr-x 表示所有者可以读、写和执行，用户组则可以读，任何人都可以读取和执行。

建的跟踪文件的读取权限是否被批准。

```
SQL> ORADEBUG SETMYPID
Statement processed.
SQL> ALTER SESSION SET SQL_TRACE=TRUE;
Session altered.
SQL> SELECT sysdate FROM dual;
SYSDATE
-----
25-JUL-07
SQL> ORADEBUG TRACEFILE_NAME
/opt/oracle/obase/admin/TEN/udump/ten1_ora_18067.trc
SQL> !cd /opt/oracle/obase/admin/TEN/udump;ls -l ten1_ora_18067.trc
-rw-r--r-- 1 oracle oinstall 1241 Jul 25 20:53 ten1_ora_18067.trc
```

通过查看参数名称，就像你所预料的那样，_TRACE_FILES_PUBLIC 对警报日志的权限没有任何影响。

2.2 ASM 测试环境与 _ASM_ALLOW_ONLY_RAW_DISKS 参数

ASM (Automatic Storage Management, 自动存储管理) 本质上是一个卷管理器和供 ORACLE 实例专用的文件系统。卷管理功能包括镜像 (mirroring) 和条带化 (striping)。ASM 实现了 S.A.M.E. (Stripe And Mirror Everything)^① 方法。ASM 使用许多裸设备 (raw device) 连接成一个大的存储池^②，然后作为一个文件系统为 ORACLE 实例提供存储空间。裸盘 (如 SAN 里面的 LUN) 被归类为磁盘组。ASM 可以依靠 RAID 存储阵列来做镜像 (外部冗余) 或者也可以自己做镜像 (正常/高冗余)。如有必要，同一个磁盘组的磁盘可以指定失效组，以此来表明 ASM 的存储系统拓扑结构，如此，镜像复制可以放置在不同的存储阵列或者可以被不同的主机总线适配器访问。

对于想熟悉 ASM，但是没有 SAN 或者因为磁盘空间限制以及缺少权限不能创建裸设备的读者，本章将会演示怎么在 Windows 下使用 cooked file 以及 _ASM_ALLOW_ONLY_RAW_DISKS 建立一个 ASM 的测试环境。UNIX 术语会区分 raw file 和 cooked file^③。cooked file 恰恰与 raw file 相反——文件在一个文件系统中。总之，不是 raw 就是 cooked，对不对？

2.2.1 ASM 隐藏参数

关于 ASM 未在文档中说明的参数可以通过 SYS 用户执行下面的查询进行检索：

```
SQL> SELECT x.ksppinm name, y.ksppstvl value, x.ksppdesc description
FROM x$ksppi x, x$ksppcv y
WHERE x.inst_id = userenv('Instance')
AND y.inst_id = userenv('Instance')
```

① 参看 http://www.oracle.com/technology/deploy/availability/pdf/OOW2000_same_ppt.pdf。

② Oracle10g R2 在 Linux 下同样也支持块设备。这些都使用 O_DIRECT 打开，以消除 Linux 操作系统内核的缓存，对于裸设备也同样如此。在 Linux 平台下它与裸设备的性能是相同的。

③ 将原始空间所对应的磁盘称裸设备 (raw device)，而非原始空间所对应的操作系统文件称操作系统文件 (cooked file)。——译者注

```
AND x.indx = y.indx
AND x.kspinn LIKE '\_asm%' ESCAPE '\ '
ORDER BY name;
```

NAME	VALUE	DESCRIPTION
-----	-----	-----
_asm_acd_chunks	1	initial ACD chunks created
_asm_allow_only_raw_disks	TRUE	Discovery only raw devices
_asm_allow_resilver_corruption	FALSE	Enable disk resilvering for external redundancy
_asm_ausize	1048576	allocation unit size
_asm_blksize	4096	metadata block size
_asm_disk_repair_time	14400	seconds to wait before dropping a failing disk
_asm_droptimeout	60	timeout before offlined disks get dropped (in 3s ticks)
_asm_emulmax	10000	max number of concurrent disks to emulate I/O errors
_asm_emultimeout	0	timeout before emulation begins (in 3s ticks)
_asm_kfdpevent	0	KFDP event
_asm_libraries	ufs	library search order for discovery
_asm_maxio	1048576	Maximum size of individual I/O request
_asm_stripeize	131072	ASM file stripe size
_asm_stripewidth	8	ASM file stripe width
_asm_wait_time	18	Maximum time to wait before asmb exits
_asm_lib_test	0	Osmlib test event
_asm_sid	asm	ASM instance id

这些参数揭示出了 ASM 的一些内部工作原理。默认设置说明 ASM 切分一个分配单元 (_ASM_AUSIZE) 为 128 KB 的块 (_ASM_STRIPEIZE)，然后将这些块分别放置到不超过 8 块不同的磁盘上 (_ASM_STRIPEWIDTH)。对于一些非常大的数据库可能会受益于增加 _ASM_AUSIZE，但是这超出了本书的范围^①。本章仅仅涉及参数 _ASM_ALLOW_ONLY_RAW_DISKS。

2.2.2 为 ASM 配置 Oracle 集群件

为了启动 ASM 实例，一个精简版的 Oracle 集群件必须在同一系统下运行。这是通过命令 %ORACLE_HOME%\bin\localconfig add 完成的，它在 %ORACLE_HOME%\cdata\localhost\local.ocr 创建了一个 OCR (ORACLE Cluster Registry, ORACLE 集群注册)。它同样也为 OCSSD 集群件守护进程创建了一个新的 Windows 服务。OCSSD 记录日志到文件 %ORACLE_HOME%\log<host name>\cssd\ocssd.log。

```
C:> localconfig add
Step 1: creating new OCR repository
Successfully accumulated necessary OCR keys.
```

① 参看 <http://h20331.www2.hp.com/ERC/downloads/4AA0-9728ENW.pdf> 和 Metalink note 368055.1。

```

Creating OCR keys for user 'ndebes', privgrp '..
Operation successful.
Step 2: creating new CSS service
successfully created local CSS service
successfully added CSS to home

```

实现 OCSSD 的服务称为 OracleCSService。你可以通过运行 Windows 命令解释器功能执行 net start 命令验证该服务是有效的。该命令列出所有正在运行的服务。

```

C:> net start
...
OracleCSService
OracleDB_10_2TNSListener
...

```

检查 OCSSD 状态的集群件命令是 crsctl check css。

```

C:> crsctl check css
CSS appears healthy

```

通过运行 crsctl check crs，可以明显的看到对于仅允许本地的配置，只有部分集群件守护进程是活跃的，RAC 所需要的 CRS 和 EVM 守护进程不是必需的。

```

C:> crsctl check crs
CSS appears healthy
Cannot communicate with CRS
Cannot communicate with EVM

```

2.2.3 ASM 实例设置

接下来，我们将为 ASM 存储创建一些 cooked file，ASM 将使用这些文件而不是裸盘。在 Windows 上，命令 asmttool 的未公开开关 -create 被用来创建 ASM 使用的 cooked file。语法如下：

```
asmttool -create <file_name> <file_size_mb>
```

我们将使用命令 asmttool 创建 4 个文件作为“磁盘”。每个文件的大小为 512 MB。

```

C:\> mkdir C:\oradata
C:\> asmttool -create C:\oradata\ARRAY1_DISK1 512

```

使用文件名 ARRAY1_DISK2、ARRAY2_DISK1 及 ARRAY2_DISK2 重复上述 asmttool 命令 3 次以创建 3 个文件。在 UNIX 上，命令 dd 也同样可以完成 asmttool -create 执行的任务。下面是一个创建大小为 512 MB 文件的例子：

```
$ dd if=/dev/zero bs=1048576 count=512 of=ARRAY1_DISK1
```

上面的命令 dd 从设备特殊文件 /dev/zero 读取 512 个大小为 1 MB 的数据块。因为从 /dev/zero 读取数据只是返回二进制零，生成的文件按照 ASM 的要求归零。在 Windows 系统上安装 Cygwin (<http://www.cygwin.com>) 后同样也可以使用命令 dd。命令 dd 的选项 conv=notrunc 很有意思，因为它可以用来在一个文件内将一个区归零以模拟故障或引发数据块误用。

这 4 个文件被用来模拟两个磁盘阵列 (ARRAY1 和 ARRAY2)，每一个磁盘阵列有两个逻辑单元。然后我们将设置 ASM 在两个磁盘阵列间作镜像，在阵列内进行条带化。


```
C:\oradata>ls -l
total 2463840
-rw-rw-rw- 1 ndebes          mkpasswd  536870912 Nov  2 13:34 ARRAY1_DISK1
-rw-rw-rw- 1 ndebes          mkpasswd  536870912 Nov  2 13:38 ARRAY1_DISK2
-rw-rw-rw- 1 ndebes          mkpasswd  536870912 Nov  2 13:40 ARRAY2_DISK1
-rw-rw-rw- 1 ndebes          mkpasswd  536870912 Nov  2 13:38 ARRAY2_DISK2
```

我们需要一个包含 INSTANCE_TYPE=ASM 的参数文件启动 ASM 实例。参数 ASM_DISKSTRING 用来指示 ASM 应该在哪里寻找磁盘。在 %ORACLE_HOME%\database 创建一个包含以下内容并且名为 pfile+ASM.ora 的文件：

```
instance_type = ASM
asm_diskstring = 'c:\oradata\*
```

接下来，使用 oradim 为 ORACLE ASM 实例创建一个 Windows 服务：

```
C:> oradim -new -asmsid +ASM -syspwd secret -startmode manual -srvstart demand
Instance created.
```

命令 oradim 创建并启动一个名为 OracleASMSERVICE+ASM 的 Windows 服务。你可以使用 net start 验证该服务是否正在运行：

```
C:> net start | grep -i asm
OracleASMSERVICE+ASM
```

现在我们准备启动 ASM 实例。

```
C:> set ORACLE_SID=+ASM
C:> sqlplus / as sysdba
SQL*Plus: Release 10.2.0.3.0 - Production on Tue Aug 14 16:17:51 2007
Copyright (c) 1982, 2005, Oracle. All rights reserved.
Connected to an idle instance.
SQL> STARTUP NOMOUNT PFILE=?\database\pfile+ASM.ora
ASM instance started
Total System Global Area  79691776 bytes
Fixed Size                 1247396 bytes
Variable Size              53278556 bytes
ASM Cache                  25165824 bytes
```

下一步，我们创建一个服务器参数文件（SPFILE），这样 ASM 就可以在 SPFILE 中存储实例启动时就应挂载的磁盘组的名称。

```
SQL> CREATE SPFILE FROM PFILE='?\database\pfile+ASM.ora';
File created.
```

这将创建一个名为 spfile+ASM.ora 的 SPFILE。让我们看看 ASM 是否能够识别磁盘中的 cooked file。

```
SQL> SELECT path, header_status FROM v$asm_disk;
no rows selected
```

ASM 没有看到它应使用的任何磁盘。这并不奇怪，因为参数 ASM_ALLOW_ONLY_RAW_DISKS 的默认设置为 TRUE。我们需要关闭实例，并在我们可以修改之前创建的参数文件 SPFILE 之前重启实例。

```
SQL> SHUTDOWN IMMEDIATE
ORA-15100: invalid or missing diskgroup name
ASM instance shutdown
SQL> STARTUP NOMOUNT
ASM instance started
...
SQL> SHOW PARAMETER SPFILE
NAME          TYPE          VALUE
-----
spfile        string        C:\ORACLE\PRODUCT\DB10.2\DATABASE\SPFILE+ASM.ORA
```

由于 `_ASM_ALLOW_ONLY_RAW_DISKS` 是一个静态参数，在改变它之后要重启另一个实例。

```
SQL> ALTER SYSTEM SET "_asm_allow_only_raw_disks"=FALSE SCOPE=SPFILE SID='*';
System altered.
SQL> SHUTDOWN IMMEDIATE
ORA-15100: invalid or missing diskgroup name
ASM instance shutdown
SQL> STARTUP NOMOUNT
ASM instance started
...
SQL> SELECT path, header_status, library, total_mb, free_mb FROM v$asm_disk;
PATH          HEADER_STATUS LIBRARY  TOTAL_MB  FREE_MB
-----
C:\ORADATA\ARRAY1_DISK1 CANDIDATE    System      512        0
C:\ORADATA\ARRAY2_DISK2 CANDIDATE    System      512        0
C:\ORADATA\ARRAY2_DISK1 CANDIDATE    System      512        0
C:\ORADATA\ARRAY1_DISK2 CANDIDATE    System      512        0
```

这次 ASM 确实将 cooked file 识别为在一个磁盘组中使用的磁盘了，因此我们可以继续下去，创建一个外部冗余的磁盘组。通过指定失效组 array1 到第一个磁盘阵列的磁盘（文件 ARRAY1_DISK1 和 ARRAY1_DISK2）以及失效组 array2 到第二个磁盘阵列（文件 ARRAY2_DISK1 和 ARRAY2_DISK2）。ASM 在两个磁盘阵列间做镜像，并且它会自动在每个磁盘阵列中将数据条带化。

```
SQL> CREATE DISKGROUP cooked_dg NORMAL REDUNDANCY
FAILGROUP array1
DISK
'C:\ORADATA\ARRAY1_DISK1' NAME array1_disk1,
'C:\ORADATA\ARRAY1_DISK2' NAME array1_disk2
FAILGROUP array2
DISK
'C:\ORADATA\ARRAY2_DISK1' NAME array2_disk1,
'C:\ORADATA\ARRAY2_DISK2' NAME array2_disk2;
Diskgroup created.
```

之前备选的磁盘现在是一个磁盘组的成员了。

```
SQL> SELECT path, header_status, library, total_mb, free_mb FROM v$asm_disk;
PATH          HEADER_STATUS LIBRARY  TOTAL_MB  FREE_MB
-----
C:\ORADATA\ARRAY1_DISK1 MEMBER        System      512      482
C:\ORADATA\ARRAY1_DISK2 MEMBER        System      512      489
```

```
C:\ORADATA\ARRAY2_DISK1 MEMBER      System      512      484
C:\ORADATA\ARRAY2_DISK2 MEMBER      System      512      487
```

通过比较列 TOTAL_MB 和 FREE_MB 你可以看到，ASM 内部使用了相当多的空间。视图 V\$ASM_DISKGROUP 给出了磁盘组信息。如果你认真读过本章开始对隐藏 ASM 参数的概述，你将会认识以下输出中两个隐藏参数的设置：

```
SQL> SELECT name, block_size, allocation_unit_size, state,
type, total_mb, usable_file_mb
FROM v$asm_diskgroup;
NAME          BLOCK_SIZE ALLOCATION_UNIT_SIZE STATE    TYPE    TOTAL_MB USABLE_FILE_MB
-----
COOKED_DG      4096          1048576 MOUNTED  NORMAL      2048          715
```

列 BLOCK_SIZE 的值来自参数 ASM_BLKSIZE，而 ALLOCATION_UNIT_SIZE 来自 ASM_AUSIZE。你现在可以使用 DBCA 在磁盘组中创建一个数据库。请确保对所有数据文件选择的是 ASM 存储。

2.2.4 磁盘失效模拟

你很有可能从来不必处理 ASM 环境下磁盘失效的情况。但为了使你准备好应变这种情况，你可以使用本章的环境设置去模拟磁盘失效，然后获取维修 ASM 设置的经验。可以通过将 cooked file 放置到一个外部 FireWire 磁盘驱动器、USB 磁盘驱动器或 USB 记忆棒上，然后拔掉磁盘或者记忆棒的连接线来模拟磁盘失效。在 SAN 环境下，磁盘失效可以通过拔掉线缆、改变分区（zoning）配置或移除存储阵列中 LUN（Logical Unit Number，逻辑单元号）的访问权限进行模拟。术语 zoning 是用来描述 SAN（Storage Area Network，存储区域网）管理员将一个 SAN 划分为多个单元并且分配存储到这些单元的配置。在一个 SAN 中每一个磁盘或 LUN 都有一个唯一全局标识称为 WWN（World Wide Number，全球号码）。如果通过改变分区配置导致 WWN 对系统不可见，那么无论是 ASM 或 RDBMS 实例都不能使用该 LUN。

2.3 源代码库

表 2-1 列出了本章的源代码文件以及它们的功能。

表2-1 隐藏参数源代码库

文 件 名	功 能
10g_hidden_parameters.html	Oracle10g未在文档中说明的参数的完整列表，包括默认值和描述
11g_hidden_parameters.html	Oracle11g未在文档中说明的参数的完整列表，包括默认值和描述
9i_hidden_parameters.html	Oracle9i未在文档中说明的参数的完整列表，包括默认值和描述
hidden_parameters.sql	SELECT语句，用来检索所有隐藏参数的值以及描述
hidden_parameter_value.sql	SELECT语句，用来检索单个隐藏参数的值

Part 2

第二部分

数据字典基表

本 部 分 内 容

- 第 3 章 数据字典基表介绍
- 第 4 章 IND\$、V\$OBJECT_USAGE 和索引监控

每个 Oracle 数据库都包含一个保存元数据的数据字典，即关于数据库本身的数据。数据字典对象大多是集群、表、索引和大型对象。数据字典就像一辆汽车的发动机，如果它没有点燃（或者更确切地说，使用 SYS.BOOTSTRAP\$ 引导），那么所有其他花哨的功能将一无是处。从传统意义上说，所有数据字典对象都存储在表空间 SYSTEM 中。随着 Oracle10g 的发布，增加了表空间 SYSAUX。这个新的表空间包含了工作负载信息库基表（表 WRI\$* 和 WRH\$*）和其他一些对象。

如果一个 DBA 知道如何使用数据字典基表，那么他就能够完成一些通过访问建立在字典基表上的数据字典视图而无法完成的任务。当字典视图缺乏所需的功能，或者在数据字典视图出现缺陷时缺乏应对措施，数据字典基表就可以派上用场了。

数据字典在 SQL 语句 CREATE DATABASE 执行后创建，通过运行脚本 \$ORACLE_HOME/rdbms/admin/sql.bsq 创建。除了一些占位符，sql.bsq 是一个普通的 SQL * Plus 脚本。Oracle9i 包含 341 个数据字典基表，Oracle10g 包含 712 个，Oracle11g 包含 839 个。

数据库管理员和用户很少直接访问数据字典基表，因为基表是规范化的，而且晦涩难懂。而拥有前缀 DBA_*, ALL_* 和 USER_* 的数据字典视图则为访问数据库元数据提供了方便。但是，有些数据字典视图没有这三个前缀（例如，AUDIT_ACTIONS）。数据字典视图通过著名脚本 catalog.sql 创建，通过查找 catalog.sql 中视图的定义，可以很容易确定基表中的列在字典视图中所对应的列。

为了获得最佳性能，数据字典中的元数据缓存在字典的高速缓存中。为了进一步证实精心设计的 ORACLE DBMS 的特性不仅仅是只有一个简单的名字这一说法，字典的高速缓存也被称为行高速缓存（row cache）。行高速缓存一词来源于这种缓存只包含单行的行，而不是像高速缓存一样是整个块。这两种缓存都在 SGA 中。更准确地说，该字典高速缓存是共享池（shared pool）的一部分。

通过 SELECT ANY DICTIONARY 的系统权限，DBA 以只读的方式访问数据字典基表。这种权限不应轻易地给予非数据库管理员用户。在 Oracle9i 中更是如此，在 Oracle9i 中，字典基表 SYS.LINK\$ 中包含非加密数据库链接的密码，而通过 SELECT_CATALOG_ROLE 访问的字典视图 DBA_DB_LINKS 隐藏着密码。在升级到 Oracle10g 的过程中，数据库链接密码是经过加密的。表 3-1 列出了一些相关的典型数据库对象的字典表。

表3-1 数据字典基表

对 象	数据字典基表	相关联的DBA_*视图
Clusters (集群)	CLU\$	DBA_CLUSTERS、DBA_SEGMENTS
Database links (数据库链接)	LINK\$	DBA_DB_LINKS
Data files (数据文件)	FILES\$	DBA_DATA_FILES、DBA_FREE_SPACE
Free extents (自由扩展)	FET\$	DBA_FREE_SPACE
Indexes (索引)	IND\$	DBA_INDEXES
Large objects (大对象)	LOB\$	DBA_LOBS
Database objects (数据库对象)	OBJ\$	DBA_OBJECTS、DBA_LOBS、DBA_TYPES
Segments (段)	SEG\$	DBA_SEGMENTS
Tables T (表T)	TAB\$	DBA_TABLES、DBA_LOBS
Tablespaces (表空间)	TS\$	DBA_TABLESPACES、DBA_DATA_FILES、DBA_LOBS
Types (类型)	TYPE\$	DBA_TYPES
Used extents (使用扩展)	UET\$	DBA_SEGMENTS、DBA_FREE_SPACE
Users (用户)	USER\$	DBA_USERS、DBA_DB_LINKS、DBA_LOBS

当然，不能直接更改字典基表，因为这样很容易导致数据库损坏。当数据字典视图不能提供足够信息解决任务时，应该考虑查询字典基表。有时字典视图有错误，但是可以通过直接访问基表解决。脚本 sql.bsq 有很好的注释，通过读这个脚本可能有助于了解该字典基表的结构。

大对象、PCTVERSION 和 RETENTION 的对比

能否直接访问字典基表是 Oracle9i 和 Oracle10g R1 的数据字典视图 DBA_LOBS 中的关键问题。这个视图不能正确报告对 LOB 段的版本设置。在 Oracle9i 中，通过预留一定比例的 LOB 段存储 (SQL 的关键字 PCTVERSION，老办法) 或者撤销段 (SQL 关键字 RETENTION，新方法)，可以达到 LOB 段的多版本阅读的一致性。对于有自动撤销管理的数据库 Oracle9i 默认为 PCTVERSION。对于处于自动撤销管理模式的数据数据库 Oracle10g 默认为 RETENTION。设置 RETENTION 时，不能指定 SQL 语法，只能从参数 UNDO_RETENTION 复制。下面是一个在单个表中使用这两种方式的例子：

```
SQL> CREATE TABLE blog (
  username VARCHAR2(30),
  date_time DATE,
  text CLOB,
  img BLOB)
LOB (text) STORE AS blog_text_clob (RETENTION),
LOB (img) STORE AS blog_img_blob (PCTVERSION 10);
Table created.
SQL> SELECT pctversion, retention FROM user_lobs WHERE table_name='BLOG';
PCTVERSION  RETENTION
-----
10          10800
10          10800
```

```
SQL> SHOW PARAMETER undo_retention
```

NAME	TYPE	VALUE
undo_retention	integer	10800

查询数据字典视图 USER_LOBS 得到的结果显然是不正确的。查看 sql.bsq, 没有任何注释可以说明哪个列是用来区别 PCTVERSION 和 RETENTION, 但看起来列 FLAGS 很可能拥有所需要的信息。下面是 sql.bsq 的相关节选:

```
create table lob$                                     /* LOB information table */
( obj#          number not null,                      /* object number of the base table */
  ...
  lobj#          number not null,                      /* object number for the LOB */
  ...
  pctversion$    number not null,                      /* version pool */
  flags          number not null,                      /* 0x0000 = CACHE */
                                              /* 0x0001 = NOCACHE LOGGING */
                                              /* 0x0002 = NOCACHE NOLOGGING */
  ...
  retention      number not null,                      /* retention value = UNDO_RETENTION */
```

该 PCTVERSION 设置存储在列 PCTVERSION\$ 中, 而撤销 retention 的设置存储在列 RETENTION 中。由于 LOB\$.LOBJ# 对应 DBA_OBJECTS.OBJECT_ID (见文件 catalog.sql 中的 DBA_LOBS 定义), 因此通过连接 DBA_OBJECTS 和 LOB\$, 我们可以查询 LOB \$.FLAGS:

```
SQL> SELECT object_name, flags
FROM sys.lob$ l, dba_objects o
WHERE l.lobj#=o.object_id
AND o.object_name IN ('BLOG_TEXT_CLOB', 'BLOG_IMG_BLOB');
OBJECT_NAME          FLAGS
-----
BLOG_IMG_BLOB        65
BLOG_TEXT_CLOB       97
```

存在信息缺失的情况: 如果指定 retention, 那么 LOB\$.FLAGS 显然是以 32 为单位递增的一个位向量。因此, 如果使用 RETENTION, 设置一个代表 2^5 的位, 我们发现可以写出以下的查询, 它使用函数 BITAND 检测 RETENTION 是否启用:

```
SQL> SELECT owner, object_name,
CASE WHEN bitand(l.flags, 32)=0 THEN l.pctversion$
ELSE NULL
END AS pctversion,
CASE WHEN bitand(l.flags, 32)=32 THEN l.retention
ELSE NULL
END AS retention
FROM sys.lob$ l, dba_objects o
WHERE l.lobj#=o.object_id
AND o.object_type='LOB'
AND OWNER='NDEBES';
OWNER                OBJECT_NAME          PCTVERSION  RETENTION
```

NDEBES	BLOG_IMG_BLOB	10	
NDEBES	BLOG_TEXT_CLOB		10800

这个查询得到的结果与先前执行语句 CREATE TABLE 得到的结果是一致的。直接访问字典基表 SYS.LOB\$ 能解决该问题。

4

IND\$、V\$OBJECT_USAGE 和索引监控

V\$OBJECT_USAGE 视图在 *Oracle Database Reference* 手册和 *Oracle Database Administrator's Guide* 中的文档中部分有说明，它的作用是将索引划分为使用过和未使用两个类别。因为那些未使用的索引对 SELECT 语句毫无用处，而且通过 INSERT、UPDATE 和 DELETE 语句所作的修改必须保留索引，所以清除这些未使用的索引也许是一种更好的选择。

V\$OBJECT_USAGE 视图只能被与某一模式对应的登录用户用来查询该单一模式的索引信息，ALTER INDEX REBUILD 会把索引监控 (index monitoring) 关闭并把重建的索引标记为已使用，最后也挺重要的一点，索引监控有一个性能缺陷，因为当被监控的索引每次被使用时都会导致递归 SQL 语句的执行。这些特性都没有在文档中说明。*Oracle Database SQL Reference* 手册中提到 ALTER INDEX 的 MONITORING USAGE 子句只能在执行 ALTER INDEX 的用户所拥有的索引上使用，这种说法是不正确的。此外，还有一些未在文档中说明的内容，如索引使用监控不能用于索引组织表 (index-organized table) 的主键索引，会提示 ORA-25176: storage specification not permitted for primary key 错误；也不能用于域索引 (domain index)，否则会导致 ORA-29871: invalid alter option for a domain index 错误。

本章针对索引使用信息提出了一种改进的视图，它直接建立在数据字典基表的基础上，并取消了仅仅在当前用户所有的索引上检索信息的限制。该增强的视图考虑到 ALTER INDEX REBUILD 的作用，把那些被 DML 而不是索引重建操作访问过的索引标记为已使用，并允许 DBA 在一个数据库的所有模式中发现多余的索引。

4.1 模式限制

V\$ OBJECT_USAGE 是一个名不副实的视图，它仅仅基于 SYS 模式中的数据字典表，而不是 XS 固定表。V\$ 前缀表明它是一个动态性能视图，但事实并非如此。缺少 OWNER 这一列会让你质疑 DBA 如何了解一个应用程序模式中的哪些索引被使用过。毕竟，像 DBA_INDEXES 和 DBA_SEGMENTS 这样的视图都有 OWNER 这一列，而 V\$ACCESS 和 V\$SEGMENT_STATISTICS 这样的动态性能视图同样如此，这样 DBA 就能随心所欲地查看任何模式的信息。除非你是个全能的 DBA，否则你无法

随心所欲地查看索引使用信息，也不能像检索自己的 DBA 模式一样来检索其他模式的信息。要知道，想获得外模式的索引使用信息的唯一途径是连接到该外模式，而这需要知道密码或第 15 章中讨论的临时更改密码的知识。临时更改密码有一定的风险，因为当更换的密码生效后，应用程序的任何连接尝试都将无功而返。如果操作恰当，将会减小这些事情发生的空间，但仍然会导致一些问题。即使使用 ALTER SESSION SET CURRENT_SCHEMA（见第 5 章）也无济于事，因为它只对当前的模式名有作用，对登录的用户名无能为力。

以下是关于 V\$OBJECT_USAGE 中各列的定义：

```
SQL> DESCRIBE v$object_usage
```

Name	Null?	Type
INDEX_NAME	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
MONITORING		VARCHAR2(3)
USED		VARCHAR2(3)
START_MONITORING		VARCHAR2(19)
END_MONITORING		VARCHAR2(19)

仔细观察 START_MONITORING 和 END_MONITORING 这两列，它们的数据类型是 VARCHAR2 类型。我依稀记得 Oracle 公司建议使用 DATE 而不是 VARCHAR2 列来存储日期和时间信息。嗯，也许对该视图的设计概念是在愚人节被批准的。让我们看看视图的定义。

```
SQL> SET LONG 0815
SQL> SELECT text FROM dba_views WHERE owner='SYS' and
view_name='V$OBJECT_USAGE';
TEXT
-----
select io.name, t.name,
       decode(bitand(i.flags, 65536), 0, 'NO', 'YES'),
       decode(bitand(ou.flags, 1), 0, 'NO', 'YES'),
       ou.start_monitoring,
       ou.end_monitoring
from sys.obj$ io, sys.obj$ t, sys.ind$ i, sys.object_usage ou
where io.owner# = userenv('SCHEMAID')
      and i.obj# = ou.obj#
      and io.obj# = ou.obj#
      and t.obj# = i.bo#
```

罪魁祸首在这里，该视图在调用 USERENV 函数时使用了未在文档中说明的 SCHEMAID 参数。通过这种调用方式，它返回了同 SELECT user_id FROM all_users WHERE username=user 语句的查询结果相同的数字用户标识，并用该数字标识来过滤 SYS.OBJ\$ 这种基于 DBA_OBJECTS 等视图的数据字典基表。因此，DBA 无法获得外模式的索引信息。

4.2 索引使用监控案例研究

我们将采用 HR 示例模式（请参阅 *Oracle Database Sample Schemas* 手册）来做案例研究。首先，我将在 HR 模式上启动索引使用监控。为此，源代码库中的 monitor_schema_indexes.sql 文件

包含了 MONITOR_SCHEMA_INDEXES 函数，用来启动或关闭对模式上所有索引的使用监控。在开始之前，我先带你熟悉这个函数。

4.2.1 MONITOR_SCHEMA_INDEXES 函数

MONITOR_SCHEMA_INDEXES 函数的语法如下所示：

```
FUNCTION site_sys.monitor_schema_indexes (  
    ownname VARCHAR2 DEFAULT NULL,  
    failed_counter OUT NUMBER,  
    monitoring BOOLEAN DEFAULT TRUE  
) RETURN INTEGER AUTHID CURRENT_USER;
```

1. 参数

参 数	描 述
ownname	操作的模式名，如果为NULL，则使用当前的模式
failed_counter	返回由于ORA-00054 resource busy and acquire with NOWAIT specified导致的ALTER INDEX语句执行错误的次数。当另一个会话占据着索引的基表上不兼容的锁时，如表上的事务被打开时，就会发生这种错误
monitoring	用来控制监控的启停，TRUE为启动，FALSE为停止

2. 使用说明

这个函数返回那些被成功更改的索引的数目。如果 FAILED_COUNTER 的值大于 0，最好等到打开的事务已经完成，并在 FAILED_COUNTER 返回 0 之前重新运行程序，即没有对象需要被更改。

3. 示例

为 SH 模式上的所有索引启动索引监控。

```
SQL> VARIABLE success_counter NUMBER  
SQL> VARIABLE failed_counter NUMBER  
SQL> EXEC :success_counter:=site_sys.monitor_schema_indexes(ownname=>'SH', -  
> failed_counter=>:failed_counter);
```

为当前模式的所有索引关闭索引监控。

```
SQL> EXEC :success_counter:=site_sys.monitor_schema_indexes( -  
> failed_counter=>:failed_counter, monitoring=>false);
```

4.2.2 为 HR 模式启动索引监控

要在 HR 模式的所有索引上启动索引使用监控，要用 HR 用户连接，运行 SITE_SYS.MONITOR_SCHEMA_INDEXES。在操作之前，你可能需要查询 V\$OBJECT_USAGE，确认 HR 模式中的所有索引都没有被监控过。

```
SQL> CONNECT hr/secret  
SQL> SELECT * FROM v$object_usage;  
no rows selected  
SQL> VARIABLE success_counter NUMBER
```

```

SQL> VARIABLE failed_counter NUMBER
SQL> SET AUTOPRINT ON
SQL> EXEC :success_counter:=site_sys.monitor_schema_indexes( -
> failed_counter=>:failed_counter);
PL/SQL procedure successfully completed.
FAILED_COUNTER
-----
0
SUCCESS_COUNTER
-----
18
SQL> SELECT table_name, index_name, monitoring, used,
start_monitoring, end_monitoring
FROM v$object_usage ORDER BY 1, 2;
TABLE_NAME INDEX_NAME MONITORING USED START_MONITORING END_MONITORING
-----
DEPARTMENTS DEPT_ID_PK YES NO 10/04/2007 17:21:54
DEPARTMENTS DEPT_LOCATION_IX YES NO 10/04/2007 17:21:55
EMPLOYEES EMP_DEPARTMENT_IX YES NO 10/04/2007 17:21:55
EMPLOYEES EMP_EMAIL_UK YES NO 10/04/2007 17:21:55
EMPLOYEES EMP_EMP_ID_PK YES NO 10/04/2007 17:21:55
EMPLOYEES EMP_JOB_IX YES NO 10/04/2007 17:21:55
EMPLOYEES EMP_MANAGER_IX YES NO 10/04/2007 17:21:55
EMPLOYEES EMP_NAME_IX YES NO 10/04/2007 17:21:55
...

```

SQL*Plus 的 SET AUTOTRACE TRACEONLY EXPLAIN 设置告诉 SQL*Plus 仅在输入的语句上执行 EXPLAIN PLAN，而不是实际执行它们或在 SELECT 语句中取出所有行。我可以邀请你投票吗？请问 EXPLAIN PLAN 会把计划（plan）指出的索引标记为已使用吗？实际访问一个索引时有没有必要取出数据行？请在继续读本书之前投下你的一票。

```

SQL> SET AUTOTRACE TRACEONLY EXPLAIN
SQL> SELECT emp.last_name, emp.first_name, d.department_name
FROM hr.employees emp, hr.departments d
WHERE emp.department_id=d.department_id
AND d.department_name='Sales';
Execution Plan

```

Plan hash value: 2912831499

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		10	340	4 (0)
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	10	180	1 (0)
2	NESTED LOOPS		10	340	4 (0)
* 3	TABLE ACCESS FULL	DEPARTMENTS	1	16	3 (0)
* 4	INDEX RANGE SCAN	EMP_DEPARTMENT_IX	10		0 (0)

Predicate Information (identified by operation id):

```
3 - filter("D"."DEPARTMENT_NAME"='Sales')
4 - access("EMP"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
```

执行计划^① (execution plan) 表明如果查询被执行, 那么就会使用 EMP_DEPARTMENT_IX 索引。接下来让我们来看看 V\$OBJECT_USAGE。

```
SQL> SELECT table_name, index_name, monitoring, used,
start_monitoring, end_monitoring
FROM v$object_usage
WHERE table_name IN ('EMPLOYEES', 'DEPARTMENTS');
TABLE_NAME INDEX_NAME MONITORING USED START_MONITORING END_MONITORING
-----
DEPARTMENTS DEPT_ID_PK YES NO 10/04/2007 17:21:54
DEPARTMENTS DEPT_LOCATION_IX YES NO 10/04/2007 17:21:55
EMPLOYEES EMP_DEPARTMENT_IX YES YES 10/04/2007 17:21:55
EMPLOYEES EMP_EMAIL_UK YES NO 10/04/2007 17:21:55
...
```

EMP_DEPARTMENT_IX 索引确实被标记为已使用 (列 USED=YES), 尽管只执行了 EXPLAIN PLAN。

1. 索引重建

索引重建会对 V\$OBJECT_USAGE 产生影响, 关于这点并未在文档中说明。它设置 V\$OBJECT_USAGE. USED=YES 和 V\$OBJECT_USAGE.MONITORING=NO, 即会终止索引监控。

```
SQL> ALTER INDEX dept_id_pk REBUILD;
Index altered.
SQL> SELECT * FROM v$object_usage WHERE table_name='DEPARTMENTS';
INDEX_NAME TABLE_NAME MONITORING USED START_MONITORING END_MONITORING
-----
DEPT_ID_PK DEPARTMENTS NO YES 10/04/2007 17:21:54
DEPT_LOCATION_IX DEPARTMENTS YES NO 10/04/2007 17:21:55
```

结果有点令人惊讶, 因为索引重建并不是 DBA 感兴趣的一种索引使用。其他 DDL 语句, 如 ANALYZE INDEX index_name VALIDATE STRUCTURE 或者 ANALYZE TABLE table_name VALIDATE STRUCTURE CASCADE 并不会影响 V\$OBJECT_USAGE 的索引状态, 尽管它们需要访问索引段。

2. DML使用的索引

找出那些被 DML 语句使用的索引非常有价值, 而有些因素让该操作比你想象的更加复杂。我们还没有考虑过对那些被标记为已用的索引关闭索引监控的情况。通过调用 MONITOR_SCHEMA_INDEXES 并把 MONITORING 参数设置为 FALSE, 我们就能关闭 HR 模式中所有索引的索引监控。

```
SQL> EXEC :success_counter:=site_sys.monitor_schema_indexes(monitring=>false, -
> failed_counter=>:failed_counter);
FAILED_COUNTER
-----
0
SUCCESS_COUNTER
-----
17
```

① 为提高易读性, TIME 列在执行计划中被省略。

由于索引重建已经关闭了对其中一个索引的监控,而且该函数仅仅考虑那些尚未达到预期状态的索引,所以变量 SUCCESS_COUNTER 的值是 17。让我们看一下 V\$ OBJECT_USAGE 的内容。

```
SQL> SELECT table_name, index_name, monitoring AS monitored,
used, start_monitoring, end_monitoring
FROM v$object_usage
WHERE table_name IN ('EMPLOYEES', 'DEPARTMENTS');
TABLE_NAME INDEX_NAME MONITORED USED START_MONITORING END_MONITORING
-----
DEPARTMENTS DEPT_ID_PK NO YES 10/04/2007 17:21:54
DEPARTMENTS DEPT_LOCATION_IX NO NO 10/04/2007 17:21:55 10/04/2007 18:17:58
EMPLOYEES EMP_DEPARTMENT_IX NO YES 10/04/2007 17:21:55 10/04/2007 18:17:58
EMPLOYEES EMP_EMAIL_UK NO NO 10/04/2007 17:21:55 10/04/2007 18:17:58
...
```

不出所料,我们现在看到的是所有索引都为 MONITORING=NO 的情况。请注意 DEPT_ID_PK 索引和 EMP_DEPARTMENT_IX 索引两者的微妙差别。DEPT_ID_PK 索引通过使用 ALTER INDEX REBUILD 关闭索引监控,而 EMP_DEPARTMENT_IX 索引则通过 MONITOR_SCHEMA_INDEXES 函数的 ALTER INDEX index_name NOMONITORING 来关闭索引监控。前者把 END_MONITORING 设置成 NULL,而后者则需要索引监控被关闭。这可以作为区分索引重建和 DML 引起的真实索引使用的依据。

综合所有结果,必须考虑以下情况。

- 重建的索引被标记为已使用,同时对它们的监控被关闭,而 END_MONITORING 的值被设置为 NULL。由于我们只对 DML 使用的索引感兴趣,所以我们需要排除这种情况。
- DML 使用的索引保留了 MONITORING (YES) 和 END_MONITORING (NULL) 设置。
- 针对那些被 DML 使用后关闭监控的索引,保留 MONITORING=YES 设置,而将 END_MONITORING 设置成实际的时间戳,替代 NULL 值。

下面的查询结果仅仅是被 DML 而不是索引重建标记为已用的索引:

```
SQL> SELECT * FROM v$object_usage
WHERE (monitoring='YES' AND used='YES') OR
(used='YES' AND end_monitoring IS NOT NULL)
ORDER BY index_name;
INDEX_NAME TABLE_NAME MONITORING USED START_MONITORING END_MONITORING
-----
EMP_DEPARTMENT_IX EMPLOYEES NO YES 10/04/2007 17:21:55 10/04/2007 18:17:58
```

这基本上解决了索引监控的问题,但 DBA 仍然没有解决无法获取外模式信息的烦恼,除非用和模式名相同的用户名连接。这时候就该数据字典基表出场了。在研究过 catalog.sql 文件里的 V\$OBJECT_USAGE 的定义之后,写一个增强版的视图就不难了,该视图拥有一个名副其实的名称 DBA_INDEX_USAGE,即允许访问数据库中所有索引的使用信息,而不仅仅限于当前的模式。由于我不打算模仿 Oracle 公司的命名规则来增加麻烦,所以我把该视图简称为 INDEX_USAGE。创建该视图的 view_index_usage.sql 脚本如下所示。它增加了 OWNER 列,是从 SYS.USER\$ 中获取的。USER\$ 必须要和 OBJ\$ 通过 OBJ\$.OWNER#=USER\$.USER# 连接起来,从而获得索引所有者的名字。我通过在模式 SITE_SYS 中创建数据库对象,防止对 SYS 模式中数据字典的干扰。

```

SQL> CONNECT / AS SYSDBA
SQL> GRANT SELECT ON obj$ TO site_sys WITH GRANT OPTION;
SQL> GRANT SELECT ON ind$ TO site_sys WITH GRANT OPTION;
SQL> GRANT SELECT ON object_usage TO site_sys WITH GRANT OPTION;
SQL> GRANT SELECT ON user$ TO site_sys WITH GRANT OPTION;
SQL> CREATE OR REPLACE VIEW site_sys.index_usage
(owner,
INDEX_NAME,
TABLE_NAME,
MONITORING,
USED,
START_MONITORING,
END_MONITORING)
AS
SELECT u.name, io.name index_name, t.name table_name,
       decode(bitand(i.flags, 65536), 0, 'NO', 'YES'),
       decode(bitand(ou.flags, 1), 0, 'NO', 'YES'),
       ou.start_monitoring,
       ou.end_monitoring
FROM sys.obj$ io, sys.obj$ t, sys.ind$ i, sys.user$ u, sys.object_usage ou
WHERE io.owner# = t.owner#
AND io.owner# = u.user#
AND i.obj# = ou.obj#
AND io.obj# = ou.obj#
AND t.obj# = i.bo#;
-- have to grant to public, to allow non DBAs access to the view
-- used by function MONITOR_SCHEMA_INDEXES, which runs with AUTHID CURRENT_USER
GRANT SELECT ON site_sys.index_usage TO PUBLIC;

```

当使用 SYS 或任何拥有相关权限的用户连接时，我们可以获取像 HR 这些外模式的索引使用信息。

```

SQL> SELECT owner, table_name, index_name, monitoring, used
FROM site_sys.index_usage
WHERE owner='HR'
AND ((monitoring='YES' AND used='YES')
OR (used='YES' AND end_monitoring IS NOT NULL));
OWNER TABLE_NAME INDEX_NAME      MONITORING USED
-----
HR      EMPLOYEES      EMP_DEPARTMENT_IX NO      YES

```

4.2.3 小结

这部分主要包括对索引监控的案例研究。利用 MONITOR_SCHEMA_INDEXES 函数和 INDEX_USAGE 视图，DBA 可以监控任意模式的索引的使用情况。那些涵盖使用数据库的应用程序完整代码路径，且在一段时间内未使用的索引，可以被舍弃以减少索引维护开销。这里的问题在于完整的代码路径。你也许永远无法确定应用程序的全部代码路径是否都被执行过。因此，最好在开发和测试的前期使用该功能，而不是冒险在产品数据库阶段舍弃一个索引（在财年结束前的处理阶段可

能会需要它)。

有一种方法在查找使用过的索引时干扰更小，并且不会导致递归 SQL 带来的性能损失，就是在第 6 级或更高级使用 Statspack（见第 25 章）。尽管如此，这个方法风险更大，因为 Statspack 会对 SQL 语句和它们的执行计划进行采样。这样，你就可能得不到某些索引使用的数据，原因是当每次执行计划表明这些索引没有在共享池中被缓存时，都会创建一个 Statspack 快照。

4.3 源代码库

表 4-1 列出了本章的源文件以及它们的功能。

表4-1 索引监控源代码库

文 件 名	功 能
monitor_schema_indexes.sql	包含用于开启或关闭所有索引使用监控的MONITOR_SCHEMA_INDEXES函数，调用view_index_usage.sql脚本创建INDEX_USAGE视图，因为它被该函数使用过
view_index_usage.sql	包含用来访问整个数据库索引使用信息的INDEX_USAGE视图，从而替代仅仅用于当前模式的V\$OBJECT_USAGE等视图

Part 3

第三部分

事 件

本 部 分 内 容

- 第 5 章 10027 事件和死锁诊断
- 第 6 章 10046 事件和扩展 SQL 跟踪
- 第 7 章 10053 事件和基于成本的优化器
- 第 8 章 10079 事件和 Oracle 网络数据包的内容

第 1 部分介绍了在文档中部分说明的初始化参数 `EVENT`，以及使用它进行故障处理可能带来的好处。第三部分则对用 `EVENT` 参数或 `ALTER SESSION/SYSTEM SQL` 语句来设置事件的材料进行了扩展。这些方法之间的区别在于，`EVENT` 参数能够保证配置的持久性并涉及一个 `ORACLE DBMS` 实例的整个生命周期。接下来的章节将把目标集中在死锁诊断、性能数据收集和 Oracle 网络数据包转储 (Oracle Net packet dump)。

5.1 死锁

当两个或多个会话在持有锁的同时又请求另一个锁，就可能产生一个锁的循环链，从而发生死锁。如果这个新的锁请求被批准，这些会话都将陷入死锁，任何一个都将无法完成任务。因此，`ORACLE DBMS` 会检测那些锁相互依赖的循环链，当出现 `ORA-00060: deadlock detected while waiting for resource` 错误时发出信号，并针对可能发生死锁的多个会话中的一个进行回滚。当 `ORACLE` 实例检测到一个死锁就会写到跟踪文件。文档中未说明的 10027 事件能让 DBA 控制生成的诊断信息的数量和类型。

图 5-1 描述了一种发生在两个数据库会话之间的死锁情况。会话 1 在 t_1 时刻锁住了 `EMPLOYEE_ID=182` 这一行，而会话 2 在 t_2 时刻锁住了 `EMPLOYEE_ID=193` 这一行。在 t_3 时刻，会话 1 请求 `EMPLOYEE_ID=193` 这一行的锁，而锁正好被会话 2 持有；因此，会话 1 必须在 `enq:TX - row lock contention` 这个事件队列里等待。在 t_4 时刻，会话 2 请求会话 1 在 t_1 时刻持有的锁。因为授予该锁将产生一个循环链，`DBMS` 会在 t_5 时刻发出 `ORA-00060: deadlock detected while waiting for resource` 错误信号。对会话 1 在 t_3 时刻执行的 `UPDATE` 语句进行回滚。这时候，会话 2 仍然在等待 `EMPLOYEE_ID=182` 这一行上的锁，而这个锁仍然被会话 1 持有。在 `ORA-00060` 错误后会话 1 应该回滚，释放它的所有锁从而允许会话 2 完成对 182 号员工的更新。

为了避免死锁，各行需要所有的数据库会话按照相同的次序锁定。如果这样不可行，死锁和写跟踪文件带来的开销可以通过在执行 `UPDATE` 之前先执行 `SELECT FOR UPDATE NOWAIT` 来避免。如果这样操作后返回 `ORA-00054`，那么该会话需要回滚，并重新尝试整个事务。该 `ROLLBACK` 将允许其他事务完成。这种方法的缺点在于多了一些额外的处理。下面是一个相关的示例：

```

SQL> SELECT rowid FROM hr.employees WHERE employee_id=182 FOR UPDATE NOWAIT;
SELECT rowid FROM hr.employees WHERE employee_id=182 FOR UPDATE NOWAIT
*
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified
SQL> ROLLBACK;
Rollback complete.
SQL> SELECT rowid FROM hr.employees WHERE employee_id=182 FOR UPDATE NOWAIT;
ROWID
-----
AAADNLAAEAAAEi1ABb
SQL> UPDATE hr.employees SET phone_number='650.507.9878'
WHERE rowid='AAADNLAAEAAAEi1ABb';
1 row updated.

```

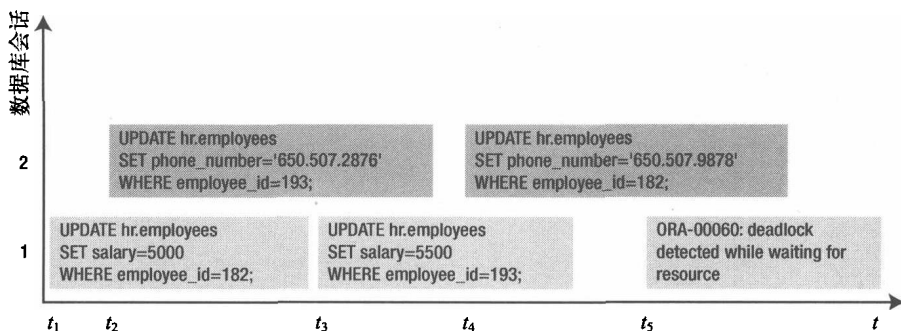


图 5-1 死锁检测

5.2 10027 事件

10027 事件能让 DBA 控制 ORA-00060 错误对应的诊断信息的数量和类型。在默认设置里，ORA-00060 错误对应的跟踪文件包含缓存游标、死锁图、处理状态、相关会话的当前 SQL 语句，以及会话等待历史（在 Oracle10g 和后续版本）。除了当前的 SQL 语句和死锁图，其他所有信息都属于接收到 ORA-00060 错误的会话。10027 事件可用来实现以下对立的目的。

- 减小与 ORA-00060 错误对应的跟踪信息的占用空间，比如，当该问题无法解决的时候，执行该操作。
- 在跟踪信息中加入系统状态转储数据或者调用栈，用来寻找死锁的根源。

在第 1 级写的跟踪信息最少，因为在这个级别，跟踪文件只包含一个死锁图和相关会话的当前 SQL 语句。以下是一个使用 10027 事件第 1 级的 ORA-00060 跟踪文件示例：

```

*** ACTION NAME:() 2007-09-08 05:34:52.373
*** MODULE NAME:(SQL*Plus) 2007-09-08 05:34:52.373
*** SERVICE NAME:(SYS$USERS) 2007-09-08 05:34:52.373
*** SESSION ID:(159.5273) 2007-09-08 05:34:52.372
DEADLOCK DETECTED ( ORA-00060 )
[Transaction Deadlock]

```

The following deadlock is not an ORACLE error. It is a deadlock due to user error in the design of an application or from issuing incorrect ad-hoc SQL. The following information may aid in determining the deadlock:
Deadlock graph:

```

-----Blocker(s)----- -----Waiter(s)-----
Resource Name      process session holds waits process session holds waits
TX-0007000f-000002a4      20      159      X      16      145      X
TX-000a0013-000002a3      16      145      X      20      159      X
session 159: DID 0001-0014-0000004B      session 145: DID 0001-0010-0000004E
session 145: DID 0001-0010-0000004E      session 159: DID 0001-0014-0000004B
Rows waited on:
Session 145: obj - rowid = 0000334B - AAADNLAAEAAAEi1ABb
(dictionary objn - 13131, file - 4, block - 18613, slot - 91)
Session 159: obj - rowid = 0000334B - AAADNLAAEAAAEi2AAE
(dictionary objn - 13131, file - 4, block - 18614, slot - 4)
Information on the OTHER waiting sessions:
Session 145:
  pid=16 serial=1880 audsid=210018 user: 34/NDEBES
  O/S info: user: oracle, term: pts/5, ospid: 24607, machine: dbserver1.oradbpro.com
  program: sqlplus@dbserver1.oradbpro.com (TNS V1-V3)
  application name: SQL*Plus, hash value=3669949024
  Current SQL Statement:
    UPDATE hr.employees SET phone_number='650.507.9878' WHERE employee_id=182
End of information on OTHER waiting sessions.
Current SQL statement for this session:
UPDATE hr.employees SET salary=5500 WHERE employee_id=193

```

包含在事件第 2 级的跟踪文件中的系统状态转储信息，可能有助于诊断死锁发生的原因。系统状态转储数据包含缓冲 SQL 和所有会话的等待历史，而不仅仅是死锁相关会话的当前 SQL 语句。因此，这就使得重现导致死锁的场景成为可能。

事件第 4 级的跟踪文件中包含的调用栈跟踪信息用途不大。它能告诉我们，当检测到死锁时，ORACLE 服务器进程正在执行哪个 C 函数。如果你第一次在应用程序中遇到死锁，推荐你使用 ALTER SYSTEM 临时将 10027 事件设置在第 2 级，如下所示：

```
SQL> ALTER SYSTEM SET EVENTS '10027 trace name context forever, level 2';
```

这将增加你找到死锁根源的可能性。如果这些设置在实例启动过程中保持不变，将需要使用 EVENT 初始化参数。

```
EVENT="10027 trace name context forever, level 2"
```

当你获得了足够的系统状态转储数据用作了进一步的分析，你可以将事件层次降为 1 级。因为锁会在 ORA-00060 跟踪文件写好后被释放，而第 1 级的 10027 事件能确保会话更快地响应。经测试，我发现这一级别的跟踪文件大小只有默认设置下文件大小的 1/100。表 5-1 总结了支持的事件级别，以及在每个级别中包含的跟踪信息^①。

① 所有会话的 SQL 语句都在系统状态转储部分。

表5-1 事件10027和跟踪文件内容

内容/级别	默 认 值	第1级	第2级	第4级
缓存游标	yes	no	yes	yes
调用栈跟踪	no	no	no	yes
死锁图	yes	yes	yes	yes
处理状态	yes	no	yes	yes
SQL语句	yes	yes	yes, 对所有会话	yes
会话等待历史	yes	no	yes, 对所有会话	yes
系统状态	no	no	yes	no

The Oracle Database Performance Tuning Guide 10g Release 2 中提到, 10046 事件在第 8 级可以用来记录等待事件的日志, 并写到一个 SQL 跟踪文件。事件的其他级别没有在文档中说明。如果想启动扩展 SQL 跟踪, 事件 10046 再合适不过了。与 ALTER SESSION 相结合, 事件 10046 是启动扩展 SQL 跟踪并且不需要 DBA 或 SYSDBA 特权的唯一方法。这个事件对那些要集成自跟踪 (self-tracing) 功能的应用程序来说很有用。自跟踪是应用程序依据最终用户的请求来生成性能问题的证据的一种能力。当最终用户对应用程序的性能不满意时, 他可能会启动扩展 SQL 跟踪, 而不需要求助于 DBA。这将能极大地节省时间, 因为 DBA 不需要去确定服务于最终用户的是哪个数据库会话, 同时它也能降低瞬态性能问题不能及时重现带来的风险。

事件 10046 在本书的很多地方都有介绍, 主要是性能诊断相关的内容。第八部分包含许多使用事件 10046 的例子。本章的目的是对该事件和它的级别做一个简单的介绍。该事件在会话和进程级别用途最大, 其中关于该事件在会话级别的使用实例在第 13 章, 而进程级别的实例则被安排在第 37 章。

表 6-1 中列出了支持的事件级别, 其中术语数据库调用 (database call) 指的是执行 SQL 语句的解析、执行和提取过程。

表6-1 SQL跟踪级别

SQL跟踪级别	数据库调用	绑定变量	等待事件
1	yes	no	no
4	yes	yes	no
8	yes	no	yes
12	yes	yes	yes

下面的例子说明了如何使用 10046 事件来跟踪 SQL 语句, 绑定变量和等待事件。跟踪文件来自 Oracle11g R1。要注意 sqlid 这个新的 Oracle11g 参数, 类似于 PARSING IN CURSOR 条目中的 V\$SQL.SQL_ID。

```
SQL> ALTER SESSION SET EVENTS '10046 trace name context forever, level 12';
Session altered.
SQL> VARIABLE id NUMBER
SQL> INSERT INTO customer(id, name, phone)
```



```
VALUES (customer_id_seq.nextval, '&name', '&phone')
RETURNING id INTO :id;
Enter value for name: Deevers
Enter value for phone: +1 310 45678923
1 row created.
```

产生的 SQL 跟踪文件的节选如下所示：

```
*** ACTION NAME:() 2007-11-28 22:02:15.625
*** MODULE NAME:(SQL*Plus) 2007-11-28 22:02:15.625
*** SERVICE NAME:(SYS$USERS) 2007-11-28 22:02:15.625
*** SESSION ID:(32.171) 2007-11-28 22:02:15.625
...
WAIT #6: nam='SQL*Net message to client' ela= 6 driver id=1111838976 #bytes=1 p3=0
obj#=15919 tim=230939271782
*** 2007-11-30 09:45:33.828
WAIT #6: nam='SQL*Net message from client' ela= 235333094 driver id=1111838976
#bytes=1 p3=0 obj#=15919 tim=231174604922
=====
PARSING IN CURSOR #4 len=122 dep=0 uid=32 oct=2 lid=32 tim=231174605324 hv=798092392
ad='6b59f600' sqlid='96032xwrt3v38'
INSERT INTO customer(id, name, phone)
VALUES (customer_id_seq.nextval, 'Deevers', '+1 310 45678923')
RETURNING id INTO :id
END OF STMT
PARSE #4:c=0,e=111,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=231174605317
BINDS #4:
Bind#0
  oacdt=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
  oacflg=03 fl2=1000000 frm=00 csi=00 siz=24 off=0
  kxsbbfp=07a7e7a0 bln=22 avl=04 flg=05
  value=370011
WAIT #4: nam='SQL*Net message to client' ela= 7 driver id=1111838976
#bytes=1 p3=0 obj#=15919 tim=231174606084
EXEC #4:c=15625,e=673,p=0,cr=0,cu=3,mis=0,r=1,dep=0,og=1,tim=231174606139
STAT #4 id=1 cnt=0 pid=0 pos=1 obj=0
op='LOAD TABLE CONVENTIONAL (cr=0 pr=0 pw=0 time=0 us)'
STAT #4 id=2 cnt=1 pid=1 pos=1 obj=15920
op='SEQUENCE CUSTOMER_ID_SEQ (cr=0 pr=0 pw=0 time=0 us)'
*** 2007-11-30 09:45:39.015
WAIT #4: nam='SQL*Net message from client' ela= 5179787 driver id=1111838976
#bytes=1 p3=0 obj#=15919 tim=231179786085
```

关于如何解释扩展 SQL 跟踪文件，以及如何自动生成一个性能诊断资源配置的详情，请参阅第八部分。

10053 事件和基于成本的优化器

如果想理解 CBO (the Cost Based Optimizer, 基于成本的优化器) 的决策和成本计算, 没有比读事件 10053 生成的优化器跟踪文件更好的了。当优化器无法找到一个能产生可接受的响应时间的执行计划时, 如果你打算提出对该优化器的技术支持请求, Oracle Support 通常会需要这样一个跟踪文件。

从本质上讲, CBO 是一个估算 SQL 语句响应时间的数学模型。它接收初始化参数、与表和索引相关的对象统计数据以及代表硬件能力的系统统计数据作为输入。按照 *Oracle9i Database Performance Tuning Guide and Reference Release 2* 所说, 使用优化器做串行计算的成本计算公式如下:

$$\frac{\text{SRds} * \text{sreadtim} + \text{MRds} * \text{mreadtim} + \frac{\text{CPU Cycles}}{\text{cpuspeed}}}{\text{sreadtim}}$$

因此, 单位成本是指完成对一个单块的读操作所花费的时间。表 7-1 阐述了公式中所使用的占位符。如果你对使用 DBMS_STATS.GATHER_SYSTEM_STATS 来收集系统统计信息很熟悉, 那么你对 sreadtim、mreadtim 和 cpuspeed 这三个占位符也不会陌生。它们三个都是所谓的工作量统计信息 (workload statistics) 的一部分, 这些数据来源于对 DBMS 所在系统的评测。设置这些参数 (甚至更多) 的文档化接口是 DBMS_STATS.SET_SYSTEM_STATS 和 DBMS_STATS.IMPORT_SYSTEM_STATS 打包程序。当前的设置可以通过调用 DBMS_STATS.GET_SYSTEM_STATS 打包程序来获取。系统统计数据在 Oracle9i 中是可选的。如果实际测量的数据没有被前面提到的任何接口导入到数据字典的 SYS.AUX_STATS\$ 表中, 那么 Oracle10g 就使用所谓的非工作量统计数据 (noworkload statistics)。

表7-1 CBO成本计算占位符

占 位 符	含 义
SRds	读单块
sreadtim	读单块的时间
MRds	读多块

(续)

占位符	含 义
mreadtim	读多块的时间，即完成一个读多块的请求所花费的时间。读多块的请求所能获取的块数量通过DB_FILE_MULTIBLOCK_READ_COUNT参数来限制
CPU Cycles	执行一条语句需要的CPU周期数
cpuspeed	每秒能执行的指令数 ^①

下面的查询结果来自于一个除 SLAVETHR（平均并行服务 I/O 吞吐量）以外所有工作量统计信息参数都被设置的系统：

```
SQL> SELECT * FROM sys.aux_stats$;
SNAME          PNAME          PVAL1 PVAL2
-----
SYSSTATS_INFO STATUS              COMPLETED
SYSSTATS_INFO DSTART            11-29-2007 13:49
SYSSTATS_INFO DSTOP            11-29-2007 13:49
SYSSTATS_INFO FLAGS              1
SYSSTATS_MAIN CPUSPEEDNW 841.336
SYSSTATS_MAIN IOSEEKTIM         10
SYSSTATS_MAIN IOTFRSPEED       4096
SYSSTATS_MAIN SREADTIM          4
SYSSTATS_MAIN MREADTIM         10
SYSSTATS_MAIN CPUSPEED         839
SYSSTATS_MAIN MBRC              14
SYSSTATS_MAIN MAXTHR           8388608
SYSSTATS_MAIN SLAVETHR
```

工作量参数会被加粗显示。MBRC 是实际的多块读取次数，它来源于 X\$固定表收集的统计信息。MBRC 受 DB_FILE_MULTIBLOCK_READ_COUNT 参数的限制，通常比这个参数略小。DB_FILE_MULTIBLOCK_READ_COUNT 在示例系统上的值为 16，而 MBRC 的实际值只有 14。MAXTHR 是最大的 I/O 吞吐量。SLAVETHR 是平均并行服务 I/O 吞吐量，而 SREADTIM 和 MREADTIM 则已经涵盖在表 7-1 中。

有 3 个非工作量统计信息参数：

- ❑ CPUSPEEDNW（非工作量 CPU 速度）
- ❑ IOSEEKTIM（I/O 查找时间，以 ms 为单位）
- ❑ IOTFRSPEED（I/O 传输速度，以 KB/s 为单位）

CPU 速度的度量单位不是 MHz，相反用的是 Oracle 公司专有的、文档中未说明的度量工具。CBO 的输出是一个执行计划，但是，这并不是来自优化器的唯一输出。此外，它还提供了以下条目：

- ❑ 语句结果集的估计行数

① Oracle10g R2 的 10053 事件跟踪文件包括这一行：
CPUSPEED: 839 millions instructions/sec
这似乎表明 DBMS 是通过计算每秒执行的指令数来测量 CPU 速度的，而这种指令并没有在文档中说明。

- 估计处理的字节数
- 查询块的名称
- 过滤谓词
- 访问谓词
- 列投影信息
- 窥视的绑定变量值
- 概要，即一个导致所选择的执行计划的提示集合（需要 Oracle10g 或更新的版本）

概要数据（outline data）引用查询块的名称，也是用来测试带提示的替代计划的理想基础。从 Oracle10g 开始，前面列表中的信息也可以使用 DBMS_XPLAN.DISPLAY_CURSOR 来获取。尽管如此，要获得概要以及窥视绑定变量（peeked bind），必须用到文档中未说明的格式选项。可使用 OUTLINE 格式选项来包含概要，而使用 PEEKED_BINDS 来获得窥视绑定变量值。还有 ADVANCED 这个文档中未说明的格式选项，它也包含了概要，但是不包括窥视的绑定变量值。下面是一个实例：

```
SQL> SELECT * FROM TABLE(dbms_xplan.display_cursor(null, null,
'OUTLINE PEEKED_BINDS -PROJECTION -PREDICATE -BYTES'));

PLAN_TABLE_OUTPUT
-----
SQL_ID 9w4xfcb47qfdn, child number 0
-----
SELECT e.last_name, e.first_name, d.department_name FROM hr.employees e,
hr.departments d WHERE e.department_id=d.department_id AND
d.department_id=:dept_id AND e.employee_id=:emp_id AND first_name=:fn
```

Plan hash value: 4225575861

Id	Operation	Name	Rows	Cost (%CPU)	Time
0	SELECT STATEMENT			2 (100)	
1	NESTED LOOPS		1	2 (0)	00:00:01
2	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	1	1 (0)	00:00:01
3	INDEX UNIQUE SCAN	DEPT_ID_PK	1	0 (0)	
4	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	1 (0)	00:00:01
5	INDEX UNIQUE SCAN	EMP_EMP_ID_PK	1	0 (0)	

Outline Data

```
/*+
  BEGIN_OUTLINE_DATA
  IGNORE_OPTIM_EMBEDDED_HINTS
  OPTIMIZER_FEATURES_ENABLE('10.2.0.3')
  ALL_ROWS
  OUTLINE_LEAF(@"SEL$1")
```

```

INDEX_RS_ASC(@"SEL$1" "D"@SEL$1" ("DEPARTMENTS"."DEPARTMENT_ID"))
INDEX_RS_ASC(@"SEL$1" "E"@SEL$1" ("EMPLOYEES"."EMPLOYEE_ID"))
LEADING(@"SEL$1" "D"@SEL$1" "E"@SEL$1")
USE_NL(@"SEL$1" "E"@SEL$1")
END_OUTLINE_DATA

*/

```

Peeked Binds (identified by position):

```

-----
1 - :DEPT_ID (NUMBER): 50
2 - :EMP_ID (NUMBER): 120
3 - :FN (VARCHAR2(30), CSID=178): 'Matthew'

```

当 DBMS_XPLAN.DISPLAY_CURSOR 前两个参数为 NULL 时, 前面 SQL 语句的 SQL_ID 标识符和 CHILD_NUMBER 子游标号默认会被使用。从前面的例子明显可以看出, 若想完全指定一个执行计划需要大量的提示。每当优化器没有给予单独的提示时, 原因可能是它必须采用自己的决策。

7.1 跟踪文件的内容

10053 跟踪文件是优化器输入、计算和输出的协议。正确的方法是使用事件级别 1, 毕竟更高的级别不代表能得到更多的输出。请注意, 跟踪文件内容和 CBO 使用的成本计算公式可能会在没有通知的情况下发生改变。Oracle10g R2 跟踪文件的主要部分如下所示:

- ❑ 优化语句相关表的查询块和对象标识符 (DBA_OBJECTS.OBJECT_ID)
- ❑ 考虑到的查询转换 (谓词 move-around、子嵌套查询, 等等)
- ❑ 说明 (使用过的缩写)
- ❑ 绑定变量窥视的结果
- ❑ 优化器参数 (在文档说明的和隐藏的参数)
- ❑ 系统统计信息 (工作量或非工作量)
- ❑ 表和索引的对象统计信息
- ❑ 每个表的单表访问路径和成本
- ❑ 联结顺序列表和成本
- ❑ 执行计划
- ❑ 谓词信息
- ❑ 包括查询块名称的提示全集, 可被用来定义存储概要

如果在启用 10053 事件后, 你没有在跟踪文件中找到上述内容, 可能是因为 CBO 使用了缓存执行计划, 而不是从头开始优化语句。你可以通过向语句插入注释来强制发生一次游标未命中 (cursor miss)。

7.2 案例研究

在随后的小节, 我们将为 HR 示例模式的表上的 5 路联结 (5-way join) 生成一个 10053 跟踪

文件。

```
SQL> VARIABLE loc VARCHAR2(30)
SQL> EXEC :loc:='South San Francisco'
SQL> ALTER SESSION SET EVENTS '10053 trace name context forever, level 1';
SQL> SELECT emp.last_name, emp.first_name, j.job_title, d.department_name, l.city,
l.state_province, l.postal_code, l.street_address, emp.email,
emp.phone_number, emp.hire_date, emp.salary, mgr.last_name
FROM hr.employees emp, hr.employees mgr, hr.departments d, hr.locations l, hr.jobs j
WHERE l.city=:loc
AND emp.manager_id=mgr.employee_id
AND emp.department_id=d.department_id
AND d.location_id=l.location_id
AND emp.job_id=j.job_id;
SQL> ALTER SESSION SET EVENTS '10053 trace name context off';
```

该案例学习中用到的查询来自源代码库中的 hr_5way_join.sql 文件，接下来的跟踪文件节选来自 Oracle10g R2。

7.2.1 查询块和对象标识符

本节列出了所有的查询块以及每个查询块中的表。因为执行的查询不包含任何子查询，所以只有一个单一的查询块。

```
Registered qb: SEL$1 0x4aea6b4 (PARSER)
signature (): qb_name=SEL$1 nbfros=5 flg=0
fro(0): flg=4 objn=51905 hint_alias="D"@SEL$1"
fro(1): flg=4 objn=51910 hint_alias="EMP"@SEL$1"
fro(2): flg=4 objn=51908 hint_alias="J"@SEL$1"
fro(3): flg=4 objn=51900 hint_alias="L"@SEL$1"
fro(4): flg=4 objn=51910 hint_alias="MGR"@SEL$1"
```

对象标识符 (objn) 可被用来标识表的所有者。

```
SQL> SELECT owner, object_name, object_type
FROM dba_objects WHERE object_id IN (51905, 51910);
OWNER OBJECT_NAME OBJECT_TYPE
-----
HR      DEPARTMENTS TABLE
HR      EMPLOYEES   TABLE
```

7.2.2 考虑的查询转换

优化器会考虑一些查询转换。要留意在未解析的查询小单元中的 SQL 标识符 2ck90x-fmsza4u，它可被包含 DBMS_XPLAN.DISPLAY_AWR 的打包程序或 Statspack 的语句用来获取过去的执行计划（见第 25 章）。

Predicate Move-Around (PM)

PM: Considering predicate move-around in SEL\$1 (#0).

PM: Checking validity of predicate move-around in SEL\$1 (#0).

CBQT: Validity checks failed for 2ck90xfmsza4u.

CVM: Considering view merge in query block SEL\$1 (#0)

Query block (04AEA6B4) before join elimination:

SQL:***** UNPARSED QUERY IS *****

```
SELECT "EMP"."LAST_NAME" "LAST_NAME", "EMP"."FIRST_NAME" "FIRST_NAME",
"J"."JOB_TITLE" "JOB_TITLE", "D"."DEPARTMENT_NAME" "DEPARTMENT_NAME",
"L"."CITY" "CITY", "L"."STATE_PROVINCE" "STATE_PROVINCE",
"L"."POSTAL_CODE" "POSTAL_CODE", "L"."STREET_ADDRESS" "STREET_ADDRESS",
"EMP"."EMAIL" "EMAIL", "EMP"."PHONE_NUMBER" "PHONE_NUMBER",
"EMP"."HIRE_DATE" "HIRE_DATE", "EMP"."SALARY" "SALARY",
"MGR"."LAST_NAME" "LAST_NAME"
FROM "HR"."EMPLOYEES" "EMP", "HR"."EMPLOYEES" "MGR",
"HR"."DEPARTMENTS" "D", "HR"."LOCATIONS" "L", "HR"."JOBS" "J"
WHERE "L"."CITY"=:B1 AND "EMP"."MANAGER_ID"="MGR"."EMPLOYEE_ID"
```

AND "EMP"."DEPARTMENT_ID"="D"."DEPARTMENT_ID"

AND "D"."LOCATION_ID"="L"."LOCATION_ID" AND "EMP"."JOB_ID"="J"."JOB_ID"

Query block (04AEA6B4) unchanged

CBQT: Validity checks failed for 2ck90xfmsza4u.

Subquery Unnest

SU: Considering subquery unnesting in query block SEL\$1 (#0)

Set-Join Conversion (SJC)

SJC: Considering set-join conversion in SEL\$1 (#0).

Predicate Move-Around (PM)

PM: Considering predicate move-around in SEL\$1 (#0).

PM: Checking validity of predicate move-around in SEL\$1 (#0).

PM: PM bypassed: Outer query contains no views.

FPD: Considering simple filter push in SEL\$1 (#0)

FPD: Current where clause predicates in SEL\$1 (#0) :

```
"L"."CITY"=:B1 AND "EMP"."MANAGER_ID"="MGR"."EMPLOYEE_ID"
AND "EMP"."DEPARTMENT_ID"="D"."DEPARTMENT_ID"
AND "D"."LOCATION_ID"="L"."LOCATION_ID"
AND "EMP"."JOB_ID"="J"."JOB_ID"
```

kkogcp: try to generate transitive predicate from check constraints for SEL\$1 (#0)

constraint: "MGR"."SALARY">0

constraint: "EMP"."SALARY">0

predicates with check constraints: "L"."CITY"=:B1

AND "EMP"."MANAGER_ID"="MGR"."EMPLOYEE_ID"

AND "EMP"."DEPARTMENT_ID"="D"."DEPARTMENT_ID"

AND "D"."LOCATION_ID"="L"."LOCATION_ID"

```

AND "EMP"."JOB_ID"="J"."JOB_ID" AND "MGR"."SALARY">0 AND "EMP"."SALARY">0
after transitive predicate generation:
"L"."CITY"=:B1 AND "EMP"."MANAGER_ID"="MGR"."EMPLOYEE_ID"
AND "EMP"."DEPARTMENT_ID"="D"."DEPARTMENT_ID"
AND "D"."LOCATION_ID"="L"."LOCATION_ID" AND "EMP"."JOB_ID"="J"."JOB_ID"
AND "MGR"."SALARY">0 AND "EMP"."SALARY">0
finally: "L"."CITY"=:B1 AND "EMP"."MANAGER_ID"="MGR"."EMPLOYEE_ID"
AND "EMP"."DEPARTMENT_ID"="D"."DEPARTMENT_ID"
AND "D"."LOCATION_ID"="L"."LOCATION_ID" AND "EMP"."JOB_ID"="J"."JOB_ID"
apadrvt-start: call(in-use=744, alloc=0), compile(in-use=47988, alloc=0)
kkoqbc-start
: call(in-use=756, alloc=0), compile(in-use=49312, alloc=0)

```

优化器从数据字典中获取检查约束，并用来生成更多的谓词（AND "MGR"."SALARY">0 AND "EMP"."SALARY">0）。

7.2.3 说明

该说明部分列出了跟踪文件中使用的缩写。

说明

优化器跟踪使用了如下缩写。

CBQT - cost-based query transformation
 JPPD - join predicate push-down
 FPD - filter push-down
 PM - predicate move-around
 CVM - complex view merging
 SPJ - select-project-join
 SJC - set join conversion
 SU - subquery unnesting
 OBYE - order by elimination
 ST - star transformation
 qb - query block
 LB - leaf blocks
 DK - distinct keys
 LB/K - average number of leaf blocks per key
 DB/K - average number of data blocks per key
 CLUF - clustering factor
 NDV - number of distinct values
 Resp - response cost
 Card - cardinality
 Resc - resource cost
 NL - nested loops (join)
 SM - sort merge (join)
 HA - hash (join)
 CPUSPEED - CPU Speed
 IOTFRSPEED - I/O transfer speed
 IOSEKTIM - I/O seek time
 SREADTIM - average single block read time
 MREADTIM - average multiblock read time


```

MBRC - average multiblock read count
MAXTHR - maximum I/O system throughput
SLAVETHR - average slave I/O throughput
dmeth - distribution method
    1: no partitioning required
    2: value partitioned
    4: right is random (round-robin)
    512: left is random (round-robin)
    8: broadcast right and partition left
    16: broadcast left and partition right
    32: partition left using partitioning of right
    64: partition right using partitioning of left
    128: use hash partitioning dimension
    256: use range partitioning dimension
    2048: use list partitioning dimension
    1024: run the join in serial
    0: invalid distribution method
sel - selectivity
ptn - partition

```

7.2.4 绑定变量窥视的结果

本节包含了查询中用到的所有绑定变量，以及它们的数据类型（oacdty）和值。该输出结果和在第 4 或 12 级别使用 10046 事件的扩展 SQL 跟踪格式相同。

```

*****
Peeked values of the binds in SQL statement
*****
kkscoacd
Bind#0
  oacdty=01 mxl=32(30) mxlc=00 mal=00 scl=00 pre=00
  oacflg=03 fl2=1000000 frm=01 csi=178 siz=32 off=0
  kxsbbfbfp=04c3ae00 bln=32 avl=19 flg=05
  value="South San Francisco"

```

7.2.5 优化器参数

本节列出了 184 个在文档中说明的和隐藏的参数，它们直接影响优化器的计算和决策，并被组织成三类：

- (1) 带可变值的参数；
- (2) 带默认值的参数；
- (3) 配有 OPT_PARAM 提示的参数，例如，OPT_PARAM ('optimizer_index_cost_adj'30)，这个文档中未说明的提示可能是存储概要的一部分。

来自该示例跟踪文件的第 1 类和第 3 类为空，因为所有的参数都使用默认值。

```

*****
PARAMETERS USED BY THE OPTIMIZER
*****

```

```

*****
PARAMETERS WITH ALTERED VALUES
*****
*****
PARAMETERS WITH DEFAULT VALUES
*****
optimizer_mode_hinted                = false
optimizer_features_hinted            = 0.0.0
parallel_execution_enabled           = true
parallel_query_forced_dop            = 0
parallel_dml_forced_dop              = 0
parallel_ddl_forced_degree           = 0
parallel_ddl_forced_instances        = 0
_query_rewrite_fudge                 = 90
optimizer_features_enable            = 10.2.0.1
_optimizer_search_limit              = 5
cpu_count                           = 2
active_instance_count                = 1
parallel_threads_per_cpu             = 2
hash_area_size                      = 131072
bitmap_merge_area_size              = 1048576
sort_area_size                      = 65536
sort_area_retained_size              = 0
_sort_elimination_cost_ratio         = 0
_optimizer_block_size               = 8192
_sort_multiblock_read_count          = 2
_hash_multiblock_io_count            = 0
_db_file_optimizer_read_count        = 16
_optimizer_max_permutations          = 2000
pga_aggregate_target                = 119808 KB
_pga_max_size                       = 204800 KB
_query_rewrite_maxdisjunct           = 257
_smm_auto_min_io_size               = 56 KB
_smm_auto_max_io_size               = 248 KB
_smm_min_size                       = 128 KB
_smm_max_size                       = 23961 KB
_smm_px_max_size                    = 59904 KB
_cpu_to_io                          = 0
_optimizer_undo_cost_change          = 10.2.0.1
parallel_query_mode                  = enabled
parallel_dml_mode                    = disabled
parallel_ddl_mode                    = enabled
optimizer_mode                      = all_rows
sqlstat_enabled                     = false
_optimizer_percent_parallel          = 101
_always_anti_join                   = choose
_always_semi_join                   = choose
_optimizer_mode_force               = true
_partition_view_enabled              = true
_always_star_transformation          = false

```

```

_query_rewrite_or_error      = false
_hash_join_enabled          = true
cursor_sharing               = exact
_b_tree_bitmap_plans        = true
star_transformation_enabled  = false
_optimizer_cost_model        = choose
_new_sort_cost_estimate      = true
_complex_view_merging        = true
_unnest_subquery             = true
_eliminate_common_subexpr    = true
_pred_move_around            = true
_convert_set_to_join         = false
_push_join_predicate         = true
_push_join_union_view        = true
_fast_full_scan_enabled      = true
_optim_enhance_nnull_detection = true
_parallel_broadcast_enabled   = true
_px_broadcast_fudge_factor    = 100
_ordered_nested_loop         = true
_no_or_expansion             = false
optimizer_index_cost_adj     = 100
optimizer_index_caching      = 0
_system_index_caching        = 0
_disable_datalayer_sampling  = false
query_rewrite_enabled        = true
query_rewrite_integrity      = enforced
_query_cost_rewrite          = true
_query_rewrite_2              = true
_query_rewrite_1              = true
_query_rewrite_expression    = true
_query_rewrite_jgmigrate     = true
_query_rewrite_fpc            = true
_query_rewrite_drj           = true
_full_pwise_join_enabled     = true
_partial_pwise_join_enabled  = true
_left_nested_loops_random    = true
_improved_row_length_enabled  = true
_index_join_enabled          = true
_enable_type_dep_selectivity  = true
_improved_outerjoin_card     = true
_optimizer_adjust_for_nulls   = true
_optimizer_degree            = 0
_use_column_stats_for_function = true
_subquery_pruning_enabled    = true
_subquery_pruning_mv_enabled = false
_or_expand_nvl_predicate     = true
_like_with_bind_as_equality  = false
_table_scan_cost_plus_one    = true
_cost_equality_semi_join     = true
_default_non_equality_sel_check = true

```

```
_new_initial_join_orders      = true
_oneside_colstat_for_equijoins = true
_optim_peek_user_binds        = true
_minimal_stats_aggregation     = true
_force_temptables_for_gsets    = false
workarea_size_policy          = auto
_smm_auto_cost_enabled         = true
_gs_anti_semi_join_allowed     = true
_optim_new_default_join_sel    = true
optimizer_dynamic_sampling     = 2
_pre_rewrite_push_pred         = true
_optimizer_new_join_card_computation = true
_union_rewrite_for_gs          = yes_gset_mvs
_generalized_pruning_enabled   = true
_optim_adjust_for_part_skews    = true
_force_datefold_trunc         = false
statistics_level               = typical
_optimizer_system_stats_usage  = true
skip_unusable_indexes          = true
_remove_aggr_subquery          = true
_optimizer_push_down_distinct  = 0
_dml_monitoring_enabled        = true
_optimizer_undo_changes        = false
_predicate_elimination_enabled = true
_nested_loop_fudge             = 100
_project_view_columns          = true
_local_communication_costing_enabled = true
_local_communication_ratio     = 50
_query_rewrite_vop_cleanup     = true
_slave_mapping_enabled         = true
_optimizer_cost_based_transformation = linear
_optimizer_mjc_enabled         = true
_right_outer_hash_enable       = true
_spr_push_pred_refspr         = true
_optimizer_cache_stats         = false
_optimizer_cbqt_factor         = 50
_optimizer_squ_bottomup        = true
_fic_area_size                 = 131072
_optimizer_skip_scan_enabled   = true
_optimizer_cost_filter_pred     = false
_optimizer_sortmerge_join_enabled = true
_optimizer_join_sel_sanity_check = true
_mmv_query_rewrite_enabled     = true
_bt_mmv_query_rewrite_enabled  = true
_add_stale_mv_to_dependency_list = true
_distinct_view_unnesting       = false
_optimizer_dim_subq_join_sel    = true
_optimizer_disable_strans_sanity_checks = 0
_optimizer_compute_index_stats  = true
_push_join_union_view2         = true
```

```

_optimizer_ignore_hints          = false
_optimizer_random_plan          = 0
_query_rewrite_setopgrw_enable  = true
_optimizer_correct_sq_selectivity = true
_disable_function_based_index   = false
_optimizer_join_order_control    = 3
_optimizer_cartesian_enabled     = true
_optimizer_starplan_enabled     = true
_extended_pruning_enabled       = true
_optimizer_push_pred_cost_based  = true
_sql_model_unfold_forloops      = run_time
_enable_dml_lock_escalation      = false
_bloom_filter_enabled           = true
_update_bji_ipdml_enabled       = 0
_optimizer_extended_cursor_sharing = udo
_dm_max_shared_pool_pct         = 1
_optimizer_cost_hjsmj_multimatch = true
_optimizer_transitivity_retain   = true
_px_pwg_enabled                 = true
optimizer_secure_view_merging   = true
_optimizer_join_elimination_enabled = true
_flashback_table_rpi            = non_fbt
_optimizer_cbqt_no_size_restriction = true
_optimizer_enhanced_filter_push  = true
_optimizer_filter_pred_pullup    = true
_rowsrc_trace_level             = 0
_simple_view_merging            = true
_optimizer_rownum_pred_based_fkr = true
_optimizer_better_inlist_costing = all
_optimizer_self_induced_cache_cost = false
_optimizer_min_cache_blocks      = 10
_optimizer_or_expansion          = depth
_optimizer_order_by_elimination_enabled = true
_optimizer_outer_to_anti_enabled = true
_selfjoin_mv_duplicates         = true
_dimension_skip_null            = true
_force_rewrite_enable           = false
_optimizer_star_tran_in_with_clause = true
_optimizer_complex_pred_selectivity = true
_gby_hash_aggregation_enabled   = true
*****

```

PARAMETERS IN OPT_PARAM_HINT

当在会话级别改变 DB_FILE_MULTIBLOCK_READ_COUNT 时，将通过可变值这个类别（第 1 类）中未在文档中说明的 DB_FILE_OPTIMIZER_READ_COUNT 参数反映出来。

PARAMETERS WITH ALTERED VALUES

```

_db_file_optimizer_read_count    = 64

```

下面的节选说明了使用第 3 类的 OPT_PARAM 提示, 在语句级别调整 OPTIMIZER_INDEX_COST_ADJ 的效果。

```
*****
PARAMETERS IN OPT_PARAM HINT
*****
optimizer_index_cost_adj          = 30
```

在这些参数中, 和示例中使用的 SELECT 语句相关的小部分参数可通过以下查询得到:

```
SQL> SELECT name FROM V$SQL_OPTIMIZER_ENV WHERE sql_id='2ck90xfmsza4u';
```

7.2.6 系统统计信息

在进行案例研究的系统上, 工作量统计信息使用以下的匿名 PL/SQL 块 (set_system_stats.sql 文件) 在数据字典中设置。

```
SQL> BEGIN
    dbms_stats.set_system_stats('sreadtim', 4);
    dbms_stats.set_system_stats('mreadtim', 10 );
    dbms_stats.set_system_stats('cpuspeed', 839);
    dbms_stats.set_system_stats('mbrc', 14);
    dbms_stats.set_system_stats('maxthr', 8 * 1048576);
END;
/
```

这在下面的 10053 跟踪信息中反映出来。

```
*****
SYSTEM STATISTICS INFORMATION
*****
Using WORKLOAD Stats
CPUSPEED: 839 millions instructions/sec
SREADTIM: 4 milliseconds
MREADTIM: 10 milliseconds
MBRC: 14.000000 blocks
MAXTHR: 8388608 bytes/sec
SLAVETHR: -1 bytes/sec
```

这些系统统计信息, 是使用 DBMS_STATS.GATHER_SYSTEM_STATS 对系统统计信息进行收集并对结果取平均值得到的。CPUSPEED 的值依赖于使用的硬件, 并随分配给 DBMS 的 CPU 时间上下浮动。我在本章的源代码库中加入了 gather_get_system_stats.sql 脚本, 该脚本收集系统在几个连续时间间隔的统计信息, 并存储在数据字典以外的统计表中。默认情况下, 它使用 4 个时间间隔, 每个间隔为 15 分钟。它还包含用来获得系统统计数据的 PL/SQL 代码。该脚本不会对任何优化器参数或决策产生影响。工作量统计信息必须导入数据字典, 从而影响 CBO 生成的执行计划。

当非工作量统计信息被使用时, 10053 系统统计信息跟踪文件的片段如下所示:

```
*****
SYSTEM STATISTICS INFORMATION
*****
Using NOWORKLOAD Stats
```

```
CPUSPEED: 485 millions instruction/sec
IOTFRSPEED: 4096 bytes per millisecond (default is 4096)
IOSEEKTIM: 10 milliseconds (default is 10)
```

CPUSPEED 的值依赖于所使用的硬件，然而 Oracle10g 的参数 IOTFRSPEED 和 IOSEEKTIM 不管在 UNIX 还是在 Windows 平台上都具有相同的值。

7.2.7 表和索引的对象统计信息

本小节包括来自 DBA_TABLES、DBA_TAB_COL_STATISTICS 和 DBA_INDEXES 的统计信息。如果语句访问分区对象，那么来自 DBA_TAB_PARTITIONS 和 DBA_IND_PARTITIONS 的统计信息将呈现。如果为某些列创建直方图，那么来自 DBA_TAB_HISTOGRAMS 的统计信息将显示出来。在接下来的节选里，LOCATION_ID 列有一个带有 7 个存储桶的直方图，而 DEPARTMENT_ID 列则没有直方图。只有那些用作过滤器或访问谓词的候选列被列出来。本节中使用的缩写以及它们的含义如表 7-2 所示。

```
*****
BASE STATISTICAL INFORMATION
*****
Table Stats::
  Table: JOBS Alias: J
    #Rows: 19  #Blks: 5  AvgRowLen: 33.00
  Column (#1): JOB_ID(VARCHAR2)
    AvgLen: 8.00 NDV: 19 Nulls: 0 Density: 0.052632
Index Stats::
  Index: JOB_ID_PK Col#: 1
    LVLS: 0  #LB: 1  #DK: 19  LB/K: 1.00  DB/K: 1.00  CLUF: 1.00
Table Stats::
  Table: DEPARTMENTS Alias: D
    #Rows: 27  #Blks: 5  AvgRowLen: 20.00
  Column (#1): DEPARTMENT_ID(NUMBER)
    AvgLen: 4.00 NDV: 27 Nulls: 0 Density: 0.037037 Min: 10 Max: 270
  Column (#4): LOCATION_ID(NUMBER)
    AvgLen: 3.00 NDV: 7 Nulls: 0 Density: 0.018519 Min: 1400 Max: 2700
  Histogram: Freq  #Bkts: 7  UncompBkts: 27  EndPtVals: 7
Index Stats::
  Index: DEPT_ID_PK Col#: 1
    LVLS: 0  #LB: 1  #DK: 27  LB/K: 1.00  DB/K: 1.00  CLUF: 1.00
  Index: DEPT_LOCATION_IX Col#: 4
    LVLS: 0  #LB: 1  #DK: 7  LB/K: 1.00  DB/K: 1.00  CLUF: 1.00
```

表7-2 在“基本统计信息”部分使用的缩写

缩 写	含 义	字典视图（和列）
#Bkts	直方图的存储桶个数	DBA_TAB_HISTOGRAMS
#Blks	低于高水位的表数据块个数	DBA_TABLES.BLOCKS
#DK	不同的键	DBA_INDEXES.DISTINCT_KEYS
#LB	索引叶块（leaf block）的数目	DBA_INDEXES.LEAF_BLOCKS

(续)

缩 写	含 义	字典视图 (和列)
#Rows	表行的数目	DBA_TABLES.NUM_ROWS
AvgLen	平均列长度	DBA_TAB_COLUMNS.AVG_COL_LEN
AvgRowLen	平均行长度	DBA_TABLES.AVG_ROW_LEN
EndPtVals	直方图末点的值	DBA_TAB_HISTOGRAMS
CLUF	群集因子	DBA_INDEXES.CLUSTERING_FACTOR
DB/K	每个键的索引数据块的平均数目	DBA_INDEXES.AVG_DATA_BLOCKS_PER_KEY
LB/K	每个键的索引叶块的平均数目	DBA_INDEXES.AVG_LEAF_BLOCKS_PER_KEY
LVLS	平衡树 (B-tree) 级别	DBA_INDEXES.BLEVEL
NDV	不同值的数目	DBA_TAB_COLUMNS.NUM_DISTINCT
UncompBkts	未压缩的存储桶	n/a

群集因子 (clustering factor) 是在一个表的段中集群 (有益的) 的或随机分布 (有害的) 的数据分布的指标。索引总是排好序的。当个别索引块指向许多表块时, 与一些索引的键相关的数据就会在表的段中随机分布, 从而导致集群因子较高, 影响索引的访问。

当没有直方图时, 用以下方法来计算密度:

$$\text{密度} = \frac{1}{\text{NDV}}$$

缺少的统计信息

只要 SQL 语句引用对象的其中一个有对象统计信息, CBO 将利用这些信息来计算最佳的执行计划。当缺少一个或多个表格、索引或分区的统计信息时, 计算将会出错。10053 跟踪文件会指出哪些对象缺少统计信息, 并包含 CBO 使用的默认值来代替使用 DBMS_STATS 计算出的实际值。

```
Table Stats::
  Table: LOCATIONS Alias: L (NOT ANALYZED)
    #Rows: 409 #Blks: 5 AvgRowLen: 100.00
  Column (#1): LOCATION_ID(NUMBER) NO STATISTICS (using defaults)
    AvgLen: 22.00 NDV: 13 Nulls: 0 Density: 0.07824
Index Stats::
  Index: LOC_CITY_IX Col#: 4 (NOT ANALYZED)
    LVLS: 1 #LB: 25 #DK: 100 LB/K: 1.00 DB/K: 1.00 CLUF: 800.00
```

7.2.8 单表访问路径和成本

优化器在单表访问路径部分会忽略联结条件, 唯一会考虑的是那些用常量或绑定变量赋值的访问谓词。当表中没有这样的谓词时, 只能考虑使用全表扫描或索引快速全表扫描。优化器估计将获得多少行, 并计算出为得到这些行成本最低的替代方案。在接下来的节选中, 优化器证明了使用 LOC_CITY_IX 索引比执行全表扫描的成本更低。

```
SINGLE TABLE ACCESS PATH
  Column (#4): CITY(VARCHAR2)
    AvgLen: 9.00 NDV: 23 Nulls: 0 Density: 0.043478
  Table: LOCATIONS Alias: L
```



```

Card: Original: 23 Rounded: 1 Computed: 1.00 Non Adjusted: 1.00
Access Path: TableScan
Cost: 3.01 Resp: 3.01 Degree: 0
Cost_io: 3.00 Cost_cpu: 41607
Resp_io: 3.00 Resp_cpu: 41607
Access Path: index (AllEqRange)
Index: LOC_CITY_IX
resc_io: 2.00 resc_cpu: 14673
ix_sel: 0.043478 ix_sel_with_filters: 0.043478
Cost: 2.00 Resp: 2.00 Degree: 1
Best:: AccessPath: IndexRange Index: LOC_CITY_IX
Cost: 2.00 Degree: 1 Resp: 2.00 Card: 1.00 Bytes: 0
*****

```

SINGLE TABLE ACCESS PATH

```

Table: JOBS Alias: J
Card: Original: 19 Rounded: 19 Computed: 19.00 Non Adjusted: 19.00
Access Path: TableScan
Cost: 3.01 Resp: 3.01 Degree: 0
Cost_io: 3.00 Cost_cpu: 38837
Resp_io: 3.00 Resp_cpu: 38837
Best:: AccessPath: TableScan
Cost: 3.01 Degree: 1 Resp: 3.01 Card: 19.00 Bytes: 0
*****

```

据 Jonathan Lewis[Lewi 2005]所说, CBO 用这个公式计算出了平衡树 (B-tree) 索引访问的成本。

$$\text{LVLS} + \text{ceiling}(\#LB * ix_sel) + \text{ceiling}(\text{CLUF} * ix_sel_with_filters)$$

ix_sel (索引选择性) 和 $ix_sel_with_filters$ (有效的表选择性) 的值能在跟踪文件中找到。LVLS (平衡树级别)、 $\#LB$ (叶块的数量) 以及 CLUF (集群因子) 在 7.2.7 节已介绍过。当该公式应用于 LOC_CITY_IX 索引时, 它能得到和优化器跟踪信息相同的结果。

$$0 + \text{ceiling}(1 * 0.043478) + \text{ceiling}(1 * 0.043478) = 2$$

该公式在应用于更复杂的案例时也能给出正确的结果。

动态采样

动态采样是从 Oracle9i 引入的一个 CBO 特性, 它是 CBO 在小样本的行的基础上计算统计信息的能力, 因为它能对查询进行优化。该特性由 OPTIMIZER_DYNAMIC_SAMPLING 参数控制, 在 Oracle9i R2 以及后续版本中默认启用。接下来的节选描述了表的对象统计信息被删除后, 优化器使用 SELECT 语句对表 LOCATIONS 动态采样。

```

** Generated dynamic sampling query:
query text :
SELECT /* OPT_DYN_SAMP */ /*+ ALL_ROWS opt_param('parallel_execution_enabled',
'false')
NO_PARALLEL(SAMPLESUB) NO_PARALLEL_INDEX(SAMPLESUB)
NO_SQL_TUNE */ NVL(SUM(C1),0), NVL(SUM(C2),0), NVL(SUM(C3),0)
FROM (SELECT /*+ NO_PARALLEL("L") INDEX("L" LOC_CITY_IX)
NO_PARALLEL_INDEX("L") */ 1 AS C1, 1 AS C2, 1 AS C3

```

```

FROM "LOCATIONS" "L" WHERE "L"."CITY"=:B1
AND ROWNUM <= 2500) SAMPLESUB
*** 2007-11-30 16:44:25.703
** Executed dynamic sampling query:
    level : 2
    sample pct. : 100.000000
    actual sample size : 23
    filtered sample card. : 1
filtered sample card. (index LOC_CITY_IX): 1
orig. card. : 23
block cnt. table stat. : 5
block cnt. for sampling: 5
max. sample block cnt. : 4294967295
sample block cnt. : 5
min. sel. est. : 0.01000000
index LOC_CITY_IX selectivity est.: 0.04347826

```

7.2.9 联结顺序

本节描述了经 CBO 详细检查的所有的联结顺序。对一个 n 路 (n -way) 联结, 有 $n!$ 种联结顺序。因此, 5路联结有120种联结顺序, 6路联结则有720种。对于那些更多种顺序的联结, 如果尝试所有可能的联结顺序将造成 CPU 资源的浪费。所以, CBO 不会考虑所有的联结顺序, 而会在成本超过目前最好的方案时停止检查联结顺序。

```

*****
OPTIMIZER STATISTICS AND COMPUTATIONS
*****
GENERAL PLANS
*****
Considering cardinality-based initial join order.
*****
Join order[1]:  LOCATIONS[L]#0  JOBS[J]#1  DEPARTMENTS[D]#2
EMPLOYEES[EMP]#3  EMPLOYEES[MGR]#4
*****
Now joining: JOBS[J]#1
*****
NL Join
  Outer table: Card: 1.00  Cost: 2.00  Resp: 2.00  Degree: 1  Bytes: 48
  Inner table: JOBS Alias: J
  Access Path: TableScan
    NL Join: Cost: 5.02  Resp: 5.02  Degree: 0
      Cost_io: 5.00  Cost_cpu: 53510
      Resp_io: 5.00  Resp_cpu: 53510
    Best NL cost: 5.02
      resc: 5.02 resc_io: 5.00 resc_cpu: 53510
      resp: 5.02 resp_io: 5.00 resp_cpu: 53510
Join Card: 19.00 = outer (1.00) * inner (19.00) * sel (1)
Join Card - Rounded: 19 Computed: 19.00
Best:: JoinMethod: NestedLoop

```

```

Cost: 5.02 Degree: 1 Resp: 5.02 Card: 19.00 Bytes: 75
...
*****
Now joining: DEPARTMENTS[D]#2
*****
...
*****
Now joining: EMPLOYEES[EMP]#3
*****
...
*****
Now joining: EMPLOYEES[MGR]#4
*****
...
*****
Best so far: Table#: 0 cost: 2.0044 card: 1.0000 bytes: 48
              Table#: 1 cost: 5.0159 card: 19.0000 bytes: 1425
              Table#: 2 cost: 6.0390 card: 73.2857 bytes: 6862
              Table#: 3 cost: 9.5672 card: 15.1429 bytes: 2400
              Table#: 4 cost: 11.6050 card: 15.0013 bytes: 2580
*****
Join order[2]: LOCATIONS[L]#0 JOBS[J]#1 DEPARTMENTS[D]#2
EMPLOYEES[MGR]#4 EMPLOYEES[EMP]#3
*****
Now joining: EMPLOYEES[MGR]#4
...
*****

```

对每个可能的联结顺序，优化器会对以下 3 种联结方法的成本进行评估。

- ☐ NL (Nested Loops, 嵌套循环) 联结
- ☐ SM (Sort-Merge, 排序合并) 联结
- ☐ HA (HAsH join, 散列) 联结

在每部分开头为 Now joining 的内容末尾，CBO 会报告最佳的联结方法（例如，Best:: JoinMethod: NestedLoop）。

```

Join order[8]: LOCATIONS[L]#0 DEPARTMENTS[D]#2 EMPLOYEES[EMP]#3
JOBS[J]#1 EMPLOYEES[MGR]#4
*****
Now joining: EMPLOYEES[EMP]#3
*****
NL Join
Outer table: Card: 3.86 Cost: 3.01 Resp: 3.01 Degree: 1 Bytes: 67
Inner table: EMPLOYEES Alias: EMP
Access Path: TableScan
NL Join: Cost: 8.09 Resp: 8.09 Degree: 0
Cost_io: 8.00 Cost_cpu: 316513
Resp_io: 8.00 Resp_cpu: 316513
Access Path: index (AllEqJoinGuess)
Index: EMP_DEPARTMENT_IX
resc_io: 1.00 resc_cpu: 13471

```

```

ix_sel: 0.091767 ix_sel_with_filters: 0.091767
NL Join: Cost: 4.65 Resp: 4.65 Degree: 1
  Cost_io: 4.63 Cost_cpu: 52993
  Resp_io: 4.63 Resp_cpu: 52993
Best NL cost: 4.65
  resc: 4.65 resc_io: 4.63 resc_cpu: 52993
  resp: 4.65 resp_io: 4.63 resp_cpu: 52993
Join Card: 15.14 = outer (3.86) * inner (107.00) * sel (0.036691)
Join Card - Rounded: 15 Computed: 15.14
SM Join
  Outer table:
    resc: 3.01 card 3.86 bytes: 67 deg: 1 resp: 3.01
  Inner table: EMPLOYEES Alias: EMP
    resc: 3.02 card: 107.00 bytes: 66 deg: 1 resp: 3.02
    using dmeth: 2 #groups: 1
  SORT resource      Sort statistics
    Sort width:      138 Area size:      131072 Max Area size:      24536064
    Degree:          1
    Blocks to Sort:   1 Row size:          84 Total Rows:          4
    Initial runs:     1 Merge passes:      0 IO Cost / pass:      0
    Total IO sort cost: 0      Total CPU sort cost: 3356360
    Total Temp space used: 0
  SORT resource      Sort statistics
    Sort width:      138 Area size:      131072 Max Area size:      24536064
    Degree:          1
    Blocks to Sort:   2 Row size:          83 Total Rows:          107
    Initial runs:     1 Merge passes:      0 IO Cost / pass:      0
    Total IO sort cost: 0      Total CPU sort cost: 3388500
    Total Temp space used: 0
SM join: Resc: 8.04 Resp: 8.04 [multiMatchCost=0.00]
SM cost: 8.04
  resc: 8.04 resc_io: 6.00 resc_cpu: 6842201
  resp: 8.04 resp_io: 6.00 resp_cpu: 6842201
HA Join
  Outer table:
    resc: 3.01 card 3.86 bytes: 67 deg: 1 resp: 3.01
  Inner table: EMPLOYEES Alias: EMP
    resc: 3.02 card: 107.00 bytes: 66 deg: 1 resp: 3.02
    using dmeth: 2 #groups: 1
    Cost per ptn: 0.50 #ptns: 1
    hash_area: 0 (max=0) Hash join: Resc: 6.53 Resp: 6.53 [multiMatchCost=0.00]
HA cost: 6.53
  resc: 6.53 resc_io: 6.00 resc_cpu: 1786642
  resp: 6.53 resp_io: 6.00 resp_cpu: 1786642
Best:: JoinMethod: NestedLoop
  Cost: 4.65 Degree: 1 Resp: 4.65 Card: 15.14 Bytes: 133
*****
Now joining: JOBS[J]#1
*****
NL Join

```

```

...
*****
Best so far: Table#: 0  cost: 2.0044  card: 1.0000  bytes: 48
              Table#: 2  cost: 3.0072  card: 3.8571  bytes: 268
              Table#: 3  cost: 4.6454  card: 15.1429  bytes: 1995
              Table#: 1  cost: 5.6827  card: 15.1429  bytes: 2400
              Table#: 4  cost: 7.7205  card: 15.0013  bytes: 2580
...
Join order[76]: EMPLOYEES[MGR]#4 EMPLOYEES[EMP]#3
DEPARTMENTS[D]#2 LOCATIONS[L]#0 JOBS[J]#1
*****
Now joining: DEPARTMENTS[D]#2
...
Join order aborted: cost > best plan cost
*****
(newjo-stop-1) k:0, spcnt:0, perm:76, maxperm:2000
Number of join permutations tried: 76
*****
(newjo-save) [1 2 4 0 3 ]
Final - All Rows Plan: Best join order: 8
      Cost: 7.7205 Degree: 1 Card: 15.0000 Bytes: 2580
      Resc: 7.7205 Resc_io: 7.6296 Resc_cpu: 305037
      Resp: 7.7205 Resp_io: 7.6296 Resc_cpu: 305037

```

在针对某个特定联结顺序的报表末尾，CBO 会打印出目前为止检测到的最佳联结顺序，以及它的成本、基数和数据容量（字节）。所有三个数字会在 Best so far 标题下累积显示，其原理和执行计划中行源（row source）的成本包括它依赖的行源成本一样。

使用 CBO 得出联结顺序 8 的成本最低，它按照 LOCATIONS、DEPARTMENTS、EMPLOYEES（别名 EMP）、JOBS 和 EMPLOYEES（别名 MGR）的顺序联结表。该联结顺序可通过下节要介绍的执行计划和 LEADING 提示反映出来。

7.2.10 执行计划

本节包括一个格式良好的执行计划。除了 DBMS_XPLAN.DISPLAY_CURSOR、V\$SQL_PLAN、AWR 和 Statspack 外，10053 跟踪文件是执行计划另一个可靠的来源。记住 EXPLAIN PLAN 是出了名的不可靠，绝不能在优化工程中使用。Oracle 公司让 DBA 警惕“由于通常带有绑定变量，所以 EXPLAIN PLAN 的输出结果不能代表真正的执行计划”（*Oracle Database Performance Tuning Guide 10g Release 2*, 19-4 页）。最好检查一下计划表中的基数（Rows 列），当它们非常不正确时将不会得到满意的结果。

```

=====
Plan Table
=====

```

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT				8	
1	NESTED LOOPS		15	2580	8	00:00:01

2	NESTED LOOPS		15	2400	6 00:00:01
3	NESTED LOOPS		15	1995	5 00:00:01
4	NESTED LOOPS		4	268	3 00:00:01
5	TABLE ACCESS BY INDEX ROWID	LOCATIONS	1	48	2 00:00:01
6	INDEX RANGE SCAN	LOC_CITY_IX	1		1 00:00:01
7	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	4	76	1 00:00:01
8	INDEX RANGE SCAN	DEPT_LOCATION_IX	4		0
9	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	4	264	1 00:00:01
10	INDEX RANGE SCAN	EMP_DEPARTMENT_IX	10		0
11	TABLE ACCESS BY INDEX ROWID	JOBS	1	27	1 00:00:01
12	INDEX UNIQUE SCAN	JOB_ID_PK	1		0
13	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	12	1 00:00:01
14	INDEX UNIQUE SCAN	EMP_EMP_ID_PK	1		0

7.2.11 谓词信息

谓词和计划部分的输出结果和在你调查的语句之后立即运行 SELECT*FROM table (DBMS_XPLAN.DISPLAY_CURSOR())的输出结果基本相同。

Predicate Information:

```

-----
6 - access("L"."CITY"=:LOC)
8 - access("D"."LOCATION_ID"="L"."LOCATION_ID")
10 - access("EMP"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
12 - access("EMP"."JOB_ID"="J"."JOB_ID")
14 - access("EMP"."MANAGER_ID"="MGR"."EMPLOYEE_ID")

```

7.2.12 提示和查询块名称

本节由包含查询块名称的全套提示构成。这些提示可被用来定义存储概要，从而对 CBO 选择的计划进行修复。这些数据通过提示的正确语法来显示。

```

Outline Data:
/*+
BEGIN_OUTLINE_DATA
IGNORE_OPTIM_EMBEDDED_HINTS
OPTIMIZER_FEATURES_ENABLE('10.2.0.1')
ALL_ROWS
OUTLINE_LEAF(@"SEL$1")
INDEX(@"SEL$1" "L"@"SEL$1" ("LOCATIONS"."CITY"))
INDEX(@"SEL$1" "D"@"SEL$1" ("DEPARTMENTS"."LOCATION_ID"))
INDEX(@"SEL$1" "EMP"@"SEL$1" ("EMPLOYEES"."DEPARTMENT_ID"))
INDEX(@"SEL$1" "J"@"SEL$1" ("JOBS"."JOB_ID"))
INDEX(@"SEL$1" "MGR"@"SEL$1" ("EMPLOYEES"."EMPLOYEE_ID"))
LEADING(@"SEL$1" "L"@"SEL$1" "D"@"SEL$1" "EMP"@"SEL$1" "J"@"SEL$1" "MGR"@"SEL$1")
USE_NL(@"SEL$1" "D"@"SEL$1")
USE_NL(@"SEL$1" "EMP"@"SEL$1")
USE_NL(@"SEL$1" "J"@"SEL$1")
USE_NL(@"SEL$1" "MGR"@"SEL$1")

```

```
END_OUTLINE_DATA
*/
```

7.3 源代码库

表 7-3 列出了本章的源文件以及它们的功能。

表7-3 10053事件源代码库

文 件 名	功 能
hr_5way_join.sql	HR示例模式中表的5路联结
set_system_stats.sql	该脚本用于在数据字典里设置工作量系统统计信息
gather_get_system_stats.sql	该脚本用来收集几个连续间隔的系统统计信息（默认为4个间隔，每个间隔15分钟），并把这些信息存储在数据字典之外的统计信息表中
hr.dmp	传统的导出转储文件，包括来自HR示例模式的表（从Oracle10g R2开始创建）

10079 事件和 Oracle 网络数据包的内容

未在文档中说明的 10079 事件可用来将 Oracle 网络流量转储到跟踪文件。它可用于快速确定哪些 SQL 语句、PL/SQL 调用或 SQL * Plus 命令发送了敏感数据，比如发送未加密的密码。

10079 事件类似于 Oracle 网络跟踪，因为它会转储数据库客户端和服务端之间的网络数据包的完整内容。这比通过更改 sqlnet.ora 来启动对 Oracle 网络数据包内容的转储更加方便。和 sqlnet.ora 中的 trace_level_client 不一样，它也可以用来对已经建立的数据库会话启动数据包转储。表 8-1 列出了支持的事件级别。

表8-1 事件10079支持的级别

级 别	用 途
1	跟踪来自或发往客户端的网络操作
2	除了进行级别1的操作外，还能转储数据
4	跟踪来自或发往数据库链接的网络操作
8	除了进行级别4的操作外，还能转储数据

案例研究

在随后的内容中，我们假定 Oracle 网络流量加密中没有使用高级安全选项。关于发送用 PASSWORD 这个 SQL*Plus 命令修改的密码时有没有加密这个问题，*SQL *Plus User's Guide and Reference* 并没有提到，这时候可使用 10079 事件查明。

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> ALTER SESSION SET EVENTS '10079 trace name context forever, level 2';
Session altered.
SQL> PASSWORD ndebes
Changing password for ndebes
New password:
Retype new password:
Password changed
```



```
SQL> ORADEBUG SETMYPID
Statement processed.
SQL> ORADEBUG TRACEFILE_NAME
/opt/oracle/obase/admin/TEN/udump/ten1_ora_20364.trc
```

生成的跟踪文件包括以下的转储数据包：

```
C850BD0 FFD668BF 646E06BF 73656265 00000030 [.h....ndebes0...]
C850BE0 54554110 454E5F48 53415057 524F5753 [.AUTH_NEWPASSWOR]
C850BF0 0000C044 31384000 39314642 38373930 [D....@81BF190978]
C850C00 41323232 39363642 45453539 42303242 [222AB66995EEB20B]
C850C10 46323546 30324343 30313239 39453434 [F52FCC20921044E9]
C850C20 34423130 32353232 45423332 44393431 [01B42252238E149D]
C850C30 30304542 00003245 00270000 410D0000 [BE00E2....'....A]
C850C40 5F485455 53534150 44524F57 00000000 [UTH_PASSWORD....]
```

显然，密码是在加密后发送的。因此，SQL*Plus 的 PASSWORD 命令是一种安全地更改密码的方法，而 ALTER USER user_name IDENTIFIED BY new_password 命令却不安全，因为它将未加密的密码与 SQL 语句文本一起发送。顺便说一下，前面的加密和 DBA_USERS.PASSWORD 中的密码散列值不一样，所以即使通信连接被窃听也不能用来获取存储在字典基表 USER\$ 中的密码散列值。OCI (Oracle Call Interface, Oracle 调用接口) 提供了 OCIPasswordChange() 函数，如果 SQL*Plus 使用这个函数来实现 PASSWORD 命令也是安全的。不幸的是，手册中并未说明 OCIPasswordChange() 这个函数是否对密码进行加密。

某些应用程序使用密码保护的角​​色，使得仅当用户与该应用程序相连时才启用某些特权。这主要是为了限制用户与 SQL*Plus 或其他应用程序连接时的特权。事件 10079 可被用来证明，无论是 SQL 语句 SET ROLE role_name IDENTIFIED BY password 还是 DBMS_SESSION.SET_ROLE 都是把角色未加密的密码发送到 DBMS 服务器上。这意味着任何用户，只要对 Oracle 网络足够了解，就可以从转储的数据包里获得未加密的角色密码。由于最终用户不能把 ALTER SESSION 语句添加到应用程序中，因此，需要一种 Oracle 网络数据包转储的替代方案。你仅仅需要把 tnsnames.ora 和 sqlnet.ora 复制到用户的根目录下，并且设置 TNS_ADMIN 指向同一目录。然后，把以下两行添加进 sqlnet.ora：

```
trace_level_client=support
trace_directory_client=<user's home directory>
```

重启应用程序，明文密码就可以从跟踪文件^①里获取了。

```
[28-NOV-2007 23:10:54:156] nspsend: 00 2E 42 45 47 49 4E 20 |..BEGIN.|
[28-NOV-2007 23:10:54:156] nspsend: 64 62 6D 73 5F 73 65 73 |dbms_ses|
[28-NOV-2007 23:10:54:156] nspsend: 73 69 6F 6E 2E 73 65 74 |sion.set|
[28-NOV-2007 23:10:54:156] nspsend: 5F 72 6F 6C 65 28 3A 72 |_role(:r|
[28-NOV-2007 23:10:54:156] nspsend: 6F 6C 65 5F 63 6D 64 29 |ole_cmd)|
[28-NOV-2007 23:10:54:156] nspsend: 3B 20 45 4E 44 3B 0A 00 |;.END;..|
[28-NOV-2007 23:10:54:156] nspsend: 01 00 00 00 01 00 00 00 |.....|
[28-NOV-2007 23:10:54:156] nspsend: 00 00 00 00 00 00 00 00 |.....|
```

① Oracle 网络跟踪文件的命名约定在 cli_spid.trc 中，其中 spid 是客户端的进程标识符。

```

[28-NOV-2007 23:10:54:156] nspsend: 00 00 00 00 00 00 00 00 |.....|
[28-NOV-2007 23:10:54:156] nspsend: 00 00 00 00 08 00 00 00 |.....|
[28-NOV-2007 23:10:54:156] nspsend: 00 00 00 00 00 00 00 00 |.....|
[28-NOV-2007 23:10:54:156] nspsend: 00 00 00 00 00 00 00 00 |.....|
[28-NOV-2007 23:10:54:156] nspsend: 00 00 00 00 01 01 03 00 |.....|
[28-NOV-2007 23:10:54:156] nspsend: 00 32 00 00 00 00 00 00 |.2.....|
[28-NOV-2007 23:10:54:156] nspsend: 00 00 00 00 00 00 00 00 |.....|
[28-NOV-2007 23:10:54:156] nspsend: 00 00 00 00 00 82 00 01 |.....|
[28-NOV-2007 23:10:54:156] nspsend: 00 00 00 00 00 07 1F 61 |.....a|
[28-NOV-2007 23:10:54:156] nspsend: 70 70 72 6F 6C 65 20 69 |pprole.i|
[28-NOV-2007 23:10:54:156] nspsend: 64 65 6E 74 69 66 69 65 |dentifie|
[28-NOV-2007 23:10:54:156] nspsend: 64 20 62 79 20 74 6F 70 |d.by.top|
[28-NOV-2007 23:10:54:156] nspsend: 73 65 63 72 65 74      |secret  |

```

结合 Oracle 互联网目录和安全的应用程序角色来使用代理身份验证是执行特权的安全方式，但它仅仅在与应用程序连接时有效。

当然，同样的弱点在使用 CREATE USER user_name IDENTIFIED BY password 语句时也存在。该语句发送的也是明文密码。

```

[08-SEP-2007 09:28:23:864] nspsend: 08 23 43 52 45 41 54 45 |.#CREATE|
[08-SEP-2007 09:28:23:864] nspsend: 20 55 53 45 52 20 68 72 |.USER.hr|
[08-SEP-2007 09:28:23:864] nspsend: 20 49 44 45 4E 54 49 46 |.IDENTIF|
[08-SEP-2007 09:28:23:864] nspsend: 49 45 44 20 42 59 20 73 |IED.BY.s|
[08-SEP-2007 09:28:23:864] nspsend: 65 63 72 65 74 01 00 00 |ecret...|

```

因此，你应该创建外部标识的用户，然后使用 SQL * Plus 命令 PASSWORD 来更改密码。

Part 4

第四部分

X\$ 固定表

本 部 分 内 容

- 第 9 章 X\$ 固定表介绍
- 第 10 章 X\$BH 表及锁争用
- 第 11 章 X\$KSLED 以及增强的会话等待数据
- 第 12 章 X\$KFFXP 与 ASM 元数据

Oracle 官方文档对一些 X\$表有所提及，但是绝大多数在文档中是查不到的。比如 X\$BH 就在 *Oracle9i Performance Tuning Guide* 和 *Oracle10g Performance Tuning Guide* 中提及，*Oracle10g Warehouse Builder Installation and Administration Guide* 中就包含了通过访问 X\$KGLLK^①表来解决涉及库缓存锁问题的步骤。

很多X\$表相比于构建在其之上的GV\$视图容纳了更多的信息。一个例子就是GV\$或V\$视图提供的信息就是不充分的，要详查其底层的X\$表。有很多X\$表是不作为GV\$视图的基表的。

9.1 X\$固定表与 C 语言编程

Oracle 数据库管理系统的内核代码，即使不是全部，至少重要部分是用 C 语言编写的。许多数据都以二维数组的形式保存。在 UNIX 系统上，位于 SGA 的 C 语言数组可以被商用的第三方性能诊断工具读取，这些工具都附加持有 SGA[®]的一个或多个共享内存段。就个人而言，虽然我对那些针对 SGA 内部数据结构进行逆向工程的开发者怀有崇敬之情，但我并不提倡此类的 SGA 取样工具。毕竟它们只是取样工具，并可能遗漏相关数据。在我看来，扩展 SQL 跟踪数据并且结合 Statspack 快照（或者 AWR）就能提供足够的数据库性能诊断数据，而并非通过取样来收集。

V\$视图背后的基本思路是使数据库管理员了解到C语言数据结构。这是通过某些中间层将V\$视图映射到C语言数据结构实现的。X\$表是一个中间层，确切地说，它们是最接近C语言的层。当然，在X\$表里table的意思与在SQL上下文里几乎是完全不同的。更不用说X\$表在DBA_SEGMENTS里面没有一个与之相关的段。访问它们的行源是FIXED TABLE FULL这个事实，更进一步证明了X\$表的独特性。为了加快访问速度，在X\$表中维护了索引。对于一个固定表，与索引有关的行数据源访问就是所谓的FIXED TABLE FIXED INDEX。本着同样的精神，DBA_VIEWS中V\$视图没有任何元数据。相反地，可以通过DBA_VIEWS查看到V_\$视图的元数据，V_\$视图可用来授权对V\$视图的访问。

一些 X\$表是与磁盘存储相关的。例如，X\$固定表 X\$KCCDC 的列 DCNAM，保存了通过 RMAN

① 用户手册中展示了怎样通过访问 X\$KGLLK 解决错误 ORA-04021 timeout occurred while waiting to lock object。

② 参看 Richmond Shee 的 *Oracle Wait Interface: A Practical Guide to Performance Diagnostics and Tuning*。

命令 COPY DATAFILE 创建的数据文件副本的路径名。这些路径名的磁盘存储位于控制文件中。你可以通过查看 V\$CONTROLFILE_RECORD_SECTION 视图或者该视图的底层 X\$KCCRS 表, 找到一个名为 DATAFILE COPY 的部分。这是控制文件在 X\$KCCDC (KCCDC 全称 Kernel Cache Control file Data file Copy) 里的直观展现。

关于 Oracle 数据库管理系统, 有句老话是这么说的: 任何设计优良的特性都有一个以上的命名方式。V\$ 动态性能视图与 V\$ 固定视图这一对同义词组就是一个典型例子。另外一个例子就是 AUM (Automatic Undo Management, 自动撤销管理), 它也被称为 SMU (System Managed Undo, 系统管理撤销) 或者 SCN (System Change Number, 系统修改号), SCN 有时也称为系统托管号 (System Commit Number)。

9.2 分层系统结构

X\$ 固定表是通过如图 9-1 所示的分层方式进行访问。对每个 V\$ 或者 GV\$ 固定视图, 公共同义词都以相同的名字存在。这些与 V_\$ 和 GV_\$ 视图有关的同义词属于 SYS 用户, 并且这些视图在 DBA_VIEWS 中是带有元数据的真实视图, 它们的 DDL (Data Description Language, 数据描述语言) 见文件 catalog.sql。形式如下:

```
CREATE OR REPLACE VIEW {g|v}$_<view_name> AS
SELECT * FROM {g|v$}<fixed_view_name>;
```

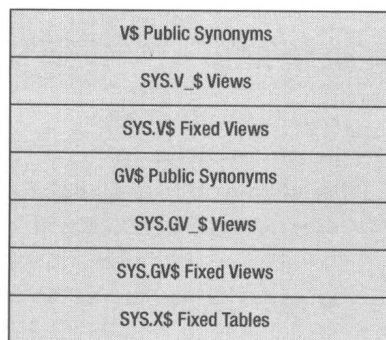


图 9-1 V\$ 固定视图、GV\$ 固定视图以及 X\$ 固定表的层次结构

在 V_\$ 和 GV_\$ 视图上的 SELECT 操作被授予了 SELECT_CATALOG_ROLE 权限。所有 V\$ 的动态性能视图所持有的信息都与当前数据库实例相关, 即用户连接到的实例。V\$ 固定视图基于 GV\$ 固定视图, 同时 GV\$ 是由一个或多个 X\$ 固定表所构成的。在运行 RAC (Real Application Clusters, 实时应用集群) 的系统上, GV\$ 固定视图还提供对挂载在相同数据库上其他实例信息的访问功能。C 语言中数组从 0 开始偏移, 而在 SQL 中最小的索引从 1 开始, 典型的例子就是虚拟数据行的 ROWNUM 值从 1 开始。这就是为什么你看到在定义 GV\$ 视图时会在 X\$ 表的一些列里面加 1 了。

请注意, V\$OBJECT_USAGE 是唯一一个在 DBA_VIEWS 中不基于 GV\$ 视图, 并且还带有元数据的真实视图。相反, 它从 SYS 用户拥有的表空间 SYSTEM 中, 通过数据字典表检索信息。因为它

违反了前面提到的 V\$视图的规则，所以它不会有前缀 V\$。很明显，任何一位配备了 Oracle 客户服务号的专业人士都不会因为怀疑一个无所谓的前缀而开启服务请求。

许多 X\$表名遵循严格的命名规则，其中开始的几个字母代表了 ORACLE 内核中的一个层或者模块。例如 KC 就表示内核缓存（Kernel Cache），KT 表示内核事物（Kernel Transaction）。表 9-1 为 X\$表所用的缩写词及其对应的推测意思。

表9-1 X\$固定表名用到的缩写

缩 写	推测意思
K	Kernel（内核）
KC	Kernel Cache（内核缓存）
KCB	Kernel Cache Buffer（内核缓存缓冲区）
KCBW	Kernel Cache Buffer Wait（内核缓存缓冲区等待）
KCC	Kernel Cache Control file（内核缓存控制文件）
KCCB	Kernel Cache Control file Backup（内核缓存控制文件备份）
KCCCF	Kernel Cache Copy Flash recovery area（内核缓存副本闪回恢复区）
KCCDC	Kernel Cache Control file Data file Copy（内核缓存控制文件数据文件副本）
KCP	Kernel Cache transPortable tablespace（内核缓存可移植表空间）
KCR	Kernel Cache Redo（内核缓存重写）
KCT	Kernel Cache insTance（内核缓存实例）
KG	Kernel Generic（通用内核）
KGL	Kernel Generic Library cache（通用内核库高速缓存）
KGLJ	Kernel Generic Library cache Java（Java通用内核库高速缓存）
KS	Kernel Service（内核服务）
KSB	Kernel Service Background（内核服务配置）
KSM	Kernel Service Memory（内核服务存储）
KSU	Kernel Service User（内核服务用户）
KSUSE	Kernel Service User SEssion（内核服务用户会话）
KSUSECON	Kernel Service User SEssion COnnection（内核服务用户会话连接）
KSUSEH	Kernel Service User SEssion History（内核服务用户会话历史）
KT	Kernel Transaction（内核事务）
KTU	Kernel Transaction Undo（内核事务撤销）
KX	Kernel eXecution（内核执行）
KXS	Kernel eXecution Sql（Sql内核执行）

9.3 授权访问 X\$表与 V\$视图

对于除 SYS 之外的用户，SELECT_CATALOG_ROLE 这个角色的权限已经足够用来访问位于 SQL 语句和匿名数据块的 V\$视图。由于在内部存储的 PL/SQL 程序中，比如包中，角色是被禁用的，想要通过 PL/SQL 访问 V\$或者 GV\$视图的用户，必须从相应的 V_\$或者 GV_\$视图直

接获得 SELECT 操作授权，而不是间接地通过角色进行访问。另外一种选择就是授予系统特权 SELECT ANY DICTIONARY。后一种选择在使用的时候需要注意，因为在 Oracle9i 中它会授权访问位于 SYS.LINK\$中未加密的密码数据。如果安全是需要考虑的一个重点，即使是给予角色 SELECT_CATALOG_ROLE 访问 DBA_USERS 中的密码散列值也不是你所希望的，因为入侵者会尝试去破解这些散列值。

与 V\$固定视图以及与之相对应的 V_\$视图相反，X\$固定表并没有相对应的 X_\$视图。因此，没有什么快速的方法授予除 SYS 之外其他用户去访问 X\$表。首选的方法是通过模拟 V\$视图所采用的方式去实现这种访问——在 X\$表上创建属于 SYS 用户的真实视图然后授予它们相应的访问权限。按照命名约定，这些视图应该使用 X_\$作为前缀。除去前缀之外，这正是在 Statspack 的实现中所采用的方式。Statspack 安装脚本会创建 3 个这样的视图。它们的命名和目的在表 9-2 中进行了总结。对这些视图的访问仅被授予了 PERFSTAT 用户，但是也可以依次授予其他用户。

表9-2 基于X\$固定表的Statspack视图

公共同义词	视 图	X\$基表	相关V\$视图
STATS\$X\$KSPPI V\$PARAMETER2 V\$SGA_CURRENT_RESIZE_OPS V\$SGA_RESIZE_OPS V\$SYSTEM_PARAMETER2 V\$SYSTEM_PARAMETER	STATS\$X_\$KSPPI	X\$KSPPI（参数名以及描述）	V\$PARAMETER
STATS\$X\$KSPPSV V\$SYSTEM_PARAMETER2	STATS\$X_\$KSPPSV	X\$KSPPSV（系统级参数值）	V\$SYSTEM_PARAMETER
STATS\$X\$KCBFWAIT	STATS\$X_\$KCBFWAIT	X\$KCBFWAIT（等待时间以及在 数据文件级别等待的数量）	n/a

9.4 从 V\$视图深入剖析 X\$固定表

接下来的几节介绍了一种从 V\$视图（通过它底层的 GV\$视图）一直到最底层的一个或多个 X\$表的逐层分析方法。这种方法适用于当排除故障或性能诊断工作时，V\$视图展示的信息是一个限制性因素的情况。尽管 Oracle 公司在最近的几个发行版本中显示 X\$表中越来越多的信息，但有些时候还有必要从 X\$表中搜集更多的信息。当然了，因为 X\$表实际上在文档中并没有进行说明，对它们作出的修改是不会预先通知的。

互联网上的一些文章赞同这一观点：对于 X\$表，在很多情况下都可以从 V\$视图中获取同样有用的信息。举个例子，在 Oracle9i 中引入的新特性，用户可以通过访问 V\$ SEGMENT_STATISTICS 检测争用和热点数据块，而不是通过访问 X\$BH 获得。Statspack 的 7 级或更高级别的报告就包含了从 V\$SEGMENT_STATISTICS 中获取的数据。就个人而言，我从来没有从 X\$表获取信息来解决性能问题。解决挂起问题是另一回事，因为在文档中有详细说明的视图 DBA_BLOCKERS 并不考虑库

缓存引脚。在这种情况下，了解 X\$KGLLK 是非常有优势的。

从 V\$PARAMETER 逐层剖析底层 X\$表

最为人所熟知的 X\$表应该就是 X\$KSPPI 和 X\$KSPPCV。本节将通过从 V\$PARAMETER 进行深入分析，说明如何去揭示 X\$KSPPI 和 X\$KSPPCV。每一个 DBA 都很有可能听说过、用过在文档中未说明的参数（或隐藏的参数）。在文档中未作说明的参数以下划线 “_” 开头。但是在 V\$PARAMETER 视图中并没有发现这一类参数。在 Oracle10g 中，你可能会注意到在实例的服务器参数文件中，双下划线参数与启动自动共享内存管理有关（比如 __db_cache_size）。V\$PARAMETER 视图中的列名以及相关类型如下：

```
SQL> DESCRIBE v$parameter
```

Name	Null?	Type
NUM		NUMBER
NAME		VARCHAR2(80)
TYPE		NUMBER
VALUE		VARCHAR2(512)
DISPLAY_VALUE		VARCHAR2(512)
ISDEFAULT		VARCHAR2(9)
ISSES_MODIFIABLE		VARCHAR2(5)
ISSYS_MODIFIABLE		VARCHAR2(9)
ISINSTANCE_MODIFIABLE		VARCHAR2(5)
ISMODIFIED		VARCHAR2(10)
ISADJUSTED		VARCHAR2(5)
ISDEPRECATED		VARCHAR2(5)
DESCRIPTION		VARCHAR2(255)
UPDATE_COMMENT		VARCHAR2(255)
HASH		NUMBER

马上，你将了解到如何为没有在文档中说明的参数生成带有默认值以及相关描述信息的列表。我们将以已在文档中作出了说明的 VS 视图 V\$FIXED_VIEW_DEFINITION 作为出发点。这个 VS 视图是一个实例中所有 VS、GV\$视图以及 X\$表的信息库。探究视图 V\$PARAMETER 得到如下信息：

```
SQL> COLUMN view_definition FORMAT a80 WORD_WRAPPED
SQL> SELECT view_definition FROM v$fixed_view_definition
WHERE view_name='V$PARAMETER';
VIEW_DEFINITION
-----
select NUM , NAME , TYPE , VALUE , DISPLAY_VALUE , ISDEFAULT ,
ISSES_MODIFIABLE, ISSYS_MODIFIABLE , ISINSTANCE_MODIFIABLE, ISMODIFIED,
ISADJUSTED, ISDEPRECATED, DESCRIPTION, UPDATE_COMMENT, HASH
from GV$PARAMETER where inst_id = USERENV('Instance')
```

我们可以知道，V\$PARAMETER 视图是基于 GV\$PARAMETER 视图的，并且前者通过过滤来自其他实例的行数据，删除了在后者中发现的跨实例信息。这里没有什么奇特之处，除了附加的列 INST_ID 之外。GV\$PARAMETER 与 V\$PARAMETER 具有相同的结构。


```
SQL> DESC gv$parameter
```

Name	Null?	Type
INST_ID		NUMBER
NUM		NUMBER
NAME		VARCHAR2(80)
TYPE		NUMBER
VALUE		VARCHAR2(512)
DISPLAY_VALUE		VARCHAR2(512)
ISDEFAULT		VARCHAR2(9)
ISSES_MODIFIABLE		VARCHAR2(5)
ISSYS_MODIFIABLE		VARCHAR2(9)
ISINSTANCE_MODIFIABLE		VARCHAR2(5)
ISMODIFIED		VARCHAR2(10)
ISADJUSTED		VARCHAR2(5)
ISDEPRECATED		VARCHAR2(5)
DESCRIPTION		VARCHAR2(255)
UPDATE_COMMENT		VARCHAR2(255)
HASH		NUMBER

进一步挖掘 V\$FIXED_VIEW_DEFINITION 获得如下信息:

```
SQL> SELECT view_definition FROM v$fixed_view_definition
WHERE view_name='GV$PARAMETER';
VIEW_DEFINITION
```

```
-----
select x.inst_id,x.indx+1,ksppinm,ksppity,ksppstvl, ksppstdvl, ksppstdf,
decode(bitand(ksppiflg/256,1),1,'TRUE','FALSE'),
decode(bitand(ksppiflg/65536,3),1,'IMMEDIATE',2,'DEFERRED',
3,'IMMEDIATE','FALSE'), decode(bitand(ksppiflg,4),4,'FALSE',
decode(bitand(ksppiflg/65536,3), 0, 'FALSE', 'TRUE')),
decode(bitand(ksppstvf,7),1,'MODIFIED',4,'SYSTEM_MOD','FALSE'),
decode(bitand(ksppstvf,2),2,'TRUE','FALSE'),
decode(bitand(ksppilrmflg/64, 1), 1, 'TRUE', 'FALSE'), ksppdesc,
ksppstcmt, ksppihash
from x$ksppi x, x$ksppcv y where (x.indx = y.indx) and
((translate(ksppinm,'_','#') not like '##%') and
((translate(ksppinm,'_','#') not like '%#') or (ksppstdf = 'FALSE')
or (bitand(ksppstvf,5) > 0)))
```

为大家所熟知的 X\$表 X\$KSPPI 以及 X\$KSPPCV 在这里就出现了。仔细观察 where 子句, 很明显可以发现带有一个或两个下划线的参数就藏匿于 DBA 们充满求知欲的双眼之下。

下一步, 我们需要理解隐蔽列的名称。因为我们都知道 GV\$PARAMETER 视图的列序列, 通过从上到下遍历视图的 select 列表, 我们可以将 GV\$PARAMETER 中可理解的列名称作为列的别名添加到视图的定义中去。如下所示:

```
select x.inst_id AS inst_id,
x.indx+1 AS num, /* C language arrays start at offset 0,
but SQL stuff usually starts at offset 1*/
```

```
• ksppinm AS name,
  ksppity AS type,
  ksppstvl AS value,
  ksppstdvl AS display_value,
  ksppstdf AS isdefault,
  decode(bitand(ksppiiflg/256,1),1,'TRUE','FALSE') AS isses_modifiable,
  decode(bitand(ksppiiflg/65536,3),1,'IMMEDIATE',2,'DEFERRED',
3,'IMMEDIATE','FALSE') AS issys_modifiable,
  decode(bitand(ksppiiflg,4),4,'FALSE',
  decode(bitand(ksppiiflg/65536,3), 0, 'FALSE', 'TRUE')) AS isinstance_modifiable,
  decode(bitand(ksppstvf,7),1,'MODIFIED',4,'SYSTEM_MOD','FALSE') AS ismodified,
  decode(bitand(ksppstvf,2),2,'TRUE','FALSE') AS isadjusted,
  decode(bitand(ksppilrmflg/64, 1), 1, 'TRUE', 'FALSE') AS isdeprecated,
  ksppdesc AS description,
  ksppstcmnt AS update_comment,
  ksppihash AS hash
from x$ksppi x, x$ksppcv y
where (x.indx = y.indx)
and (
  (translate(ksppinm,'_','#') not like '##%')
  and
  (
    (translate(ksppinm,'_','#') not like '##%')
    or (ksppstdf = 'FALSE') or (bitand(ksppstvf,5) > 0)
  )
);
```

因此，我们就得到了 X\$KSPPPI 和 X\$KSPPCV 表中隐蔽列名称到 GV\$PARAMETER 视图中可理解的列名称的映射。表 9-3 以及表 9-4 分别都是 X\$KSPPPI 和 X\$KSPPCV 的映射。

表9-3 X\$KSPPPI表的列映射

X\$表中的列	GV\$视图中的列
X\$KSPPPI.ADDR	n/a
X\$KSPPPI.INDX	GV\$PARAMETER.NUM
X\$KSPPPI.INST_ID	GV\$PARAMETER.INST_ID
X\$KSPPPI.KSPPINM	GV\$PARAMETER.NAME
X\$KSPPPI.KSPPITY	GV\$PARAMETER.TYPE
X\$KSPPPI.KSPPDESC	GV\$PARAMETER.DESRIPTION
X\$KSPPPI.KSPPIFLG	GV\$PARAMETER.ISSES_MODIFIABLE、 GV\$PARAMETER.ISSYS_MODIFIABLE、 GV\$PARAMETER.ISINSTANCE_MODIFIABLE
X\$KSPPPI.KSPPILRMFLG	GV\$PARAMETER.ISDEPRECATED
X\$KSPPPI.KSPPIHASH	GV\$PARAMETER.HASH

列 X\$ KSPPPI.KSPPIFLG 是一个标记，它在 GV\$PARAMETER 视图使用 BITAND 和 DECODE 可以扩展为 3 个独立的列。

表9-4 X\$ KSPPCV表的列映射

X\$ 表中的列	GV\$视图中的列
X\$KSPPCV.ADDR	n/a
X\$KSPPCV.INDX	GV\$PARAMETER.NUM
X\$KSPPCV.INST_ID	GV\$PARAMETER.INST_ID
X\$KSPPCV.KSP PSTVL	GV\$PARAMETER.VALUE
X\$KSPPCV.KSP PSTDVL	GV\$PARAMETER.DISPLAY_VALUE
X\$KSPPCV.KSP PSTDF	GV\$PARAMETER.ISDEFAULT
X\$KSPPCV.KSP PSTVF	GV\$PARAMETER.ISMODIFIED
X\$KSPPCV.KSP PSTCMNT	GV\$PARAMETER.UPDATE_COMMENT

若要检索在文档中未作说明的参数，我们需要对 where 子句作出如下修改——只有带有下列划线的参数才满足限制条件。因此，我们在原始 GV\$视图定义的基础上采用 LIKE ESCAPE 替换掉不适用的 TRANSLATE（见文件 `hidden_parameters.sql`）。

```
SELECT kspinnm name,
       kspstvl value,
       kspdesc description
FROM x$ksppi x, x$ksppcv y
WHERE (x.indx = y.indx)
AND x.inst_id=userenv('instance')
AND x.inst_id=y.inst_id
AND kspinnm LIKE '\_%' ESCAPE '\'
ORDER BY name;
```

在 Oracle10g R2 上执行这个查询最终得到 1 124 个未在文档中说明的参数，这是一个相当可观的数字（在 Oracle9i R2 中是 540，而在 Oracle11g 中是 1 627）。如下所示是查询结果的部分引用，包含了在 Oracle10g 中引入的双下划线参数。

NAME	VALUE	DESCRIPTION
-----	-----	-----
_4031_dump_bitvec	67194879	bitvec to specify dumps prior to 4031 error
...		
_PX_use_large_pool	FALSE	Use Large Pool as source of PX buffers
_db_cache_size	473956352	Actual size of DEFAULT buffer pool for standard block size buffers
_dg_broker_service_names	orcl_XPT	service names for broker use
_java_pool_size	4194304	Actual size in bytes of java pool
_large_pool_size	4194304	Actual size in bytes of large pool
_shared_pool_size	117440512	Actual size in bytes of shared pool
_streams_pool_size	4194304	Actual size in bytes of streams pool
_abort_recovery_on_join	FALSE	if TRUE, abort recovery on join reconfigurations
...		
_yield_check_interval	100000	interval to check whether actses should yield
1124 rows selected.		

9.5 X\$表与 V\$视图之间的关系

如果有这样一份文档，通过查阅它可以看到任意 V\$视图底层的 X\$固定表，或者任意 X\$固定表的上层 V\$视图，那岂不是很方便？这样一份文档对上一节介绍的剖析过程大有帮助，并且它最好是可以自动生成的。以下 4 点是编码生成这样一份文档的主要环节。

- 动态性能视图 V\$SESSION 的列 PREV_HASH_VALUE 保存了一次数据库会话中之前执行的 SQL 语句的散列值。
- SQL 语句的执行计划可以通过它的散列值从视图 V\$SQL_PLAN 中得到。
- 访问 V\$SQL_PLAN.OBJECT_NAME 可以得到一个已经缓存了的执行计划中所包含的对象名称。
- X\$表中所有的行数据源都包含字符串“FIXED_TABLE”。

基于以上这些资料，我们可以设计出这样一个算法——通过在 V\$视图上执行 SELECT 语句从 V\$SQL_PLAN 视图中获取底层 X\$表的名称。最后的任务是在一个表中保存 V\$视图与 X\$表的联系，如下所示：

```
CREATE TABLE x_v_assoc (
  x_id number,
  v_id number);
```

不需要存储固定表或者视图的名称，表 X_V_ASSOC 保存了它们的对象标识符。可以选取二者中的任意一列与 V\$FIXED_TABLE.OBJECT_ID 联合查找得到它们的名称。

以下是上述算法的伪代码：

```
LOOP over all V$ view names and their OBJECT_ID in V$FIXED_TABLE
  Parse and execute SELECT * FROM view_name using dynamic SQL (DBMS_SQL)
  Get the value of V$SESSION.PREV_HASH_VALUE for the current session
  LOOP over all object names from the execution plan for the previous SQL
statement in V$SQL_PLAN,
    considering only row sources that contain the string "FIXED TABLE"
    Translate the object name to the fixed table number
V$FIXED_TABLE.OBJECT_ID
  Insert the object identifier of the X$ table and the object identifier
of the associated V$ view
    into the table X_V_ASSOC
  END LOOP
END LOOP
```

完整的源代码太长以至于不能在这里重现，但可在源代码库中的文件 x_v_assoc.sql 中查看到。一旦表 X_V_ASSOC 中填充了数据，就可以基于它们在以下的查询中检索 X\$固定表和 V\$视图：

```
SQL> SELECT
  f1.name x_name,
  f2.name v_name
FROM x_v_assoc a, v$fixed_table f1, v$fixed_table f2
WHERE a.x_id=f1.object_id
AND a.v_id=f2.object_id
ORDER BY x_name;
```

在 Oracle10g R2 中，这个查询将返回 727 行，要获得 V\$视图底层的 X\$固定表的列表，运行

以下查询：

```
SQL> SELECT
f1.name v_name,
f2.name x_name
FROM x_v_assoc a, v$fixed_table f1, v$fixed_table f2
WHERE a.v_id=f1.object_id
AND a.x_id=f2.object_id;
```

表 9-5 描述了由上面这个查询返回的关系中的一个小的子集。

表9-5 部分V\$视图以及它们底层的X\$固定表

V\$视图	相应的底层 X\$ table
V\$ARCHIVED_LOG	X\$KCCAL
V\$BH	X\$SBH
V\$CONTROLFILE	X\$KCCCF
V\$DATAFILE	X\$KCCFN
V\$DB_PIPES	X\$KGLOBAL
V\$INSTANCE	X\$KVIT、X\$QUIESCE、X\$KSUXSINST
V\$LATCH	X\$KSLLD
V\$MYSTAT	X\$KSUSGIF、X\$KSUMYSTA
V\$PROCESS	X\$KSUPR
V\$RECOVER_FILE	X\$KCVFHMRR
V\$SEGMENT_STATISTICS	X\$KTSSO、X\$KSOLSFTS
V\$SESSION	X\$KSLED、X\$KSUSE
V\$SQL	X\$KGLCURSOR_CHILD
V\$SQL_BIND_DATA	X\$KXSBD
V\$SQL_PLAN	X\$KQLFXPL

通过打印脚本文件 x_v_assoc.sql 的输出，一个包含所有 V\$视图与 X\$表联系的文本文档就生成了。

9.6 源代码库

表 9-6 列出了本章的源文件和它们相应的功能。

表9-6 X\$固定表的源代码库

文 件 名	功 能
hidden_parameters.sql	列出来自X\$KSPPI以及X\$KSPPCV的隐藏参数
x_v_assoc.sql	生成V\$固定视图与X\$固定表之间的关系

在 *Oracle Database Performance Tuning Guide 10g Release 2* 中对 X\$固定表 X\$BH 只做出了部分说明。本章则提供了 X\$BH 表的补充说明。与你的预期相反，我将会通过仔细审查在文档中未作详细说明的 X\$表（比如 X\$BH）来说明它并不是优化系统性能的捷径。

闕锁（latch）是 ORACLE 数据库管理系统用来保护内存结构的一种低层次锁机制。等待事件 latch free 用来解决进程尝试获取闕锁的等待时间，而闕锁在第一次请求时是不可用的。在 Oracle10g 中，闕锁有几个与之相关的专用等待事件，这些事件通常由闕锁争用引发。对于这些事件，闕锁的名称将会出现在等待事件的名称中，等待事件 latch: library cache 以及 latch: cache buffers chains 可以作为典型例子。Oracle10g 中的其他等待事件，消除了找出一个通用 latch free 等待所属的闕锁的需求。在 Oracle9i 中，V\$SESSION_WAIT.P2 包含了等待闕锁的数量。闕锁的数量与 V\$LATCH.LATCH#是一致的。通过联结 V\$SESSION_WAIT 和 V\$LATCH，可以检索到 V\$LATCH.NAME 中闕锁的名称。

缓存缓冲区链的闕锁用来保护处于缓冲区中缓存的缓冲区列表。如果要从缓冲区列表中查找、添加或者删除缓冲区就需要使用闕锁。这些闕锁的争用通常表明对处于某些闕锁保护之下的数据块存在着争用的情况。可以通过查看固定视图 V\$LATCH_CHILDREN 的列 MISSES 以检测争用情况的存在。下面的查询标识了丢失计数最高的子闕锁：

```
SQL> SELECT name, addr, latch#, child#, misses, sleeps
FROM v$latch_children
WHERE misses > 10000
ORDER BY misses;
NAME                ADDR          LATCH# CHILD# MISSES SLEEPS
-----
cache buffers chains 697ACFD8      122    190  11909    125
session idle bit     699B2E6C       7     2   13442     75
library cache pin    68BC3C90     216     2   30764     79
library cache        68BC3B58     214     2  178658    288
```

地址为 697ACFD8 的缓存缓冲区子闕锁是未命中计数最高的闕锁之一。V\$LATCH_CHILDREN 中的计数器是自实例启动时开始运行的。我之所以选择查询谓词 MISSES>10000，只是因为这可以把查询结果限制在 4 行。当然，你应该检查出现性能问题的时间间隔段的数字，而不是自实例启动以来的所有数字。

使用第 9 章所述的理解 X\$表中隐藏列名称的方法，就能得到表 10-1 所示 GV\$BH 和 X\$BH 中列的映射。值得注意的是，X\$BH 中的部分列就包含了 GV\$BH 所有列。X\$BH 表中大约有一半的列在 GV\$BH 没有得到具体化。表 10-1 中，表头为“GV\$BH 列名”的列中，值 n/a 就代表所对应的列在 GV\$BH 中没有具体化。

表10-1 GV\$BH视图中的列到X\$BH表的映射

X\$BH列名	GV\$BH列名	含 义
ADDR	n/a	缓存头部地址
INDX	n/a	缓存头部索引 (0-n)
INST_ID	INST_ID	实例编号，对应于GV\$INSTANCE.INSTANCE_NUMBER
HLADDR	n/a	子锁地址
BLSIZ	n/a	数据块大小
FLAG	DIRTY、TEMP、PING、STALE、DIRECT	数据块类型以及状态标志
TS#	TS#	表空间编号，对应于V\$TABLESPACE.TS#
FILE#	FILE#	文件编号，对应于V\$DATAFILE.FILE#
DBARFIL	n/a	相对文件编号
DBABLK	BLOCK#	从数据块地址开始的编号
CLASS	CLASS#	分类号
STATE	STATUS	数据块状态（free、xcu、cr等）
LE_ADDR	LOCK_ELEMENT_ADDR	锁元素地址
OBJ	OBJD	数据段中包含的字典对象数目
CR_SCN_BAS	n/a	一致性读取SCN base
CR_SCN_WRP	n/a	一致性读取SCN wrap
TCH	n/a	读取计数touch count
TIM	n/a	读取时间touch time

通过连接 X\$BH 和 DBA_OBJECTS，我们可以找出子锁保护的数据块属于哪些数据库对象（见脚本文件 latch_vs_blocks.sql）。

```
SQL> SELECT bh.file#, bh.dbablk, bh.class,
decode(bh.state,0,'free',1,'xcu',2,'scur',3,'cr',
4,'read',5,'mrec',6,'irec',7,'write',8,'pi',
9,'memory',10,'mwrite',11,'donated') AS status,
decode(bitand(bh.flag,1), 0, 'N', 'Y') AS dirty, bh.tch,
o.owner, o.object_name, o.object_type
FROM x$bh bh, dba_objects o
WHERE bh.obj=o.data_object_id
AND bh.hladdr='697ACFD8'
ORDER BY tch DESC;
FILE# DBABLK CLASS STATUS DIRTY TCH OWNER OBJECT_NAME OBJECT_TYPE
-----
4 476 1 xcu N 24 NDEBES SYS_IOT_TOP_53076 INDEX
1 48050 1 xcu N 4 SYS SYS_C00651 INDEX
1 7843 1 xcu N 2 SYS ICOL$ TABLE
```

1	7843	1	xcur	N	2	SYS	IND\$	TABLE
1	7843	1	xcur	N	2	SYS	COL\$	TABLE
1	7843	1	xcur	N	2	SYS	CLU\$	TABLE
1	7843	1	xcur	N	2	SYS	C_OBJ#	CLUSTER

索引 SYS_IOT_TOP_53076 在所有被子锁保护的对象中具有最高的命中计数^①。该索引是索引组织表的基础部分。

```
SQL> SELECT table_name, table_type
FROM dba_indexes
WHERE index_name='SYS_IOT_TOP_53076';
TABLE_NAME          TABLE_TYPE
-----
CUSTOMER              TABLE
```

最大的问题是整个调研是否值得 DBA 去花费时间和精力，或者他是否打算被 CTD 折磨。CTD (Compulsive Tuning Disorder, 强制调优障碍症) 是由 Gaja Krishna Vaidyanatha 创造的一个术语，他是 *Oracle Performance Tuning 101* 一书的合著者。当然，CTD 只是一个对严重精神障碍 OCD (Obsessive Compulsive Disorder, 强迫症) 的双关语。通过查看 Statspack 报告中引起锁争用的量化工作负载的可以看出，等待事件 latch: cache buffers chains 的影响是微不足道的。我修改了 Statspack 报告参数 top_n_events，使之在报告中包括排名前十的定时事件（默认为 5）。尽管如此，缓存缓冲区链的等待事件还是没能包括在其中。

Snapshot	Snap Id	Snap Time	Sessions	Curs/Sess	Comment
Begin Snap:	662	15-Oct-07 20:20:20	16	5.1	contention customer
End Snap:	663	15-Oct-07 20:24:30	17	6.1	contention customer
Elapsed:		4.17 (mins)			
...					
Top 10 Timed Events					
~~~~~					
Event	Waits	Time (s)	Avg wait (ms)	%Total Call Time	
-----					
db file sequential read	3,109	89	29	39.7	
CPU time		59		26.3	
control file sequential read	532	23	43	10.1	
db file parallel write	191	17	87	7.4	
db file scattered read	580	9	15	3.8	
buffer busy waits	12,530	7	1	3.2	
read by other session	420	4	8	1.6	
os thread startup	5	3	597	1.3	
log file sync	59	3	48	1.3	
SQL*Net more data to client	88,397	3	0	1.2	
...					
Wait Events DB/Inst: TEN/ten Snaps: 662-663					
...					

① 也许可以使用 where 子句 WHERE bh.file# = e.file_id AND bh.dbablk BETWEEN e.block_id AND e.block_id+e.blocks-1 联合 X\$BH 与 DBA_EXTENTS。然而，这样一个查询的响应时间将会相当长。



Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
db file sequential read	3,109	0	89	29	97.2
...					
latch: cache buffers chains	53	0	0	5	1.7
...					

在 Statspack 5 min 时间间隔内的报告中，有 165 ms 的时间花在等待缓存缓冲区链的锁。那么这个等待事件会对独立的会话产生影响吗？这个可以通过查询 V\$SESSION_EVENT 得到确认。

```
SQL> SELECT s.sid, s.serial#, p.spid, e.event, e.total_waits, e.time_waited
FROM v$session s, v$process p, v$session_event e
WHERE s.paddr=p.addr
AND s.sid=e.sid
AND s.type='USER'
AND e.event LIKE 'latch%'
ORDER BY time_waited;
SID SERIAL# SPID EVENT TOTAL_WAITS TIME_WAITED
```

```
---
27 769 4400 latch: cache buffers chains 49 0
27 769 4400 latch: library cache pin 3 0
32 1921 5232 latch: library cache pin 29 0
32 1921 5232 latch: cache buffers chains 129 1
32 1921 5232 latch: library cache 86 1
27 769 4400 latch: library cache 35 1
```

```
SQL> ORADEBUG SETOSPID 5232
Oracle pid: 18, Windows thread id: 5232, image: ORACLE.EXE (SHAD)
SQL> ORADEBUG EVENT 10046 trace name context forever, level 8
Statement processed.
SQL> ORADEBUG TRACEFILE_NAME
c:\programs\oracle\product\admin\ten\udump\ten_ora_5232.trc
```

因为 SID 为 32、操作系统进程标识 SPID 为 5332 的会话对锁 latch: cache buffers chains 有最多的等待次数，我决定采用 ORADEBUG（详见第 37 章）在第 8 级跟踪这个会话。然后我使用免费的扩展 SQL 跟踪分析器 ESQLTRCPROF 从跟踪文件中生成了一个资源负荷表，ESQLTRC-PROF 工具包含在了本书中。

```
C:> esqltrcprof.pl ten_ora_5232.trc
ORACLE version 10.2 trace file. Timings are in microseconds (1/1000000 sec)
Resource Profile
=====

Response time: 43.387s; max(tim)-min(tim): 58.216s
Total wait time: 26.738s
-----
```

```
Note: 'SQL*Net message from client' waits for more than 0.005s
are considered think time
Wait events and CPU usage:
```

Duration	Pct	Count	Average Wait Event/CPU Usage/Think Time	
22.439s	51.72%	188693	0.000119s	SQL*Net message from client
16.250s	37.45%	188741	0.000086s	total CPU
1.124s	2.59%	2	0.561751s	log file switch (checkpoint incomplete)
0.774s	1.78%	3	0.257905s	think time
0.642s	1.48%	362	0.001774s	enq: TX - index contention
0.625s	1.44%	4543	0.000138s	buffer busy waits
0.569s	1.31%	188696	0.000003s	SQL*Net message to client
0.399s	0.92%			unknown
0.329s	0.76%	4	0.082256s	log file switch completion
0.170s	0.39%	5	0.034085s	log buffer space
0.057s	0.13%	4	0.014250s	db file sequential read
0.005s	0.01%	1	0.004899s	log file sync
0.002s	0.00%	20	0.000093s	latch: library cache
0.001s	0.00%	13	0.000056s	latch: cache buffers chains
0.001s	0.00%	10	0.000054s	latch free
0.000s	0.00%	28	0.000015s	buffer deadlock
0.000s	0.00%	10	0.000022s	latch: cache buffers lru chain
0.000s	0.00%	1	0.000101s	latch: library cache pin
-----				
43.387s	100.00%	Total response time		

Total number of roundtrips (SQL*Net message from/to client): 188696

CPU usage breakdown

```
-----
parse CPU:    0.00s (12 PARSE calls)
exec  CPU:   16.25s (188707 EXEC calls)
fetch CPU:    0.00s (22 FETCH calls)
```

...

通过访问 X\$BH 表，可以提早识别出在表 CUSTOMER 上开销最大的 SQL 语句是 INSERT 语句。

Statements Sorted by Elapsed Time (including recursive resource utilization)

=====

Hash Value: 1256130531 - Total Elapsed Time (excluding think time): 42.608s

```
INSERT INTO customer(id, name, phone)
VALUES (customer_id_seq.nextval, :name, :phone)
RETURNING id INTO :id
```

DB Call	Count	Elapsed	CPU	Disk	Query	Current	Rows
PARSE	0	0.0000s	0.0000s	0	0	0	0
EXEC	188695	19.6005s	16.2344s	4	4241	581188	188695
FETCH	0	0.0000s	0.0000s	0	0	0	0
-----							
Total	188695	19.6005s	16.2344s	4	4241	581188	188695

Wait Event/CPU Usage/Think Time	Duration	Count
---------------------------------	----------	-------

-----		
SQL*Net message from client	22.439s	188692
total CPU	16.234s	188695
think time	0.774s	3
enq: TX - index contention	0.642s	362
buffer busy waits	0.625s	4543
SQL*Net message to client	0.569s	188694
log file switch completion	0.329s	4
log buffer space	0.170s	5
db file sequential read	0.023s	2
latch: library cache	0.002s	20
latch: cache buffers chains	0.001s	13
latch free	0.001s	10
buffer deadlock	0.000s	28
latch: cache buffers lru chain	0.000s	10
latch: library cache pin	0.000s	1

同样地，等待事件 latch: cache buffers chains 的作用是微不足道的。问题在别的地方：51% 的响应时间是因为客户端的 SQL*Net 消息，它是整体响应时间最主要的构成因素。通过查看 INSERT 语句的统计信息中列 Count 和 Rows 的数值可以看出，它们是完全一样的。此外，语句的执行数目和网络的往返数目（即来自客户端的 SQL*Net 消息）基本上完全一样。这意味着应用程序是单行插入，例如，每执行一次 INSERT 语句就有一次网络往返。这一观点可以通过查看跟踪文件中 EXEC 条目的行数得到证实，这些条目的网络往返次数均为 1。

```
WAIT #3: nam='SQL*Net message to client' ela= 3 driver id=1413697536
#bytes=1 p3=0 obj#=-1 tim=347983945867
EXEC #3:c=0,e=79,p=0,cr=0,cu=2,mis=0,r=1,dep=0,og=1,tim=347983945892
WAIT #3: nam='SQL*Net message from client' ela= 109 driver id=1413697536
#bytes=1 p3=0 obj#=-1 tim=347983946105
```

我们可以对应用程序重新编码，以数组的形式执行插入，这样可以大幅减少响应时间。对于怎样在互联网上搜索由访问 X\$表造成的卡顿争用，我将给出如下建议，这条建议不会给你铺平性能调优的康庄大道，恰恰相反，可能会使你走上强制调整障碍症这条“不归路”：始终使用基于响应时间的分析方法，切勿深钻一些感观上的异常或者你认为过高的比率。几年前，我有幸与著名的神经科专家 Jaak Penksepp 谈话。时至今日，我还记得他问道：“你有哪些数据可以支撑这个观点呢？”很多时候，那些没有事实依据的论断都可以被这个问题驳回。我们作为 DBA 应该更好地采用类似于医学界的科学方法，而不是贸然下结论然后随意实施假设方案。

## 源代码库

表 10-2 列出了本章的源代码文件以及它们的功能。

表10-2 固定表X\$BH源代码库

文 件 名	功 能
latch_vs_blocks.sql	检索一个子门锁保护的数据库对象

# 11

## X\$KSLED 以及增强的会话等待数据

固定表 X\$KSLED 是一个未在文档中说明的表。而动态性能视图 V\$SESSION_WAIT 是基于固定表 X\$KSLED 以及 X\$KSUSECST 的。X\$KSLED 提供等待事件名给 V\$SESSION_WAIT, X\$KSUSECST 则持有定时信息。无论是在 Oracle9i 中还是在 Oracle10g 中, 都没有一个视图可以在会话级别给单个等待事件提供超过厘秒的精度^①。也没有任何一个视图去整合等待信息和 V\$PROCESS 中的操作系统进程信息。并且, 要正确理解视图 V\$SESSION_WAIT 中的列 WAIT_TIME 和 SECONDS_IN_WAIT 有点麻烦, 它们两个的值取决于列 STATE 的值。

直接访问 X\$固定表使得我们能够对等待事件的跟踪达到微秒精度, 而不需要启用 SQL 跟踪。此外, 也可以构建一个结合了来自 V\$SESSION、V\$SESSION_WAIT 和 V\$PROCESS 视图信息的增强版 V\$SESSION_WAIT, 这样更有利于去理解它。注意到在 Oracle9i 中 V\$SESSION 不包含等待事件信息, 然而在 Oracle10g 中又纳入了等待事件信息。本章介绍的增强会话等待视图与 Oracle9i、Oracle10g 以及 Oracle11g 均兼容。

### 11.1 深度探讨视图 V\$SESSION_WAIT

通过第 9 章对 V\$SESSION_WAIT 的深入剖析, 我们可以很明显地看出 V\$SESSION_WAIT 视图是基于表 X\$KSLED 和表 X\$KSUSECST 的。对应于 GV\$SESSION 中的列名, 给从 V\$FIXED_VIEW_DEFINITION 视图中检索得到的视图定义添加列别名, 得到如下 SELECT 语句:

```
SELECT s.inst_id AS inst_id,  
       s.indx AS sid,  
       s.ksusseq AS seq#,  
       e.kslednam AS event,  
       e.ksledp1 AS p1text,  
       s.ksussp1 AS p1,  
       s.ksussp1r AS p1raw,
```

① 在 Oracle11g 中, 动态性能视图 V\$SESSION_WAIT 添加了如下新的列: WAIT_TIME_MICRO、TIME_REMAINING_MICRO 以及 TIME_SINCE_LAST_WAIT_MICRO。

```

e.ksledp2 AS p2text,
s.ksussp2 AS p2,
s.ksussp2r AS p2raw,
e.ksledp3 AS p3text,
s.ksussp3 AS p3,
s.ksussp3r AS p3raw,
decode(s.ksusstim,0,0,-1,-1,-2,-2,
decode(round(s.ksusstim/10000),0,-1,round(s.ksusstim/10000)))
AS wait_time,
s.ksusewtm AS seconds_in_wait,
decode(s.ksusstim, 0, 'WAITING', -2, 'WAITED UNKNOWN TIME', -1,
'WAITED SHORT TIME', 'WAITED KNOWN TIME') AS state
FROM x$ksusecst s, x$ksled e
WHERE bitand(s.ksSPAflg,1) != 0
and bitand(s.ksuseflg,1) != 0
and s.ksusseq != 0
and s.ksusopc = e.indx

```

我们可以看到在表 X\$KSUSECST 中的微秒精度通过除以 10 000 被人为地降低到了厘秒精度。在降低的精度级别中，我们不可能知道短等待事件（比如 db 文件顺序读取、db 文件分散读取或者是 RAC 中与全局缓存相关的等待事件）花了多长时间。等待时间少于 1 厘秒就会在 V\$SESSION_WAIT 中显示为-1。在这个精度级别，我们不可能看到会话级别的磁盘访问时间。只要等待事件的持续时间一直低于 1 厘秒，I/O 服务时间的峰值仍然会被忽视，但是它通常会在 1 厘秒之下。以下就是一个例子：

```

SQL> SELECT event, wait_time, seconds_in_wait, state
FROM v$session_wait
WHERE (state='WAITED KNOWN TIME' or state='WAITED SHORT TIME')
AND event != 'null event';

```

EVENT	WAIT_TIME	SECONDS_IN_WAIT	STATE
db file sequential read	-1	0	WAITED KNOWN TIME
SQL*Net message from client	-1	0	WAITED KNOWN TIME
SQL*Net message to client	-1	0	WAITED KNOWN TIME

## 11.2 改进的 V\$SESSION_WAIT 视图

现在视图 V\$SESSION_WAIT 的限制已经显而易见了，我们可以尝试着去改进它。改进的目标如下。

- 提供毫秒精度的等待事件。
- 整合进程、会话以及会话等待信息。
- 以一种方便直观的格式展示等待状态、等待事件，而不需要视图用户进一步地处理。

进程和会话信息可以通过基于 X\$表的视图 V\$PROCESS 以及 V\$SESSION 得到，它们分别基于表 X\$KSUSE 和 X\$KSUPR。构建一个符合上述目标的相对较大的视图得需要些毅力。下面就是创建该视图的数据描述语言，我给它命名为 X_\$SESSION_WAIT（见脚本文件 x_session_wait.sql）：

```

CREATE OR REPLACE view x_$session_wait AS
SELECT s.inst_id AS inst_id,
s.indx AS sid,
se.ksuseser AS serial#,
-- spid from v$process
p.ksuprpri AS spid,
-- columns from v$session
se.ksuudlna AS username,
decode(bitand(se.ksuseidl,11),1,'ACTIVE',0,
decode(bitand(se.ksuseflg,4096),0,'INACTIVE','CACHED'),2,'SNIPED',3,
'SNIPED','KILLED') AS status,
decode(kspspatyp,1,'DEDICATED',2,'SHARED',3,'PSEUDO','NONE') AS server,
se.ksuseunm AS osuser,
se.ksusepid AS process,
se.ksusemnm AS machine,
se.ksusetid AS terminal,
se.ksusepnm AS program,
decode(bitand(se.ksuseflg,19),17,'BACKGROUND',1,'USER',2,'RECURSIVE','?') AS type,
se.ksusesqh AS sql_hash_value,
se.ksusepha AS prev_hash_value,
se.ksuseapp AS module,
se.ksuseact AS action,
se.ksuseclid AS client_identifier,
se.ksuseobj AS row_wait_obj#,
se.ksusefil AS row_wait_file#,
se.ksuseblk AS row_wait_block#,
se.ksuseslt AS row_wait_row#,
se.ksuseltn AS logon_time,
se.ksusegrp AS resource_consumer_group,
-- columns from v$session_wait
s.ksusseq AS seq#,
e.kslednam AS event,
e.ksledp1 AS p1text,
s.ksussp1 AS p1,
s.ksussp1r AS p1raw,
e.ksledp2 AS p2text,
s.ksussp2 AS p2,
s.ksussp2r AS p2raw,
e.ksledp3 AS p3text,
s.ksussp3 AS p3,
s.ksussp3r AS p3raw,
-- improved timing information from x$ksusecst
decode(s.ksusstim,
-2, 'WAITED UNKNOWN TIME',
-1, 'LAST WAIT < 1 microsecond', -- originally WAITED SHORT TIME
0, 'CURRENTLY WAITING SINCE ' || s.ksusewtm || 's',
'LAST WAIT ' || s.ksusstim/1000 || ' milliseconds (' ||
s.ksusewtm || 's ago)') wait_status,
to_number(decode(s.ksusstim,0,NULL,-1,NULL,-2,NULL, s.ksusstim/1000))
AS wait_time_milli

```



```

from x$ksusecst s, x$ksled e, x$ksuse se, x$ksupr p
where bitand(s.ksspflg,1)!=0
and bitand(s.ksuseflg,1)!=0
and s.ksusseq!=0
and s.ksussopc=e.indx
and s.indx=se.indx
and se.ksusepro=p.addr;

```

列 WAIT_TIME_MILLI 中的等待时间的单位为毫秒。并且保留了毫秒的小数位，这样我们就得到了微秒的精度。列 WAIT_STATUS 指示会话当前是否在等待中。如果会话在等待，视图将显示出它已经等待了多少秒。对于那些当前没有等待的会话，视图将会报告上一次等待事件的持续时间以及自上一次等待事件开始所经过的时间。如下所示的查询证实了等待事件确实是以毫秒精度计时的：

```

SQL> SELECT sid, serial#, spid, username, event, wait_status, wait_time_milli
FROM x_$session_wait
WHERE wait_time_milli > 0 and wait_time_milli <10;

```

SID	SERIAL#	SPID	USERNAME	EVENT	WAIT_STATUS	WAIT_TIME_MILLI
24	58259	1188090	SYS	db file sequential read (0 s ago)	LAST WAIT 6.541 ms	6.541
22	48683	966786	SYS	SQL*Net message to client (0 s ago)	LAST WAIT .003 ms	.003

我已经包含了标识符为 SPID 的操作系统级进程，在这个视图中它由 X\$固定表 X\$KSUPR 的列 KSUPRPID 所代表。该列对应于 V\$PROCESS.SPID。它对于 SQL 跟踪 ORADEBUG SETOSPID 以及 ORADEBUG EVENT 很有用（参见第 37 章）。当操作系统工具比如 top、prstat 或者 nmon 用来标识资源密集进程（比如高 I/O 等待百分比）时，列 SPID 就基于这些工具所显示的进程标识符，提供对等待信息的即时访问。图 11-1 为对视图 CV_\$SESSION_WAIT 查询所得的结果，包括 SPID 以及上一次等待事件的详细信息。

SID	SERIAL#	SPID	USERNAME	PROGRAM	MODULE	ACTION	CLIENT_IDENTIFIER	EVENT	WAIT_STATUS	WAIT_TIME_MILLI
1	26	91322	NOBES	SQL Developer	SQL Developer	(null)	(null)	SQL*Net message to client	LAST WAIT .005 ms (0 s ago)	0.005
2	28	13404212	SYSTEM	exp.exe	exp.exe	(null)	(null)	db file sequential read	LAST WAIT 22.324 ms (0 s ago)	22.324
3	49	2791162	NOBES	sqlplus.exe	SQL*Plus	(null)	(null)	Streams AQ: waiting for me	CURRENTLY WAITING SINCE 197 s	(null)
4	42	439996	(null)	ORACLE.EX...	(null)	(null)	(null)	jobq slave wait	CURRENTLY WAITING SINCE 26 s	(null)
5	27	8305484	NOBES	SQL Developer	Oracle SQL Developer	Code Insight	(null)	SQL*Net message from client	CURRENTLY WAITING SINCE 3303 s	(null)
6	26	2842364	HR	sqlplus.exe	SQL*Plus	(null)	(null)	SQL*Net message from client	CURRENTLY WAITING SINCE 3261 s	(null)
7	21	7682232	NOBES	perl.exe	img_load	(null)	(null)	log file sync	CURRENTLY WAITING SINCE 0 s	(null)

图 11-1 对增强版会话等待视图 CV_\$SESSION_WAIT 的查询结果

SYS 以外的用户可以获准访问 X_\$SESSION_WAIT，这通过授予视图的 SELECT 特权然后创建一

个公共同义词来达到。下面是一些模仿内建 V\$以及 GV\$视图层次化方法所写的 DDL 语句。在模式 SITE_SYS 中创建了一些额外的数据库对象，以防止扰乱具有特定对象的数据字典。模式 SITE_SYS 中视图的特权被授予 SELECT_CATALOG_ROLE。在视图和同义词的名称中自定义使用字母 C 与内建的动态性能视图进行区分。用公共同义词 CV\$SESSION_WAIT 以及 CGV\$SESSION_WAIT 分别访问增强版视图 V_\$SESSION_WAIT 和 GV_\$SESSION_WAIT。

```
SQL> GRANT SELECT ON x_$session_wait TO site_sys WITH GRANT OPTION;
Grant succeeded.
SQL> CREATE OR REPLACE VIEW site_sys.cgv_$session_wait AS
SELECT * FROM sys.x_$session_wait;
View created.
SQL> CREATE OR REPLACE VIEW site_sys.cv_$session_wait AS
SELECT * FROM sys.x_$session_wait WHERE inst_id=userenv('instance');
View created.
SQL> GRANT SELECT ON site_sys.cgv_$session_wait TO select_catalog_role;
Grant succeeded.
SQL> GRANT SELECT ON site_sys.cv_$session_wait TO select_catalog_role;
Grant succeeded.
SQL> CREATE OR REPLACE PUBLIC SYNONYM cgv$session_wait
FOR site_sys.cgv_$session_wait;
Synonym created.
SQL> CREATE OR REPLACE PUBLIC SYNONYM cv$session_wait FOR site_sys.cv_$session_wait;
Synonym created.
```

11.3 源代码库

表 11-1 列出了本章的源代码文件及其相应的功能。

表11-1 X\$KSLED源代码库

文 件 名	功 能
x_session_wait.sql	该脚本包含访问会话等待事件的增强版视图。除了视图V\$SESSION_WAIT本身的列之外，这些视图还包括来自V\$PROCESS和V\$SESSION的信息。等待时间精确到微秒。该脚本还包含授权和公共同义词



X\$固定表 X\$KFFXP 在 Oracle10g 以及 Oracle11g 中均没有在文档中说明。通过表 X\$KFFXP 可以查看与镜像相关的 ASM 元数据, 以及 ASM 文件区到 ASM 磁盘的分配单位的任务信息。DBA 深入了解 X\$KFFXP 表之后, 可以直接使用操作系统命令访问存储在 ASM 中的数据库参数与服务器参数文件。除了教学用途, 对 ASM 存储形式的理解有助于拯救数据以及排除故障。

## 12.1 固定表 X\$KFFXP

每一个 ASM 文件由一个或多个数据区组成。这些区被条带化到文件所在的磁盘组。一个数据区的大小和一个 ASM 分配单位的大小 (参数 ASM_AUSIZE) 是一样的。由于采用了条带化技术, ASM 磁盘中邻近的 ASM 文件区和非邻近的存储文件之间就需要维护一张映射表。

一个 ORACLE 的数据段就由多个数据区组成。每个数据区包括一个从数据文件开始以某个偏移量偏移的连续数据块。在数据文件中, 一个数据段的数据区位置和大小可以通过查询字典视图 DBA_EXTENTS 得到。这适用于数据文件所有的存储选择方式, 比如文件系统、裸设备或者 ASM 文件。

X\$固定表 X\$KFFXP 中保存有 ASM 磁盘中 ASM 文件区与分配单元之间的映射。它保留了每一个 ASM 文件区被条带化和镜像之后位置的跟踪信息。你必须连接到一个 ASM 实例并从 DBA_EXTENTS 检索, 通过指定数据段中一个数据块的编号, 就可以找出 ASM 磁盘中哪个磁盘块持有该数据块的数据。

ASM 磁盘组中的每一个文件通过文件编号以及数据库实体 (incarnation) 进行标识。这两个数据都是视图 V\$ASM_ALIAS 中文件名的一部分。V\$ASM_ALIAS 是建立在固定表 X\$KFFIL 上, 而不是 X\$KFFXP。实际上, 没有一个 V\$固定视图是基于 X\$KFFXP 的。使用命令行工具 ASMCMD 可以像访问文件系统那样访问 ASM 磁盘组。当执行命令 `ls -l` 时, ASMCMD 就会显示文件编号和数据库实体。在下面的例子中, 别名 `spfileTEN.ora` 就指向文件编号为 265、数据库实体为 632700769 的 ASM 文件:

```
$ asmcmd
ASMCMD> cd DG/TEN
ASMCMD> ls -l spfileTEN.ora
Type Redund Striped Time Sys Name
N spfileTEN.ora => +DG/TEN/PARAMETERFILE/spfile.265.63
2700769
```

X\$KFFXP 同样包含了文件编号以及数据库实体。因为该视图必须跟踪所有被条带化和镜像了的数据区，所以它包括一些其他持有磁盘组、磁盘、ASM 文件等信息的列。表 12-1 列出了表 X\$KFFXP 中的列：

表12-1 X\$ KFFXP表中的列

列 名	含 义
ADDR	地址
INDX	行索引
INST_ID	实例标识符（1代表单个实例，1..n代表RAC）
GROUP_KFFXP	磁盘组数目，对应于V\$ASM_DISKGROUP.GROUP_NUMBER
NUMBER_KFFXP	文件数据，对应于V\$ASM_FILE.FILE_NUMBER
COMPOUND_KFFXP	混合索引，对应于V\$ASM_FILE.COMPOUND_INDEX
INCARN_KFFXP	数据库实体，对应于V\$ASM_FILE.INCARNATION
PXN_KFFXP	未知的
XNUM_KFFXP	数据区编号
LXN_KFFXP	逻辑数据区编号（0代表主数据区，1代表镜像副本）
DISK_KFFXP	磁盘数，对应于V\$ASM_DISK.DISK_NUMBER
AU_KFFXP	设备中的偏移量，为分配单元大小的倍数（V\$ASM_DISKGROUP.ALLOCATION_UNIT_SIZE）
FLAGS_KFFXP	未知的
CHK_KFFXP	未知的

## 12.2 抢救 SPFILE 文件

大概我不是唯一一个不得不面对服务器参数文件（SPFILE）遭受 ALTER SYSTEM 修改这一问题的 DBA，它会使得实例启动时加载不适合的 SPFILE 文件。在启动时它会以清单的形式打印一些不同的错误信息。如下是一个例子：

```
SQL> STARTUP
ORA-00821: Specified value of sga_target 356M is too small, needs to be at least
564M
ORA-01078: failure in processing system parameters
```

SPFILE 是一个不可修改的二进制文件，除非在一个正在运行的实例上执行 ALTER SYSTEM 命令。但是，在这种情况下实例就不会以 SPFILE 启动。如果 SPFILE 存储在文件系统中，可以使用文本编辑器去除其中非 ASCII 字符，然后 SPFILE 就可以转换为一个文本参数文件。然而，如果 SPFILE 是一个 ASM 文件并且你没有它最近的 RMAN 备份，那就束手无策了。当然，如果你知道怎么从 ASM 磁盘中直接获取 SPFILE 就另当别论了，你即将在接下来的内容中学习它。

对一个 ASM 实例运行以下的 5 路联结操作，它会获取本可以用 UNIX 命令 dd^①从 ASM 存储来检索 SPFILE 的所有信息。该查询接收 SPFILE 的名称作为输入（不是磁盘组中分配给 SPFILE 参

① 在 Windows 平台，需要在 Cygwin 下执行 dd 命令。

数的绝对路径)。下面的例子就来自一个采用外部冗余的磁盘组。因此，对于镜像仅仅只有主分配单元，没有次级分配单元（second allocation unit）。

```
SQL> SELECT a.name, a.group_number AS group#, a.file_number AS file#,
f.bytes, allocation_unit_size AS au_size, au_kffxp AS au#, decode(x.lxn_kffxp, 0,
'PRIMARY', 1, 'MIRROR') AS type,
d.failgroup AS failgrp, d.path
FROM v$asm_alias a, v$asm_file f, x$kffxp x, v$asm_disk d, v$asm_diskgroup dg
WHERE lower(a.name)=lower('spfileTEN.ora')
AND a.group_number=f.group_number
AND a.file_number=f.file_number
AND f.group_number=x.group_kffxp
AND f.file_number=x.number_kffxp
AND x.disk_kffxp=d.disk_number
AND f.group_number=dg.group_number;
NAME          GROUP# FILE# BYTES AU_SIZE AU# TYPE   FAILGRP PATH
-----
spfileTEN.ora      1   265   2560 1048576 240 PRIMARY SDA9    /dev/oracleasm/disks
                                         /SDA9
```

对于正常冗余（normal redundancy），也就是 ASM 双向镜像，你就可以看到一个主分配单元和一个次级分配单元，它们被分配到属于不同失效组的不同磁盘。

```
NAME          GROUP# FILE# BYTES AU_SIZE AU# TYPE   FAILGRP PATH
-----
spfileten.ora    1   257   2560 1048576 23 PRIMARY DC1     /dev/rlv_asm_dc1_dsk09
spfileten.ora    1   257   2560 1048576 26 MIRROR  DC2     /dev/rlv_asm_dc2_dsk09
```

查询输出（X\$KFFXP.AU_KFFXP）中，列 AU# 的值是从 ASM 磁盘中第一个数据块开始的分配单元偏移量。一个 ASM 分配单元默认大小为 1 048 576 字节（1 MB）。考虑到这一点，一个大小为 2 560 字节的 SPFILE 文件，存储在偏移量为 240 的分配单元中，则可以使用如下命令管道（pipeline）进行检索：

```
$ dd if=/dev/oracleasm/disks/SDA9 bs=1048576 skip=240 count=1 \
| dd bs=2560 count=1 | strings > spfile.txt
1+0 records in
1+0 records out
1+0 records in
0+0 records out
```

命令 dd 的选项中，bs 代表数据块大小，skip 代表从文件开头跳过的数据块数量，count 代表读取的数据块数量。命令管道最后的 strings 命令则用来删除所有非 ASCII 的字符。因此，我们最后得到一个可编辑的 spfile.txt 文件。命令 head 和 tail 则用来显示文件的开头与结尾，以确认 SPFILE 全部内容都被检索到了。

```
$ head -3 spfile.txt
*.audit_file_dest='/opt/oracle/obase/admin/TEN/adump'
*.background_dump_dest='/opt/oracle/obase/admin/TEN/bdump'
*.compatible='10.2.0.3.0'
$ tail -3 spfile.txt
TEN1.undo_tablespace='UNDOTBS1'
```

```
TEN2.undo_tablespace='UNDOTBS2'
*.user_dump_dest='/opt/oracle/obase/admin/TEN/udump'
```

## 12.3 映射数据段到 ASM 存储

现在，我们已经成功地完成了准备工作，我们已经准备好去完成映射数据库段中一个数据块到相应 ASM 磁盘的艰巨任务。除了在 ASM 磁盘中映射一个正确的分配单元，这还需要在分配单元中找到正确的数据块。一个单独的 ASM 分配单元可能包含来自多个数据段的数据块。请记住，带有 AUTOALLOCATE 选项的本地管理表空间中最小的数据区大小为 64 KB。下面的案例使用表空间 USERS 中的 LOCATIONS 数据段作为例子。我们重复利用上一节的查询，但是这一次传入表空间 USERS 中单独的数据文件作为输入。

```
SQL> SELECT x.xnum_kffxp AS extent, a.group_number AS grp#, a.file_number AS file#,
f.bytes, allocation_unit_size AS au_size, au_kffxp AS au,
decode(x.lxn_kffxp, 0, 'PRIMARY', 1, 'MIRROR') AS type,
d.failgroup AS failgrp, d.path
FROM v$asm_alias a, v$asm_file f, x$kffxp x, v$asm_disk d, v$asm_diskgroup dg
WHERE lower(a.name)=lower('USERS.263.628550085')
AND a.group_number=f.group_number
AND a.file_number=f.file_number
AND f.group_number=x.group_kffxp
AND f.file_number=x.number_kffxp
AND x.disk_kffxp=d.disk_number
AND f.group_number=dg.group_number
ORDER BY x.xnum_kffxp;
```

文件 USERS.263.628550085 的大小是 175 MB。在下面代码清单所示的查询结果中，显示了 ASM 是怎样在 5 个磁盘间对文件进行条带化的。默认情况下，只要磁盘组有足够的磁盘，ASM 就会在 8 个磁盘上进行条带化（参数 ASM_STRIPEWIDTH）。

EXTENT	GRP#	FILE#	BYTES	AU_SIZE	AU	TYPE	FAILGRP	PATH
0	1	263	183508992	1048576	123	PRIMARY	SDA11	/dev/oracleasm/disks/SDA11
1	1	263	183508992	1048576	126	PRIMARY	SDA6	/dev/oracleasm/disks/SDA6
2	1	263	183508992	1048576	123	PRIMARY	SDA9	/dev/oracleasm/disks/SDA9
3	1	263	183508992	1048576	125	PRIMARY	SDA10	/dev/oracleasm/disks/SDA10
4	1	263	183508992	1048576	125	PRIMARY	SDA5	/dev/oracleasm/disks/SDA5
5	1	263	183508992	1048576	124	PRIMARY	SDA11	/dev/oracleasm/disks/SDA11
6	1	263	183508992	1048576	127	PRIMARY	SDA6	/dev/oracleasm/disks/SDA6
7	1	263	183508992	1048576	124	PRIMARY	SDA9	/dev/oracleasm/disks/SDA9
8	1	263	183508992	1048576	126	PRIMARY	SDA10	/dev/oracleasm/disks/SDA10
9	1	263	183508992	1048576	126	PRIMARY	SDA5	/dev/oracleasm/disks/SDA5
10	1	263	183508992	1048576	125	PRIMARY	SDA11	/dev/oracleasm/disks/SDA11

接下来的步骤包括从所有这些 ASM 文件区中定位正确的数据库块。我们可以使用包 DBMS_ROWID 从数据段检索数据块，其中保留了某一数据行。DBMS_ROWID.ROWID_BLOCK_NUMBER 返回的数据块编号随后必须被映射到 ASM 文件中正确的数据区中。我们将使用表 LOCATIONS 的数据

行作为例子，该表中包含城市名 Los Angeles。

```
SQL> SELECT f.file_name, e.relative_fno AS rel_fno,
e.extent_id, e.block_id AS "1st BLOCK", e.blocks,
dbms_rowid.rowid_block_number(l.ROWID) AS block_id
FROM locations l, dba_extents e, dba_data_files f
WHERE l.city='Los Angeles'
AND f.relative_fno=e.relative_fno
AND e.relative_fno=dbms_rowid.rowid_relative_fno(l.ROWID)
AND dbms_rowid.rowid_block_number(l.ROWID)
BETWEEN e.block_id AND e.block_id + e.blocks -1;
FILE_NAME                                REL_FNO EXTENT_ID 1st BLOCK BLOCKS BLOCK_ID
-----
+DG/ten/datafile/users.263.628550085      4         0    21793      8    21798
```

LOCATIONS.CITY='Los Angeles'的数据行处于文件 4、数据库块 21 798 中（见查询结果的列 BLOCK_ID）。数据块 21 798 位于数据区 0 中，从数据块 21 793 开始。数据区 0 总共有 8 个数据块。文件名 4 表明这是一个 ASM 文件。下一步，我们需要找到包含该数据块的 ASM 分配单元。下面的查询把数据块的大小为 8 192 字节以及 ASM 分配单元大小为 1 048 576 字节纳入考虑范围。数据段中的偏移是以数据块作为度量的（对应于 DB_BLOCK_SIZE 或表空间的数据块大小），而 ASM 磁盘则是以分配单元作为度量的。ASM 文件中包含数据块的数据区如下所示：

$$\text{floor} \frac{21\,798 * 8\,192}{1\,048\,576} = 170$$

下面的查询返回 ASM 磁盘以及在磁盘内寻找的分配单元：

```
SQL> SELECT x.xnum_kffxp AS extent, a.group_number AS grp#, a.file_number AS file#,
f.bytes, allocation_unit_size AS au_size, au_kffxp AS au#,
decode(x.lxn_kffxp, 0, 'PRIMARY', 1, 'MIRROR') AS type, d.failgroup, d.path
FROM v$asm_alias a, v$asm_file f, x$kffxp x, v$asm_disk d, v$asm_diskgroup dg
WHERE lower(a.name)=lower('USERS.263.628550085')
AND a.group_number=f.group_number
AND a.file_number=f.file_number
AND f.group_number=x.group_kffxp
AND f.file_number=x.number_kffxp
AND x.disk_kffxp=d.disk_number
AND f.group_number=dg.group_number
AND x.xnum_kffxp=trunc(21798*8192/1048576);
EXTENT GRP# FILE#      BYTES AU_SIZE AU# TYPE      FAILGROUP PATH
-----
170     1     263 183508992 1048576 238 PRIMARY SDA9      /dev/oracleasm/disks/SDA9
```

最终，我们需要在 1 MB 大小的分配单元内找到正确的数据库块。下面的公式用来计算一个数据块从 ASM 磁盘开头的偏移量：

$$\frac{AU\# * AUSize + Block\# * DB_BLOCK_SIZE - ASMExtent\# * AU_Size}{DB_BLOCK_SIZE}$$

在上面这个公式中，AU#是一个 ASM 磁盘（对应于 X\$KFFXP.AU_KFFXP）中的分配单元偏移

量, AU size 是 ASM 分配单元的大小, BLOCK# 是一个数据库段中的数据块编号, DB_BLOCK_SIZE 是数据库块大小 (或者表空间块大小), ASM Extent# 是 ASM 文件中数据区的编号。这样得到的偏移量是数据库 (或者表空间) 块大小的倍数。输入上述查询结果的数据到该公式中, 得到:

$$\frac{238 * 1\,048\,576 + 21\,798 * 8\,192 - 170 * 1\,048\,576}{8\,192} = 30\,502$$

包含有字符串 “Los Angeles” 的数据库块距离磁盘头部偏移了 30 502 个数据块, 并且每一个数据块的大小为 8 192 字节。为了提取这个数据块, 我们需要使用 dd 命令, 如下所示:

```
$ dd if=/dev/oracleasm/disks/SDA9 bs=8192 skip=30502 count=1 | strings | \
grep "Los Angeles"
1+0 records in
1+0 records out
Los Angeles
```

当然, 我没有人为地去创建一个表并且在每一行都包含字符串 “Los Angeles”, 相邻的数据块是不包含这个字符串的。

```
$ dd if=/dev/oracleasm/disks/SDA9 bs=8192 skip=30501 count=1 | strings | \
grep "Los Angeles"
1+0 records in
1+0 records out
$ dd if=/dev/oracleasm/disks/SDA9 bs=8192 skip=30503 count=1 | strings | \
grep "Los Angeles"
1+0 records in
1+0 records out
```

图 12-1 描绘的是数据库段 LOCATIONS 中数据区 0 的数据块 21 978 到 ASM 磁盘分配单元 238 的映射。在分配单元 238 中该数据块与第一个数据块相差 38。因为数据段的区由 8 个数据块组成, ASM 分配单元包含的其余的区则可能属于不同的数据段。

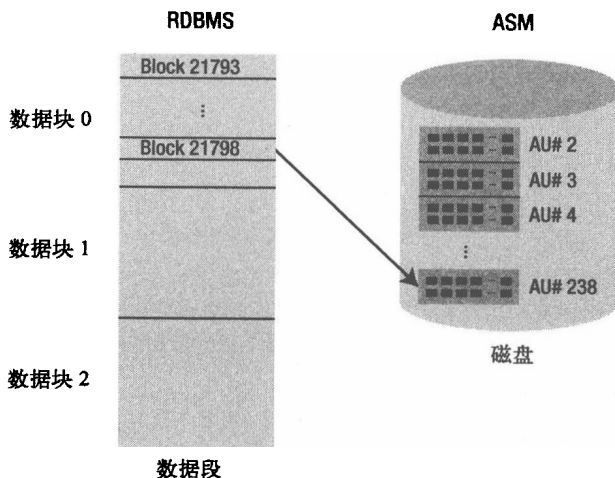


图 12-1 数据库段中数据块到 ASM 磁盘分配单元中数据块的映射

# Part 5

## 第五部分

# SQL 语句

### 本部分内容

- 第 13 章 ALTER SESSION/SYSTEM SET EVENTS
- 第 14 章 ALTER SESSION SET CURRENT_SCHEMA
- 第 15 章 ALTER USER IDENTIFIED BY VALUES
- 第 16 章 SELECT FOR UPDATE SKIP LOCKED

# ALTER SESSION/SYSTEM SET EVENTS

ALTER SESSION SET EVENTS 和 ALTER SYSTEM SET EVENTS 在 Oracle9i 和 Oracle10g 的 *Oracle Database SQL Reference* 手册以及 *Oracle Database SQL Language Reference 11g Release 1* 中都没有文档说明。尽管 *Oracle Database Performance Tuning Guide 10g Release 2* 手册中包括针对最有名的 10046 事件在第 8 级别的 ALTER SESSION 语句，然而，关于如何关闭 10046 事件，以及 10046 事件中非常有用的其他级别，也没有在文档中说明。此外，还有数以百计的能用 ALTER SESSION 和 ALTER SYSTEM 设置的有用事件同样没有在文档中说明。

直到今天，对一个拥有 ALTER SESSION 权限但是没被授予 DBA 角色的数据库用户，ALTER SESSION SET EVENTS 是在他自己的会话中启动扩展 SQL 跟踪的唯一方法，通过这种方法将等待事件和/或绑定变量包含在 SQL 跟踪文件中。当某种 ORA-*nnnnn* 错误发生时，ALTER SESSION SET EVENTS 和 ALTER SYSTEM SET EVENTS 也都可被用来请求诊断转储。

## 13.1 跟踪你自己的会话

一般来说每个 DBA 和开发人员都了解 ALTER SESSION SET SQL_TRACE=TRUE 这条 SQL 语句，它用来创建一个跟踪文件，记录执行该命令的会话中的所有 SQL 语句。实际上它记录的内容远不止于此，例如会话标识信息、应用程序插桩条目、时间戳等，详情请参阅第 24 章。但是它缺少等待事件以及绑定变量的信息^①，两者对可靠的性能诊断和性能问题再现是大有裨益的。

Oracle10g 是第一个对启用绑定变量跟踪的接口给出文档说明的版本。DBMS_MONITOR 让该功能成为可能，它还支持等待事件的跟踪。但是 DBMS_MONITOR 的执行权限仅仅赋给了角色 DBA (见 \$ORACLE_HOME/rdbms/admin/dbmsmntr.sql)。未在文档中说明的 DBMS_SYSTEM 包也同样如此，它能够设置 10046 等事件。DBMS_SUPPORT 包也没有在文档中说明，但是它也提供了对等待事件和绑定变量的跟踪，此外还包括对那些连 DBA 都没有执行权限的数据库调用（解析、执行、获取）的跟踪。对启用绑定变量跟踪的接口给出文档说明的的功能都非常强大，普通的数据库用户都无权执行它们。还有一条同样重要的 SQL*Plus 命令 ORADEBUG，仅仅 SYS 拥有其执行权限。因

① 当使用 DBMS_SESSION.SET_SQL_TRACE(TRUE)打包程序启动 SQL 跟踪时，都不会有等待事件和绑定变量的信息。



此,一般来说,那些精通性能诊断的数据库用户,如果想在麻烦 DBA 的情况下创建跟踪文件,即使不是因为 ALTER SESSION SET EVENTS,通常也会不知所措。

## 13.2 ALTER SESSION SET EVENTS

如果想在应用程序中加入自我跟踪功能,并且在登录触发器中启用 SQL 跟踪,ALTER SESSION SET EVENTS 再合适不过了。通过自我跟踪,我的用意是应用程序启用 SQL 跟踪的能力能够依赖于一个环境变量或者图形用户界面中的一个菜单项。启停事件的语法如下所示:

```
ALTER SESSION SET EVENTS 'event_number TRACE NAME CONTEXT [FOREVER,] LEVEL lvl'
```

这类事件是由 event_number 这个整数决定,而通常用来控制详情的事件级别则用 lvl 这个整数来设置。通过包括 FOREVER 关键字,事件会一直保留,但是如果没有它,事件仅在瞬间启动。通常,事件必须在较长一段时间内保持开启状态,因此,FOREVER 几乎总处于使用状态。举个例子,如果你要执行 ALTER SESSION SET EVENTS '10046 trace name context level 1',随后生成的 SQL 跟踪文件会记录 ALTER SESSION 语句,当结束时停止跟踪。但是这并不是非常有用。相反,你可以使用 ALTER SESSION SET EVENTS '10046 trace name context forever,level 12' 来追踪后续语句的等待事件和绑定变量。

关闭事件的语法如下所示:

```
ALTER SESSION SET EVENTS 'event_number trace name context off'
```

通常的做法是首先启用 10046 事件,然后执行一条需要优化的代码路径,最后关闭 10046 事件。接着可使用 TKPROF 或者 ESQTRCPROF 扩展 SQL 跟踪 profiler 工具(书中关于该工具的介绍见第 27 章)来做性能诊断。如果你怀疑或已经知道优化器为某些跟踪语句选择了次优的执行计划,你也应该用和 10046 事件同样的语法和相同的时间间隔来启动 10053 事件。10053 事件会指示基于成本的优化器把它的决策过程日志写入跟踪文件。级别 1 对于该事件来说是正确的级别。

通过使用 ALTER SESSION SET EVENTS,也可以获得所有基于名称的转储,如系统状态、库缓存、堆、控制文件,甚至更多。ORADEBUG DUMPLIST 这条 SQL*Plus 命令打印所有可用的转储清单。前段时间, Pete Finnigan 指出,库高速缓存转储可用于收集 ALTER USER 语句中使用的密码,这对 Oracle8 来说不是一个好消息,但是这个问题在 Oracle10g 里已经解决,从下面的测试可得到证明:

```
SQL> ALTER USER hr IDENTIFIED BY secret;
User altered.
SQL> ALTER SESSION SET EVENTS 'immediate trace name library_cache level 10';
Session altered.
```

生成的跟踪文件用* (星号) 代替了密码。

```
BUCKET 76429:
LIBRARY OBJECT HANDLE: handle=6c15e8e8 mutex=6C15E99C(1)
name=ALTER USER hr IDENTIFIED BY *****
```

可能还有其他类似的问题,让你觉得赋予 ALTER SESSION 权限很危险。来自 Hotsos® 的免费 ILO

① ILO 可以从 <http://sourceforge.net/projects/hotsos-ilo> 上下载。

(Instrumentation Library for ORACLE)，包括HOTSOS_ILO_TIMER，可用来启用或禁用对数据库调用、等待事件和绑定变量的跟踪，而不用为事件或转储提供权限。但是，请务必注意，你应该回收HOTSOS_SYSUTIL上、随ILO安装的PUBLIC执行权限，否则任何数据库用户可以用WRITE_TO_ALERT程序写入警报日志。当然，你也可以围绕DBMS_SUPPORT和DBMS_MONITOR构建自己的包装器，但是ILO也为应用程序性能评测（模块、动作和客户端标识符，见第 28 章）提供了有趣的功能。

### 13.3 ALTER SYSTEM SET EVENTS

ALTER SYSTEM SET EVENTS 是和 ALTER SESSION SET EVENTS 对立的实例级别，它为所有未来的数据库会话设置事件。通过这种方式设置的事件在实例重启时不会继续存在。如果每次启动实例时都必须设置事件，那么就可以使用 EVENT 参数（参见 2.2.1 节）。

这是一个在实例级别使用事件的场景，也许你永远也不希望面对。让我们假设在表的段有一些数据块损坏了。不幸的是，数据库处于无归档日志模式，所以不能选择数据文件，或者数据块级的恢复。导入最近的导出转储会导致数据丢失。以下未在文档中说明的事件会在全表扫描期间跳过损坏的数据块^①。

```
$ oerr ora 10231
10231, 00000, "skip corrupted blocks on _table_scans_"
// *Cause:
// *Action: such blocks are skipped in table scans, and listed in trace files
```

你也许能够挽救大部分修改，因为最近的导出是表级别导出，然而，没有其他办法能让 exp 导出实用程序在连接之后设置事件。这时 ALTER SYSTEM SET EVENTS 能派上用场。通过使用 DBMS_SYSTEM.READ_EV，我们可以确认的是，这样设置的事件确实已经在新的数据库会话中了。

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> VARIABLE lev NUMBER
SQL> SET AUTOPRINT ON
SQL> ALTER SYSTEM SET EVENTS '10231 trace name context forever, level 10';
System altered.
SQL> EXECUTE sys.dbms_system.read_ev(10231, :lev)
      LEV
-----
      0
SQL> CONNECT / AS SYSDBA
Connected.
SQL> EXECUTE sys.dbms_system.read_ev(10231, :lev)
      LEV
-----
     10
```

10231 事件在运行 ALTER SYSTEM SET EVENTS 的数据库会话中没有被启用。在启动一个新的数

① 在相关文档中说明的 DBMS_REPAIR.SKIP_CORRUPT_BLOCKS 程序在 Oracle9i 和后续版本中提供了同样的功能。

数据库会话后，该事件被设置。在这一点上，导出实用程序可被启动用于挽救数据。当它完成时，该事件就可关闭了。

```
SQL> ALTER SYSTEM SET EVENTS '10231 trace name context off';
System altered.
SQL> CONNECT / AS SYSDBA
Connected.
SQL> EXECUTE sys.dbms_system.read_ev(10231, :lev)
      LEV
-----
      0
```

## 13.4 ALTER SESSION/SYSTEM SET EVENTS 和诊断转储

ALTER SESSION SET EVENTS 和 ALTER SYSTEM SET EVENTS 也可被发生 ORA-*nnnnn* 错误的事件用来请求某些诊断转储。生成的跟踪文件可发送给 Oracle Support 分析。通常，这个跟踪文件用来记载当错误发生时的实例状态。因此，这证明在查明缺陷的原因上它们还是有用的。当发生 ORA-*nnnnn* 错误时请求转储的语法如下所示：

```
ALTER {SESSION|SYSTEM} SET EVENTS 'error_code TRACE NAME dump_name LEVEL lvl'
```

可使用 SQL*Plus 中的 ORADEBUGDUMPLIST 命令来打印有用的转储清单。接下来是发生 ORACLE 错误的事件中禁用转储的语法：

```
ALTER {SESSION|SYSTEM} SET EVENTS 'error_code TRACE NAME dump_name OFF'
```

让我们假设有几个会话遇到了 ORA-04031 错误，例如 ORA-04031: unable to allocate 4096 bytes of shared memory ("java pool","unknown object","joxs heap","Intern")。若要在任何数据库会话每次收到 ORACLE 4031 错误创建一个堆转储，你可以运行下面的 ALTER SYSTEM 语句^①：

```
SQL> ALTER SYSTEM SET EVENTS '4031 trace name heapdump level 536870914';
System altered.
```

这个事件的有效级别是 1、2、3、32、536870912，以及 1 到 32 之间的一个级别。在其他级别中，生成的跟踪文件包含调用栈跟踪信息和 SQL 语句，它是在错误发生时激活的。

```
ioc_allocate (size: 4096, heap name: *** SGA ***, flags: 110009) caught 4031
*** 2007-11-12 16:20:54.359
ksedmp: internal or fatal error
ORA-04031: unable to allocate 4096 bytes of shared memory ("java pool",
"unknown object","joxs heap","Intern")
Current SQL statement for this session:
SELECT SYNNAM, DBMS_JAVA.LONGNAME(SYNNAM), DBMS_JAVA.LONGNAME(SYNTAB),
TABOWN, TABNODE, PUBLIC$, SYNOWN, SYNOWNID, TABOWNID, SYNOBJNO
FROM SYS.EXU9SYN WHERE SYNOWNID = :1 ORDER BY SYNTIME
----- Call Stack Trace -----
calling      call      entry      argument values in hex
location      type      point      (? means dubious value)
...

```

① 当 ORA-4031 错误被引发，Oracle10g 会自动写入跟踪文件。

在收集到足够的诊断数据后，可使用以下语句来禁用转储事件：

```
SQL> ALTER SYSTEM SET EVENTS '4031 trace name heapdump off';
System altered.
```

## 13.5 立即转储

使用下面的语法，也可以请求立即采取转储：

```
ALTER SESSION SET EVENTS 'IMMEDIATE TRACE NAME dump_name LEVEL lvl'
```

如果你怀疑一个实例正挂起，可能需要类似这样的系统状态转储：

```
SQL> ALTER SESSION SET EVENTS 'immediate trace name systemstate level 10';
Session altered.
```

系统状态转储的头部如下所示：

```
SYSTEM STATE
```

```
-----
```

System global information:

processes: base 6ED426BC, size 50, cleanup 6ED47FF4

allocation: free sessions 6ED60D60, free calls 00000000

control alloc errors: 0 (process), 0 (session), 0 (call)

PMON latch cleanup depth: 0

seconds since PMON's last scan for dead processes: 21

system statistics:

166 logons cumulative

18 logons current

1082648 opened cursors cumulative

25 opened cursors current

# ALTER SESSION SET CURRENT_SCHEMA

ALTER SESSION SET CURRENT_SCHEMA 在 Oracle8 和之前的版本都没有在文档中说明，但它在 Oracle8i、Oracle9i 和 Oracle10g 的 SQL 参考手册中部分有文档说明。存在一些同某些数据库对象相关的未在文档中说明的限制，这些限制应用于高级队列、SQL 语句 RENAME 和数据库链接。CURRENT_SCHEMA 会话参数提供了在某个模式中对对象进行操作的便捷方式，而不是在当前用户的模式中进行操作，并且这种方式不需要对对象进行模式名授权。另外，它不会影响会话的权限。

## 14.1 特权用户与模式用户

ORACLE DBMS 区分了模式用户标识和特权用户标识，特权用户标识决定了哪些权限可被用来创建或访问数据库对象，而模式用户标识则提供了语句解析和执行的上下文环境。登录 DBMS 实例后，拿数据库 SYSTEM 用户举例，分配给 SYSTEM 用户的特权生效，SYSTEM 模式希望得到任何涉及的未授权的数据库对象。有人可能会说，默认情况下，SYSTEM 用户的当前模式是 SYSTEM 模式。当使用的数据库对象被 SYSTEM 模式名授权后，DBMS 作出反应。但是因为 SYSTEM 用户的默认当前模式是 SYSTEM，DBMS 不需要模式名的授权。

用户名和当前模式不一定相同。这从 V\$SQL 视图的 PARSING_USER_ID, PARSING_SCHEMA_ID, PARSING_SCHEMA_NAME 列和 SQL 跟踪文件可以很明显看出，它们可用于在 PARSING IN CURSOR 条目中区分用户标识 (uid) 和逻辑标识 (lid)。

这个功能和 UNIX 系统很相似，它可用来区别实际用户标识和有效用户标识。UNIX 中拥有最高权限的 root 用户可以通过 su 命令切换到另一个用户。随后产生的进程便同时拥有切换用户的有效用户标识和 root 的实际用户标识。

在足以改变解析模式标识的情况下，ALTER SESSION SET CURRENT_SCHEMA 对临时更改密码以及随后使用 ALTER USER IDENTIFIED BY VALUES 重置来说，是一种入侵性更小的替代方案（见第 15 章）。

以下的查询是通过查找 V\$SQL 视图从共享池中找出 HR 用户运行的 SQL 语句。请注意在那些解析的用户标识和解析的模式标识不同的地方没有 SQL 语句。

```
SQL> SELECT s.parsing_user_id, u.username, s.parsing_schema_id,
s.parsing_schema_name, substr(s.sql_text,1,15) sql_text
FROM v$sql s, dba_users u
WHERE s.parsing_user_id=u.user_id
AND s.parsing_schema_name='HR';
PARSING_USER_ID USERNAME PARSING_SCHEMA_ID PARSING_SCHEMA_NAME SQL_TEXT
-----
38 HR 38 HR SELECT USER FRO
38 HR 38 HR BEGIN DBMS_OUTP
```

在以 SYSTEM 身份运行 IMPORT 实用程序 (imp) 并向 HR 模式导入一些表格之后, 重复之前在 V\$SQL 的查询, 结果显示有些 SQL 语句的解析用户标识是 SYSTEM, 而解析模式标识却是 HR。

```
PARSING_USER_ID USERNAME PARSING_SCHEMA_ID PARSING_SCHEMA_NAME SQL_TEXT
-----
38 HR 38 HR BEGIN sys.dbm
38 HR 38 HR ALTER SESSION S
38 HR 38 HR ALTER SESSION S
5 SYSTEM 38 HR BEGIN SYS.DBMS
5 SYSTEM 38 HR BEGIN SYS.DBMS
```

如果我们跟踪 IMPORT 实用程序涉及的 SQL 语句, 我们会在 SQL 跟踪文件中找到一条 ALTER SESSION SET CURRENT_SCHEMA 语句。可使用 TRACE=TRUE 这个未在文档中说明的命令行选项, 在导入会话中设置 SQL_TRACE=TRUE。

```
$ imp trace=true
Import: Release 10.2.0.1.0 - Production on Tue Jun 19 10:09:40 2007
Copyright (c) 1982, 2005, Oracle. All rights reserved.
Username:
```

顺便提一下, 数据转储导出 (expdp) 和导入 (impdp) 也拥有相同的未在文档中说明的 TRACE 转换方法, 产生的 SQL 跟踪文件包含以下行:

```
PARSING IN CURSOR #5 len=38 dep=0 uid=5 oct=42 lid=5 tim=63968187622 hv=886929406 ad
='6786940c'
ALTER SESSION SET CURRENT_SCHEMA= "HR"
END OF STMT
PARSE #5:c=0,e=523,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=63968187614
EXEC #5:c=0,e=63,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=63968187802
XCTEND rlbk=0, rd_only=1
=====
PARSING IN CURSOR #5 len=113 dep=0 uid=38 oct=13 lid=5 tim=63968189065 hv=0 ad='8eaf
7d4'
CREATE SEQUENCE "LOCATIONS_SEQ" MINVALUE 1 MAXVALUE 9900 INCREMENT BY 100
START WITH 3300 NOCACHE NOORDER NOCYCLE
END OF STMT
PARSE #5:c=0,e=711,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=63968189057
```

跟踪文件显示, ALTER SESSION SET CURRENT_SCHEMA="HR" 语句的解析用户标识是 5 (lid=5, SYSTEM), 解析模式标识也是 5 (uid=5)。该 ALTER SESSION 语句把解析的模式名设置为 HR,

这可通过随后的 CREATE SEQUENCE 语句得到证明。CREATE SEQUENCE 语句被解析时的解析模式标识是 38 (uid=38)，对应的正是 HR 模式。

## 在外模式中创建数据库对象

假设一位使用 HR 账户的软件开发者需要大量的新表用于测试工作，HR 数据库用户没有 CREATE TABLE 权限，因为开发者仅仅允许使用 HR 模式中存在的表。因此，开发者不能自己在 HR 模式中创建表。开发者发送一个用来创建表的脚本，但是该脚本中的数据库对象名并不是以模式名作前缀。

该情景是使用 ALTER SESSION SET CURRENT_SCHEMA 语句的一个例子。如果没有该语句，必须选择以下解决方案中的一个。

- DBA 必须把 CREATE TABLE 临时授权给 HR 数据库用户，使得开发人员可以创建自己的表。
- DBA 必须让开发人员为脚本中的每个对象名加上 HR 模式名前缀，这样 DBA 才能运行该脚本。

DBA 可使用 CURRENT_SCHEMA 来保证运行的脚本没有变化，下面是一条在 ALTER SESSION SET CURRENT_SCHEMA 语句后执行的 CREATE TABLE 语句。这个例子说明了该数据库对象是在解析模式标识下创建的，而不是特权模式标识。

```
SQL> SHOW USER
USER is "SYSTEM"
SQL> ALTER SESSION SET current_schema=hr;
Session altered.
SQL> CREATE TABLE country (
    country_id char(2),
    country_name varchar2(40),
    region_id number,
    CONSTRAINT pk_country PRIMARY KEY (country_id)
);
Table created.
SQL> SELECT owner, object_type
FROM dba_objects
WHERE object_name IN ('COUNTRY', 'PK_COUNTRY');
OWNER OBJECT_TYPE
-----
HR      TABLE
HR      INDEX
```

数据库对象在 HR 模式中被创建。当使用 ALTER SESSION SET CURRENT_SCHEMA 时，跟某些数据库对象相关的限制将会应用，但这些限制未在文档中说明。

## 14.2 ALTER SESSION SET CURRENT_SCHEMA 限制

切换到不同的解析模式标识不能用于高级队列。此外，它不可能在外模式中创建私有数据库链接。它可以在外模式中创建存储概要。接下来几节将详细介绍这些限制。

### 14.2.1 高级队列

ALTER SESSION SET CURRENT_SCHEMA 对使用 AQ (Advanced Queuing, 高级队列) 包 DBMS_AQADM 和 DBMS_AQ 做名称解析没有任何影响。文档中提到, 同义词不能用于高级队列中。因此, 结合 PL/SQL 包 DBMS_AQ 和 DBMS_AQADM 是在外模式中使用队列表和队列的唯一方法, 就是对带该模式名的对象名授权。同样, 只要当前模式中没有同义词, 外模式中的表就需要被授权。下在的代码示例假设第 16 章创建的队列驻留在 NDEBES 模式中。

```
$ sqlplus / as sysdba
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
SQL> SELECT owner FROM dba_queues WHERE name='CAUGHT_IN_SLOW_Q_AGAIN';
OWNER
-----
NDEBES
SQL> ALTER SESSION SET CURRENT_SCHEMA=ndebes;
Session altered.
SQL> EXEC dbms_aqadm.stop_queue('CAUGHT_IN_SLOW_Q_AGAIN')
BEGIN dbms_aqadm.stop_queue('CAUGHT_IN_SLOW_Q_AGAIN'); END;
*
ERROR at line 1:
ORA-24010: QUEUE SYS.CAUGHT_IN_SLOW_Q_AGAIN does not exist
ORA-06512: at "SYS.DBMS_AQADM_SYS", line 4913
ORA-06512: at "SYS.DBMS_AQADM", line 240
ORA-06512: at line 1
SQL> CONNECT ndebes/secret
Connected.
SQL> EXEC dbms_aqadm.stop_queue('CAUGHT_IN_SLOW_Q_AGAIN')
PL/SQL procedure successfully completed.
```

SYS 执行 STOP_QUEUE 失败的原因不是权限不足。当 SYS 对带正确模式名的队列名授权时, STOP_QUEUE 的执行将不会有任何问题。

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> EXEC dbms_aqadm.stop_queue('NDEBES.CAUGHT_IN_SLOW_Q_AGAIN')
PL/SQL procedure successfully completed.
```

### 14.2.2 RENAME

数据库的表名可用 RENAME 这条 SQL 语句来更改。它是为数不多的不支持带模式名的数据库对象授权的 SQL 语句。使用 ALTER SYSTEM SET CURRENT_SCHEMA 这种限制就不存在了, 因为紧接着的 RENAME 语句将会导致 ORA-03001 错误。

```
SQL> SHOW USER
USER is "NDEBES"
SQL> ALTER SESSION SET CURRENT_SCHEMA=hr;
Session altered.
SQL> RENAME employees TO emp;
```



```

RENAME employees TO emp
*
ERROR at line 1:
ORA-03001: unimplemented feature

```

该解决方案是在持有需重命名的表的相同模式中创建存储过程，并且在该过程中将 EXECUTE IMMEDIATE 和 RENAME 结合使用。RENAME 通过存储过程成功完成，因为它是在持有该表的模式的上文环境下执行的。

### 14.2.3 私有数据库链接

要在外模式中创建一个私有数据库链接是不可能的，因为数据库链接不可能以模式名作前缀。这个限制一直存在，即使是使用 ALTER SESSION SET CURRENT_SCHEMA。

```

SQL> CONNECT / AS SYSDBA
Connected.
SQL> ALTER SESSION SET CURRENT_SCHEMA=ndebes;
Session altered.
SQL> CREATE DATABASE LINK lnk CONNECT TO remote_user
IDENTIFIED BY pwd USING 'dbserver1.oradbpro.com';
CREATE DATABASE LINK lnk CONNECT TO remote_user
IDENTIFIED BY pwd USING 'dbserver1.oradbpro.com'
*
ERROR at line 1:
ORA-01031: insufficient privileges

```

该错误信息一看就可以理解，因为毕竟没有如 CREATE ANY DATABASE LINK 这样的特权。如果你真的需要在外模式建立一个数据库链接，那么你可以使用以下存储过程的技巧：

```

CREATE OR REPLACE PROCEDURE ndebes.create_db_link
IS
BEGIN
    EXECUTE IMMEDIATE 'CREATE DATABASE LINK lnk
CONNECT TO remote_user IDENTIFIED BY pwd
USING ''dbserver1.oradbpro.com''';
END;
/
Procedure created.
SQL> EXEC ndebes.create_db_link
PL/SQL procedure successfully completed.
SQL> SELECT owner, db_link, username, host
FROM dba_db_links
WHERE db_link LIKE 'LNK%';
OWNER  DB_LINK                USERNAME    HOST
-----
NDEBES LNK.ORADBP.COM REMOTE_USER dbserver1.oradbpro.com

```

由于存储过程的运行默认需要所有者的权限，所以 CREATE DATABASE LINK 语句被 NDEBES 特权用户和 NDEBES 解析用户执行，以致数据库链接在和过程相同的模式中被创建。类似于数据库链接，CREATE DIRECTORY 语句中的目录名不能以模式名作前缀。但是，所有的目录的所有权归

SYS，因此，目录究竟被哪个用户创建就无关紧要了。

### 14.2.4 存储概要

存储概要可用来为 SQL 语句创建执行计划，这样优化器就可以一直使用这个来自存储概要的计划，而不是在当前环境下优化语句。这也使得在外模式中创建存储概要成为可能，尽管没有 CREATE ANY OUTLINE 特权。

```
SQL> CONNECT system/secret
Connected.
SQL> ALTER SESSION SET CURRENT_SCHEMA=ndebes;
Session altered.
SQL> CREATE OUTLINE some_outline ON
SELECT emp.last_name, emp.first_name, d.department_name
FROM hr.employees emp, hr.departments d
WHERE emp.department_id=d.department_id;
Outline created.
SQL> SELECT node, stage, join_pos, hint FROM dba_outline_hints
WHERE owner='NDEBES'
AND name='SOME_OUTLINE'
ORDER by node, join_pos;
NODE STAGE JOIN_POS HINT
-----
1      1      0 USE_NL(@"SEL$1" "D"@"SEL$1")
1      1      0 LEADING(@"SEL$1" "EMP"@"SEL$1" "D"@"SEL$1")
1      1      0 OUTLINE_LEAF(@"SEL$1")
1      1      0 ALL_ROWS
1      1      0 IGNORE_OPTIM_EMBEDDED_HINTS
1      1      0 OPTIMIZER_FEATURES_ENABLE('10.2.0.1')
1      1      1 FULL(@"SEL$1" "EMP"@"SEL$1")
1      1      2 INDEX(@"SEL$1" "D"@"SEL$1" ("DEPARTMENTS"."DEPARTMENT_ID"))
```

NDEBES 用户是该概要的所有者，概要由那些详细描述语句执行计划的提示构成。

# ALTER USER IDENTIFIED BY VALUES

`ALTER USER username IDENTIFIED BY VALUES 'password_hash'` 是一条未在文档中说明的 SQL 语句。它通过导入实用程序在内部使用，用来在数据字典基表 `SYS.USER$` 中存储密码的散列值，而原先是用导出实用程序保存。在 DBA 需要以某一特定用户的身份连接，但又不知道或者不必要知道该用户密码的情况下，恢复保存的密码散列值的功能可以节省时间，并帮助 DBA 摆脱请求密码的烦恼。

`IDENTIFIED BY VALUES` 子句也可以和 `CREATE USER` 一起使用，用来在数据库中创建一个新的用户账号。鉴于潜在的密码足够长且复杂，足以承受强力的密码攻击，`CREATE USER username IDENTIFIED BY VALUES` 可用于那些在不暴露明文密码的情况下创建某些数据库账户的脚本中。在这种情况下，暴力破解方法试图通过字符的所有组合来猜测密码，同时计算每个猜测密码的密码散列值，当发生匹配时停止。据我所知，还没有任何工具能够暴力破解 15 个或更多字符的密码。即使是仅包含字母、数字和下划线（共 38 个字符选择）的 15 个字符构成的密码，也有  $4.9746E+23$  种组合，假设硬件设备每秒能够处理 1 万亿密码，仍然需要超过 15 000 年来尝试。因此，这么长的密码是很难被暴力破解的。

## 15.1 密码游戏

在常规情况下，要求 DBA 在外模式中创建数据库对象，或者调查那些只在使用某账户工作时出现的故障。通常，DBA 不知道以某一特定用户连接的密码，公司的政策甚至不会允许他获取同事的密码，或者任何一位值班人员都不知道密码。在这种情况下，如果 DBA 拥有相关模式的权限，他通常能更快地完成任务。作为一个解决方案，DBA 应该能记录该模式的密码散列值，能更改密码，能使用已更改的密码连接，也能重置密码散列值。本章将介绍通过使用文档中未说明的 SQL 语句 `ALTER USER IDENTIFIED BY VALUES` 实现以上想法的方法。有些任务只能通过连接到有争议的模式来完成，并且本处提到的方法成为解决该问题的唯一途径。而有些例子则只需查询 `V$INDEX_USAGE` 视图或运行不以模式名作为数据库对象前缀的第三方脚本，而不用修改代码。

下面的例子使用拥有 DBA 权限的 `SYSTEM` 用户，以及应用程序模式 `HR`。第一步就是以 DBA 身份连接，并保存当前的未知密码。

```
SQL> CONNECT system
Enter password:
Connected.
SQL> SPOOL pwd.log
SQL> SELECT password FROM dba_users WHERE username='HR';
PASSWORD
-----
2AB46277EE8215C4
SQL> SPOOL OFF
```

pwd.log 文件现在包含了 HR 模式的密码散列值。接下来，编辑 pwd.log，这样它就包含了可用来把密码重置为原始值的 SQL 语句。语法是 ALTER USER username IDENTIFIED BY VALUES 'password_hash'，其中 password_hash 是上面得到的密码散列值。编辑之后，pwd.log 文件就包含了以下行：

```
ALTER USER hr IDENTIFIED BY VALUES '2AB46277EE8215C4';
```

现在密码可能被临时改变。

```
SQL> ALTER USER hr IDENTIFIED BY secret;
User altered.
```

请注意，前面的语句将临时的密码不经加密就发送给 DBMS 实例。如果你不放心的话可以使用 SQL*Plus 命令 PASSWORD，这样在数据字典中存储的密码散列就被改变了。

```
SQL> SELECT password FROM dba_users WHERE username='HR';
PASSWORD
-----
D370106EA83A3CD3
```

下一步，启动一个或多个适用于诊断和/或解决手头问题的应用程序。为了减小间隔同时使更改的密码生效，可以在应用程序连接到 DBMS 实例后立即恢复原来的密码。

```
$ sqlplus hr/secret
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
SQL>
```

现在，在 pwd.log 文件中使用 ALTER USER 命令把密码重置为原始值。

```
SQL> SET ECHO ON
SQL> @pwd.log
SQL> ALTER USER hr IDENTIFIED BY VALUES '2AB46277EE8215C4';
User altered.
```

原来的密码现在已经被恢复，这样就不能再使用临时密码 secret 来连接了。

```
SQL> SELECT password FROM dba_users WHERE username='HR';
PASSWORD
-----
2AB46277EE8215C4
SQL> CONNECT hr/secret
ERROR:
ORA-01017: invalid username/password; logon denied
Warning: You are no longer connected to ORACLE.
```

密码的散列值取决于用户名，也就是说，不同用户使用的相同密码会产生不同的散列值，如下面的例子所示：

```
SQL> CREATE USER U1 IDENTIFIED BY "Rattle And Hum";
User created.
SQL> CREATE USER U2 IDENTIFIED BY "Rattle And Hum";
User created.
SQL> SELECT username, password FROM dba_users WHERE username IN ('U1', 'U2');
USERNAME                                PASSWORD
-----
U1                                       07A31E4964AEAC50
U2                                       31019CA688540357
```

## 15.2 用 ALTER USER IDENTIFIED BY VALUES 锁定账户

如果你按照 Oracle 公司的建议锁定 CTXSYS、MDSYS、XDB、OLAPSYS 等这些内部模式账户，那么攻击者会据此发现用户安装了哪些组件，从而专门利用这些组件的漏洞。究其原因，原来是 ORA-28000: the account is locked 错误把某模式的存在告诉了攻击者。当你使用 ALTER USER IDENTIFIED BY VALUES 设置一个安全性极高的密码散列值时，你可能更倾向于公开账户。多次尝试连接将导致 ORA-01017: invalid username/password; logon denied，这样攻击者就不知道哪些用户名存在。由于不可能为错误的密码散列值指定一个匹配的密码，所以这个账户就被有效地锁定，甚至没有过期的密码。

```
SQL> ALTER USER ctxsys IDENTIFIED BY VALUES 'LOCKED' ACCOUNT UNLOCK;
User altered.
SQL> SELECT password FROM dba_users WHERE username='CTXSYS';
PASSWORD
-----
LOCKED
SQL> CONNECT ctxsys/impossible_to_crack_incorrectly_encoded_password
ERROR:
ORA-01017: invalid username/password; logon denied
```

无论你更喜欢哪种方法，你总可以通过设置 AUDIT_TRAIL=DB 来审计多次失败的连接尝试，并使用 AUDIT CONNECT WHENEVER NOT SUCCESSFUL 来启动对多次连接失败的审计。接下来的查询将导致多次连接尝试失败。

```
SQL> SELECT username, os_username, userhost, terminal, timestamp, returncode
FROM dba_audit_trail
WHERE action_name='LOGON'
AND returncode!=0;
USERNAME OS_USERNAME          USERHOST          TERMINAL  TIMESTAMP          RETURNCODE
-----
CTXSYS   DBSERVER\ndebes        WORKGROUP\DBSERVER DBSERVER  28.09.07 20:22      1017
MDSYS    DBSERVER\ndebes        WORKGROUP\DBSERVER DBSERVER  28.09.07 20:37      28000
```

在安全方面，值得一提的是，建立在像带 DBCA 的 General Purpose or Transaction Processing 这样的种子数据库基础上的 Oracle10g 数据库，包括一个新的未在文档中说明的配置 MONITORING_

PROFILE, 并被分配给了 DBSNMP 用户。该配置使得登录失败的尝试次数不受限制, 然而 DEFAULT 标准配置 (在 Oracle9i 和之前版本中也存在), 使得 Oracle10g 连续 10 次登录失败后就会锁定账户。

```
SQL> SELECT profile, limit FROM dba_profiles
WHERE resource_name='FAILED_LOGIN_ATTEMPTS';
PROFILE                                LIMIT
-----
DEFAULT                                10
MONITORING_PROFILE                     UNLIMITED
```

该设置使得 DBSNMP 账户成为密码破解例程的可能目标。此漏洞对那些手动创建或使用 DBCA 的 Custom Database 创建的数据库来说并不适用。

### 15.3 ALTER USER 和未加密的密码

如果你对在网络上发送未加密的密码不放心——毕竟这是为什么抛弃 telnet 和 ftp 而使用 SSH (Secure SHell) 的原因之一。你应该知道 ALTER USER IDENTIFIED BY 正是未加密的, 除非你的站点已经授权并安装了高级安全选项, 从而对所有的 Oracle 网络通信加密。这点很容易证明, 因为当最高级跟踪级别 support 启动时, 未在文档中说明的 Oracle 网络跟踪文件格式包括通过数据库客户端传输的网络数据包的 ASCII 转储。在 Windows 客户端上设置 sqlnet.ora 中的以下参数后, 跟踪文件将被写入 C:\temp 中:

```
trace_level_client=support
trace_directory_client=c:\temp
```

在运行 ALTER USER ndebes IDENTIFIED BY secret 这条 SQL 语句后, 跟踪文件就包含了未加密的密码。

```
[28-SEP-2007 18:07:38:305] nspsend: 00 26 41 4C 54 45 52 20 |.&ALTER.|
[28-SEP-2007 18:07:38:305] nspsend: 55 53 45 52 20 6E 64 65 |USER.nde|
[28-SEP-2007 18:07:38:305] nspsend: 62 65 73 20 49 44 45 4E |bes.IDEN|
[28-SEP-2007 18:07:38:305] nspsend: 54 49 46 49 45 44 20 42 |TIFIED.B|
[28-SEP-2007 18:07:38:305] nspsend: 59 20 73 65 63 72 65 74 |Y.secret|
```

这个漏洞不适用于 PASSWORD 这条 SQL*Plus 命令, 这一点从 Oracle 网络跟踪文件就可以看出来。按照以下的步骤更改密码:

```
SQL> PASSWORD ndebes
Changing password for ndebes
New password:
Retype new password:
Password changed
```

之后, 你可以在跟踪文件中看到加密的密码。

```
[28-SEP-2007 18:12:17:602] nspsend: 06 6E 64 65 62 65 73 10 |.ndebes.|
[28-SEP-2007 18:12:17:602] nspsend: 00 00 00 10 41 55 54 48 |...AUTH|
[28-SEP-2007 18:12:17:602] nspsend: 5F 4E 45 57 50 41 53 53 |_NEWPASS|
[28-SEP-2007 18:12:17:602] nspsend: 57 4F 52 44 40 00 00 00 |WORD@...|
```

```
[28-SEP-2007 18:12:17:602] nspsend: 40 44 38 36 38 43 39 36 |@D868C96|
[28-SEP-2007 18:12:17:602] nspsend: 41 42 34 43 42 37 39 39 |AB4CB799|
[28-SEP-2007 18:12:17:602] nspsend: 36 41 44 34 31 36 36 31 |6AD41661|
[28-SEP-2007 18:12:17:602] nspsend: 32 44 43 41 36 46 42 37 |2DCA6FB7|
[28-SEP-2007 18:12:17:602] nspsend: 43 46 44 39 35 41 35 33 |CFD95A53|
[28-SEP-2007 18:12:17:602] nspsend: 34 35 33 41 45 35 34 39 |453AE549|
[28-SEP-2007 18:12:17:602] nspsend: 35 36 39 34 46 45 37 36 |5694FE76|
[28-SEP-2007 18:12:17:602] nspsend: 36 33 31 38 44 43 43 43 |6318DCCC|
[28-SEP-2007 18:12:17:602] nspsend: 31 00 00 00 00 0D 00 00 |1.....|
[28-SEP-2007 18:12:17:602] nspsend: 00 0D 41 55 54 48 5F 50 |..AUTH_P|
[28-SEP-2007 18:12:17:602] nspsend: 41 53 53 57 4F 52 44 00 |ASSWORD.|
```

# 16

## SELECT FOR UPDATE SKIP LOCKED

SELECT FOR UPDATE SKIP LOCKED 在 Oracle9i 和 Oracle10g 的文档中未作说明。高级队列 (Advanced Queuing, 集成到 ORACLE DBMS 中的可靠消息服务) 在后台使用它。Oracle11g R1 是第一个包含关于 SELECT FOR UPDATE SKIP LOCKED 文档的 DBMS 版本。对于那些试图并发地更新表中相同行集的应用程序来说, SKIP LOCKED 提高了它们的可扩展性。它减少了 TX 锁的等待时间, 同时保留了一致性和隔离性。DBMS 服务器为每一个关注重叠结果集的数据库客户端分配了公平份额的行。

### 16.1 高级队列

高级队列 (AQ) 是集成到 ORACLE DBMS 中的可靠消息服务的实现。AQ 从 Oracle8 开始便可用。到了 Oracle10g, 它被重命名为 Streams AQ (参照 *Streams Advanced Queuing User's Guide and Reference Oracle10g Release 2*), 因为 Stream (一个针对高级复制的替代复制机制) 建立在 AQ 消息队列和消息传播的基础上。Stream 的基础是重做日志挖掘 (Log Miner), 然而高级复制 (Advanced Replication) 却是基于触发器的。

AQ 消息通常是由使用 CREATE TYPE 创建的、用户定义的抽象数据类型 (ADT, 参见 *Application Developer's Guide—Advanced Queuing*) 构成。后者称为消息的有效负载。从最简单的形式来看, 有效负载仅仅是 BLOB, 而不是 ADT。消息可通过调用 DBMS_AQ.ENQUEUE 来创建, 然后从一个队列传播到另一个队列 (甚至从一个数据库到另一个数据库, 或者到不同供应商的另一个消息系统), 最后被 DBMS_AQ.DEQUEUE 使用。具体细节已经超出了本书讨论的范围。请记住, AQ 已在没有任何额外开销的前提下整合到了所有的 ORACLE DBMS 版本中。受益于 ORACLE DBMS 服务器的基础设施提供了崩溃和介质恢复机制, AQ 的可靠性非常高。另外, 它还拥有 Java、PL/SQL 和 C/C++ 接口。所以, 如果你需要在即将开展的项目中使用消息队列机制, AQ 也许是一个不错的选择。

如果你曾经仔细研究过 AQ 的隐藏特性, 可能已经注意到 SELECT FOR UPDATE SKIP LOCKED 这条未在文档中说明的语句。如果你没有幸运地获得后台通行证, 接下来你就会立即知道这个不



为人知的秘密。

当用 DBMS_AQ.DEQUEUE 从队列删除消息时，AQ 使用 SKIP LOCKED，通过阻止对 TX 锁的等待来保证可扩展性。因为当多个进程从同一队列同时出队时，锁定将会严重限制那些为出队而使用并行进程的应用程序的可扩展性。确切地说，这仅适用于应用程序请求那些已准备好出队的消息的情况，这是主要的方法。当并行进程请求特定消息时，争用的可能性就更小了。

接下来的例子将展示如何获得用来详细阐述 AQ 使用 SKIP LOCKED 的 SQL 跟踪文件。在运行下面的代码之前，要确保你拥有足够的权限来执行 DBMS_AQADM 包，这个包在创建队列表和队列时需要。

```
SQL> EXEC dbms_aqadm.create_queue_table('post_office_queue_table', 'raw');
SQL> EXEC dbms_aqadm.create_queue('caught_in_slow_q_again', -
'post_office_queue_table');
SQL> EXEC dbms_aqadm.start_queue('caught_in_slow_q_again');
SQL> ALTER SESSION SET SQL_TRACE=TRUE;
Session altered.
SQL> DECLARE
    dequeue_options dbms_aq.dequeue_options_t;
    message_properties dbms_aq.message_properties_t;
    payload blob;
    msgid raw(16);
BEGIN
    dequeue_options.wait:=dbms_aq.no_wait; -- default is to patiently wait forever
    DBMS_AQ.DEQUEUE (
        queue_name => 'caught_in_slow_q_again',
        dequeue_options => dequeue_options,
        message_properties => message_properties,
        payload => payload,
        msgid => msgid);
END;
/
DECLARE
*
ERROR at line 1:
ORA-25228: timeout or end-of-fetch during message dequeue from
NDEBES.CAUGHT_IN_SLOW_Q_AGAIN
ORA-06512: at "SYS.DBMS_AQ", line 358
ORA-06512: at "SYS.DBMS_AQ", line 556
ORA-06512: at line 8
```

在该 SQL 跟踪文件中，你将看到带 FOR UPDATE SKIP LOCKED 子句的 SELECT 样例。

```
PARSING IN CURSOR #27 len=367 dep=0 uid=30 oct=47 lid=30 tim=69919173952 hv=38306557
86 ad='6771d48c'
DECLARE
    dequeue_options dbms_aq.dequeue_options_t;
    message_properties dbms_aq.message_properties_t;
    payload blob;
    msgid raw(16);
```

```

BEGIN
  dequeue_options.wait:=dbms_aq.no_wait;
  DBMS_AQ.DEQUEUE (
    queue_name => 'caught_in_slow_q_again',
    dequeue_options => dequeue_options,
    message_properties => message_properties,
    payload => payload,
    msgid => msgid);
END;
END OF STMT
PARSE #27:c=0,e=123,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=69919173943
=====
PARSING IN CURSOR #26 len=565 dep=1 uid=0 oct=3 lid=0 tim=69919341626 hv=319671114 a
d='674e3128'
select /*+ FIRST_ROWS(1) */ tab.rowid, tab.msgid, tab.corrid,
tab.priority, tab.delay, tab.expiration, tab.retry_count,
tab.exception_qschema, tab.exception_queue, tab.chain_no,
tab.local_order_no, tab.enq_time, tab.time_manager_info, tab.state,
tab.enq_tid, tab.step_no, tab.sender_name, tab.sender_address,
tab.sender_protocol, tab.dequeue_msgid, tab.user_prop, tab.user_data
from "NDEBES"."POST_OFFICE_QUEUE_TABLE" tab where q_name = :1 and (state = :2 )
order by q_name, state, enq_time, step_no, chain_no, local_order_no
for update skip locked
END OF STMT
EXEC #26:c=0,e=168,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=2,tim=69919341618
FETCH #26:c=0,e=139,p=0,cr=3,cu=0,mis=0,r=0,dep=1,og=2,tim=69919363741
EXEC #27:c=0,e=38468,p=0,cr=12,cu=0,mis=0,r=0,dep=0,og=1,tim=69919378626
ERROR #27:err=25228 tim=0

```

## 16.2 Contention 和 SELECT FOR UPDATE SKIP LOCKED

我们假定 SKIP LOCKED 扩展不存在。若想调查多个进程试图同时使用消息（任何可用的消息）时会发生什么，我们需要先将一些消息入队。

```

SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  enqueue_options dbms_aq.enqueue_options_t;
  message_properties dbms_aq.message_properties_t;
  payload blob;
  msg raw(64);
  msgid raw(16);
BEGIN
  dbms_lob.createtemporary(payload, true);
  msg:=utl_raw.cast_to_raw('message in a bottle');
  dbms_lob.writeappend(payload, utl_raw.length(msg), msg);
  DBMS_AQ.ENQUEUE (
    queue_name => 'caught_in_slow_q_again',
    enqueue_options => enqueue_options,
    message_properties => message_properties,
    payload => payload,

```

```

        msgid => msgid);
    dbms_output.put_line(rawtohex(msgid));
END;
/
89A118C42BFA4A22AE31932E3426E493
PL/SQL procedure successfully completed.
SQL> COMMIT;
Commit complete.

```

除了刚入队的那条包含内容 MSGID=89A118C42BFA4A22AE31932E3426E493 的消息，我们看看队列里有哪些消息。

```

SQL> SELECT tab.msgid, tab.state
FROM "NDEBES"."POST_OFFICE_QUEUE_TABLE" tab
WHERE q_name='CAUGHT_IN_SLOW_Q_AGAIN';
MSGID                                STATE
-----
3DB7BE5A803B4ECB91EF7B021FB223F4      1
00E146F0560C4760886B7AEEEDCF7BF2      1
34F01D98AF0444FF91B10C6D00CB5826      0
54783A999BB544419CDB0D8D44702CD3      0
B78950028B3A4F42A5C9460DDDB9F9D7      0
89A118C42BFA4A22AE31932E3426E493      0
25CC997C9ADE48FFABCE33E62C18A7F3      0
353D44D753494D78B9C5E7B515263A6D      0
2544AA9A68C5A9FB9B6FE410574D85A      0
F7E0192C2AEF45AEAE5661F183261CC      1
10 rows selected.

```

在当前队列里有 10 条消息（每行代表一条消息），STATE=0 代表消息准备出队，而 STATE=1 则意味着该消息入队时带针对延迟出队的延时（message_properties.delay>0，延时是秒级别的）。

当两个进程同时出队时会发生什么？我们使用在出队操作的 SQL 跟踪文件中找到的 SELECT FOR UPDATE 简化版本，并在不带 SKIP LOCKED 的会话 1 中运行它，同时保留绑定变量。

```

SQL> VARIABLE q_name VARCHAR2(30)
SQL> VARIABLE state NUMBER
SQL> BEGIN
    :q_name:='CAUGHT_IN_SLOW_Q_AGAIN';
    :state:=0;
END;
/
PL/SQL procedure successfully completed.
SQL> ALTER SESSION SET SQL_TRACE=TRUE;
Session altered.
SQL> SELECT userenv('sid') FROM dual;
USERENV('SID')
-----
134
SQL> SET ARRAYSIZE 1
SQL> SET PAGESIZE 1
SQL> SET TIME ON

```

```

23:41:04 SQL> SET PAUSE "Hit return to continue"
23:41:04 SQL> SET PAUSE ON
23:41:04 SQL> SELECT tab.msgid
FROM "NDEBES"."POST_OFFICE_QUEUE_TABLE" tab
WHERE q_name=:q_name and (state=:state)
FOR UPDATE;
Hit return to continue

```

会话 1 已检索到由这一点决定的一行，即使 SQL*Plus 还没有显示该行。这在包含带 r=1 (r 是行的缩写) 的 FETCH 调用的 SQL 跟踪输出中得到了证明。

```

$ tail -f ten_ora_1724.trc
=====
PARSING IN CURSOR #2 len=112 dep=0 uid=30 oct=3 lid=30 tim=78932802402 hv=2531579934

ad='6792e910'
SELECT tab.msgid
FROM "NDEBES"."POST_OFFICE_QUEUE_TABLE" tab
WHERE q_name=:q_name and (state=:state)
FOR UPDATE
END OF STMT
PARSE #2:c=0,e=115,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=78932802391
EXEC #2:c=0,e=1290,p=0,cr=16,cu=9,mis=0,r=0,dep=0,og=1,tim=78932838870
FETCH #2:c=0,e=459,p=0,cr=12,cu=0,mis=0,r=1,dep=0,og=1,tim=78932843524

```

从理论上说，这为会话 2 留下了 6 行 STATE=0 的有用消息。我们看看在会话 2 中发生了什么。

```

SQL> ALTER SESSION SET EVENTS '10046 trace name context forever, level 8';
Session altered.
SQL> SELECT userenv('sid') FROM dual;
USERENV('SID')
-----
158
SQL> SET ARRAYSIZE 1
SQL> SET PAGESIZE 1
SQL> SET TIME ON
23:45:28 SQL> SET PAUSE "Hit return to continue"
23:45:28 SQL> SET PAUSE ON
23:45:28 SQL> SELECT tab.msgid
FROM "NDEBES"."POST_OFFICE_QUEUE_TABLE" tab
WHERE q_name=:q_name and (state=:state)
FOR UPDATE;

```

第二个会话不能获取任何数据。这是来自那个会话的第 8 级 SQL 跟踪输出。

```

*** 2007-07-10 23:45:40.319
WAIT #1: nam='enq: TX - row lock contention' ela= 2999807 name|mode=1415053318 usn<<
16 | slot=327721 sequence=660 obj#=16567 tim=79706625648
WAIT #1: nam='enq: TX - row lock contention' ela= 3000541 name|mode=1415053318 usn<<
16 | slot=327721 sequence=660 obj#=16567 tim=79709637248
WAIT #1: nam='enq: TX - row lock contention' ela= 2999946 name|mode=1415053318 usn<<
16 | slot=327721 sequence=660 obj#=16567 tim=79712642844
WAIT #1: nam='enq: TX - row lock contention' ela= 3000759 name|mode=1415053318 usn<<

```

```

16 | slot=327721 sequence=660 obj#=16567 tim=79715649132
*** 2007-07-10 23:45:52.347
WAIT #1: nam='enq: TX - row lock contention' ela= 2999721 name|mode=1415053318 usn<<
16 | slot=327721 sequence=660 obj#=16567 tim=79718655012

```

由于一个锁（入队）请求被持续重复尝试，我们会看到对一个 TX 锁的重复等待。通过查看 V\$LOCK 就可以确认 SID=134 的会话 1 阻塞了 SID=158 的会话 2。

```

SQL> SELECT sid, type, id1, id2, lmode, request, block
FROM v$lock
WHERE sid IN (134,158)
ORDER BY 1;
SID TYPE      ID1 ID2 LMODE REQUEST BLOCK
-----
134 TM        16567 0    3      0      0
134 TX        458764 657 6      0      1
158 TM        16567 0    3      0      0
158 TX        458764 657 0      6      0

```

V\$LOCK 中的 TYPE 和 LMODE 在扩展 SQL 跟踪文件中以 name|mode=1415053318 方式表示出来。这是一个十进制数字，两个高字节以 ASCII 编码来表示入队的名称，而低字节则表示锁定模式。当 V\$SESSION_WAIT.EVENT 中的等待是针对 TX 入队时，名称和模式相对 V\$SESSION_WAIT.P1 是等价的。Oracle9i 使用名称为 enqueue 通用等待事件，然而 Oracle10g 使用 enq: TX - row lock contention。Oracle10g 也在 V\$SESSION.EVENT 和 V\$SESSION.P1 中提供了相同的信息。

把名称和模式转换成更人性化的可读格式，你有两种选择。一种是使用十进制到十六制和十进制到 ASCII 的转换，而另一种则完全依赖于 SQL。UNIX 实用程序 bc（一个任意精度的计算器）可被用于十进制和十六进制数之间的转换（bc 的大部分实现不支持#注释，但是自由软件基金会实现的版本支持）。下面是 bc 会话完成转换的输出。

```

$ bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
obase=16 # ask for hexadecimal output, i.e. output base 16
1415053318 # convert name and mode to hexadecimal
54580006
# lock mode is 6 (exclusive) and enqueue name is hex 5458, let's
# convert that to decimal
obase=10 # ask for output in decimal, i.e. base 10
ibase=16 # input will be hexadecimal
54 # hex
84
# is 84 decimal
58 # hex
88
# is 88 decimal

```

现在我们可以使用 SQL*Plus 中的 CHR 函数将十进制转换成 ASCII。

```

SQL> SELECT chr(84)||chr(88) AS name FROM dual;
NAME

```

```
----
TX
```

另外，一个包含 BITAND 函数的 SELECT 语句，可用来提取来自十进制名称和模式的信息。

```
SQL> VARIABLE p1 NUMBER
SQL> EXEC :p1:=1415053318
SQL> SELECT chr(to_char(bitand(:p1,-16777216))/16777215)||
      chr(to_char(bitand(:p1, 16711680))/65535) "Name",
      to_char( bitand(:p1, 65535) ) "Mode"
FROM dual;
Name Mode
-----
TX      6
```

回到会话 1，然后敲回车键，这样它就可以继续下去了。

```
23:41:04 SQL> SELECT tab.msgid
FROM "NDEBES"."POST_OFFICE_QUEUE_TABLE" tab
WHERE q_name=:q_name and (state=:state)
FOR UPDATE;
Hit return to continue
34F01D98AF0444FF91B10C6D00CB5826
Hit return to continue
```

当前显示的该行是 SQL*Plus 会话 1 之前提取的，SQL 跟踪文件显示另一条 FETCH 调用被执行。

```
FETCH #1:c=0,e=158,p=0,cr=12,cu=0,mis=0,r=1,dep=0,og=1,tim=79430535911
*** 2007-07-10 23:48:01.833
FETCH #1:c=0,e=19466,p=0,cr=1,cu=0,mis=0,r=2,dep=0,og=1,tim=79848141763
```

SQL*Plus 在显示一行后会再次暂停，因为设置了 PAGESIZE=1。切换到会话 2，仍然没有取得进展。虽然我阻止 SQL*Plus 使用单一的 FETCH 调用，并通过将 ARRAYSIZE 从默认值 15 减少到 1 的方式来获取所有相关的行，但是整个过程是单线程的，因此就没有从多处理器系统受益的空间。

最后，在获取会话 1 中所有的行和提交后，会话 2 能获得相应的匹配行。在现实世界中，并没有多余的行让会话 2 来处理，因为会话 1 在完成任务后会将状态更改为非 0 值。这里是会话 1 中的 COMMIT：

```
Hit return to continue
2544AA9A68C54A9FB9B6FE410574D85A
7 rows selected.
23:56:03 SQL>
23:56:05 SQL> COMMIT;
Commit complete.
23:56:17 SQL>
```

现在会话 2 被唤醒。

```
Hit return to continue
34F01D98AF0444FF91B10C6D00CB5826
...
2544AA9A68C54A9FB9B6FE410574D85A
7 rows selected.
```

尽管 SQL*Plus 的正常处理被更改,但这次测试的结果令人失望。不过,考虑到关系数据库的锁定策略,观察到的行为正是所期待的。让我们看一下当加入 SKIP LOCKED 时会发生什么。

会话 1:

```
00:50:41 SQL> ALTER SESSION SET EVENTS '10046 trace name context forever, level 8';
Session altered.
00:50:51 SQL> SET ARRAYSIZE 1
00:50:51 SQL> SET PAGESIZE 1
00:50:51 SQL> SET TIME ON
00:50:51 SQL> SET PAUSE "Hit return to continue"
00:50:51 SQL> SET PAUSE ON
00:50:51 SQL> SELECT tab.msgid
00:50:51 FROM "NDEBES"."POST_OFFICE_QUEUE_TABLE" tab
00:50:51 WHERE q_name=:q_name and (state=:state)
00:50:51 FOR UPDATE SKIP LOCKED;
Hit return to continue
```

会话 2:

```
00:50:44 SQL> ALTER SESSION SET EVENTS '10046 trace name context forever, level 8';
Session altered.
00:51:00 SQL> SET FEEDBACK ON
00:51:00 SQL> SET ARRAYSIZE 1
00:51:00 SQL> SET PAGESIZE 1
00:51:00 SQL> SET TIME ON
00:51:00 SQL> SET PAUSE "Hit return to continue"
00:51:00 SQL> SET PAUSE ON
00:51:00 SQL> SELECT tab.msgid
FROM "NDEBES"."POST_OFFICE_QUEUE_TABLE" tab
WHERE q_name=:q_name and (state=:state)
FOR UPDATE SKIP LOCKED;
Hit return to continue
```

两个会话都在等待输入,以显示第一行。注意会话 2 是怎样在此时打印 Hit return to continue 的。这不是第一次测试的情况,因为等待 TX 入队导致会话 2 的 FETCH 调用不能完成。现在在会话 2 中敲回车键。

```
...
Hit return to continue
54783A999BB544419CDB0D8D44702CD3
Hit return to continue
```

会话 2 成功检索到一行,尽管会话 1 首先执行 SELECT FOR UPDATE SKIP LOCKED。现在在会话 1 中敲回车键。

```
...
Hit return to continue
34F01D98AF0444FF91B10C6D00CB5826
Hit return to continue
```

会话 1 也检索到了一行。当两个会话交替时敲击回车键,则表示两者都获得了公平数目的行。查看 V\$LOCK 可以确认两个会话都没有被阻塞。

```
SQL> SELECT sid, type, id1, id2, lmode, request, block
FROM v$lock WHERE sid IN (134,158)
ORDER BY 1;
SID TYPE      ID1 ID2 LMODE REQUEST BLOCK
-----
134 TM        16567 0    3      0      0
134 TX        196644 644 6      0      0
158 TX         65539 666 6      0      0
158 TM         16567 0    3      0      0
```

表 16-1 按时间顺序阐明了整个事件。表中每往下一行对应一个稍后的时间点。因为 SET TIME ON 时间戳（格式是 HH24:MI:SS）被 SQL*Plus 打印出来，有助于进一步记录续发事件。

表 16-1 并发的 SELECT FOR UPDATE SKIP LOCKED

Session 1	Session 2
00:50:51 SQL> SELECT tab.msgid FROM "NDEBES"."POST_OFFICE_QUEUE_TABLE" tab WHERE q_name=:q_name AND (state=:state) FOR UPDATE SKIP LOCKED;	
Hit return to continue	
34F01D98AF0444FF91B10C6D00CB5826	
Hit return to continue	
25CC997C9ADE48FFABCE33E62C18A7F3	
Hit return to continue	
353D44D753494D78B9C5E7B515263A6D	
3 rows selected.	
00:57:43 SQL>	
	00:51:00 SQL> SELECT tab.msgid FROM "NDEBES"."POST_OFFICE_QUEUE_TABLE" tab WHERE q_name=:q_name AND (state=:state) FOR UPDATE SKIP LOCKED;
	Hit return to continue
	54783A999BB544419CDB0D8D44702CD3
	Hit return to continue
	B78950028B3A4F42A5C9460DDDB9F9D7
	Hit return to continue
	89A118C42BFA4A22AE31932E3426E493
	Hit return to continue
	2544AA9A68C54A9FB9B6FE410574D85A
	4 rows selected.
	00:57:45 SQL>

这一次的结果都非常理想。两个会话都获得了相同数量的行，没有一行被多个会话获得，也没有一行被跳过。两个会话由于锁定被适当分离，所以一个会话不会阻塞另一个。两个会话的第 8 级扩展 SQL 跟踪文件确认它们在整个 FETCH 调用序列期间没有包含对一个 TX 锁的单一等待。在会话达到 3 个甚至更多时，该方法同样有效。

在本章的开头，我极力推荐使用 AQ。正如我们在学习，它在本质上是可扩展的，因



为它在消息出队时使用 SKIP LOCKED 这条未在文档中说明的子句。使用 AQ 的另一个极具吸引力的原因是不需要去查找消息是否可用。我见过一些高负荷的系统，是由于一些应用程序每秒向 DBMS 实例发出几十次请求，询问有无工作可做。按照 AQ 的说法，应该说这些应用程序在寻找消息来消费。如此差的实现也导致 CPU 运行状态一直处于 100%。

仔细研究一下 DBMS_AQ.DEQUEUE 的实现，发现在消息到来前等待一秒或多秒是可能的。请求消息的数据库会话在等待时并没有让 CPU 一直处于忙碌状态。相反，Streams AQ: waiting for messages in the queue（在 Oracle10g 中）等待事件会让会话处于睡眠状态。该等待事件的第一个参数（V\$SESSION_EVENT.P1）是数据字典中队列的对象标识符（DBA_OBJECTS.OBJECT_ID）。V\$SESSION_EVENT.P3 持有助于调用 DBMS_AQ.DEQUEUE 的时间（秒级）。

```
SQL> SELECT p1, p1text, p2, p2text, p3, p3text
FROM v$session_wait
WHERE event='Streams AQ: waiting for messages in the queue';
   P1 P1TEXT                P2 P2TEXT                P3 P3TEXT
-----
16580 queue id 1780813772 process# 120 wait time
SQL> SELECT object_name, object_type FROM dba_objects WHERE object_id=16580;
OBJECT_NAME                OBJECT_TYPE
-----
CAUGHT_IN_SLOW_Q_AGAIN QUEUE
```

顺便说一下，如果你只是想保护共享资源避免被不同进程同时使用，可考虑使用 DBMS_LOCK 而不是用一个表来实现一种相互排斥或信号机制。在此，之所以想解决这个问题，是因为我在网上看到了一些帖子，它们建议使用 SELECT FOR UPDATE SKIP LOCKED 来实现一个资源控制表，也就是说，一个专用表，表中每个资源一行并有一个状态列。状态列的值将表明资源是否可用。显然，对这种表的频繁并行访问将导致对 TX 锁的等待，除非使用未在文档中说明的 SKIP LOCKED 子句。

## 16.3 DBMS_LOCK——题外话

DBMS_LOCK 使得你在共享模式（shared mode）和独占模式（exclusive mode）下都可以请求和释放锁。更重要的是，锁甚至能在两个模式间转换。至于 redo generation，调用 DBMS_LOCK 也实现一个基于表的锁定机制要更好些，毕竟后者需要更新后才能反映锁的状态。而使用 DBMS_LOCK.REQUEST/RELEASE 获取或释放锁不会产生任何重做操作。DBMS_LOCK 在 *PL/SQL Packages and Types Reference* 中有完整的文档说明，但是关于它的使用实例却不多。

锁的名称或号码必须由应用程序的所有组件共同约定。为了确保不同的应用程序在不小心使用相同的锁号码时相互不发生干扰，DBMS_LOCK 提供了一种方式把锁的名称转换成锁的句柄，然后取代无意义的锁号码。所有希望使用相同锁的会话（例如，下面例子中的 MYAPP_MUTEX1）必须调用 DBMS_LOCK.ALLOCATE_UNIQUE 一次，从而把锁的名称转换成锁的句柄。

```
SQL> VARIABLE lockhandle VARCHAR2(128)
SQL> BEGIN
    -- get a lock handle for the lockname that was agreed upon
```

```

-- make sure you choose a unique name, such that other vendors' applications
-- won't accidentally interfere with your locks
DBMS_LOCK.ALLOCATE_UNIQUE(
    lockname => 'MYAPP_Mutex1',
    lockhandle => :lockhandle
);
END;
/
print lockhandle
LOCKHANDLE
-----
10737418641073741864187

```

现在在表 SYS.DBMS_LOCK_ALLOCATED 中新增了一行。

```

SQL> SELECT * FROM sys.dbms_lock_allocated /* no public synonym */;
NAME                                LOCKID EXPIRATION
-----
DROP_EM_USER:SYSMAN                1073741824 13.03.07 17:45
ORA$KUPV$MT-SYSTEM.SYS_EXPORT_SCHEMA_01 1073741844 09.03.07 15:51
ORA$KUPV$JOB_SERIALIZE              1073741845 09.03.07 15:51
ORA$KUPM$SYSTEM$SYS_EXPORT_SCHEMA_01    1073741846 09.03.07 15:48
MYAPP_Mutex1                        1073741864 20.07.07 19:52

```

从以上条目可以明显看出，ORACLE DBMS 使用 DBMS_LOCK 作为内部用途。

```

SQL> VARIABLE result NUMBER
SQL> BEGIN
    -- request the lock with the handle obtained above in exclusive mode
    -- the first session which runs this code succeeds
    :result:=DBMS_LOCK.REQUEST(
        lockhandle => :lockhandle,
        lockmode => DBMS_LOCK.X_MODE,
        timeout => 0,
        release_on_commit => TRUE /* default is false */
    );
END;
/
SELECT decode(:result,0,'Success',
    1,'Timeout',
    2,'Deadlock',
    3,'Parameter error',
    4,'Already own lock specified by id or lockhandle',
    5,'Illegal lock handle') Result
FROM dual;
RESULT
-----
Success

```

第二个会话（最初的 DBMS_LOCK.ALLOCATE_UNIQUE 调用被忽略）可接着运行相同的代码。

```

SQL> VARIABLE result NUMBER
SQL> BEGIN

```

```

:result:=DBMS_LOCK.REQUEST(
    lockhandle => :lockhandle,
    lockmode => DBMS_LOCK.X_MODE,
    timeout => 0,
    release_on_commit => TRUE /* default is false */
);
END;
/
SQL> SELECT decode(:result,1,'Timeout') Result FROM dual;
RESULT
-----
Timeout

```

第二个会话不能获得第一个会话持有的处于独占模式的锁。在 Oracle9i 中，等待锁的时间通过事件 enqueue 计算，而在 Oracle10g 中则通过 enq: UL-contention 计算。UL (User Lock) 缩写的意思是用户锁定。在会话 1 提交后，会话 2 就能够获得锁，因为会话 1 在它的锁请求中指明了 RELEASE_ON_COMMIT=TRUE。

会话 1:

```

SQL> COMMIT;
Commit complete.

```

会话 2:

```

BEGIN
    :result:=DBMS_LOCK.REQUEST(
        lockhandle => :lockhandle,
        lockmode => DBMS_LOCK.X_MODE,
        timeout => 0,
        release_on_commit => TRUE /* default is false */
    );
END;
/
SQL> SELECT decode(:result,0,'Success') Result FROM dual;
RESULT
-----
Success

```

正如你所看到的，使用 DBMS_LOCK 锁定资源独用仅需要非常少的代码。就像使用 AQ 一样，任何资源密集型的轮询已没有必要，因为当等待一个锁时可使用一个非零的超时。当 enq: UL-contention 事件发生时等待会话开始睡眠，以防另一个会话获得的锁处于不兼容的模式。这个等待事件的第二个参数 (V\$SESSION_WAIT.P2) 是 SYS.DBMS_LOCK_ALLOCATED 字典视图中的用户锁标识符 (LOCKID)。

```

SQL> SELECT p1, p1text, p2, p2text, wait_class, seconds_in_wait, state
FROM v$session_wait
WHERE event='enq: UL - contention';

```

P1	P1TEXT	P2	P2TEXT	WAIT_CLASS	SECONDS_IN_WAIT	STATE
1431044102	name mode	1073741864	id	Application	121	WAITING

```
SQL> SELECT name
FROM sys.dbms_lock_allocated la, v$session_wait sw
WHERE sw.event='enq: UL - contention'
AND la.lockid=sw.p2;
NAME
-----
MYAPP_MUTEX1
```

在这一天结束时，DBMS_LOCK 能与 ORACLE 数据库服务器通过 PL/SQL 接口具体化的强大的人队机制相媲美。

### 16.4 源代码库

表 16-2 列出了本章的源文件以及它们的功能。

表16-2    SELECT FOR UPDATE SKIP LOCKED源代码库

文 件 名	功 能
create_queue_deq.sql	用来创建AQ队列表，并尝试让消息出队
aq_enq.sql	用来将消息入队
dbms_lock.sql	说明如何使用DBMS_LOCK包

# Part 6

## 第六部分

### 提供的 PL/SQL 程序包

#### 本 部 分 内 容

- 第 17 章 DBMS_BACKUP_RESTORE
- 第 18 章 DBMS_IJOB
- 第 19 章 DBMS_SCHEDULER
- 第 20 章 DBMS_SYSTEM
- 第 21 章 DBMS_UTILITY

包 DBMS_BACKUP_RESTORE 在 *Oracle9i Supplied PL/SQL Packages and Types Reference* 以及 Oracle10g 的 *Oracle Database PL/SQL Packages and Types Reference* 一书及其后续发行版本中都没有文档说明。使用 RMAN 恢复数据库文件要么需要一个已挂载并带有这些待恢复文件簿记信息的控制文件，要么需要一个带有类似信息到 RMAN 目录的数据库会话。DBMS_BACKUP_RESTORE 使得在出现灾难情况时不用 RMAN 恢复成为可能。这种情况的特点是丢失了当前的所有控制文件，以及缺失（或不可用）恢复目录或包含最新数据文件记录和归档 redo 日志文件备份的控制文件备份。请注意，控制文件备份的簿记信息老化移出取决于初始化参数 CONTROL_FILE_RECORD_KEEP_TIME 的设置。该参数的默认设置仅为 7 天。

## 17.1 恢复管理器

RMAN (Recovery Manager, 恢复管理器) 是在 Oracle8 中引入的一个备份和恢复工具。RMAN 是一个数据库客户端，类似于 SQL*Plus、Data Pump Export/Import、SQL*Loader，以及可以通过 OCI 与 ORACLE 实例交互的任何程序。

RMAN 本身不会备份或恢复一个 ORACLE 数据库的任何部分。如果你还记得，RMAN 可以在一个网络内运行在任何一个机器上，然后连接到远程数据库服务器的一个 ORACLE 实例执行备份和恢复工作，我们可以很清楚地看到 RMAN 既不需要也没有能力去读写任何数据库文件。相反，RMAN 使用文档中未说明的包 DBMS_BACKUP_RESTORE 去指示一个 ORACLE 实例执行备份和恢复操作。

ORACLE 数据库管理系统支持挂载的文件系统和所谓的媒体管理器 (media manager) 来存储备份文件。媒体管理器是通过 SBT (System Backup to Tape, 参见 *Oracle Database Backup and Recovery Basics 10g Release 2*) 接口与一个 ORACLE 实例结合，并且能够控制磁带设备存储备份文件的软件。SBT 接口是 Oracle 公司提出的规范。该规范发放给软件公司的目的是希望他们能支持基于 RMAN 的备份与恢复。SBT 接口在 ORACLE 数据库管理文档中没有记载，通常实现为一个共享库，又名 DLL (Dynamic Link Library, 动态链接库)，程序 \$ORACLE_HOME/bin/oracle (在 Windows 上为 oracle.exe) 使用它实现一个 ORACLE 实例。Oracle 公司提供 SBT 接口的一种实现，它默认与可执行文件 oracle[.exe] 链接。通常，该可执行文件同媒体管理软件发布的共享库

重新链接，使得 ORACLE 实例可以与媒体管理器通信。

Oracle 公司最近通过提供 Oracle Secure Backup (参见 *Oracle Secure Backup Administrator's Guide*) 加入到了提供媒体管理软件公司的圈子。其他有名的参与者有惠普公司的 OmniBack，赛门铁克公司的 NetBackup 以及 Tivoli 软件的 Tivoli Storage Manager。

DBMS_BACKUP_RESTORE 广泛使用控制文件来跟踪备份的三个方面：何时 (when)、何物 (what) 以及怎样 (how) 备份。在这里，何时是指在什么时候开始备份，以及在什么时候结束备份。何物是指文件的类型，如控制文件、数据文件或归档的 redo 日志文件。怎样则指持有备份文件的媒体，如挂载在数据库服务器上的文件系统或媒体管理器，以及在哪一增量级别执行的备份。请注意，Oracle10g 仅支持增量级别 0 和 1，而 Oracle9i 以及之前的发行版本支持级别 0~4。考虑到在默认的差异备份基础上有增量备份，也就没有必要使用超过两个级别的备份了。

接下来，控制文件内部的数据描绘了何时、何物以及怎样，它们被称为备份元数据。请参阅表 17-1 中对几个 V\$ 视图的概述，这些视图提供对元数据的访问。

表17-1 备份相关V\$视图

V\$视图	目 的	相关RMAN命令
V\$ARCHIVED_LOG	归档redo日志文件，它们的线程编号、序列号以及状态（可用、删除、过期、不可用）	BACKUP ARCHIVELOG
DELETE ARCHIVELOG CATALOG ARCHIVELOG		
V\$BACKUP_DATAFILE	数据文件 (FILE# > 0) 和控制文件的备份 (CONTROLFILE_TYPE IS NOT NULL)；两种类型的文件可能存在于同一备份块中，因为属于表空间SYSTEM的文件1的备份，同样备份了控制文件，除非启用了自动控制文件备份 (AUTOBACKUP)	BACKUP DATABASE
BACKUP TABLESPACE BACKUP DATAFILE BACKUP CURRENT CONTROLFILE		
V\$BACKUP_PIECE	备份块，包括创建时间、设备类型、状态以及路径名（列HANDLE）	BACKUP
V\$BACKUP_SET	备份集由一个或多个备份块组成，以防备份集中包含的文件超过了最大的备份块大小	BACKUP
V\$BACKUP_SPFILE	服务器参数文件备份	BACKUP SPFILE
V\$BACKUP_REDOLOG	归档redo日志备份	BACKUP ARCHIVELOG

ORACLE 数据库管理系统发行的版本到 Oracle8i 第 3 版 (8.1.7) 之前有相当大的一个漏洞，这是由于数据库（准确地说是控制文件）被用来跟踪备份。当然，RMAN 刚好从 Oracle8i 的第 1 个发行版本支持恢复目录，用来复制控制文件中与恢复相关的信息。恢复目录使得其在没有有效控制文件时也可以执行恢复操作。与控制文件相反，恢复目录中的备份元数据是不能覆盖的（参数 CONTROLFILE_RECORD_KEEPTIME）。这就给该架构留下了以下两个漏洞。

□ 对于那些没有连接到一个目录 (NOCATALOG 命令行开关) 进行处理，以及在运行了 RMAN

之后没有随之传播的备份文件元数据，它们没有被注册到恢复目录。比方说，万一可用性测试失败，有一个足够智能的备份脚本，可以检查目录的可用性和带 NOCATALOG 选项运行 RMAN。但万一你在目录和控制文件重新同步之前丢失了最新的控制文件，你将不能恢复最近的归档 redo 日志文件，即使恢复目录中断已经结束。

□ 在丢失了最新的控制文件之后，如果恢复目录有过中断也不可能进行恢复。

使用恢复目录从控制文件重新同步备份元数据不需要特别的动作。如果需要进行同步，每当连接到一个目录运行 RMAN 命令时，RMAN 会自动从控制文件传播元数据记录到恢复目录。此功能可以用来处理在两个或更多处于不同数据库服务器上的恢复目录间的控制文件元数据的同步，从而使目录达到更高的可用性而不用依靠于集群或复制技术。仅需要在备份完成之后再运行一次 RMAN，连接到另外的恢复目录（例如使用 CONNECT CATALOG），然后执行 RMAN 命令 RESYNC CATALOG。

上述漏洞可以通过创建一个控制文件副本（ALTER DATABASE BACKUP CONTROLFILE TO 'path'），然后使用文件系统备份工具备份该副本得到解决。这种方法的缺点是执行恢复操作的管理员既需要熟悉 RMAN 和文件系统备份工具的使用，也需要使用这两种工具的特权。通常情况下，文件系统恢复的特权仅限操作系统特权用户，比如 UNIX 系统的 root 用户或者 Windows 系统的 Administrators 用户组。因此，DBA 需要在系统管理员的帮助下执行恢复操作。

Oracle9i RMAN 发布了新的控制文件自动备份功能^①，解决了以上讨论的问题。在每一次备份之后自动备份控制文件可确保最新的控制文件副本（即包含恢复上次备份所有簿记信息的唯一副本）要么被备份到磁盘，要么就是媒体管理器。但是，该功能默认是关闭的，导致数据库仍旧脆弱，除非 DBA 启用此新功能。

多亏了控制文件自动备份，这是第一次我们可以无需访问恢复目录而恢复最新的控制文件。下一个代码段展示了怎样从控制文件自动备份中恢复一个控制文件的例子。恢复操作需要存储在控制文件中的数据库标识符（V\$DATABASE.DBID）。

要在丢失了当前所有控制文件的情况下恢复一个控制文件自动备份，需要使用命令 SET DBID 设置数据库标识符^②。

```
C:> rman target /
Recovery Manager: Release 10.2.0.1.0 - Production on Mon Jul 2 18:32:08 2007
connected to target database: TEN (not mounted)
RMAN> SET DBID 2848896501;
executing command: SET DBID
RMAN> RUN {
    RESTORE CONTROLFILE FROM AUTOBACKUP;
}
Starting restore at 02.07.07 18:32
```

① 见[OL92 2002]中命令 CONFIGURE CONTROLFILE AUTOBACKUP ON 和 CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE 的文档。

② 在 Oracle10g，可以在 FORMAT 规范中使用占位符 %I 在备份块名称中嵌入数据库标识符。因此，即使没有以前备份文件的日志文件，仍然可以从媒体管理器库的备份块名称得到数据库标识符。



```

using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=156 devtype=DISK
channel ORA_DISK_1: looking for autobackup on day: 20070702
channel ORA_DISK_1: autobackup found: c-2848896501-20070702-02
channel ORA_DISK_1: control file restore from autobackup complete
output filename=C:\ORADATA\TEN\CONTROL01.CTL
Finished restore at 02.07.07 18:32

```

自动控制文件备份的默认格式为%F，这转换成 C-database_id-YYYYMMDD-QQ，其中 QQ 是介于 0 和 FF 之间的十六进制序列号（十进制为 0~256）。其余的格式字符串遵循众所周知的 SQL 日期和时间格式模式。如果设置了自定义格式（我强烈反对这么做），对一个自动控制文件执行 RESTORE 命令必须首先执行命令 SET CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE device_type TO 'autobackup_format'，让 RMAN 知道非默认的格式。顺便说一下，RMAN 如何使用十六进制序列号 QQ 在文档中没有记录。它用来为同一天内的自动控制文件备份生成唯一的文件名。请记住，由于日期格式 YYYYMMDD 并不包含时间部分，所以在同一天内所有的自动控制文件备份将有相同的文件名或句柄（V\$BACKUP_PIECE.HANDLE）。当时钟在 12 点敲响，新的一天即将到来之时，RMAN 就开始使用 YYYYMMDD 新的值并且重置 QQ 为 0。但是如果同一天内写入了超过 256 个自动文件备份呢？RMAN 会保留 QQ 的值为 FF，并且会使用序列 FF（即 QQ=FF）覆盖备份块。下面就是证据：

```

SQL> SELECT completion_time, handle
FROM v$backup_piece
WHERE handle LIKE '%FF';
COMPLETION_TIME HANDLE
-----
02.07.07 22:04 C-2848896501-20070702-FF

```

在同一天另一个自动文件备份之后，该备份块的名字被重复使用，查询得到：

```

SQL> SELECT completion_time, handle
FROM v$backup_piece
WHERE handle LIKE '%FF';
COMPLETION_TIME HANDLE
-----
02.07.07 22:30 C-2848896501-20070702-FF

```

一天以后，自动控制文件备份再次使用了十六进制序列号 00。

```

SQL> SELECT completion_time, handle
FROM v$backup_piece
WHERE handle like '%00';
COMPLETION_TIME HANDLE
-----
02.07.07 17:17 C-2848896501-20070702-00
03.07.07 22:51 C-2848896501-20070703-00

```

这是我知道的唯一情况，RMAN 使用新数据覆盖了现有的备份块。在所有其他情况下，RMAN 会拒绝覆盖现有的文件并会终止 BACKUP 命令，并抛出错误 ORA- 19506: failed to create sequential

file, name="string", parms="string" for device type SBT_TAPE, 或者 ORA-19504: failed to create file "string" for device type disk, 以及 ORA-27038: created file already exists。

这种现象阐明了 RMAN 使用的一种算法, 它用来从一个自动控制文件备份中寻找控制文件时生成备份块句柄。其中, 命令 RESTORE CONTROLFILE FROM AUTOBACKUP 的可选参数 MAXDAYS 和 MAXSEQ 的设置决定了 RMAN 所生成的句柄名称。如果 MAXSEQ 没有设置, RMAN 使用默认的 256 (转换过来就十六进制的 FF), 并按照之前提及的 %F 格式构建句柄名称。如果没有这样的备份块存在, 序列编号 QQ 就按 1 递减直到为 0。如果在这样的过程中没有找到任何控制文件备份, RMAN 会继续前进到前一天, 并从十六进制序列编号 FF 重新开始该过程。MAXDAYS 默认值为 7 并允许在 1~366 之间波动。如果已经尝试了早于当前 MAXDAYS 的日期, RMAN 仍然无法找到一个合适的备份块, 命令 RESTORE 将会终止并抛出错误 RMAN-06172: no autobackup found or specified handle is not a valid copy or piece。使用一个较小的 MAXSEQ, 比如 10 可以在某种程度上对搜索加速, 众所周知相对此 BACKUP 命令, 数据文件与归档日志备份每天不会执行超过 10。

新的自动控制文件备份功能 (同时也会备份服务器参数文件 (SPFILE), 如果有的话) 默认是被禁用的 (参阅 *Oracle Database Backup and Recovery Reference 10g Release 2* 2-69 页)。当然生产系统仍然是脆弱的, 因为带有最新恢复相关元数据的控制文件备份 (无论是否为自动) 并不存在。

毫无疑问, 前面描述的情景 1 和 2 的可能性很低。不过令人欣慰的是, 即使在备份脚本质量不够的系统上, 如果你采用未在文档中说明的包 DBMS_BACKUP_RESTORE 的相关建议, 完全恢复的大门仍然是敞开的, 我将很快介绍这一功能。

## 17.2 TDPO 灾难恢复案例研究

TDPO (Tivoli Data Protection for ORACLE, ORACLE Tivoli 数据保护) 是一款符合 Oracle 公司 SBT 规范的软件, 并且通过一个共享库将基于 RMAN 的备份/恢复与 TSM (Tivoli Storage Manager, Tivoli 存储管理器) 结合起来 (参阅 <http://www.tivoli.com>)。它支持 AIX、HP-UX、Solaris、Linux 以及 Windows 客户端平台。

从 TSM 服务器角度来看, 一个 ORACLE 实例就是一个备份客户端。使用 TSM 工具 dsmadm 可以检索备份块的名称。如果一个控制文件可用, 这些名称将匹配 V\$BACKUP_PIECE.HANDLE 中的备份块名称。TSM 服务器接受用来检索客户端备份文件的 SQL 语句。列 NODE_NAME 存储了数据库服务器主机使用的 TSM 客户端名称。如果你给了控制文件、数据文件以及归档日志备份以不同的文件名前缀, 现在终于守得云开见月明了。

```
dsmadm> SELECT * FROM backups WHERE node_name='DBSERVER'
ANR2963W This SQL query may produce a very large result table, or may
require a significant amount of time to compute.
Do you wish to proceed? (Yes (Y)/No (N)) y
NODE_NAME: DBSERVER
FILESPEC_NAME: /adsmorc
FILESPEC_ID: 1
STATE: ACTIVE_VERSION
TYPE: FILE
```

```

HL_NAME: //
LL_NAME: CF-DB-NINE-20071010-vs1u61og
OBJECT_ID: 201461790
BACKUP_DATE: 2007-10-10 14:43:27.000000
DEACTIVATE_DATE:
OWNER: oraoper
CLASS_NAME: DEFAULT

```

在 dsmadm 的输出中，你会寻找最新的控制文件备份或包含控制文件的最新备份块。每当文件 1（表空间 SYSTEM 的第一个文件）备份的时候控制文件也会被备份。下面的代码段转载的是脚本 dbms_backup_restore_cf_tdp.sql，它可以用来恢复一个控制文件而无需挂载数据库和访问目录或自动控制文件备份。在调用 DBMS_BACKUP_RESTORE 时 TSM 库中的 LL_NAME 被用作块名称。

```

variable type varchar2(10)
variable ident varchar2(10)
variable piece1 varchar2(513)
begin
    :type:='SBT_TAPE';
    :ident:='channel1';
    :piece1:='CF-DB-NINE-20071010-vs1u61og';
end;
/
set serveroutput on
DECLARE
    v_devtype  VARCHAR2(100);
    v_done     BOOLEAN;
    v_maxPieces NUMBER;
    TYPE t_pieceName IS TABLE OF varchar2(513) INDEX BY binary_integer;
    v_piece_name_tab t_pieceName;
BEGIN
    -- Define the backup pieces (names from the RMAN Log file or TSM repository)
    v_piece_name_tab(1) := :piece1;
    --v_piece_name_tab(2) := '<backup piece name 2>';
    v_maxPieces := 1;
    -- Allocate a channel (Use type=>null for DISK, type=>'sbt_tape' for TAPE)
    v_devtype := DBMS_BACKUP_RESTORE.deviceAllocate(
        type=>:type,
        ident=> :ident,
        params => 'ENV=(TDPO_OPTFILE=/usr/tivoli/tsm/client/oracle/bin64/
tdpo.opt)'
    );
    dbms_output.put_line('device type '||v_devtype);
    -- begin restore conversation
    DBMS_BACKUP_RESTORE.restoreSetDataFile(check_logical=>false);
    -- set restore location with CFNAME parameter
    DBMS_BACKUP_RESTORE.restoreControlFileTo(cfname=>'/tmp/control.ctl');
    FOR i IN 1..v_maxPieces LOOP
        dbms_output.put_line('Restoring from piece '||v_piece_name_tab(i));
        DBMS_BACKUP_RESTORE.restoreBackupPiece(handle=>v_piece_name_tab(i),
            done=>v_done, params=>null);
    END LOOP;

```

```
        exit when v_done;
    END LOOP;
    -- Deallocate the channel
    DBMS_BACKUP_RESTORE.deviceDeAllocate(:ident);
    EXCEPTION WHEN OTHERS THEN
        DBMS_BACKUP_RESTORE.deviceDeAllocate(:ident);
    RAISE;
END;
/
```

当然，必须运行一个实例才能执行 PL/SQL，但 STARTUP NOMOUNT 足以用来在 SQL*Plus 中运行脚本。正常的 PL/SQL 包只能在数据库打开时才能执行，因为要读取数据字典中所谓的 PL/SQL mpcode。一些在文档中没有记录的“魔法”使我们甚至在没有挂载数据库的时候能够执行 DBMS_BACKUP_RESTORE。以下是一个运行控制文件恢复脚本 dbms_backup_restore_cf_tdpo.sql 的示例：

```
$ sqlplus "/ as sysdba"
SQL*Plus: Release 9.2.0.8.0 - Production on Wed Oct 10 21:33:44 2007
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.8.0 - 64bit Production
SQL> @dbms_backup_restore_cf_tdpo.sql
PL/SQL procedure successfully completed.
device type SBT_TAPE
Restoring from piece CF-DB-NINE-20071010-vsiu610g
PL/SQL procedure successfully completed.
SQL> !ls -l /tmp/control.ctl
-rw-r----- 1 oracle dba          1413120 Oct 10 21:33 /tmp/control.ctl
```

控制文件被恢复为文件/tmp/control.ctl。如果该控制文件包含最新的数据文件和归档日志备份的备份块名称，那么就on挂挂载数据库然后以正常的方式使用 RMAN 继续恢复。但是，如果带有最新备份块名称的控制文件备份不存在——可能因为在上一次备份后控制文件没有被备份（例如，因为自动控制文件备份被禁用），就必须通过调用 DBMS_BACKUP_RESTORE 恢复数据文件和归档日志。基于这些目的，源代码库包含脚本 dbms_backup_restore.sql 和 dbms_backup_restore_arch_tdpo.sql。

17.3 源代码库

表 17-2 列出了本章的源代码文件以及它们的功能。

表17-2 DBMS_BACKUP_RESTORE源代码库

文 件 名	功 能
dbms_backup_restore.sql	灾难恢复脚本，使用磁盘通道恢复一个控制文件以及多个数据文件
dbms_backup_restore_arch_tdpo.sql	灾难恢复脚本，使用TDPO（设备类型SBT_TAPE）从一个备份集恢复归档redo日志，该备份集可能由数个备份块组成
dbms_backup_restore_cf_tdpo.sql	灾难恢复脚本，使用TDPO（设备类型SBT_TAPE）恢复一个控制文件

包 DBMS_IJOB 是一个未被文档化的 PL/SQL 包，被 DBMS_JOB 内部调用。通过直接使用 DBMS_IJOB，DBMS_JOB 固有的局限性可能被克服。使用 DBMS_IJOB 可以创建和删除其他模式中的作业，以 PL/SQL 脚本的形式导出作业以及改变作业的 NLS 环境和运行作业的数据库用户。角色 DBA 包含了执行 DBMS_IJOB 的权限。除非在 ORACLE_HOME 安装了 2009 年 1 月的重要补丁更新，能访问 DBMS_IJOB 的数据库用户可以规避审计。

## 18.1 介绍 DBMS_JOB

包 DBMS_JOB 提交应定期运行的 PL/SQL 程序到作业队列。作业队列通过设置初始化参数 JOB_QUEUE_PROCESSES 的值大于 0 启用，通过协调进程 CJQO 与作业队列从属进程 JNND 处理作业队列。文档中说明的作业队列接口就是包 DBMS_JOB。该包不允许数据库管理员在外模式中创建、修改和删除作业。

DBMS_JOB 通过调用未文档化的包 DBMS_IJOB 来完成它的工作。直接使用 DBMS_IJOB 使得数据库管理员可以克服上述限制。当执行了 COMMIT 之后，使用 DBMS_IJOB 对作业队列的改变就会生效（与 DBMS_JOB 相同）。DBMS_JOBS 的底层数据字典表为 SYS.JOB\$。

DBMS_JOB 和 DBMS_IJOB 共有过程 BROKEN、REMOVE 和 RUN。这些过程在两个包中有相同的参数。对于它们中的每一个我都提供了示例以说明 DBMS_IJOB 如何使得 DBA 完全控制数据库中所有作业。

## 18.2 BROKEN 过程

该过程可以用来改变任何模式中作业的状态，从而克服了 DBMS_JOB.BROKEN 的限制。状态为 broken，即 DBA_JOBS.BROKEN=Y 的作业不会自动运行，但可以使用 DBMS_JOB.RUN 手动运行。

### 18.2.1 语法

```
DBMS_IJOB.BROKEN (  
    job IN BINARY_INTEGER,  
    broken IN BOOLEAN,  
    next_date IN DATE DEFAULT SYSDATE);
```

18.2.2   参数

参   数	描   述
job	作业编号，对应于DBA_JOBS.JO
broken	TRUE：标记作业为broken，FALSE：标记作业为not broken
next_date	下次执行运行作业的时间和日期

18.2.3   使用说明

因为没有你需要的 DBMS_IJOB 公共同义词去授予带有模式名 SYS 的包名称。对于状态为 BROKEN=Y 的作业，列 NEXT_DATE 的值始终为 January 1st, 4000。

18.2.4   范例

下面是标记一个外模式中的作业为 broken 的例子。使用 DBMS_JOB 会失败，而使用 DBMS_IJOB 时成功。

```
SQL> SHOW USER
USER is "NDEBES"
SQL> SELECT role FROM session_roles WHERE role='DBA';
ROLE
-----
DBA
SQL> SELECT job, what, broken FROM dba_jobs WHERE priv_user='PERFSTAT';
JOB WHAT          BROKEN
-----
1 statspack.snap; N
SQL> EXEC dbms_job.broken(1,true)
BEGIN dbms_job.broken(1,true); END;
*
ERROR at line 1:
ORA-23421: job number 1 is not a job in the job queue
ORA-06512: at "SYS.DBMS_IJOB", line 529
ORA-06512: at "SYS.DBMS_JOB", line 245
SQL> EXEC sys.dbms_ijob.broken(1, true)
PL/SQL procedure successfully completed.
SQL> COMMIT;
Commit complete.
SQL> SELECT job, what, broken FROM dba_jobs WHERE priv_user='PERFSTAT';
JOB WHAT          BROKEN
-----
1 statspack.snap; Y
```

18.3   FULL_EXPORT 过程

该过程返回持有 PL/SQL 调用的字符串来重新创建一个作业。它可用于导出作业定义。

### 18.3.1 语法

```
DBMS_IJOB.FULL_EXPORT(  
    job IN BINARY_INTEGER,  
    mycall IN OUT VARCHAR2,  
    myinst IN OUT VARCHAR2);
```

### 18.3.2 参数

参 数	描 述
job	作业编号，对应于DBA_JOBS.JOB
mycall	在成功调用该过程后，返回值持有代表调用了DBMS_IJOB.SUBMIT的字符串
myinst	在成功调用该过程后，返回值持有代表调用了DBMS_JOB.INSTANCE的字符串

### 18.3.3 范例

脚本 `dbms_ijob_full_export.sql` 以适合重新创建作业的格式导出单个作业的定义。该脚本的主要部分转载如下：

```
variable job number  
variable submit_call varchar2(4000)  
variable instance_call varchar2(4000)  
exec :job:=&job_number;  
begin  
    dbms_ijob.full_export(:job, :submit_call, :instance_call);  
end;  
/  
print submit_call  
print instance_call
```

以下是一个运行该脚本导出获取 Statspack 快照的作业的例子：

```
$ sqlplus -s / as sysdba @dbms_ijob_full_export.sql  
JOB WHAT  
-----  
21 statspack.snap;  
Enter value for job_number: 21  
sys.dbms_ijob.submit(job=>21,luser=>'PERFSTAT',puser=>'PERFSTAT',cuser=>'PERFSTAT',  
next_date=>to_date('2007-12-07:21:00:00','YYYY-MM-DD:HH24:MI:SS'),  
interval=>'trunc(SYSDATE+1/24','HH)'),  
broken=>FALSE,what=>'statspack.snap;',nlsenv=>'NLS_LANGUAGE='AMERICAN' NLS_TERRIT  
RY='AMERICA' NLS_CURRENCY='$' NLS_ISO_CURRENCY='AMERICA' NLS_NUMERIC_CHARACTER  
S='.',' NLS_DATE_FORMAT='dd.mm.yyyy hh24:mi:ss' NLS_DATE_LANGUAGE='AMERICAN' NL  
S_SORT='BINARY',env=>'0102000200000000');  
dbms_job.instance(job=>21, instance=>1, force=>TRUE);
```

作业 21 通过执行 DBMS_IJOB.FULL_EXPORT 生成的 PL/SQL 调用而被重新创建。

## 18.4    REMOVE 过程

使用该过程删除任何模式中的作业。

### 18.4.1    语法

```
DBMS_IJOB.REMOVE(  
    job IN BINARY_INTEGER);
```

### 18.4.2    参数

参    数	描    述
job	作业编号

### 18.4.3    范例

以下是一个在外模式中删除一个作业的例子：

```
SQL> SHOW USER  
USER is "NDEBES"  
SQL> SELECT priv_user FROM dba_jobs WHERE job=29;  
PRIV_USER  
-----  
PERFSTAT  
SQL> EXEC sys.dbms_ijob.remove(29)  
PL/SQL procedure successfully completed.  
SQL> COMMIT;  
Commit complete.
```

## 18.5    RUN 过程

该过程可以用来在任何模式中运行作业，无论它们处于何种状态（BROKEN=Y/N）。该过程通过用户前台进程运行来执行 DBMS_IJOB，而不是通过作业队列进程。这有利于对作业排除故障或进行性能诊断，因为其会话跟踪或检测是已知的。DBA_JOBS.NEXT_DATE 会被重新计算。

### 18.5.1    语法

```
DBMS_IJOB.RUN(  
    job IN BINARY_INTEGER,  
    force IN BOOLEAN DEFAULT FALSE);
```

### 18.5.2    参数

参    数	描    述
job	作业编号
force	如果为TRUE，DBMS_IJOB.RUN忽略为DBMS_JOB.SUBMIT指定了参数INSTANCE和FORCE=TRUE的作业实例关联设置。如果在调用DBMS_IJOB.RUN时FORCE=TRUE，那么该作业可以在任何实例运行，从而忽略实例关联



18.5.3 使用说明

假设这样一个场景，一个数据库用户找到 DBA 并要求 DBA 诊断一个失败的作业，该 DBA 通常需要以该作业所有者的身份开启一个数据库会话来重现该问题。使用 DBMS_IJOB，DBA 可以运行和诊断该作业，而无需知道作业所有者的密码或要求作业所有者登录。

Oracle10g 调度器 (DBMS_SCHEDULER) 记录一个作业失败的原因到数据字典视图 DBA_SCHEDULER_JOB_RUN_DETAILS，与之相反，DBMS_JOB 作业队列并不在数据库中记录来自作业运行的错误。来自 DBMS_JOB 失败的错误被记录到后台转储目的地 (参数 BACKGROUND_DUMP_DEST) 中的作业队列从属进程跟踪文件。作业开发者需要实现他们自己的错误日志记录。三个在文档中没有说明的 PL/SQL 元数据变量可用于 DBMS_JOB 作业。它们的变量名、数据类型以及描述见表 18-1。

表18-1 DBMS_JOB元数据

变 量 名	数据类型	描 述
job	BINARY_INTEGER	当前执行的作业编号
next_date	DATE	下一个预定的作业运行日期和时间，该变量可用于覆盖一个作业的重复时间间隔
broken	BOOLEAN	无论当前作业处于何种状态，该变量的初始值为FALSE。它可以用来标记一个作业为broken。如果该变量被赋值为TRUE，并且该作业完成没有抛出任何异常，该作业就会被标记为broken (ALL_JOBS.BROKEN='Y')

变量 broken 被用来捕捉作业异常。如果作业在内部捕获了异常，作业队列不能检测失败的作业。如果捕获到异常，在一个执行作业的匿名 PL/SQL 数据块中设置 broken=TRUE 使得作业开发者可标记该作业为 broken。因此，他可以根据自己的策略设置作业的状态。在经历了 16 次失败之后，那些确实抛出异常的作业会被标记为 broken。作业队列进程使用指数退避策略 (exponential backoff strategy) 计算失败作业的下一次预定运行日期。作业会在初次失败一分钟之后重新尝试。每失败一次等待时间间隔将会增加一倍。作业的下一次预定运行可以通过赋予变量 next_date 一个调整值改写，这可以用来为重新尝试失败的作业实现一个自定义的策略。源代码库中包含文件 dbms_job_metadata.sql，它带有使用表 18-1 中三个 PL/SQL 元数据参数作业的样本实现。

18.5.4 范例

以下是通过在前台进程中使用 DBMS_IJOB.RUN 运行一个失败作业，以用来在外模式中调试的场景：

```
SQL> SHOW USER
USER is "SYS"
SQL> SELECT job, priv_user, failures FROM dba_jobs WHERE job=1;
      JOB PRIV_USER      FAILURES
-----
      1 PERFSTAT          1
SQL> EXEC dbms_job.run(1)
BEGIN dbms_job.run(1); END;
*
```

```
ERROR at line 1:
ORA-23421: job number 1 is not a job in the job queue

SQL> EXEC sys.dbms_job.run(1)
BEGIN sys.dbms_job.run(1); END;
*
ERROR at line 1:
ORA-12011: execution of 1 jobs failed
ORA-06512: at "SYS.DBMS_IJOB", line 406
ORA-06512: at line 1
SQL> ORADEBUG SETMYPID
Statement processed.
SQL> ORADEBUG TRACEFILE_NAME
/opt/oracle/obase/admin/TEN/udump/ten1_ora_19365.trc
SQL> !tail -4 /opt/oracle/obase/admin/TEN/udump/ten1_ora_19365.trc
ORA-12012: error on auto execute of job 1
ORA-00376: file 3 cannot be read at this time
ORA-01110: data file 3: '+DG/ten/datafile/sysaux.261.628550067'
ORA-06512: at "PERFSTAT.STATSPACK", line 5376
```

18.6    源代码库

表 18-2 列出了本章的源代码文件以及它们的功能。

表18-2   DBMS_IJOB源代码库

文   件   名	功      能
dbms_job_full_export.sql	通过从DBA_JOBS（更精确地说是SYS.JOB\$）检索元数据生成PL/SQL代码，用来重新创建一个作业
dbms_job_metadata.sql	该脚本用来在一个数据库表中创建一个表，用来实现DBMS_JOB自定义的日志记录。作业定义使用PL/SQL作业元数据变量记录作业编号以及下次预定运行日期。如果出现异常，该作业捕获它、记录该错误到日志表然后标记自己为broken

数据库调度器 (the database scheduler) 是 Oracle10g 及其后发行版本中一种内建的、先进的作业调度能力。包 DBMS_SCHEDULER 是到作业调度器的接口, 它在 *Oracle Database Administrator's Guide and the PL/SQL Packages and Types Reference* 中已经有完整的文档说明。数据库调度器支持在 DBMS 内部运行的作业, 同样也支持运行在 DBMS 外操作系统级的作业。后一种类型的作业被称为外部作业 (external job)。有关外部作业执行的重要方面, 如退出代码处理、环境变量清除、程序参数传递细节、除 SYS 用户之外其他用户运行外部程序的要求, 以及配置文件 externaljob.ora 定义的外部作业默认权限, 这些在文档中都没有进行说明。

本章提供了外部作业这一主题的所有详细资料: 它们如何运行, 哪些环境变量和特权对它们有效, 怎样发送信息指示成功和失败, 以及怎样将自定义日志记录与调度器自己的日志记录整合。

## 19.1 使用数据库调度器运行外部作业

数据库调度器是自 Oracle10g 及其后版本发行的一种先进的作业调度功能。通过 PL/SQL 包 DBMS_SCHEDULER 这一接口可以访问一系列丰富的作业调度器功能。其中, Oracle10g R1 是第一个带有运行数据库外部作业能力的 ORACLE 数据库管理器发行版本。该调度器支持三种类型的作业。

- 存储过程。
- PL/SQL 块。
- 可执行文件, 即运行于数据库引擎之外的外部程序。

作业链 (job chain) 是 Oracle10g 引入的另一项新功能。作业链由多个作业组成。通过使用规则决定下一次执行链中的哪一个作业。因为调度器支持运行在数据库引擎之内以及数据库引擎之外的作业, 那么使用它控制涉及操作系统级以及数据库内作业步骤的复杂处理是有意义的。另一个在操作系统级运行作业的选择是企业管理器作业系统。就我自己的经验看来, 数据库调度器工作得十分完美, 但我亲眼目睹了企业管理器作业系统的一些故障。因此, 相比于企业管理器作业系统, 我建议使用数据库调度器。企业管理器网格控制 (Enterprise Manager Grid Control) 包括了二者基于 Web 的接口。企业管理器数据库控制 (Enterprise Manager Database Control) 则有一个基于 Web 的数据库调度器接口。

在高可用性环境中，如故障转移集群或实时应用集群，调度器解决了即使有一个节点或实例失效的情况下作业的运行问题。因为高可用性解决方案确保在一个集群至少有一个实例存活（准确地说是 在一个故障转移集群中，每个数据库一个实例），并且该实例可以执行作业。因此，为了实现调度，没有必要去保护企业管理器代理（Enterprise Manager agent）以及它的作业系统或由集群软件提供的第三方调度解决方案。

数据库调度器这些在文档中没有说明的方面在 Windows 及 UNIX 上均适用，而其他方面则是平台有关的。接下来的两节包括了一些通用的方面。有关 UNIX 和 Windows 平台特定的功能在单独的章节中讨论。在 *Oracle Database Administrator's Guide* 中记录说明了一个有趣的调度器通用功能，就是它捕获并存储上限为 200 字节的标准错误，并输出到字典视图 DBA_SCHEDULER_JOB_RUN_DETAILS 的列 ADDITIONAL_INFO 中。

### 19.1.1 退出代码处理

在 UNIX 以及 Windows 系统上，程序通过返回一个退出代码到父进程用来发送信号指示成功或失败。退出代码 0 指示成功执行，而 1~255 之间的退出代码则指示为一个失败或至少一个警告。在 Bourne、Korn 以及 Bash shell 中，8 位退出代码可以用变量 \$? 得到。如果一个程序由一个信号终止，退出代码为 128+n，其中 n 为终止该程序的信号编号。在 UNIX 系统上，信号编号定义在 C 编程语言的包含文件 /usr/include/sys/signal.h 中。下面是一个例子，启动 UNIX 程序 sleep，然后在 sleep 仍然运行时使用 Ctrl+C 中断该进程。

```
$ sleep 10
$ echo $?
130
```

echo 语句从 sleep 程序获取退出代码。因为 130-128 为 2，我们需要在 signal.h 中寻找信号 2。

```
#define SIGINT 2      /* interrupt */
```

信号 2 为中断信号，它同样可以使用 kill-INT pid 发送到一个进程，其中 pid 为进程标识符。

在 Perl 中，\$? 提供信号信息（如果有的话），在结束一个子进程以及除了退出代码之外是否还生成一个内核转储方面发挥作用。因此在 Perl 中 \$? 是一个 16 位的值。数据库调度器在这方面怎样进展？调度器怎样决定一个外部作业成功与否在文档中并未说明。

在 Windows 系统上，%ERRORLEVEL% 与 UNIX 系统上的 \$? 具有相同的用途。以下是一个使用 Cygwin 调用程序 true.exe 和 false.exe 的例子。Cygwin 是 Windows 系统上高度推荐的、免费的类 UNIX 环境的软件集合，它提供所有的 UNIX 工具，诸如 bash、find、awk、grep、vim (Vi iMproved) 以及 X11（用来重定向 UNIX 系统上比如 Oracle Universal Installer 的 X11 客户端输出到一个 Windows 机器上的位图显示）。

```
C:> false.exe
C:> echo %ERRORLEVEL%
1
C:> true.exe
C:> echo %ERRORLEVEL%
0
```

让我们返回数据库调度器。一个作业成功执行的特点是视图 DBA_SCHEDULER_JOB_LOG 中列 STATUS 值为 SUCCEEDED。失败由值 FAILED 指示（按照 *Oracle Database Administrator's Guide 10g Release 2 2-27* 页中的规定不是 FAILURE）。但是这是怎样与可执行文件的退出代码关联起来的呢？除了信号成功或失败，退出代码可能还指示了外部作业失败的原因。难道可以通过 DBA_SCHEDULER_JOB_LOG.ADDITIONAL_INFO 或者视图 DBA_SCHEDULER_JOB_RUN_DETAILS 中的列 STATUS、ERROR#或 ADDITIONAL_INFO 获取？一些测试揭露了这一点。

- 返回退出代码为 0 的可执行文件被赋予作业状态 SUCCEEDED。所有其他的退出代码都将导致状态为 FAILED。
- 退出代码本身是不可用的。
- 列 DBA_SCHEDULER_JOB_RUN_DETAILS.STATUS 在 *Oracle 10g Release 2 Database Reference* 并未说明的值的范围与视图 DBA_SCHEDULER_JOB_LOG 中列 STATUS 的值的范围相同（SUCCEEDED 或 FAILED）。

## 19.1.2 标准错误输出

UNIX 系统的 shell 脚本和 windows 系统的命令行脚本支持相同的输出重定向语法。这两种环境提供了三种默认的文件句柄。表 19-1 中进行了总结。

表19-1 输入和输出句柄

句柄名称	等效数字	描 述
STDIN	0	键盘输入
STDOUT	1	输出到终端窗口
STDERR	2	错误输出到终端窗口

默认情况下，UNIX 命令 echo 打印它的参数到标准输出（STDOUT）。要重定向一条错误信息到标准错误输出（STDERR），使用如下命令：

```
$ echo Command failed. 1>&2
Command failed.
```

该命令在 Windows 上完全一样。

```
C:> echo Command failed. 1>&2
Command failed.
```

到目前为止，已报告的调查结果的证据都来自于执行一个小小的命令行脚本，并且其以退出代码 1 终止。接下来的例子是在 Windows 系统上编码的，但是结果同样适用于 UNIX。Windows 命令行脚本的源代码（文件 failure.bat）转载如下：

```
echo This is script %0.
echo About to exit with exit code 1. 1>&2
exit 1
```

下面的 PL/SQL 块创建了一个运行上面脚本的外部作业。其所提供的日程表达式 FREQ=MINUTELY;INTERVAL=3 安排该作业每 3 分钟运行一次。请注意，该作业都是以用户

SYS 创建和运行的。

```
C:\> sqlplus / as sysdba
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
SQL> BEGIN
    DBMS_SCHEDULER.CREATE_JOB(
        job_name => 'failure_test',
        job_type => 'EXECUTABLE',
        job_action => 'C:\home\ndebes\bin\failure.bat',
        start_date => systimestamp,
        repeat_interval => 'FREQ=MINUTELY;INTERVAL=3',
        enabled=>true /* default false! */
    );
END;
/
PL/SQL procedure successfully completed.
```

该作业可以通过调用过程 DBMS_SCHEDULER.RUN_JOB 手动运行。

```
SQL> EXEC dbms_scheduler.run_job('failure_test')
BEGIN dbms_scheduler.run_job('failure_test'); END;
*
ERROR at line 1:
ORA-27369: job of type EXECUTABLE failed with exit code: Incorrect function.
ORA-06512: at "SYS.DBMS_ISCHED", line 154
ORA-06512: at "SYS.DBMS_SCHEDULER", line 450
ORA-06512: at line 1
```

由于非 0 退出代码，该作业被列为 FAILED。经过前面的手动作业执行和一个单一的计划执行，查询 DBA_SCHEDULER_JOB_LOG 以及 DBA_SCHEDULER_JOB_RUN_DETAILS 得到如下结果：

```
SQL> SELECT jl.log_id, jl.status, jl.additional_info AS log_addtl_info,
jd.status, jd.additional_info AS details_addtl_info
FROM dba_scheduler_job_log jl, dba_scheduler_job_run_details jd
WHERE jl.job_name='FAILURE_TEST'
AND jl.log_id=jd.log_id
ORDER BY jl.log_id;
```

LOG_ID	STATUS	LOG_ADDTL_INFO	STATUS	DETAILS_ADDTL_INFO
245	FAILED		FAILED	ORA-27369: job of type EXECUTABLE failed with exit code: Incorrect function. STANDARD_ERROR="About to exit with exit code 1. "
246	FAILED	REASON="manually run"	FAILED	ORA-27369: job of type EXECUTABLE failed with exit code: Incorrect function. STANDARD_ERROR="About to exit with exit code 1. "

以上展示了该测试结果，同样额外的测试总结如下。

- 不论外部作业的退出代码如何，视图 DBA_SCHEDULER_JOB_RUN_DETAILS 的列 ADDITIONAL_INFO 都会捕捉其标准错误输出。该列类型为 CLOB，这样理论上可以捕获多达 8 TB 或 128 TB 的数据，这取决于参数 DB_BLOCK_SIZE 的设置（参见 *Database Reference 10g Release 2*，页 A1）。
- 对 DBA_SCHEDULER_JOB_RUN_DETAILS.ADDITIONAL_INFO 有一个 200 字节的大小限制。
- DBA_SCHEDULER_JOB_RUN_DETAILS.ADDITIONAL_INFO 从来不捕获标准输出。
- 视图 DBA_SCHEDULER_JOB_LOG 列 ADDITIONAL_INFO 的值为 NULL 则是针对基于一个计划表运行的作业，如果其值为 REASON="manually run"则是针对通过调用过程 DBMS_SCHEDULER.RUN_JOB 手动运行的作业。

为了删除命名为 FAILURE_TEST 的作业，执行对 DBMS_SCHEDULER.DROP_JOB 的调用，转载如下：

```
SQL> EXEC dbms_scheduler.drop_job('failure_test')
```

## 19.2 UNIX 系统上的外部作业

如果能教会一个数据库通过将 RMAN 作为外部作业运行来备份它自己岂不是很好？毫无戒心的 DBA 可能会认为所需要做的就是使用带有与 shell 完全相同的命令行选项的调度器运行 RMAN。然而，实际测试表明问题更复杂。环境变量和参数传递都是文档中未提及的障碍，我们必须克服这两个问题。

在源代码库中，我提供了在第 36 章讨论的一个通过管道接口控制 RMAN 的作业原型（文件 `rman_backup.sql`）。它创建了一个使用调度器运行 RMAN 的外部作业和类型为 PLSQL_BLOCK 的作业。后者通过手动执行上述的外部作业启动 RMAN，然后通过管道接口传递命令给 RMAN。因此，该数据库就可以备份它自己了。不需要在 DBMS 实例之外安排备份。在一个高可用性环境下，自动保护 DBMS 实例以防失效同样也保护了备份作业。

调度器为类型为 PLSQL_BLOCK 的作业提供了几种类型的元数据属性。Oracle10g 和 Oracle11g 支持相同的属性（参见 *Oracle Database PL/SQL Packages and Types Reference 11g Release 1* 114-61 页）。不幸的是，用来给视图 DBA_SCHEDULER_JOB_LOG 和 DBA_SCHEDULER_JOB_RUN_DETAILS 中的列 LOG_ID 编号的序列 SYS.SCHEDULER\$_INSTANCE_S 的值并不在其中，而它对于整合一个数据库表中的自定义日志和调度器本身的日志将非常有用。然而，有一个解决办法。在已计划的作业运行后，DBA_SCHEDULER_JOBS.NEXT_RUN_DATE 的值会被复制到 DBA_SCHEDULER_JOB_RUN_DETAILS.REQ_START_DATE（要求的开始日期）。一个类型为 PLSQL_BLOCK 的作业解决了该集成问题，其选择 NEXT_RUN_DATE 并使之成为自定义日志表的一部分。在该作业完成之后，自定义日志表可以通过使用列 REQ_START_DATE 与 DBA_SCHEDULER_JOB_RUN_DETAILS 进行联合查询。一个示例实现包含在源代码库文件 `rman_backup.sql` 中。

### 19.2.1 清除环境变量

文档中没有说明的是，调度器在它开始一个 UNIX 进程去实现一个外部作业之前清除了所有

的环境变量。通过让调度器运行一个调用程序 env 的 shell 脚本可以很容易收集到该特征的证据。在 UNIX 系统上，如果调用时不带参数，env 就会打印出它所继承的所有环境变量。下面是一个 shell 脚本的代码，它用来在标准错误输出上打印所有环境变量的排序列表，并且使用命令 tee 保存该输出的一个副本到文件/tmp/env.out。文件 env.sh 的执行权限是必需的，这使用 chmod 进行设置。

```
$ more env.sh
#!/bin/sh
env | sort | tee /tmp/env.out 1>&2
exit 0
$ chmod +x env.sh
```

为了让数据库调度器运行该 shell 脚本，我们创建了下面的调度器程序 ENV：

```
SQL> BEGIN
    DBMS_SCHEDULER.CREATE_PROGRAM(
        program_name=>'env',
        program_action=>' /home/oracle/env.sh',
        program_type=>'EXECUTABLE',
        number_of_arguments=>0,
        comments=>'environment variables',
        enabled=>true);
end;
/
```

然后我们使用前面的程序创建一个作业：

```
SQL> BEGIN
    sys.dbms_scheduler.create_job(
        job_name => 'env_job',
        program_name => 'env',
        auto_drop => FALSE,
        enabled => false);
END;
/
```

该作业被定义为禁用状态，这消除了提供一个调度器的需要。我们现在准备手动运行作业。

```
SQL> EXEC dbms_scheduler.run_job(job_name => 'env_job')
PL/SQL procedure successfully completed.
SQL> SELECT status, error#, additional_info
FROM dba_scheduler_job_run_details
WHERE job_name='ENV_JOB'
AND owner='SYS';
STATUS          ERROR# ADDITIONAL_INFO
-----
SUCCEEDED      0 STANDARD_ERROR="PWD=/
                SHLVL=1
                _=/bin/env"
```

由于该 shell 脚本以退出代码 0 终止，该作业是成功的（SUCCEEDED）。作业调度器保存该



作业的标准错误输出到视图 DBA_SCHEDULER_JOB_RUN_DETAILS 中的列 ADDITIONAL_INFO。仅仅只有三个环境变量并且这些绝不是你所期望的, PATH、ORACLE_HOME 和 ORACLE_SID 都失踪。PWD (Process's Working Directory, 进程工作目录) 是比较熟悉的一个。SHLVL 和 “_” 出现在这里是由于存在该测试是在 Linux 系统上这一事实, 其中 / bin / sh 是一个到 / bin / bash 的符号链接。SHLVL (shell 级别) 在 bash 每次以子进程的形式启动另一个 bash 实例时以 1 递增。bash 在环境变量 “_” (下划线) 中存放了每一个命令的完整路径, 然后由子进程继承。

如果我们坚持要实现一个可以自行备份自身的数据库, 我们必须以这样一种方式使用调度器运行 RMAN: 环境变量 ORACLE_HOME 和 ORACLE_SID 可见, 同样也应该设置 NLS_DATE_FORMAT, 因为它控制 RMAN 所使用的日期和时间格式。我想到了如下几种方法。

- 通过调度器运行 /usr/bin/env, 使用 env 的命令行参数设置环境变量, 然后使得 env 按如下示例运行 RMAN:

```
$ /usr/bin/env ORACLE_HOME=/opt/oracle/product/db10.2 ORACLE_SID=TEN \
NLS_LANG=dd.Mon.yy-hh24:mi:ss /opt/oracle/product/db10.2/bin/rman target / \
cmdfile /opt/oracle/admin/scripts/backup.rcv msglog /opt/oracle/admin/log/rman.log
```

- 写一个 shell 脚本设置所必要的环境变量, 然后调用 RMAN。让调度器执行该 shell 脚本。
- 写一个通用的包装脚本接收带有必要设置的环境变量和运行程序的文件名称作为输入。

第三种方法应该是最有希望的, 因为可以实现重用, 此外, 包装脚本能打印程序的退出代码到标准错误输出, 而且以后可以通过视图 DBA_SCHEDULER_JOB_RUN_DETAILS 检索到。源代码库的文件 extjob.sh 中包含这样一个包装脚本的示例实现。

## 19.2.2 命令行处理

另一个未在文档中说明的方面是有关调度器怎样运行指定了过程 DBMS_SCHEDULER.CREATE_PROGRAM 或 DBMS_SCHEDULER.CREATE_JOB 其中之一的 PROGRAM_ACTION。Oracle Database PL/SQL Packages and Types Reference 10g Release 2 告诉我们: “对于一个类型为 EXECUTABLE 的程序, PROGRAM_ACTION 为外部可执行文件的名称, 包括完整的路径名和任何命令行参数。” (93-35 页, [OL10 2005]) 这句话的第一部分是正确的, 因为尝试去运行一个外部程序而不指定完整路径名将会失败并给出错误信息 ORA-27369: job of type EXECUTABLE failed with exit code: No such file or directory。当一个作业失败后, 调度器错误信息可以通过数据字典视图 DBA_SCHEDULER_JOB_RUN_DETAILS 的列 ADDITIONAL_INFO 检索到。

然而, 那句话的第二部分仅适用于 Windows。在 UNIX 系统上, 外部程序的参数必须使用过程 DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT 定义, 而不是将它们添加到 PROGRAM_ACTION。否则基于该程序的作业就会失败并给出错误信息 ORA-27369: job of type EXECUTABLE failed with exit code: No such file or directory。对于那些不基于程序的外部作业不能接受参数。

在 UNIX 系统上, 程序 \$ORACLE_HOME/bin/extjob0 负责运行属于 SYS 用户的外部作业。使用系统调用跟踪工具比如 truss 或 strace 可以看到 extjob0 使用 UNIX 系统调用 access 去验证该程序动作是可执行的。一个包含参数的程序动作在这个测试中会失败, 并抛出 UNIX 错误

ENOENT (No such file or directory, 没有该文件或文件夹)。系统调用跟踪同样表明可执行文件直接使用系统调用 `execve` 运行。这意味着程序参数中的字符, 比如 `?` 或 `*` 对于 shell 来说有特殊意义, 不能回避。

另一层含义是用来指定运行文本文件的解释器的伪注释 `#!/bin/sh` (或 `#!/bin/ksh`、`#!/bin/bash` 等), 必须出现在 shell 脚本的第一行。如果对一个命令解释器的指定缺失, 就会报告错误 `ORA-27369: job of type EXECUTABLE failed with exit code: 255`, 同时 `DBA_SCHEDULER_JOB_RUN_DETAILS.ADDITIONAL_INFO` 包含 `STANDARD_ERROR="execve: Exec format error"`。

为了将 Perl 脚本作为外部作业运行, `perl` 也同样必须指定为命令解释器。因此, Perl 脚本的第一行应该指定 `perl` 解释器的绝对路径, 如下所示:

```
#!/opt/oracle/product/db10.2/perl/bin/perl
```

一个移植性更好的方法是使用 `/usr/bin/env` 去运行 `perl`, `/usr/bin/env` 在所有 UNIX 系统上有相同的绝对路径, 如下所示:

```
#!/usr/bin/env perl
```

在数据库调度器上下文中, 后一种方法的缺点是环境变量 `PATH` 被 `extjobo` 清除了。因此, 必须实现一些在 Perl 脚本运行之前设置 `PATH` 的机制。为了达到这个目的我们必须再次使用 `/usr/bin/env`, 通过定义一个程序调用 `env` 然后在程序参数中设置 `PATH`。以下就是一个例子, 应当由调度器执行的 Perl 脚本 `test.pl` 所包含的代码如下:

```
#!/usr/bin/env perl
printf STDERR "This is perl script $0 executed by UNIX process $$.\\n";
exit 0;
```

下面的 PL/SQL 块创建了一个带有两个参数的程序来运行 Perl 脚本 `test.pl`。为了使程序包含参数, 所有参数必须通过单独调用包过程 `DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT` 进行定义。

```
begin
    dbms_scheduler.create_program(
        program_name=>'perl_program',
        program_type=>'EXECUTABLE',
        program_action=> '/usr/bin/env',
        number_of_arguments=>2,
        enabled=>false
    );
    dbms_scheduler.define_program_argument(
        program_name=>'perl_program',
        argument_position=>1,
        argument_name=>'env',
        argument_type=>'VARCHAR2',
        default_value=>'PATH=/opt/oracle/product/db10.2/perl/bin:/home/oracle'
    );
    dbms_scheduler.define_program_argument(
        program_name=>'perl_program',
```

```

argument_position=>2,
argument_name=>'script',
argument_type=>'VARCHAR2',
default_value=>'test.pl'
);
dbms_scheduler.enable('perl_program');
dbms_scheduler.create_job(
job_name=>'perl_job',
program_name=>'perl_program',
enabled=>false,
auto_drop=>false
);
end;
/

```

该作业成功了，因为已经被调度器清除掉了的环境变量 PATH 被显式地作为程序参数提供。文件 test.pl 必须位于分配给 PATH 的目录中的一个。

```

SQL> EXEC dbms_scheduler.run_job('perl_job')
PL/SQL procedure successfully completed.
SQL> SELECT status, additional_info
FROM dba_scheduler_job_run_details
WHERE log_id=(SELECT max(log_id) FROM dba_scheduler_job_run_details);
STATUS      ADDITIONAL_INFO
-----
SUCCEEDED STANDARD_ERROR="This is perl script /home/oracle/test.pl
executed by UNIX process 5387."

```

### 19.2.3 外部作业与非特权用户

在 UNIX 系统上，特权用户 SYS 拥有的外部作业使用 ORACLE 软件所有者（其通常为 UNIX 系统用户 oracle）特权运行。除 SYS 用户之外的数据库用户所拥有的外部作业的执行是默认启用的。在文档中并没有说明运行这些外部作业的 UNIX 用户具体是谁。对于既没有特权 SYSDBA 也没有角色 DBA 的用户，则需要系统特权 CREATE JOB 和 CREATE EXTERNAL JOB 去成功地创建和运行外部作业。因此，为了创建一个刚好拥有足够特权的新用户 EXTJOB 去运行外部作业，我们可以使用下面的 SQL 语句：

```

SQL> CREATE USER extjob IDENTIFIED BY secret;
SQL> GRANT CONNECT TO extjob;
SQL> GRANT CREATE JOB TO extjob;
SQL> GRANT CREATE EXTERNAL JOB TO extjob;

```

当调用时不带有参数，UNIX 程序 id 显示用户名和当前用户的用户组。该程序可以用来找出非特权用户是使用哪一个 UNIX 用户运行外部程序。调用一个 shell 脚本重定向 id 输出到调度器捕捉的标准错误，得到如下信息：

```

SQL> SELECT additional_info
FROM all_scheduler_job_run_details
WHERE log_id=(SELECT max(log_id) FROM all_scheduler_job_run_details);

```

```
ADDITIONAL_INFO
```

```
-----
STANDARD_ERROR="uid=99(nobody) gid=99(nobody) groups=800(oinstall),801(dba)"
```

这里使用 UNIX 用户和用户组 nobody。基于这个原因, UNIX 用户 nobody 的存在是 ORACLE DBMS 安装指南中提到的安装先决条件。Oracle10g R2 UNIX 安装指南中错误地指定程序 \$ORACLE_HOME/bin/extjob 必须为 nobody 所有。在这种情况下, 外部作业将会失败并在 ALL_SCHEDULER_JOB_RUN_DETAILS.ADDITIONAL_INFO 中抛出如下错误信息:

```
ADDITIONAL_INFO
```

```
-----
ORA-27369: job of type EXECUTABLE failed with exit code: 274662
STANDARD_ERROR="Oracle Scheduler error: Config file is not owned by root or is
writable by group or other or extjob is not setuid and owned by root"
```

错误信息中提到的 config file 是指文件 externaljob.ora, 它位于目录 \$ORACLE_HOME/rdbms/admin 中。该文件在 Oracle10g 中没有进行说明, 而在 *Oracle Database Administrator's Guide 11g Release 1* 中进行了部分说明。它必须为 root 用户所有并且必须只有其所有者才可写:

```
$ ls -l $ORACLE_HOME/rdbms/admin/externaljob.ora
-rw-r----- 1 root oinstall 1534 Dec 22 2005 /opt/oracle/product/db10.2/rdbms/admin
/externaljob.ora
```

文件 externaljob.ora 的内容转载如下^①:

```
# This configuration file is used by dbms_scheduler when executing external
# (operating system) jobs. It contains the user and group to run external
# jobs as. It must only be writable by the owner and must be owned by root.
# If extjob is not setuid then the only allowable run_user
# is the user Oracle runs as and the only allowable run_group is the group
# Oracle runs as.
run_user = nobody
run_group = nobody
```

extjob 正确的权限是 setuid root。

```
$ ls -l $ORACLE_HOME/bin/extjob
-rwsr-x--- 1 root oinstall 64920 Jul 21 17:04 /opt/oracle/product/db10.2/bin/extjob
```

setuid 权限是必须的, 以使得程序 extjob 可以通过调用 C 语言库函数 seteuid 将它的有效用户 ID 改为 nobody 的 ID。有效用户组 ID 通过调用 setegid 设置。因为有效用户和用户组 ID 在使用 execve 运行外作业之前都被改为 nobody, 仅仅用户和用户组 nobody 的权限对不属于 SYS 的外部作业有效。该机制必须用来防止外部作业以 SYS 用户连接数据库, 否则将构成一个严重的安全威胁。

① 尽管是一个正确的配置, 但由于 externaljob.ora 中的一行超过 100 个字符, 就会抛出错误 ORA-27369: job of type EXECUTABLE failed with exit code: 274668 STANDARD_ERROR="Oracle Scheduler error: Invalid or missing run_group in configuration file."。

Metalink note 391820.1 建议设置 `run_user=oracle` 和 `run_group=oinstall` 作为解决错误 ORA-27369: job of type EXECUTABLE failed with exit code: Operation not permitted 和 ORA-27369: job of type EXECUTABLE failed with exit code: 274662 的一部分。从安全的角度来看, 这很成问题。通常, UNIX 用户 `oracle` 是 `OSDBA` 用户组 (通常为 `dba` 用户组) 的成员, 并且可以无需提供密码以 `SYS` 用户连接。通过允许除了 `SYS` 的用户以 `OSDBA` 用户组成员的身份去执行外部作业, 这些用户在它们的外部作业中可以以 `SYS` 用户连接。这样, 具有 `CREATE JOB` 和 `CREATE EXTERNAL JOB` 特权的任何用户都可以以 `SYS` 用户连接! 因此正确的解决方法就是以 `SYS` 用户创建和运行作业。属于 `SYS` 用户并且由 `SYS` 用户运行的作业总是以 `ORACLE` 软件所有者的身份执行。运行这些作业的程序 `$ORACLE_HOME/bin/extjobo` 不使用配置文件 `externaljob.ora`。`extjobo` 的 `setuid` 权限也不是必需的, 因为该程序不改变有效用户或用户组标识符。

## 19.3 Windows 系统上的外部作业

Windows 系统上的数据库调度器实现与 UNIX 系统上的实现有以下 3 点不同之处:

- 命令行参数处理;
- 环境变量;
- 非特权用户执行外部作业。

接下来的 3 节详细说明了这 3 个议题。

### 19.3.1 命令行参数处理

与 UNIX 系统相反, Windows 系统上的调度器程序或作业动作可以包含命令行参数。源代码库中包含了一个使用调度器进行 RMAN 备份的完整原型 (zip 文件 `exec_rman.zip`)。该原型包括备份脚本 `exec_rman.bat`, 其需要下面两个参数:

- 一个带有配置变量的批处理脚本, 如日志文件的目录 (`LOG_DIR`) 和一个到 RMAN 目录 (`CATALOG_CONNECT`) 的连接字符串;
- 运行 RMAN 脚本的名称, 例如, 对数据库或归档 redo 日志备份。

脚本 `exec_rman.bat` 调用配置批处理文件 (例如 `TEN_config.bat`) 来读取配置, 然后运行所要求的 RMAN 脚本。它校验 RMAN 的退出代码并打印到标准错误输出, 这样它就可以从视图 `DBA_SCHEDULER_JOB_RUN_DETAILS` 中检索到。如果 RMAN 以非 0 退出代码终止, `exec_rman.bat` 同样也以非 0 退出代码终止, 以此方式发送备份成功或失败的信号给调度器。

以下是一个使用 `exec_rman.bat` 的作业并给其提供所需命令行参数的例子:

```
SQL> BEGIN
    DBMS_SCHEDULER.CREATE_JOB(
        job_name => 'rman_online_backup',
        job_type => 'EXECUTABLE',
        job_action => 'c:\home\ndebes\bin\exec_rman.bat
C:\home\ndebes\rman\TEN_config.bat backup_online.rcv',
        start_date => systimestamp,
        repeat_interval => 'FREQ=DAILY;BYHOUR=22',
```

```

        enabled=>true /* default false! */
    );
END;
/

```

在 Windows 系统上, 进程浏览器 (Process Explorer) ^① 是用来可视化进程之间关系的合适工具。图 19-1 显示调度器使用 %ORACLE_HOME%\bin\extjobo.exe 和 Windows 命令解释器 (cmd.exe) 去运行作业动作中定义的批处理脚本。选项 /C 指示 cmd.exe 以命令的形式执行传入的字符串并且在命令完成后尽快退出。

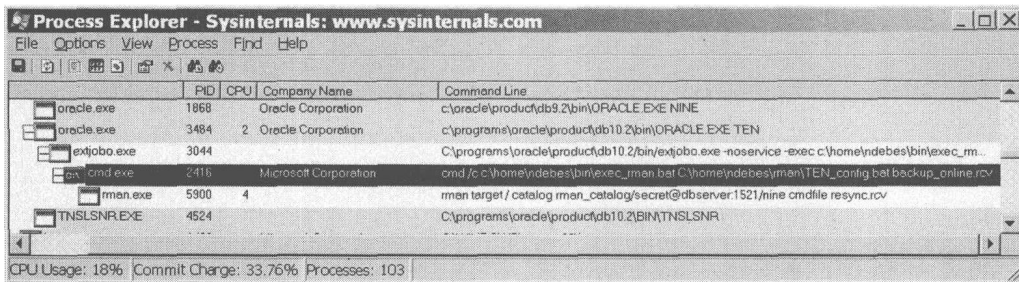


图 19-1 进程浏览器中的一个外部作业

### 19.3.2 Windows 环境变量

在上一节中, 我们看到使用 cmd/c 运行 Windows 上的外部作业。因为 cmd.exe 检索并设置全系统以及用户特定的环境变量, 我们应该料到环境变量可用于外部作业。一个快速的测试证实这个假设是正确的。默认情况下, 实现一个 ORACLE 实例的服务以用户 SYSTEM 的身份运行。因此, 只有全系统的环境变量是有效的。但是, 如果该服务以特定用户的身份运行, 该用户设置的环境变量也同样是有效的。下面的批处理脚本 (文件 environment.bat) 可以作为外部作业运行去检测环境变量:

```

echo PATH=%PATH% > c:\temp\environment.log
echo ORACLE_HOME=%ORACLE_HOME% >> c:\temp\environment.log
echo ORACLE_SID=%ORACLE_SID% >> c:\temp\environment.log
echo NLS_DATE_FORMAT=%NLS_DATE_FORMAT% >> c:\temp\environment.log
echo TNS_ADMIN=%TNS_ADMIN% >> c:\temp\environment.log
echo PERL5LIB=%PERL5LIB% >> c:\temp\environment.log

```

一个有趣的结果是, ORACLE_SID 通常不定义为环境变量, 被设置为与 V\$INSTANCE.INSTANCE_NAME 相同的值。Windows 系统上 DBMS 的这种行为同我们在 UNIX 系统上看到的正好相反。而 extjob 和 extjobo 在 UNIX 系统上清除所有的环境变量, 相同的程序在 Windows 系统上则设置 ORACLE_SID (或至少它们在从另一个进程继承之后不会清除它)。

① 进程浏览器可以在 <http://www.sysinternals.com> 免费获取。它用来找出哪一个进程打开了一个 DLL 或文件。我推荐除 Sysinternals 之外的其他两个工具: Regmon 用来监测注册表访问、Filemon 用来监测文件访问。

### 19.3.3 外部作业与非特权用户

属于数据库用户 SYS 的外部作业在 Windows 系统上都以用户 SYSTEM 运行。SYSTEM 是 Administrators 用户组的成员。从安全角度来看，这就好像外部作业在 UNIX 系统上以 root 用户运行。必须注意外部作业运行的脚本或可执行文件不能被潜在的入侵者修改。

默认安装后。除了 SYS 之外用户拥有的作业会失败，并抛出如下错误：

```
SQL> EXEC dbms_scheduler.run_job('id')
BEGIN dbms_scheduler.run_job('id'); END;
*
ERROR at line 1:
ORA-27370: job slave failed to launch a job of type EXECUTABLE
ORA-27300: OS system dependent operation:accessing execution agent failed with
status: 2
ORA-27301: OS failure message: The system cannot find the file specified.
ORA-27302: failure occurred at: sjsec 6a
ORA-27303: additional information: The system cannot find the file specified.
ORA-06512: at "SYS.DBMS_ISCHED", line 150
ORA-06512: at "SYS.DBMS_SCHEDULER", line 441
ORA-06512: at line 1
```

### 19.3.4 ORADIM 实用工具创建的服务

ORADIM 是一个创建 Windows 服务的命令行工具，它实现了 RDBMS 和 ASM 实例。如果你曾仔细观察过 ORADIM 创建的服务，你可能会注意到它在 Oracle10g 及其后续版本中创建了两个服务。例如，如果你创建了一个如下名为 TEST 的 RDBMS 实例。

```
C:> oradim -new -sid TEST -syspwd secret -startmode manual -srvstart demand
```

你将看到这两个服务：

- ❑ OracleServiceTEST;
- ❑ OracleJobSchedulerTEST。

服务 OracleServiceTEST 实现 RDBMS 实例。使用 regedit.exe 打开注册表可以看到服务 HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\OracleServiceTEST 基于可执行文件 oracle.exe。OracleJobSchedulerTEST 处于禁用状态，且由 extjob.exe 实现。该服务必须启动以使得除了 SYS 之外数据库用户的外部作业顺利完成。

### 19.3.5 OracleJobScheduler 服务

新的服务 OracleJobSchedulerORACLE_SID 在 Oracle10g 中并没有文档说明，其中 ORACLE_SID 是使用 oradim.exe 创建的实例名称。这在 *Oracle Database Platform Guide 11g Release 1 for Microsoft Windows* 有文档说明。该 Oracle11g 文档指出该服务必须配置为在启动之前以一个低特权的用户运行它。默认情况下，配置 OracleJobSchedulerORACLE_SID 以用户 SYSTEM 运行。出于安全原因，此设置必须修改。否则，任何有足够权限执行外部作业的数据库用户都可以使用操

作系统身份认证以 SYS 用户连接，进而控制本地系统上的数据库。用来运行 OracleJobScheduler-ORACLE_SID 的账户不应该是 Administrators 或 ORA_DBA 用户组的成员。

### 19.4    源代码库

表 19-2 列出了本章的源代码文件以及它们的功能。

表19-2   DBMS_SCHEDULER源代码库

文   件   名	功            能
environment.bat	保存Windows环境变量设置到一个文件，供外部作业使用
exec_rman.zip	Windows系统上使用数据库调度器获得RMAN备份的原型
extjob.sh	通过数据库调度器运行外部作业的shell脚本包装。运行适当的作业前设置环境变量
failure.bat	模拟作业失败的Windows命令行脚本
perl_job.sql	在UNIX系统上运行perl脚本test.pl的调度器程序和作业
rman_backup.sql	创建一个启用了管道接口运行RMAN的外部作业。另外一个类型为PLSQL_BLOCK的作业通过管道接口传递命令给RMAN。RMAN接收和发送的管道消息都记录在自定义数据库表中。该表与调度器本身的日志记录整合在一起。第36章提到的包RMAN_PIPE_IF是必须的
test.pl	Perl脚本，用来写入脚本名称和执行该脚本进程的进程标识符到标准错误输出



## DBMS_SYSTEM

Oracle9i 及其后续版本中默认安装包 DBMS_SYSTEM。该包的执行权限仅限于用户 SYS。在相关手册中有一些地方提及包 DBMS_SYSTEM，但是在 *Oracle9i Supplied PL/SQL Packages and Types References* 和 *Oracle Database PL/SQL Packages and Types Reference of Oracle10g Release 2 and Oracle11g Release* 中却没有进行记录说明。

包 DBMS_SYSTEM 有很多有用的功能，比如在任何数据库会话所有支持的级别中启用 SQL 跟踪、设置事件、不使用 SQL*Plus 命令 ORADEBUG 生成转储、给警报日志或跟踪文件编写自定义项、获取环境变量值以及在运行中的会话中改变参数。由于包 DBMS_SUPPORT 默认没有安装，DBMS_SYSTEM 是使用 DBMS_SUPPORT 跟踪会话的替代方案。

## 20.1 GET_ENV 过程

该过程提供了对服务一个数据库客户端进程的环境变量的只读访问。如果你曾经绝望地寻找一个途径去询问数据库实例在什么 ORACLE_HOME 下运行，那么现在你可以不用找了。虽然一个数据库实例在哪个机器上运行（V\$INSTANCE.HOST_NAME）和实例的 ORACLE_SID（V\$INSTANCE.INSTANCE_NAME）都可以通过视图 V\$INSTANCE 获得，但没有任何一个 VS 视图告诉你 ORACLE_HOME 的设置是什么。过程 GET_ENV 在 Oracle9i 中不可用。

### 20.1.1 语法

```
DBMS_SYSTEM.GET_ENV(  
    var IN VARCHAR2,  
    val OUT VARCHAR2);
```

### 20.1.2 参数

参 数	描 述
Var	环境变量名称
Val	环境变量值

### 20.1.3 使用说明

如果没有一个环境变量匹配传入参数 var 的值, 就会在 val 中返回一个空字符串。如果 val 的最大长度不足以保存环境变量的值, 就会抛出错误 ORA-06502: PL/SQL: numeric or value error: character string buffer too small。为了避免这种情况, 我建议你总是给 val 分配尽可能大的空间。相比于 PL/SQL 中的 VARCHAR2(32767), 在 SQL*Plus 中最大值是 VARCHAR2(4000)。环境变量名称在 UNIX 系统上是大小写敏感的, 而在 Windows 系统上则不是。

### 20.1.4 范例

下面的 PL/SQL 代码可以用来获取 SQL*Plus 中环境变量 ORACLE_HOME 的值:

```
SQL> SET AUTOPRINT ON
SQL> VARIABLE val VARCHAR2(4000)
SQL> BEGIN
    dbms_system.get_env('ORACLE_HOME', :val);
END;
/
PL/SQL procedure successfully completed.
VAL
-----
/opt/oracle/product/10.2
```

## 20.2 KCFRMS 过程

该过程重置每个事件最大等待时间 (V\$SESSION_EVENT.MAX_WAIT)、一个数据文件的最大读取时间 (V\$FILESTAT.MAXIORTM) 和数据文件的最大写入时间 (V\$FILESTAT. MAXIOWTM) 为 0。该过程在观察到文件访问 (V\$FILESTAT) 或等待事件 (V\$SESSION_EVENT) 的峰值的情况下非常有用。通过调用 DBMS_SYSTEM.KCFRMS 之前按小时保存来自这些 V\$视图的值, 我们就可以确定每天在什么时间和在何种程度上会出现高峰。

### 20.2.1 语法

```
DBMS_SYSTEM.KCFRMS();
```

### 20.2.2 使用说明

V\$SESSION_EVENT 中所有会话的值都会被设置为 0, 而不仅仅是调用过程 DBMS_SYSTEM.KCFRMS 的会话。

### 20.2.3 范例

以下是一些在调用 DBMS_SYSTEM.KCFRMS 之前 V\$SESSION_EVENT 的行数据样本 (所有时间的单位为秒, 列 MAX_WAIT 本身的值的单位为厘秒):

```
SQL> SELECT sid, wait_class, event,
round(time_waited_micro/1000000,3) AS time_waited_sec,
max_wait/100 AS max_wait_sec
FROM v$session_event WHERE sid in(140, 141)
ORDER BY sid, wait_class, event;
SID WAIT_CLASS EVENT TIME_WAITED_SEC MAX_WAIT_SEC
-----
```

140	Application	enq: TX - row lock contention	4.929	3
140	Commit	log file sync	.009	.01
140	Idle	SQL*Net message from client	53.167	29.72
140	Network	SQL*Net message to client	0	0
140	System I/O	control file sequential read	.006	.01
140	User I/O	db file scattered read	.208	.13
140	User I/O	db file sequential read	.603	.04
140	User I/O	direct path write temp	0	0
141	Application	SQL*Net break/reset to client	.022	0
141	Commit	log file sync	.04	.01
141	Idle	SQL*Net message from client	2432.647	171.39
141	Network	SQL*Net message to client	.001	0
141	Other	events in waitclass Other	1.042	1.04
141	System I/O	control file sequential read	2.166	0
141	User I/O	db file sequential read	.267	.02

在调用 DBMS_SYSTEM.KCFRMS 后，对 V\$SESSION_EVENT 相同的查询给出如下结果：

```
SID WAIT_CLASS EVENT TIME_WAITED_SEC MAX_WAIT_SEC
-----
```

140	Application	enq: TX - row lock contention	4.929	0
140	Commit	log file sync	.009	0
140	Idle	SQL*Net message from client	379.432	0
140	Network	SQL*Net message to client	0	0
140	System I/O	control file sequential read	.006	0
140	User I/O	db file scattered read	.208	0
140	User I/O	db file sequential read	.603	0
140	User I/O	direct path write temp	0	0
141	Application	SQL*Net break/reset to client	.022	0
141	Commit	log file sync	.04	0
141	Idle	SQL*Net message from client	2460.816	28.17
141	Network	SQL*Net message to client	.001	0
141	Other	events in waitclass Other	1.042	0
141	System I/O	control file sequential read	2.166	0
141	User I/O	db file sequential read	.267	0

正如你从这些结果看到的，列 MAX_WAIT（列 MAX_WAIT_SEC）中所有列的值都被设置为 0。对于会话 141，MAX_WAIT 的值又变为非 0，因为它在 DBMS_SYSTEM.KCFRMS 完成之后曾变得活跃。在调用 DBMS_SYSTEM.KCFRMS 之前的 V\$FILESTAT 内容如下：

```
SELECT file#, phydrds, phywrts, readtim, writetim, maxiortm, maxiowtm
FROM v$filestat;
FILE# PHYDRDS PHYWRTS READTIM WRITETIM MAXIORTM MAXIOWTM
-----
```

1	8194	845	36635	810	6	8
---	------	-----	-------	-----	---	---

2	26	1655	88	1698	0	1
3	720	1324	3060	1534	4	0
4	213	10	997	0	8	0
5	5	1	23	1	0	0

在调用 DBMS_SYSTEM.KCFRMS 之后，查询 V\$FILESTAT 返回以下数据：

```
SELECT file#, phydrds, phywrts, readtim, writetim, maxiortm, maxiowtm
FROM v$filestat;
```

```
FILE# PHYDRDS PHYWRTS READTIM WRITETIM MAXIORTM MAXIOWTM
```

```
-----
1      8194      849  36635      812        0        0
2       26      1655      88      1698        0        0
3      720      1324     3060     1534        0        0
4      213       10      997        0        0        0
5        5        1       23        1        0        0
```

## 20.3 KSDDDT 过程

该过程写入一个时间戳到 SQL 跟踪文件。由 PRACLE DBMS 日期和时间格式模式（参见 *Oracle Database SQL Reference 10g Release 2* 第 2 章）代表的时间戳格式为 YYYY-MM-DD HH24:MI:SS.FF3。

### 20.3.1 语法

```
DBMS_SYSTEM.KSDDDT();
```

### 20.3.2 使用说明

如果服务于当前数据库会话的进程还没有跟踪文件，如执行语句 ALTER SESSION SET sql_trace=TRUE 则创建一个跟踪文件。时间戳的格式不取决于会话的 NLS（National Language Support，国际语言支持）设置，即 NLS_DATE_FORMAT 和其他相关参数不影响时间戳格式。

### 20.3.3 范例

```
C:> sqlplus / as sysdba
SQL*Plus: Release 10.2.0.1.0 - Production on Mon Jun 25 13:17:07 2007
SQL> ORADEBUG SETMYPID
Statement processed.
SQL> ORADEBUG TRACEFILE_NAME
Statement processed.
SQL> EXEC dbms_system.ksdddt();
PL/SQL procedure successfully completed.
SQL> ORADEBUG TRACEFILE_NAME
c:\programs\admin\ten\udump\ten_ora_4588.trc
SQL> $type c:\programs\admin\ten\udump\ten_ora_4588.trc
Dump file c:\programs\admin\ten\udump\ten_ora_4588.trc
Mon Jun 25 13:17:25 2007
```

```
...
Windows thread id: 4588, image: ORACLE.EXE (SHAD)
...
*** SERVICE NAME:(SYS$USERS) 2007-06-25 13:17:25.923
*** SESSION ID:(147.755) 2007-06-25 13:17:25.923
*** 2007-06-25 13:17:25.923
```

上述例子最后一行是 DBMS_SYSTEM.KSDDDT 写入的时间戳。正如你所看到的，当 ORADEBUG TRACEFILE_NAME 第一次被调用时，它不返回跟踪文件的路径，因为在那个时间点不存在跟踪文件。通过运行 DBMS_SYSTEM.KSDDDT 创建了一个跟踪文件，其路径则在第二次调用 ORADEBUG TRACEFILE_NAME 时返回。

20.4 KSDFLS 过程

该过程刷新所有挂起输出到目标文件（警报日志和/或跟踪文件）。

20.4.1 语法

```
DBMS_SYSTEM.KSDFLS();
```

20.4.2 使用说明

就个人而言，我从来没有发现在哪种情况下需要调用 DBMS_SYSTEM.KSDFLS。

20.4.3 范例

```
SQL> EXEC dbms_system.ksdfls
```

20.5 KSDIND 过程

该过程用来缩进使用 DBMS_SYSTEM.KSDWRT 写入到 SQL 跟踪文件的下一个字符串，通常是通过将一个或多个由参数 lvl 指定的冒号 (:) 放置到该行的开头。

20.5.1 语法

```
DBMS_SYSTEM.KSDIND(
    lvl IN BINARY_INTEGER);
```

20.5.2 参数

参 数	描 述
lvl	缩进级别

20.5.3 使用说明

该过程在使用 DBMS_SYSTEM.KSDWRT 写入警报日志时不起作用。

20.5.4 范例

```
SQL> BEGIN
      sys.dbms_system.ksdind(3);
      sys.dbms_system.ksdwrt(1, 'indented string');
END;
/
```

以上匿名块将下面一行代码写入跟踪文件：

```
:::indented string
```

20.6 KSDWRT 过程

该过程写入一个字符串到服务数据库客户端的服务器进程的 SQL 跟踪文件、实例的警报日志或同时写入 SQL 跟踪文件和警报日志。如果 SQL_TRACE 在服务器进程中没有被启用（通过 ALTER SESSION、DBMS_SYSTEM、DBMS_SUPPORT^①等）从而还不存在跟踪文件，那么在这里就会创建一个跟踪文件。因此，可以在不启用 SQL 跟踪的情况下写入会话记录。

20.6.1 语法

```
DBMS_SYSTEM.KSDWRT(
    dest IN BINARY_INTEGER,
    tst IN VARCHAR2);
```

20.6.2 参数

参 数	描 述
dest	目标文件，1代表SQL跟踪文件，2代表警报日志，3代表前两者
tst	写入到目标文件的字符串

20.6.3 使用说明

当写入到一个 SQL 跟踪文件时，考虑到该会话已经停用超过 10 秒，一个时间戳（如 DBMS_SYSTEM.KSDDT 写入的时间戳）就会被自动地放置在写入字符串的前一行。为了保证要写入的字符串前面总有一个时间戳，在调用 DBMS_SYSTEM.KSDWRT 之前显式调用 DBMS_SYSTEM.KSDDT。

当写入到警报日志时，写入字符串的前面会被无条件地添加时间戳，其格式为 Dy Mon DD HH24:MI:SS YYYY（如 Mon Jun 25 15:17:37 2007）。

由于包 DBMS_SYSTEM 具有仅供特权会话访问的功能，因此最好是通过包装程序提供对 DBMS_SYSTEM 的访问。这种包装程序是 Hotsos^②提供的开源 ILO 的一部分。它们的名字是

① 包 DBMS_SUPPORT 可以使用脚本 \$ORACLE_HOME/rdbms/admin/dbmssupp.sql 安装。对外部会话的跟踪使用过程 START_TRACE_IN_SESSION 初始化，然后使用过程 STOP_TRACE_IN_SESSION 终止。  
② 见 <http://www.hotsos.com>。

HOTSOS_SYSUTIL.WRITE_DATESTAMP 和 HOTSOS_SYSUTIL.WRITE_TO_TRACE。这些包的执行权限被授予 PUBLIC。

## 20.6.4 范例

20

下面的匿名块演示了对写入到跟踪文件中代价高的任务的信息是怎样计时的。每行开头的字符串“===”用以确保该行会被 TKPROF 工具忽略。在 Oracle9i 中不提供 DBMS_UTILITY.GET_CPU_TIME。

```
SQL> DECLARE
    elapsed_time_t1 number;
    elapsed_time_t2 number;
    cpu_time_t1 number;
    cpu_time_t2 number;
BEGIN
    elapsed_time_t1:=dbms_utility.get_time;
    cpu_time_t1:=dbms_utility.get_cpu_time;
    dbms_stats.gather_schema_stats(user); -- do something expensive
    elapsed_time_t2:=dbms_utility.get_time;
    cpu_time_t2:=dbms_utility.get_cpu_time;
    sys.dbms_system.ksdiddt;
    sys.dbms_system.ksdwrt(1, '=== Elapsed time: ' ||
        to_char((elapsed_time_t2 - elapsed_time_t1)/100)||
        ' sec CPU: ' || to_char((cpu_time_t2 - cpu_time_t1)/100) || ' sec');
END;
```

该匿名块写入如下的条目到跟踪文件：

```
*** 2007-06-25 16:23:12.316
=== Elapsed time: 1.15 sec CPU: .68 sec
```

另一个利用 DBMS_SYSTEM.KSDWRT 的例子就是通过调用 KSDWRT 创建一个 SERVERERROR 触发器去记录那些没有正常地写入到警报日志的错误。举例来说，Oracle9i 在发生错误 ORA-01555: snapshot too old 时不会把该条目写入到警报日志（但 Oracle10g 会）。通过实现一个 SERVERERROR 触发器去检查错误代码 1555，该错误和失败的语句就可以写入到 Oracle9i 实例的警报日志。许多检测工具如 Enterprise Manager 或 Tivoli TEC 能够分析实例的警报日志并传递找到的错误到管理控制台。通过写入自定义的错误比如 ORA-20000 到警报日志，就可以构建一个简单高效的集成。

## 20.7 READ_EV 过程

该过程读取一个事件的启用级别，对禁用的事件返回 0。

### 20.7.1 语法

```
DBMS_SYSTEM.READ_EV(
    iev IN BINARY_INTEGER,
    oev OUT BINARY_INTEGER);
```

20.7.2   参数

参   数	描   述
iev	事件编号，通常介于10000~10999之间
oev	该事件被设置在哪个级别，level=0代表该事件被禁用

20.7.3   使用说明

当 SQL 跟踪已经通过设置事件 10046 启用，DBMS_SYSTEM.READ_EV 就会按预期工作，例如使用 ALTER SESSION、DBMS_MONITOR.SESSION_TRACE_ENABLE 或 DBMS_SYSTEM.SET_EV。

DBMS_SYSTEM.READ_EV 在 Oracle10g 和 Oracle11g 中并不与 ALTER SESSION SET SQL_TRACE=TRUE 一起协作工作，因为 DBMS_SYSTEM.READ_EV 在语句被执行时返回的 oev 仍然为 0（Oracle9i 中返回 oev=1，即它在 Oracle9i 中按预期工作）。在后一种情况下，语句 SELECT value FROM v\$parameter WHERE name='sql_trace' 可以用来找出 SQL 跟踪是否在两个发行版本中开启。

20.7.4   范例

下面的例子演示了在 Oracle10g 中使用 DBMS_MONITOR 启用对 SQL 语句的跟踪,以及使用等待事件在级别 8 设置事件 10046:

```
SQL> VARIABLE lev number
SQL> SET AUTOPRINT ON
SQL> EXECUTE sys.dbms_system.read_ev(10046, :lev)
PL/SQL procedure successfully completed.
LEV
-----
0
SQL> EXEC dbms_monitor.session_trace_enable
PL/SQL procedure successfully completed.
SQL> SELECT sql_trace, sql_trace_waits, sql_trace_binds FROM v$session
WHERE sid=userenv('sid')
SQL_TRACE  SQL_TRACE_WAITS SQL_TRACE_BINDS
-----
ENABLED    TRUE                FALSE
SQL> EXECUTE sys.dbms_system.read_ev(10046,:lev)
PL/SQL procedure successfully completed.
LEV
-----
8
```

20.8   SET_INT_PARAM_IN_SESSION 过程

该过程在外部数据库会话中设置一个整型参数。



## 20.8.1 语法

```
DBMS_SYSTEM.SET_INT_PARAM_IN_SESSION(
    sid IN NUMBER,
    serial# IN NUMBER,
    parnam IN VARCHAR2,
    intval IN BINARY_INTEGER);
```

## 20.8.2 参数

参 数	描 述
sid	会话标识符，对应于V\$SESSION.SID
serial#	会话序号，对应于\$SESSION.SERIAL#
parnam	参数名，对应于V\$PARAMETER.NAME
intval	赋给一个参数的整数值

## 20.8.3 使用说明

传入一个错误的 SID 或 SERIAL#时不会引发异常。

## 20.8.4 示例

下面的例子展示了怎样在一个会话中增加参数 SORT_AREA_SIZE 的设置，例如由于参数 SORT_AREA_SIZE 的默认值不足导致排序很慢^①。首先，启动一个会话并通过调用包 DBMS_UTILITY^② 检索 SORT_AREA_SIZE 的当前设置。

```
$ sqlplus hr/hr
SQL> VARIABLE result NUMBER
SQL> VARIABLE sort_area_size NUMBER
SQL> VARIABLE dummy VARCHAR2(255)
SQL> BEGIN
    :result:=dbms_utility.get_parameter_value(parnam=>'sort_area_size',
        intval=>:sort_area_size, strval=>:dummy);
END;
/
PL/SQL procedure successfully completed.
SQL> PRINT sort_area_size
SORT_AREA_SIZE
-----
        65536
```

下一步，以用户 SYS 启动另一个 SQL*Plus 会话，检索你想修改的会话的 SID 和 SERIAL#，并通过调用包 DBMS_SYSTEM 增加 SORT_AREA_SIZE 到 1 048 576 字节。

- ① 在大多数情况下，SORT_AREA_SIZE 和其他 *_AREA_SIZE 参数不应该再用来取代 PGA_AGGREGATE_TARGET。SORT_AREA_SIZE 不能使用 ALTER SYSTEM 修改。
- ② 在任何会话中都可以使用 DBMS_UTILITY 检索参数的设置，而对 V\$PARAMETER 的访问需要角色 SELECT_CATALOG_ROLE 或其他合适的特权。

```
$ sqlplus "/ as sysdba"
SQL> SELECT sid, serial# FROM v$session WHERE username='HR';
      SID      SERIAL#
-----
          9          19
SQL> EXEC sys.dbms_system.set_int_param_in_session(9, 19, -
'sort_area_size', 1048576);
PL/SQL procedure successfully completed.
现在，回到 HR 的第一个会话，证实参数 SORT_AREA_SIZE 确实被改变了。
SQL> BEGIN
      :result:=dbms_utility.get_parameter_value(parnam=>'sort_area_size',
      intval=>:sort_area_size, strval=>:dummy);
END;
/
PL/SQL procedure successfully completed.
SQL> PRINT sort_area_size
SORT_AREA_SIZE
-----
      1048576
```

20.9 SET_BOOL_PARAM_IN_SESSION 过程

该过程在外部会话中设置布尔型参数。

20.9.1 语法

```
DBMS_SYSTEM.SET_BOOL_PARAM_IN_SESSION(
      sid IN NUMBER,
      serial# IN NUMBER,
      parnam IN VARCHAR2,
      bval IN BOOLEAN);
```

20.9.2 参数

参 数	描 述
sid	会话标识符，对应于V\$SESSION.SID
serial#	会话序号，对应于V\$SESSION.SERIAL#
parnam	参数名，对应于V\$PARAMETER.NAME
bval	布尔值，即TRUE或FALSE

20.9.3 使用说明

传入一个错误的 SID 或 SERIAL#时不会引发异常。

20.9.4 示例

在一个 SID=12 和 SERIAL#=16 会话中设置布尔型参数 TIMED_STATISTICS 为 TRUE 实现如下：

```
SQL> EXEC dbms_system.set_bool_param_in_session(12, 16, 'timed_statistics', TRUE);
```

## 20.10 SET_EV 过程

该过程设置一个会话汇总的数字事件或通过制定转储的名称生成 SGA 或 PGA 中包含的信息转储（即 SYSTEMSTATE 转储）^①。它通常用于启用对 SQL 语句、等待事件和一个会话中显示性能问题与无关调用者会话之间绑定的跟踪。请注意，强烈建议跟踪等待事件（处于级别 8 和级别 12 的事件 10046）和/或绑定。建议不要使用 DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION，因为它在级别 1 启用 SQL 跟踪，即没有绑定变量和等待事件。

### 20.10.1 语法

```
DBMS_SYSTEM.SET_EV(  
    si IN BINARY_INTEGER,  
    se IN BINARY_INTEGER  
    ev IN BINARY_INTEGER  
    le IN BINARY_INTEGER  
    nm IN VARCHAR2);
```

### 20.10.2 参数

参 数	描 述
si	会话标识符，对应于V\$SESSION.SID
se	会话序号，对应于V\$SESSION.SERIAL#
ev	数字事件中介于10 000~10 999之间的事件编号，如10053（优化器跟踪）。如果nm不是NULL，并且ev是一个位于10 000~10 999之外的ORACLE错误编号，当会话抛出指定错误时，就采用以nm命名的转储。如果ev=65535并且nm不为NULL，就采用一个由nm指定类型的转储类事件（immediate dump）。这相当于ALTER SESSION SET EVENTS 'IMMEDIATE TRACE NAME event_name LEVEL level'，其中event_name为转储采用的名称（例如SYSTEMSTATE、PROCESSSTATE或ERRORSTACK）、level为事件级别。命名转储仅可能在调用者的会话内存在，不能在外部会话中采用
le	事件级别，0代表禁用事件。每个事件支持某些特定级别。最高级别通常为10
nm	名称，例如，用于采取诊断转储。允许的转储名称列表可以通过在SQL*Plus中以用户SYS调用ORADEBUG DUMPLIST获取。如果nm是一个空字符串（''，即在两个单引号之间没有任何字符），对DBMS_SYSTEM.SET_EV调用等效于ALTER SESSION SET EVENTS 'event TRACE NAME CONTEXT FOREVER, LEVEL level'，其中event为数字事件比如10046，level为事件级别

### 20.10.3 使用说明

当使用一个数字事件比如 10046，设置 nm 为一个空字符串（传入 NULL 并不起作用，尽管 DBMS 通常会认为空字符串为 NULL）。如果 SET_EV 没有被正确使用或不存在的 si 或 se，该过程调用就不会产生任何影响并且不会抛出异常。

### 20.10.4 范例

下面的例子说明了 DBMS_SYSTEM.SET_EV 的两个用途：

① SYSTEMSTATE 转储揭示了一个实例当前的状态，可供 Oracle Support 诊断一些悬而未决的问题。

- 在会话级别设置事件（推荐共享服务器数据库会话）；
- 采用命名转储。

### 1. 使用SET_EV启用SQL跟踪

让我们假设用户 HR 的会话非常消耗资源。因此我们就需要去跟踪 SQL 语句和等待事件，弄清楚到底是什么原因。下面的例子演示了怎样从 V\$SESSION 中获取 SID 和 SERIAL# 并使用 DBMS_SYSTEM.SET_EV 启用跟踪。

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> SELECT sid, serial# FROM v$session WHERE username='HR';
      SID      SERIAL#
-----
      140         862
SQL> EXECUTE dbms_system.set_ev(140, 862, 10046, 8, '')
PL/SQL procedure successfully completed.
```

### 2. 采用命名转储

命名转储只能在与调用 DBMS_SYSTEM.SET_EV 相同的会话中采用。为了在一个外部会话中采用转储类事件 ERRORSTACK，我们使用 ORADEBUG（见第 37 章）。下面的匿名块采用级别 3 的 ERRORSTACK 转储：

```
SQL> VARIABLE sid NUMBER
SQL> VARIABLE serial NUMBER
SQL> BEGIN
  SELECT sid, serial# INTO :sid, :serial
  FROM v$session
  WHERE sid=(SELECT sid FROM v$mystat WHERE rownum=1);
  sys.dbms_system.set_ev(:sid, :serial, 65535, 3, 'errorstack');
END;
/
```

通过一个更简单的 ALTER SESSION 语句可以得到相同的结果：

```
SQL> ALTER SESSION SET EVENTS 'IMMEDIATE TRACE NAME ERRORSTACK LEVEL 3';
```

错误栈转储显示了一个程序进行转储时正在执行哪一个子程序。它同样也列出了所有调用栈 (call stack)，即子程序以什么样的顺序互相调用。当前正在执行的子程序位于调用栈的顶部。ORACLE 调用栈和通过使用调试器（比如 adb、sdb 或 gdb）读取核心文件获取的调用栈相似。采用一个 ERRORSTACK 转储不会终止 ORACLE 进程。但是，你应该预期到该进程在写入跟踪文件时会停止响应。

在级别 3，错误栈转储包含打开的游标，因而有助于对应于某一游标编号找到 SQL 语句的文本，以防一个特定游标的 PARSING IN CURSOR 条目在 SQL 跟踪文件丢失。在 Oracle10g 中，游标会如下所示进行转储：

```
Cursor#1(07470C24) state=BOUND curiob=07476564
curflg=4c fl2=0 par=00000000 ses=6998B274
sqltxt(66F31AA0)=ALTER SESSION SET EVENTS 'IMMEDIATE TRACE NAME ERRORSTACK LEVEL 3'
```

## 20.11 SET_SQL_TRACE_IN_SESSION 转储

该过程在一个数据库会话中启用 SQL 跟踪。

### 20.11.1 语法

```
DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION(  
    sid IN NUMBER,  
    serial# IN NUMBER,  
    sql_trace IN BOOLEAN);
```

### 20.11.2 参数

参 数	描 述
sid	会话id, 对应于V\$SESSION.SID
serial#	会话序号, 对应于V\$SESSION.SERIAL#
sql_trace	TRUE则启动跟踪, FALSE则关闭跟踪

### 20.11.3 使用说明

当参数 SID 或 SERIAL#其中之一不正确时不会引发异常。仅在你确信不需要跟踪等待事件和绑定变量时使用封装过程 DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION。

### 20.11.4 范例

```
SQL> SELECT sid, serial# FROM v$session WHERE username='NDEBES';  
      SID      SERIAL#  
-----  
      35        283  
SQL> EXEC dbms_system.set_sql_trace_in_session(35, 283, true);
```

## 20.12 WAIT_FOR_EVENT 过程

该过程会导致被调用的会话人为地等待一个指定的事件几秒钟。该事件必须是来自 V\$EVENT_NAME.NAME 的一个等待事件。如果在级别 8 或 12 启用 SQL 跟踪, 人为生成的等待事件会被发送到一个跟踪文件。WAIT_FOR_EVENT 对于使用扩展 SQL 跟踪分析器的开发者很有用, 他们需要确认他们的分析器软件能够理解所有可能被发送到跟踪文件的等待事件。编写一个可以使用 Oracle10g 中 872 个等待事件测试一个分析器的软件是很困难的。Oracle10g 等待事件带有各种有意义的参数名代替了 Oracle9i 及更早版本中的 p1、p2、p3 等, 使编写这样一个软件更加困难。

### 20.12.1 语法

```
DBMS_SYSTEM.WAIT_FOR_EVENT(  
    event IN VARCHAR2,
```

```
extended_id IN BINARY_INTEGER,  
timeout IN BINARY_INTEGER);
```

20.12.2   参数

参   数	描   述
event	等待事件名称，对应于V\$EVENT_NAME.NAME
extended_id	事件的额外信息
timeout	等待一个事件的时间（单位为秒）

20.12.3   使用说明

该会话等待指定的超时时间并使用参数 EVENT 的值填充 V\$SESSION_WAIT.EVENT 以及 EXTENDED_ID 的值填充 V\$SESSION_WAIT.P1。对于拥有超过一个参数的等待事件，剩下的参数就被设置为默认值，这限制了测试扩展 SQL 跟踪分析器的过程的功效。在 Oracle9i 中，EXTENDED_ID 以格式 p1=extended_id 被发送到一个跟踪文件，而在 Oracle10g 中格式为 field_name=extended_id，其中 field_name 可以通过下一节介绍的 V\$EVENT_NAME.PARAMETER1 检索到。如果 event 不是来自 V\$EVENT_NAME 的有效事件，就会引发异常 ORA-29352: event 'event_name' is not an internal event。

20.12.4   范例

```
SQL> CONNECT / AS SYSDBA  
Connected.  
SQL> ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT FOREVER, LEVEL 8';  
Session altered.  
SQL> EXECUTE dbms_system.wait_for_event('db file scattered read', 1, 1);  
PL/SQL procedure successfully completed.  
SQL> EXECUTE dbms_system.wait_for_event('index block split', 204857603, 1);  
PL/SQL procedure successfully completed.  
SQL> ORADEBUG SETMYPID  
Statement processed.  
SQL> ORADEBUG TRACEFILE_NAME  
/opt/oracle/admin/ten/udump/ten_ora_2928.trc  
SQL> !grep 'scattered|split' /opt/oracle/admin/ten/udump/ten_ora_2928.trc  
BEGIN dbms_system.wait_for_event('db file scattered read', 1, 1); END;  
WAIT #1: nam='db file scattered read' ela= 993052 file#=1 block#=0 blocks=0 obj#=-1  
tim=456558169208  
BEGIN dbms_system.wait_for_event('index block split', 204857603, 1); END;  
WAIT #1: nam='index block split' ela= 994057 rootdba=204857603 level=0 childdba=0 ob  
j#=-1 tim=456559821542  
SQL> SELECT parameter1 FROM v$event_name WHERE name='db file scattered read';  
PARAMETER1  
-----  
file#
```

PL/SQL 包 DBMS_UTILITY 的大部分函数和过程都有详实的记录。过程 NAME_RESOLVE 接受一个数据库对象的名称并将其分解成它的组成部分,比如一个模式、一个对象名及潜在的数据库链接。根据 *Oracle Database PL/SQL Packages and Types Reference 10g Release 2* 的描述,该过程支持 4 种对象类型:同义词、过程、函数以及包。实际测试表明它总共支持 8 种对象类型,从而使它的功效得到扩展。

DBMS_UTILITY.NAME_RESOLVE 可用于应用程序处理数据库对象名称。它消除了从应用程序名称转换为完全合格的数据库对象指定的负担。SQL 标识符是大小写不敏感的,除非其由双引号包围。考虑到标识符可能大小写敏感或包含空格和标点字符,显然可以重用现有的功能。

## 21.1 NAME_RESOLVE 过程

本章讨论了封装过程 DBMS_UTILITY.NAME_RESOLVE 一些在文档中没有记录的方面。该过程解析一个可能包含引号、空格和大小写混合的名称到一个数据库对象的明确指代。返回参数值与数据字典中模式和对象名称的拼写相同。对象标识符与被解析对象的类型一起返回,其对应于 DBA_OBJECTS.OBJECT_ID。

### 21.1.1 语法

```
DBMS_UTILITY.NAME_RESOLVE (  
    name IN VARCHAR2,  
    context IN NUMBER,  
    schema OUT VARCHAR2,  
    part1 OUT VARCHAR2,  
    part2 OUT VARCHAR2,  
    dblink OUT VARCHAR2,  
    part1_type OUT NUMBER,  
    object_number OUT NUMBER);
```

21.1.2 参数

参 数	描 述
name	数据库对象的名称，格式为[schema.]identifier1[.identifier2]@database_link]，其中所有占位符为有效的SQL标识符。除了identifier1外所有部分都是可选的
context	在该上下文中解析名称。一个介于1和7之间的整数（并不是文档中所说的0和8） ^① 。不同对象类型的名称必须在正确的上下文中解析。见表21-1为对象类型到上下文的映射
schema	包含已解析对象的模式
part1	数据库对象名称
part2	低一级别的数据库对象名称，比如表中的一列或包中的一个过程。在该级别不检查其正确性
dblink	数据库链接名称，如果有的话在参数名称中出现
part1_type	代表part1对象类型的数字代码。见表21-2为类型代码到对象类型的映射
object_number	来自DBA_OBJECTS.OBJECT_ID的唯一数字对象标识符。如果名称包含一个数据库链接，那么object_number就为0

如果一个对象的类型是未知的，所有上下文都必须尝试去解析一个名称，表 21-1 列出了支持的对象类型以及其对应的上下文编号。集群、数据库链接（它们自身的）、目录、索引、LOB 列名称、队列、规则名称以及规则集都不能被解析。使用封装过程 DBMS_UTILITY.NAME_TOKENIZE 分解这些对象的名称可以在数据字典中实现更简单的查找。

表21-1 上下文参数vs对象类型

对象类型	上下文参数
Package（包）	1
Sequence（序列）	2
Synonym（同义词）	2
View（视图）	2
Table（表）	2
Trigger（触发器）	3
Type（类型）	7

与 OUT 参数 PART1_TYPE 有关的值范围见表 21-2。

表21-2 PART1_TYPE到对象类型的映射

PART1_TYPE	对象类型
0	数据库链接跟随的对象名称
2	Table

① 超出范围 1~7 的值会导致错误 ORA-20005: ORU-10034: context argument must be 1 or 2 or 3 or 4 or 5 or 6 or 7。



(续)

PART1_TYPE	对象类型
4	View
6	Sequence
7	Procedure
8	Function
9	Package
12	Trigger
13	Type

21.1.3 使用说明

要解析的名称可能包括 schema.table_name.column_name 或 schema.package_name.subroutine_name 中的三个标识符。这些表达式的类型被分别解析为 schema.table_name 和 schema.package_name。无论是对 column_name 还是 subroutine_name 都没有检查其正确性。当检测到一个数据库链接时，返回 PART1_TYPE=0 并且没有对对象名称也进行检查。要解析这些名称，连接到远程对象存在的数据库，并在那里重复进行名称解析或使用数据库链接执行一个远程过程调用。

21.1.4 异常

如果名称不能以指定的上下文进行解析，就会引发异常 ORA-06564: object object_name does not exist，其中 object_name 为传入到过程 NAME_RESOLVE 的参数 NAME 的值。如果一个已有的对象在错误的上下文中被解析，就会抛出异常 ORA-04047: object specified is incompatible with the flag specified。

21.1.5 范例

在 Oracle10g 中，公共同义词 PRODUCT_USER_PROFILE 指向模式 SYSTEM 中一个带有 SQL*Plus 配置数据的表。下面的例子解析了该公共同义词并显示它引用的表（见源代码库中的文件 name_resolve_table.sql）：

```
SQL> VARIABLE name VARCHAR2(100)
SQL> VARIABLE context NUMBER
SQL> VARIABLE schema VARCHAR2(30)
SQL> VARIABLE part1 VARCHAR2(30)
SQL> VARIABLE part2 VARCHAR2(30)
SQL> VARIABLE dblink VARCHAR2(30)
SQL> VARIABLE part1_type NUMBER
SQL> VARIABLE object_number NUMBER
SQL> BEGIN
    :context:=2; -- 1: package, 2: table
```

```

:name:= ' "SYSTEM" . Product_User_Profile '; -- name to resolve
DBMS_UTILITY.NAME_RESOLVE (
    name => :name,
    context => :context,
    schema => :schema,
    part1 => :part1,
    part2 => :part2,
    dblink => :dblink,
    part1_type => :part1_type,
    object_number => :object_number
);
end;
/
PL/SQL procedure successfully completed.

```

在成功调用 DBMS_UTILITY.NAME_RESOLVE 之后，绑定变量包含了被引用数据库对象的组成部分。

```

SQL> SELECT -- resolved_name
' ' || :schema || ' ' || nvl2(:part1, ' ' || :part1 || ' ', null) ||
nvl2(:part2, ' ' || :part2 || ' ', NULL) ||
nvl2(:dblink, '@' || :dblink || ' ', NULL) || ' is ' ||
-- translate part1_type to object type
decode(:part1_type, 0, 'an object at a remote database',
2, 'a table',
4, 'a view',
6, 'a sequence',
7, 'a procedure',
8, 'a function',
9, 'a package',
12, 'a trigger',
13, 'a type') ||
' (PART1_TYPE=' || :part1_type || ', OBJECT_NUMBER=' ||
:object_number || ' )' AS detailed_info
FROM dual;
DETAILED_INFO
-----
"SYSTEM"."SQLPLUS_PRODUCT_PROFILE" is a table (PART1_TYPE=2, OBJECT_NUMBER=10209)

```

OUT 参数 OBJECT_NUMBER 可用于从字典视图 DBA_OBJECTS 检索一个数据库对象的额外信息。

```

SQL> SELECT owner, object_name, object_type, status, created
FROM all_objects
WHERE object_id=:object_number;
OWNER  OBJECT_NAME                OBJECT_TYPE STATUS  CREATED
-----
SYSTEM SQLPLUS_PRODUCT_PROFILE TABLE    VALID   30.Aug.05

```

脚本最底端的 ALL_OBJECTS 的查询结果证实该名称被正确解析。因此，公共同义词 PRODUCT_USER_PROFILE 解析到表 SQLPLUS_PRODUCT_PROFILE 中。以包含一个数据库链接 ("SYSTEM". Product_User_Profile @ db_link) 的名称作为最后一个例子，结果如下：

```
DETAILED_INFO
```

```
-----
"SYSTEM"."PRODUCT_USER_PROFILE"@ "DB_LINK" is an object at a remote database
(PART1_TYPE=0, OBJECT_NUMBER=0)
no rows selected
```

这次，对 ALL_OBJECTS 的查询没有返回结果，因为 OUT 参数 OBJECT_NUMBER 的值为 0。

21

## 21.2 对象统计信息的名称解析和提取

一个 ORACLE 性能优化任务可能涉及基于成本的优化器（CBO）所使用的对象统计信息提取。我经常使用脚本 statistics.sql，它会报告表结构、表分区和子分区（如果存在）、表和索引基数、列中相异的值、相异的索引键、最近统计收集的时间戳、索引、表空间、数据库大小以及 LOB 列的详细信息。从该报告中可以看出一些问题，比如过时或丢失的统计信息、非选择性列上的索引，或会导致直接路径读或写等待的 LOB 中 NOCACHE 选项。该报告还可以方便地识别具有高选择性的列，它们是建索引非常好的备选方案，假定这些列在 where 从句中以谓词出现。我高度推荐使用该脚本去调查性能问题。

DBMS_UTILITY.NAME_RESOLVE 的使用使得调用该脚本更加方便，因为它足以在报告中提供一个表（或同义词）的名称的小写格式，而不是与数据字典（通常为大写）中拼写完全相同的所有者（或模式）和表。该脚本以一个数据库对象的名称作为输入，解析该名称，然后从 ALL_* dictionary 视图（这是很严肃的事情，并非全明星赛！）获取相关信息，比如 ALL_TABLES、ALL_INDEXES、ALL_LOBS 等。该脚本无需 DBA 特权运行，因为使用的是 ALL_* views，而没有使用 DBA_* views。运行该脚本的语法如下：

```
sqlplus -s user/password @statistics[.sql] { [schema.]table_name | synonym }
```

包含小写字母的数据字典对象名称必须被引用。以下图解显示了来自脚本 statistics.sql 的输出示例：

```
$ sqlplus -s hr/hr @statistics.sql employees
```

Table Owner and Name	Number of Rows	Blocks	Empty Blocks	Average Space	Chain Count	Average Row Len	Global Stats	User Stats	Sample Size	Last Analyze
HR . EMPLOYEES	107	5	0	0	0	68	YES	NO	2,000	01.Oct 07 18:09

Tablespace	Block- size	Size (MB)	Moni- toring Pool	Buffer IOT	Degree	Cluster Type	IOT Name
EXAMPLE	8 KB	0	YES	DEFAULT	1		

Column Name	Column Details	Distinct Values	Density	Number of Buckets	Number Nulls	Global Stats	User Stats	Sample Last Size Analyze
EMPLOYEE_ID	NUMBER(6), NOT NULL	107	.0093	1	0	YES	NO	01.Oct 07 18:09
FIRST_NAME	VARCHAR2(20)	91	.0110	1	0	YES	NO	01.Oct 07 18:09
LAST_NAME	VARCHAR2(25), NOT NULL	102	.0098	1	0	YES	NO	01.Oct 07 18:09
EMAIL	VARCHAR2(25), NOT NULL	107	.0093	1	0	YES	NO	01.Oct 07 18:09
PHONE_NUMBER	VARCHAR2(20)	107	.0093	1	0	YES	NO	01.Oct 07 18:09
HIRE_DATE	DATE, NOT NULL	98	.0102	1	0	YES	NO	01.Oct 07 18:09
JOB_ID	VARCHAR2(10), NOT NULL	19	.0047	19	0	YES	NO	01.Oct 07 18:09
SALARY	NUMBER(8,2)	57	.0175	1	0	YES	NO	01.Oct 07 18:09
COMMISSION_PCT	NUMBER(2,2)	7	.1429	1	72	YES	NO	01.Oct 07 18:09
MANAGER_ID	NUMBER(6)	18	.0047	18	1	YES	NO	01.Oct 07 18:09
DEPARTMENT_ID	NUMBER(4)	11	.0047	11	1	YES	NO	01.Oct 07 18:09

Index Owner and Name	Created	Last DDL	Last Analyze	Sample Buffer Size Pool	Degree
HR . EMP_EMAIL_UK	01.Oct 07 18:09	05.Oct 07 09:08	01.Oct 07 18:09	2,500	DEFAULT 1
HR . EMP_EMP_ID_PK	01.Oct 07 18:09	05.Oct 07 00:42	01.Oct 07 18:09	2,500	DEFAULT 1
HR . EMP_DEPARTMENT_IX	01.Oct 07 18:09	05.Oct 07 00:42	01.Oct 07 18:09	2,500	DEFAULT 1
HR . EMP_JOB_IX	01.Oct 07 18:09	05.Oct 07 00:42	01.Oct 07 18:09	2,500	DEFAULT 1
HR . EMP_MANAGER_IX	01.Oct 07 18:09	05.Oct 07 00:42	01.Oct 07 18:09	2,500	DEFAULT 1
HR . EMP_NAME_IX	01.Oct 07 18:09	05.Oct 07 00:42	01.Oct 07 18:09	2,500	DEFAULT 1

Index Owner and Name	Index Type	Uni Tree	Leaf Blks	Distinct Keys	Number of Rows	Average Leaf Blocks Per Key	Average Data Blocks Per Key	Cluster Factor	Global Stats	User Stats
HR . EMP_EMAIL_UK	NORMAL	YES	0	107	107	1	1	19	YES	NO
HR . EMP_EMP_ID_PK	NORMAL	YES	0	107	107	1	1	2	YES	NO
HR . EMP_DEPARTMENT_IX	NORMAL	NO	0	11	106	1	1	7	YES	NO
HR . EMP_JOB_IX	NORMAL	NO	0	19	107	1	1	8	YES	NO
HR . EMP_MANAGER_IX	NORMAL	NO	0	18	106	1	1	7	YES	NO
HR . EMP_NAME_IX	NORMAL	NO	0	107	107	1	2	15	YES	NO

Index Owner and Name	Column Name	Col Column Pos	Details
HR . EMP_DEPARTMENT_IX	DEPARTMENT_ID	1	NUMBER(4)
HR . EMP_EMAIL_UK	EMAIL	1	VARCHAR2(25), NOT NULL
HR . EMP_EMP_ID_PK	EMPLOYEE_ID	1	NUMBER(6), NOT NULL
HR . EMP_JOB_IX	JOB_ID	1	VARCHAR2(10), NOT NULL
HR . EMP_MANAGER_IX	MANAGER_ID	1	NUMBER(6)
HR . EMP_NAME_IX	LAST_NAME	1	VARCHAR2(25), NOT NULL
	FIRST_NAME	2	VARCHAR2(20)

## 21.3 源代码库

表 21-3 列出了本章的源代码文件以及它们的功能。

表21-3 DBMS_UTILITY源代码库

文 件 名	功 能
name_resolve.sql	该脚本包含一个匿名块，通过过程DBMS_UTILITY.NAME_RESOLVE解析所支持的任何对象类型的名称
name_resolve_procedure.sql	该脚本创建了一个存储过程NAME_RESOLVE。该过程解析DBMS_UTILITY.NAME_RESOLVE所支持的所有对象类型。过程NAME_RESOLVE与包DBMS_UTILITY.NAME_RESOLVE接受相同的参数，尝试所有的上下文，如果成功，返回和DBMS_UTILITY.NAME_RESOLVE相同的信息。此外，它返回合格的已解析名称。已解析名称单独的部分在双引号中返回，以保留大小写敏感
name_resolve_table.sql	使用DBMS_UTILITY解析一个表名称的示例
statistics.sql	该脚本报告表和索引列的优化器统计信息。用于在调整SQL语句时检查列的选择性和索引



# Part 7

第七部分

## 应用程序开发

### 本 部 分 内 容

- 第 22 章 Perl DBI 与 DBD::Oracle
- 第 23 章 应用程序插桩及端到端跟踪

Perl 是一种通用的解释型编程语言，支持访问很多常用的数据库系统。文档中没有提到 Oracle10g 和 Oracle11g 中 ORACLE_HOME 包含 Perl 的安装这一事实，其中包括用来访问 ORACLE DBMS 实例的 Perl 模块 DBI 和 DBD::Oracle。这样，就无需再安装 Perl、DBI 和 DBD::Oracle^①了，否则还需要下载 C 编译器并搭上宝贵的时间。可以用 Perl 和 DBI 来编写你自己的监测和基准测试工具、为长期的档案文件提取数据（包括 LOB）到平面文件以及从操作系统文件插入或更新 LOB。此外，Perl DBI 是很好的原型测试工具。可以通过复制 Perl 子目录以及 ORACLE_HOME 下的一些目录，给那些不需要完整安装 RDBMS 服务器的机器创建一个小型的 Perl DBI Oracle 客户端。

## 22.1 常见的 Perl DBI 陷阱

除了指出 Perl DBI 已经位于 Oracle10g 的 ORACLE_HOME 中并如何使用它，那么在本章我还能带给读者什么帮助呢？根据我的经验，Perl DBI 用户一般会挣扎于连接到一个 ORACLE 实例的很多方法和怎样在 Perl DBI 脚本中实现它们，因为 CPAN（Comprehensive Perl Archive Network，Perl 程序库，见 <http://www.cpan.org>）上的 Perl DBI 和 DBD::Oracle 文档并没有对此给出详尽的解释。

本章的目的是给你提供一个 ORACLE 环境下 Perl 编程的完整资料，涉及以下针对 ORACLE 的特定内容：

- 命名绑定变量；
- 通过 TCP/IP、bequeath 和 IPC 协议连接；
- 使用 SYSDBA 和 SYSOPER 特权连接；
- 使用带有（以及不带）Net 服务名称的连接字符串。

基本上任何一本书都应该能够自圆其说。然而，本书中不包括对 Perl、DBI 和 DBD::Oracle 的深入探讨。更何况，这将会使得页数变得不合适。通过给出解决 ORACLE 特定连接问题的详尽信息以及提供一个执行 SELECT、INSERT 语句、调用 PL/SQL 和包含事务与异常处理的样例 Perl 脚本，我希望能给 Perl DBI 新手灌输足够的信心去开始 Perl DBI 脚本编码。

^① 需要用来与 Perl 一起构建图形用户界面的 Tk 包没有包括在内。



就个人而言, 相比于使用 SQL*Plus 脚本和命令行 shell (比如 Bourne Shell 或 Korn Shell), 我强烈建议使用 Perl DBI。例如, Perl 中的错误处理就好很多, 多个数据库会话可以被同时打开, 文件操作也更加成熟, 等等。一旦你有一些编程的 Perl DBI 的实践, 开发时间将会大大减少。我鼓励所有 DBA、甚至是对 Perl 不甚了解的 DBA 去学习这种语言。一旦精通, 你将发现很多容易摘取的“果实”。

如果你在使用随 Oracle 软件发行的 Perl 过程中遇到任何 bug, 我建议你尝试来自 ActiveState 公司 (<http://www.activestate.com>) 最近的 Perl 发行版本。ActiveState 是一家专供脚本语言 (如 Perl) 的公司, 从 Perl 版本 5.10 开始提供包含 DBD::Oracle 的免费分发。ActiveState 公司提供的 DBI 和 DBD::Oracle 版本比随 Oracle11g 发行的版本要新很多。

## 22.2 Perl 与 DBI 简史

Perl 语言由 Larry Wall 发明, 于 1987 年首次发布。现在它使用 GNU 通用公共许可证和 Perl Artistic 许可证。但你知道吗? Larry 说, Perl 实际上代表 Pathologically Eclectic Rubbish Lister, 不要告诉别人哦!

缩写 Perl 实际上代表 Practical Extraction and Reporting Language。由 Larry Wall^①合著的 *Programming Perl* (又名 *The Camel Book* 或 *The Perl Bible*) 一书表示: “Perl 的设计目的是使简单的工作更简单, 而不是使艰难的工作变得不可能完成。” [WaCh 2000] 虽然乍看之下 Perl 就像解释型语言, 但其实际上使用与 Java 相当的虚拟机技术来编译和运行 Perl 代码。

Perl DBI (Database Independent Interface, 数据库独立接口) 由 Tim Bunce 编写和维护, 他是 *Programming the Perl DBI* [Bunc 2000] 一书的作者。他的座右铭是“让简单的事情简单, 让复杂的事情成为可能”。除了设计和实现 DBI, Tim Bunce 从 1994 年开始就是 Perl5 的移植者, 促进了 Perl 语言及其很多核心模块的开发。他在 <http://search.cpan.org/~timb> 提供了有关 Perl DBI 和 DBD::Oracle 的附加信息。

## 22.3 为 Perl 与 DBI 设置环境变量

作为 Oracle10g 与 Oracle11g 的 ORACLE_HOME 一部分分发的 Perl 安装驻留在目录 \$ORACLE_HOME/perl 中。它由 Oracle 公司的数据库控制和网格控制管理工具使用。Perl DBI 作为一个数据库客户端, 需要包含 OCI 例程的 ORACLE DBMS 客户端共享库与一个 ORACLE 实例通信。利用 \$ORACLE_HOME 中 Perl DBI 的唯一障碍就是寻找一个合适的 ORACLE_HOME 客户端共享库和设置平台特定的环境变量。

### 22.3.1 UNIX 环境

许多 UNIX 系统已经有了一个 Perl 安装。通常, 在这个安装中不包含数据库访问的 DBI 模块, 需要一个 C 语言编译器来安装它。更简单的方法是通过以下描述的步骤设置一些环境变量直

① 有关 Larry Wall 的更多信息, 请参见 [http://en.wikipedia.org/wiki/Larry_Wall](http://en.wikipedia.org/wiki/Larry_Wall)。

接使用 ORACLE_HOME 中的 Perl。

### 1. PATH

首先，需要修改可执行文件搜索路径变量 PATH，这样 \$ORACLE_HOME/perl/bin 就会先于随操作系统发行的 Perl 解释器目录被搜索到。

```
$ export PATH=$ORACLE_HOME/perl/bin:$PATH
$ which perl
/opt/oracle/product/db10.2/perl/bin/perl
```

### 2. PERL5LIB

第二步是设置 Perl 模块搜索路径。Perl 程序一般具有扩展名.pl，而 Perl 模块的扩展名为.pm。DBI 通过名为 DBI.pm 的文件实现。这就是 Perl 解释器需要找到的文件。幸运的是，该变量不取决于平台的字长或 Perl 的版本。一般可以使用下面的设置：

```
export PERL5LIB=$ORACLE_HOME/perl/lib:$ORACLE_HOME/perl/lib/site_perl
```

在一个 32 位的 Linux 系统上，所需要的文件是 \$ORACLE_HOME/perl/lib/site_perl/5.8.3/i686-linux-thread-multi/DBI.pm。Perl 同样通过搜索 PERL5LIB 指定目录的子目录找到该文件。它开始在为 Perl 构建版本和发行版本指定的目录中寻找，然后再尝试目录树中的目录，最后通过搜索指定目录停止。

在上面的例子中，Perl 的版本 (perl -version) 是 5.8.3，并且其构建版本为 i686-linux-thread-multi (Intel x86 32 位机架构并使用 Perl 多线程)。如果 PERL5LIB 设置不当，或 DBI 没有在 Perl 安装中出现（例如随操作系统发行的版本），你将会得到一个与下面类似的错误：

```
$ echo $PERL5LIB
/opt/oracle/product/db10.2/perl/lib
$ echo "use DBI;" | perl
Can't locate DBI.pm in @INC (@INC contains: /opt/oracle/product/db10.2/perl/lib/5.8.3/ i686-linux-thread-multi /opt/oracle/product/db10.2/perl/lib/5.8.3 /opt/oracle/product/db10.2/perl/lib) at dbi.pl line 3.
BEGIN failed--compilation aborted at dbi.pl line 3.
```

在前面的例子中，PERL5LIB 缺少目录 \$ORACLE_HOME/perl/lib/site_perl。解决办法是将该目录添加到 PERL5LIB。在设置多个搜索目录时使用冒号作为分隔符。如果你需要使用在 \$ORACLE_HOME/perl/lib 中没有找到的其他 Perl 模块，甚至编写你自己的模块（相信我，这并不难），你同样必须将它们的位置添加到 PERL5LIB。源代码库中包含一个基于 DBI 构建的 Perl 模块。

### 3. 共享库搜索路径

目前所有的 UNIX 系统实现都是使用共享库或者运行时链接而不是静态链接。使用静态链接，一些库（比如标准 C 语言库）就在运行时被静态地链接到程序。动态链接则在运行时添加该库到程序的文本段（其中是机器代码指令）。这种方法就会使得可执行文件更小，并拥有在可执行文件下一次运行时自动载入新版本共享库的优势。在许多 UNIX 平台，命令 ldd 可以用来找出一个可执行文件需要哪些共享库。

```
$ ldd `which perl`
libnsl.so.1 => /lib/libnsl.so.1 (0x0551c000)
libdl.so.2 => /lib/libdl.so.2 (0x00862000)
libm.so.6 => /lib/tls/libm.so.6 (0x00868000)
libcrypt.so.1 => /lib/libcrypt.so.1 (0x053f9000)
libutil.so.1 => /lib/libutil.so.1 (0x00cd8000)
libpthread.so.0 => /lib/tls/libpthread.so.0 (0x0097e000)
libc.so.6 => /lib/tls/libc.so.6 (0x00734000)
/lib/ld-linux.so.2 (0x0071a000)
```

在之前的输出中，libc.so.6 是标准 C 语言库，libm.so 是数学库，以及 libdl.so 是动态链接器本身的一部分。

正确的共享库搜索路径的方法的第一步是决定 \$ORACLE_HOME/perl/bin/perl 是一个 32 位可执行文件还是 64 位可执行文件。这可以通过使用 UNIX 命令文件完成。在一个 AIX 系统上，这可能会得到如下信息：

```
$ file $ORACLE_HOME/perl/bin/perl
/opt/oracle/product/10.2.0.2.1/perl/bin/perl: executable (RISC System/6000) or
object module not stripped
```

由于在前面的输出中没有提到 64 位，perl 就是一个 32 位可执行文件。在一个 32 位 Linux 系统上，它可能看起来像下面这样：

```
$ file `which perl`
/opt/oracle10/app/oracle/product/10.2.0.2/db_104/perl/bin/perl: ELF 32-bit LSB
executable, Intel 80386, version 1 (SYSV), for GNU/Linux 2.2.5,
dynamically linked (uses shared libs), not stripped
```

在该 Solaris 9 系统上，perl 是一个 64 位可执行文件。

```
$ file /opt/oracle/product/10.2.0.2.0/perl/bin/perl
/opt/oracle/product/10.2.0.2.0/perl/bin/perl: ELF 64-bit MSB executable SPARCv9
Version 1, dynamically linked, not stripped
```

下一步包括找到与 perl 有相同字长（32 位或 64 位）的 ORACLE 客户端共享库 libclntsh.so。ORACLE 安装可以是 32 位或 64 位。ORACLE DBMS 在 AIX、HP-UX 等平台上只有 64 位版本，而对于 Sparc Solaris、Solaris x86 以及 Linux 有 32 位版本。对于仅支持 32 位可执行文件的操作系统例如 Linux x86 问题很简单。在这些平台上，\$ORACLE_HOME/lib 包含一个合适的 32 位客户端共享库。

在一个 64 位 ORACLE_HOME 中，64 位共享库位于 \$ORACLE_HOME/lib 以及 32 位库位于 \$ORACLE_HOME/lib32 中。这里有一个来自 AIX 系统的例子：

```
$ file $ORACLE_HOME/lib/libclntsh.so $ORACLE_HOME/lib32/libclntsh.so
/opt/oracle/product/10.2.0.2.1/lib/libclntsh.so: 64-bit XCOFF executable or object
module not stripped
/opt/oracle/product/10.2.0.2.1/lib32/libclntsh.so: executable (RISC System/6000) or
object module not stripped
```

在一个 Solaris 系统上，它应该看起来如下所示：

```
$ file $ORACLE_HOME/lib/libclntsh.so $ORACLE_HOME/lib32/libclntsh.so
/opt/oracle/product/10.2.0.2.1/lib/libclntsh.so:      ELF 64-bit MSB dynamic lib
SPARCv9 Version 1, dynamically linked, not stripped
/opt/oracle/product/10.2.0.2.1/lib32/libclntsh.so:    ELF 32-bit MSB dynamic lib
SPARC Version 1, dynamically linked, not stripped
```

最后，下面是来自一个 64 位 Linux 系统的输出：

```
file $ORACLE_HOME/lib/libclntsh.so* $ORACLE_HOME/lib32/libclntsh.so*
/opt/oracle10/app/oracle/product/10.2.0.2/db_ rac/lib/libclntsh.so:      symbolic
link to `libclntsh.so.10.1'
/opt/oracle10/app/oracle/product/10.2.0.2/db_ rac/lib/libclntsh.so.10.1:  ELF 64-bit
LSB shared object, AMD x86-64, version 1 (SYSV), not stripped
/opt/oracle10/app/oracle/product/10.2.0.2/db_ rac/lib32/libclntsh.so:    symbolic
link to `libclntsh.so.10.1'
/opt/oracle10/app/oracle/product/10.2.0.2/db_ rac/lib32/libclntsh.so.10.1: ELF 32-bit
LSB shared object, Intel 80386, version 1 (SYSV), not stripped
```

现在我们准备好设置平台特定环境变量，其控制着共享库搜索路径。表 22-1 列出了最常见的平台以及在每一个平台上变量的名称。

表22-1 每一平台的共享库搜索路径环境变量

操作系统	共享库搜索路径环境变量
AIX	LIBPATH
HP-UX 32-bit	SHLIB_PATH
HP-UX 64-bit	LD_LIBRARY_PATH and SHLIB_PATH
Linux	LD_LIBRARY_PATH
Mac OS X ^①	DYLD_LIBRARY_PATH
Solaris	LD_LIBRARY_PATH
Tru64 UNIX	LD_LIBRARY_PATH

除非使用了正确的变量并设置了正确的搜索路径，否则 perl 就不能加载 ORACLE 驱动程序模块 DBD::Oracle。最简洁的测试包括运行如下命令：

```
$ echo "use DBI;use DBD::Oracle" | perl
Can't load '/opt/oracle/product/db10.2/perl/lib/site_perl/5.8.3/i686-linux-thread-multi/auto/DBD/Oracle/Oracle.so' for module DBD::Oracle: libclntsh.so.10.1: cannot open shared object file: No such file or directory at /opt/oracle/product/db10.2/perl/lib/5.8.3/i686-linux-thread-multi/DynaLoader.pm line 229.
at - line 1
Compilation failed in require at - line 1.
BEGIN failed--compilation aborted at - line 1.
```

如果你得到这样一个错误，你需要修复共享库搜索路径。错误信息在每个平台有些微小的差别，但总是提到无法找到 libclntsh.so。

下面的 Perl DBI 程序名为 perl-dbi-test.pl，用来测试到一个 ORACLE 实例的连接非常理想。

① Oracle10g 第一个发行版本适用于 Mac OS X。在撰写本书时，Oracle10g 第二个发行版本并没有在 Mac OS X 上运行的计划。

它提示输入用户名、密码以及 Net 服务名称，然后尝试连接到 DBMS。如果成功，它选择数据库登录用户名并打印到屏幕。

```
#!/usr/bin/env perl
# RCS: $Header: /home/ndebes/it/perl/RCS/perl-dbi-test.pl,v 1.1 2007/01/26 16:07:13
ndebes Exp ndebes $

# Perl DBI/DBD::Oracle Example

use strict;
use DBI;

print "Username: \n";
my $user = <STDIN>;
chomp $user;
print "Password: \n";
my $passwd = <STDIN>;
chomp $passwd;
print "Net Service Name (optional, if ORACLE instance runs locally and ORACLE_SID is
set): \n";
my $net_service_name = <STDIN>; # Oracle Net service name from tnsnames.ora or other
name resolution method
chomp $net_service_name;

if ($net_service_name) {
    print "Trying to connect to $user/$passwd@$net_service_name\n";
}
else {
    print "Trying to connect to $user/$passwd\n";
}

# Connect to the database and return a database handle
my $dbh = DBI->connect("dbi:Oracle:${net_service_name}", $user, $passwd)
    or die "Connect failed: $DBI::errstr";

my $sth = $dbh->prepare("SELECT user FROM dual"); # PARSE
$sth->execute(); # EXECUTE
my @row = $sth->fetchrow_array(); # FETCH
printf ("Connected as user %s\n", $row[0]);
$sth->finish;
$dbh->disconnect; # disconnect from ORACLE instance
```

如下所示例子为在 64 位 ORACLE_HOME 中设置 32 位的 Perl。

```
export PATH=$ORACLE_HOME/perl/bin:$ORACLE_HOME/bin:/usr/bin:/usr/ccs/bin
export PERL5LIB=$ORACLE_HOME/perl/lib:$ORACLE_HOME/perl/lib/site_perl
export LD_LIBRARY_PATH=$ORACLE_HOME/lib32
```

让我们确认一下设置是正确的。

```
$ perl perl-dbi-test.pl
Username:
ndebesPassword:
```

```
secret
Net Service Name (optional, if ORACLE instance runs locally and ORACLE_SID is set):
ten_tcp.world
Trying to connect to ndebes/secret@ten_tcp.world
Connected as user NDEBES
```

该脚本顺利执行。到 DBMS 的连接以及 SELECT user FROM dual 语句的执行顺利完成没有出错。

### 22.3.2 Windows 环境

在 Windows 上, OUI (Oracle10g Release 2 Universal Installer, Oracle10gR2 通用安装程序) 设置系统环境变量 PATH 和 PERL5LIB (如图 22-1 所示)。你可以通过依次点击“控制面板”>“系统”>“高级”>“环境变量”并浏览系统变量列表查看 PATH 和 PERL5LIB。然而, 只需要添加 %ORACLE_HOME%\bin 到 PATH, 除了 PERL5LIB 中正确的目录外, 它还设置了一些不存在 (例如, %ORACLE_HOME%\perl\5.8.3\lib\MSWin32-x86) 的目录和不必要的目录。

在 Windows 系统上, 环境变量可以被设置为系统全局和用户特定以及在命令行解释器 cmd.exe^①的一个实例中。用户特定设置会覆盖系统全局设置。如果命令行解释器中一个环境变量发生改变, 就会使用被修改的值而不是通过控制面板设置的用户或系统变量。在启动时, 命令行解释器从用户特定和系统全局继承环境变量。如果你没有修改系统全局设置的权限, 仍然可以通过用户特定设置覆盖它们。

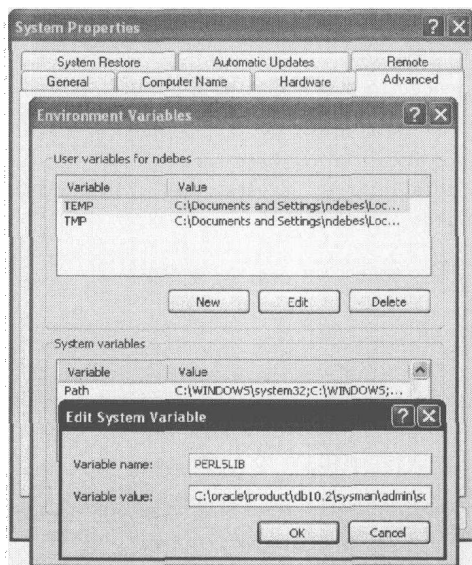


图 22-1 PERL5LIB 系统环境变量

① 点击“开始”>“运行”或按住 Windows 键 (会显示标志) 然后按 R 键, 输入 cmd, 然后点击 OK 启动一个命令行解释器。

当然，直接通过“控制面板”➤“系统”给所有用户一次性设置变量总比每次启动一个命令行解释器找寻带有环境变量的文件要方便很多。一个聪明的办法是在一个命令行解释器中设置环境变量用来测试，如果结果正确则把它们存储在注册表中。这是在后面的章节所采取的做法。

首先，我们将设置 ORACLE_HOME，这样在设置剩下的环境变量时它就可以被重用。

```
C:> set ORACLE_HOME=C:\ORACLE\product\db10.2
```

下一步我们需要添加 perl.exe 所在目录到 PATH。在 Windows 系统上，它驻留在构建版本特定目录 MSWin32-x86-multi-thread。指定多个目录的分隔符是分号 (;)。

```
C:> set PATH=%ORACLE_HOME%\perl\5.8.3\bin\MSWin32-x86-multi-thread;%ORACLE_HOME%\bin;C:\WINDOWS\System32
```

额外的 Windows 平台特定目录就从上面的设置中删除了。现在就可以运行 perl.exe 了：

```
C:> perl -version
This is perl, v5.8.3 built for MSWin32-x86-multi-thread
```

```
Copyright 1987-2003, Larry Wall
```

```
Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.
```

```
Complete documentation for Perl, including FAQ lists, should be found on
this system using `man perl' or `perldoc perl'. If you have access to the
Internet, point your browser at http://www.perl.com/, the Perl Home Page.
```

如果你打算使用除了 Perl 解释器本身之外的其他 Perl 安装组件，你需要添加%ORACLE_HOME%\perl\5.8.3\bin 到 PATH，然后你可以使用 perldoc 读取 Perl 安装或 pod2html 中的文档库，从 Perl 源文件（见 perldoc perlpod）中嵌入的 POD（Plain Old Documentation）语句生成 HTML 文档。

```
C:> set PATH=%ORACLE_HOME%\perl\5.8.3\bin\MSWin32-x86-multi-thread;%ORACLE_HOME%\perl\5.8.3\bin;%ORACLE_HOME%\bin;C:\WINDOWS\System32
```

```
C:> perldoc perltoc
NAME
    perltoc - perl documentation table of contents
```

#### DESCRIPTION

```
This page provides a brief table of contents for the rest of the Perl
documentation set. It is meant to be scanned quickly or grepped through
to locate the proper section you're looking for.
```

#### BASIC DOCUMENTATION

```
perl - Practical Extraction and Report Language
```

```
...
```

事实证明，并不需要在 Windows 系统的 ORACLE_HOME 中设置 PERL5LIB，Perl 也能正常工作，因为 perl.exe 自动从它被调用的目录树中进行搜索。在 Windows 系统上，没有单独用来搜索 DLL（Dynamic Link Library，动态链接库，Windows 上各种各样的共享库）的环境变量。PATH 既作为

命令搜索路径也作为 DLL 搜索路径。需要用于连接到一个 DBMS 实例的 ORACLE DLL 是 OCI.DLL。由于用于运行 perl.exe 的 perl58.dll 和 perl.exe 一起配置在 %ORACLE_HOME%\perl\5.8.3\bin\MSWin32-x86-multi-thread, 以及 OCI.DLL 在 %ORACLE_HOME%\bin, 你所需要做的事情就是设置一次 PATH, 使它包含这两个目录。

OUI 同样在控制面板中添加目录 %ORACLE_HOME%\sysman\admin\scripts 到 PERL5LIB 的设置。这应该被保留以防损坏数据库控制或网格控制。请注意, 一个网格控制管理代理 (Grid Control Management Agent) 的安装同样包含了一个包括 DBI 和 DBD::Oracle 的 perl 安装。毕竟, 这些组件都是使用 Perl DBI 构建的。所以你其实可以用如下方式简化环境变量 PERL5LIB:

```
set PERL5LIB=<ORACLE_HOME>\sysman\admin\scripts
```

尽管记得在通过控制面板调整用户或系统环境变量中的设置时使用 ORACLE_HOME 的实际路径, 从那里读取该变量时 %ORACLE_HOME% 没有被展开。

让我们重复之前在 UNIX 系统上执行的测试, 确认设置是正确的:

```
C:> set ORACLE_HOME
ORACLE_HOME=C:\oracle\product\db10.2
C:> set PATH

Path=C:\oracle\product\db10.2\perl\5.8.3\bin\MSWin32-x86-multi-thread;C:\oracle\product\db10.2\bin;C:\WINDOWS\System32

PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
C:> set PERL5LIB
PERL5LIB=C:\oracle\product\db10.2\sysman\admin\scripts
C:> perl perl-dbi-test.pl
Username:
ndebes
Password:
secret
Net Service Name (optional, if ORACLE instance runs locally and ORACLE_SID is set):
TEN.oradbpro.com
Trying to connect to ndebes/secret@TEN.oradbpro.com
Connected as user NDEBES
```

## 22.4 在 UNIX 系统上透明地运行 Perl 程序

Perl 脚本总是可以通过输入 Perl 解释器的名称 (perl) 然后将 Perl 脚本的名称作为参数传入去运行, 如下所示:

```
$ cat args.pl
print "Script name: $0\n";
$ perl args.pl
Script name: args.pl
```

当一个 UNIX 文本文件使用 chmod +x filename 变为可执行文件, 并使用 ./filename 执行时, 就会产生一个 UNIX 系统默认的 shell 并将文件名作为参数传递进去。Linux 的默认 shell 是 bash (Bourne Again Shell), 而 sh (Bourne Shell) 是大多数其他系统的默认命令。一个文本文件第一



行的伪代码注释`#!`用来指定除默认 shell 之外其他的程序去处理该文本文件。乍一看，恰当的做法是将 Perl 解释器的绝对路径放置在伪代码注释（`#!`）之后：

```
$ cat args.pl
#!/usr/bin/perl
print "Script name: $0\n";
$ chmod +x args.pl
$ ./args.pl
Script name: ./args.pl
```

这样做没有什么问题，但是移植性不高。在另外一个系统上，Perl 解释器可能就位于目录 `/usr/local/bin/perl` 而不是 `/usr/bin/perl`，那么 Perl 脚本在这样一个系统上就不能运行。

```
$ ./args.pl
: bad interpreter: No such file or directory
```

更好的方法是通过首先使用 UNIX 命令 `env` 这种间接的方式作为解释器可执行文件，`env` 在所有 UNIX 系统上都位于目录 `/usr/bin`。`env` 具有基于命令行参数设置环境变量并运行其他程序的能力。当 `env` 尝试去定位要运行的程序时考虑了环境变量 `PATH`。因此，脚本代码在其他系统上运行时不需要做出修改。只有 `PATH` 变量必须包含 Perl 解释器所在目录。

```
$ cat args.pl
#!/usr/bin/env perl
print "Script name: $0"
$ ./args.pl
Script name: ./args.pl
```

## 22.5 在 Windows 系统上透明地运行 Perl 程序

问题在 Windows 系统上稍微复杂一些。但话又说回来，Windows 提供忽略扩展名 `.pl` 这一选项，但是 Perl 程序普遍带有扩展名。就像他们所说，你得到一些，就会失去一些。采用如下方法，你看如何？首先，你给 Windows 定义一个新的文件类型并告知它由哪一个可执行文件负责该文件类型。显然，可执行文件为 `perl.exe`。请注意，Windows 需要可执行文件的绝对路径。在搜索可执行文件时环境变量 `PATH` 没有被考虑进来，虽然环境变量可能在定义中使用。命令 `FTYPE` 用于定义文件类型。在下面的代码示例中我使用文件类型 `PerlProgram`：

```
C:> FTYPE PerlProgram="C:\oracle\product\db10.2\perl\5.8.3\bin\MSWin32-x86-multi-thread\perl.exe" %1 %*
PerlProgram="C:\oracle\product\db10.2\perl\5.8.3\bin\MSWin32-x86-multi-thread\perl.exe" %1 %*
```

在这里，Oracle10g 安装在 `C:\oracle\product\db10.2`。假设你已经定义了环境变量 `ORACLE_HOME`，你同样可以像这样调用 `FTYPE`：

```
C:> set ORACLE_HOME=C:\oracle\product\db10.2
C:> FTYPE PerlProgram="%ORACLE_HOME%\perl\5.8.3\bin\MSWin32-x86-multi-thread\perl.exe" %1 %*
PerlProgram="C:\oracle\product\db10.2\perl\5.8.3\bin\MSWin32-x86-multi-thread\perl.exe" %1 %*
```

字符串%1和%*分别代表传递到 perl.exe 的第一个参数和从第二个开始到最后一个参数。前者(%1)将包含到脚本文件的完整路径,而后者(%*)将传递剩下的参数到 perl.exe。

其次,Windows 需要文件名称的扩展名与文件类型的关联。关联关系通过命令 ASSOC 维护。由于 Perl 程序的文件名称的扩展名是.pl,我们需要去建立.pl 与我们之前定义的文件类型 PerlProgram 之间的关联。

```
C:> ASSOC .pl=PerlProgram
.pl=PerlProgram
```

现在,无需使用 perl filename.pl 运行 Perl 程序,仅输入 filename.pl 就已经足够了。为了证明这两种方法确实是等价的,我们将使用如下的 Perl 程序:

```
print "Script name: $0\n";
for ($i=0; $i < 10; $i++) {
    if (defined $ARGV[$i]) {
        printf "Argument %d: %s\n", $i, $ARGV[$i];
    }
}
```

下面是以老的方式运行 args.pl 的输出,仅仅需要多打一些字而已:

```
C:> perl args.pl first second third fourth
Script name: args.pl
Argument 0: first
Argument 1: second
Argument 2: third
Argument 3: fourth
```

args.pl 所在目录应该在命令搜索路径变量 PATH 中。利用之前定义的关联,你现在可以按照如下方式运行:

```
C:> args.pl first second third fourth
Script name: C:\home\ndebes\it\perl\args.pl
Argument 0: first
Argument 1: second
Argument 2: third
Argument 3: fourth
```

两种方法的结果是相同的,除了脚本名称,它在使用关联时显示的是完整路径。在定义了.pl 作为可执行文件的额外扩展后,你就完全可以沉浸在懒惰中了。这是通过使用环境变量 PATHEXT 实现的。

```
C:> set PATHEXT
PATHEXT=.COM;.EXE;.BAT;.CMD
C:> set PATHEXT=%PATHEXT%;.PL
C:> args first second third fourth
Script name: C:\home\ndebes\it\perl\args.pl
Argument 0: first
Argument 1: second
Argument 2: third
Argument 3: fourth
```

通过使用一个单独的空白作为文件类型就可以关联，如下所示：

```
C:> ASSOC .pl= ␣
.pl=
```

图形字符␣代表一个空白符。

## 22.6 连接到一个 ORACLE DBMS 实例

Perl DBI 编程新手经常遇到问题的地方就是决定连接到一个 ORACLE DBMS 实例的众多方法中哪一个最合适。Perl DBI 文档仅仅涵盖了部分这方面的内容，因此我决定包含一个对所有现存变化的概述。

表 22-2 列出了用于连接到一个 ORACLE 实例最常见的 Oracle Net TNS (Transparent Network Substrate, 透明网络底层) 协议适配器。使用带有 IPC 或 TCP 的 Oracle Net 服务名称描述而不是从配置文件 tnsnames.ora 中获取的选项暂时被忽略。

表22-2 Oracle Net协议适配器

方 法	需要监听与否	需要tnsnames.ora 与否	说 明
Bequeath adapter	no	no	必须配置环境变量ORACLE_SID。在UNIX系统上ORACLE_HOME同样也必须配置
IPC adapter	yes	yes	用于没有网络适配器或其中数据库客户端使用与服务器不同的本地ORACLE_HOME
TCP/IP adapter	yes	yes	最常用的方法，TCP/IP网络基础设施必须到位

### 22.6.1 DBI 连接语法

建立一个数据库会话的 DBI 调用是 connect。它的语法如下：

```
$dbh = DBI->connect($data_source, $username, $auth, \%attr);
```

在 Perl 中，标量变量的前缀为\$，然后通过%散列（解释术语 scalar 和 hash 请运行 perldoc perlintro）。表 22-3 解释了 connect 调用的参数。

表22-3 connect参数

参 数	意 思
\$dbh	数据库会话的数据库句柄
\$data_source	数据源即连接到一个本地ORACLE实例或联系一个侦听的规范。支持如下的格式： "DBI:Oracle:", "DBI:Oracle:<Net service name>", "DBI:Oracle:host=<host_name>;port=<port_number>;sid=<ORACLE_SID>", "DBI:Oracle:<host_name>:<port_number>/<instance_service_name>", ^① 或者undef
\$username	数据库用户名，"/" 为OS认证，或者undef
\$auth	密码或者undef
\%attr	对一个带有连接选项的散列可选引用

① Oracle Net 的简便连接格式在 Oracle10g 及其后的版本中有效。

Perl 关键字 `undef` 代表一个未定义的值，与 SQL 中 `NULL` 类似。如果 `$data_source`、`$username` 或 `$auth` 之一为 `undef`，那么如果环境变量可以访问，就会使用环境变量。表 22-4 列出了参数以及相应的环境变量。

表22-4 Perl DBI环境变量

参 数	环境变量
<code>\$data_source</code>	<code>DBI_DSN</code>
<code>\$username</code>	<code>DBI_USER</code>
<code>\$auth</code>	<code>DBI_PASS</code>

`$data_source` 的值必须以字符串 `"DBI:Oracle:"` 开头。请注意字符串 `"Oracle"` 中的大写字母 `"O"` 以及字符串 `"Oracle"` 之后的冒号 `":"` 是必须要有的。使用小写字母 `"o"` 将会导致如下错误：

```
DBD::oracle initialisation failed: Can't locate object method "driver" via package
"DBD::oracle"
```

DBI 使用字符串 `"Oracle"` 构建大小写敏感的驱动模块名称。连接到 ORACLE 实例的驱动模块是 `DBD::Oracle` 而不是 `DBD::oracle`，否则会出错。在市场上也存在用于大多数数据库产品的驱动模块。`DBD::ODBC` 就是一个支持 ODBC（Open Database Connectivity，开放式数据库连接）、可工作于任何数据库产品的驱动。

22.6.2 使用 Bequeath 适配器连接

当使用 `bequeath` 适配器连接一个 DBMS 实例时，请确保环境变量 `ORACLE_HOME`（UNIX 系统上必需，Windows 系统上可选）和 `ORACLE_SID` 已经设置。下面两行 Perl 代码足以用于连接：

```
use DBI;
$dbh = DBI->connect("DBI:Oracle:", "ndebes", "secret") or die "Connect failed:
$DBI::errstr";
```

在 Windows 系统上，如果 `ORACLE_SID` 没有正确设置就会抛出如下异常：

```
Connect failed: ORA-12560: TNS:protocol adapter error (DBD ERROR: OCIServerAttach).
```

该错误消息显示实现 ORACLE DBMS 实例的 Windows 服务没有运行起来。该 Windows 服务可以用命令 `net start OracleServiceORACLE_SID` 来启动。在 UNIX 系统上，错误消息是不同的，这是由 ORACLE DBMS 在 UNIX 和 Windows 系统上架构的重大差异（UNIX 系统上采用共享内存多进程架构，而 Windows 系统上采用单进程多线程架构）导致的。

```
$ export ORACLE_SID=ISDOWN
$ perl dbi.pl
DBI connect('', 'ndebes', ...) failed: ORA-01034: ORACLE not available
ORA-27101: shared memory realm does not exist
Linux Error: 2: No such file or directory (DBD ERROR: OCISessionBegin) at dbi.pl
line 5
$ unset ORACLE_SID
$ perl dbi.pl
```

```
DBI connect('', 'ndebes', ...) failed: ORA-12162: TNS:net service name is incorrectly
specified (DBD ERROR: OCIServerErrorAttach) at dbi.pl line 5
```

顺便说一下，当使用 `bequeath` 适配器连接时，`V$SESSION.SERVICE_NAME` 总是为默认值 `SYS$USERS`。

### 22.6.3 使用 IPC 适配器连接

使用 IPC 协议意味着，DBMS 实例、侦听器以及客户端运行在同一系统上。对于仅支持 IPC 协议的侦听器配置文件 `listener.ora` 中的一个条目如下：

```
LISTENER =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = TEN))
    )
  )
```

通过之前代码段中所定义的侦听器来完成连接的文件 `tnsnames.ora` 中一个 Net 服务名称定义如下：

```
TEN_IPC.WORLD =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = TEN))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = TEN)
    )
  )
```

通过 `bequeath` 进行连接与通过 IPC 进行连接之间的唯一区别在于，`tnsnames.ora` 中 Net 服务名称（例如 `TEN_IPC.WORLD`）被插入在 `$data_source` 中 `DBI:Oracle:` 之后，并且没有必要设置环境变量 `ORACLE_SID`，因为参数 `SERVICE_NAME` 的值反而来自 Oracle Net 服务的名称定义。

```
use DBI;
my $dbh = DBI->connect("DBI:Oracle:TEN_IPC.WORLD", "ndebes", "secret") or die
"Connect failed: $DBI::errstr";
```

Net 服务名称定义在区域 `CONNECT_DATA` 可能还包含 `(SID=TEN)` 而不是 `(SERVICE_NAME=TEN)`。推荐使用 `SERVICE_NAME`，因为服务名称会反映在视图 `V$SESSION` 中，并且服务名称与 `DBMS_MONITOR` 一起使用还可以跟踪会话的子集。

```
SQL> SELECT sid, service_name, module, action
FROM v$session
WHERE program='perl.exe';
  SID SERVICE_NAME      MODULE      ACTION
-----
  137 TEN              perl.exe
SQL> EXEC dbms_monitor.serv_mod_act_trace_enable('TEN','perl.exe','', true, true);
PL/SQL procedure successfully completed.
```

不幸的是，使用这种方法，视图 V\$SESSION 并不能反映跟踪已经被开启的事实。

```
SQL> SELECT sql_trace, sql_trace_waits, sql_trace_binds
FROM v$session
WHERE sid=137;
SQL_TRACE SQL_TRACE_WAITS SQL_TRACE_BINDS
-----
DISABLED FALSE FALSE
```

当参数 SID 在 tnsnames.ora 中被使用时，V\$SESSION.SERVICE_NAME 具有相当无意义的默认设置 SYS\$USERS。

## 22.6.4 通过 TCP/IP 适配器连接

TCP/IP 协议的使用使得客户端可以从一个网络中的任意系统连接到 DBMS 实例。下面是对一个 TCP/IP 连接的 Oracle Net 服务名称定义示例：

```
TEN_TCP.WORLD =
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST=dbserver.oradbpro.com)(PORT = 1521))
  )
  (CONNECT_DATA =
    (SERVICE_NAME = TEN)
  )
)
```

侦听器也需要支持 TCP/IP 协议。

```
LISTENER =
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = IPC)(KEY = TEN))
    (ADDRESS = (PROTOCOL = TCP)(HOST=dbserver.oradbpro.com)(PORT = 1521))
  )
)
```

这是使用服务名称 TEN_TCP.WORLD 连接的 Perl 代码：

```
use DBI;
my $dbh = DBI->connect("DBI:Oracle:TEN_TCP.WORLD", "ndebes", "secret") or die
"Connect failed: $DBI::errstr";
```

正如你所看到的，该 Perl 代码与 IPC 连接类似。只是作为 \$data_source 的一部分传递的 Oracle Net 服务名称已经改变。用于连接的三个参数都可以为 undef，如果相对应环境变量 DBI_DSN、DBI_USER 以及 DBI_PASS 都已经使用 Windows 系统上的 SET 和 UNIX 系统上的 export 进行设置。

```
my $dbh = DBI->connect(undef, undef, undef) or die "Connect failed: $DBI::errstr";
```

存在另外两种无需使用命名方法（比如，本地命名即 tnsnames.ora，或轻量目录访问协议 LDAP）进行 Net 服务名称解析并通过 TCP/IP 适配器连接的方法。第一种语法类似于 Java JDBC URL，其指定主机 (host)、端口 (port) 以及 ORACLE_SID 作为数据源 \$dsn 的一部分。如下例所示：

```
my $dbh = DBI->connect("DBI:Oracle:host=dbserver.oradbpro.com;port=1521;sid=TEN",
"ndebes", "secret")
or die "Connect failed: $DBI::errstr";
```

端口号可以忽略。如果被忽略，就会尝试1521和1526两个端口。字段 host、port 和 sid 的顺序是无关紧要的。另一种方法是给参数 \$dsn 提供完整的描述。这可以从配置文件 tnsnames.ora 中获取到。

```
my $dbh = DBI->connect( "DBI:Oracle:(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)
(P)(HOST=dbserver.oradbpro.com)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=TEN)))", "nde
bes", "secret") or die "Connect failed: $DBI::errstr";
```

再次，相比于 SID，在 CONNECT_DATA 区域更倾向于使用 SERVICE_NAME。

### 22.6.5 简易连接

在 Oracle10g 及其后续发行版本中，Oracle Net 简易连接可用于连接字符串。一个简易连接规格包括格式 host_name:port/instance_service_name。所有需要用于联系一个侦听器的信息都嵌入在连接字符串中，因此就无需进行 Net 服务名解析。下面是一个使用简易连接的 DBI 连接调用：

```
my $dbh = DBI->connect("DBI:Oracle:dbserver:1521/ELEVEN", "ndebes", "secret") or die
"Connect failed: $DBI::errstr";
```

相比于老的格式 host=host_name;port=port_number;sid=ORACLE_SID，我们更倾向于使用简易连接，因为它使用实例服务名称^①。

### 22.6.6 使用 SYSDBA 或 SYSOPER 特权连接

自从 DBMS 发行版本 Oracle9i，以用户 SYS 连接仅限于 SYSDBA 特权。SYSDBA 特权可以通过操作系统用户组成员身份或在使用密码文件（REMOTE_LOGIN_PASSWORDFILE=exclusive 或 shared）时通过授予 SYSDBA 这两种方式分配。

必须注意区分 SYSDBA（或 SYSOPER）特权与操作系统认证，这将会在下一节讨论。连接字符串"/ AS SYSDBA"同时使用二者。斜线 (/) 说明使用了操作系统认证而不是密码文件认证，而 AS SYSDBA 指明该会话需要 SYSDBA 特权。我将首先示范怎样使用数据库密码文件认证以 SYSDBA 连接。存在下列的条件。

- DBMS 实例必须使用 REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE 或 SHARED 启动。
- 事前使用 orapwd 创建的密码文件必须存在（否则实例将不会以上面的参数设置启动）。在 UNIX 系统上，密码文件路径名为 \$ORACLE_HOME/ dbs/orapw\$ORACLE_SID，在 Windows 系统上则为 %ORACLE_HOME%\database\pwd%ORACLE_SID%.ora。
- 必须授予打算以 AS SYSDBA 连接的用户 SYSDBA 特权（在 Windows 系统上的 sqlnet.ora 文件中，SQLNET.AUTHENTICATION_SERVICES 没有被设置或不包含 NTS，否则就可能使用操作系统认证）。

^① 请参考本书开始的介绍部分中对术语 instance service name 的定义。

当满足这些条件时，就可以在 SQL*Plus 中使用任一协议适配器 bequeath、IPC 或 TCP/IP 以 AS SYSDBA 连接。

```
$ sqlplus "ndebes/secret@ten_tcp.world AS SYSDBA"
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
SQL> EXIT
Disconnected from Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
```

在 SYSDBA 特权从用户 NDEBES 撤销之后该连接命令就会失败。

```
SQL> SHOW USER
USER is "SYS"
SQL> REVOKE SYSDBA FROM ndebes;
Revoke succeeded.
SQL> CONNECT ndebes/secret@ten_tcp.world AS SYSDBA
ERROR:
ORA-01031: insufficient privileges
```

现在我们可以使用与 Perl DBI 相同的方式测试连接了。到目前为止，我还没有解决连接方法的参数 %attr。反斜线说明该方法需要一个引用，而百分号 (%) 则说明其需要一个 Perl 散列。数据库驱动模块中定义的特殊常量必须作为 \%attr 的一部分传入，使得其能以 SYSDBA 或 SYSOPER 连接。因此，我们不能再依靠 DBI 为我们自动加载驱动模块 DBD::Oracle。相反，我们必须用命令 use DBD::Oracle 显式加载该模块并请求常量 ORA_SYSDBA 和 ORA_SYSOPER 被加载到 Perl 符号表。下面是以与之前使用 SQL*Plus 相同的方式实现连接的 Perl 代码：

```
use DBI;
use DBD::Oracle qw(:ora_session_modes); # imports ORA_SYSDBA and ORA_SYSOPER
my $dbh = DBI->connect("DBI:Oracle:", "ndebes", "secret",
    {ora_session_mode => ORA_SYSDBA})
    or die "Connect failed: $DBI::errstr";
```

请在测试 Perl 代码之前，确保被授予了 SYSDBA 特权并且其他要求都被满足。

## 22.6.7 使用操作系统认证连接

在 UNIX 系统上，通常用来分配 SYSDBA 特权的 DBA 用户组名是 dba，而 OPER 用户组和 SYSOPER 特权的用户组名建议设置为 oper。这些 UNIX 用户组名只是建议而已。当使用 OUI 安装时，也可以选择其他的用户组名。

在 Windows 系统上，SYSDBA 特权的 DBA 用户组名一直是 ORA_DBA，并且 OPER 用户组名为 ORA_OPER。sqlnet.ora 中 NTS (NT Security) 必须以认证服务的形式启用，使得操作系统认证可以工作。这可以通过 sqlnet.ora 中如下行完成：

```
SQLNET.AUTHENTICATION_SERVICES = (NTS) # required for connect / as sysdba
```

在 UNIX 系统上，DBA 用户组的成员资格已经足够高而无需密码进行连接。在默认设置 (REMOTE_OS_AUTHENT=FALSE) 下，以 SYSDBA 的身份连接并无需提供密码仅限于使用 bequeath 适配器。该默认行为不应被改变，因为设置 REMOTE_OS_AUTHENT=TRUE 将会带来很大的安全隐患。



就像以前，验证 CONNECT / AS SYSDBA 在 SQL*Plus 中起作用是一种很好的实践，在使用如下 Perl DBI 代码做出相同尝试之前：

```
use DBI;
use DBD::Oracle qw(:ora_session_modes); # imports ORA_SYSDBA and ORA_SYSOPER
my $dbh = DBI->connect("DBI:Oracle:", "/", undef, {ora_session_mode => ORA_SYSDBA})
    or die "Connect failed: $DBI::errstr";
```

请注意，上面的例子中不仅使用了 SYSDBA 特权也使用操作系统认证。后者的特点在于传递 "/" 作为 \$user 和 undef（即无密码）作为 \$auth。

让我们看一下一个非特权用户怎样使用操作系统认证进行连接。无需在脚本中嵌入密码或给命令行传递密码对于执行批处理任务非常有用，更何况通过 UNIX 命令 ps 查询进程列表就可以偷看到密码。假设我们希望允许 UNIX 用户 oracle 无需密码进行连接。用于该用途的数据库用户名取决于初始化参数 OS_AUTHENT_PREFIX 的设置。在下面的示例中，为其设置了默认值 ops\$：

```
SQL> SHOW PARAMETER os_authent_prefix
NAME                TYPE        VALUE
-----
os_authent_prefix    string      ops$
```

接下来，我们为 UNIX 用户名为 oracle 的用户使用 ops\$ 作为前缀来创建一个数据库用户。

```
SQL> CREATE USER ops$oracle IDENTIFIED EXTERNALLY;
SQL> GRANT CONNECT TO ops$oracle;
```

为了测试新的数据库用户的连接，我们必须以 UNIX 用户 oracle 登录到系统。

```
$ id
uid=503(oracle) gid=504(oinstall) groups=504(oinstall),505(dba)
$ sqlplus /
Connected.
```

以下 Perl 程序 os.pl 使用操作系统身份认证，通过执行 SELECT user FROM dual 来检索数据库用户名：

```
#!/usr/bin/env perl
use DBI;
my $dbh=DBI->connect("dbi:Oracle:", "/", undef) or die "Failed to connect.\n";
my $sth=$dbh->prepare("SELECT user FROM dual");
$sth->execute;
my @row=$sth->fetchrow_array;
printf "Connected as user %s.\n", $row[0];
$ chmod +x os.pl
$ ./os.pl
Connected as user OPS$ORACLE.
```

## 22.6.8 连接属性

有 4 个额外的属性会被传递给连接方法。它们是 AutoCommit、ora_module_name、PrintError 以及 RaiseError。接下来的章节对每一个属性都有详细的解释，并且给出与 ORACLE DBMS 协作的最优值建议。你可以传递这些属性中的一个或多个到连接方法，用逗号对它们进行分隔。

### 1. AutoCommit属性

AutoCommit 属性的设置决定了每次执行 INSERT、UPDATE、DELETE 或 MERGE 语句之后是否立即提交，以及之后隐式或显式地执行\$dbh->commit。AutoCommit 的默认设置为 1（已启用）。由于对于每一次改变单独提交会严重降低应用程序的响应时间，显式地设置 AutoCommit 为 0（禁用）则显得非常重要。

### 2. Module Name属性

属性 ora_module_name 可用于设置 V\$SESSION.MODULE，这对于数据库管理员识别一个应用程序很有用。设置 ora_module_name 的结果就如同在一个匿名块中调用 DBMS_APPLICATION_INFO.SET_MODULE 一样，但是其几乎不需要额外的编码工作。默认值在 UNIX 系统上是 perl 而在 Windows 上是 perl.exe。请注意这种方式设置的模式名会被 DBMS_MONITOR（参见相关数据字典视图 DBA_ENABLED_TRACES 和 DBA_ENABLED_AGGREGATIONS）控制的跟踪和客户端信息统计功能忽略。

### 3. PrintError属性

PrintError 属性控制 Perl DBI 在发生错误时的行为。默认值为 1（已启用），意味着错误会被打印在标准错误输出。当设置 PrintError=0，则应用程序负责在适当的时候打印错误消息。由于并非所有的错误都是致命的、甚至不是一个问题的指示，推荐设置为 0（禁用）。

### 4. RaiseError属性

RaiseError 属性控制无论何时发生错误时 DBI 是否抛出一个 Perl 异常。它适用于除 connect 之外所有的 DBI 方法。作为替代方案，可以检查每一个单独的 DBI 调用的返回代码。由于相比于根据返回代码为每一个 DBI 调用 if else 语句编码，在一个捕获异常的 eval 块中嵌入许多 DBI 调用要方便，我们就更加倾向于使用 eval。RaiseError 的默认值是 0（禁用）。当设置 RaiseError=1（已启用）时，无论何时 DBI 调用失败都会抛出异常。

## 22.7 完整 Perl DBI 示例程序

现在是时候把它们放在一起了。下面的 Perl DBI 程序说明了 Perl DBI 编程的许多方面。它使用了 4 个连接属性并在一个循环中插入数据行到表 CUSTOMER 中。customer 标识符 (CUSTOMER.ID) 按序列生成。用于运行示例的数据库对象的 CREATE 语句以 POD 区域的方式被嵌入在 Perl 源代码中。POD 区域会被 Perl 编译器忽略。

用于插入的数据行数被作为参数传递进来。为了确保良好的性能，该程序在退出之前仅提交一次，在进入循环之前只使用绑定变量解析一次 INSERT 语句，并只在循环之内产生执行和绑定调用。另一种性能提升是通过与 INSERT RETURNING 一起使用 INSERT TRIGGER，而不是首先使用 SELECT 语句获取下一个序列号然后在 INSERT 语句将该序列号传递回 DBMS 实例。后一种方法会损害性能，因为它带来客户端与服务器之间不必要的网络往返。网络延迟越高，对响应时间造成的影响就越严重。

顺便说一句，对使用一个序列作为主键的对象插入数据行最快的方法是参考 INSERT 语句中的序列 sequence_name.NEXTVAL。这要比 PL/SQL 触发器更少的 CPU 周期。以下是一个例子：

```

INSERT INTO customer(id, name, phone)
VALUES (customer_id_seq.nextval, :name, :phone)
RETURNING id INTO :id

```

缺点就是对序列的访问需要在每一个应用程序中编码，而如果使用触发器的话就可以集中编码。

该程序的特色是基于一个环境变量设置启用 SQL_TRACE 的能力。这是每一个应用程序都应具有的功能，因为它减少了编译性能诊断数据所要付出的努力。DBI 方法 do 用于执行 ALTER SESSION 语句。该方法同样也适合执行不可重复使用的语句比如 CREATE TABLE。

该程序还演示了怎样通过准备并执行一个匿名块来调用 PL/SQL 包。另一种调用 PL/SQL 例程（函数、包以及过程）的方法就是 SQL 语句 CALL（请参阅 *Oracle Database SQL Reference 10g Release 2* 一书 13-53 页），它同 DBI 一起工作，但并没有在以下例子中使用。我添加大量注释，用于指出在 Perl 程序 insert_perf4.pl 中发生的一切：

```

1  #!/usr/bin/env perl
2
3  =pod
4
5  create table customer(
6      id number(*,0) not null,
7      name varchar2(10),
8      phone varchar2(30)
9  );
10 create sequence customer_id_seq;
11 create or replace trigger ins_customer before insert on customer for each row
12 begin
13     SELECT customer_id_seq.nextval INTO :new.id FROM dual;
14 end;
15 /
16 variable id number
17 INSERT INTO customer(name, phone) VALUES ('&name', '&phone') RETURNING id INTO
: id;
18 print id
19
20 CLEANUP
21 =====
22 drop table customer;
23 drop sequence customer_id_seq;
24
25 =cut
26
27 if ( ! defined $ARGV[0] ) {
28     printf "Usage: $0 iterations\n";
29     exit;
30 }
31
32 # declare and initialize variables
33 my ($id, $name, $phone, $sth)=(undef, "Ray", "089/4711", undef);

```

```

34
35 use File::Basename;
36 use DBI; # import DBI module
37 print "DBI Version: $DBI::VERSION\n"; # DBI version is available after use DBI
38 use strict; # variables must be declared with my before use
39 my $dbh = DBI->connect("DBI:Oracle:TEN_IPC.WORLD", "ndebes", "secret",
40 # set recommended values for attributes
41 {ora_module_name => basename($0),RaiseError=>1, PrintError=>0,AutoCommit => 0
42 })
43 or die "Connect failed: $DBI::errstr";
44 # DBD::Oracle version is available after connect
45 print "DBD::Oracle Version: $DBD::Oracle::VERSION\n";
46 # start eval block for catching exceptions thrown by statements inside the
47 block
48 eval {
49 # tracing facility: if environment variable SQL_TRACE_LEVEL is set,
50 # enable SQL trace at that level
51 my $trc_ident=basename($0); # remove path component from $0, if present
52 if ( defined($ENV{SQL_TRACE_LEVEL}) ) {
53     $dbh->do("alter session set tracefile_identifier='$trc_ident'");
54     $dbh->do("alter session set events
55         '10046 trace name context forever, level $ENV{SQL_TRACE_LEVEL}'");
56 }
57 # parse an anonymous PL/SQL block for retrieving the ORACLE DBMS version
58 # and compatibility as well as the database and instance names
59 # V$ views are not used, since they may be accessed by privileged users only
60 # quoting with q{<SQL or PL/SQL statements>} is used, since
61 # it avoids trouble with quotes ("', ')
62 $sth = $dbh->prepare(q{
63     begin
64         dbms_utility.db_version(:version, :compatibility);
65         :result:=dbms_utility.get_parameter_value('db_name',:intval, :db_name
66 );
67         :result:=dbms_utility.get_parameter_value('instance_name', :intval,
68             :instance_name);
69     end;
70 });
71 my ($version, $compatibility, $db_name, $instance_name, $result, $intval);
72 $sth->bind_param_inout(":version", \ $version, 64);
73 $sth->bind_param_inout(":compatibility", \ $compatibility, 64);
74 $sth->bind_param_inout(":db_name", \ $db_name, 9);
75 $sth->bind_param_inout(":instance_name", \ $instance_name, 16);
76 $sth->bind_param_inout(":intval", \ $intval, 2);
77 $sth->bind_param_inout(":result", \ $result, 1);
78 $sth->execute;
79
80 $sth = $dbh->prepare(q{SELECT userenv('sid'),
81     to_char(sysdate, 'Day, dd. Month yyyy hh24:mi:ss "(week" IW)"') FROM dual});

```

```

);
81  $sth->execute;
82  my ($sid, $date_time);
83  # pass reference to variables which correspond to columns in
84  # SELECT from left to right
85  $sth->bind_columns(\ $sid, \ $date_time);
86  my @row = $sth->fetchrow_array;
87
88  printf "Connected to ORACLE instance %s, release %s (compatible=%s)",
89  $instance_name, $version, $compatibility;
90  printf "; Database %s\n", $db_name;
91  # due to bind_columns, may use meaningful variable names instead of $row[0],
etc.
92  printf "Session %d on %s\n", $sid, $date_time;
93
94  $sth = $dbh->prepare("INSERT INTO customer(name, phone) VALUES (:name, :phone
)
95      RETURNING id INTO :id", { ora_check_sql => 0 });
96  # loop, number of iterations is in command line argument
97  for (my $i=0; $i < $ARGV[0]; $i++) {
98      # bind_param_inout is for receiving values from the DBMS
99      $sth->bind_param_inout(":id", \ $id, 38);
100     # bind_param is for sending bind variable values to the DBMS
101     # assign value to bind variable (placeholder :name)
102     $sth->bind_param(":name", $name);
103     # assign value to bind variable "phone"
104     $sth->bind_param(":phone", $phone);
105     # execute the INSERT statement
106     $sth->execute();
107     printf "New customer with id %d inserted.\n", $id;
108 }
109 };
110 # check for exceptions
111 if ($?) {
112     printf STDERR "ROLLBACK due to Oracle error %d: %s\n", $dbh->err, $@;
113     # ROLLBACK any previous INSERTs
114     $dbh->rollback;
115     exit;
116 } else {
117     # commit once at end
118     $dbh->commit;
119 }
120 $sth->finish; # close statement handle
121 $dbh->disconnect; # disconnect from ORACLE instance

```

第 35 行导入了包 `File::Basename`，其中包含函数 `basename`。Perl 命令 `basename` 与 UNIX 系统中相同名称的命令有一样的效用。它通过从字符串剥离一个或多个目录，返回来自一个路径名的文件组成部分。命令 `basename` 用于确保无论是模块名称还是 `TRACEFILE_IDENTIFIER` 都不包含非法或不想要的字符，比如可能出现在 Perl 变量 `$0` 中的目录分隔符 (`/`)。

第 39 行中的连接语句，被设置为推荐值 `RaiseError(1)`、`PrintError(0)` 以及 `AutoCommit(0)`。第 47~109 行用于捕捉来自 DBI 方法请求异常的 `eval` 块。对环境变量 `SQL_TRACE_LEVEL` 的检查位于第 51 行。该值的范围同事件 10046 相同（1、4、8、12，请参阅第 24 章）。如果设置了该环境变量，第 53 行就使用 `ALTER SESSION SET EVENTS` 启动了 SQL 跟踪。在第 52 行，Perl 程序的名称被用作 `TRACEFILE_IDENTIFIER`，以方便定位设置为 `USER_DUMP_DEST` 的目录里的跟踪文件。

第 62~77 行演示了怎样嵌入 PL/SQL 到一个匿名块中对其进行调用。可公开访问的包 `DBMS_UTILITY` 在单个匿名块中被调用了三次，用以检索实例和数据库上的信息。

第 79~86 行以及第 92 行举例说明了使用 DBI 编码如何便于获取数据。使用 `bind_columns` 方法，带有有意义名称的变量被用于访问使用 Perl DBI 方法 `fetchrow_array` 检索的列值。如果没有使用 `bind_columns` 方法，就必须使用数组语法，比如 `$row[column_index]`，其中 `column_index` 从 0 开始并从左到右指派到 `SELECT` 列表中的列。

包含绑定变量并因此可以重用的 `INSERT` 语句，在进入循环之前仅通过在第 94 行调用一次 `prepare` 进行解析。将 `prepare` 放置在循环之内将会是一个代价很高的错误——将会发生由于解析调用而导致的额外网络往返以及多余的软解析导致的过量 CPU 消耗。

在 `for` 循环内部，从第 97 行延伸到第 108 行，`bind_param_inout` 方法用于告诉 DBI 哪一个变量被用于接收 SQL 语句 `INSERT RETURNING id INTO :id` 返回给客户端的序列号。绑定变量 `name` 和 `phone` 用来发送值到 DBMS。这是使用 DBI 方法 `bind_param` 完成的。

`eval` 块后跟着一个 `if` 语句，它用于检查特殊的 Perl 变量 `$@` 中的异常。如果发生了一个异常，`$@` 就会包含错误消息。否则 `$@` 就是一个空字符串，Perl 的布尔表达式就认为其为 `FALSE`，这样第 112 行的 `if` 分支就不会进入。如果发生异常，通过在第 114 行产生回滚，所有已经被插入的值就会被丢弃。

如果一切正常，就会在第 118 行调用一次 `commit`，该语句句柄 `$sth` 在第 120 行被释放，并且该客户端在第 121 行从 DBMS 断开连接。下面是运行 Perl 程序 `insert_perf4.pl` 的副本：

```
$ ./insert_perf4.pl 3
DBI Version: 1.41
DBD::Oracle Version: 1.15
Connected to ORACLE instance ten, release 10.2.0.1.0 (compatible=10.2.0.1.0);
Database TEN
Session 145 on Thursday , 19. July      2007 21:57:54 (week 29)
New customer with id 1 inserted.
New customer with id 2 inserted.
New customer with id 3 inserted.
```

## 22.8 异常处理

在结束本章之前，我想实际使用 `eval` 演示一下异常处理。设置表 `CUSTOMER` 所在的表空间状态为只读就会导致第 106 行的 `execute` 方法调用失败。程序将会继续在第 111 行的 `if` 语句执行。下面是一个循序渐进的测试：

```

SQL> CONNECT system
Enter password:
Connected.
SQL> ALTER TABLESPACE users READ ONLY;
Tablespace altered.
SQL> EXIT
Disconnected from Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Produc
tion
$ ./insert_perf4.pl 1
DBI Version: 1.48
DBD::Oracle Version: 1.16
Connected to ORACLE instance ten, release 10.2.0.1.0 (compatible=10.2.0.1.0); Databa
se TEN
Session 145 on Thursday , 19. July      2007 23:52:32 (week 29)
ROLLBACK due to Oracle error 372: DBD::Oracle::st execute failed: ORA-00372: file 4
cannot be modified at this time
ORA-01110: data file 4: 'F:\ORADATA\TEN\USERS01.DBF' (DBD ERROR: error possibly near
<*> indicator at char 12 in 'INSERT INTO <*>customer(name, phone) VALUES (:name,
:phone)
RETURNING id INTO :id') [for Statement "INSERT INTO customer(name, phone)
VALUES (:name, :phone)
RETURNING id INTO :id" with ParamValues: :name='Ray', :phone='089/4711',
:id=undef] at ./insert_perf4.pl line 106.

```

从前面的输出可以很明显地看出, DBI 提供了比 ORACLE DBMS 错误消息 ORA-00372: file 4 cannot be modified at this time 更多的信息。它检索错误信息栈的第二条错误消息 (ORA-01110: data file 4: 'F:\ORADATA\TEN\USERS01.DBF'), 告诉我们哪一条 SQL 语句导致了错误, 在 Perl 源代码文件中的行号, 使用了哪些变量以及它们的值是什么。

## 22.9 源代码库

表 22-5 列出了本章的源代码文件以及它们的功能。

表22-5 Perl DBI源代码库

文 件 名	功 能
ORADBB.pm	运行dbb.pl需要Perl模块ORADBB.pm。它包含一些Perl子程序, 它们可以被任何Perl DBI应用程序重用 (例如, 使用SYSDBA或SYSOPER特权连接)
args.pl	打印命令行参数到标准输出的Perl程序
dbb.pl	可以执行任意SQL和PL/SQL语句的命令行工具。检索结果的列宽可以自动调整
insert_perf4.pl	演示异常处理、PL/SQL执行、绑定变量、ROLLBACK以及COMMIT的Perl程序
os.pl	使用操作系统认证连接的Perl程序
perl-dbi-test.pl	用于测试到一个ORACLE DBMS实例连通性的Perl程序

*the Oracle Database Performance Tuning Guide 10g Release 2* 和 *the Oracle Call Interface Programmer's Guide 10g Release 2* 这两本书描述了端到端应用跟踪特性。*Oracle Database JDBC Developer's Guide and Reference 10g Release 2* 一书描述了 JDBC 支持的端到端度量。*Oracle Database Concepts 10g Release 2* 描述了端到端监控特性。这三个术语都涉及了一个包括跟踪、监控、插桩的特性集合。这个特性集合通过三个应用编程接口 (OCI、PL/SQL、java/JDBC)、字典视图、V\$动态性能视图以及 SQL 跟踪来实现。插桩的实现要借助 AWR、ASH、Statspack、Enterprise Manager、数据库资源管理器以及 TRCSESS 实用程序等多个工具。关于这个特性集合尚未在文档中说明的内容非常多, 因此我不可能把它们全部列出来^①。例如, 下面这点就没有在文档中说明: 一个共享的服务进程在为多个会话服务时, 不会向 SQL 跟踪文件重复发送监测信息。当一个包含多个会话的共享服务进程所产生的 SQL 跟踪文件被 TRCSESS 处理时, 可能会发生错误。

本章将讲述一个完整的监测案例, 该案例可以清楚地表现前述组件的优点, 并给出一些尚未在文档中说明的方面, 比如 JDBC 端到端度量所产生的 SQL 跟踪文件的条目。本章还将讲述一个变通方法, 用来避免在共享服务环境下使用 TRCSESS 产生错误结果。本章还给出了一个对应用监测与资源管理进行集成的实例, 该实例提供了一种尚未在文档中说明的语法来根据服务名、模块及行为等分配资源。最后, 将讨论 TRCSESS 与 Oracle9i 的后向兼容问题。

## 23.1 插桩简介

根据维基百科, “插桩 (instrumentation) 包括: 监控和度量产品的性能、诊断错误、记录跟踪信息……” 在编程领域, 插桩是应用程序的一种能力, 包括以下方面。

- 代码跟踪: 接收运行时应用程序有用的执行信息。
- 性能度量: 允许你跟踪应用程序性能的功能组件。
- 事件日志: 允许你探测和跟踪应用程序执行过程中主要事件的功能组件^②。

简而言之, 插桩可以使软件度量它自己的表现。ORACLE DBMS 的插桩性 (instrument) 很好。它维持了数以百计的计数器和计时装置, 它们反映了工作负载和 SQL 语句的执行性能、内

① 源代码库中的 Java 源文件 ApplicationInstrumentation.java 列出了尚未在文档中说明的 10 个方面。

② [http://en.wikipedia.org/wiki/Instrumentation_%28computer_programming%29](http://en.wikipedia.org/wiki/Instrumentation_%28computer_programming%29)。



存使用情况、文件、网络的可获得性等方面的情况。在实例级别、会话级别、SQL 或 PL/SQL 语句级别都有很好的度量指标。在由 Oracle 公司的 Anjo Kolk、Shari Yamaguchi 和 Jim Viscusi 于 1999 年发表的论文 Yet Another Performance Profiling Method (or YAPP-Method) 中, 提出了下列公式:

$$\text{响应时间} = \text{服务时间} + \text{等待时间}$$

尽管这篇论文缺乏一个基于快照 (snapshot-based) 的方法来度量性能^①, 而且错误地指出等待事件 SQL*NET message from client 应该在会话级别被忽略 (该错误限制了性能诊断的解释能力, 并且现在的很多书中仍可以见到该错误^②), 但是它仍然是树立新的调优范式的里程碑。简而言之, 服务时间是所消耗的 CPU 时间, 而等待时间则是花费在等待某个等待事件 (有数以百计的等待事件) 的时间, 这样的等待事件可能是硬盘启动、同步或网络延迟 (见附录 Oracle 数据库参考中的 Oracle 等待事件以及动态性能视图 V\$EVENT_NAME)。数据库服务器的插桩很好地提供了上述度量。插桩本身也会对性能表现有一定的影响, 这被称为度量入侵 (measurement intrusion)。一般地, 一个扩展的 SQL 跟踪文件是按微秒记录数据库会话的运行时间。在实际操作中, Oracle 数据库管理系统的一部分代码路径被插桩得不如别的部分多, 这样导致 SQL 跟踪文件不能准确计量一个完整数据库会话的运行时间。

为终端用户所感知的响应时间受一些网络延迟、在中间层 (如应用服务器) 运行的进程等其他因素的影响。但是, 很明显, 数据库服务器有它自己的响应时间, 这个时间不同于应用服务器的响应时间, 也不同于终端用户感受到的响应时间。例如, Oracle 10g 中的插桩为在扩展 SQL 跟踪文件中的等待事件 (WAIT 条目) 打上了时间戳。而过去, 只是在数据库调用解析、执行和读取等操作时才加上时间戳。从数据库服务器的角度看, 一个 SQL 语句的响应时间是指从一个语句到达数据库服务器开始, 到响应发送给客户端的时间间隔。前一个时间点由等待事件 SQL*Net message to client 所标记, 后一个时间点由等待事件 SQL*Net message to client 所标记。由于网络延迟的缘故, 客户端并非在等待事件 SQL*Net message to client 完成的时候就可以收到响应。因此, 从客户端度量的响应时间比从数据库服务端度量的时间要长。

并非数据库服务器中所有的代码路径都完整地插桩了, 而且, 度量入侵会使度量不准确问题变得更严重。不过, 通过跟踪文件中的时间戳计算所得的运行时间 (参数 tim, 见第 24 章), 与通过累加 CPU 消耗与等待时间所得的运行时间, 两者之间的差距不会超过 10%。这表明, 插桩的精度是很高的, 度量入侵的影响是很小的, 因此, 可以使用监测所得的数据来实现可靠的性能诊断。

我使用术语应用程序插桩来将 ORACLE DBMS 的插桩和应用程序或客户端内部的插桩区分开来。一般地, 数据库管理系统的插桩功能是核心功能之一, 但是当应用程序也被插桩时, 它的内涵就上升到一个更大的范畴。这就是我要在本章讲述的主题。数据库管理系统有应用编程接口,

① Statspack 和 AWR 都实施了一个基于快照的方法来获取性能数据。但是, 当实例或会话启动后, 所得的数据就不是基于快照的了。

② 请参考第 27 章的关于 SQL*Net message from client 的信息, 并理解如何从这一等待事件中衍生出思考时间。

可以设置模块、行为和客户端标识符。这里，一个模块标识一个较长的代码路径，可能对应一批作业、一个商业任务或者更大应用下的一个功能组件。比如，人力资源和基础设施可能是一个企业资源计划系统中的模块。一个模块包含一个或多个行为，也就是说，行为是比模块更细粒度的。

一个客户端标识用于标识一个特定的终端客户。在一个连接池环境中，专用的服务器进程批量产生，然后被重复用于操作 DML，为不同的终端用户提供服务。一个 DBA 不知道哪个特定的服务进程服务于哪个终端用户，除非该应用程序被插桩，并且设定了客户端标识符。因为连接池中的所有会话都被一个相同的应用服务器所启动，且连接到单个数据库用户上，因此，视图 V\$SESSION 中的列 USERNAME、OSUSER 和 MACHINE 都是无用的。如果有应用被插桩，那么 DBA 就可以识别哪个数据库会话服务于哪个终端用户。DBA 还可以看到哪个模块或行为被执行了。当然，你可能希望加入更多的插桩代码，而不只是简单地设定模块、行为和客户端标识符。比如说，你可以加入这样的代码：它们与数据库管理系统中的跟踪或统计信息收集无关，而关注应用程序的功能或与度量数据库管理系统无关的响应时间。特别是当应用包含没有与 DBMS 实例交互的模块时，你可能需要加入一些统计花在这些模块中的时间的代码。毕竟，DBMS 只能统计花在数据库调用的时间，以及等待接收其他数据库调用的时间，即等待 SQL*Net message from client 事件的时间。

除了模块、行为和客户端标识外，应用程序中使用的实例服务名（见本书引言中的“实例服务名与网络服务名的对比”）也可用来帮助监控和追踪。不同的应用程序（或连接池）在运行一个实例时，应该使用不同的服务名。这就为通过服务名来进行跟踪和监控提供了可能性。

## 23.2 案例研究

我选择了 Java、JDBC 和新的 JDBC 端到端度量（由 Oracle10g 引入）作为本案例研究中的示例代码。在 Oracle11g 中，JDBC 端到端度量接口已经从文档中移除，换成了动态监控服务（Dynamic Monitoring Service）。尽管如此，该接口仍然在 JDBC 驱动版本 11.1.0.6.0 中，并且与在 Oracle10g 中完全一样。PL/SQL 中的插桩工具已经运行很多年了，且工作得很好。但 JDBC 端到端度量则不然，由 Oracle 在 *Oracle Database JDBC Developer's Guide and Reference 10g Release 2* 中提供的关于 JDBC 端到端度量的代码示例并不完整，语法上也不正确。本案例研究了插桩对于 V\$视图以及 SQL 跟踪文件的影响，并讲述如何使用 DBMS_MONITOR、TRCSESS 以及数据库资源管理器来实现插桩。

### JDBC 端到端度量示例代码

JDBC 端到端度量是一种用于插桩的轻量级方法。与 PL/SQL 包中 DBMS_APPLICATION_INFO 及 DBMS_SESSION 直接设置模块、行为及客户端标识不同，在 JDBC 端到端度量中，变化的值只是存储在一个 Java 字符串数组中，通过接下来的数据库调用被传递到 DBMS。这样，它与 PL/SQL 不同，不会导致额外的网络往返。Thin JDBC 驱动也跟 JDBC OCI 驱动一样，具备该功能。API 接口只在 Oracle JDBC 驱动 10.1 以后的版本中出现。第三方 JDBC 驱动和 Oracle 驱动从版本 9.2

开始具备此功能，之前的版本不具备。它的编程十分简单。你可以声明一个字符串数组，用一些常量来设定字符串的长度和数组元素，这些值表示模块、行为和客户端标识。接下来的代码节选来自于 Java 程序 `ApplicationInstrumentation.java`，它们都可以从源码库中找到：

```

1 DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
2 java.util.Properties prop = new java.util.Properties();
3 // properties are evaluated once by JDBC Thin when the session is created. Not suitable
  for setting the program or other information after getConnection has been
  called
4 prop.put("user", username);
5 prop.put("password", pwd);
6 prop.put("v$session.program", getClass().getName()); // undocumented property V$session.program
  works with JDBC Thin only; if specified, then set as program and module, as expected end-to-end
  metrics overwrites the module; program is not overwritten
7 Connection conn = DriverManager.getConnection(url, prop);
8 conn.setAutoCommit(false);
9 // Create Oracle DatabaseMetaData object
10 DatabaseMetaData metadata = conn.getMetaData();
11 // gets driver info:
12 System.out.println("JDBC driver version: " + metadata.getDriverVersion() + "\n");
13 System.out.println("\nPlease query V$SESSION and hit return to continue when done.\n");
14 System.in.read(buffer, 0, 80); // Pause until user hits enter
15 // end-to-end metrics interface
16 String app_instrumentation[] = new String[OracleConnection.END_TO_END_STATE_INDEX_MAX];
17 app_instrumentation[OracleConnection.END_TO_END_CLIENTID_INDEX]="Ray.Deever";
18 app_instrumentation[OracleConnection.END_TO_END_MODULE_INDEX]="mod";
19 app_instrumentation[OracleConnection.END_TO_END_ACTION_INDEX]="act";
20 ((OracleConnection)conn).setEndToEndMetrics(app_instrumentation,(short)0);
21 Statement stmt = conn.createStatement();
22 ResultSet rset = stmt.executeQuery("SELECT userenv('sid'), to_char(sysdate, 'Month dd. yyyy hh24:mi') FROM dual");
23 while (rset.next())
24     System.out.println("This is session " + rset.getString(1) + " on " + rset.getString(2));
25 rset.close();
26 System.out.println("\nPlease query V$SESSION and hit return to continue when done.\n");
27 System.in.read(buffer, 0, 80); // Pause until user hits enter
28 // with connection pooling, execute this code before returning session to connection pool
29 app_instrumentation[OracleConnection.END_TO_END_CLIENTID_INDEX]="";
30 app_instrumentation[OracleConnection.END_TO_END_MODULE_INDEX]="";
31 app_instrumentation[OracleConnection.END_TO_END_ACTION_INDEX]="";
32 ((OracleConnection)conn).setEndToEndMetrics(app_instrumentation,(short)0);

```

连接的建立是这样实现的：先创建一个 `java.util.Properties` 类的实例，然后设定用户名和密码，然后通过 `DriverManager.getConnection` 方法将该实例传递给驱动管理器。该程序可以使

用一个未在文档中说明的属性 `v$session.program` (见第 6 行) 来为 JDBC Thin 连接设定 `V$SESSION` 视图中的程序和模块名。在这种连接中, 默认的程序和模块名通常是 JDBCThin 客户端, 当然这并不是绝对的。JDBC OCI 忽略这个属性。

如果你打算深入地插桩 JDBC Thin 应用, 我推荐你至少设定上述属性的优先性比向 DBMS 建立一个连接的优先性要高。这样, DBA 将能够识别哪个数据库会话是由你的 Java 程序运行的, 但他不能通过 DBMS_MONITOR 来使用这个属性, 因为它只能适用于文档化编程接口 PL/SQL、OCI 以及 Java。模块名将不仅出现在 `V$SESSION.MODULE` 中, 也将出现在 Statspack 以及 AWR 中。当然, 后者适用于任何设定 `V$SESSION.MODULE` 的程序, 不管它是通过 DBMS_APPLICATION_INFO.SET_MODULE、OCI 还是 JDBC 方法实现。该模块还可以通过 JDBC OCI 来设定, 但是那需要更多的代码。适用于 JDBC OCI 的默认程序和模块在 UNIX 平台上是 `java@client_host_name`, 在 Windows 平台上是 `java.exe`, 这里的 `client_host_name` 是 JDBC OCI 程序所连接到的系统名。

在第 16 行, 变量 `app_instrumentation` 被声明为一个字符串数组。常量 `OracleConnection.END_TO_END_STATE_INDEX_MAX` 被用于设定数组的长度。客户端标识、模块、行为在第 17~19 行中设定。第 20 行对 `((OracleConnection)conn).setEndToEndMetrics` 的调用使得新的配置可以为端到端度量 API 所调用。这里, 通过类型限定强制要求使用 Oracle 公司的子类。在第 20~25 行, 一个 SELECT 语句被执行。这个语句导致了解析、执行以及读取等数据库调用。新的配置将通过这些由 SELECT 语句引起的第一个数据库调用传送。这样, 当程序等待用户在第 27 行处敲 Return (回车键) 时, 关于模块、行为和客户端标识的新配置就可以出现在 `V$SESSION` 视图中。在第 29~32 行, 模块、行为和客户端标识被设置为空字符串。这里这么做是为了展现 DBMS_MONITOR 与 TRCSSESS 的集成。

源代码库中还包括了一个使用 PL/SQL 实施插桩的 Java 程序, 该程序和 Oracle9i 及 Oracle10g 都兼容。使用一个 Oracle10g JDBC 驱动来连接一个 Oracle9i 的实例也能够支持。然而, 当对 Oracle9i 执行一个端到端的度量代码时, 它是不起任何作用的。因此, 至少要设置模块和行为, 在 Oracle9i 上运行时, Java 应用程序必须使用 PL/SQL。Java 类 `OracleConnection` 中包含一个方法 `public void setClientIdentifier(java.lang.String clientId)` 可以设定客户端标识。源代码库中的 Java 程序 `JdbcInstrumentationOracle9i.java` 只可以用 PL/SQL 插桩, 可以在 Oracle9i 以及以后版本上运行。

## 23.3 程序编译

因为在 Oracle10g 的 `ORACLE_HOME` 中有 JDK (Java Developer's Kit, Java 开发者工具箱), 因此无需安装额外的软件来编译 Java 程序。我们只需要设置环境变量 `CLASSPATH`, 并将 Java 编译器 `javac` 包含在命令行搜索路径变量 `PATH` 中即可。在 UNIX 中, 冒号 (:) 被用来分隔 `CLASSPATH` 中的独立条目, 而在 Windows 中使用的是分号 (;)。接下来是一个 Windows 中的例子:

```
C:> set ORACLE_HOME=C:\Oracle\product\db10.2
C:> set CLASSPATH=%ORACLE_HOME%\jdbc\lib\ojdbc14.jar;.
C:> set PATH=%ORACLE_HOME%\bin;%ORACLE_HOME%\jdk\bin;%SystemRoot%\system32
C:> javac ApplicationInstrumentation.java
```

经过上述步骤后，被编译好的 Java 程序就存储在文件 `ApplicationInstrumentation.class` 中。这个编译好的文件其实也已经包含在源代码库中了，所以你无需编译源文件。

## 23.4 插桩的运行

现在我们可以运行程序了。因为我们在前一节的 `CLASSPATH` 设定中使用了“.”，因此要运行 `ApplicationInstrumentation.class`，就必须先把它放到当前目录下。该程序需要下列 3 个参数：

- 数据库用户名
- 用户密码
- JDBC URL (JDBC Thin 类型或者 JDBC OCI 类型)

JDBC OCI 类型的 URL 以 `jdbc:oracle:oci:@net_service_name` 的形式出现，这里 `net_service_name` 是一个在 `tnsnames.ora` 中或通过目录服务定义的网络服务名。用于连接到一个特定实例服务的 JDBC Thin 类型的 URL 以 `jdbc:oracle:thin:@//host_name:port/instance_service_name` 的形式出现，这里的 `host_name` 是 DBMS 实例所在系统的地址，`port` 是用于监听的端口号，`instance_service_name` 是在 `lsnrctl services` 中列出的服务名。旧的 JDBC Thin URL 语法是 `jdbc:oracle:thin:@host_name:port:ORACLE_SID`，但是不再用了，因为它要求的服务名是默认的 `V$SESSION.SERVICE_NAME` 中的 `SYS$USERS`。这使得无法为各种不同应用分配不同的实例服务名，也无法满足 RAC 环境中的集群服务的要求。

### 配置 SQL 跟踪、统计信息收集和资源管理器

为了展现端到端跟踪的插桩，客户端统计信息收集和资源管理器之间的集成，我们需要做一些准备工作。我们将调用 `DBMS_MONITOR`，以针对实例服务名、模块和行为的特定组合进行 SQL 跟踪或统计信息收集。调用打包后的程序 `DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE` 时，只有实例服务名是必须指定的。如果没有指定特定模块，那么配置将影响所有模块。同样，如果只指定了服务名和模块，配置将影响所有行为。针对特定客户端标识进行跟踪和收集统计信息也是可以的。下面的例子中，在不同级别先打开、然后关闭跟踪和统计收集功能。

```
SQL> EXEC dbms_monitor.serv_mod_act_trace_enable('TEN.oradbpro.com', 'mod', 'act')
SQL> EXEC dbms_monitor.client_id_trace_enable('Ray.Deever')
SQL> EXEC dbms_monitor.serv_mod_act_stat_enable('TEN.oradbpro.com', 'mod', 'act')
SQL> EXEC dbms_monitor.client_id_stat_enable('Ray.Deever')
SQL> EXEC dbms_monitor.client_id_stat_disable('Ray.Deever')
SQL> EXEC dbms_monitor.serv_mod_act_stat_disable('TEN.oradbpro.com', 'mod', 'act')
SQL> EXEC dbms_monitor.client_id_trace_disable('Ray.Deever')
SQL> EXEC dbms_monitor.serv_mod_act_trace_disable('TEN.oradbpro.com', 'mod', 'act')
```

`DBMS_MONITOR` 的参数是区分大小写的。服务名的拼写必须与在 `DBA_SERVICES.NETWORK_NAME` 相应列中所存储的值匹配。这些设置在实例重启后仍然有效。从字典视图 `DBA_ENABLED_TRACES` 和 `DBA_ENABLED_AGGREGATIONS` 可以查看当前配置。如果我们想跟踪实例服务 `TEN.oradbpro.com` 的所有会话，并对该服务下的模块 `mod` 进行统计信息收集，我们可以通过下列代码实现：

```

SQL> SELECT network_name FROM dba_services WHERE network_name LIKE 'TEN.%';
NETWORK_NAME
-----
TEN.oradbpro.com
SQL> EXEC dbms_monitor.serv_mod_act_trace_enable(service_name=>'TEN.oradbpro.com')
SQL> SELECT trace_type, primary_id, waits, binds FROM dba_enabled_traces;
TRACE_TYPE PRIMARY_ID      WAITS BINDS
-----
SERVICE      TEN.oradbpro.com TRUE  FALSE
SQL> EXEC dbms_monitor.serv_mod_act_stat_enable('TEN.oradbpro.com', 'mod')
SQL> SELECT aggregation_type, primary_id, qualifier_id1, qualifier_id2
FROM dba_enabled_aggregations;
AGGREGATION_TYPE PRIMARY_ID      QUALIFIER_ID1 QUALIFIER_ID2
-----
SERVICE_MODULE      TEN.oradbpro.com mod

```

针对特定的服务名、模块、行为组合的统计信息可以从动态性能视图 V\$SERV_MOD_ACT_STATS 中检索到；而针对客户端标识的统计信息则需从 V\$CLIENT_STATS 中检索。

在插桩设置的基础上为消费者群 (consumer group) 分配数据库会话是 Oracle10g 数据库资源管理的一项新功能。其中，分配可以根据服务名、模块名、行为的组合或根据模块和行为的组合来进行，这两种组合中都只有第一个组成部分必须指定具体名称。通过检索视图 DBA_RSRC_GROUP_MAPPINGS 可以查看当前生效的配置。新功能可以用来优化模块和服务。在资源管理器中有一项默认计划称为 SYSTEM_PLAN。该计划包括消费者群：SYS_GROUP、OTHER_GROUPS 及 LOW_GROUP（见动态性能视图 V\$RSRC_CONSUMER_GROUP）。正如名字所显示的，LOW_GROUP 中包括的是优先级最低的消费者群。在 CPU 资源紧缺时，SYS_GROUP、OTHER_GROUPS 将被给予 CPU 时间片，而 LOW_GROUP 中的会话将放弃 CPU。因为资源管理器默认是关闭的，必须开启它才能使用这些新功能。这可以在运行时使用 ALTER SYSTEM 命令来实现。

```

SQL> ALTER SYSTEM SET resource_manager_plan=system_plan;
System altered.

```

在对 DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING 调用中将实例服务名、模块和行为的组合设定为一个单一字符串的语法没有在文档中说明。也没有陈述大小写敏感问题。实际的测试显示：圆点 (.) 被用于分隔服务名和模块，以及分隔模块与行为。与 DBMS_MONITOR 不同，这里的名称是大小写无关的。下面的代码将构建一个资源管理器，将一个对应的实例服务名为 TEN.oradbpro.com，模块名和行为名分别是 mod 和 act 的会话放入消费者群 LOW_GROUP（文件 rsrc_mgr.sql）。

```

begin
  dbms_resource_manager.create_pending_area();
  dbms_resource_manager.set_consumer_group_mapping(
    attribute=>dbms_resource_manager.service_module_action,
    value=>'TEN.oradbpro.com.mod.act',
    consumer_group=>'LOW_GROUP'
  );

```

```

        dbms_resource_manager.submit_pending_area();
        dbms_resource_manager.clear_pending_area();
end;
/
SQL> SELECT * FROM dba_rsrc_group_mappings;
ATTRIBUTE                VALUE                CONSUMER_GROUP STATUS
-----
SERVICE_MODULE_ACTION  TEN.ORADBP...COM.MOD.ACT  LOW_GROUP
ORACLE_USER              SYS                      SYS_GROUP
ORACLE_USER              SYSTEM                   SYS_GROUP

```

这个设置会马上生效，也就是说，它可以马上应用到已经连接到数据库的会话上。这样，准备工作就完成了。如果我们现在运行 Java 程序，我们希望能实现下面 3 件事情。

(1) 创建一个 SQL 文件。

(2) 针对特定的服务名、模块、行为的组合进行统计信息的收集。

(3) 一旦设置模块名为 mod 及行为名为 act，这个会话就会被归到消费者群 LOW_GROUP。

现在让我们验证软件是否按期望运行。通过启动一个 Java 虚拟机，我们就启动了前述的 Java 程序，同时需要传递前面编译的 java 类和其他参数作为程序的参数。

```

C:> java ApplicationInstrumentation ndebes secret jdbc:oracle:thin:@//localhost:1521
/ TEN.oradbpro.com
JDBC driver version: 10.2.0.3.0
Please query V$SESSION and hit return to continue when done.

```

这时，程序已经启动了，但是还没有向端到端度量接口发起任何调用。现在，开启另一个窗口，并在 SQL*Plus 平台上运行如下查询：

```

SQL> SELECT program, module, action, resource_consumer_group AS consumer_group
FROM v$session
WHERE service_name='TEN.oradbpro.com';
PROGRAM                MODULE                ACTION CONSUMER_GROUP
-----
ApplicationInstrumentation ApplicationInstrumentation  OTHER_GROUPS

```

运行结果表明，用于设定程序和模块的尚未在文档中说明的方法已经起作用了。这个会话目前被归类为 OTHER_GROUPS。接下来，在 Java 程序运行的窗口上按回车。这将使得程序进入到设定客户端标识、模块、行为的界面。它还将执行一个查询来检索会话标识和当前的日期：

```

This is session 40 on October 30, 2007 22:37
Please query V$SESSION and hit return to continue when done.

```

程序又暂停了。这时你可以在视图 V\$SESSION 和 V\$SERV_MOD_ACT_STATS 中查看调用端到端度量 API 的效果（时间按微秒来计量）。

```

SQL> SELECT program, module, action, client_identifier,
resource_consumer_group AS consumer_group
FROM v$session
WHERE service_name='TEN.oradbpro.com';
PROGRAM                MODULE ACTION CLIENT_IDENTIFIER CONSUMER_GROUP
-----

```

```

ApplicationInstrumentation mod    act    Ray.Deever    LOW_GROUP
SQL> SELECT stat_name, value
FROM v$serv_mod_act_stats
WHERE service_name='TEN.oradbpro.com' AND module='mod' AND value > 0;
STAT_NAME                VALUE
-----
user calls                2
DB time                   5555
DB CPU                    5555
parse count (total)       1
parse time elapsed        63
execute count             2
sql execute elapsed time  303
opened cursors cumulative 1

```

在保留程序名字的同时，模块、行为和客户端标识被设定。正如所期望的，会话被归入到 **LOW_GROUP**。有关服务和模块的统计信息被收集，并且可以通过视图 **V\$SERV_MOD_ACT_STATS** 来查看。现在按回车，Java 程序将继续运行。

```

application instrumentation settings removed
Please query V$SESSION and hit return to continue when done.

```

这时，程序已经将模块、行为和客户端标识设置为空字符串。这时程序会暂停，我们可以观察程序的运行效果。会话将还原到过去所属的消费者群。

```

SQL> SELECT program, module, action, client_identifier,
resource_consumer_group AS consumer_group
FROM v$session
WHERE service_name='TEN.oradbpro.com';
PROGRAM                MODULE ACTION CLIENT_IDENTIFIER CONSUMER_GROUP
-----
ApplicationInstrumentation                OTHER_GROUPS

```

此时如果你再按回车，程序将断开连接并退出。这时在通过命令 **USER_DUMP_DEST** 设定的目录下会有一个扩展的 SQL 跟踪文件，该文件中有等待事件，但没有绑定。为了包括绑定，我们需要设定调用 **DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE** 时的布尔参数 **BINDS** 的值为 **TRUE**。跟踪文件中的相关部分重现如下：

```

*** SERVICE NAME:(TEN.oradbpro.com) 2007-10-30 22:52:52.703
*** SESSION ID:(47.2195) 2007-10-30 22:52:52.703
WAIT #0: nam='SQL*Net message to client' ela= 3 driver id=1413697536 #bytes=1 p3=0
obj#=-1 tim=392092318798
...
*** 2007-10-30 22:53:03.984
WAIT #0: nam='SQL*Net message from client' ela= 11260272 driver id=1413697536
#bytes=1 p3=0 obj#=-1 tim=392103603777
*** ACTION NAME:(act) 2007-10-30 22:53:03.984
*** MODULE NAME:(mod) 2007-10-30 22:53:03.984
*** CLIENT ID:(Ray.Deever) 2007-10-30 22:53:03.984
=====

```



```

PARSING IN CURSOR #2 len=75 dep=0 uid=61 oct=3 lid=61 tim=392103604193 hv=536335290
ad='66afd944'
SELECT userenv('sid'), to_char(sysdate, 'Month dd. yyyy hh24:mi') FROM dual
END OF STMT
PARSE #2:c=0,e=90,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=392103604185
EXEC #2:c=0,e=53,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=392103608774
WAIT #2: nam='SQL*Net message to client' ela= 4 driver id=1413697536 #bytes=1 p3=0
obj#=-1 tim=392103608842
WAIT #2: nam='SQL*Net message from client' ela= 52777 driver id=1413697536 #bytes=1
p3=0 obj#=-1 tim=392103661760
WAIT #2: nam='SQL*Net message to client' ela= 4 driver id=1413697536 #bytes=1 p3=0
obj#=-1 tim=392103661904
FETCH #2:c=0,e=93,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=1,tim=392103661941
WAIT #2: nam='SQL*Net message from client' ela= 233116 driver id=1413697536 #bytes=1
p3=0 obj#=-1 tim=392103895174
*** ACTION NAME:() 2007-10-30 22:53:04.281
*** MODULE NAME:() 2007-10-30 22:53:04.281
*** CLIENT ID:() 2007-10-30 22:53:04.281
STAT #2 id=1 cnt=1 pid=0 pos=1 obj=0 op='FAST DUAL (cr=0 pr=0 pw=0 time=14 us)'
=====
PARSING IN CURSOR #2 len=63 dep=0 uid=61 oct=3 lid=61 tim=392103896865 hv=2359234954
ad='672f1af4'
SELECT 'application instrumentation settings removed' FROM dual
END OF STMT

```

模块和行为名出现在 SELECT 语句（它检索会话标识符）之前，这一事实表明插桩设置同发送来解析 SELECT 语句的网络数据包（network packet）息息相关。JDBC 端到端度量并没有导致额外的网络回路开销。在 IP 级别，额外的数据有可能需要一个额外的数据包，但是并没有存在额外的回路开销被报告为等待事件 SQL*Net message from client。

## 23.5 TRCSESS 的使用

TRCSESS 是 Oracle10g 及其后续版本的一个新工具。它可以从一个或多个跟踪文件中，根据如下选项的配置提取相关部分组成一个单独的文件：

- 服务名 (V\$SESSION.SERVICE_NAME)
- 会话标识 (V\$SESSION.SID 及 V\$SESSION.SERIAL#)
- 模块名 (V\$SESSION.MODULE)
- 行为名 (V\$SESSION.ACTION)
- 客户端标识 (V\$SESSION.CLIENT_IDENTIFIER)

会话标识、客户端标识、服务名、行为、模块，这几个选项中，必须指定一个。如果指定了多个选项，将从跟踪文件中选择满足所有选项的部分来组成输出文件。所有选项的值都是大小写无关的。在 TRCSESS 将跟踪信息生成输出文件后，这个文件将被 TKPROF 或者本书中介绍的扩展的 SQL 跟踪处理器 ESQLTRCPROF 进一步处理。

当不用任何参数调用 TRCSESS 时，将打印它的用法信息。

```
$ trcsess
trcsess [output=<output file name >] [session=<session ID>] [clientid=<clientid>]
[service=<service name>] [action=<action name>] [module=<module name>]
<trace file names>

output=<output file name> output destination default being standard output.
session=<session Id> session to be traced.
Session id is a combination of session Index & session serial number e.g. 8.13.
clientid=<clientid> clientid to be traced.
service=<service name> service to be traced.
action=<action name> action to be traced.
module=<module name> module to be traced.
<trace_file_names> Space separated list of trace files with wild card '*' supported.
```

在专用服务模型中，所有针对某个特定会话的跟踪文件的所有条目都放在一个单一的文件中。但在共享服务和连接池环境中，情形则有所不同。在共享服务模型中，针对某个特定会话的跟踪文件条目分布在由服务程序配置所规定的多个文件中。这是由于一个用户会话经常得到多个不同的共享服务进程的服务。这使得对一个会话进行完整资源分析比较困难。连接池环境下的情况则更糟，除非应用程序已经被监测并设置了一个客户端标识（这使得 DBA 可以找出一个数据库会话正服务于哪个终端用户），否则一个 DBA 不能将一个向他提出请求支持的终端用户与一个特定的数据库会话联系起来。

继续前面章节的例子，我将展示当会话执行 mod 模块和 act 行为时，如何使用 TRCSESS 抽取会话中形成的 SQL 语句。这样，对于本章 23.2 节的代码摘要，你可以刚好抽取里面的检索会话标识的 SELECT 语句到所形成的跟踪文件中。

```
$ trcsess module=mod action=act *.trc
*** 2007-10-30 22:53:03.984
*** 2007-10-30 22:53:03.984
=====
PARSING IN CURSOR #2 len=75 dep=0 uid=61 oct=3 lid=61 tim=392103604193 hv=536335290
ad='66afd944'
SELECT userenv('sid'), to_char(sysdate, 'Month dd. yyyy hh24:mi') FROM dual
END OF STMT
PARSE #2:c=0,e=90,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=392103604185
EXEC #2:c=0,e=53,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=392103608774
WAIT #2: nam='SQL*Net message to client' ela= 4 driver id=1413697536 #bytes=1 p3=0
obj#=-1 tim=392103608842
WAIT #2: nam='SQL*Net message from client' ela= 52777 driver id=1413697536 #bytes=1
p3=0 obj#=-1 tim=392103661760
WAIT #2: nam='SQL*Net message to client' ela= 4 driver id=1413697536 #bytes=1 p3=0
obj#=-1 tim=392103661904
FETCH #2:c=0,e=93,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=1,tim=392103661941
WAIT #2: nam='SQL*Net message from client' ela= 233116 driver id=1413697536 #bytes=1
p3=0 obj#=-1 tim=392103895174
```

要注意所有其他应用监测条目，包括客户端标识（根据行为和模块名生成）是如何从 TRCSESS 的输出中移除的。因为我们很难将模块、行为与特定的客户端标识联系起来，所以我

们需要一个有意义的名字，来说明这个文件被如何处理过。用选项 `session` 调用 TRCSESS 时，将完全保留其他监测条目。而指定其他任何选项时则将移除所有的监测条目而仅仅保留跟踪文件中这些条目的时间戳。因为只有会话标识（`session identification`）选项的格式在 Oracle9i 与 Oracle10g 中是一致的，TRCSESS 对 Oracle9i 跟踪文件的后向兼容能力是很有限的。现在，一切都像你预期的那样。接下来将着力解决在共享服务环境中使用 TRCSESS 的缺陷。

## TRCSESS 与共享服务

根据文档，TRCSESS 是支持共享服务环境的（见 *Oracle Database Performance Tuning Guide 10g Release 2*, 20-26 页）。但是，在共享服务环境中以任何非 `session` 的选项使用 TRCSESS 时，仍有可能得到错误的结果。当一个共享服务进程开始服务于一个开启了 SQL 跟踪的会话时，它仅仅发送回会话标识（`V$SESSION.SID`）和会话序列号（`V$SESSION.SERIAL#`）。所有其他监测条目都没有重复写入跟踪文件。当前的 TRCSESS 版本（版本 10.2.0.3 和 11.1.0.6）显然没有考虑一个会话设置了哪些监测条目。因此，TRCSESS 在没有碰到新的服务名、模块、行为或客户端标识时，就会将跟踪文件中属于其他会话的内容放入到输出文件。为了能够正常工作，每次当共享服务进程服务一个新的会话的时候，TRCSESS 都必须观察到服务名、模块、行为和客户端标识。因为这些条目只写一次，因此使用 TRCSESS 所得到的结果就很可能不正确。后面的测试用例就说明了这一点。如果在你的测试环境中没有开启共享服务选项，你可以用如下方法开启它：

```
SQL> ALTER SYSTEM SET shared_servers=1;
System altered.
SQL> ALTER SYSTEM SET dispatchers='(PROTOCOL=TCP)(DISPATCHERS=1)';
System altered.
```

通过使用一个单一的共享服务进程，我能确保多个会话将出现在同一个跟踪文件中。然后，以 DBA 的身份登录，使用 DBMS_MONITOR 根据服务名、模块、行为进行追踪。

```
SQL> EXEC dbms_monitor.serv_mod_act_trace_enable('TEN.oradbpro.com', 'mod', 'act');
SQL> SELECT trace_type, primary_id, qualifier_id1, qualifier_id2, instance_name
FROM dba_enabled_traces;
TRACE_TYPE          PRIMARY_ID          QUALIFIER_ID1 QUALIFIER_ID2 INSTANCE_NAME
-----
SERVICE_MODULE_ACTION TEN.oradbpro.com mod          act
```

在接下来的测试中，我需要两个会话，这两个会话都开启了 SQL 跟踪。第一个会话手动设定 SQL 跟踪。

```
$ sqlplus ndebes/secret@ten.oradbpro.com
SQL> ALTER SESSION SET sql_trace=true;
Session altered.
```

第二个会话使用的实例服务是通过前述的对 DBMS_MONITOR 的调用设定的，而将模块和行为设定为调用 DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE 时所使用的值。这样就会开启该会话的 SQL 跟踪（`SQL_TRACE`）。在接下来的代码示例中，我已经将 `SQL*Plus` 设置为 `SQLPROMPT`，这样读者可以很容易地识别哪个会话执行了什么。

```
$ sqlplus ndebes/secret@ten.oradbpro.com
SQL> SET SQLPROMPT "MOD_ACT> "
MOD_ACT> exec dbms_application_info.set_module('mod','act');
MOD_ACT> SELECT server, service_name, module, action
FROM v$session WHERE sid=userenv('SID');
SERVER      SERVICE_NAME      MODULE      ACTION
-----
SHARED      TEN.oradbpro.com mod          act
MOD_ACT> SELECT 'mod_act' AS string FROM dual;
STRING
-----
mod_act
```

回到第一个会话，我们可以从 DUAL 表中查找一个字符串。这样做可以在跟踪文件中添加一个独特的字符串，在后面我们将检索它。

```
SQL> SELECT 'not instrumented' AS string FROM dual;
STRING
-----
not instrumented
SQL> ALTER SESSION SET sql_trace=false;
Session altered.
```

现在第一个会话的 SQL 跟踪已经关闭了。回到第二个会话。为了划分跟踪文件以便 TRCSESS 进行提取，并为了关闭追踪，我们将会话的模块和行为设置为空字符串。

```
MOD_ACT> exec dbms_application_info.set_module('', '');
```

你现在可以进入由参数 BACKGROUND_DUMP_DEST^①设定的目录下并运行 TRCSESS。

```
$ trcsess output=mod_act.trc module=mod action=act ten1_s000_8558.trc
$ grep "not instrumented" mod_act.trc
SELECT 'not instrumented' AS string FROM dual
```

尽管第 1 个会话没有将模块和行为当做选项值传递给 TRCSESS，但是由 TRCSESS 生成的输出文件仍包含第 1 个会话中的 SELECT 字符串字面量的语句。这个缺陷在 TRCSESS10.2 和 11.1 中都存在。如前所述，造成这个问题的根源是一个共享服务进程在转向其他会话时将不会重写该会话的所有监测条目。下面是一个共享服务进程的跟踪文件的摘要，它可以验证上述论点（该摘要中包括了所有的监测条目）：

```
1 *** ACTION NAME:(act) 2007-09-06 20:23:52.514
2 *** MODULE NAME:(mod) 2007-09-06 20:23:52.514
3 *** SERVICE NAME:(TEN.oradbpro.com) 2007-09-06 20:23:52.514
4 *** SESSION ID:(133.39) 2007-09-06 20:23:52.514
5 =====
6 PARSING IN CURSOR #7 len=59 dep=0 uid=34 oct=47 lid=34 tim=1161233430189798
hv=919966564 ad='22c5ec10'
7 BEGIN dbms_application_info.set_module('mod','act'); END;
```

① 会话级别的参数 TRACE_FILE_IDENTIFIER 可以使得不同会话的跟踪文件更容易区分，但是那只适用于专用服务进程的情形。在那儿，参数的值将被包含在跟踪文件名中，而共享服务进程忽略此参数。

```

8 END OF STMT
9 ...
10 *** SESSION ID:(146.361) 2007-09-06 20:26:18.442
11 ...
12 *** SESSION ID:(133.39) 2007-09-06 20:26:48.229
13 ...
14 *** SESSION ID:(146.361) 2007-09-06 20:27:02.759
15 =====
16 PARSING IN CURSOR #7 len=45 dep=0 uid=34 oct=3 lid=34 tim=1161233615975704
hv=2134668354 ad='2527ef28'
17 SELECT 'not instrumented' AS string FROM dual
18 ...
19 *** SESSION ID:(133.39) 2007-09-06 20:27:27.502
20 ...
21 PARSING IN CURSOR #2 len=53 dep=0 uid=34 oct=47 lid=34 tim=1161233640139316
hv=3853446089 ad='22c484d4'
22 BEGIN dbms_application_info.set_module('',''); END;
23 END OF STMT
24 PARSE #2:c=0,e=2,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=1161233640139316
25 *** ACTION NAME:() 2007-09-06 20:27:27.503
26 *** MODULE NAME:() 2007-09-06 20:27:27.503

```

第 1、2 行包括的是第 2 个会话调用 DBMS_APPLICATION_INFO 时传递的模块和行为名。第 10~14 行显示了共享服务进程间歇地为另一个进程 (SID=146, SERIAL#=361) 提供服务。第 17 行显示了用户标记第一个会话 (该会话并没有被监测) 的 SELECT 语句的字符串字面量。第 25、26 行标记了跟踪文件关于模块 mod 和行为 act 部分的结束。这时, 因为对第 2 个会话的跟踪已经关闭, 上述模块名和行为名与 DBMS_MONITOR 设置的 DBA_ENABLED_TRACES 的值就不再匹配。也就是说, 端到端跟踪将监听新的模块和行为而关闭 SQL 跟踪^①。尽管在第 10~19 行有一些条目属于其他会话, 但因为没有出现与模块和行为有关的条目, TRCSESS 会把这些行的所有条目都提取出来。

正确地操作 TRCSESS 做法是要记住每个 SID 和 SERIAL# 组合的监测配置, 并据此排除还是包括跟踪文件中新的 SID 和 SERIAL# 组合值后的条目。这种方法可以作为一个变通方案。可以在跟踪文件中紧跟模块和行为之后的部分寻找感兴趣的 SESSION ID。上面的例子中, 我们需要将第 4 行所包含的 SID 和 SERIAL# 的值 (即 133.39) 传递给 TRCSESS, 以便获取正确的输出文件。如果在设置模块和行为之前就开启了 SQL 跟踪, 那么你需要在 ACTIVE NAME 或 MODULE NAME 条目前面查找 SESSION ID。此时, 我们并不将模块和行为名传递给 TRCSESS, 而是将从跟踪文件中获取到的 SID 和 SERIAL# 的值传递给 TRCSESS。

```

$ trcse ss output=session_133_39.trc session=133.39 ten1_s000_8558.trc
$ grep "not instrumented" session_133_29.trc

```

这时, 用于标识第 1 个会话的字符串字面量就没有再出现在输出文件中了。这表明, 只要使

① 专用服务进程在服务于一个由 JDBC 端到端度量进行监测的应用时, 会在发送可关闭 SQL 跟踪的新的模块和行为的设置值之前就关闭 SQL 跟踪。因为专用服务进程在开启 SQL 跟踪时总是重新发送模块、行为和客户端标识, 因此前述情形不会导致错误结果。

用 session 选项, TRCSESS 就可以正确地处理在共享服务进程的情形下生成的跟踪文件。

## 23.6 插桩与程序调用栈

我们前面讲述的例子只是非常简单的情形。实际应用中可能会有许多互相调用的例程。被调用的例程可能并不知道谁调用了它, 而它也并非总是被同一个例程所调用。为了跟踪程序调用栈, 每个例程在它自己的插桩配置来覆盖当前监测配置之前, 需要将当前监测配置压入到栈中。当从一个例程中退出时, 需要从栈顶取出配置并恢复为当前配置。当一个例程由于异常或错误而异常结束时, 从栈中提取配置的操作就需要尤其小心。据我所知, Hotsos 的 ILO^①将这种类型的栈第一次应用于 PL/SQL 平台上。

图 23-1 描述了一个应用中例程之间互相调用的情形。执行在时刻  $t_1$  从例程 A 开始, 此时例程 A 的插桩配置生效。在时刻  $t_2$ , 例程 A 调用例程 B。为了保存例程 A 的插桩配置, 例程 B 将它们压入栈中, 而将自己的配置作为当前配置。在时间点  $t_3$ , 例程 B 调用程序 C, 例程 C 将 B 的插桩配置压入栈中, 而用自己的配置覆盖当前配置。在时间点  $t_4$ , 应用退出例程 C, 此时, 例程 C 就将 B 的插桩配置从栈顶取出来作为当前配置。这个过程继续直至在时间点  $t_7$ , 整个程序从例程 A 退出。右侧的坐标轴指示了在特定的时间点正在执行的例程。如果一个应用的所有代码路径都能应用这种方式进行监测, 那么就可以使用 DBMS_MONITOR 和 V\$SERV_MOD_ACT_STATS 确定每个模块各自消耗的时间。V\$SERV_MOD_ACT_STATS 提供的统计信息包括 DB time (数据库响应时间)、DB CPU (数据库 CPU 消耗时间)、physical reads (物理读)、physical writes (物理写)、cluster wait time (集群等待事件)、concurrency wait time (同步等待时间)、application wait time (应用等待时间) 和 user I/O wait time (用户 I/O 等待时间)。

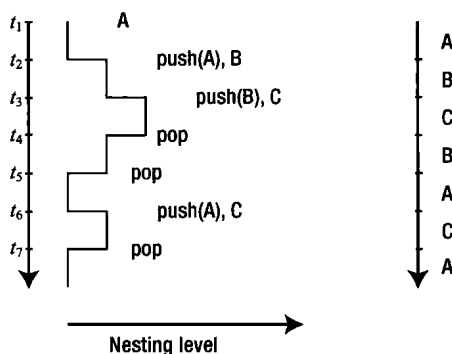


图 23-1 程序调用栈及其插桩

① Hotsos ILO库是免费软件, 可以从<http://sourceforge.net/projects/hotsos-ilo>下载。

## 23.7 源代码库

表 23-1 列出了本章的源文件以及它们的功能。

表23-1 应用插桩源代码库	
文 件 名	功 能
ApplicationInstrumentation.class	源文件ApplicationInstrumentation.java编译所得到的Java程序文件
ApplicationInstrumentation.java	实现Oracle10g JDBC端到端度量的Java源程序文件
JdbcInstrumentationOracle9i.class	源文件JdbcInstrumentationOracle9i.java编译所得到的Java程序文件
JdbcInstrumentationOracle9i.java	用PL/SQL进行监测的Java示例程序的源文件
rsrc_mgr.sql	用于设定资源管理器中从服务名、模块、行为到消费者群映射的匿名PL/SQL程序块





# Part 8

## 第八部分

## 性能

### 本 部 分 内 容

- 第 24 章 扩展 SQL 跟踪文件格式参考
- 第 25 章 Statspack
- 第 26 章 整合扩展 SQL 跟踪和 AWR
- 第 27 章 ESQLTRCPROF 扩展 SQL 跟踪分析器
- 第 28 章 MERITS 性能优化方法

SQL 跟踪文件包含解析和执行 SQL 语句，以及数据库客户端的提取调用的准确记录。这包括 CPU 时间以及解析、执行和提取操作的运行时间。SQL 跟踪文件也可以包括等待事件信息，这是准确的性能诊断不可缺少的信息。最后，很重要是绑定变量的真实值也同样会包含在内，例如当 PL/SQL 包 DBMS_MONITOR 用于启用 SQL 跟踪时。

SQL 跟踪文件的格式在文档中未作说明，即使工具 TKPROF 和 TRCSSESS 是在 SQL 跟踪文件的基础上进行分析。理解扩展 SQL 跟踪文件的格式，是每一个面临性能问题或故障排除任务的 DBA 的必备技能。由于使用 TKPROF 格式化跟踪文件隐藏了重要信息，如语句散列值、时间戳、递归调用深度、实例服务名称、模块、动作以及 SQL 标识符^① (V\$SQL.SQL_ID)，往往就强制我们去阅读和理解跟踪文件本身。

## 24.1 扩展 SQL 跟踪文件介绍

扩展 SQL 跟踪文件大体而言是对一个数据库客户端^②所执行的 SQL 和 PL/SQL 语句的逐条记录。这种文件中的条目分为如下 4 个大类：

- 数据库调用（解析、执行和提取）；
- 等待事件；
- 绑定变量值；
- 其他项（时间戳、实例服务名称、会话、模块、动作以及客户端标识符）。

其他项类别中的数据库调用、会话 ID 以及其他详细信息在跟踪以级别 1 启动时会被记录，例如使用 ALTER SESSION SET SQL_TRACE=TRUE，而对等待事件和绑定变量值的记录需要单独启动。

怎样在不同的级别获取跟踪文件的相关细节是第 28 章的议题。跟踪级别以及它们所启用的跟踪文件条目类型在表 24-1 中进行了总结。

^① TKPROF 版本 11.1.0.7 是第一个在报告中包含 SQL 标识符的发行版本。

^② 后台进程也可能被跟踪，但是它们通常不是性能问题的关键所在。

表24-1 SQL跟踪级别

SQL跟踪级别	数据库调用	绑定变量值	等待事件
1	yes	no	no
4	yes	yes	no
8	yes	no	yes
12	yes	yes	yes

有时候扩展 SQL 跟踪文件被称为原始 SQL 跟踪文件。这两个名词确实是同义的。因为这些文件没有什么特别原始的信息——它们是可以完全被人类读取的，因此我决定不使用形容词“原始的”，为了简洁起见，我坚持使用名词“扩展 SQL 跟踪文件”或简称“跟踪文件”。

## 24.2 SQL 和 PL/SQL 语句

术语“游标”（cursor）经常与 SELECT 语句以及迭代获取查询反馈的数据行一起使用。然而，ORACLE DBMS 使用游标执行任意 SQL 或 PL/SQL 语句，而不仅仅是 SELECT 语句。一个跟踪文件中 SQL 和 PL/SQL 语句依靠它们的游标编号进行识别。客户端发送的 SQL 语句的游标编号从 1 开始。在每一个条目中游标编号是位于#之后的数字，例如 PARSING IN CURSOR #1、PARSE #1、EXEC #1、FETCH #1、WAIT #1 和 STAT #1。这些都涉及相同的游标编号 1。客户端运行另外一个 SQL 语句就会收到另一个游标编号，除非在某一游标关闭之后重用该游标编号。同一语句的条目通过游标编号进行相互关联。

不是所有执行的操作都会被分配一个适当的游标编号。一个值得注意的例外就是通过 OCI 使用大对象（LOB）。当使用 LOB 时，你可能会看到游标编号 0 或缺失了解析调用的游标编号，即使刚好在连接之后打开了跟踪。这对 PL/SQL LOB 接口 DBMS_LOB 并不适用。

一个单一数据库会话中的游标编号可以被重用。当客户端关闭了一个游标，DBMS 写入代表执行计划的 STAT 条目到跟踪文件。在这个阶段，游标编号可被一个不同的 SQL 语句重用。某一游标的 SQL 语句文本会在第一个 PARSING IN CURSOR #n 条目之后，以及任何带有相同游标编号 n 的 EXEC #n、FETCH #n、WAIT #n 或 STAT #n 的上面打印出来。

## 24.3 递归调用深度

任何使用了 TKPROF 工具的人大概会熟悉递归和内部 SQL 语句的概念。一个数据库客户端发送的 SQL 语句在递归调用深度 0 执行。当一个 SQL 语句激活其他语句时，比如 INSERT 语句会激活一个插入触发器执行，然后这些语句将会在递归调用深度 1 执行。一个触发器主体可能会执行额外的语句，这将会在下一个更高的递归调用深度引发递归 SQL。下面是一个在递归调用深度 0 执行 INSERT 语句的例子。该 INSERT 语句激活一个触发器。触发器主体中访问一个序列的递归调用深度为 1。请注意顶层的 INSERT 语句 (EXEC #3) 在其依赖的语句 (EXEC #2) 从序列中的 SELECT 语句执行完毕之后怎样被写入跟踪文件。

```
PARSING IN CURSOR #3 len=78 dep=0 uid=61 oct=2 lid=61 tim=771237502562 hv=3259110965
ad='6c5f86dc'
```

```

INSERT INTO customer(name, phone) VALUES (:name, :phone) RETURNING id INTO :id
END OF STMT
PARSE #3:c=0,e=1314,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=771237502553
=====
PARSING IN CURSOR #2 len=40 dep=1 uid=61 oct=3 lid=61 tim=771237506650 hv=1168215557
ad='6c686178'
SELECT CUSTOMER_ID_SEQ.NEXTVAL FROM DUAL
END OF STMT
PARSE #2:c=0,e=1610,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=1,tim=771237506643
EXEC #2:c=0,e=57,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=1,tim=771237507496
FETCH #2:c=0,e=54,p=0,cr=0,cu=0,mis=0,r=1,dep=1,og=1,tim=771237507740
EXEC #3:c=0,e=4584,p=0,cr=1,cu=3,mis=1,r=1,dep=0,og=1,tim=771237508046

```

当客户端运行一个匿名 PL/SQL 块时，该块本身就在递归调用深度 0 执行，但是该块内部的语句的递归调用深度为 1。另一个例子就是插入到表 SYS.AUD\$ 的审计条目。它们在比触发审计的语句高一个递归调用的深度执行。

递归解析、执行和获取操作被列在触发递归操作的执行语句之前。在递归调用深度 0 执行的 SQL 语句的统计数据包括依赖语句的 CPU 时间、运行时间、一致读取等开销。这一点必须加以考虑，以避免在评估 SQL 跟踪文件时重复计算。

## 24.4 数据库调用

数据库调用分类包含 3 个子分类：解析、执行和获取。请注意，这些条目与通过调用包 DBMS_SQL 的子例程 DBMS_SQL.PARSE、DBMS_SQL.EXECUTE 和 DBMS_SQL.FETCH_ROWS 来运行动态 SQL 的三步相一致。

在其他指标中，数据库调用条目表示一个服务器进程（代表一个数据库客户端）在 ORACLE 内核中所消耗的 CPU 和挂钟时间（总运行时间）。在一个跟踪文件中，来自数据库调用的 CPU 和挂钟时间总是和动态性能视图 V\$SESS_TIME_MODEL 中的会话级别的统计信息 DB CPU 和 DB 时间密切相关，该视图在 Oracle10g 及其后续的版本中可用。

### 24.4.1 解析

解析涉及 SQL 语句的语法和语义分析以及确定一个合适的执行计划。由于这可能是一个代价昂贵的操作，ORACLE DBMS 具有在 SGA 中所谓的库缓存中缓存解析调用结果的能力，以供其他使用相同 SQL 语句文本的语句使用。

在 SQL 语句中使用绑定变量对于重用已缓存的语句是至关重要的。无法使用绑定变量会导致解析 CPU 消耗的增加、库缓存的争用、由于使用常量重复解析调用不能重用的语句所带来的客户端与服务器之间的过度通信开销，以及因为 TKPROF 工具不能聚集相同语句（除了常量之外）而造成诊断性能问题时的困难。我建议在设计开始和对一个应用程序编码之前阅读 *Oracle Database Performance Tuning Guide 10g Release 2* 一书第 3~4 页的章节 Top Ten Mistakes Found in Oracle Systems。然而为了在高容量事务处理实现高可扩展性，绑定变量是必需的，在数据仓库应用中通常倾向于使用常量为 CBO 提供尽可能多的信息以避免绑定变量窥视内在的不可预知

性。CBO 在它第一次遇到一个语句时查看绑定变量，但不会后续执行相同的语句，这样选择的计划会对初始执行进行优化而对后续执行不适合。该功能被称为绑定变量窥视。它使用隐藏参数设置 OPTIM_PEEK_USER_BINDS=TRUE 默认启动。

解析在跟踪文件中通常通过两个相邻的条目表示。第一个是 PARSING IN CURSOR，第二个是 PARSE。启用解析相关条目的最小 SQL 跟踪级别是 1。下面是来自 Oracle10g 的一个跟踪文件的例子，其中 PARSING IN CURSOR 后面跟了一个 PARSE。

```
PARSING IN CURSOR #3 len=92 dep=0 uid=30 oct=2 lid=30 tim=81592095533 hv=1369934057
ad='66efcb10'
INSERT INTO poem (author, text) VALUES(:author, empty_clob())
RETURNING text INTO :lob_loc
END OF STMT
PARSE #3:c=0,e=412,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=81592095522
```

PARSING IN CURSOR 条目相关的参数解释见表 24-2。

表24-2 PARSING IN CURSOR参数

参 数	意 义
len	SQL语句文本的字节长度
dep	递归调用深度
uid	解析用户标识符，对应于ALL_USERS.USER_ID以及V\$SQL.PARSING_USER_ID
oct	ORACLE命令类型，对应于V\$SQL.COMMAND_TYPE以及V\$SESSION.COMMAND
lid	解析模式标识符，对应于ALL_USERS.USER_ID 以及V\$SQL.PARSING_SCHEMA_ID，可能与uid不同（见第14章“ALTER SESSION SET CURRENT_SCHEMA”）
tim	微秒单位的时间戳，在关联的PARSE条目中通常比tim的值早一点
hv	散列值，对应于V\$SQL.HASH_VALUE
ad	地址，对应于V\$SQL.ADDRESS
sqlid	SQL标识符，对应于V\$SQL.SQL_ID（Oracle11g中发表）

24.4.2 PARSING IN CURSOR 条目的格式

在共享池中缓存的 SQL 语句基于来自 SQL 或 PL/SQL 语句文本的散列值。优化设置的改变不会对散列值产生影响，而只是稍微改变一下语句文本，比如插入一个空格或制表符。两个具有不同语句文本的语句而有相同的散列值这种情况是非常罕见的。

该散列值可以从许多 VS视图检索到，比如 V\$SQL、V\$SQLTEXT、V\$SQLAREA、V\$OPEN_CURSOR 以及 V\$SESSION。它在实例启动过程中保持不变，但可能在升级到一个新的版本之后发生改变。实际上，用于计算散列值的算法在 Oracle10g 已经改变了。兼容之前版本的散列值可以通过视图 V\$SQL 和 V\$SQLAREA 中的列 OLD_HASH_VALUE 得到。只有该散列值被发送到跟踪文件。由于 Statspack 继续使用老版本的散列值，但只有新版本散列值被发送到跟踪文件，当在 Statspack 库中搜索与一个跟踪文件中的语句相关的信息时，这就增加了从新散列值翻译到旧散列值的复杂度（更多信息请见第 25 章）。

Oracle10g 为之前提到的一些 VS视图引入了新的列 SQL_ID。这个新列的值在 Oracle11g 之前

的发行版本中不会被写入到 SQL 跟踪文件，但会被用于 AWR 报告，这样在 AWR 中查找一个语句的信息时就需要将新散列值（列 HASH_VALUE）翻译为 SQL_ID。对于已被缓存了的 SQL 语句，SQL_ID、HASH_VALUE 与 OLD_HASH_VALUE 之间的转换使用 V\$SQL 实现。对于那些不再缓存但是会被 Statspack 快照捕捉的语句，STATS\$SQL_SUMMARY 就充当了转换表的角色（SQL 语句标识符的罗塞塔石碑^①）。AWR 不具有将跟踪文件中的散列值翻译为对应 SQL_ID 的功能。在 Oracle11g 之前的发行版本中，匹配两种捕获的语句文本是用于提取来自 AWR 语句历史信息的唯一时间开销，比如之前的执行时间和计划。考虑到 Statspack 不需要额外的许可并包含会话级别捕捉与报告（当心错误 5145816，见第 25 章表 25-4），AWR 的这个缺点就使得我们偏爱 Statspack 又多了一个理由。

除了发送散列值，Oracle11g 是第一个发送 SQL 标识符（V\$SQL.SQL_ID）到跟踪文件的 DBMS 发行版本。因此扩展 SQL 跟踪与 AWR 之间语句匹配问题对于 Oracle11g 用户来讲已经成为了过去。下面是一个 Oracle11g 中 PARSING IN CURSOR 条目的例子：

```
PARSING IN CURSOR #3 len=116 dep=0 uid=32 oct=2 lid=32 tim=15545747608 hv=1256130531
ad='6ab5ff8c' sqlid='b85s0yd5dy1z3'
INSERT INTO customer(id, name, phone)
VALUES (customer_id_seq.nextval, :name, :phone) RETURNING id INTO :id
END OF STMT
```

在以字符串 PARSING IN CURSOR 开头的行之后，解析了的 SQL 语句作为新的一行打印在这里。以 END OF STMT 开头的行标记了该 SQL 语句的结束。从数值命令类型（参数 oct）到命令名称的映射可以通过运行 SELECT action、name FROM audit_actions 得到。表 24-3 包含了大多数常见的命令类型以及一些可能用于应用程序的额外命令类型。请注意这些数值命令类型并不与 OCI SQL 命令代码相对应（Oracle Call Interface Programmer's Guide 10g Release 2，附录 A）。

表24-3 SQL与PL/SQL命令类型

数值命令类型	SQL或PL/SQL命令
2	INSERT
3	SELECT
6	UPDATE
7	DELETE
26	LOCK TABLE
44	COMMIT
45	ROLLBACK
46	SAVEPOINT
47	PL/SQL block
48	SET TRANSACTION
55	SET ROLE
90	SET CONSTRAINTS
170	CALL
189	MERGE

① 罗塞塔石碑（rosetta stone）原指一块致使人类破解了古埃及语言与文化的特殊石头。——译者注

### 24.4.3 PARSE 条目的格式

在其他指标中，PARSE 条目代表解析操作消耗 CPU 与挂钟时间。通过查看参数 mis（库缓存丢失）就可以从跟踪文件得到库缓存命中率。低的命中率通常说明应用程序没有使用绑定变量。PARSE 条目中参数的详细信息见表 24-4。

表24-4 PARSE参数

参 数	意 义
c	CPU开销
e	运行时间
p	物理读
cr	一致读
cu	当前处理的数据块
mis	游标丢失，0为软解析，即在库缓存中找到语句；1为硬解析，即语句没有找到
r	被处理的数据行
dep	递归调用深度
og	优化目标，1为ALL_ROWS，2为FIRST_ROWS，3为RULE，4为CHOOSE；Oracle9i默认为CHOOSE，ORACLE10g以及Oracle11g默认为ALL_ROWS
plh	执行计划散列值，对应于V\$SQL_PLAN.PLAN_HASH_VALUE、V\$SQL_PLAN_STATISTICS_ALL.PLAN_HASH_VALUE以及V\$SQLSTATS.PLAN_HASH_VALUE，其中V\$SQLSTATS.PLAN_HASH_VALUE在发行版本11.1.0.7中引入（详见本章的24.4.7节）
tim	时间戳，单位为微秒

24

下面的 SELECT 语句证实了跟踪文件条目与数据字典视图之间的联系：

```
SQL> SELECT s.sql_text, u1.username user_name,
u2.username schema_name, optimizer_mode
FROM v$sql s, all_users u1, all_users u2
WHERE hash_value='1369934057'
AND address=upper('66efcb10')
AND u1.user_id=s.parsing_user_id
AND u2.user_id=s.parsing_schema_id;
SQL_TEXT USER_NAME SCHEMA_NAME OPTIMIZER_MODE
-----
INSERT INTO poem (author, text) NDEBES NDEBES ALL_ROWS
VALUES(:author, empty_clob())
RETURNING text INTO :lob_loc
```

假设 PARSE #n 丢失，这样 PARSING IN CURSOR #n 后面就会直接跟上 EXEC #n，例如在这个摘自调用 Perl DBI 的 \$dbh->do 函数的跟踪文件中（详见第 22 章 Perl DBI）：

```
PARSING IN CURSOR #4 len=69 dep=0 uid=30 oct=42 lid=30 tim=81591952901 hv=3164292706
ad='67339f7c'
alter session set events '10046 trace name context forever, level 12'
END OF STMT
EXEC #4:c=0,e=68910,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=81591952889
```

### 24.4.4 . PARSE ERROR 条目的格式

解析调用失败由不正确的语法或权限不足导致，比如 ORA-00942: table or view does not exist 或 ORA-00904: invalid identifier 错误。这些错误会被 PARSE ERROR 条目标记。下面是一个由错误拼写列名导致的解析错误：

```
PARSING IN CURSOR #6 len=93 dep=0 uid=30 oct=2 lid=30 tim=170048888062 hv=986445513
ad='676bb350'
INSERT INTO poem (author, txt) VALUES(:author, empty_clob())
RETURNING ROWID INTO :row_id
END OF STMT
PARSE #6:c=0,e=457,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=170048888050
=====
PARSING IN CURSOR #7 len=198 dep=1 uid=0 oct=3 lid=0 tim=170048915712 hv=4125641360
ad='67b0c8d4'
...
PARSE ERROR #6:len=93 dep=0 uid=30 oct=2 lid=30 tim=170048972426 err=904
INSERT INTO poem (author, txt) VALUES(:author, empty_clob())
RETURNING ROWID INTO :row_id
```

正如前面的例子所示，DBMS 可能需要运行递归 SQL 语句 (dep=1) 处理解析调用以加载字典缓存。其中使用了类似于 PARSE 条目的缩略语，除了最后一个字段 err，它指示 ORACLE 错误编号。UNIX 系统上，错误消息文本可以通过在 shell 命令行中调用 oerr ora error_number 检索到。在适当的情况下，oerr 的输出同样包含错误的可能原因并给出了行动建议。相同的信息可以通过在 *Oracle Database Error Messages* 手册中查找错误编号得到。

```
$ oerr ora 942
00942, 00000, "table or view does not exist"
// *Cause:
// *Action:
```

### 24.4.5 EXEC 条目的格式

EXEC 条目和 PARSE 条目具有相同的格式。EXEC 是 execution 的简写。用于启用 EXEC 条目的最小 SQL 跟踪级别是 1。INSERT、UPDATE、DELETE、MERGE、PL/SQL 和 DDL 操作都有一个执行阶段，但没有获取阶段。当然，代表这些操作的递归获取可能在一个更高的递归调用深度上发生。当使用 TKPROF 格式化包含大量这些操作的跟踪文件时，你可能需要使用选项 exeela (execute elapsed time, 执行运行时间) 对语句进行排序。执行运行时间包括 EXEC 条目消耗的 CPU 时间，尽管偶尔会报告 CPU 时间比运行时间高。

### 24.4.6 FETCH 条目的格式

FETCH 条目遵循与 PARSE 以及 EXEC 条目相同的格式。启用 FETCH 条目的最小 SQL 跟踪级别为 1。当使用 TKPROF 格式化一个获取操作占用了绝大部分运行时间的跟踪文件时，你可能需要使用选项 fchela (fetch elapsed time, 获取运行时间) 对语句进行排序。获取运行时间包括获取 CPU 时间。



### 24.4.7 执行计划散列值

从 Oracle9i 开始, 执行计划可以通过 V\$视图 V\$SQL_PLAN 以及 V\$SQL_PLAN_STATISTICS_ALL 访问。以前, 执行计划仅仅在 SQL 跟踪文件中可见。对于单个 SQL 语句, 随着时间推移可能会有多个执行计划, 这些执行计划也可能同时执行。不同的优化器统计数据、与基于成本优化器 (又名优化环境) 相关的参数之间的差别以及 Oracle11g 中引入的自适应游标共享 (adaptive cursor sharing) 是造成多个执行计划的潜在原因。发行版本 11.1.0.7 (Oracle11g PS1) 是第一个在 SQL 跟踪文件中包含执行计划散列值的版本。它通过参数 PARSE、EXEC 以及 FETCH 条目 (见表 24-4) 的 plh 参数表示。这个新特性解决了 SQL 跟踪文件存在的一个问题——一个语句的多个计划出现在 SGA 中但没有一个计划被发送到 SQL 跟踪文件。以前, 是因为它不能决定哪一个计划被应用到被跟踪的会话中。而版本 11.1.0.7 中包含了 plh, 这就使得从 V\$SQL_PLAN 检索丢失的计划变得没有意义。这个新参数同时也有利于使用函数 DBMS_XPLAN.DISPLAY_CURSOR 检索一个执行计划的额外信息, 相比于被发送到 SQL 跟踪文件的消息, 该函数返回执行计划更多的消息。实际上, 它提供了与事件 10053 所产生信息的范围相匹配的异常详尽的细节。DBMS_XPLAN.DISPLAY_CURSOR 的签名转载如下:

```
FUNCTION display_cursor(sql_id VARCHAR2 DEFAULT NULL,
  cursor_child_no INTEGER DEFAULT 0,
  format VARCHAR2 DEFAULT 'TYPICAL')
RETURN dbms_xplan_type_table
PIPELINED;
```

该函数返回的细节总量取决于参数 FORMAT。为了达到与事件 10053 输出相同级别的细节, 需要提供两个未在文档中说明的格式选项 ADVANCED 和 PEEKED_BINDS。ADVANCED 在一个标题为 Outline Data 的部分包含了一组充分说明执行计划的提示。顾名思义, 格式选项 PEEKED_BINDS 报告了被窥视的绑定变量值, 除非使用 _OPTIM_PEEK_USER_BINDS=FALSE 将绑定变量窥视禁用。下一个例子基于跟踪文件条目与 V\$视图的相互关系说明了 DBMS_XPLAN.DISPLAY_CURSOR 的使用。

### 24.4.8 计划散列值案例研究

在下面的代码示例中使用了样例模式表 EMPLOYEES 和 DEPARTMENTS 的联合, 以突出强调通过 SQL 跟踪和 V\$视图二者可见的信息的范围。

```
SQL> ALTER SESSION SET SQL_TRACE=TRUE;
Session altered.
SQL> VARIABLE dept_id NUMBER
SQL> VARIABLE emp_id NUMBER
SQL> VARIABLE fn VARCHAR2(30)
SQL> EXEC :dept_id:=50; :emp_id:=120; :fn:='Matthew'
SQL> SELECT emp.last_name, emp.first_name, d.department_name
FROM hr.employees emp, hr.departments d
WHERE emp.department_id=d.department_id
AND d.department_id=:dept_id
```

```
AND emp.employee_id=:emp_id
AND first_name=:fn;
PL/SQL procedure successfully completed.
```

在版本 11.1.0.7 中，上面的 equi-join 会导致下面的跟踪文件条目（节选）：

```
PARSING IN CURSOR #2 len=201 dep=0 uid=32 oct=3 lid=32 tim=1000634574600
hv=3363518900 ad='1d7f9688' sqlid='9w4xpcb47qfdn'
SELECT e.last_name, e.first_name, d.department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id=d.department_id
AND d.department_id=:dept_id
AND e.employee_id=:emp_id
AND first_name=:fn
END OF STMT
PARSE #2:c=0,e=0,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,plh=0,tim=1000634574600
EXEC #2:c=0,e=0,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,plh=4225575861,tim=1000634574600
FETCH #2:c=0,e=0,p=0,cr=4,cu=0,mis=0,r=1,dep=0,og=1,plh=4225575861,tim=1000634574600
STAT #2 id=1 cnt=1 pid=0 pos=1 obj=0 op='NESTED LOOPS (cr=4 pr=0 pw=0 time=0 us cost
=2 size=38 card=1)'
STAT #2 id=2 cnt=1 pid=1 pos=1 obj=19136 op='TABLE ACCESS BY INDEX ROWID DEPARTMENTS
(cr=2 pr=0 pw=0 time=0 us cost=1 size=16 card=1)'
STAT #2 id=3 cnt=1 pid=2 pos=1 obj=19137 op='INDEX UNIQUE SCAN DEPT_ID_PK (cr=1 pr=0
pw=0 time=0 us cost=0 size=0 card=1)'
STAT #2 id=4 cnt=1 pid=1 pos=2 obj=19139 op='TABLE ACCESS BY INDEX ROWID EMPLOYEES (
cr=2 pr=0 pw=0 time=0 us cost=1 size=22 card=1)'
STAT #2 id=5 cnt=1 pid=4 pos=1 obj=19143 op='INDEX UNIQUE SCAN EMP_EMP_ID_PK (cr=1 p
r=0 pw=0 time=0 us cost=0 size=0 card=1)'
```

由于该 SQL 语句没有被缓存在库缓存中，该解析调用就会引起一次游标丢失 (mis=1)。因为该游标丢失，我们还不知道该计划的散列值。如果该语句已经被缓存了 (mis=0)，那么 plh 就会被发送。EXEC 与 FETCH 调用都包含执行计划散列值 (plh)。为了使用 DBMS_XPLAN.DISPLAY_CURSOR 获取上面 STAT 条目代表的执行计划额外信息，必须确定子游标编号。

```
SQL> SELECT DISTINCT child_number FROM v$sql_plan WHERE sql_id='9w4xpcb47qfdn'
AND plan_hash_value=4225575861;
CHILD_NUMBER
-----
0
```

如果遇到上述的查询返回多余一行数据的情况，V\$视图 V\$SQL_SHARED_CURSOR 就说明了哪些类型不适于创建多个子游标。尽管如此，带有相同 SQL 标识符 (splid) 和计划散列值的语句仍被同一执行计划处理。下面是描述发生不匹配时可能原因的一个例子，我们从 61 列中选取了 3 列：

```
SQL> SELECT child_number, user_bind_peek_mismatch, optimizer_mode_mismatch,
bind_mismatch
FROM v$sql_shared_cursor
WHERE sql_id='9w4xpcb47qfdn';
CHILD_NUMBER USER_BIND_PEEK_MISMATCH OPTIMIZER_MODE_MISMATCH BIND_MISMATCH
-----
0 N N N
1 N Y N
```

在这个阶段 SQL 标识符与子游标编号都是已知的，这样就可以调用 DBMS_XPLAN。格式选项 ALLSTATS 是 IOSTATS（计划中的列 Buffers）与 MEMSTATS（自动 PGA 内存管理统计数据）组合的简写。选项 LAST 相比于一个语句的所有执行，它仅请求最后的执行统计数据。为了提高可读性，下面的计划表被分为两个部分，列 Id 在二者中是重复的：

```
SQL> SELECT *
FROM TABLE (dbms_xplan.display_cursor('9w4xpcb47qfdn', 1,
'ADVANCED ALLSTATS LAST +PEEKED_BINDS'));

PLAN_TABLE_OUTPUT
-----
SQL_ID 9w4xpcb47qfdn, child number 1
-----
SELECT e.last_name, e.first_name, d.department_name FROM hr.employees
e, hr.departments d WHERE e.department_id=d.department_id AND
d.department_id=:dept_id AND e.employee_id=:emp_id AND first_name=:fn
```

Plan hash value: 4225575861

Id	Operation	Name	Starts	E-Rows	E-Bytes
0	SELECT STATEMENT		1		
1	NESTED LOOPS		1	1	38
2	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	1	1	16
* 3	INDEX UNIQUE SCAN	DEPT_ID_PK	1	1	
* 4	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	1	22
* 5	INDEX UNIQUE SCAN	EMP_EMP_ID_PK	1	1	

Id	Cost (%CPU)	E-Time	A-Rows	A-Time	Buffers
0	2 (100)		1	00:00:00.01	4
1	2 (0)	00:00:01	1	00:00:00.01	4
2	1 (0)	00:00:01	1	00:00:00.01	2
* 3	0 (0)		1	00:00:00.01	1
* 4	1 (0)	00:00:01	1	00:00:00.01	2
* 5	0 (0)		1	00:00:00.01	1

Query Block Name / Object Alias (identified by operation id):

- 1 - SEL\$1
- 2 - SEL\$1 / D@SEL\$1
- 3 - SEL\$1 / D@SEL\$1
- 4 - SEL\$1 / E@SEL\$1
- 5 - SEL\$1 / E@SEL\$1

Outline Data

```
-----
/*+
BEGIN OUTLINE DATA
```

```

IGNORE_OPTIM_EMBEDDED_HINTS
OPTIMIZER_FEATURES_ENABLE('11.1.0.7')
DB_VERSION('11.1.0.7')
FIRST_ROWS
OUTLINE_LEAF(@"SEL$1")
INDEX_RS_ASC(@"SEL$1" "D"@"SEL$1" ("DEPARTMENTS"."DEPARTMENT_ID"))
INDEX_RS_ASC(@"SEL$1" "E"@"SEL$1" ("EMPLOYEES"."EMPLOYEE_ID"))
LEADING(@"SEL$1" "D"@"SEL$1" "E"@"SEL$1")
USE_NL(@"SEL$1" "E"@"SEL$1")
END_OUTLINE_DATA
*/
Peeked Binds (identified by position):
-----
1 - (NUMBER): 50
2 - (NUMBER): 120
3 - (VARCHAR2(30), CSID=178): 'Matthew'
Predicate Information (identified by operation id):
-----
3 - access("D"."DEPARTMENT_ID"=:DEPT_ID)
4 - filter(("FIRST_NAME"=:FN AND "E"."DEPARTMENT_ID"=:DEPT_ID))
5 - access("E"."EMPLOYEE_ID"=:EMP_ID)
Column Projection Information (identified by operation id):
-----
1 - "D"."DEPARTMENT_NAME"[VARCHAR2,30], "FIRST_NAME"[VARCHAR2,20], "E"."LAST_NAME"
[VARCHAR2,25]
2 - "D"."DEPARTMENT_NAME"[VARCHAR2,30]
3 - "D".ROWID[ROWID,10]
4 - "FIRST_NAME"[VARCHAR2,20], "E"."LAST_NAME"[VARCHAR2,25]
5 - "E".ROWID[ROWID,10]

68 rows selected.

```

DBMS_XPLAN 的输出提供了估计行 (E-Rows) 相比于实际行 (A-Rows)、估计时间 (E-Time) 相比于实际时间、窥视绑定、谓语句和列投射的信息。除了实际行与实际时间之外, 该信息不存在于 SQL 跟踪文件中。如果一个不准确的估计造成一个较慢的执行计划, 它可以使用 SQL profile 进行更正, 这是因为我们获得了诊断和调优包二者的许可证。通过 CBO 更正一个错误可能会产生一个更快的执行计划。

### 24.4.9 CLOSE 条目的格式

顾名思义, CLOSE 条目指示一个游标关闭。这种类型的条目随发行版本 11.1.0.7 (Oracle11g 版本 1 PSI) 引入。启用 CLOSE 条目的最小跟踪级别为 1。下面一个例子演示了游标 2 怎样被解析调用打开、执行更新、发送该执行计划以及关闭游标等一系列操作:

```

PARSING IN CURSOR #2 len=42 dep=0 uid=32 oct=6 lid=32 tim=1177102683619
hv=2139321929 ad='1f4b3f6c' sqlid='12kkd2pzs6xk9'
UPDATE hr.employees SET salary=salary*1.10
END OF STMT

```

```
PARSE #2:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,plh=964452392,tim=1177102683619
EXEC #2:c=31250,e=167015,p=0,cr=7,cu=236,mis=0,r=108,dep=0,og=1,plh=964452392,
tim=1177102850634
STAT #2 id=1 cnt=0 pid=0 pos=1 obj=0 op='UPDATE EMPLOYEES (cr=7 pr=0 pw=0
time=0 us)'
STAT #2 id=2 cnt=108 pid=1 pos=1 obj=19139 op='TABLE ACCESS FULL EMPLOYEES (cr=7
pr=0 pw=0 time=0 us cost=3 size=428 card=107)'
CLOSE #2:c=0,e=0,dep=0,type=0,tim=1177105944410
```

- 表 24-5 解释了与 CLOSE 条目相关的参数。一个游标被硬关闭 (type=0)，可能包括很多原因。
- ❑ 该游标与一个 DDL 语句相关联。DDL 语句不能被缓存。
  - ❑ 服务器端游标缓存被禁用，即初始化参数 SESSION_CACHED_CURSORS 的值为 0。
  - ❑ SQL 语句可以被缓存并且服务器端游标缓存已经打开，但是该语句的执行次数少于 3 次。

表24-5 CLOSE参数

参 数	意 义
c	CPU时间
e	运行时间
dep	递归调用深度
type	close操作类型： 0，硬关闭，该游标没有被放到服务器端游标缓存 1，该游标被缓存到服务器端游标缓存以前的空slot中，因为它被执行了至少3次 2，该游标被放置于服务器端游标缓存的slot中，其代价是将另一个游标作废，因为它被执行了至少3次 3，该游标仍在服务器端游标缓存中
tim	时间戳

CLOSE 条目适合于计算服务器端游标缓存的效率。Oracle11g 是第一个启用了服务器端游标缓存 (默认大小为 50 slot) 的发行版本。在之前的版本中服务器端游标缓存默认为禁用状态。Oracle 公司 2005 年发表的论文 *Designing applications for performance and scalability* [Engs 2005] 中表明：重复软解析相同 SQL 语句的应用程序的可扩展性在 SESSION_CACHED_CURSORS 为一个足够大的值时确实会有所改进。避免重复软解析的应用程序会更加具有可扩展性，例如通过启用一个客户端游标缓存。启用客户端游标缓存的方法取决于使用的 Oracle API。Oracle JDBC 驱动包含对通过类 OracleConnection 的方法 setImplicitCachingEnabled 和 setStatementCacheSize 使用客户端游标缓存的支持。

24.5 COMMIT 与 ROLLBACK

ORACLE DBMS 不需要客户端显式地开始一个事务。DBMS 在第一个数据项目被修改或分布式操作执行后自动打开一个事务，比如通过数据库链接从一个表执行 SELECT。后一个操作使用了 TX 和 DX 入队，它们在 COMMIT 结束时被释放。在跟踪文件中事务的边界通过 XCTEND 条目标记。启用 XCTEND 条目的最小 SQL 跟踪级别为 1。它们的格式如下：

```
XCTEND rlbk=[0-1], rd_only=[0-1]
```

表 24-6 解释了 XCTEND 条目中使用的参数。

表24-6 XCTEND参数	
参 数	意 义
rlbk	rollback（回滚）的简写，rlbk=0代表COMMIT，rlbk=1代表ROLLBACK
rd_only	只读事务，rd_only=0代表发生的读写操作，rd_only=1代表只读，没有数据发生改变

XCTEND 条目的两个参数允许 4 种组合。表 24-7 进行了总结。请注意 rd_only=1 不意味着该会话事前发出了语句 SET TRANSACTION READ ONLY。

表24-7 XCTEND参数组合		
XCTEND参数	客户端操作	例 子
rlbk=0, rd_only=1	COMMIT，没有数据被修改	从远程表SELECT之后COMMIT释放锁
rlbk=0, rd_only=0	COMMIT，数据被修改	在INSERT/UPDATE/DELETE一行或多行数据之后COMMIT
rlbk=1, rd_only=1	ROLLBACK，没有数据被修改	从本地表SELECT之后ROLLBACK，例如SET TRANSACTION READ ONLY之后增加快照SCN
rlbk=1, rd_only=0	ROLLBACK，数据被修改	在修改数据之后ROLLBACK，例如通过一个MERGE语句

24.6 UNMAP

当在 Oracle10g 中使用 Ctrl+C 中断正在进行的磁盘排序时，UNMAP 条目被写入到跟踪文件，不考虑参数 WORKAREA_SIZE_POLICY 的设置。这显然涉及在 V\$SORT_USAGE 中观察到的排序段的清理。在我所有 Oracle9i 和 Oracle10g 的测试中，SELECT 语句从触发一个磁盘排序运行到结束的过程中，我从来都没看到一个 UNMAP 条目。我的结论就是 UNMAP 条目不是正常操作的一部分，不值得进一步调研。下面是一个例子：

```
FETCH #3:c=1972837,e=30633987,p=4725,cr=4405,cu=25,mis=0,r=1,dep=0,og=1,tim=18549919
3369
FETCH #3:c=0,e=3113,p=0,cr=0,cu=0,mis=0,r=15,dep=0,og=1,tim=185499197879
...
UNMAP #3:c=0,e=136661,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,tim=185552120691
```

UNMAP 参数与 PARSE 条目相同。

24.7 执行计划、统计信息与 STAT 条目的格式

STAT 条目报告了执行计划和统计数据。一组 STAT 条目的每一行代表了形成语句结果的行源。名词行源指从索引或表中检索的数据或一个较大执行计划的中间结果。因为不能同时对两个以上的表进行连接，一个 3 路连接包含连接其他两个表的中间结果的行源。启用 STAT 条目的最小 SQL 跟踪级别为 1。表 24-8 总结了 STAT 条目的参数。

表24-8 STAT参数

参 数	意 义
id	在执行计划中指示行数据源顺序的标识符，通常一个执行计划的第一条STAT行的id=1
cnt	被处理的行数
pid	父标识符，通常一个执行计划的第一条STAT行的pid=0。通过比一个计划的父步骤高一个级别缩进的依赖步骤，TKPROF和ESQLTRCPROF使用id以及pid生成适当缩进的执行计划
pos	父步骤中一个步骤的位置
obj	对象标识符，对应于ALL_OBJECTS.OBJECT_ID和V\$SQL_PLAN.OBJECT#
op	执行的行数据源操作，比如表访问、索引扫描、排序、联合等，对应于V\$SQL_PLAN.OPERATION。在Oracle10g中，在行数据源信息后面op的圆括号中包含实际语句执行指标

### 24.7.1 Oracle9i 中 STAT 条目的格式

Oracle9i 和 Oracle10g 中 STAT 条目唯一的不同就是参数 op 所传递的消息总数。Oracle9i 的 op 实现中不含实际执行指标。下面是一个散列连接的例子，它是两个全表扫描的父行源：

```
STAT #3 id=1 cnt=106 pid=0 pos=1 obj=0 op='SORT ORDER BY '
STAT #3 id=2 cnt=106 pid=1 pos=1 obj=0 op='HASH JOIN '
STAT #3 id=3 cnt=27 pid=2 pos=1 obj=6764 op='TABLE ACCESS FULL DEPARTMENTS '
STAT #3 id=4 cnt=107 pid=2 pos=2 obj=6769 op='TABLE ACCESS FULL EMPLOYEES '
```

上面的 STAT 条目经 TKPROF 格式化后如下：

```
Rows      Row Source Operation
-----
106      SORT ORDER BY
106      HASH JOIN
27       TABLE ACCESS FULL DEPARTMENTS
107      TABLE ACCESS FULL EMPLOYEES
```

### 24.7.2 Oracle10g 和 Oracle11g 中 STAT 条目的格式

在 Oracle10g 与 Oracle11g 中，与 Oracle9i 相反，STAT 条目仅在 TIMED_STATISTICS=TRUE 并且 SQL 跟踪级别至少为 1 时才被写入。请注意 Oracle10g 和 Oracle11g 中设置 STATISTICS_LEVEL=BASIC (默认为 TYPICAL) 时会隐式设置 TIMED_STATISTICS=FALSE。该行为可以通过显式设置 TIMED_STATISTICS=TRUE 覆盖。除了操作末尾圆括号中额外的统计信息（下面的代码示例中加粗部分）之外，Oracle9i 及其后续版本中 STAT 条目的格式相同：

```
STAT #4 id=3 cnt=107 pid=2 pos=1 obj=16496 op='TABLE ACCESS FULL EMPLOYEES (cr=7
pr=0 pw=0 time=725 us)'
```

额外信息对于标识出执行计划中昂贵的行数据源很有用，它取决于 DBMS 的发行版本，在表 24-9 中总结如下：

表24-9 STAT执行统计数据

参 数	意 义
cr	一致性读
pr	物理读
pw	物理写
time	估计的运行时间，单位为微秒
cost	CBO计算的执行计划成本（需要Oracle11g）
size	估计的数据量，单位为字节（需要Oracle11g），该估计是基于对象统计数据（DBA_TABLES等）的，如果对象统计数据不可用则会使用来自数据段头部的信息
card	估计的基数，即被处理的行数（需要Oracle11g），该估计也基于对象统计数据

Oracle11g 的 STAT 条目有最详细的格式。

下面的例子（跟踪文件节选）展示了在一个 Oracle10g 跟踪文件中，一个 2 路联结的执行计划是怎样被 STAT 条目所表示的。

```
PARSING IN CURSOR #4 len=140 dep=0 uid=5 oct=3 lid=5 tim=105385553438 hv=782962817
ad='670e3cf4'
SELECT e.last_name, e.first_name, d.department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id=d.department_id
ORDER BY 1, 2
END OF STMT
...
STAT #4 id=1 cnt=106 pid=0 pos=1 obj=0 op='SORT ORDER BY (cr=115 pr=0 pw=0 time=5720
us)'
STAT #4 id=2 cnt=106 pid=1 pos=1 obj=0 op='NESTED LOOPS (cr=115 pr=0 pw=0 time=7302
us)'
STAT #4 id=3 cnt=107 pid=2 pos=1 obj=16496 op='TABLE ACCESS FULL EMPLOYEES (cr=7 pr=
0 pw=0 time=725 us)'
STAT #4 id=4 cnt=106 pid=2 pos=2 obj=16491 op='TABLE ACCESS BY INDEX ROWID DEPARTMEN
TS (cr=108 pr=0 pw=0 time=4336 us)'
STAT #4 id=5 cnt=106 pid=4 pos=1 obj=16492 op='INDEX UNIQUE SCAN DEPT_ID_PK (cr=2 pr
=0 pw=0 time=1687 us)'
```

在格式化执行计划时，TKPROF 保留参数 op 提供的额外信息。

```
Rows      Row Source Operation
-----
106  NESTED LOOPS (cr=223 pr=0 pw=0 time=166 us)
107  TABLE ACCESS FULL EMPLOYEES (cr=11 pr=0 pw=0 time=1568 us)
106  TABLE ACCESS BY INDEX ROWID DEPARTMENTS (cr=212 pr=0 pw=0 time=4747 us)
106  INDEX UNIQUE SCAN DEPT_ID_PK (cr=106 pr=0 pw=0 time=2151 us)(object id
51906)
```

该跟踪文件摘录展示了 Oracle11g 更为详细的格式：

```
STAT #4 id=1 cnt=106 pid=0 pos=1 obj=0 op='MERGE JOIN (cr=19 pr=0 pw=0 time=0 us cos
t=6 size=3604 card=106)'
```

```
STAT #4 id=2 cnt=27 pid=1 pos=1 obj=16967 op='TABLE ACCESS BY INDEX ROWID DEPARTMENT
```



```
S (cr=12 pr=0 pw=0 time=26 us cost=2 size=432 card=27)'
STAT #4 id=3 cnt=27 pid=2 pos=1 obj=16968 op='INDEX FULL SCAN DEPT_ID_PK' (cr=6 pr=0
pw=0 time=11 us cost=1 size=0 card=27)'
STAT #4 id=4 cnt=106 pid=1 pos=2 obj=0 op='SORT JOIN' (cr=7 pr=0 pw=0 time=24 us cost
=4 size=1926 card=107)'
STAT #4 id=5 cnt=108 pid=4 pos=1 obj=16970 op='TABLE ACCESS FULL EMPLOYEES' (cr=7 pr=
0 pw=0 time=4 us cost=3 size=1926 card=107)'
```

## 24.8 等待事件

捕捉等待事件对于解决由等待资源，比如磁盘访问、锁或 RAC 实例间通信等引起的性能问题是必不可少的。发行版本 Oracle10g 第一个具有用于跟踪等待事件并在文档中说明了的接口——PL/SQL 包 DBMS_MONITOR。

### 24.8.1 WAIT 条目的格式

启用 WAIT 条目的最小 SQL 跟踪级别为 8。在调查性能问题时不要使用在 8 以下的 SQL 跟踪级别。否则，等待事件对响应时间的影响就不可知，由此阻碍了对等待和非 CPU 消耗占绝大部分响应时间的情况下的性能问题进行诊断。

每一个等待事件都与 3 个以上提供事件更详细信息的参数相关联。许多等待事件在 *Oracle Database Reference* (例如, *Oracle Database Reference 10g Release 2* 的附录 C) 都有记录。可以通过查询视图 V\$EVENT_NAME 获取等待参数和等待事件的完整列表，如下所示：

```
SQL> SELECT name, parameter1, parameter2, parameter3 FROM v$event_name ORDER BY 1;
```

该查询结果中的一些示例行如下所示：

NAME	PARAMETER1	PARAMETER2	PARAMETER3
SQL*Net message from client	driver id	#bytes	
SQL*Net message from dblink	driver id	#bytes	
SQL*Net message to client	driver id	#bytes	
SQL*Net message to dblink	driver id	#bytes	
db file parallel write	requests	interrupt	timeout
db file scattered read	file#	block#	blocks
db file sequential read	file#	block#	blocks
direct path read	file number	first dba	block cnt
direct path write	file number	first dba	block cnt
enq: MR - contention	name mode	0 or file #	type
enq: ST - contention	name mode	0	0
enq: TM - contention	name mode	object #	table/partition
enq: TX - allocate ITL entry	name mode	usn<<16   slot	sequence
enq: TX - contention	name mode	usn<<16   slot	sequence
enq: TX - index contention	name mode	usn<<16   slot	sequence
enq: TX - row lock contention	name mode	usn<<16   slot	sequence
enq: UL - contention	name mode	id	0
enq: US - contention	name mode	undo segment #	0
latch free	address	number	tries

latch: gcs resource hash	address	number	tries
latch: ges resource hash list	address	number	tries
latch: library cache	address	number	tries
latch: library cache lock	address	number	tries
latch: library cache pin	address	number	tries
latch: redo allocation	address	number	tries
latch: redo copy	address	number	tries
latch: shared pool	address	number	tries

WAIT 条目的参数与数据字典或 V\$视图相关。例如，参数 file#和 file number 二者都对应于 V\$DATAFILE.FILE#，参数 object#指的是 DBA_OBJECTS.OBJECT_ID，以及门锁等待参数 address 对应于 V\$LATCH.ADDRESS。因为门锁以及入队等待事件名称在 Oracle10g 中更明确，通过一个等待事件的参数去剖析引用的入队或门锁的需求相比于 Oracle9i 就有大幅减少。

24.8.2 Oracle9i 中的 WAIT

Oracle9i 中 WAIT 条目参数一般被称作 p1、p2 以及 p3。这些参数的意义在 Oracle10g 中不能直接从名称的字面意识理解。相反，这些参数的意义必须使用文档或 V\$EVENT_NAME 从等待事件名称得到。Oracle9i 中 WAIT 条目的参数见表 24-10。

表24-10 Oracle9i中WAIT参数

参 数	意 义
ela	运行时间，单位微秒（在Oracle8i及更早的已经不支持的版本中为厘秒）
p1	等待事件的第一个参数。该参数的意义可以在V\$EVENT_NAME.PARAMETER1找到。它的值对应于V\$SESSION_WAIT.P1
p2	等待事件的第二个参数，与p1相似，即从V\$EVENT_NAME.PARAMETER2查看该参数的意义
p3	等待事件的第三个参数，与p1相似

下面这个例子是与游标 4 相关联的由单一数据块磁盘读取的 WAIT 条目：

```
WAIT #4: nam='db file sequential read' ela= 19045 p1=1 p2=19477 p3=1
```

在上面的等待事件中，p1 是数据文件编号（V\$DATAFILE.FILE#），p2 是数据文件中的数据块，p3 指定了数据块读取的编号。正如你即将看到的，Oracle9i 跟踪文件中的 wait 条目缺少 Oracle10g 中 wait 条目的时间戳（tim）。

24.8.3 Oracle10g 和 Oracle11g 中的 WAIT

Oracle10g 有 872 个等待事件（Oracle11g 有 959 个）。部分原因是因为 Oracle9i 中单个等待事件 enqueue 已经被对应于每种入队类别的 208 个单个等待事件取代。同样，Oracle10g 对于门锁有 27 个等待事件而 Oracle9i 仅有 2 个。

Oracle10g 和 Oracle11g 中 wait 条目用富有意义的参数名称取代了 Oracle9i 中的 p1、p2 和 p3 参数，并且也添加了时间戳。下面是一个例子：

```
WAIT #3: nam='db file scattered read' ela= 22652 file#=4 block#=253 blocks=4  
obj#=14996 tim=81592211996
```

Oracle10g 中存在 276 个不同的等待事件参数 (Oracle11g 为 293 个)。其中, 很大一部分参数具有相同意义但拼写不同, 比如 `retry count` 和 `retry_count`。另一个例子是参数 `obj#` 和 `object #` 以及 `object_id`。这些参数可以通过运行下面的查询得到:

```
SQL> SELECT PARAMETER1
FROM v$event_name
UNION
SELECT PARAMETER2
FROM v$event_name
UNION
SELECT PARAMETER3
FROM v$event_name
ORDER BY 1;
```

## 24.9 绑定变量

24

要获取最大数量的诊断数据, 应启动绑定变量跟踪。绑定变量的详细信息包括绑定变量的数据类型和值。如果没有这些信息, 就不可能找出一个索引是否由于被索引列的数据类型与绑定变量的数据类型不匹配而没有被使用, 例如一个被索引的 `DATE` 列和一个数据类型为 `TIMESTAMP` 的绑定变量。绑定数据类型不匹配可能同样会造成 CPU 使用率增加, 因为需要从一种数据类型转换到另一种。绑定变量值可用于在与捕获它们时完全相同的情况下调整查询。如果没有绑定变量值, 查询调整的示例值必须从表中检索。这是一个非常枯燥和耗时的过程。相比于在查找示例值上浪费时间, 还不如接受由绑定变量跟踪带来的额外文件大小和更高的度量误差。

### 24.9.1 BINDS 条目的格式

启用 BINDS 条目的最小 SQL 跟踪级别为 4。一个 BINDS 条目的结构由后面跟着游标编号的单词 BINDS 和每一个绑定变量单独的子部分 (`bind n` 或 `BIND #n`) 组成。

```
BINDS #m:
<subsection 0>
<details of bind variable 0>
...
<subsection n>
<details of bind variable n>
```

绑定变量在语句文本内从左到右从 0 开始编号。当将绑定变量与子部分相关联时, 不用关心数字, 它们会被包含在绑定变量的名称中 (例如: `B1`)。正如你将在后面的例子中看到的, 绑定变量名称: `B<n+1>` 可能出现在: `Bn` 前面, 其中 `n` 为整数。这实在是混乱。正确的关联是通过从左到右、从上到下读取语句文本形成的。Subsection 0 提供遇到的第一个绑定变量的详细信息, subsection 1 为第二个, 依此类推。

#### 1. Oracle9/中的绑定

下面的匿名 PL/SQL 块作为检测 BINDS 条目的例子:

```
DECLARE
sal number(8,2):=4999.99;
```

```

dname varchar2(64):='Shipping';
hired date:=to_date('31.12.1995','dd.mm.yyyy');
BEGIN
FOR emp_rec IN ( SELECT e.last_name, e.first_name, e.salary, d.department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id=d.department_id
AND e.salary > sal
AND e.hire_date > hired
AND d.department_name=dname) LOOP
null;
END LOOP;
END;
/

```

在SQL跟踪级别4或12跟踪该数据块得到包含BINDS部分的跟踪文件条目，如下所示：

```

PARSING IN CURSOR #2 len=209 dep=1 uid=0 oct=3 lid=0 tim=118435024029 hv=3341549851
ad='19cf2638'
SELECT e.last_name, e.first_name, e.salary, d.department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id=d.department_id
AND e.salary > :b3
AND e.hire_date > :b2
AND d.department_name=:b1
END OF STMT
PARSE #2:c=15625,e=75118,p=2,cr=150,cu=0,mis=1,r=0,dep=1,og=0,tim=118435024022
...
BINDS #2:
bind 0: dty=2 mxl=22(21) mal=00 scl=00 pre=00 oacflg=03 oacfl2=1 size=64 offset=0
bfp=05babee8 bln=22 avl=04 flg=05
value=4999.99
bind 1: dty=12 mxl=07(07) mal=00 scl=00 pre=00 oacflg=03 oacfl2=1 size=0 offset=24
bfp=05babf00 bln=07 avl=07 flg=01
value="12/31/1995 0:0:0"
bind 2: dty=1 mxl=32(08) mal=00 scl=00 pre=00 oacflg=03 oacfl2=1 size=0 offset=32
bfp=05babf08 bln=32 avl=08 flg=01
value="Shipping"

```

绑定变量值被打印为字符串而不是一种内部表示法。因此，很容易通过使用被捕获的绑定变量值重现被跟踪的语句。Subsection 0 (bind 0) 与绑定变量:b3 相关联。Subsection 1 (bind 1) 与绑定变量:b2 相关联。Oracle9i 使用 14 个参数表达绑定变量的详细信息。这些参数的详细解释见表 24-11。

表24-11 Oracle9i中的BIND参数

参 数	意 义
dty	数据类型代码
mxl	绑定变量值的最大长度（圆括号中的私有最大长度）
mal	数组长度

(续)

参 数	意 义
scl	比例
pre	精度
oacflg	特殊标志, 指示绑定选项
oacflg2	oacflg的第二个部分
size	给该chunk分配的内存总量
offset	该绑定缓冲区在chunk中的偏移
bfp	绑定地址
bln	绑定缓冲区长度
avl	实际值的长度
flg	绑定状态标志
value	该绑定变量的值

Oracle Database SQL Reference 手册包括一个将数字的数据类型代码翻译为对应的数据类型名称的表。这个表在该手册第 2 章的 Oracle Built-In Datatypes 部分列出了 Oracle9i 中实现的 21 种数据类型的 18 种不同类型的代码（国际字符集 NCHAR、NVARCHAR2 以及 NCLOB 分别与 CHAR、VARCHAR2 以及 CLOB 使用相同的类型代码）。表 24-12 总结了最为常用的数据类型代码和名称。

表24-12 数据类型代码和名称

数据类型代码	数据类型名称
1	VARCHAR2、NVARCHAR2
2	NUMBER
12	DATE
96	CHAR、NCHAR
112	CLOB、NCLOB
113	BLOB
180	TIMESTAMP

内置的 SQL 函数 DUMP (expr[.format[.position[.length]]) 可与列值的内部表示法一起显示该列的数据类型代码。DUMP 的第一个参数是用于显示内部表示法的格式。支持的格式有八进制 (format=8)、十进制 (默认值, format=10)、十六进制 (format=16) 以及独立的单字节字符 (format=17)。如果在前 3 个格式设置上加上 1000 (format=format+1000), 字符集的信息就被包含在内了。当参数 position 和 length 被忽略时, 就返回该条目的内部表示法。否则, 仅从偏移量为 position 的位置开始、长度为 length 字节的部分被考虑在内。下面是一个例子:

```
SQL> SELECT dump(employee_id, 16, 1, 1) empid_dmp,
dump(last_name, 1017,1,2) name_dmp, dump(hire_date, 10, 1, 1) date_dmp
FROM hr.employees
WHERE rownum=1;
EMPID_DMP      NAME_DMP      DATE_DMP
```

-----  
Typ=2 Len=3: c2 Typ=1 Len=8 CharacterSet=WE8MSWIN1252: 0,C Typ=12 Len=7: 119

正如我们所料，例子中 Typ 的值对应于表 24-12 中的数据类型代码。

2. Oracle10g和Oracle11g中的绑定

Oracle10g 和 Oracle11g 中 Bind#*n* 部分与 Oracle9i 中的 Bind#*n* 部分有所不同。Oracle10g 和 Oracle11g 使用 17 个参数表达绑定变量的详细信息。目前关于 3 个新的参数没有任何信息。一些出现在 Oracle9i 中的参数同样也被重命名了。考虑到这一点，表 24-13 列出了可用的信息。

表24-13 Oracle10g和Oracle11g中BINDS参数

参 数	意 义
oacdtty	数据类型代码
mxl	绑定变量值的最大长度（圆括号中的私有最大长度）
mxlc	未知
mal	数组长度
sc1	比例
pre	精度
oacflg	特殊标志，指示绑定选项
fl2	oacflg的第二部分
frm	未知
csi	数据库字符集或国际字符集的字符集标识符（见表24-14）
siz	给该chunk分配的内存总量
off	该绑定缓冲区在chunk中的偏移
kxsbbfbp	绑定地址
bln	绑定缓冲区长度
avl	实际值的长度
flg	绑定状态标志
value	绑定变量值

表 24-14 显示了一些字符集标识符（Character Set Identifier, CSI）参数的常见值。请注意，国际字符集从 Oracle9i 开始都一直是 Unicode 字符集。

表24-14 常见字符集和它们的标识符

CSI	字符集名称	描 述
1	US7ASCII	ASCII 7位美国
31	WE8ISO8859P1	ISO 8859-1 8位西欧
46	WE8ISO8859P15	ISO 8859-15 8位西欧
170	EE8MSWIN1250	MS Windows Code Page 1250 8位东欧
178	WE8MSWIN1252	MS Windows Code Page 1252 8位西欧
871	UTF8	Unicode 3.0 UTF-8通用字符集
873	AL32UTF8	Unicode 4.0 UTF-8通用字符集
2000	AL16UTF16	Unicode 4.0 UTF-16通用字符集

下面是一个 Oracle10g 中 BINDS 部分的例子。这是跟踪与在之前 Oracle9i 例子中所使用相同 SQL 语句的结果。

```
PARSING IN CURSOR #1 len=205 dep=1 uid=67 oct=3 lid=67 tim=8546016035 hv=3746754718
ad='2548145c'
SELECT E.LAST_NAME, E.FIRST_NAME, E.SALARY, D.DEPARTMENT_NAME
FROM HR.EMPLOYEES E, HR.DEPARTMENTS D
WHERE E.DEPARTMENT_ID=D.DEPARTMENT_ID
AND E.SALARY > :B3
AND E.HIRE_DATE > :B2
AND D.DEPARTMENT_NAME=:B1
END OF STMT
PARSE #1:c=0,e=96,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=1,tim=8546016029
BINDS #1:
kkscoacd
Bind#0
oacdty=02 mxl=22(21) mxlc=00 mal=00 scl=00 pre=00
oacflg=03 fl2=1206001 frm=00 csi=00 siz=160 off=0
kxsbbbf=07d9e508 bln=22 avl=04 flg=05
value=4999.99
Bind#1
oacdty=12 mxl=07(07) mxlc=00 mal=00 scl=00 pre=00
oacflg=03 fl2=1206001 frm=00 csi=00 siz=0 off=24
kxsbbbf=07d9e520 bln=07 avl=07 flg=01
value="12/31/1995 0:0:0"
Bind#2
oacdty=01 mxl=128(64) mxlc=00 mal=00 scl=00 pre=00
oacflg=03 fl2=1206001 frm=01 csi=178 siz=0 off=32
kxsbbbf=07d9e528 bln=128 avl=08 flg=01
value="Shipping"
```

Oracle10g 引入了两种额外的数据类型。它们是 BINARY_FLOAT 和 BINARY_DOUBLE，数据类型的代码（oacdty）分别为 21 和 22。

## 24.9.2 语句调优、执行计划以及绑定变量

当调整一个使用 SQL 跟踪捕获到的并包含绑定变量的语句时，不要使用常量代替绑定变量。如果这么做，优化器可能会做出不同的决定。相反，当在 SQL*Plus 中调整该语句时，声明 SQL*Plus 绑定变量匹配跟踪文件（参数 dty 或 oacdty）中绑定变量的数据类型。由于 SQL*Plus 变量并非支持所有的数据类型，例如没有提供 DATE 和 TIMESTAMP，你可能不得不采用一个匿名 PL/SQL 块去完整地复制数据类型。使用转换函数是另一种选择，例如当原始的数据类型不可用时，对一个类型为 VARCHAR2 的 SQL*Plus 变量使用 TO_DATE 或 TO_TIMESTAMP，但这同样可能影响优化器的计划选择。即使你完整地复制该绑定数据类型，你可能仍然得到一个与之前的执行不同的计划，因为之前的执行可能重用了一个基于不同绑定变量窥视值构建的计划。与此同时，如果该计划从共享池老化移出，你可能得到一个基于窥视当前绑定变量值的不同计划。因此，使用 AWR 快照或级别 6 的 Statspack 快照捕获计划是一个好方法。

SQL 语句 EXPLAIN PLAN 可用来给包含绑定变量的 SQL 语句生成一个执行计划。然而, EXPLAIN PLAN 对绑定变量数据类型和值一无所知。你甚至不需要声明 SQL*Plus 变量, 就可以在一个包含绑定变量的语句上顺利使用 EXPLAIN PLAN。

EXPLAIN PLAN 是极不可靠的, 它不应该被用于语句调优, 因为它在语句执行时所报告的执行计划经常同实际使用的计划不同。Oracle Database Performance Tuning Guide 10g Release 2 这样解释 EXPLAIN PLAN (19-4 页):

Oracle 不支持 EXPLAIN PLAN 对语句的日期绑定变量执行隐式类型转换。一般而言, 对于带有绑定变量的语句, EXPLAIN PLAN 的输出可能不能代表真正的执行计划。

SQL 跟踪文件、V\$SQL_PLAN、V\$SQL_PLAN_STATISTICS_ALL、AWR 以及 Statspack 的库表 STATS\$SQL_PLAN 都包含有 V\$SQL_PLAN 的快照, 是执行计划的可靠来源。需要提醒的是 V\$SQL_PLAN 可能对同一个 HASH_VALUE 或 SQL_ID 持有多个计划, 并且也很难搞清楚使用的是哪一个计划。为了避免这一缺陷, 可以在跟踪文件增长时运行 tail -f 显示该文件的尾部。STAT 条目中没有被格式化的执行计划阅读起来也是相当令人头疼的。Oracle 10g 中的管道表函数 DBMS_XPLAN.DISPLAY_CURSOR 拥有迄今为止最为成熟的解决方案。它的语法如下:

```
dbms_xplan.display_cursor(
  sql_id IN VARCHAR2 DEFAULT NULL,
  child_number IN NUMBER DEFAULT NULL,
  format IN VARCHAR2 DEFAULT 'TYPICAL')
RETURN dbms_xplan_type_table PIPELINED;
```

如果调用 DBMS_XPLAN.DISPLAY_CURSOR 时没有传递 sql_id 和 child_number, 该会话执行的上一个游标的计划就会被显示出来, 使得该函数成为 SQL*Plus AUTOTRACE 的理想替代品。格式参数可以精确控制输出。可以通过使用格式 'ALL ALLSTATS LAST' 获取仅仅与一条语句上一次执行相关的最详细输出。要收集内存和 I/O 统计信息, 必须在会话级别设置 STATISTICS_LEVEL=ALL。在接下来的例子中展示的列 A-Rows (实际行)、A-Time (实际时间)、Buffers 以及 Reads 在默认设置为 STATISTICS_LEVEL=TYPICAL 时不会显示。随后的例子说明了以下几点。

- ❑ 刷新缓冲区缓存以引发磁盘读取, 这将会影响使用 DBMS_XPLAN.DISPLAY_CURSOR 所生成的执行计划的 Buffers 列所报告的值。
- ❑ 在 SQL 语句中使用 SQL*Plus 变量作为绑定变量。
- ❑ 优化器环境的作用, 特别是执行计划的参数 OPTIMIZER_INDEX_COST_ADJ。
- ❑ 怎样同一个执行计划一起检索过滤谓词和访问谓词。WHERE 子句中的列被称为谓词。
- ❑ 怎样检索查询数据块名称, 这将会对使用优化器提示进行调优很有用。

该示例包括运行一个使用不同优化器参数设置的表 EMPLOYEES 和 DEPARTMENTS 的 2 路连接。第一次迭代在默认优化器环境运行并导致一个嵌套循环连接, DEPARTMENTS 全面扫描和一个到 EMPLOYEES 的索引访问。更小的表 DEPARTMENTS 在该连接中被用作驱动表。

对于第二次迭代, OPTIMIZER_INDEX_COST_ADJ 被设置为它的最大值 10 000, 通过标记具有较高成本的索引访问引发全表扫描。让我们一步一步来看这个示例。首先, 缓冲区缓存已经被刷



新^①并且 STATISTICS_LEVEL 也已经被改变。

```
SQL> ALTER SYSTEM FLUSH BUFFER_CACHE;
System altered.
SQL> ALTER SESSION SET statistics_level=all;
Session altered.
```

下一步，我们声明 3 个 SQL*Plus 绑定变量，来模仿在上一节的节选中我们在 SQL 跟踪文件中看到绑定变量:B1、:B2 以及 :B3，并指定跟踪文件中的绑定变量值。因为 SQL*Plus 不支持数据类型为 DATE 的变量，VARCHAR2 用于变量 HIRED，并且函数 TO_DATE 被应用于将 VARCHAR2 转换为 DATE。

```
SQL> VARIABLE sal NUMBER
SQL> VARIABLE hired VARCHAR2(10)
SQL> VARIABLE dname VARCHAR2(64)
SQL> EXEC :sal :=4999.99; :hired:='31.12.1995'; :dname:='Shipping'
PL/SQL procedure successfully completed.
```

除了使用比:Bn 更具可读性的绑定变量名称以及 TO_DATE 之外，同样也使用了跟踪文件中的语句文本。

```
SQL> SELECT e.last_name, e.first_name, e.salary, d.department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id=d.department_id
AND e.salary > :sal
AND e.hire_date > to_date(:hired, 'dd.mm.yy')
AND d.department_name=:dname;
```

LAST_NAME	FIRST_NAME	SALARY	DEPARTMENT_NAME
Weiss	Matthew	8000	Shipping
Fripp	Adam	8200	Shipping
Vollman	Shanta	6500	Shipping
Mourgos	Kevin	5800	Shipping

下一步，调用 DBMS_XPLAN 检索上一次执行语句的执行计划。我将执行计划输出分为两部分，使之更具可读性。列 E-Time、E-Bytes 和 E- Rows 中的大写字母 E 是 Estimated 的缩写。请注意，下面是第一次迭代对子游标编号 0 的报告。

```
SQL> SELECT *
FROM table (DBMS_XPLAN.DISPLAY_CURSOR(null, null, 'ALL ALLSTATS LAST'));

PLAN_TABLE_OUTPUT
-----
SQL_ID b70r97ta66g1j, child number 0
-----
SELECT e.last_name, e.first_name, e.salary, d.department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id=d.department_id
AND e.salary > :sal
```

① 刷新缓冲区缓存至少需要版本 Oracle10g。

```
AND e.hire_date > to_date(:hired, 'dd.mm.yy')
AND d.department_name=:dname
```

Plan hash value: 2912831499

Id	Operation	Name	Starts	E-Rows
* 1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	3
2	NESTED LOOPS		1	3
* 3	TABLE ACCESS FULL	DEPARTMENTS	1	1
* 4	INDEX RANGE SCAN	EMP_DEPARTMENT_IX	1	10

E-Bytes	Cost (%CPU)	E-Time	A-Rows	A-Time	Buffers	Reads
90	1 (0)	00:00:01	4	00:00:00.03	13	15
138	4 (0)	00:00:01	47	00:00:00.02	10	7
16	3 (0)	00:00:01	1	00:00:00.01	8	6
	0 (0)		45	00:00:00.01	2	1

Query Block Name / Object Alias (identified by operation id):

```
1 - SEL$1 / E@SEL$1
3 - SEL$1 / D@SEL$1
4 - SEL$1 / E@SEL$1
```

Predicate Information (identified by operation id):

```
1 - filter(("E"."SALARY">:SAL AND "E"."HIRE_DATE">TO_DATE(:HIRED,'dd.mm.yy'))
3 - filter("D"."DEPARTMENT_NAME"=:DNAME)
4 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
```

Column Projection Information (identified by operation id):

```
1 - "E"."FIRST_NAME"[VARCHAR2,20], "E"."LAST_NAME"[VARCHAR2,25], "E"."SALARY"[NUMBER,22]
2 - "D"."DEPARTMENT_NAME"[VARCHAR2,30], "E".ROWID[ROWID,10]
3 - "D"."DEPARTMENT_ID"[NUMBER,22], "D"."DEPARTMENT_NAME"[VARCHAR2,30]
4 - "E".ROWID[ROWID,10]
```

通过对执行计划中每一步进行查询，投影信息列出了列的子集。“投影”（projection）这个词是一个来自关系代数的学术名词。对于第二次迭代，让我们改变优化器环境然后看一下这是怎么影响该执行计划的。

```
SQL> ALTER SESSION SET optimizer_index_cost_adj=10000;
Session altered.
SQL> SELECT e.last_name, e.first_name, e.salary, d.department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id=d.department_id
AND e.salary > :sal
AND e.hire_date > to_date(:hired, 'dd.mm.yy')
AND d.department_name=:dname;
```

LAST_NAME	FIRST_NAME	SALARY	DEPARTMENT_NAME
Weiss	Matthew	8000	Shipping
Fripp	Adam	8200	Shipping
Vollman	Shanta	6500	Shipping
Mourgos	Kevin	5800	Shipping

请注意在下面的代码示例中报告的子游标编号为 1。这是由于不同的计划，它产生于一个被改变了的优化器环境。该计划的散列值同样改变了，而 SQL_ID 一直保持不变。文档中没有说明哪一个参数的改变迫使优化器考虑了一个新的计划。当对一个语句调优时，在每次执行前给语句文本添加独特的注释是很明智的。这迫使其为硬解析并确认优化器考虑当前环境的每个方面，其中可能包括更新了的对象、系统统计数据以及被修改了的初始化参数。

由于缓存，该语句被第二次运行时就会产生一次单独的磁盘读取。这次，我将执行计划分为 3 部分以方便阅读，因为来自于散列连接的内存使用统计数据让它变得更宽。

```
SQL> SELECT *
FROM table (DBMS_XPLAN.DISPLAY_CURSOR(null, null, 'ALL ALLSTATS LAST'));
PLAN_TABLE_OUTPUT
```

```
SQL_ID b70r97ta66g1j, child number 1

SELECT e.last_name, e.first_name, e.salary, d.department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id=d.department_id
AND e.salary > :sal
AND e.hire_date > to_date(:hired, 'dd.mm.yy')
AND d.department_name=:dname
```

Plan hash value: 2052257371

Id	Operation	Name	Starts	E-Rows	E-Bytes
* 1	HASH JOIN		1	3	138
* 2	TABLE ACCESS FULL	DEPARTMENTS	1	1	16
* 3	TABLE ACCESS FULL	EMPLOYEES	1	31	930

Cost (%CPU)	E-Time	A-Rows	A-Time
7 (15)	00:00:01	4	00:00:00.01
3 (0)	00:00:01	1	00:00:00.01
3 (0)	00:00:01	45	00:00:00.01

Buffers	Reads	OMem	1Mem	Used-Mem
12	1	887K	887K	267K (0)
7	0			
5	1			

```
-----
Query Block Name / Object Alias (identified by operation id):
-----
```

```
1 - SEL$1
2 - SEL$1 / D@SEL$1
3 - SEL$1 / E@SEL$1
```

```
Predicate Information (identified by operation id):
-----
```

```
1 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
2 - filter("D"."DEPARTMENT_NAME"=:DNAME)
3 - filter(("E"."SALARY">:SAL AND "E"."HIRE_DATE">TO_DATE(:HIRED,'dd.mm.yy')))
```

```
Column Projection Information (identified by operation id):
-----
```

```
1 - (#keys=1) "D"."DEPARTMENT_NAME"[VARCHAR2,30], "E"."FIRST_NAME"[VARCHAR2,20], "E"
  ."LAST_NAME"[VARCHAR2,25], "E"."SALARY"[NUMBER,22]
2 - "D"."DEPARTMENT_ID"[NUMBER,22], "D"."DEPARTMENT_NAME"[VARCHAR2,30]
3 - "E"."FIRST_NAME"[VARCHAR2,20], "E"."LAST_NAME"[VARCHAR2,25], "E"."SALARY"[NUMBER
  ,22], "E"."DEPARTMENT_ID"[NUMBER,22]
```

通过在级别 4 或 12 跟踪 SQL*Plus 会话，证明除了数据类型为 DATE(oacdtty=12)的绑定变量外，绑定变量（参数 oacdtty）的数据类型匹配原始跟踪文件中的数据类型，数据类型为 DATE 的绑定变量不能使用一个 SQL*Plus 变量复制。以下是一个 Oracle10g 跟踪文件的相关部分：

```
BINDS #3:
kkscoacd
Bind#0
oacdtty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=03 fl2=1000000 frm=00 csi=00 siz=184 off=0
kxsbbbf=04d99cb0 bln=22 avl=04 flg=05
value=4999.99
Bind#1
oacdtty=01 mxl=32(10) mxlc=00 mal=00 scl=00 pre=00
oacflg=03 fl2=1000000 frm=01 csi=178 siz=0 off=24
kxsbbbf=04d99cc8 bln=32 avl=10 flg=01
value="31.12.1995"
Bind#2
oacdtty=01 mxl=128(64) mxlc=00 mal=00 scl=00 pre=00
oacflg=03 fl2=1000000 frm=01 csi=178 siz=0 off=56
kxsbbbf=04d99ce8 bln=128 avl=08 flg=01
value="Shipping"
```

这个例子指出了通过复制与绑定变量的值和数据类型尽可能相近的语句，如何利用来自跟踪文件中 BINDS 条目的信息。这种对于跟踪语句的精确复制是调优的最佳出发点。

为了使用 DISPLAY_CURSOR 功能，该调用用户必须具有对 V\$SQL、V\$SQL_PLAN 以及 V\$SQL_PLAN_STATISTICS_ALL 的 SELECT 特权。可以授予角色 SELECT_CATALOG_ROLE，确保这些特权都可用。前面的示例展示了不带参数调用 DBMS_XPLAN.DISPLAY_CURSOR 检索前一条来自 V\$SQL_PLAN 的语句的计划，即使在单个语句存在多个执行计划（即一些子游标）的情况下。该功能在 Oracle9i 中不能

被复制，因为列 `V$SESSION.PREV_CHILD_NUMBER`^①不可用。

即使在 Oracle10g 中，SQL*Plus AUTOTRACE 使用 EXPLAIN PLAN，它同样得承受之前提到的 EXPLAIN PLAN 的缺陷。这同样适用于 TKPROF 选项 EXPLAIN=user/password，它也运行 EXPLAIN PLAN，这时即使来自于不同数据库会话生成的跟踪文件都被提供给 TKPROF，这样得到错误结果的概率反而更大了。当然，你永远不应该使用这个 TKPROF 开关，而是让 TKPROF 格式化跟踪文件中的 STAT 行。如果在跟踪被激活时游标没有被关闭，那么跟踪文件中不会有任何那个特殊游标的 STAT 行。在这种情况下，你需要查询 `V$SQL_PLAN`（使用 Oracle10g 中的 `DBMS_XPLAN`），但仅在这个争论中的语句仍然被缓存时才会成功。如果被争论的语句不再被缓存，使用脚本 `$ORACLE_HOME/rdbms/admin/sprepsql.sql` 访问 Statspack 库或使用 `DBMS_XPLAN.DISPLAY_AWR` 访问 AWR。

## 24.10 跟踪文件条目其他项

24

其他项包括其他条目记录哪一个会话、模块或动作生成跟踪文件条目。这些条目中有一些是被自动写入，而其他的则需要应用程序编码。

### 24.10.1 会话标识符

会话标识符 (Session Identification) 始终被发送到 SQL 跟踪文件。相比于模块或动作标识符，它不需要任何应用程序编码。会话标识符的格式在 Oracle9i 和 Oracle10g 中是相同的，这样 Oracle10g 的 TRCSESS 工具可以用于从多个 Oracle9i 或 Oracle10g 共享服务器跟踪文件中给单个会话提取跟踪信息。

无论使用哪种服务器模型（专用或共享），每一个会话通过实例生命周期中的两个数字的组合被唯一标识。也就是说，不会重用相同的数字组合，除非该 DBMS 实例被关闭然后重启。将这个与 `V$SESSION.AUDSID` 相比，后者来源于序列 `SYS.AUDSES$`，被用于审计用途，并且可以通过调用 `USERENV('SESSIONID')` 访问。谈到 `USERENV`，Oracle10g 及其后续版本终于为非特权用户提供了对 `V$SESSION.SID` 的访问，这是通过函数 `USERENV` 中没有说明的选项 `SID` 即 `USERENV('SID')`^②来完成的。在这种上下文中，非特权是指对 `V$` 视图的访问没有被授予，例如通过 `SELECT_CATALOG_ROLE`。

顺便说一句，最适当的一般方法是弄明白 Oracle9i 中 `SID` 是用于运行 `SELECT sid FROM v$mystat WHERE ROWNUM=1` 的。在这里，“一般”是指这对非特权会话以及使用 `SYSDBA` 和 `SYSOPER` 特权的会话都适用。查询 `SELECT sid FROM v$session WHERE audsid = userenv('sessionid')` 并不适用于获取特权会话的 `SID`，因为这些没有被指定一个唯一的审计会话标识符 (`V$SESSION.AUDSID`)。特权会话在 Oracle9i 中 `AUDSID=0`，在 Oracle10g 中 `AUDSID=4294967295`。这一事实在 Oracle9i 和 Oracle10g *Database Reference* 中并未提及。

① 在 Oracle9i，底层的 `X$KSUSE` 同样不持有前一语句的子游标编号。

② 文档中记录的替代方案是 `SELECT sys_context('USERENV', 'SID') FROM dual`。

该会话 ID 的第一个数字是会话标识符，而第二个是会话序列号。前者可以在 V\$SESSION.SID 找到，而后者可以通过 V\$SESSION.SERIAL# 访问并且每次重用 SID 时都会递增。该条目的格式如下所示：

```
*** SESSION ID:(sid.serial#) YYYY-MM-DD HH24:MI:SS.FF3
```

该行结束位置的时间戳使用 ORACLE 日期格式模型（见 *Oracle Database SQL Reference 10g Release 2*, 2-58 页）描述。FF3 代表秒的三个小数位。一个实际的条目如下所示，其会话 V\$SESSION.SID=147, V\$SESSION.SERIAL#=40, 并且在 2007 年 2 月 6 日被写入：

```
*** SESSION ID:(147.40) 2007-02-06 15:53:20.844
```

### 24.10.2 服务名称 ID

服务名称 ID 总是被发送到 Oracle10g 和 Oracle11g 的 SQL 跟踪文件。Oracle9i 不具有此特性。跟踪文件中的服务名称是指实例服务名称。不要将这与在 tnsnames.ora 中定义的 Oracle Net 服务名称或目录服务相混淆（见本书前言“实例服务名称和网络服务名称的对比”对模棱两可的定义）。一个会话的服务名称使用 bequeath 适配器建立——这类会话总是与 DBMS 实例运行在同一台机器上，不通过监听器，并且需要设置 ORACLE_SID 为该实例启动时使用的值——SYSSUSERS。

这类会话在连接字符串中不提供 Oracle Net 服务名称。连接字符串中仅仅包含用户名和密码，如下例所示：

```
$ sqlplus ndebes/secret
```

通过监听器建立的会话，需要在连接字符串中指定 Oracle Net 服务名称，该字符串可以包含任何该监听器已知的服务名称，所提供的 Net 服务名称定义包含对 SERVICE_NAME=instance_service_name 而不是 SID=oracle_sid 的指定。该条目的格式如下：

```
*** SERVICE NAME:(instance_service_name) YYYY-MM-DD HH24:MI:SS.FF3
```

Instance_service_name 可以是 SYSSUSERS、SYSSBACKGROUND（跟踪来自于一个后台进程）或监听服务于特定实例的任何已知实例服务名称三者中的任何一个。下面是一些例子：

```
*** SERVICE NAME:(SYSSUSERS) 2007-06-12 08:43:24.241
```

```
*** SERVICE NAME:(TEN.world) 2007-06-13 17:38:55.289
```

对于实例的强制性后台进程，如 CKPT（校验点）、SMON（系统监控）和 LGWR（日志写入器），它们的服务名称是一个空字符串，例如在这个例子中：

```
$ grep "SERVICE NAME" ten_lgwr_3072.trc
```

```
*** SERVICE NAME:() 2007-06-13 17:37:04.830
```

当使用已封装过程 DBMS_SERVICE.CREATE_SERVICE 创建服务时，参数 network_name（不是 service_name）的值被监听器注册为一个实例服务名称（要求至少为 Oracle10g）。因此，在 tnsnames.ora 中 network_name 需要被用于 Net 服务名称定义，并会在 SERVICE NAME 条目中出现。这同样对使用 DBCA 或 srvctl 创建的 RAC 集群数据库服务适用，因为这些都是基于包 DBMS_SERVICE 的功能。

### 24.10.3 应用程序插桩

名词应用程序插桩 (instrumentation) 是指一种编程技术, 即一个程序具有产生它自身执行时间记录的能力。ORACLE DBMS 具有很好的插桩性 (等待事件、计数器)。然而, 当一个数据库客户端通知 DBMS 它所执行的任务时, 可以更大程度地利用该插桩。本节讨论与应用程序插桩相关的跟踪文件条目。这些条目的格式从 Oracle9i 到 Oracle10g 已发生了很大变化, 所以相关材料取决于具体的发行版本。启用本节讨论的条目的最小 SQL 跟踪级别为 1。

#### 1. Oracle10g与Oracle11g中的应用程序插桩条目

表 24-15 使用 PL/SQL 和 OCI 接口生成了下面的列表, 并按字母顺序列出了 Oracle10g 和 Oracle11g 中的插桩条目。请注意, Oracle JDBC 驱动的 Java 插桩接口比通过 Java 调用 PL/SQL (见第 23 章) 更高效。在最低的级别, 应用程序插桩使用 Oracle OCI 函数 OCIAttrSet 实现 (见 *Oracle Call Interface Programmer's Guide*)。

24

表24-15 插桩条目的PL/SQL和OCI接口

跟踪文件条目	PL/SQL接口	OCIAttrSet属性
ACTION NAME	DBMS_APPLICATION_INFO.SET_MODULE DBMS_APPLICATION_INFO.SET_ACTION	OCI_ATTR_ACTION
CLIENT ID	DBMS_SESSION.SET_IDENTIFIER	OCI_ATTR_CLIENT_IDENTIFIER ^①
MODULE NAME	DBMS_APPLICATION_INFO.SET_MODULE	OCI_ATTR_MODULE

当在 SQL*Plus 中运行如下的代码时, 插桩条目的所有三种类型都被写入一个跟踪文件:

```
C:> sqlplus ndebes/secret@ten_g.oradbpro.com
Connected.
SQL> BEGIN
dbms_application_info.set_module('mod', 'act');
dbms_session.set_identifier(sys_context('userenv','os_user') ||
'@' || sys_context('userenv','host') || ' (' ||
sys_context('userenv','ip_address') || ') ');
END;
/
PL/SQL procedure successfully completed.
SQL> ALTER SESSION SET sql_trace=TRUE;
Session altered.
```

产生的跟踪文件包含如下的这些行:

```
*** ACTION NAME:(act) 2007-08-31 18:02:26.578
*** MODULE NAME:(mod) 2007-08-31 18:02:26.578
*** SERVICE NAME:(orcl.oradbpro.com) 2007-08-31 18:02:26.578
*** CLIENT ID:(DBSERVER\ndebes@WORKGROUP\DBSERVER (192.168.10.1)) 2007-08-31 18:02:26.578
*** SESSION ID:(149.21) 2007-08-31 18:02:26.578
```

① DBMS_APPLICATION_INFO.SET_CLIENT_INFO 和 OCI 属性 OCI_ATTR_CLIENT_INFO 设置 V\$SESSION.CLIENT_INFO。这个设置不会被发送到跟踪文件, 不能与包 DBMS_MONITOR 一起使用。

SERVICE NAME 的值 `orcl.oradbpro.com` 源自在对 Net 服务名称 `ten_g.oradbpro.com` 的定义中，将字符串 `orcl.oradbpro.com` 作为 SERVICE_NAME 使用。

这些就是 Oracle10g TRCSESS 工具被用于从一个或多个跟踪文件中提取相关部分时，所搜索的跟踪文件条目类型。下面的章节为个别条目提供了更多细节信息。有关 TRCSESS 的详细信息见第 23 章。

## 2. 模块名称

模块名称是为了给 DBMS 传递应用程序的名称或更大的模块。默认设置为 NULL。SQL*Plus 和 Perl DBI 自动设置一个模块名称。下面的例子来自于一个 SQL*Plus 会话：

```
*** MODULE NAME:(SQL*Plus) 2007-02-06 15:53:20.844
```

## 3. 动作名称

动作名称代表代码或子例程的一个更小的单元。一个模块可能调用多个子例程，而每一个子例程设置一个不同的动作名称。默认设置 NULL 会使动作名称的长度为 0。

```
*** ACTION NAME:() 2007-02-06 15:53:20.844
```

## 4. 客户端标识符

在三层架构环境中的性能问题或挂起问题可能会极其难以追查，其中应用程序使用中间应用服务层维护数据库连接池。由于连接池位于中间层，性能分析师查看 V\$视图或扩展 SQL 跟踪文件，但是并不能在报告数据库很缓慢的应用程序用户同服务于特定用户的 ORACLE 实例中的数据库会话或服务进程之间建立联系。没有办法为正在抱怨的最终用户找出哪一条 SQL 语句正在运行，除非该应用程序很好地被插桩了，而这是在我的 DBA 职业生涯中一直都没有使用的方法。

客户端标识符是解决这一难题的答案。包 DBMS_SESSION 提供了一种方法，使得应用程序与 DBMS 之间使用唯一指派了应用程序用户的标识符通信。这个标识符成为列 V\$SESSION.CLIENT_IDENTIFIER 的值。如果启用了 SQL 跟踪，相同的标识符同样会被嵌入到 SQL 跟踪文件格式如下：

```
*** CLIENT ID:(client_identifier) YYYY-MM-DD HH24:MI:SS.FF3
```

Client_identifier 是客户端标识符，通过在应用程序代码中调用过程 DBMS_SESSION.SET_IDENTIFIER 去设置。为了从一个或多个 SQL 跟踪文件提取某一个客户端标识符的跟踪信息，可以使用 `trcsess clientid=client_identifier`。下面的行显示了来自一个跟踪文件的实际条目，使用的客户端标识符为 ND，该条目于 2007 年 2 月 6 日被写入。

```
*** CLIENT ID:(ND) 2007-02-06 15:53:20.844
```

一个客户端标识符的最大长度是 64 字节。超过此长度的字符串会被自动截断。当使用 DBMS_SESSION 作为插桩来监控应用程序时，认为过程 DBMS_SESSION.CLEAR_IDENTIFIER 不向跟踪文件写入 CLIENT ID 条目，保持该客户端标识符有效直到使用 DBMS_SESSION.SET_IDENTIFIER 改变了它。当使用了连接池时，这可能会导致与不同客户端标识符的相关部分在跟踪文件中不能划定。该解决方案包括通过向封装过程 DBMS_SESSION.SET_IDENTIFIER 传递 NULL 设置一个空的客户端标识



符，而不是通过调用过程 DBMS_SESSION.CLEAR_IDENTIFIER 实现。

5. Oracle9i中的应用程序插桩条目

在 Oracle9i 上运行与 24.10.3 中第 1 小节中相同的插桩代码,会造成与 Oracle10g 和 Oracle11g 相比被写入到跟踪文件中的条目少很多。实例服务名称既不可以通过 V\$SESSION 访问,也不可以在跟踪文件中找到。Oracle9i 跟踪文件包含如下所示的行:

```
*** SESSION ID:(10.697) 2007-08-31 18:19:10.000
APPNAME mod='mod' mh=781691722 act='act' ah=3947624709
```

很显然,在 Oracle9i 中模块和动作的格式不相同。模块和动作通常作为一个单独的行被记录在关键词 APPNAME 后面,即使是只使用 DBMS_APPLICATION_INFO.SET_ACTION 对动作进行了设置。对 APPNAME 条目中使用的缩写的解释见表 24-16。

表24-16 Oracle9i APPNAME参数

参 数	意 义
mod	模块, 对应于V\$SESSION.MODULE
mh	模块散列值, 对应于V\$SESSION.MODULE_HASH
act	动作, 对应于V\$SESSION.ACTION
ah	在Oracle9i中, 客户端标识符没有被写入到跟踪文件。它仅在V\$SESSION.CLIENT_IDENTIFIER ^① 中被设置

```
SQL> SELECT client_identifier FROM v$session WHERE sid=10;
CLIENT_IDENTIFIER
-----
ndebbs@WORKGROUP\DBSERVER
```

24.10.4 ERROR 条目的格式

在 SQL 语句执行期间的错误通过 ERROR 条目进行标记。这些可能会在那些被成功解析、但不能成功执行的语句身上发生。启用 ERROR 条目的最小 SQL 跟踪级别为 1。下面是一个例子:

```
PARSING IN CURSOR #6 len=94 dep=0 uid=30 oct=2 lid=30 tim=171868250869 hv=3526281696
ad='6778c420'
INSERT INTO poem (author, text) VALUES(:author, empty_clob())
RETURNING ROWID INTO :row_id
END OF STMT
PARSE #6:c=0,e=150,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=171868250857
EXEC #6:c=10014,e=52827,p=0,cr=2,cu=5,mis=0,r=0,dep=0,og=1,tim=171868303859
ERROR #6:err=372 tim=17186458
```

#后面的数字是失败语句的游标编号。前面的示例中 INSERT 语句失败的原因是 ORA-00372: file 4 cannot be modified at this time。包含表 POEM 的表空间的状态为只读。ERROR 条目的参数见表 24-17。

① 奇怪的是,来自于 Oracle 9.2.0.1.0 实例使用的客户端标识符的 IP 地址会被自动截断。

表24-17   ERROR参数

参   数	意   义
.err	错误编号
tim	时间戳，单位厘秒（在Oracle9i中好像一直都是0，但在Oracle10g中具有有意义的值）

有时应用程序不报告 ORACLE 错误编号或错误信息。在这种情况下，非常值得执行级别 1 的 SQL 跟踪去发现 DBMS 抛出的是哪一个错误。它甚至可以通过查看跟踪文件中的时间戳和 tim 值找出什么时候发生了错误。每一个跟踪文件头部附近就是时间戳，如下所示：

```
*** SESSION ID:(150.524) 2007-06-22 15:42:41.018
```

在非运行期间，DBMS 自动使用如下格式将额外的时间戳写入跟踪文件：

```
*** 2007-06-22 15:42:51.924
```

间歇性时间戳可以通过调用 DBMS_SYSTEM.KSDDT 强制写入。运行如下的脚本将会生成两个相差 10 秒的错误，因为调用了 DBMS_LOCK.SLEEP：

```
SQL> ALTER SESSION SET sql_trace=TRUE;
SQL> ALTER TABLESPACE users READ WRITE /* is read write, will fail */;
SQL> EXEC dbms_lock.sleep(10) /* sleep for 10 seconds */
SQL> EXEC dbms_system.ksddt /* write timestamp to trace file */
SQL> ALTER TABLESPACE users READ WRITE /* is read write, will fail */;
```

由此产生的 Oracle10g 跟踪文件如下所示（摘录）：

```
1 *** SESSION ID:(150.524) 2007-06-22 15:42:41.018
2 PARSING IN CURSOR #1 len=55 dep=1 uid=0 oct=3 lid=0 tim=176338392169 hv=1950821498
  ad='6784bfac'
...
3 EXEC #1:c=20028,e=208172,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=176339220462
4 ERROR #1:err=1646 tim=17633550
...
5 PARSING IN CURSOR #1 len=33 dep=0 uid=0 oct=47 lid=0 tim=176339264800
  hv=2252395675
  ad='67505814'
6 BEGIN dbms_lock.sleep(10); END;
7 END OF STMT
8 PARSE #1:c=0,e=140,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=176339264790
9 *** 2007-06-22 15:42:51.924
10 EXEC #1:c=0,e=9997915,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=1,tim=176349294618
...
11 EXEC #1:c=70101,e=74082,p=0,cr=2,cu=0,mis=0,r=0,dep=0,og=1,tim=176349582606
12 ERROR #1:err=1646 tim=17634587
```

头部时间戳后面第一个 tim 值将作为参考。在该例子中，第 1 行中的 2007-06-22 15:42:41.018 大约与第 2 行中 176 338 392 169 微秒相同。第 4 行 ERROR 条目上面最近的 tim 值为第 3 行的 176 339 220 462 微秒。因此在第 2 行与第 3 行之间，已经过了(176 339 220 462－176 338 392 169)/1 000 000 也就是 0.82 秒。第 2 个 ERROR 条目上面最近的 tim 值是第 11 行的 176 349 582 606。在第 3 行与第 11 行之间，已经过了(176 349 582 606－176 339 220 462)/1 000 000 也就是 10.36 秒。这是合理的，

因为该会话睡眠了 10 秒。第 9 行强制使用 DBMS_SYSTEM.KSDDT 的间歇性时间戳证实了这一点。ERROR 条目时间戳的值仅为厘秒精度。这很明显，当从 17 634 587 (第 12 行的 ERROR 条目) 减去 17 633 550 (第 4 行的 ERROR 条目) 后得到的结果为 10.37 秒  $((17\ 634\ 587 - 17\ 633\ 550) / 100)$ 。

使用相同的方法，我们可以计算位于第 12 行的第 2 个错误在 15:42:41.018 之后的  $(176\ 349\ 582\ 606 - 176\ 338\ 392\ 169) / 1\ 000\ 000$  也就是 11.190 秒发生。因此第 2 个错误发生在 15:42:52.208，这大概是在跟踪文件头部被写入 10 秒之后，也大概是 DBMS_SYSTEM.KSDDT 写入时间戳之后的 0.3 秒。当然，这种方法对于所有具有 tim 字段的跟踪文件条目都有效，而不仅仅是 ERROR 条目。为什么在我的测试中，ERROR 条目厘秒精度的 tim 值和其他条目中微秒精度的 tim 值之间会有大概 3.7 秒的偏移？这是 Oracle 开发中存在的问题。

我记得一次由升级服务请求引发的问题。大家都认为 DBMS 错误地抛出了 ORA-06502:PL/SQL: numeric or value error。通过在级别 1 进行跟踪然后查看 ERROR 条目，我可以证明在 DBMS 实例和客户端之间的通信不存在问题。原来，这是一个由缓冲区太小引发的应用程序编码错误。是由 Oracle Forms 运行时环境中的 PL/SQL 引擎抛出的 ORA-06502 错误，而不是由 ORACLE DBMS 中的 PL/SQL 引擎抛出。因此，在 SQL 跟踪文件中就没有了 ERROR 条目。

### 24.10.5 应用程序插桩与并行执行进程

在 3 种应用程序插桩条目中，只有来自并行执行进程的客户端标识符被发送到跟踪文件。这些跟踪文件在参数 BACKGROUND_DUMP_DEST 指定的目录中被创建，并遵照 Oracle10g 中的命名规则 ORACLE_SID_pnnn_spid.trc，其中 *n* 是介于 0~9 之间的数字，并且 spid 对应于 V\$PROCESS.SPID。

术语“查询协调器”(query coordinator)是指一个控制并行执行 slave 的进程。slave 执行实际的工作。查询协调器是服务于数据库客户端的同样的进程。因为资源消耗(比如并行执行 slave 的 CPU 时间和 I/O 请求)不被纳入到查询协调器的统计报告中，客户端标识符可与 TRCSESS 工具结合使用，得到查询协调器消耗的资源以及它所管理的并行执行进程的完整明细。

随后的例子说明如何使用 TRCSESS 工具将来自查询协调器的跟踪文件以及来自并行执行进程的 4 个跟踪文件合并为一个跟踪文件，并使用 TKPROF 格式化。一个具有 SELECT ANY DICTIONARY 特权的用户可以使用表 SYS.SOURCE\$测试并行查询。FULL 和 PARALLEL 提示对于并行地扫描表是必需的。使用事件 10046 启用跟踪，因为过程 DBMS_MONITOR.SESSION_TRACE_ENABLE 对于并行执行进程没有任何影响。在客户端标识符中嵌入审计会话标识符保证其为唯一客户端标识符。

```
SQL> ALTER SESSION SET EVENTS '10046 trace name context forever, level 8';
Session altered.
SQL> VARIABLE client_identifier VARCHAR2(64)
SQL> EXEC :client_identifier:='pqtest_' || userenv('sessionid')
PL/SQL procedure successfully completed.
SQL> PRINT client_identifier
CLIENT_IDENTIFIER
-----
pqtest_1894
SQL> EXEC dbms_session.set_identifier(:client_identifier)
PL/SQL procedure successfully completed.
```

```
SQL> SELECT /*+ FULL(s) PARALLEL (s ,4) */ count(*) FROM sys.source$ s;
COUNT(*)
```

```
-----
298767
```

```
SQL> SELECT statistic, last_query
FROM v$pq_sesstat
WHERE statistic='Queries Parallelized';
STATISTIC                                LAST_QUERY
```

```
-----
Queries Parallelized                      1
SQL> EXEC dbms_session.set_identifier(NULL)
PL/SQL procedure successfully completed.
```

对 V\$PQ_SESSTAT 的查询证实 SELECT 语句并行运行。此时，该客户端标识符已被发送到来自并行执行进程的 4 个跟踪文件。

```
C:\oracle\admin\orcl\bdump> grep pqtest_1894 *.trc
orcl_p000_5412.trc:*** CLIENT ID:(pqtest_1894) 2007-08-31 23:14:38.421
orcl_p001_2932.trc:*** CLIENT ID:(pqtest_1894) 2007-08-31 23:14:38.421
orcl_p002_4972.trc:*** CLIENT ID:(pqtest_1894) 2007-08-31 23:14:38.421
orcl_p003_1368.trc:*** CLIENT ID:(pqtest_1894) 2007-08-31 23:14:38.421
```

TRCSESS 被用来将 4 个并行执行进程的跟踪文件和查询协调器的跟踪文件合并为一个单独的名为 pqtest_1894.trc 的跟踪文件。客户端标识符使用选项 clientid 传递给 TRCSESS。因为 TRCSESS 支持通配符并扫描所有匹配的文件，给本地目录 bdump 传递 *.trc 以及给查询协调器跟踪文件作为输入文件规范所在的目录传递..\udump*.trc 就已经足够了。

```
C:\oracle\admin\orcl\bdump> trcsess output=pqtest_1894.trc clientid=pqtest_1894 *.trc
c ..\udump\*.trc
```

输出文件是一个跟踪文件，包含数据库调用以及所有与并行执行相关的进程的等待事件。合并后的跟踪文件可以使用 TKPROF 处理。

```
C:\oracle\admin\orcl\bdump> tkprof pqtest_1894.trc pqtest_1894.tkp
```

其中查询协调器的跟踪文件报告零磁盘读取以及零一致性读取^①，合并并进行了格式化的跟踪文件提供了资源消耗的准确明细。

```
OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS
call      count      cpu    elapsed    disk    query    current    rows
-----
Parse       7      0.00      0.00      0        0        0        0
Execute     7      0.18      7.05    5209    5368        0        1
Fetch       4      0.01      1.87      0        0        0        2
-----
total      18      0.20      8.93    5209    5368        0        3
```

磁盘读取是因为并行执行进程使用直接路径读取，而不使用 SGA 中的缓冲区缓存。因为并行执行会避免缓冲区缓存，它就得益于操作系统在文件系统级别的缓存。

① FETCH #2:c=15625,e=1875092,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=1,tim=108052435005.

Statspack 是一个持续存储基于快照、与单个 ORACLE DBMS 或一个 RAC 实例集合相关的性能数据的 PL/SQL 包。它在 *Oracle9i Database Performance Tuning Guide and Reference* 手册一书以及 UNIX 和 Windows® 平台上的文件 \$ORACLE_HOME/rdbms/admin/spdoc.txt 中都进行了说明。后一个文档明显比 *Performance Tuning Guide and Reference* 更为详细。由于在 Oracle10g 中引入了 AWR, 已将 Statspack 文档从 *Oracle Database Performance Tuning Guide 10g Release 2* 手册中移除。

本章内容包括 Statspack 用法的高级方面, 比如未在文档中说明的报告参数, 以及如何将从 SQL 跟踪文件中找到的散列值识别的 SQL 语句与 Statspack 报告和 Statspack 库中的信息建立关联。此外, 它展示了大部分在文档中未说明的库结构, 并解释了怎样在 SQL 跟踪文件中找到使用的索引以及语句当前和过去的执行计划。它同样讲解了怎样通过使用分析函数 LAG 在大量的 Statspack 快照中识别高资源消耗的时期。

## 25.1 Statspack 介绍

Statspack 的介绍文档在 *Oracle9i Database Performance Tuning Guide and Reference Release 2* 中。有关 Statspack 的章节已经从 Oracle10g 的文档中移除。*Oracle Database Performance Tuning Guide 10g Release 2* 仅仅声明 Statspack 已被 AWR 取代, 事实并非如此, 因为 Statspack 在 Oracle10g 和 Oracle11g 中仍然可以使用。

在这里我就不全面介绍 Statspack 了, 但是为 Statspack 新手准备了一些开始使用包的最基本指令。详情请参考表 25-1。SQL*Plus 脚本的默认文件名的扩展名 .sql 在表中被省略。

将 Statspack 安装到模式 PERFSTAT 必须以用户 SYS 执行。除了截断操作之外, Statspack 表的所有其他操作都可以由具有 DBA 特权的用户运行。Statspack 通过 \$ORACLE_HOME/rdbms/admin 目录中许多名称为 sp*.sql 的脚本实现。

```
$ ls sp*.sql
spauto.sql   spcusr.sql   sppurge.sql   sprepsql.sql  spup816.sql
spcpkg.sql   spdropsql   spreprecon.sql sprsqins.sql  spup817.sql
spcreate.sql spdtab.sql   sprepreins.sql sptrunc.sql   spup90.sql
spctab.sql   spdusr.sql   sprepreport.sql spup10.sql    spup92.sql
```

① Windows 语法为 %ORACLE_HOME%\rdbms\admin\spdoc.txt。

请注意，使用环境变量 SQLPATH 将 ORACLE_HOME 设置为 SQL 脚本目录搜索路径的一部分，就省去了提供 Statspack SQL 脚本完整路径名称以及其他指定目录的脚本的必要。在 UNIX 系统上，使用冒号 (:) 作为多个目录的分隔符。

```
$ export SQLPATH=$ORACLE_HOME/rdbms/admin:$HOME/it/sql
```

在 Windows 系统上，使用分号 (;)。

```
C:> set SQLPATH=%ORACLE_HOME%\rdbms\admin;C:\home\ndebes\it\sql
```

表25-1 Statspack快速参考

动 作	SQL*Plus中输入的命令	运行用户
安装	SQL> @spcreate	SYS
性能数据的手动快照，使用可选的会话级快照和注释	SQL> EXEC statspack.snap (i_snap_level=>snapshot_level[, i_session_id=>sid_from_v\$session ][, i_ucomment=>'comment' ]) ^①	DBA
作业队列进程 (DBMS_JOB) 每小时整点执行自动快照	SQL> @spauto	DBA
报告	SQL> @spreport	DBA
通过快照ID范围清除过时Statspack数据，防止用户 PERFSTAT默认表空间溢出 ^②	SQL> @sppurge	DBA
截断包含快照数据的表	SQL> @sptrunc	PERFSTAT
卸载	SQL> @spdrop	DBA

我奉劝你至少每小时获取一个快照。如果你不经常性地抓取性能快照，当数据库用户打电话告知你他们在最近的某个时候遇到的性能问题时，你将会不知所措。你将不可能应答请求并找出原因，你能做的也就只好耸耸肩了。但是如果有了历史性能数据，你就可以查询它是在哪个时间发生，给在该时刻之前和之后的快照生成 Statspack 报告，就有可能通过使用脚本 sprepsql.sql^③ (需要快照级别 6 或更高) 查看一条昂贵语句的执行计划来进行分析，识别出问题的原因并解决它。

① Snapshot_level 是范围 1~10 之间的整数，sid_from_v\$session 是一个被捕捉了 CPU 消耗、等待事件和会话统计数据的会话的 V\$SESSION.SID，comment 是一个注释，当运行 spreport.sql 时它可以同快照 ID、时间和级别一起重现。Statspack 的发行版本包括 Oracle10g R2 在内，都有一个与特定会话报告生成相关的软件缺陷。仅在最后的快照中而不是在开始的快照中出现的等待事件，由于一个遗失的外部连接被遗漏了。我报告了这个问题，通过 bug 5145816 进行跟踪。该 bug 在 Oracle11g 中得到了修复。之前的版本没有进行修复，但是你可以使用源代码库中我提供的修复。

② 在 Oracle10g，清除 (purge) 功能是 Statspack 包 (STATSPACK.PURGE) 的一部分。源代码库包含对 Oracle9i 中该过程的移植。

③ 使用脚本 spauto.sql 创建的作业不在一个指定的快照级别调用 STATSPACK 包。它使用 STATSPACK_PARAMETER 中的默认快照级别，这个参数可以通过调用过程 STATSPACK.MODIFY_STATSPACK_PARAMETER(i_snap_level=>6)修改。由于在从 V\$SQL_PLAN 选择记录时的一个内部错误，快照级别 6 可能会造成 STATSPACK.SNAP 失败。试试使用 ALTER SYSTEM FLUSH SHARED_POOL 去刷新共享池，以防这个问题再出现。如果这不能阻止一个内部错误的发生，将快照级别降到 5 然后在下一个维护窗口重启实例。

### 25.1.1 检索捕获到的 SQL 语句文本

Statspack 是其前身 bstat/estat 的一个巨大改善。然而，令人不快的是所有的 SQL 语句被重现在 V\$SQLTEXT.SQL_TEXT 中并被强制加了换行符，并且超过 5 行语句的文本会被截断。下面是一个包含被截断语句的 Statspack 报告的摘录：

```
SQL ordered by CPU DB/Inst: ORCL/orcl Snaps: 2-3
-> Resources reported for PL/SQL code includes the resources used by all SQL
    statements called by the code.
-> Total DB CPU (s):                11
-> Captured SQL accounts for      48.4% of Total DB CPU
-> SQL reported below exceeded   1.0% of Total DB CPU
```

CPU Time (s)	Executions	CPU per Exec (s)	%Total	Elapsd Time (s)	Buffer Gets	Old Hash Value
10.34	11	0.94	12.3	31.65	162,064	1455318379

```
SELECT emp.last_name, emp.first_name, j.job_title, d.department_
name, l.city, l.state_province, l.postal_code, l.street_address,
emp.email, emp.phone_number, emp.hire_date, emp.salary, mgr.las
t_name FROM employees emp, employees mgr, departments d, locatio
ns l, jobs j WHERE emp.manager_id=mgr.employee_id AND emp.depart
```

让我们假定该语句消耗了太多的资源，应该被调整。但在你没有完整文本的情况下，你怎么调整该语句呢？在创建 Statspack 报告的时候，应用程序可能早已终止了。如果应用程序已被终止，使用 SQL 跟踪捕捉该语句为时已晚。

为了给你展示示例语句的原始格式是什么样的，我已经使用 SQL 跟踪捕获了它。

```
*** SERVICE NAME:(SYS$USERS) 2007-07-28 13:10:44.703
*** SESSION ID:(158.3407) 2007-07-28 13:10:44.703
...
*** ACTION NAME:(EMPLIST) 2007-07-28 13:10:57.406
*** MODULE NAME:(HR) 2007-07-28 13:10:57.406
...
PARSING IN CURSOR #8 len=416 dep=0 uid=0 oct=3 lid=0 tim=79821013130 hv=3786124882
ad='2d8f5f1c'
SELECT emp.last_name, emp.first_name, j.job_title, d.department_name, l.city,
       l.state_province, l.postal_code, l.street_address, emp.email,
       emp.phone_number, emp.hire_date, emp.salary, mgr.last_name
FROM hr.employees emp, hr.employees mgr, hr.departments d, hr.locations l, hr.jobs j
WHERE emp.manager_id=mgr.employee_id
AND emp.department_id=d.department_id
AND d.location_id=l.location_id
AND emp.job_id=j.job_id
END OF STMT
PARSE #8:c=46875,e=42575,p=0,cr=4,cu=0,mis=1,r=0,dep=0,og=1,tim=79821013122
```

开发人员很精细地格式化了语句，将 SELECT 语句的每一个子句放在了单独的行，并将很长的选择列表使用制表符进行了缩进。正如我们所见到的，所有这些在 Statspack 报告中是看不到的。

AWR 经过 HTML 格式化了了的报告确实包含 SQL 语句的完整文本（见图 25-1）。默认情况下，AWR 文本报告最多包含 4 行 SQL 语句文本。AWR 和数据库诊断包（Database Diagnostics Pack）捆绑在一起，它的价格为\$3000/处理器或\$60/用户（见 <http://oraclestore.oracle.com>）。一些 DBA 错误地认为使用 AWR 不需要额外的费用，因为它是默认安装和启用（STATISTICS_LEVEL=TYPICAL）的。

数据库诊断包包含 AWR、用于获取 AWR 快照和 AWR 管理的 DBMS_WORKLOAD_REPOSITORY 包、AWR 报告（awrrpt.sql）、DBA_HIST_* 和 DBA_ADVISOR_* 视图、ASH（比如 V\$ACTIVE_SESSION_HISTORY）、以及 ADDM（Automatic Database Diagnostic Monitor，自动数据库诊断监测）。当然，没有适当的许可，这些特性不可能通过企业管理网格控制或数据库控制进行访问。谈到 AWR 和 Statspack，AWR 快照不能像 Statspack 快照（见图 25-1 中的 i_session_id）那样包含特定会话数据。活跃会话历史对所有会话的度量进行取样，而一个会话级别的 Statspack 快照保存准确的、非取样度量的单个会话。

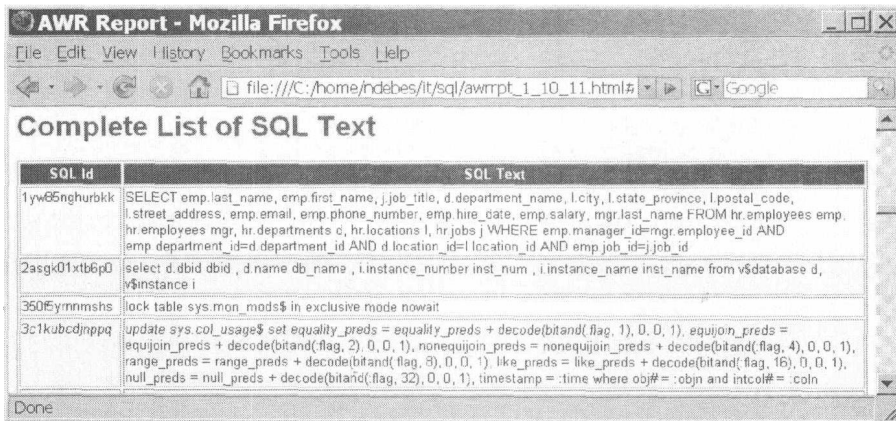


图 25-1 AWR 报告部分展示了完整版的 SQL 语句文本

如果你曾经阅读过 spdoc.txt 中 Oracle10g Statspack 文档，你可能知道在 Oracle10g 中，文件 sprepcn.sql（Statspack 报告配置）包含了很多参数，它们可以控制 Statspack 报告的外观以及哪些语句会在报告中^①。这是从 spdoc.txt 中节选的相关部分：

SQL section report settings - num_rows_per_hash

~~~~~

This is the upper limit of the number of rows of SQL Text to print for each SQL statement appearing in the SQL sections of the report. This variable applies to each SQL statement (i.e. hash\_value). The default value is 4, which means at most 4 lines of the SQL text will be printed for each SQL statement. To change this value, change the value of the variable num\_rows\_per\_hash.

e.g.

```
define num_rows_per_hash = 10;
```

<sup>①</sup> 在 Oracle9i 中，sprepcn.sql 并不存在，num\_rows\_per\_hash 在文档中也未作说明。

下面是 sprepcn.sql 相关代码节选：

```
-- SQL related report settings

-- Number of Rows of SQL to display in each SQL section of the report
define top_n_sql = 65;

-- Number of rows of SQL Text to print in the SQL sections of the report
-- for each hash_value
define num_rows_per_hash = 4;
...
define top_pct_sql = 1.0;
```

因此，用于获取性能差的 SQL 语句完整文本的解决方案就是增加 num\_rows\_per\_hash 的值。由于文档中并没有说明 Statspack 怎样存储捕获的 SQL 语句，我们就简单地设置 num\_rows\_per\_hash 为任意一个较大的值比如 1000，然后再次运行 Statspack 报告（脚本 \$ORACLE\_HOME/rdbms/admin/spreport.sql）。这次，完整地 SQL 语句文本就在报告中。

| CPU
Time (s) | Executions | CPU per
Exec (s) | %Total | Elapsed
Time (s) | Buffer Gets | Old
Hash Value |
|---|------------|---------------------|--------|---------------------|-------------|-------------------|
| 10.34 | 11 | 0.94 | 12.3 | 31.65 | 162,064 | 1455318379 |
| SELECT emp.last_name, emp.first_name, j.job_title, d.department_name, l.city, l.state_province, l.postal_code, l.street_address, emp.email, emp.phone_number, emp.hire_date, emp.salary, mgr.last_name FROM employees emp, employees mgr, departments d, locations l, jobs j WHERE emp.manager_id=mgr.employee_id AND emp.department_id=d.department_id AND d.location_id=l.location_id AND emp.job_id=j.job_id | | | | | | |

不幸的是，它还不是一个遵循正确 SQL 语法的格式。剩下的任务包括从报告中拷贝该语句然后编辑它，这样标识符中间的换行、SQL 保留字和常量都被移除了。这个过程很令人厌烦，特别是当语句超过 50 行左右。

更适宜的方法是直接到 PERFSTAT 模式中的 Statspack 库并从中检索语句，采用这种方法你的付出最终会得到回报。请注意，Statspack 从 V\$SQLTEXT 中复制 SQL 语句文本，而不是从 V\$SQLTEXT\_WITH\_NEWLINES。只有后一个视图在开发者原来放置的位置包含了换行符和制表符。如果你正在处理一个更复杂的语句，适当地格式化可以显著地减轻理解该语句的过程。从 V\$SQL 和 V\$SQLTEXT 中检索的 SQL 语句经过一个在文档中未说明的规范化过程处理后，删除了换行符和制表符。

在 Oracle10g 中，列 SQL\_FULLTEXT 被添加到 V\$SQL 中，以 CLOB 这个类型提供带有完整格式的 SQL 语句。

```
SQL> DESCRIBE v$sql
Name                               Null?    Type
-----
SQL_TEXT                           VARCHA2(1000)
```

| | |
|--------------|--------------|
| SQL_FULLTEXT | CLOB |
| SQL_ID | VARCHAR2(13) |

...

当使用 Oracle10g 时, 考虑到该语句文本仍然被缓存在 SGA 的共享池中, 下面的 SQL\*Plus 脚本检索带有完整格式的语句文本:

```
$ cat sql_fulltext.sql
-- pass old_hash_value as the single argument to the script
define old_hash_value='&1'
set verify off
set long 100000
set trimout on
set trimspool on
set feedback off
set heading off
set linesize 32767
col sql_fulltext format a32767
spool sp_sqltext_&old_hash_value.lst
SELECT sql_fulltext FROM v$sql WHERE old_hash_value=&old_hash_value;
spool off
exit
```

让我们使用 SELECT 语句测试需要调优的脚本 sql\_fulltext.sql:

```
$ sqlplus -s system/secret @sql_fulltext.sql 1455318379
SELECT emp.last_name, emp.first_name, j.job_title, d.department_name, l.city,
       l.state_province, l.postal_code, l.street_address, emp.email,
       emp.phone_number, emp.hire_date, emp.salary, mgr.last_name
FROM hr.employees emp, hr.employees mgr, hr.departments d, hr.locations l, hr.jobs j
WHERE emp.manager_id=mgr.employee_id
AND emp.department_id=d.department_id
AND d.location_id=l.location_id
AND emp.job_id=j.job_id
```

这一次, 位于原来位置的第 2 行和第 3 行的制表符以及换行符被保留了下来。即使语句从共享池老化移出或 DBMS 实例重启也无需感到绝望。下一节将介绍使用 Oracle9i PL/SQL 中引入的管道表函数来解决这一问题。

25.1.2 访问 STATS\$SQLTEXT

仔细查看主要的 Statspack 报告文件 sprepsins.sql, 你很快就会发现完整的语句文本被存放在表 STATS\$SQLTEXT 中, 而该语句的度量数据则在表 STATS\$SQL\_SUMMARY 中。spcpkg.sql 中包 STATSPACK 的源代码显示 STATS\$SQLTEXT 从 V\$SQLTEXT 复制语句文本。两个 V\$视图将 SQL 语句切分为最大字符长度为 64 的一些 VARCHAR2 片段。

```
SQL> DESCRIBE v$sqltext
Name                               Null?    Type
-----
ADDRESS                            RAW(4)
HASH_VALUE                          NUMBER
```

```

SQL_ID                                VARCHAR2(13)
COMMAND_TYPE                          NUMBER
PIECE                                 NUMBER
SQL_TEXT                              VARCHAR2(64)

```

```
SQL> DESCRIBE perfstat.stats$sqltext
```

| Name | Null? | Type |
|----------------|----------|--------------|
| OLD_HASH_VALUE | NOT NULL | NUMBER |
| TEXT_SUBSET | NOT NULL | VARCHAR2(31) |
| PIECE | NOT NULL | NUMBER |
| SQL_ID | | VARCHAR2(13) |
| SQL_TEXT | | VARCHAR2(64) |
| ADDRESS | | RAW(8) |
| COMMAND_TYPE | | NUMBER |
| LAST_SNAP_ID | | NUMBER |

这就解释了为什么不可能在 Statspack 报告中获得原始格式的语句文本。对表 STATS\$SQLTEXT 进行如下查询：

```

SQL> SELECT sql_text
FROM perfstat.stats$sqltext
WHERE old_hash_value=1455318379
ORDER BY piece;
SQL_TEXT

```

```

-----
SELECT emp.last_name, emp.first_name, j.job_title, d.department_
name, l.city, l.state_province, l.postal_code, l.street_address,
emp.email, emp.phone_number, emp.hire_date, emp.salary, mgr.las
t_name FROM employees emp, employees mgr, departments d, locatio
ns l, jobs j WHERE emp.manager_id=mgr.employee_id AND emp.depart
ment_id=d.department_id AND d.location_id=l.location_id AND emp.
job_id=j.job_id
7 rows selected.

```

由于最大片段长度的限制，就没有办法去掉 64 个字符之后的强制换行符。没有 SQL\*Plus 格式选项可以将连续的行衔接在一起。然而，如果有一些 PL/SQL 编程的背景，创建连续行的衔接来解决眼前这个问题也是比较简单的。

算法如下所示。

- (1) 创建一个抽象数据类型，其持有语句的散列值和 CLOB 类型的语句本身。请记住，单个 CLOB 可以存储至少 2 GB，而列 VARCHAR2 被限制在 4000 字节内。
- (2) 创建一个管道表函数，从 STATS\$SQLTEXT 中逐片选择数据行。
- (3) 使用 DBMS\_LOB.WRITEAPPEND 将每一片段添加到一个临时 CLOB 中，这样，将每一个片段衔接在一起消除强制换行符。
- (4) 当一个 SQL 或 PL/SQL 语句的所有片段都被用尽，使用行管道 (PIPE ROW (object\_type\_instance)) 将抽象数据类型的一个实例传递给该函数的调用者。
- (5) 从 SQL\*Plus 或其他数据库客户端使用 SELECT 语句的子句 TABLE 调用管道表函数 (SELECT \* FROM TABLE(function\_name(optional\_arguments)))。

管道表函数在 RETURN 子句后面需要关键字 PIPELINED。这个关键字说明该函数迭代地返回数据行。管道表函数的返回类型必须为集合类型。这个集合类型可以用 CREATE TYPE 声明为模式级别或在一个包内部。该函数迭代地返回该集合类型的单独元素。该集合类型的元素必须为受支持的 SQL 数据类型，比如 NUMBER 或 VARCHAR2。PL/SQL 数据类型，比如 PLS\_INTEGER 和 BOOLEAN 在管道表函数中不能作为集合元素。我们将给管道使用下面的对象类型：

```
CREATE OR REPLACE TYPE site_sys.sqltext_type AS OBJECT (
    hash_value NUMBER,
    sql_text CLOB
);
/
CREATE OR REPLACE TYPE site_sys.sqltext_type_tab AS TABLE OF sqltext_type;
/
```

管道表函数 SP\_SQLTEXT 的代码节选如下所示。对象被创建在模式 SITE\_SYS 中，因为 SYS 被保留给数据字典，并且模式 SYS 中的对象没有被完全导出。需要一些额外的工作获取在 Oracle9i 和 Oracle10g 中函数的兼容性。在 Oracle10g 和后续版本中，表 STATS\$SQLTEXT 中列 HASH\_VALUE 被重命名为 OLD\_HASH\_VALUE，因为在视图 V\$SQL 和 V\$SQLAREA 也有同样的重命名。Oracle9i 使用了一个同义词，而在 Oracle10g 和后续版本中则创建了一个视图以弥补被重命名的列。因此函数 SITE\_SYS.SP\_SQLTEXT 的代码在这两个版本的数据库上都保持不变，但是切记不要使用带有 DBMS\_SQL 的动态 SQL。

```
$ cat sp_sqltext.sql
-- run as a DBA user
CREATE USER site_sys IDENTIFIED BY secret PASSWORD EXPIRE ACCOUNT LOCK;
/* note that show errors does not work when creating objects in a foreign schema.
If you get errors either run this script as SITE_SYS after unlocking
the account or access DBA_ERRORS as below:
col text format a66
SELECT line,text from dba_errors where name='SP_SQLTEXT' ORDER BY line;
*/
-- cleanup, e.g. for database upgraded to 10g
begin
    execute immediate 'DROP SYNONYM site_sys.stats$sqltext';
    execute immediate 'DROP VIEW site_sys.stats$sqltext';
exception when others then null;
end;
/
GRANT SELECT ON perfstat.stats$sqltext TO site_sys;
/* for 9i:
CREATE OR REPLACE SYNONYM site_sys.stats$sqltext FOR perfstat.stats$sqltext;
for 10g, create this view:
CREATE OR REPLACE VIEW site_sys.stats$sqltext(hash_value, piece, sql_text) AS
SELECT old_hash_value, piece, sql_text FROM perfstat.stats$sqltext;
*/
declare
    version varchar2(30);
    compatibility varchar2(30);
```

```

begin
  dbms_utility.db_version(version, compatibility);
  if to_number(substr(version,1,2)) >= 10 then
    execute immediate 'CREATE OR REPLACE VIEW site_sys.stats$sqltext
      (hash_value, piece, sql_text) AS
      SELECT old_hash_value, piece, sql_text
      FROM perfstat.stats$sqltext';
  else
    execute immediate 'CREATE OR REPLACE SYNONYM site_sys.stats$sqltext
      FOR perfstat.stats$sqltext';
  end if;
end;
/
/*
  p_hash_value is either the hash value of a specific statement in
  STATS$SQLTEXT to retrieve or NULL.
  When NULL, all statements in the Statspack repository are retrieved.
  The column is called old_hash_value in Oracle10g
*/
CREATE OR REPLACE function site_sys.sp_sqltext(p_hash_value number default null)
RETURN sqltext_type_tab PIPELINED
AS
  result_row sqltext_type:=sqltext_type(null, empty_clob);
  cursor single_stmt(p_hash_value number) is
  select hash_value, piece, sql_text from stats$sqltext
  where p_hash_value=hash_value
  order by piece;

  cursor multi_stmt is
  select hash_value, piece, sql_text from stats$sqltext
  order by hash_value, piece;
  v_sql_text stats$sqltext.sql_text%TYPE;
  v_piece binary_integer;
  v_prev_hash_value number:=NULL;
  v_cur_hash_value number:=0;

BEGIN
  dbms_lob.CREATETEMPORARY(result_row.sql_text, true);
  IF p_hash_value IS NULL THEN
    open multi_stmt; -- caller asked for all statements
  ELSE
    open single_stmt(p_hash_value); -- retrieve only one statement
  END IF;
  LOOP
    IF p_hash_value IS NULL THEN
      FETCH multi_stmt INTO v_cur_hash_value, v_piece, v_sql_text;
      EXIT WHEN multi_stmt%NOTFOUND;
    ELSE
      FETCH single_stmt INTO v_cur_hash_value, v_piece, v_sql_text;
      EXIT WHEN single_stmt%NOTFOUND;
    
```

```

        END IF;
        IF v_piece=0 THEN -- new stmt starts
            IF v_prev_hash_value IS NOT NULL THEN
                -- there was a previous statement which is now finished
                result_row.hash_value:=v_prev_hash_value;
                pipe row(result_row);
                -- trim the lob to length 0 for the next statement
                dbms_lob.trim(result_row.sql_text, 0);
                -- the current row holds piece 0 of the new statement - add it to
CLOB
                dbms_lob.writeappend(result_row.sql_text, length(v_sql_text),
v_sql_text);
            ELSE
                -- this is the first row ever
                result_row.hash_value:=v_cur_hash_value;
                dbms_lob.writeappend(result_row.sql_text, length(v_sql_text),
v_sql_text);
            END IF;
        ELSE
            -- append the current piece to the CLOB
            result_row.hash_value:=v_cur_hash_value;
            dbms_lob.writeappend(result_row.sql_text, lengthb(v_sql_text),
v_sql_text);
        END IF;
        v_prev_hash_value:=v_cur_hash_value;
    END LOOP;
    -- output last statement
    pipe row(result_row);
    dbms_lob.freetemporary(result_row.sql_text);
    IF p_hash_value IS NULL THEN
        CLOSE multi_stmt;
    ELSE
        CLOSE single_stmt;
    END IF;
    return;
END;
/
GRANT EXECUTE ON site_sys.sp_sqltext TO dba;

```

下面的 SQL 脚本检索不带有强制换行符的语句文本，并将它保存在一个名为的 `sp_sqltext_hash_value.lst` 的 spool 文件中，其中 `hash_value` 是传递到该脚本的参数：

```

$ cat sp_sqltext_get.sql
define hash_value=81
set verify off
set long 100000
set trimout on
set trimspool on
set feedback off
set heading off

```

```

set linesize 32767
col sql_text format a32767
spool sp_sqltext_&hash_value..lst
select sql_text from table(site_sys.sp_sqltext(&hash_value));
spool off
exit

```

让我们使用正在讨论的语句散列值 1 455 318 379 测试该脚本。

```
$ sqlplus -s system/secret @sp_sqltext_get.sql 1455318379
```

```

SELECT emp.last_name, emp.first_name, j.job_title, d.department_name, l.city,
l.state_province, l.postal_code, l.street_address, emp.email,
emp.phone_number, emp.hire_date, emp.salary, mgr.last_name
FROM hr.employees emp, hr.employees mgr,
hr.departments d, hr.locations l, hr.jobs j
WHERE emp.manager_id=mgr.employee_id
AND emp.department_id=d.department_id
AND d.location_id=l.location_id
AND emp.job_id=j.job_id

```

现在，整个语句就处于文本中的单独的一行<sup>①</sup>（SQL\*Plus 在该文件的开头插入一个空白行，这样总的行数是 2）。

```

$ wc -l sp_sqltext_1455318379.lst
2 sp_sqltext_1455318379.lst

```

我们终于实现了目标，检索出具有正确 SQL 语法的语句。当不带参数或带有一个 NULL 参数调用时，函数 SP\_SQLTEXT 从 STATS\$SQLTEXT 中检索所有的语句。

25.1.3 使用保留格式捕获 SQL 语句

我们已经取得了很大的进展，但我们可以通过让 Statspack 保存带有保留换行符和制表符的 SQL 语句，往前更进一步。如果你不理解怎样改变包 STATSPACK 的包体（package body）中的单一行，请照着下面做。

请记住，V\$SQLTEXT\_WITH\_NEWLINES 保留换行符和制表符，而 Statspack 查询的 V\$SQLTEXT 则不会。首先，我们需要授权用户 PERFSTAT 访问动态性能视图 V\$SQLTEXT\_WITH\_NEWLINES。

```

SQL> CONNECT / AS SYSDBA
SQL> GRANT SELECT ON v_$sqltext_with_newlines TO PERFSTAT;

```

我使用 RCS<sup>②</sup>（Revision Control System，版本控制系统）保存 spcpkg.sql 的原始版本，它包含 STATSPACK 包体，版本为 1.1。由于 V\$SQLTEXT 在这个文件中仅被引用一次，足够我们修改单独一行了，如下所示（行编号来自 Oracle10g 版本的 spcpkg.sql）：

- ① UNIX 命令 wc 计算文件中的行数与字符数。如果需要在 Windows 上使用，从 <http://www.cygwin.com> 下载安装广受称赞的类 UNIX 环境 Cygwin，以访问 wc 和其他许多 UNIX 工具比如 awk、grep 和 find。
- ② RCS 是开源软件。在 Windows 平台的 Cygwin 和大多数的 Linux 分发包中包含了预编译可执行文件。命令 rcsdiff 会显示一个文件不同版本之间的差别。

```
$ rcsdiff spcpkg.sql
=====
RCS file: RCS/spcpkg.sql,v
retrieving revision 1.1
diff -r1.1 spcpkg.sql
4282c4282
<                                , v$sqltext vst
---
>                                , v$sqltext_with_newlines vst
```

请注意，做出的修改仅仅对 Statspack 新捕获的 SQL 语句有影响。对于其散列值已经出现在 STATS\$SQLTEXT.(OLD\_)HASH\_VALUE 的任何语句，都不会被再次捕捉。为了重新捕获已存在的语句，导出模式 PERFSTAT 保存以往的快照然后运行 `sptrunc.sql` 清理所有快照。这会从 STATS\$SQLTEXT 删除所有数据。不用担心，表 STATS\$STATSPACK\_PARAMETER 和 STATS\$IDLE\_EVENT 中 Statspack 的配置数据已经被保存，尽管在运行 `sptrunc.sql` 时会出现警告并声称它会 removes ALL data from Statspack tables（从 Statspack 表删除所有数据）。

在编辑 `spcpkg.sql` 之后，重新创建包 STATSPACK，重新运行该应用程序，然后 Statspack 就会捕获它的 SQL 语句，我们可以最终查看保留了完整格式的语句。

```
$ sqlplus -s system/secret @sp_sqltext_get.sql 1455318379
SELECT emp.last_name, emp.first_name, j.job_title, d.department_name, l.city,
       l.state_province, l.postal_code, l.street_address, emp.email,
       emp.phone_number, emp.hire_date, emp.salary, mgr.last_name
FROM hr.employees emp, hr.employees mgr, hr.departments d, hr.locations l, hr.jobs j
WHERE emp.manager_id=mgr.employee_id
AND emp.department_id=d.department_id
AND d.location_id=l.location_id
AND emp.job_id=j.job_id
```

这个改变并不会损害 Statspack 报告的显示。包含我们问题描述的部分现在变成了这样：

| CPU
Time (s) | Executions | CPU per
Exec (s) | %Total | Elapsed
Time (s) | Old
Buffer Gets | Hash Value |
|-----------------|------------|---------------------|--------|---------------------|--------------------|------------|
| 10.34 | 11 | 0.94 | 12.3 | 31.65 | 162,064 | 1455318379 |

```
SELECT emp.last_name, emp.first_name, j.job_title, d.department_
name, l.city,
       l.state_province, l.postal_code, l.street_address
, emp.email,
       emp.phone_number, emp.hire_date, emp.salary, mgr.l
ast_name
FROM hr.employees emp, hr.employees mgr, hr.departments
d, hr.locations l, hr.jobs j
WHERE emp.manager_id=mgr.employee_
id
AND emp.department_id=d.department_id
AND d.location_id=l.loc
ation_id
AND emp.job_id=j.job_id
```


25.2 未在文档中说明的 Statspack 报告参数

正如我们在前面的几节看到的，Oracle9i 中一个重要的未在文档中说明的 Statspack 报告参数是 num\_rows\_per\_hash。这个参数在 Oracle10g 中有文档说明，然而，在 Oracle10g 中仍然有很多新的和旧的未在文档中说明的报告参数。我认为它们其中任何一个都没有 num\_rows\_per\_hash 重要或有用。top\_n\_events 可能是最有意思的，它在数据库的两个版本中都没有相关的文档说明。它控制在报告开头附近的 Top Timed Events 部分显示多少行。通常这部分占用总的运行时间 (Oracle9i: Total Ela Time; Oracle10g: Total Call Time) 的最低比例小于 2%~3%，因此它的相关度很低。如果最低比例贡献者消耗掉总运行时间的比例大于等于 5%，它唯一的价值在于增加了 top\_n\_events。当然，任何这类贡献者也都将会出现在报告的 Wait Events 部分<sup>①</sup>，但占总的运行时间的百分比则仅在 Top Timed Events 部分显示。下面是一个来自 Oracle10g Statspack 报告 Top Timed Events 部分的示例：

| Top 5 Timed Events | | | Avg %Total | |
|-------------------------|-----------|----------|------------|------|
| ~~~~~ | | | wait | Call |
| Event | Waits | Time (s) | (ms) | Time |
| ----- | | | | |
| CPU time | | 671 | | 90.2 |
| db file sequential read | 1,525,262 | 36 | 0 | 4.9 |
| db file scattered read | 138,657 | 16 | 0 | 2.1 |
| latch: library cache | 7 | 15 | 2086 | 2.0 |
| log file parallel write | 5,187 | 3 | 1 | .4 |

日志文件的并行写入占用了运行时间的最小比例，其小于 1%，所以在这种情况下没有必要修改 top\_n\_events。表 25-2 列出了 Oracle10g R2 版本中未记录的 Statspack 报告参数。

表25-2 未记录的Oracle10g Statspack报告参数

| Statspack报告参数 | 用 途 | 默 认 值 |
|-------------------------|---|-------|
| avwt_fmt | 显示的平均等待时间精度 | 1 ms |
| cache_xfer_per_instance | 报告每一实例缓存转移的统计信息 | Y |
| display_file_io | 报告文件级别I/O统计信息 | Y |
| display_undostat | 报告undo数据段统计信息 | Y |
| linesize_fmt | 控制SQL*Plus输出的行的大小，如果任一列变得更宽的话也会增加 | 80 |
| streams_top_n | Oracle Streams相关统计数据的行数（例如捕获、传送和应用统计数据） | 25 |
| top_n_events | Top Timed Events报告部分的行数 <sup>②</sup> | 5 |
| top_n_undostat | undo统计数据的行数 | 35 |

① 如果某个组成部分的 CPU 时间太小以致不会出现在 Top Timed Events 中，这种情况是个特例，而这是不可能的。
② 变量 top\_n\_events 在 sprepins.sql 中进行了定义，而不是 sprepcn.sql。

25.3 Statspack 表

在 Oracle10g 中, Statspack 模式 PERFSTAT 包含 67 个表。在这些表中, 只有 STATS\$ENQUEUE\_STAT 和 STATS\$STATSPACK\_PARAMETER 被记录在了文件 spdoc.txt 中。一个 Statspack 模式包含很多数据库和打开该数据实例 (或 RAC 下的多个实例) 的信息。当解决性能问题或故障时, 直接查询这些表检测高资源利用率的快照或找出一个问题第一次发生的时间。表 25-3 包含所有表的列表, 使用 VS 视图填充每个表 (如果有的话), 以及对该表用途的一个简短说明。

表25-3 Oracle10g Statspack库表

| Statspack表 | 底层的VS视图 | 用 途 |
|-------------------------------|---|--|
| STATS\$BG_EVENT_SUMMARY | V\$SESSION_EVENT | 后台会话的等待事件 |
| STATS\$BUFFERED_QUEUES | V\$BUFFERED_QUEUES | 已缓冲队列的Stream统计信息 (已被处理的消息, 溢出到硬盘的消息等) |
| STATS\$BUFFERED_SUBSCRIBERS | V\$BUFFERED_SUBSCRIBERS | 已缓冲队列Stream订阅者的统计信息 |
| STATS\$BUFFER_POOL_STATISTICS | V\$BUFFER_POOL_STATISTICS | 数据库缓冲区缓存中每一个缓冲池的名称 (DEFAULT、KEEP、RECYCLE) 和块大小 (2~32 KB) 的缓冲池统计信息 |
| STATS\$CR_BLOCK_SERVER | V\$CR_BLOCK_SERVER | 有关RAC一致性读阻塞型服务器进程的统计信息 (全局缓存服务) |
| STATS\$CURRENT_BLOCK_SERVER | V\$CURRENT_BLOCK_SERVER | 全局缓存服务当前的块服务器统计信息 |
| STATS\$DATABASE_INSTANCE | V\$INSTANCE | Statspack已经捕获快照的DBMS实例 |
| STATS\$DB_CACHE_ADVICE | V\$DB_CACHE_ADVICE | 对数据库缓存缓冲区的每一缓冲池名称和数据块大小的调整建议 |
| STATS\$DLM_MISC | V\$DLM_MISC | 实时应用集群的全局入队服务和全局缓存服务的统计信息 |
| STATS\$DYNAMIC_REMASTER_STATS | X\$KJDRMAFNSTATS | RAC全局缓存服务资源重新调整 |
| STATS\$ENQUEUE_STATISTICS | V\$ENQUEUE_STATISTICS | 入队统计信息 |
| STATS\$EVENT_HISTOGRAM | V\$EVENT_HISTOGRAM | 等待事件的直方图统计信息 |
| STATS\$FILESTATXS | X\$KCBFWAIT、V\$FILESTAT、
V\$TABLESPACE、V\$DATAFILE | 每一数据文件的I/O统计信息 |
| STATS\$FILE_HISTOGRAM | V\$FILE_HISTOGRAM | 单个数据块物理读取时间的直方图统计, 统计数据按照每一文件、每一读取时间段 (0~2 ms、2~4 ms、4~8 ms等) 进行报告 |
| STATS\$IDLE_EVENT | V\$EVENT_NAME | Statspack认为处于空闲等待的事件 <sup>①</sup> |

① 在 Oracle10g 中, STATS\$IDLE\_EVENT 包含 Idle 等待类中的 41 个事件, Network 等待类中的 2 个事件, Other 等待类中的 3 个, 以及 Oracle10g 中不再存在的 24 个事件。有时候会涉及 STATS\$IDLE\_EVENT (参见 Metalink 上的数据库 bug), 因为有些版本的 Statspack 并没有把所有被认为是空闲等待的事件插入到表中。比如在 Oracle9i 中就不存在 RAC 的空闲等待事件 ges remote message。因此, 空闲等待事件并不会出现在名为“前 5 的等待事件”的报告部分中, 致使整个计算不可用。解决方法包括将丢失的空闲等待时间插入到表中, 重新生成 Statspack 报告。在 Oracle10g 中, Idle 等待类中的 21 个事件并没有在 STATS\$IDLE\_EVENT 中注册。在我看来, 当 dblink 中出现事件 SQL\*Net 信息的时候这一切都理所当然, 但是应该在 STATS\$IDLE\_EVENT 中出现 PL/SQL 锁定时器 (当调用 DBMS\_LOCK.SLEEP 将一个会话置为睡眠时会出现这个定时器)。

(续)

| Statspack表 | 底层的V\$视图 | 用 途 |
|--------------------------------|----------------------------|--|
| STATS\$INSTANCE_CACHE_TRANSFER | V\$INSTANCE_CACHE_TRANSFER | RAC缓存转移 |
| STATS\$INSTANCE_RECOVERY | V\$INSTANCE_RECOVERY | 对预估的平均恢复时间的统计，以防由实例、操作系统或硬件失效引发的崩溃恢复（参数FAST_START_MTTR_TARGET） |
| STATS\$JAVA_POOL_ADVICE | V\$JAVA_POOL_ADVICE | 调整Java池大小的建议（参数JAVA_POOL_SIZE） |
| STATS\$LATCH | V\$LATCH | 门锁统计信息 |
| STATS\$LATCH_CHILDREN | V\$LATCH_CHILDREN | 子门锁统计信息 |
| STATS\$LATCH_MISSES_SUMMARY | V\$LATCH_MISSES | 门锁请求失败 |
| STATS\$LATCH_PARENT | V\$LATCH_PARENT | 父门锁统计信息 |
| STATS\$LEVEL_DESCRIPTION | n/a | 快照级别0、5、6、7和10的描述 |
| STATS\$LIBRARYCACHE | V\$LIBRARYCACHE | 共享池统计信息，包括RAC特定的统计信息 |
| STATS\$MUTEX_SLEEP | V\$MUTEX_SLEEP | 互斥量（Mutex）统计信息 |
| STATS\$OSSTAT | V\$OSSTAT | 操作系统统计信息，比如空闲时间、忙碌时间、I/O等待以及系统中的CPU |
| STATS\$OSSTATNAME | V\$OSSTAT | STATS\$OSSTAT.OSSTAT_ID 的 查 找 表， 避 免 V\$OSSTAT.STAT_NAME冗余存储 |
| STATS\$PARAMETER | V\$PARAMETER | 初始化参数被捕获的值 |
| STATS\$PGASTAT | V\$PGASTAT | 自动PGA（程序全局区域）内存管理的PGA统计信息 |
| STATS\$PGA_TARGET_ADVICE | V\$PGA_TARGET_ADVICE | 当使用自动PGA内存管理时，对设置PGAAggregateTarget的建议 |
| STATS\$PROCESS_MEMORY_ROLLUP | V\$PROCESS_MEMORY | 每一类别（Freeable、Other、PL/SQL和SQL）已分配的PGA内存 |
| STATS\$PROCESS_ROLLUP | V\$PROCESS | 每一进程已分配的PGA内存 |
| STATS\$PROPAGATION_RECEIVER | V\$PROPAGATION_RECEIVER | Streams在接收（目的地）端已缓冲队列传播的统计信息 |
| STATS\$PROPAGATION_SENDER | V\$PROPAGATION_SENDER | Streams在发送（源）端已缓冲队列传播的统计信息 |
| STATS\$RESOURCE_LIMIT | V\$RESOURCE_LIMIT | 进程、会话、入队、并行执行和undo（或回滚）数据段的资源限制统计信息 |
| STATS\$ROLLSTAT | V\$ROLLSTAT | 回滚数据段统计信息 |
| STATS\$ROWCACHE_SUMMARY | V\$ROWCACHE | 每一类别（即数据段、序列和用户）的数据字典缓存（又名行数据缓存）统计信息 |
| STATS\$RULE_SET | V\$RULE_SET | 规则集合评估统计信息 |
| STATS\$SEG_STAT | V\$SEGMENT_STATISTICS | 具有高物理读取或争用的数据段，包括RAC特定的统计信息 |
| STATS\$SEG_STAT_OBJ | V\$SEGMENT_STATISTICS | STATS\$SEG_STAT 中列DATAOBJ#、OBJ#和TS#的查找表 |
| STATS\$SESSION_EVENT | V\$SESSION_EVENT | 会话特定等待事件的统计信息 |
| STATS\$SESSTAT | V\$SESSTAT | 如果参数I_SESSION_ID同STATSPACK.SNAP一起使用，就捕获特定会话的统计数据 |

(续)

| Statspack表 | 底层的V\$视图 | 用 途 |
|-------------------------------|---|---|
| STATS\$SESS_TIME_MODEL | V\$SESS_TIME_MODEL | 会话特定时间模型统计信息 |
| STATS\$SGA | V\$SGA | 与这些SGA组件有关的SGA调整信息：数据库缓冲、redo缓冲、变量大小和固定大小 |
| STATS\$SGASTAT | V\$SGASTAT | 共享池中单独池和空闲内存的统计信息 |
| STATS\$SGA_TARGET_ADVICE | V\$SGA_TARGET_ADVICE | SGA调整建议，如果启用了自动SGA内存管理（参数SGA_TARGET） |
| STATS\$SHARED_POOL_ADVICE | V\$SHARED_POOL_ADVICE | 共享池调整建议 |
| STATS\$SNAPSHOT | V\$INSTANCE、V\$SESSION | 存储每一个快照的详细数据，比如快照ID、实例编号、启动时间、快照级别、使用参数I_SESSION_ID传递给STATSPACK.SNAP的会话ID、注释和阈值 |
| STATS\$SQLTEXT | V\$SQLTEXT | 规范化的SQL语句文本被切分为最大为64字符的片段 |
| STATS\$SQL_PLAN | V\$SQL_PLAN | 被捕获的SQL语句的执行计划 |
| STATS\$SQL_PLAN_USAGE | V\$SQL_PLAN | 当使用执行计划时的快照ID、日期和时间 |
| STATS\$SQL_STATISTICS | V\$SQL | 所有SQL语句和非共享SQL语句消耗的内存 |
| STATS\$SQL_SUMMARY | V\$SQLSTATS | 性能度量，比如运行时间、CPU使用、磁盘读取、直接路径写入和被捕获的SQL语句缓冲读（get） |
| STATS\$SQL_WORKAREA_HISTOGRAM | V\$SQL_WORKAREA_HISTOGRAM | 0~1 KB、1~2 KB、2~4 KB、4~8 KB等工作区大小的直方图统计信息 |
| STATS\$STATSPACK_PARAMETER | n/a | Statspack参数，比如默认快照级别和阈值 |
| STATS\$STREAMS_APPLY_SUM | V\$STREAMS_APPLY_SERVER、
V\$STREAMS_APPLY_READER | Streams应用进程统计信息 |
| STATS\$STREAMS_CAPTURE | V\$STREAMS_CAPTURE | Streams捕获的统计信息 |
| STATS\$STREAMS_POOL_ADVICE | V\$STREAMS_POOL_ADVICE | Streams池调整建议（参数STREAMS_POOL_SIZE） |
| STATS\$SYSSTAT | V\$SYSSTAT | 全实例统计数据，比如总的CPU消耗、循环CPU使用、解析调用（hard/total，硬解析除以总的解析数）、事物数等 |
| STATS\$SYSTEM_EVENT | V\$SYSTEM_EVENT | 全实例等待事件统计信息 |
| STATS\$SYS_TIME_MODEL | V\$SYS_TIME_MODEL | 全实例时间模型统计信息 |
| STATS\$TEMPSTATXS | V\$TABLESPACE、V\$TEMPFILE、
V\$TEMPSTAT、X\$KCBFWAIT | 临时数据段排序统计信息 |
| STATS\$TEMP_HISTOGRAM | V\$TEMP_HISTOGRAM | 在储存桶（bucket）中对来自临时数据段的单个数据块1~2、2~4、4~8、8~16毫秒等持续读取操作数的直方图统计 |
| STATS\$THREAD | V\$THREAD | 在线redo日志线程信息 |
| STATS\$TIME_MODEL_STATNAME | V\$SYS_TIME_MODEL | 表STATS\$SYS_TIME_MODEL和STATS\$SESS_TIME_MODEL中列STAT_ID的查找表 |
| STATS\$UNDOSTAT | V\$UNDOSTAT | Undo数据段统计信息 |
| STATS\$WAITSTAT | V\$WAITSTAT | 数据块、扩展位图、文件头块、空闲列表、undo块、undo头部等的块争用统计 |

25.4 找出 Statspack 库中代价高的语句

为了快速得到整个 Statspack 库中昂贵语句的概况，上一节中函数 SP\_SQLTEXT 的结果可以与 STATS\$SQL\_SUMMARY 连接，后者包含捕获的所有 SQL 语句的度量数据。如果通过一条语句被执行了多少次 (STATS\$SQL\_SUMMARY.EXECUTIONS) 来规范数字，就得到了慢语句的一个初始概况。

下面是完成这一任务的脚本 sp\_sqltext\_join.sql。它以秒为单位报告运行时间 (STATS\$SQL\_SUMMARY.ELAPSED\_TIME 则为微秒)。磁盘读取和缓冲读通过该语句的执行计数规范化。这个脚本限制结果集为超过 1 秒时间完成的语句。当然，在开始一个调优会话之前，你应该确认通过这种方式找到的语句会损坏业务流程。

```
$ cat sp_sqltext_join.sql
set long 1000000
col module format a6
col snap_id format 9999999
col sql_text format a80 word_wrapped
SELECT s.snap_id, s.old_hash_value,
       round(s.elapsed_time/s.executions/1000000, 2) ela_sec_per_exec,
       floor(s.disk_reads/s.executions) read_per_exec,
       floor(s.buffer_gets/s.executions) gets_per_exec,
       s.module, t.sql_text
FROM stats$sql_summary s,
     (SELECT hash_value, sql_text from table(site_sys.sp_sqltext())) t
WHERE s.old_hash_value=t.hash_value
AND s.elapsed_time/s.executions/1000000 > 1
ORDER BY s.elapsed_time, s.disk_reads, s.buffer_gets;
```

运行前面的查询，得到了这个用作示例并贯穿本章的众所周知的语句：

```
SNAP_ID OLD_HASH_VALUE ELA_SEC_PER_EXEC READ_PER_EXEC GETS_PER_EXEC MODULE
-----
SQL_TEXT
-----
      33      1455318379          2.87          2380          14733 HR
SELECT emp.last_name, emp.first_name, j.job_title, d.department_name, l.city,
l.state_province, l.postal_code, l.street_address, emp.email, emp.phone_number,
emp.hire_date, emp.salary, mgr.last_name FROM hr.employees emp, hr.employees
mgr, hr.departments d, hr.locations l, hr.jobs j WHERE
emp.manager_id=mgr.employee_id AND emp.department_id=d.department_id AND
d.location_id=l.location_id AND emp.job_id=j.job_id
```

上面的 SELECT 语句检索从实例启动时就开始捕获的度量。它不考虑快照间隔，这样它可能丢失间歇性较慢但平均性能可接受的语句。

25.5 识别使用过的索引

除了普通的度量数据外，Statspack 在级别 6 或更高级别的快照捕获执行计划。由于使用 ALTER INDEX MONITORING 监测索引的使用情况有点鲁莽（见第 4 章），你可能考虑通过访问 Statspack 库

检查索引的使用情况（脚本 `sp_used_indexes.sql`）。

```
SQL> SELECT DISTINCT o.owner, o.object_name index_name
FROM dba_objects o, stats$sql_plan p
WHERE o.object_id=p.object#
AND o.object_type='INDEX'
AND o.owner='HR';
OWNER          INDEX_NAME
-----
HR              JOB_ID_PK
HR              LOC_ID_PK
HR              EMP_EMP_ID_PK
HR              DEPT_ID_PK
```

25.6 SQL Trace 捕获语句的执行计划

当使用 SQL Trace 跟踪应用程序时，可能出现某些语句的执行计划在跟踪文件中找不到的情况，因而在 TKPROF 格式化报告中也会找不到。在这种情况下，不要使用带选项 EXPLAIN 的 TKPROF 或手动执行 EXPLAIN PLAN，因为这会产生一个不同于应用程序所使用的执行计划的风险。相反，使用 Statspack 或 AWR（见第 26 章）从各自的工具库中检索实际使用的执行计划。使用 Statspack 的过程相对简单一些，因为被发送到跟踪文件的散列值可用于获取需要的信息。在 Oracle9i 中，V\$SQL 有一个单独的散列值，也就是这个散列值同样被发送到跟踪文件。在 Oracle10g 中，问题更复杂一些，因为现在有两个散列值：V\$SQL.OLD\_HASH\_VALUE 以及 V\$SQL.HASH\_VALUE，仅后者被写入到 SQL 跟踪文件。Oracle10g Statspack 在它的 SQL 报告中使用旧的散列值，该散列值使用 Oracle9i 中使用的算法计算。将跟踪文件中找到的 HASH\_VALUE 翻译成需要用于运行 Statspack SQL 报告的 OLD\_HASH\_VALUE，可以通过下面的查询完成（脚本 `sp_translate_hv.sql`）：

```
SQL> SELECT p.snap_id, s.snap_time, p.sql_id, p.hash_value, p.old_hash_value,
p.plan_hash_value, p.cost
FROM stats$sql_plan_usage p, stats$snapshot s
WHERE p.snap_id=s.snap_id
AND p.hash_value=3786124882
ORDER BY p.snap_id;
SNAP_ID SNAP_TIME  SQL_ID          HASH_VALUE OLD_HASH_VALUE PLAN_HASH_VALUE COST
-----
493 13. Oct 07 1yw85nghurbkk 3786124882 1455318379 4095786543 9
502 13. Oct 07 1yw85nghurbkk 3786124882 1455318379 4095786543 9
582 15. Oct 07 1yw85nghurbkk 3786124882 1455318379 3985860841 17
602 15. Oct 07 1yw85nghurbkk 3786124882 1455318379 4095786543 9
```

查询结果包含列 PLAN\_HASH\_VALUE 的一些值。因此随时间推移查询会使用不同的执行计划。为了生成 Statspack SQL 报告，运行脚本 `sprepsql.sql`（或者，如果来自另一个数据库的 Statspack 数据被导入，就运行 `sprsqins.sql`），然后输入之前查询结果中任一相邻的快照标识符。

```
SQL> @sprepsql
```

...
Specify the Begin and End Snapshot Ids

~~~~~  
Enter value for begin_snap: 582  
Begin Snapshot Id specified: 582

Enter value for end_snap: 602  
End Snapshot Id specified: 602

如果你使用的是 Oracle10g, 在脚本需要语句散列值的地方, 确保输入 OLD_HASH_VALUE。

Specify the old (i.e. pre-10g) Hash Value

~~~~~  
Enter value for hash\_value: 1455318379
Hash Value specified is: 1455318379

...
Known Optimizer Plan(s) for this Old Hash Value

~~~~~  
Shows all known Optimizer Plans for this database instance, and the Snap Id's they were first found in the shared pool. A Plan Hash Value will appear multiple times if the cost has changed  
-> ordered by Snap Id

First Snap Id	First Snap Time	Last Active Time	Plan Hash Value	Cost
493	13-Oct-07 21:41	15-Oct-07 15:38	4095786543	9
502	13-Oct-07 21:47	15-Oct-07 15:14	3985860841	17

Plans in shared pool between Begin and End Snap Ids

~~~~~  
Shows the Execution Plans found in the shared pool between the begin and end snapshots specified. The values for Rows, Bytes and Cost shown below are those which existed at the time the first-ever snapshot captured this plan - these values often change over time, and so may not be indicative of current values
-> Rows indicates Cardinality, PHV is Plan Hash Value
-> ordered by Plan Hash Value

| Operation | PHV/Object Name | Rows | Bytes | Cost |
|-------------------|------------------------|------|-------|------|
| SELECT STATEMENT | ----- 3985860841 ----- | | | 17 |
| HASH JOIN | | 105 | 17K | 17 |
| TABLE ACCESS FULL | JOBS | 19 | 513 | 3 |
| HASH JOIN | | 105 | 14K | 14 |
| TABLE ACCESS FULL | EMPLOYEES | 107 | 1K | 3 |
| HASH JOIN | | 106 | 13K | 10 |
| HASH JOIN | | 27 | 1K | 7 |
| TABLE ACCESS FULL | LOCATIONS | 23 | 1K | 3 |
| TABLE ACCESS FULL | DEPARTMENTS | 27 | 513 | 3 |

| | | | | |
|-----------------------------|------------------------|-----|-----|---|
| TABLE ACCESS FULL | EMPLOYEES | 107 | 6K | 3 |
| SELECT STATEMENT | ----- 4095786543 ----- | | | 9 |
| NESTED LOOPS | | 105 | 17K | 9 |
| NESTED LOOPS | | 105 | 12K | 7 |
| NESTED LOOPS | | 105 | 9K | 6 |
| NESTED LOOPS | | 106 | 8K | 4 |
| TABLE ACCESS FULL | EMPLOYEES | 107 | 6K | 3 |
| TABLE ACCESS BY INDEX ROWID | DEPARTMENTS | 1 | 19 | 1 |
| INDEX UNIQUE SCAN | DEPT_ID_PK | 1 | | 0 |
| TABLE ACCESS BY INDEX ROWID | EMPLOYEES | 1 | 12 | 1 |
| INDEX UNIQUE SCAN | EMP_EMP_ID_PK | 1 | | 0 |
| TABLE ACCESS BY INDEX ROWID | JOBS | 1 | 27 | 1 |
| INDEX UNIQUE SCAN | JOB_ID_PK | 1 | | 0 |
| TABLE ACCESS BY INDEX ROWID | LOCATIONS | 1 | 48 | 1 |
| INDEX UNIQUE SCAN | LOC_ID_PK | 1 | | 0 |

该脚本检索所有带有指定旧散列值语句的执行计划。因为在将 DBMS 软件升级到 Oracle10g 时执行计划可能发生改变，我建议升级前在级别 6 或更高级别捕获 Statspack 快照。

Oracle 公司提供脚本 spup10.sql，用于将一个 Statspack 库升级到 Oracle10g，但声明该升级并不能保证一定会正常工作。我对该脚本和 Oracle10g R2 的有限的经历就是：库升级没有成功。为了保留 Oracle9i 捕获的 Statspack 快照，在升级到 Oracle10g 之前使用导出工具 (exp) 将模式 PERFSTAT 导出。如果有必要，一个 Oracle9i 导出转储可被导入到一个 Oracle9i 测试数据库用于运行报告（见本章后面的 25.8 节）。

25.7 找出高资源利用率的快照

作为我咨询工作的一部分，我偶尔会被要求详细检查整个 Statspack 库。客户在他的生产数据库中创建一个 PERFSTAT 模式级别导出，然后我将它导入到一个测试数据库中（稍后详细介绍导入 Statpack 数据）。这是在你起初对哪个快照是否值得调研毫无头绪时的一个解决方案。

因为一个 Statspack 库包含来自 V\$视图性能数据的快照，Statspack 报告必须从快照时间区间结束时的度量数据中减去快照时间区间开始时的度量数据，以得到在这个时间区间的资源消耗。报告脚本 spreport.sql 将传入开始时和结束时的快照编号作为输入。因此一个简单的连接操作（或外部连接，如果必要的话）就足够了。

如果需要调研所有的快照，问题会更加复杂。这个任务就是分析函数 LAG，它将前一通过 SELECT 语句访问的数据行的列值映射到当前数据行而没有自连接，这样一个窗口就同时包含两条有效的数据行。此时，有心的读者就会理直气壮地反对：在一个关系型数据库中没有前一数据行。这就是一个排序必须使用 SQL 关键字 OVER 和 ORDER BY 定义的原因。下面是一个使用 LAG 查询 STATS\$SNAPSHOT 的例子：

```
SQL> SET NULL <NULL>
SQL> SELECT LAG(snap_id) OVER (ORDER BY snap_id) AS start_snap_id,
           snap_id AS end_snap_id
FROM stats$snapshot;
```



```
START_SNAP_ID END_SNAP_ID
```

```
-----
<NULL>          33
          33      34
          34      35
```

请注意，第一行中开始的快照标识符为 NULL，因为它在窗口之外。有关 LAG 的详细信息，请查询 *Oracle Database SQL Reference* 手册。

一个得到连续快照标识符并证明该区间为有效的视图，可以创建为基础构建块用于分析。检查区间的有效性是通过比较列 STATS\$SNAPSHOT.STARTUP\_TIME 中的值实现的。如果启动时间不匹配，那么该实例在快照之间被重启，度量也就不可用了。在 RAC 环境下，问题变得更加复杂。因为单独一个库可能包含来自许多 RAC 实例的度量数据，开始时和结束时快照的实例编号也必须匹配。这通过使用函数 LAG 提供的 ORDER BY 中的列 INSTANCE\_NUMBER 和 SNAP\_ID 完成。检索有效的连续开始和结束快照标识符的视图称作 SP\_VALID\_INTERVALS（脚本 sp\_valid\_intervals.sql）。

```
CREATE OR REPLACE VIEW site_sys.sp_valid_intervals AS
SELECT *
FROM (
  SELECT lag(dbid) over (order by dbid, instance_number, snap_id) AS start_dbid,
         dbid AS end_dbid,
         lag(snap_id) over (order by dbid, instance_number, snap_id) AS start_snap_id,
         snap_id AS end_snap_id,
         lag(instance_number) over (order by dbid, instance_number, snap_id)
         AS start_inst_nr, instance_number AS end_inst_nr,
         lag(snap_time) over (order by dbid, instance_number, snap_id)
         AS start_snap_time, snap_time AS end_snap_time,
         lag(startup_time) over (order by dbid, instance_number, snap_id)
         AS start_startup_time, startup_time AS end_startup_time
  FROM perfstat.stats$snapshot
) iv
WHERE iv.start_snap_id IS NOT NULL
AND iv.start_dbid=iv.end_dbid
AND iv.start_inst_nr=iv.end_inst_nr
AND iv.start_startup_time=iv.end_startup_time;
```

下面是一个访问视图 SP\_VALID\_INTERVALS 的查询：

```
SELECT start_snap_id, end_snap_id, start_inst_nr, start_snap_time,
trunc((end_snap_time-start_snap_time)*86400) AS interval
FROM site_sys.sp_valid_intervals;
```

| START_SNAP_ID | END_SNAP_ID | START_INST_NR | START_SNAP_TIME | INTERVAL |
|---------------|-------------|---------------|---------------------|----------|
| 87 | 88 | 1 | 15.08.2007 06:06:04 | 1898 |
| 88 | 89 | 1 | 15.08.2007 06:37:42 | 2986 |
| 90 | 91 | 1 | 15.08.2007 09:35:21 | 1323 |

25.7.1 高 CPU 使用

高 CPU 使用可能是出现性能问题的迹象，这样显示高 CPU 使用的快照区间就是实施进一步

调研的依据。CPU 使用通常通过可用 CPU 资源的百分比进行表达。应用于 Statspack，这意味着我们需要在连续快照区间查看 CPU 利用率。从 Statspack 库表中计算 CPU 利用率的算法如下。

(1) 通过从快照结束时捕获的值减去快照开始时的值，得到在该快照区间的 CPU 消耗（厘秒精度）。这个值是所有 Statspack 报告的一部分并通过 STATSPACK 包计算。从实例启动开始到现在的 CPU 消耗通过 STATS\$SYSSTAT 中的统计数据“该会话使用的 CPU”（CPU used by this session）进行了描述。

(2) 该值除以 100，将精度从厘秒变为秒。

(3) 将快照区间的 CPU 消耗除以秒。区间的长度使用分析函数 LAG 从 STATS\$SNAPSHOT 中得到。

(4) 将结果除以在快照区间 (STATS\$PARAMETER) 开始时捕获到的 CPU 个数 (参数 CPU\_COUNT)，得到平均 CPU 利用率作为 CPU 能力的百分比。一个系统的 CPU 能力是 CPU 的每秒运算次数。

让我们看一个例子。我们将使用来自下面 Statspack 报告的节选作为计算的示例数据。相关的数据字体已被加粗。

| | Snap Id | Snap Time | Sessions | Curs/Sess | Comment |
|--------------------------|---------|--------------------|----------|------------|-----------|
| Begin Snap: | 90 | 15-Aug-07 09:35:21 | 215 | 11.9 | |
| End Snap: | 91 | 15-Aug-07 09:57:24 | 177 | 10.6 | |
| Elapsed: | | 22.05 (mins) | | | |
| ... | | | | | |
| Statistic | | | Total | per Second | per Trans |
| CPU used by this session | | | 82,337 | 62.2 | 23.0 |

参数 CPU\_COUNT 的值被捕获了，但没有被打印在 Statspack 报告的最后，因为它具有默认值，因此我们需要从 STATS\$PARAMETER 检索该值。

```
SQL> SELECT value FROM stats$parameter WHERE name='cpu_count' and snap_id=90;
VALUE
-----
4
```

如果我们将之前的算法翻译为公式，它变成这样：

$$\text{CPU利用率(\%)} = \frac{\text{快照期间CPU消耗 (s)} * 100}{\text{快照区间长度 (s)} * \text{CPU\_COUNT}}$$

使用示例数据就得到这个：

$$\frac{834.37 * 100}{1323 * 4} = 15.56\%$$

下面的查询将该计算自动化了（脚本 snap\_by\_cpu\_util.sql）：

```
SELECT i.start_snap_id, i.end_snap_id,
i.start_snap_time, i.end_snap_time,
(i.end_snap_time - i.start_snap_time) * 86400 AS interval,
round((((s2.value - s1.value) / 100 / ((i.end_snap_time - i.start_snap_time) * 86400)
/ p.value) * 100,2) AS cpu_utilization
FROM site_sys.sp_valid_intervals i, stats$sysstat s1,
```

```
stats$sysstat s2, stats$parameter p
WHERE i.start_snap_id=s1.snap_id
AND i.end_snap_id=s2.snap_id
AND s1.name='CPU used by this session'
AND s1.name=s2.name
AND p.snap_id=i.start_snap_id
AND p.name='cpu_count'
ORDER BY cpu_utilization DESC;
```

运行该脚本确认介于快照 90 和 91 区间的手动计算结果：

| Start | End | CPU | | | |
|--------|--------|---------------------|---------------------|--------------|-----------------|
| SnapID | SnapID | Start Time | End Time | Interval (s) | Utilization (%) |
| 90 | 91 | 15.08.2007 09:35:21 | 15.08.2007 09:57:24 | 1323 | 15.56 |
| 88 | 89 | 15.08.2007 06:37:42 | 15.08.2007 07:27:28 | 2986 | 7.14 |
| 87 | 88 | 15.08.2007 06:06:04 | 15.08.2007 06:37:42 | 1898 | 5.28 |

这个查询快速识别出高负载时期，值得为最高 CPU 使用的开始和结束快照生成 Statspack 报告，以进行剖析。

25.7.2 高 DB 时间

要是一个性能问题是由于等待并且不通过高 CPU 使用显示它自己，会怎么样呢？那么在前一节中展示的方法就会失败。等待可能是由对门锁、锁或其他资源的争用引起的，比如较慢的磁盘。Oracle10g 提供所谓的考虑了 CPU 时间消耗和等待的时间模型统计。这些在实例 (V\$SYS\_TIME\_MODEL) 和会话 (V\$SESS\_TIME\_MODEL) 级别有效。本质上，在数据库实例中花费的时间就作出了解释。根据 *Oracle10g Database Reference* 的叙述：

DB 时间是花费于执行数据库用户级别请求的运行时间的总和。这不包括花费于实例后台进程比如 PMON 的时间。

该手册进一步规定指标 DB Time 包含如下条目：

- DB CPU
- 连接管理调用的总运行时间
- 序列加载的总运行时间
- sql 执行的总运行时间
- 解析的总运行时间
- PL/SQL 执行的总运行时间
- 入站 PL/SQL 远程过程调用的总运行时间
- PL/SQL 编译的总运行时间
- Java 执行的总运行时间

这里没有一个词提到等待时间。然而，考虑到在 SQL 跟踪文件中等待时间出现在数据库调用的运行时间之中，如果时间模型统计采用一个不同的方法就会很奇怪。可以利用未在文档中说

明的 PL/SQL 包 DBMS\_SYSTEM 生成一些人为的等待时间以进行一个快速验证。在一个新的数据库会话的开始, V\$SESS\_TIME\_MODEL 中所有的值都接近 0。

```
SQL> SELECT stat_name, value/1000000 time_secs FROM v$sess_time_model
WHERE (stat_name IN ('sql execute elapsed time','PL/SQL execution elapsed time')
      OR stat_name like 'DB%')
AND sid=userenv('sid');
STAT_NAME                                TIME_SECS
-----
DB time                                  .018276
DB CPU                                  .038276
sql execute elapsed time                 .030184
PL/SQL execution elapsed time            .007097
```

下面, 我们通过让事件 db file scattered read 人为地等待 1 秒, 生成一些等待时间, 否则该事件会在一个 SELECT 语句导致全表扫描时发生。虽然实际上该等待发生在一个 PL/SQL 过程中, 但可以解释好像是因为一条 SQL 语句导致了全表扫描。

```
SQL> EXECUTE dbms_system.wait_for_event('db file scattered read', 1, 1);
PL/SQL procedure successfully completed.
```

请注意, 指标“sql 执行的总运行时间”(sql execute elapsed time) 和“PL/SQL 执行运行时间”(PL/SQL execution elapsed time) 二者都大约只增加了 1 秒。显然, 由于该测试的人为因素, 运行时间被视为两倍。指标 DB CPU 仅稍微增长了一点, 而 DB Time 同样增加了 1 秒, 因为它是 SQL 和 PL/SQL 运行时间的合并<sup>①</sup>。

```
SQL> SELECT stat_name, value/1000000 time_secs FROM v$sess_time_model
WHERE (stat_name IN ('sql execute elapsed time','PL/SQL execution elapsed time')
      OR stat_name like 'DB%')
AND sid=userenv('sid');
STAT_NAME                                TIME_SECS
-----
DB time                                  1.030818
DB CPU                                  .045174
sql execute elapsed time                 1.017276
PL/SQL execution elapsed time            .987208
```

该测试说明作为 SQL 执行一部分发生的事件的等待出现在指标“sql 执行运行时间”中。除了等待类别为 Idle 的等待事件外<sup>②</sup>, 等待时间同样累计在 DB Time。

如同 CPU 使用那样, 我们需要以某种方式规范化 DB Time。计算机系统或多或少具有无限等待时间的能力。运行在系统上竞争相同资源的进程越多, 在系统级别累计的等待时间也就越多。当 10 个进程, 每一个都对相同的 TX 队列等待 1 秒时, 总的等待时间就是单个进程结果的 10 倍。指标 DB Time 可以通过快照区间规范化。我应该叫这个指标为“相对 DB 时间”(relative DB time),

① 有趣的是, 等待的 1 秒钟并不能解释指标 DB time 加倍。

② 在一个 Oracle10g 实例上运行下面的查询, 检索所有等待类别为 Idle 的事件:

```
SELECT name FROM v$event_name WHERE wait_class='Idle'.
```

我们将再一次从手工计算开始。一个 Oracle10g Statspack 报告的相关部分节选如下。计算“相对 DB 时间”过程中需要的数字的字体已被加粗。

| Snapshot | Snap Id | Snap Time | Sessions | Curs/Sess | Comment |
|-------------|---------|--------------------|----------|-----------|---------|
| ~~~~~ | ----- | ----- | ----- | ----- | ----- |
| Begin Snap: | 83 | 06-Sep-07 17:04:06 | 24 | 3.3 | |
| End Snap: | 84 | 06-Sep-07 17:09:54 | 24 | 3.3 | |
| Elapsed: | | 5.80 (mins) | | | |

Time Model System Stats DB/Inst: TEN/TEN1 Snaps: 83-84
-> Ordered by % of DB time desc, Statistic name

| Statistic | Time (s) | % of DB time |
|-------------------------------|----------|--------------|
| ----- | ----- | ----- |
| sql execute elapsed time | 319.3 | 100.0 |
| PL/SQL execution elapsed time | 316.7 | 99.2 |
| DB CPU | 301.4 | 94.4 |
| ... | | |
| DB time | 319.3 | |

将其表示为一个方程，相对 DB 时间如下所示：

$$\text{相对DB时间 (s)} = \frac{\text{DB 时间 (s)}}{\text{快照区间 (s)}}$$

使用示例数据得到以下结果：

$$\frac{319.3}{5.8 \cdot 60} = 0.92$$

自动化该计算过程的查询也同样基于视图 SP\_VALID\_INTERVALS（文件 snap\_by\_db\_time.sql）。

```
SQL> SELECT i.start_snap_id, i.end_snap_id,
i.start_snap_time, i.end_snap_time,
(i.end_snap_time - i.start_snap_time) * 86400 AS interval,
round((s2.value - s1.value) / 1000000 /* convert from microsec to sec */
/ ((i.end_snap_time - i.start_snap_time) * 86400 ), 2)
/* normalize by snapshot interval */
AS db_time_per_sec
FROM site_sys.sp_valid_intervals i, stats$sys_time_model s1,
stats$sys_time_model s2, stats$time_model_statname n
WHERE i.start_snap_id=s1.snap_id
AND i.end_snap_id=s2.snap_id
AND n.stat_name='DB time'
AND s1.stat_id=n.stat_id
AND s2.stat_id=n.stat_id
ORDER BY db_time_per_sec DESC;
```

| Start SnapID | End SnapID | Start Time | End Time | Interval (s) | DB time/s |
|--------------|------------|----------------|----------------|--------------|-----------|
| ----- | ----- | ----- | ----- | ----- | ----- |
| 83 | 84 | 06.09.07 17:04 | 06.09.07 17:09 | 348 | .92 |
| 49 | 50 | 05.09.07 07:45 | 05.09.07 08:00 | 850 | .02 |
| 25 | 26 | 25.07.07 19:53 | 25.07.07 20:00 | 401 | .01 |

最高相对 DB 时间发生在快照 83 和 84 区间。

25.8 从另一数据库导入 Statspack 数据

如前所述，将来自一个数据库的 Statspack 数据导入到另一个数据库用于分析是很有价值的。让我们假定一个生产数据的模式 PERFSTAT 每个月导出一次，备份到磁带中，然后 Statspack 表被删简以保存空间。被删除的快照可以导入到另一个数据库，以后应该会有调查过往快照的需求，例如检索某一特定语句上个月的执行计划。显然，生产数据库不可能是导入的目标，因为这可能会干扰正在进行的快照捕获过程。

下面将要演示的过程考虑了 Statspack 表 STATS\$IDLE\_EVENT 可能包含某一特定 Statspack 发行版本中丢失的额外等待事件。删除 PERFSTAT 拥有的所有表并导入创建它们这种强力的方式将会删除这个定制。这就是为什么下面演示的方法没有删除任何一个 Statspack 表。相反，它禁用了参照完整性约束，使用 `sptrunc.sql`<sup>①</sup>删简这些表，并使用导入设置 `IGNORE=Y` 将数据导入到已有表中。

在开始之前，需要一个已经使用 `spcreate.sql` 安装了 Statspack 的测试数据库。安装的 Statspack 的版本必须与导出转储中所包含的版本匹配。任何自动快照捕获都应该被禁用。首先，需要从 Statspack 库中删除已有的快照，运行脚本 `sptrunc.sql` 完成删除。

```
$ sqlplus perfstat/secret @sptrunc
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
Warning
~~~~~
Running sptrunc.sql removes ALL data from Statspack tables. You may
wish to export the data before continuing.

About to Truncate Statspack Tables
~~~~~
If you would like to continue, press <return>

Enter value for return:
Entered - starting truncate operation
Table truncated.
...
Truncate operation complete
SQL> EXIT
```

下一步，参照完整性约束被禁用，因为这些约束在导入过程中会被扰乱。`imp` 工具可以在导入过程运行到最后时重新启用它们。通过使用 `SQL*Plus` 生成一个包含语句 `ALTER TABLE DISABLE CONSTRAINT` 的 SQL 脚本可以禁用约束。该脚本被命名为 `gen_disable_sp_constr.sql`，它的内容如下所示：

① 脚本 `sptrunc.sql` 并不删简表 `STATS$IDLE_EVENT`。

```

set linesize 200
set trimout on
set trimspool on
set heading off
set pagesize 0
set feedback off
set heading off
spool disable.sql
select 'ALTER TABLE perfstat.' || table_name || ' DISABLE CONSTRAINT ' ||
constraint_name || ';'
from dba_constraints
where owner='PERFSTAT' and constraint_type='R';
prompt exit
exit

```

接下来一步，以用户 PERFSTAT 运行该脚本。

```

$ sqlplus -s perfstat/secret @gen_disable_sp_constr.sql
ALTER TABLE perfstat.STATS$BG_EVENT_SUMMARY
DISABLE CONSTRAINT STATS$BG_EVENT_SUMMARY_FK;

```

```

...
ALTER TABLE perfstat.STATS$WAITSTAT DISABLE CONSTRAINT STATS$WAITSTAT_FK;

```

SQL\*Plus 将生成的 ALTER TABLE 语句写入到文件 disable.sql。运行脚本 disable.sql 禁用模式 PERFSTAT 中所有参照完整性约束。

```

$ sqlplus -s perfstat/secret @disable
Table altered.

```

```

...
Table altered.

```

此时，该模式已经做好了导入以往快照数据的准备。请注意导入选项 IGNORE=Y 被用于导入到已有表。导入过程将会提示一些 ORA-00001 和 ORA-02264 错误。这些都是无关紧要的，可以忽略掉。

```

$ imp system/secret file=perfstat.dmp full=y ignore=y log=imp.log
Import: Release 9.2.0.1.0 - Production on Wed Sep 19 18:09:00 2007
. importing PERFSTAT's objects into PERFSTAT
...
ORA-00001: unique constraint (PERFSTAT.STATS$IDLE_EVENT_PK) violated
Column 1 smon timer
...
. . importing table "STATS$STATSPACK_PARAMETER" 1 rows imported
IMP-00017: following statement failed with ORACLE error 2264:
"ALTER TABLE "STATS$STATSPACK_PARAMETER" ADD CONSTRAINT "STATS$STATSPACK_P_P"
"IN_CK" CHECK (pin_statpack in ('TRUE', 'FALSE')) ENABLE NOVALIDATE"
IMP-00003: ORACLE error 2264 encountered
ORA-02264: name already used by an existing constraint
...
. . importing table "STATS$WAITSTAT" 126 rows imported
...
About to enable constraints...
Import terminated successfully with warnings.

```

现在我们可以运行 Statspack 报告了。请注意不能使用脚本 spreport.sql，因为它只考虑了当前实例获取的快照。相反，必须使用与它一起的脚本 sprepins.sql。一旦它确定了数据块标识符和实例编号，后者就会被 spreport.sql 调用。而使用 sprepins.sql，需要通过手工设置这两个数字。当运行 sprepins.sql 时，它在 STATS\$DATABASE\_INSTANCE 中列出数据库实例。选择需要的实例，最好使用复制和粘贴得到。

```
SQL> @sprepins
Instances in this Statspack schema
~~~~~
DB Id      Inst Num DB Name      Instance      Host
-----
4273840935      1 PRODDB      IPROD        dbserver3
Enter value for dbid: 4273840935
Using 4273840935 for database Id
Enter value for inst_num: 1
Using 1 for instance number
Completed Snapshots

Instance      DB Name      Snap
Id      Snap Started      Snap
Level Comment
-----
PR02          PROD          104 15 Aug 2006 16:30      10
...
                202 22 Aug 2006 09:15      10
                203 22 Aug 2006 09:30      10
```

此时，该脚本以通常的方式请求开始和结束时的快照标识符以及报告文件名称。一旦输入了实例名称和实例编号，当前实例捕获快照和被导入快照数据的报告就没有任何差别了。

25.9 源代码库

表 25-4 列出了本章源代码文件以及它们的功能。

表25-4 Statspack源代码库

| 文 件 名 | 功 能 |
|---------------------------|---|
| gen_disable_sp_constr.sql | 生成一个SQL脚本，禁用模式PERFSTAT中所有参照完整性约束 |
| snap_by_cpu_util.sql | 列出高CPU使用的快照区间 |
| snap_by_db_time.sql | 列出高DB时间的快照区间 |
| sp_sqltext.sql | PL/SQL管道表函数，用于从Statspack库搜索语法正确（即没有强制换行符）的SQL语句 |
| sp_sqltext_get.sql | 调用管道表函数SITE_SYS.SP_SQLTEXT的SQL脚本 |
| sp_sqltext_join.sql | 从Statspack库检索每一次执行运行时间超过1秒的语句 |
| sp_translate_hv.sql | 将一个来自Oracle10g SQL跟踪文件的散列值翻译为旧散列值，供Statspack使用，以及通过查询STATS\$SQL_PLAN_USAGE将其翻译为SQL标识符（SQL_ID）供AWR使用 |
| sp_used_indexes.sql | 识别一个模式中使用的索引，需要快照级别6或更高级别 |
| sp_valid_intervals.sql | 使用分析函数LAG检索有效快照区间的视图 |

(续)

| 文 件 名 | 功 能 |
|----------------------|---|
| sprepins_fix_10g.sql | 修复Oracle10g中错误会话级别报告 (bug 5145816), 替换了原始文件sprepins.sql |
| sprepins_fix_9i.sql | 修复Oracle9i中错误会话级别报告 (bug 5145816), 替换原始文件sprepins.sql |
| sql_fulltext.sql | 用于从V\$SQL中检索一个通过散列值识别的SQL语句的SQL*Plus脚本 (需要至少是Oracle10g) |
| statspack_purge.sql | Oracle9i的快照清除PL/SQL过程 |

AWR 和 SQL 跟踪文件都用来捕获与 SQL 和 PL/SQL 执行文件相关的性能数据。这两个数据源可被用来解释那些在性能问题调查中经常出现的问题，但这点并没有文档说明。将 SQL 跟踪文件和 AWR 整合起来，就可以找出一些不同的执行计划是否或在什么时候被用于一条特定的语句。此外，与 EXPLAIN PLAN 相反，当某些语句的计划不在 SQL 跟踪文件中时，AWR 对执行计划来说是一个可靠的来源。

26.1 检索执行计划

Statspack 10.2 在获取 SQL\_ID 的同时，也为 SQL 语句全文获取其老散列值和新散列值（分别为 V\$SQL.OLD\_HASH\_VALUE 和 V\$SQL.HASH\_VALUE）。而 AWR 则和 Statspack 10.2 不同，因为 AWR 只是从 V\$SQL 中获取 SQL\_ID。在 Oracle10g 中，对过去的执行计划和 SQL 跟踪文件捕获的语句统计数据恢复上有一点小问题。原因是 Oracle10g 的跟踪文件在 V\$SQL 中不包含 SQL\_ID。Oracle11g SQL 跟踪文件包含 sqlid 这个新的参数（见 24.4.2 节），它对应于 V\$SQL 中的 SQL\_ID。这样，SQL 语句全文向 SQL\_ID 的映射问题已经在 Oracle11g 中得到解决。

注意到执行计划只在游标关闭时才会转向 SQL 跟踪文件，这样就会导致在跟踪文件里没有针对某些语句的执行计划。如果这些语句因为执行计划确实不存在而从共享池中超时退出，AWR 或者 Statspack（见 25.6 节）是恢复该计划的唯一选择。有时候，优化器会为超时的相同语句选择不同的计划。一个执行计划可能导致一个适当的响应时间，而另一个的响应时间则不理想。下面要介绍的程序显示了如何通过 SQL 跟踪和 AWR 恢复 SQL 语句的所有执行计划。25.1.1 节描述了 HR 示例模式中表的 5 路连接，此处我们把它作为一个例子来讲述。在 Oracle11g 之前的版本，我们需要通过为 SQL 语句确定 SQL\_ID 来启动，因为这些版本不会写入 SQL 跟踪文件。AWR 从 V\$SQL.SQL\_FULLTEXT 中获取 SQL 语句全文作为字符大对象 (CLOB)，而该操作是通过使用 DBMS\_LOB 查找 SQL 语句全文或部分来实现的。使用 AWR 获取的语句存储在 WRH\$\_SQLTEXT 这个数据字典基表中，可以通过 DBA\_HIST\_SQLTEXT 视图 (awr\_sqltext.sql 脚本) 来访问。

```
SQL> SET LONG 1048576
SQL> COLUMN sql_text FORMAT a64 WORD_WRAPPED
SQL> SELECT sql_id, sql_text
FROM dba_hist_sqltext
```

```
WHERE dbms_lob.instr(sql_text, '&pattern', 1, 1) > 0;
Enter value for pattern: FROM hr.employees emp, hr.employees mgr
```

```
SQL_ID          SQL_TEXT
-----
1yw85nghurbkk  SELECT emp.last_name, emp.first_name, j.job_title,
                d.department_name, l.city,
                l.state_province, l.postal_code, l.street_address, emp.email,
                emp.phone_number, emp.hire_date, emp.salary, mgr.last_name
                FROM hr.employees emp, hr.employees mgr, hr.departments d,
                hr.locations l, hr.jobs j
                WHERE emp.manager_id=mgr.employee_id
                AND emp.department_id=d.department_id
                AND d.location_id=l.location_id
                AND emp.job_id=j.job_id
```

在得到 SQL\_ID 之后,我们现在就可以查找捕获该 SQL 语句的 AWR 快照了。DBA\_HIST\_SQLSTAT 视图不仅包括该快照标识,而且还可以访问执行统计信息、执行计划的散列值和使用过的优化器环境 (awr\_sqlstat.sql 脚本)。

```
SQL> SELECT st.snap_id,
to_char(sn.begin_interval_time,'dd. Mon yy hh24:mi') begin_time,
st.plan_hash_value, st.optimizer_env_hash_value opt_env_hash,
round(st.elapsed_time_delta/1000000,2) elapsed,
round(st.cpu_time_delta/1000000,2) cpu,
round(st.iowait_delta/1000000,2) iowait
FROM dba_hist_sqlstat st, dba_hist_snapshot sn
WHERE st.snap_id=sn.snap_id
AND st.sql_id='1yw85nghurbkk'
ORDER BY st.snap_id;
```

| SNAP_ID | BEGIN_TIME | PLAN_HASH_VALUE | OPT_ENV_HASH | ELAPSED | CPU | IOWAIT |
|---------|------------------|-----------------|--------------|---------|-----|--------|
| 72 | 13. Oct 07 21:39 | 4095786543 | 611815770 | 1.28 | .05 | 1.21 |
| 73 | 13. Oct 07 21:42 | 4095786543 | 611815770 | .32 | .06 | .27 |
| 73 | 13. Oct 07 21:42 | 3985860841 | 3352456078 | 1.82 | .38 | 1.60 |
| 81 | 15. Oct 07 11:24 | 4095786543 | 611815770 | .16 | .06 | .10 |

通过查询结果可以看到,对同一 SQL\_ID“1yw85nghurbkk”,PLAN\_HASH\_VALUE 列和 OPT\_ENV\_HASH 列的值并不相同,这证明同一 SQL 语句使用了多个执行计划,并且该语句在运行时使用不同的优化器参数设置。实际上,有一个优化器使用的参数叫做 OPTIMIZER\_INDEX\_COST\_ADJ, 它的值从 100 (默认) 到 10 000 不等,与显示的效果有直接的关系。OPTIMIZER\_INDEX\_COST\_ADJ 的增长导致优化器认为索引访问的成本要高 100 倍。因此,优化器选择了全表扫描和散列连接的执行计划,而不是索引访问和嵌套循环。

有两种方法从 AWR 库中恢复执行计划:

- 管道表表函数 DBMS\_XPLAN.DISPLAY\_AWR
- SQL 语句的 AWR 报告脚本 \$ORACLE\_HOME/rdbms/admin/awrsqrpt.sql

除非你想获取查询块的名称在提示中使用 (他们并没有显示在 AWR 报告中), 否则第二种

方法是最佳选择，因为它不仅包含某特定 SQL\_ID 的所有执行计划，而且还包括执行的统计信息。调用 DBMS\_XPLAN.DISPLAY\_AWR 需要 SQL\_ID、计划散列值和数据库标识符 (V\$DATABASE.DBID) 作为输入参数。前两个参数的值已经从 DBA\_HIST\_SQLSTAT 中获得，所以纯粹的数据库标识符必须在 DBMS\_XPLAN 能被调用前查询。

```
SQL> SELECT dbid FROM v$database;
      DBID
-----
2870266532
SQL> SELECT * FROM
TABLE (dbms_xplan.display_awr('1yw85nghurbkk', 4095786543, 2870266532, 'ALL'));
PLAN_TABLE_OUTPUT
-----
SQL_ID 1yw85nghurbkk
-----
SELECT emp.last_name, emp.first_name, j.job_title, d.department_name, l.city,
...
Plan hash value: 4095786543

PLAN_TABLE_OUTPUT
-----
| Id | Operation                | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT          |      |       |       |    9 (100)|          |
|  1 |   NESTED LOOPS             |      |    105 | 18060 |    9 (12)| 00:00:01 |
...
Query Block Name / Object Alias (identified by operation id):
-----
  1 - SEL$1

PLAN_TABLE_OUTPUT
-----
  5 - SEL$1 / EMP@SEL$1
...
```

awrsqrpt.sql 这个 AWR 报告脚本请求 SQL\_ID 和开始、结束快照标识符，而这些值最初是从 DBA\_HIST\_SQLSTAT 中获得的。图 26-1 描绘了一个 SQL\_ID 对应两个执行计划的 HTML 格式的 AWR SQL 报告。

26.2 小结

从 AWR 中为使用 SQL 跟踪的 SQL 语句恢复执行计划，对从 SQL 跟踪文件中重新获得已不存在的执行计划或者对比当前和过去的执行计划来说非常有用。久而久之，执行计划可能会由于以下的这些原因而改变：

- 优化器参数的变化
- 更新的优化器统计信息(DBMS\_STATS 或 ANALYZE)

- ❑ 软件升级或降级
- ❑ 以前没有分区的表分区或索引分区
- ❑ 绑定变量窥视

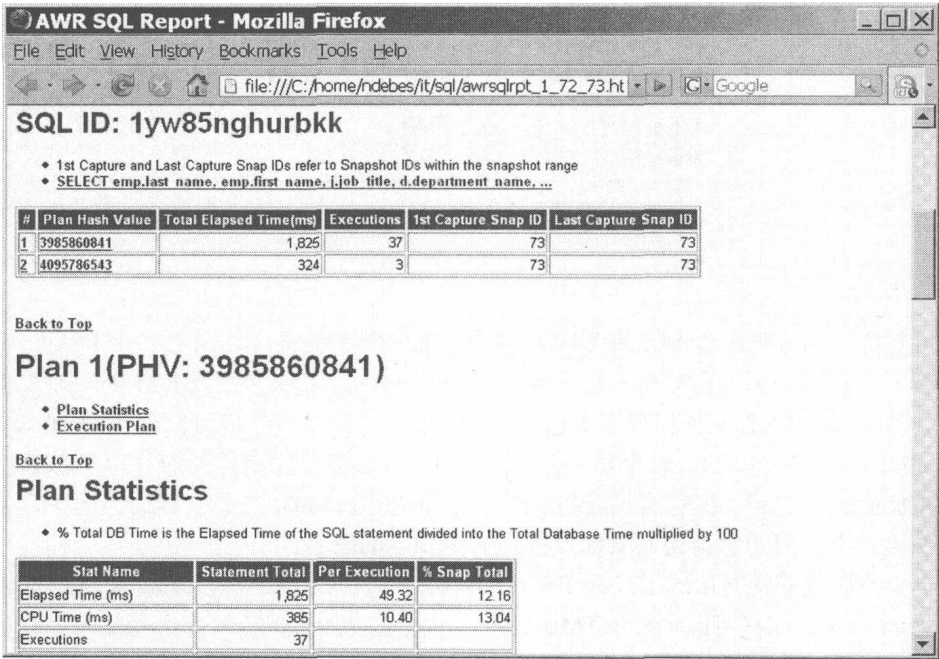


图 26-1 一条语句对应多个计划的 AWR SQL 报告

在优化器环境中计算并在 AWR 库中存储的散列值，可用来作为执行计划在实例或会话级别因为不同的参数设置而改变的证据。因为更新的优化器统计数据可能会导致计划发生变化，我推荐使用 DBMS\_STATS.EXPORT\_SCHEMA\_STATS 打包程序在统计数据在覆盖之前对其进行保存。DBMS\_STATS 包包括用来创建统计数据表的 CREATE\_STAT\_TABLE 程序。在 Oracle10g 中，可使用 DBMS\_STATS.LOCK\_SCHEMA\_STATS 来防止对模式进行自动统计数据的重计算。

26.3 源代码库

表 26-1 列出了本章的源代码文件和它们的功能。

表26-1 扩展SQL跟踪和AWR源代码库

| 文 件 名 | 功 能 |
|-----------------|---|
| awr_sqltext.sql | 从基于SQL语句全文子集的DBA_HIST_SQLTEXT中获取SQL_ID和完整的语句全文。SQL_ID可作为输入被传送到awrsqrpt.sql这个AWR脚本 |
| awr_sqlstat.sql | 通过某SQL_ID获取捕捉SQL语句的AWR快照。一些执行的统计信息，如运行时间、CPU使用率等，它们与执行计划和优化器环境的散列值一起显示出来 |

ESQLTRCPROF 扩展 SQL 跟踪分析器

有人可能会说为分析扩展 SQL 跟踪文件开发分析工具在前几年很流行。TKPROF——Oracle 自己的分析器 (profiler)，自从第一次支持 Oracle9i 的等待事件之后只有过很小的改进。相比 TKPROF，其他工具对跟踪文件提供了更进一步的分析。首先有一个 Oracle 免费的 Trace Analyzer (查看 Metalink note 224270.1)。这个基于 PL/SQL 的工具创建了非常详细的 HTML 报告。它的不足是需要安装在数据库中安装，但这有可能阻碍对产品系统的 ad-hoc 分析。理想情况下，它运行在跟踪的同一实例中，因为它会将对象标识符转换为相应的段名。

另一个免费而且非常有用的工具是 TVD\$XTAT，由 Apress 会员作家 Christian Antognini 编写。它是一个 Java 程序，同样可以创建 HTML 报告。但是它更加有特色的一个功能是可以创建等待事件直方图。同 Trace Analyze 不一样，它的运行不依赖于 Oracle DBMS 实例，而且更快。它不能将等待事件中的数据库对象标识符转换为表和索引的段名。

ESQLTRCPROF 是 Oracle9i、Oracle10g、Oracle11g 扩展 SQL 跟踪文件的基于 Perl 的分析器。ESQLTRCPROF 能够解析未在文档中说明的扩展 SQL 跟踪文件格式。它能够为整个 SQL 跟踪文件和跟踪文件中的每个游标计算出一份资源配置文件。它看起来好像是唯一支持将思考时间作为独立的一部分纳入响应时间的分析器。它通过递归调用深度也能够分解 (break down) 运行时间、CPU 时间以及等待时间，从而可以指出大部分响应时间是消耗在哪一个调用深度。同提到的其他分析器一样，ESQLTRCPROF 是 TKPROF 的一个替代品，因为它只解决了 Oracle 公司官方的 SQL 跟踪分析工具的一些不足。

27.1 分类等待事件

最终目的是为了能从扩展 SQL 跟踪文件自动创建资源配置文件。即使在第 24 章已经讨论过文件格式，但为了达到这个目的还需要更多准备。暂时还没有讨论的一个问题是将等待事件分类为数据库内调用等待事件 (intra database call wait event) 和数据库间调用等待事件 (inter database call wait event)。这对得到正确的响应时间计算是必不可少的。数据库内调用等待事件在数据库调用上下文中发生。完成数据库调用需要执行的代码路径不仅包括 CPU 消耗，还有可能引起等

待类似磁盘、锁、队列等的资源。在数据库调用中花费的等待时间由数据库内调用等待事件计算。类似这样的等待事件的例子有 latch free、enqueue、db file sequential read、db file、scattered read 和 buffer busy waits。事实上,大多数等待事件是数据库内调用等待事件。当 DBMS 服务器在等待接收下一个数据库调用时会发生数据库间调用等待事件。也就是说,由于客户端不发送请求,DBMS 服务器处于空闲状态。

根据 Millsap 和 Holt ([MiHo 2003], 第 88 页),以下等待事件是数据库间调用等待事件:

- SQL\*Net message from client
- SQL\*Net message to client
- pmon timer
- smon timer
- rdbms ipc message

在这些等待事件中,pmon timer、smmon timer 和 rdbms ipc message 只发生在后台进程中。因此,与调优应用程序相关的数据库间调用等待事件只有 SQL\*Net message from client 和 SQL\*Net message to client<sup>①</sup>。

图 27-1 是扩展 SQL 跟踪文件中的数据库调用、等待事件、事务项 (XCTEND) 的图形化表示。 X 轴表示时间 (t), Y 轴表示递归调用深度 (dep)。数据库间调用等待事件用白体字体和灰色背景表示。注意,在递归调用深度 0,所有这些等待事件要么同一个游标号码 n ($n>0$) 相关联,要么是默认游标 0。

27

27.2 计算响应时间和统计信息

根据 Millsap 和 Holt ([MiHo 2003]),SQL 跟踪文件代表的响应时间 R 被定义为花费在数据库调用 (e 值) (在递归调用深度为 0 时, $dep=0$) 的运行时间的总和,再加上所有从数据库间调用等待事件的 ela 值的总和。当处理数据库调用时累计的等待时间 (ela 值) 都归结到产生这个等待的数据库调用的参数 e 中。前一节中讨论的等待事件分类在计算 R 时用到了。花费在等待数据库内调用等待事件的时间一定不能加到 R 中,因为这有可能产生重复计算。数据库调用的 e 值已经包含了所有数据库内调用等待事件的等待时间。在跟踪文件结束之前发出数据库调用,这是数据库内调用等待事件的 WAIT 项出现在 PARSE、EXEC 和 FETCH 项之前的原因。

① 在 [MiHo 2003] 中的列表也包含 pipe get 和 single-task message。我省略了 pipe get, 因为我的测试显示这一等待事件只有在 PL/SQL 包 DBMS\_PIPE 上制定执行数据库调用时才发生。如果调用 DBMS\_PIPE.RECEIVE\_MESSAGE 过程中有一个非 0 超时,花费在等待信息的时间计算在等待事件 pipe get 之内,而且归结到相关 EXEC 项的参数 e 中。因此,pipe get 是一个数据库内调用等待事件。

自从 Oracle 9i 之后,所有 ORACLE DBMS 的完成都仅使用双任务,也就是服务器进程和客户端进程都是独立的任务,各自运行在自己的地址空间。因此不再使用等待事件 single-task message (同样可以参考 Metalink note 62227.1)。

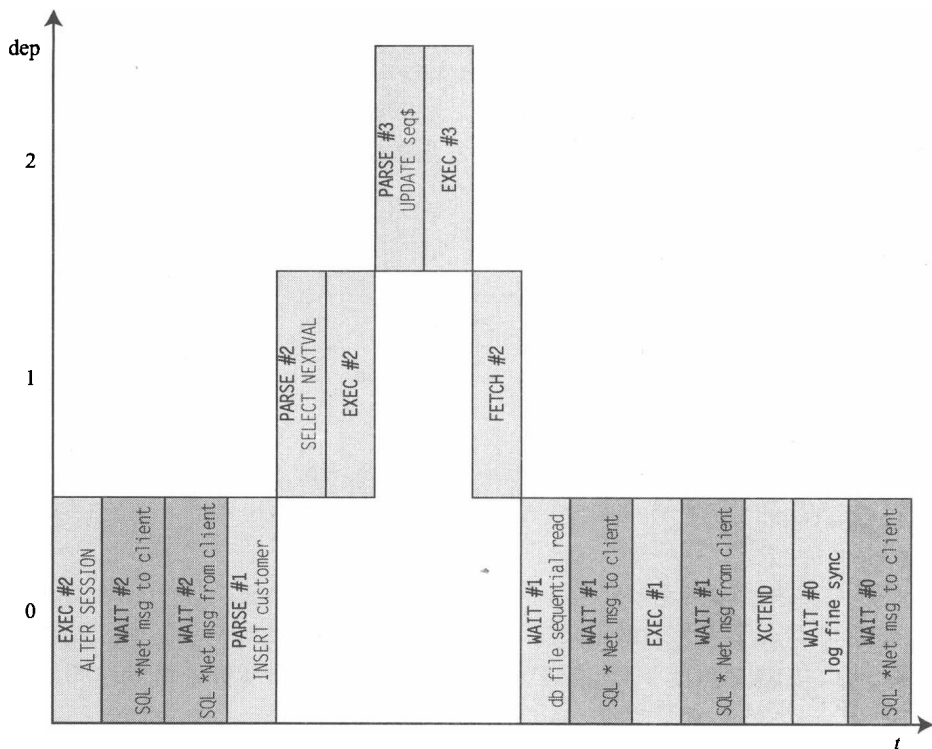


图 27-1 递归调用深度和数据库内调用等待事件<sup>①</sup>

递归调用深度为 0 之外的实时统计信息，比如一致性读、物理写和 db 块获得，都归结到递归调用深度为 0 的 PARSE、EXEC 和 FETCH 调用。同数据库内调用等待事件的 ela 值一样，一定不能重复计算。为了全面了解扩展 SQL 跟踪分析器是如何从跟踪文件计算资源配置文件的，我将通过一个小的跟踪文件手动计算一些数字，以使你明白。

27.2.1 案例研究

这个案例研究基于一个小的名为 insert\_customer.pl 的 Perl DBI 程序。这个程序插入单独一行到表 CUSTOMER。每一个顾客都有唯一的一个数字标识，这个数字由序列器 CUSTOMER\_ID\_SEQ 产生。使用 INSERT 触发器将序列器中的下一个值赋值到表的 ID 列。我故意使用 NOCACHE 选项创建了序列器，因此这会导致递归 SQL（也就是说 dep 的值要大于 0）<sup>②</sup>。这个案例有两个作用。

- (1) 用来证明如何计算响应时间 R。
- (2) 用来证明递归调用深度大于 0 的实时统计信息都归结到更高递归调用深度的统计信息中，而且最终会归结到递归调用深度为 0。

<sup>①</sup> 在等待事件名称中使用缩写 msg 代替单词 message。
<sup>②</sup> 在任何现实应用程序中不要使用 NOCACHE，因为它会降低性能。

请记住直接被数据库客户端执行的语句的递归调用深度为 0。这个程序暂停两次以允许用户查询动态性能视图 V\$SESSTAT，一次是启用 SQL 跟踪之前，另一次是刚好在从 ORACLE 实例断开之前。因为 V\$SESSTAT 并不受之前讨论的重复计算的影响，它保留会话级统计信息的精确表达。

27.2.2 运行 Perl 程序

重复这个案例，打开两个终端窗口。一个是为了运行 Perl 程序，另一个是为了查询 V\$SESSTAT。在第一个终端窗口运行 insert\_customer.pl。<sup>①</sup>

```
$ insert_customer.pl
Hit return to continue
```

在程序等待输入的时候，在第二个窗口查询 V\$SESSTAT。

```
SQL> SELECT n.name, s.value
FROM v$sesstat s, v$statname n, v$session se
WHERE s.statistic#=n.statistic#
AND n.name IN ('db block gets', 'consistent gets')
AND s.sid=se.sid
AND se.program LIKE 'perl%';
NAME                                VALUE
-----
db block gets                        0
consistent gets                     195
```

现在在第一个窗口返回了 hit。程序启用 SQL 跟踪，然后插入单独一行。过了一会程序再次请求输入。

```
Hit return to continue
```

这个时候，完成了 INSERT 语句和后来的 COMMIT。对 V\$SESSTAT 的另一查询揭露了 db 块读和一致性读的数字分别上升到 9 和 197。

```
SQL> SELECT n.name, s.value
FROM v$sesstat s, v$statname n, v$session se
WHERE s.statistic#=n.statistic#
AND n.name IN ('db block gets', 'consistent gets')
AND s.sid=se.sid
AND se.program='perl.exe';
NAME                                VALUE
-----
db block gets                        9
consistent gets                     197
```

这个时候也许你第二次收到了 hit 返回值，断开和终止了程序。从最终数字中减去初始数字得到 9 个 db 块读和 2 个一致性读。理论上，扩展 SQL 跟踪文件应该包含相同的数字，但是却存在一点小偏差。

这一测试的跟踪文件非常小，以致可以手动计算。注意你第一次运行测试产生的跟踪文件也

<sup>①</sup> 使用 Perl DBI 环境变量 DBI\_USER、DBI\_PASS 和 DBI\_DSN 指定用户名、密码、数据源和连接字符串（查看第 22 章）。

许会比以下重新产生的文件要大，因为需要加载字典缓存和库缓存。除了一些头部信息，完整的跟踪文件（加上了行编号）描述如下：

```

1 Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production
2 With the Partitioning, Oracle Label Security, OLAP and Data Mining options
3
4 *** ACTION NAME:() 2007-11-20 15:39:38.546
5 *** MODULE NAME:(insert_customer.pl) 2007-11-20 15:39:38.546
6 *** SERVICE NAME:(TEN.oradbpro.com) 2007-11-20 15:39:38.546
7 *** SESSION ID:(44.524) 2007-11-20 15:39:38.546
8 =====
9 PARSING IN CURSOR #2 len=68 dep=0 uid=61 oct=42 lid=61 tim=789991633616
hv=740818757 ad='6be3972c'
10 alter session set events '10046 trace name context forever, level 8'
11 END OF STMT
12 EXEC #2:c=0,e=98,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=789991633607
13 WAIT #2: nam='SQL*Net message to client' ela= 5 driver id=1413697536 #bytes=1
p3=0 obj#=-1 tim=789991638001
14 WAIT #2: nam='SQL*Net message from client' ela= 569 driver id=1413697536 #bytes=1
p3=0 obj#=-1 tim=789991638751
15 =====
16 PARSING IN CURSOR #1 len=87 dep=0 uid=61 oct=2 lid=61 tim=789991639097
hv=2228079888 ad='6cad992c'
17 INSERT INTO customer(name, phone) VALUES (:name, :phone)
18 RETURNING id INTO :id
19 END OF STMT
20 PARSE #1:c=0,e=84,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=789991639091
21 =====
22 PARSING IN CURSOR #2 len=40 dep=1 uid=61 oct=3 lid=61 tim=789991640250
hv=1168215557 ad='6cbaf25c'
23 SELECT CUSTOMER_ID_SEQ.NEXTVAL FROM DUAL
24 END OF STMT
25 PARSE #2:c=0,e=72,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=1,tim=789991640243
26 EXEC #2:c=0,e=62,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=1,tim=789991641167
27 =====
28 PARSING IN CURSOR #3 len=129 dep=2 uid=0 oct=6 lid=0 tim=789991641501
hv=2635489469 ad='6bdb9be8'
29 update seq$ set increment$=:2,minvalue=:3,maxvalue=:4,cycle#=:5,order$=:6,
cache=:7,highwater=:8,audit$=:9,flags=:10 where obj#=:1
30 END OF STMT
31 PARSE #3:c=0,e=68,p=0,cr=0,cu=0,mis=0,r=0,dep=2,og=4,tim=789991641494
32 EXEC #3:c=0,e=241,p=0,cr=1,cu=2,mis=0,r=1,dep=2,og=4,tim=789991642567
33 STAT #3 id=1 cnt=1 pid=0 pos=1 obj=0 op='UPDATE SEQ$ (cr=1 pr=0 pw=0
time=195 us)'
34 STAT #3 id=2 cnt=1 pid=1 pos=1 obj=102 op='INDEX UNIQUE SCAN I_SEQ1
(cr=1 pr=0 pw=0 time=25 us)'
35 FETCH #2:c=0,e=1872,p=0,cr=1,cu=3,mis=0,r=1,dep=1,og=1,tim=789991643213
36 WAIT #1: nam='db file sequential read' ela= 33297 file#=4 block#=127140 blocks=1
obj#=54441 tim=789993165434
37 WAIT #1: nam='SQL*Net message to client' ela= 5 driver id=1413697536 #bytes=1

```

```

p3=0 obj#=54441 tim=789993165747
38 EXEC #1:c=1500000,e=1525863,p=1,cr=2,cu=8,mis=0,r=1,dep=0,og=1,tim=789993165858
39 WAIT #1: nam='SQL*Net message from client' ela= 232 driver id=1413697536 #bytes=1
p3=0 obj#=54441 tim=789993166272
40 XTEND rlbk=0, rd_only=0
41 WAIT #0: nam='log file sync' ela= 168 buffer#=5320 p2=0 p3=0
obj#=54441 tim=789993166718
42 WAIT #0: nam='SQL*Net message to client' ela= 2 driver id=1413697536 #bytes=1
p3=0 obj#=54441 tim=789993166829
43 *** 2007-11-20 15:39:49.937
44 WAIT #0: nam='SQL*Net message from client' ela= 9864075 driver id=1413697536
#bytes=1 p3=0 obj#=54441 tim=790003031019
45 XTEND rlbk=0, rd_only=1
46 STAT #2 id=1 cnt=1 pid=0 pos=1 obj=53073 op='SEQUENCE CUSTOMER_ID_SEQ
(cr=1 pr=0 pw=0 time=1878 us)'
47 STAT #2 id=2 cnt=1 pid=1 pos=1 obj=0 op='FAST DUAL (cr=0 pr=0 pw=0 time=15 us)'

```

27.2.3 计算统计信息

递归调用深度为 0 之外的数据库调用统计信息都归结到递归调用深度为 0 的统计信息中。为了计算跟踪文件中的 db 块读的总数目，我们必须只考虑 dep=0 的 PARSE、EXEC 和 FETCH 项的 cu 参数。数据库调用参数 cu（也就是当前读）相当于统计信息 db 块读。在第 38 行（cu=8），递归调用深度为 0 的仅有的 cu 值大于 0。同从 V\$SESSTAT 检索得到的 db 块读取值差 1。注意在递归调用深度为 1 和更低时已经发生了 3 次 db 块读（第 35 行）。在递归调用深度为 2 时发生了 2 次 db 块读（第 32 行）。db 块读的总数目是 9（正如查询 V\$SESSTAT 一样）的事实证明了较低级的数据库调用统计信息都会归结到递归调用深度为 0 的统计信息中。在任一递归调用深度的 cu 值的总和是 13。如果递归调用深度为 n 的 cu 值并没有包括递归调用深度为 n+1 的 cu 值，我们在 V\$SESSTAT 中至少可以看到 13 个 db 块读。

27.2.4 计算响应时间

响应时间 R 定义为 dep=0 的所有 e 值的总和，以及从数据库间调用等待事件的所有 ela 值的总和（[MiHo 2003]，94 页）。

$$R = \sum_{\text{dep}=0} e + \sum_{\text{inter db call}} \text{ela}$$

dep=0 的所有 e 值的总和是从第 12、20、38 行导出来的。

$$\sum_{\text{dep}=0} e = 98 + 84 + 1\,525\,863 = 1\,526\,045$$

SQL\*Net message from client 和 SQL\*Net message to client 在跟踪文件中是数据库间调用等待事件。数据库间调用等待事件的所有 ela 值的总和是从第 13、14、37、39、42、44 行导出来的。

$$\sum_{\text{inter db call}} \text{eal} = 5 + 569 + 5 + 232 + 2 + 9\,864\,075 = 9\,864\,888$$

因为 e 和 ela 值都是以微秒为单位，R 用秒表示的结果如下：

$$R = (1\,526\,045 + 9\,864\,888) / 1\,000\,000 = 11.390\text{ s}$$

为了检查估计的代码路径是否充分插桩，从跟踪文件中的第一个 tim 值和最后一个 tim 值计算运行时间总是一个好的想法。跟踪文件最开始的最低时间戳是 789 991 633 616。第 44 行的最高时间戳是 790 003 031 019。因此，跟踪文件覆盖的间隔是（以秒为单位）：

$$(790\,003\,031\,019 - 789\,991\,633\,616) / 1\,000\,000 = 11.379\text{ s}$$

如果覆盖跟踪文件的 R 和运行时间之间存在很大的差别，说明 ORACLE 内核并没有适当地插桩部分代码路径。

花费在数据库调用的运行由 CPU 消耗和等待组成。理想情况下，每一个递归调用深度为 0 的数据库调用的 e 值会等于 CPU 使用量 (c) 加上数据库间调用等待时间 (ela)。

$$\sum_{\text{dep}=0} \text{e} = \sum_{\text{dep}=0} \text{c} + \sum_{\text{intra db call}} \text{ela}$$

实际中，在这两个值之间通常会存在一个差值。但是并不知道哪个地方的时间并没有计算到 c 中而花费了 ela。在整个跟踪文件中对响应时间做出贡献的这个未知贡献者 (U) 可以按照如下方式计算：

$$U = \sum_{\text{dep}=0} \text{e} - \sum_{\text{dep}=0} \text{c} - \sum_{\text{intra db call}} \text{ela}$$

在这个例子中，这个差值非常小，稍微大于 7 ms。

$$[(98 + 84 + 1\,525\,863) - 1\,500\,000 - 33\,297] / 1\,000\,000 = -0.007\,252\text{ s}$$

理所当然，利用扩展 SQL 跟踪分析器自动计算这些数字是更加便利的事情，比如使用 ESQTRCPROF。这是下一节的主题。

27.3 ESQTRCPROF 参考

ESQTRCPROF 是一个扩展 SQL 跟踪分析器，是作者用 Perl 编写的。它有以下特点。

- 为整个 SQL 跟踪文件计算资源配置文件。
- 为跟踪文件中的每一个 SQL 或 PL/SQL 语句计算资源配置文件。
- 由于网络往返和思考时间，将数据库内调用等待事件 SQL\*Net message from client 归为不可避免延迟类型。
- 按总运行时间（解析、执行和提取运行时间的总和，再加上除了思考时间之外的数据库内调用等待时间）排序语句。
- 从 STAT 项抽取执行计划。
- 各种统计数据的计算，例如物理读、一致读、db 块读、每秒处理事务数以及缓冲区缓存命中率。

- 通过独立队列分配队列等待。
- 通过独立锁分解锁等待。
- 总结报告中的 SQL 语句的散列值, 以快速找到跟踪文件中的语句, 然后用 Statspack 集成。
- 总结报告中的 SQL 语句的标识符 (sqlid), 以快速找到跟踪文件中的语句, 然后用 AWR (需要 Oracle11g 跟踪文件作为输入) 集成。
- 总结报告中的模块、行为和递归调用深度。

从数据库服务器的角度来看, 思考时间是数据库客户端没有在 DBMS 上发出任何命令的积累的运行时间。也就是说, 客户端不会发送任何请求到 DBMS 服务器。默认情况下, ESQLTRCPROF 将 SQL\*Net message from client 超过 5 ms 的任何等待分类为假的等待事件 think time。对于这类等待, 在资源配置文件中思考时间会被认为是响应时间的一部分。

27.3.1 命令行选项

ESQLTRCPROF 接受改变其进程的 3 个选项。如果没有任何参数就调用它, 它将打印关于用法、支持选项和含义的信息。

```
$ esqltrcprof.pl
Usage: esqltrcprof.pl -v -r <ORACLE major release>.<version> -t <think time in
milliseconds> <extended sql trace file>
-v verbose output; includes instances of think time
-r value must be in range 8.0 to 10.2
```

超过 SQL\*Net message from client 分类为思考时间的阈值用选项 -t (也就是 threshold) 配置 (单位为微秒)。默认阈值 (5 ms) 通常适合 LAN (Local Area Networks, 局域网), 但是需要通过 WAN (Wide Area Network, 广域网) 的数据库客户端的跟踪文件中升级。为了废弃 SQL\*Net message from client 分类, 也许是为了同 TKPROF 报告或其他工具比较数字的安全着想, 将思考时间的阈值设置为一个大值, 比如 10 小时 (36 000 000 ms)。这样可以让假等待事件思考时间从资源配置文件中消失。

选项 -v (也就是 verbose) 用来打印思考时间的实例。当使用 -v 时, 每次 ESQLTRCPROF 遇到跟踪文件的思考时间实例, 都会在跟踪文件中打印思考时间长度和位置的信息。模块和动作也会出现在信息中, 以指出在代码路径的哪个地方检测到思考时间。

```
Found 9.864 s think time (Line 64, Module 'insert_customer.pl' Action 'undefined')
```

因为 ESQLTRCPROF 也支持 Oracle8, 但 Oracle8 只分析 e、c 和 ela 值的厘秒精度, ESQLTRCPROF 决定了跟踪文件头部的计时数据单元。从锁号映射到锁名同样是同版本有关的。因此, ESQLTRCPROF 不能处理在头部缺少版本信息的跟踪文件。Oracle10g 的 TRCSESS 实用工具创建缺少头部的跟踪文件。当传递这样的文件作为 ESQLTRCPROF 的输入时, 会出现以下错误信息, 并退出:

```
$ esqltrcprof.pl insert_cust.trcse
No trace file header found reading trace file up to line 9. ORACLE release unknown.
Please use switch -r to specify release
```

```
Usage: esqltrc_profiler.pl -v -r <ORACLE major release>.<version> -t <think time in
milliseconds> <extended sql trace file>
-v verbose output; includes instances of think time
-r value must be in range 8.0 to 10.2
```

提供选项-r(也就是 release)是为了让 ESQLTRCPROF 知道创建跟踪文件的 ORACLE DBMS 的版本。

```
$ esqltrcprof -r 10.2 insert_cust.trcsess
Assuming ORACLE release 10.2 trace file.
Resource Profile
=====

Response time: 11.391s; max(tim)-min(tim): 11.397s
...
```

27.3.2 ESQLTRCPROF 报告部分

对于跟踪文件中的每个语句, ESQLTRCPROF 报告都包括会话级的资源配置文件、会话级的统计信息、语句级资源配置文件和语句级统计信息。这个报告以名为“Resource Profile”的会话级部分开始。

1. 会话级资源配置文件

在以下代码例子中会重新产生前一学习案例中使用的跟踪文件的资源配置文件。由人工计算出来的 R 和 db 块读的数字同 ESQLTRCPROF 报告的数字相同。7 ms 的未知时间同手工计算的同样匹配。这个报告包括最大时间戳和最小时间戳 (tim) 的差值, 它以“max (tim) - min (tim)”的形式出现在响应时间下面。

```
$ esqltrcprof.pl ten_ora_6720_insert_customer.pl.trc
ORACLE version 10.2 trace file. Timings are in microseconds (1/1000000 sec)
Warning: WAIT event 'log file sync' for cursor 0 at line 61 without prior PARSING IN

CURSOR #0 - all waits for cursor 0 attributed to default unknown statement
with hash value -1
Resource Profile
=====

Response time: 11.391s; max(tim)-min(tim): 11.397s
Total wait time: 9.898s
-----

Note: 'SQL*Net message from client' waits for more than 0.005s are considered think
time
Wait events and CPU usage:
```

| Duration | Pct | Count | Average Wait Event/CPU Usage/Think Time |
|----------|--------|-------|---|
| 9.864s | 86.59% | 1 | 9.864075s think time |
| 1.500s | 13.17% | 8 | 0.187500s total CPU |
| 0.033s | 0.29% | 1 | 0.033297s db file sequential read |

```

0.001s 0.01%      2 0.000400s SQL*Net message from client
0.000s 0.00%      1 0.000168s log file sync
0.000s 0.00%      3 0.000004s SQL*Net message to client
-0.007s -0.06%      unknown
-----
11.391s 100.00% Total response time
Total number of roundtrips (SQL*Net message from/to client): 3

CPU usage breakdown
-----
parse CPU: 0.00s (3 PARSE calls)
exec CPU: 1.50s (4 EXEC calls)
fetch CPU: 0.00s (1 FETCH calls)

```

会话级资源配置文件以 CPU 使用的详细信息结束。在 `exec CPU` 分类中，改变了很多行的会话有最高 CPU 消耗，与此同时，总是处于读的会话消耗了大部分 CPU 使用，以 `fetch CPU` 对此作出说明。

2. 会话级统计信息

报告的下一部分包括统计信息。每秒的事务次数是基于 `R` 和 `XCTEND` 项计算出来的。使用类似 `XCTEND rlbk=0, rd_only=0` 形式的项作为事务结尾标记。事务结尾标记数目除以 `R` 就得到了每秒的事务次数。事务是什么呢？当然，这是完全依赖应用程序的。不同的应用程序执行不同类型的事务。只有在比较调优前后的同一应用程序性能时才使用这个数字。注意，在优化之后代码路径的响应时间有可能得到很大的改进，而每秒的事务次数却有可能降低。这种情况发生在消除一些不必要的提交之后。

```

Statistics:
-----
COMMITs (read write): 1 -> transactions/sec 0.088
COMMITs (read only): 1
ROLLBACKs (read write): 0
ROLLBACKs (read only): 0
rows processed: 1
cursor hits (soft parses): 3
cursor misses (hard parses): 0
consistent gets: 2
db block gets: 8
physical reads: 1
buffer cache hit ratio: 90.00%

Physical read breakdown:
-----
single block: 1
multi-block: 0

Latch wait breakdown
-----

```

Enqueue wait breakdown (enqueue name, lock mode)

表 27-1 总结了所有计算的统计信息以及它们的来源。

表27-1 统计信息

| 统计信息 | 来 源 |
|------------|---|
| 缓冲区高速缓存命中率 | PARSE、EXEC和FETCH项中的cr、cu和p |
| 一致性读 | PARSE、EXEC和FETCH项中的cr |
| 游标命中和缺失 | PARSE项中参数mis |
| db块读 | PARSE、EXEC和FETCH项中cu |
| 排队等待 | nam='enqueue'的WAIT项或者'enq: enqueue_details', 而enqueue_details包括队
列名称和简短描述 (比如nam='enq:TX -contention') |
| 开锁等待 | nam='latch free'的WAIT项或者'latch: latch_details', 而latch_details包括开
锁名称和简短描述 (比如nam='latch: cache buffers chains') |
| 物理读 | PARSE、EXEC和FETCH项中的p |
| 处理行 | PARSE、EXEC和FETCH项中的r |
| 单块读和多块读 | P3 (Oracle9i及以前版本) 或者等待事件的块db file sequential read和db file
scattered read |
| 每秒的事务次数 | R和XCTEND r1bk=0, rd_only=0 |

在前一学习案例中使用的小跟踪文件并没有包含队列或开锁的任何等待。因此在前面代码例子中开锁等待和排队等待部分是空的。以下是 ESQLTRCPROF 报告的节录，它导致同时是排队信息的跟踪数据库会话变为 Advanced Queuing 队列表。这个节录描述了会话同时竞争开锁和排队。

```
Latch wait breakdown
-----
row cache objects      waits:    1 sleeps:    0
library cache pin      waits:    2 sleeps:    0
commit callback allocation  waits:    1 sleeps:    0
cache buffers chains   waits:    2 sleeps:    1
dml lock allocation    waits:    1 sleeps:    0
library cache          waits:    9 sleeps:    4
enqueue hash chains    waits:    1 sleeps:    0

Enqueue wait breakdown (enqueue name, lock mode)
-----
HW,X waits:           4
TX,S waits:           7
```

在独占模式 (X) 请求排队 HW，而在共享模式 (S) 请求排队 TX。当段需要增长时请求排队 HW（高水位线），而在串行化写访问行和数据库块开始部分 ITL（Interested Transaction List，感兴趣的事务表）时使用排队 TX。

在 Oracle10g 中，有可能使用固定视图 V\$ENQUEUE\_STATISTICS 产生一系列 208 排队类型和名称，伴随着也会有一些描述以及一个或多个请求排队的理由。表 27-2 包含了一些更加常见的队列类型的列表。

表27-2 排队类型

| 排队类型 | 名 称 | 描 述 | 请求排队的原因 |
|------|--|---------------------------|----------|
| CF | Controlfile Transaction (控制文件事务) | 同步访问控制文件 | 争用 |
| CI | Cross-Instance Call Invocation (跨实例调用) | 调整跨实例调用 | 争用 |
| DX | Distributed Transaction (分布式事务) | 紧密耦合分布式事务处理系统串行化 | 争用 |
| HV | Direct Loader High Water Mark (直接载入高水位线标识) | 在并行插入期间, 使用该锁代理高水位线标识 | 争用 |
| HW | Segment High Water Mark (代码段高水位标识) | 在并行插入期间, 使用该锁代理高水位线标识 | 争用 |
| JQ | Job Queue (作业队列) | 使用该锁阻止运行单个任务产生多个实例 | 争用 |
| JS | Jobs Scheduler (作业调度器) | 使用该锁用于在崩溃的 RAC 实例上运行时恢复任务 | 任务恢复锁 |
| JS | Job Scheduler | 使用该锁阻止任务乱跑 | 任务运行同步锁 |
| JS | Job Scheduler | 在间隔调度队列中锁 | 队列锁 |
| JS | Job Scheduler | 非全局排队调度器 | 本地排队调度器 |
| JS | Job Scheduler | 当清除q内存时获得的锁 | 清除q内存时的锁 |
| JS | Job Scheduler | 当增加用户到事件q时得到的锁 | 增加事件用户 |
| JS | Job Scheduler | 当从事件q删除用户时得到的锁 | 删除事件用户 |
| JS | Job Scheduler | 当打开/关闭窗口时得到的锁 | 窗口锁 |
| JS | Job Scheduler | 在事件通知阶段得到的锁 | 事件通知 |
| JS | Job Scheduler | 同步访问任务高速缓存 | 争用 |
| SQ | Sequence Cache (序列高速缓存) | 保证只有一个进程能填满顺序高速缓存的锁 | 争用 |
| SS | Sort Segment (排序段) | 确保在并行DML操作创建的排序段没有过早地被清除掉 | 争用 |
| ST | Space Transaction (空间事务) | 在字典管理表空间中同步空间管理活动 | 争用 |
| TM | DML | 同步访问对象 | 争用 |
| TS | Temporary Segment (临时代码段) | 串行化访问临时段 | 争用 |
| TX | Transaction | 在分割阶段在索引上占用的锁, 以组织进行其他操作 | 索引争用 |
| TX | Transaction | 事务占用的锁, 以允许其他事务等待它 | 争用 |
| TX | Transaction | 事务在一个特殊行上占用的锁, 以阻止其他事务修改它 | 行锁争用 |
| TX | Transaction | 分配一个ITL项以开始一个事务 | 分配ITL项 |
| UL | User-defined | 被用户应用程序使用的锁 | 争用 |
| US | Undo Segment (撤销代码段) | 在撤销段上执行DDL占用的锁 | 争用 |

① 等待排队 UL 发生在应用程序完成包 DBMS\_LOCK 的同步化时。

以下查询产生这个表的行：

```
SQL> SELECT eq_type, eq_name, req_description, req_reason
FROM v$enqueue_statistics
WHERE eq_type IN ('CF', 'CI', 'DX', 'HV', 'HW', 'JQ', 'JS', 'SQ',
'SS', 'ST', 'TM', 'TS', 'TX', 'UL', 'US')
ORDER BY eq_type;
```

如果你遇到了一个既没有归档又没有出现在表 27-2 的排队，查询 V\$ENQUEUE\_STATISTICS 会让你对这个排队用来做什么以及你正在优化的应用程序为什么会请求这个排队有初步的了解。Oracle9i 和 Oracle10g 的很多排队类型都是相同的。因此你很有可能发现同样适用于 Oracle9i 的信息。

3. 语句级资源配置文件

ESQLTRCPROF 报告的最后一部分包括遇到的 SQL 语句、语句级资源配置文件、语句级统计信息和执行计划。每个语句都单独用一小节描述，以语句的散列值和语句的总运行时间开始。散列值通常是唯一的，而且对于快速在跟踪文件中找到语句很有帮助。此外，可以使用散列值从 Statspack 库中（查看第 25 章）检索到某一特定语句当前和过去的执行计划。

● 语句排序顺序

同 TKPROF 不同，ESQLTRCPROF 没有用于排序的命令行选项。初看会认为这点是劣势，然而这却是 ESQLTRCPROF 的强大之处之一。ESQLTRCPROF 总是用每个语句的总运行时间进行排序。在扩展 SQL 跟踪文件级别，这意味着 ESQLTRCPROF 通过将某一特定游标的 PARSE、EXEC 和 FETCH 项 e 值的总和加上数据库内调用等待事件（除了 SQL\*Net message from client waits 之外，它被分类为思考时间）的 ela 值来计算总运行时间。忽略了思考时间，因为它并不是 DBMS 实例问题的一部分。这种方法优于 TKPROF 使用的排序语句的方法，具体表现在以下三个值中。

- TKPROF 要么按照执行运行时间（execla）进行排序，要么按照提取运行时间（fchela）排序，但是并没有按照 PARSE、EXEC 和 FETCH 项的总运行时间。这种排序的目的，它同时花费在哪里并不相关，因为响应时间才是关键。
- 在排序语句时，TKPROF 忽略了数据库内调用等待事件。
- TKPROF 减去了递归资源使用（dep 值大于 0），而只报告递归调用深度为 0 的语句。这让按照客户端执行代价最大的语句进行排序变得不现实。

在我看来，不减去递归资源使用会更加合理。这样由客户端发送的语句（dep=0）在排序的顺序上会比同一语句产生的递归语句靠前。因此，这就是最应该执行哪个语句和成为更高递归调用深度的证据。理想情况下，ESQLTRCPROF 报告应该要创建一个单独的部分用以描述语句间的递归关系，但是在当前版本是无能为力的。

ESQLTRCPROF 在处理同游标标号为 0 以及缺少 PARSING IN CURSOR 项的游标有关的跟踪文件项时有一个非常特殊的特性。当一个应用程序正处于处理中期就开始跟踪时，有可能出现后一种情况（缺少 PARSING IN CURSOR 项的游标）。有可能存在一个游标有 EXEC、FETCH、WAIT 项而没

有相应 PARSING IN CURSOR 项。因此，不能确定这种游标的 SQL 语句文本<sup>①</sup>。游标 0 绝对不会有 PARSING IN CURSOR 项。通过 OCI 访问 LOB 也有可能同游标 0 相关。不像 TKPROF 在单独语句部分一样忽略同这种游标有关的项，ESQLTRCPROF 为说明任何这样的跟踪文件项而定义了“散列值”为-1（不存在）的默认游标。前一部分学习案例的基于跟踪文件的 ESQLTRCPROF 报告的剩余部分在以下代码示例中。注意，即使这样简短的跟踪文件都包括了大部分思考时间和同步日志文件的 10 秒钟，同游标 0 相关。

Statements Sorted by Elapsed Time (including recursive resource utilization)

Hash Value: 2228079888 - Total Elapsed Time (excluding think time): 1.526s

INSERT INTO customer(name, phone) VALUES (:name, :phone)

RETURNING id INTO :id

| DB Call | Count | Elapsed | CPU | Disk | Query | Current | Rows |
|---------|-------|---------|---------|------|-------|---------|------|
| PARSE | 1 | 0.0001s | 0.0000s | 0 | 0 | 0 | 0 |
| EXEC | 1 | 1.5259s | 1.5000s | 1 | 2 | 8 | 1 |
| FETCH | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| Total | 2 | 1.5259s | 1.5000s | 1 | 2 | 8 | 1 |

| Wait Event/CPU Usage/Think Time | Duration | Count |
|---------------------------------|----------|-------|
| total CPU | 1.500s | 2 |
| db file sequential read | 0.033s | 1 |
| SQL*Net message from client | 0.000s | 1 |
| SQL*Net message to client | 0.000s | 1 |

Hash Value: 1168215557 - Total Elapsed Time (excluding think time): 0.002s

SELECT CUSTOMER\_ID\_SEQ.NEXTVAL FROM DUAL

| DB Call | Count | Elapsed | CPU | Disk | Query | Current | Rows |
|---------|-------|---------|---------|------|-------|---------|------|
| PARSE | 1 | 0.0001s | 0.0000s | 0 | 0 | 0 | 0 |
| EXEC | 1 | 0.0001s | 0.0000s | 0 | 0 | 0 | 0 |
| FETCH | 1 | 0.0019s | 0.0000s | 0 | 1 | 3 | 1 |
| Total | 3 | 0.0020s | 0.0000s | 0 | 1 | 3 | 1 |

| Wait Event/CPU Usage/Think Time | Duration | Count |
|---------------------------------|----------|-------|
|---------------------------------|----------|-------|

① 第 28 章出现的测量脚本 sp\_capture.sql、awr\_capture.sql 创建了一个级别为 2 的错误栈转储，因为这种转储可能包含丢失的 SQL 语句文本。

total CPU 0.000s 3

Execution Plan:

Step Parent Rows Row Source

```

-----
1      0      1 SEQUENCE  CUSTOMER_ID_SEQ (cr=1 pr=0 pw=0 time=1878 us)
(object_id=53073)
2      1      1 FAST DUAL  (cr=0 pr=0 pw=0 time=15 us)

```

Hash Value: 740818757 - Total Elapsed Time (excluding think time): 0.001s

alter session set events '10046 trace name context forever, level 8'

| DB Call | Count | Elapsed | CPU | Disk | Query | Current | Rows |
|---------|-------|---------|---------|------|-------|---------|------|
| PARSE | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| EXEC | 1 | 0.0001s | 0.0000s | 0 | 0 | 0 | 0 |
| FETCH | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| Total | 1 | 0.0001s | 0.0000s | 0 | 0 | 0 | 0 |

| Wait Event/CPU Usage/Think Time | Duration | Count |
|---------------------------------|----------|-------|
|---------------------------------|----------|-------|

```

-----
SQL*Net message from client      0.001s      1
SQL*Net message to client        0.000s      1
total CPU                        0.000s      1

```

Hash Value: 2635489469 - Total Elapsed Time (excluding think time): 0.000s

update seq\$ set increment\$=:2,minvalue=:3,maxvalue=:4,cycle#=:5,order\$=:6,cache=:7,
highwater=:8,audit\$=:9,flags=:10 where obj#=:1

| DB Call | Count | Elapsed | CPU | Disk | Query | Current | Rows |
|---------|-------|---------|---------|------|-------|---------|------|
| PARSE | 1 | 0.0001s | 0.0000s | 0 | 0 | 0 | 0 |
| EXEC | 1 | 0.0002s | 0.0000s | 0 | 1 | 2 | 1 |
| FETCH | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| Total | 2 | 0.0003s | 0.0000s | 0 | 1 | 2 | 1 |

| Wait Event/CPU Usage/Think Time | Duration | Count |
|---------------------------------|----------|-------|
|---------------------------------|----------|-------|

```

-----
total CPU                        0.000s      2

```

Execution Plan:

Step Parent Rows Row Source

```

-----
1      0      1 UPDATE  SEQ$ (cr=1 pr=0 pw=0 time=195 us)
2      1      1 INDEX UNIQUE SCAN I_SEQ1 (cr=1 pr=0 pw=0 time=25 us)
(object_id=102)

```

Hash Value: -1 - Total Elapsed Time (excluding think time): 0.000s

Cursor 0 - unknown statement (default container for any trace file entries relating to cursor 0)

| DB Call | Count | Elapsed | CPU | Disk | Query | Current | Rows |
|---------|-------|---------|---------|------|-------|---------|------|
| PARSE | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| EXEC | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| FETCH | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| Total | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |

| Wait Event/CPU Usage/Think Time | Duration | Count |
|---------------------------------|----------|-------|
| think time | 9.864s | 1 |
| log file sync | 0.000s | 1 |
| SQL*Net message to client | 0.000s | 1 |
| total CPU | 0.000s | 0 |

● SQL标识符和Oracle11g跟踪文件

ESQLTRCPROF 的最新版本支持 Oracle11g 扩展跟踪文件。它在每一个语句级资源配置文件的头部就包括 SQL 标识符、模块、行为和递归调用深度。在 Oracle10g 引进了 SQL 标识符，但是直到 Oracle11g 才在 SQL 跟踪文件中使用。因此，只有基于 Oracle11g 跟踪文件的 ESQLTRCPROF 才包含 SQL 标识符。以下是一个例子：

Statements Sorted by Elapsed Time (including recursive resource utilization)

Hash Value: 1256130531 - Total Elapsed Time (excluding think time): 0.102s
 SQL Id: b85soyd5dy123 Module 'insert\_perf5.pl' Action 'undefined'
 Dependency Level: 0

```
INSERT INTO customer(id, name, phone)
VALUES (customer_id_seq.nextval, :name, :phone)
RETURNING id INTO :id
```

| DB Call | Count | Elapsed | CPU | Disk | Query | Current | Rows |
|---------|-------|---------|---------|------|-------|---------|------|
| PARSE | 1 | 0.0002s | 0.0000s | 0 | 0 | 0 | 0 |
| EXEC | 10 | 0.0956s | 0.0313s | 5 | 27 | 36 | 10 |
| FETCH | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| Total | 11 | 0.0958s | 0.0313s | 5 | 27 | 36 | 10 |

| Wait Event/CPU Usage/Think Time | Duration | Count |
|---------------------------------|----------|-------|
| total CPU | 0.031s | 11 |
| db file sequential read | 0.021s | 3 |
| SQL*Net message from client | 0.006s | 10 |

```
SQL*Net message to client          0.000s      10
Execution Plan:
Step Parent      Rows Row Source
-----
 1         0         0 LOAD TABLE CONVENTIONAL (cr=3 pr=4 pw=4 time=0 us)
 2         1         1 SEQUENCE  CUSTOMER_ID_SEQ (cr=3 pr=1 pw=1 time=0 us)
(object_id=15920)
```

报告中的 SQL 标识符使得同 AWR 的集成变得更加容易。有可能使用它作为 AWR SQL 报告脚本 awrsqrpt.sql 或 DBMS\_XPLAN 的输入。

```
SQL> SELECT * FROM TABLE (dbms_xplan.display_awr('b85s0yd5dy1z3'));
PLAN_TABLE_OUTPUT
-----
SQL_ID b85s0yd5dy1z3
-----

INSERT INTO customer(id, name, phone) VALUES (customer_id_seq.nextval,
:name, :phone)          RETURNING id INTO :id

Plan hash value: 2690979981
```

```
-----
| Id | Operation                | Name                | Cost |
-----
|  0 | INSERT STATEMENT          |                     |     1 |
-----

PLAN_TABLE_OUTPUT
-----
|  1 | LOAD TABLE CONVENTIONAL |                     |     |
|  2 | SEQUENCE                  | CUSTOMER_ID_SEQ    |     |
-----

Note
----
- cpu costing is off (consider enabling it)
```

可以从应用程序插桩项中提取模块和行为。如果在跟踪文件中没有这样的项，模块和行为都被标记为 undefined。可以使用 Oracle9i 的 APPNAME 项格式从 Oracle9i 跟踪文件中提取模块和行为。递归调用深度从 PARSING IN CURSOR 项的 dep 参数提取。

27.4 小结

ESQLTRCPROF 分析器接受扩展 SQL 跟踪文件作为输入，计算出会话级和语句级资源配置文件。据我所知，它是将 SQL\*Net message from client 分类为思考时间和在客户端和数据库服务器之间不可避免的网络往返的唯一分析器。假等待事件 think time 被定义为 nam='SQL\*Net message from client' 的 WAIT 项，与此同时 ela 值达到一个可以设置的阈值。记住，DBA 不能容忍降低由数据库客户端积累的思考时间。偶尔 DBMS 不会把最终用户或开发者认为的数据库性

能问题归结为性能问题。如果你遇到一个资源配置文件有很突出的思考时间——50%或者更多，思考时间的平均持续时间超过几百微秒，而在语句级 ESQLTRCPROF 报告中没有开销很高的 SQL 语句，而这可能是性能问题。如果你遇到这种情况——平均思考时间经常处于几秒的范围内，你应该要检查一下思考时间的平均持续时间以防止由于阈值太低错误地将 SQL\*Net message from client 分类为思考时间。当然，小心度量间隔也很重要。选择数据库客户端不活跃的时期作为度量间隔是不恰当的。

ESQLTRCPROF 试图解决一些 TKPROF 的不足。鉴于 Millsap 和 Holt 的书[MiHo 2003]在 2003 年才面世，但 Oracle11g 的 TKPROF 版本仍然没有包含资源配置文件，也没有按照对响应时间的贡献排序等待事件，这听起来有点令人惊奇。TKPROF 也不能够报告数据库调用和等待事件的平均持续时间。

ESQLTRCPROF 考虑了影响语句的响应时间的数据库内调用等待事件和数据库间调用等待事件。语句在递归调用深度为 0 执行多次会导致每次执行的网络往返。因此，一个语句执行一次只花费很少时间但执行了很多次，会比一个慢的语句（执行很少次）导致更多的响应时间。为了按照对响应时间的实际贡献情况排序跟踪文件的 SQL 或者 PL/SQL 语句，必须同样考虑数据库内调用等待事件。这就是为什么 ESQLTRCPROF 定义语句的总运行时间为数据库内调用等待事件（同语句相关）的所有 e 值和所有 ela 值的总和的原因。使用总运行时间排序 ESQLTRCPROF 语句级报告部分的语句。由于这个新的方法，ESQLTRCPROF 不需要像 TKPROF 那样使用排序选项。

27.5 源代码库

表 27-3 列出了本章的源码文件以及它们的功能。

表27-3 ESQLTRCPROF源码库

| 文 件 名 | 功 能 |
|--------------------|--|
| esqltrcprof.pl | 这个perl程序从扩展SQL跟踪文件创建了一个会话级资源配置文件以及语句级资源配置文件。它将等待事件SQL*Net message from client分类为思考时间和真正的网络延迟。它也计算每秒的事务和其他指标，以及将等待事件latch free和enqueue分解为独立的门锁和排队 |
| insert_customer.pl | 这个perl程序使用INSERT RETURNING插入一行到表中。被这个程序引用的创建数据库对象的DDL包含在这个perl源代码中 |
| insert_cutomer.trc | 从运行inser_customer.pl得来的扩展SQL跟踪文件 |

MERITS 性能优化方法建立在一个对扩展 SQL 跟踪文件的复杂评估上。ESQLTRCPROF 是一个扩展 SQL 跟踪工具，它能解析未在文档中说明的跟踪文件格式，用于 MERITS 方法的评估阶段。MERITS 方法把那些未在文档中说明的特性主要用在评估、再现和外推阶段。

从本质上讲，MERITS 方法是一个用于解决性能问题的框架。该方法的目标是找出导致响应时间缓慢的根本原因，并随后修改参数、数据库对象或应用程序代码，直到满足性能要求。

28.1 MERITS 方法简介

MERITS 是一种性能优化方法，包括六个阶段，并在一些阶段中依赖于未在文档中说明的特性。MERITS 这个缩写来源于以下 6 个阶段：

- (1) 测量 (Measurement)
- (2) 评估 (Assessment)
- (3) 再现 (Reproduction)
- (4) 改进 (Improvement)
- (5) 推断 (Extrapolation)
- (6) 安装 (Installation)

任何性能评测工具的第一步应该包括对应用程序或代码路径的详尽测量，此过程很慢（第一阶段）。对测量数据的评估放在第二阶段。在某些情况下，评估可能就已经发现了性能问题的根源。更复杂的情况可能需要使用测试系统中的一个测试用例来再现（第三阶段）。如果一条 SQL 语句的执行需要过多的时间，那么测试用例包括再现该 SQL 语句响应时间。第四阶段与改善应用程序、代码路径或测试用例的响应时间相关。这可能包括创建新的索引、改变优化器的参数、改变 SQL 语句、使用 DDL 改变数据库对象、介绍之前未使用的特性（例如，分区选项、存储概要、SQL 配置），等等。对代码路径进行改进后仍采用原先的方法进行测量。把对原先代码路径的测量数据与第四阶段改进后的测量数据对比，可用来推断性能改进的量级（第五阶段）。换句话说，可以使用测试用例对在第一阶段测量过的代码路径改进后的效果进行预测。如果被认为是充分的改进，必要的修改在某个时候需要被批准并安装到目标（产品）系统中。讨论 MERITS 方法每个阶段的全部细节可以单独用一本书来阐述。但是，我在每个阶段提供了足够的信息，从

而能让你将该方法作为一种性能优化任务的框架。

28.2 测量

由于扩展 SQL 跟踪是数据库会话花费时间最多的地方，并且资源配置文件可通过扩展 SQL 跟踪数据编译产生，所以，该数据源是测量的核心。然而，扩展 SQL 跟踪文件不能提供对应用程序、系统或数据库管理系统实例的全面认识。一些不为扩展 SQL 跟踪文件所涵盖的内容如下：

- 操作系统级别的负载（如 I/O 瓶颈、分页、网络拥塞、CPU 等待）
- ORACLE 数据库管理系统参数
- 会话统计信息（V\$SESSSTAT）
- 冲突的数据库会话

要想对系统有完整的认识，我建议使用 sar、iostat、vmstat 和 top 等命令来记录操作系统级别的活动。关于数据库管理系统，我主张采用 Statspack 或 AWR 快照，并和扩展 SQL 跟踪文件一样跨相同的时间间隔。Statspack 快照应该包括跟踪会话（STATSPACK.SNAP 参数 i\_session\_id）。如果首选 AWR，那么需要使用活动会话历史报告来获得与该会话相关的其他信息。有时候有必要采取多次测量并计算平均值，从而平衡响应时间的波动。AWR 和 Statspack 报告都包括一系列非默认值的初始化参数。活动会话历史报告包括一部分题为 Top Blocking Sessions 的冲突会话。

测量工具

本部分给出了两个 SQL 脚本，它们都可以作为会话和实例级别的测量工具。awr\_capture.sql 脚本基于 AWR 和 ASH，而 sp\_capture.sql 基于 Statspack。两个脚本都需要 SYSDBA 权限，它们不调用任何操作系统工具来收集操作系统的统计信息。然而，Oracle10g 的 Statspack 报告在 Host CPU 和 Memory Statistics 两部分包括操作系统级别 CPU 和内存的统计信息，而 AWR 报告则包括标题为 Operating System Statistics 的一部分内容。

AWR 和 ASH 都包括在额外费用的 Diagnostics Pack 中。ASH 的缺点是它没有提供一个资源配置文件，因为它通过抽样生成。一个会话级别的 Statspack 报告包括一个基本的资源配置文件，尽管该报告不使用该资源配置文件。V\$SESSION\_EVENT 和 V\$SESSSTAT 负责对资源配置的计算，是会话级别数据的基础。ASH 报告里有 Top Service/Module、Top SQL using literals、Top Blocking Sessions 和 Top Objects/Files/Latches 这些有趣的部分，而会话级别的 Statspack 和 TKPROF 报告没有。

1. 扩展SQL跟踪，AWR和ASH

awr\_capture.sql 脚本临时设置隐藏参数 \_ASH\_SAMPLE\_ALL=TRUE，让 ASH 能对空闲的等待事件采样，从而获得更好的诊断表现。然后该脚本会拍下 AWR 快照，并为该会话启用 12 级的 SQL 跟踪，从而展现性能问题。接下来，该脚本会询问用户应跟踪该会话的时间。有两种使用该脚本的方法。

- 跟踪该会话到预定的时间间隔，如 300 或 600 秒，这是通过进入该间隔的期望长度来实现的。该脚本调用 DBMS\_LOCK.SLEEP 在指定数量秒后暂停，拍下另一张 AWR 快照，并退出

SQL 跟踪。

- 使用一个基于事件的方法来控制测量的时间间隔。通过这种方法，当遇到关于时间间隔长度的问题时，键入 0，但不敲回车。等待直到一个事件（如最终用户报告你跟踪的会话已经返回一个结果集）发生时，敲回车键。由于等待时间间隔为 0，所以该脚本立即拍下另一张 AWR 快照，并退出 SQL 跟踪。

在测量间隔结束时，该脚本会调用 DBMS\_WORKLOAD\_REPOSITORY 包自动创建 HTML 格式的 AWR 和 ASH 报告，这些文件创建在当前目录中。请注意，已在文档中说明的 ASH 脚本不能只针对一个特定的会话进行报告。以下为使用该脚本操作的例子：

```
SQL> @awr_capture
Please enter SID (V$SESSION.SID): 143
Please enter a comment (optional): slow-app
SPID  SID  SERIAL#  USERNAME  MACHINE      SERVICE_NAME
-----
18632 143      333  NDEBES    WORKGROUP\  TEN.oradbpro.com
                        DBSERVER
PROGRAM  MODULE  ACTION  CLIENT_IDENTIFIER
-----
perl.exe perl.exe NULL    NULL
Oracle pid: 20, Unix process pid: 18632, image: oracleTEN1@dbserver1.oradbpro.com
Statement processed.
Extended SQL trace file:
/opt/oracle/obase/admin/TEN/udump/ten1_ora_18632.trc
Begin snapshot: 95
Please enter snapshot interval in seconds (0 to take end snapshot immediately): 300
End snapshot: 96
Begin time: 07.Sep.2007 03:51:33; End time: 07.Sep.2007 03:56:39; Duration (minutes)
: 5.1
ASH Report file: slow-app-SID-143-SERIAL-333-ash.html
AWR Report file: slow-app-SID-143-SERIAL-333-awr.html
```

2. 扩展 SQL 跟踪和会话级别的 Statspack 快照

Statspack 的测量脚本像是 AWR 的变体，它同样是对一个会话进行跟踪，并为该会话启用 12 级的 SQL 跟踪。该脚本没有拍摄 AWR 快照，相反，拍摄的是 Statspack 快照，其中包括了会话级别数据。该脚本使用 ORADEBUG 来获得扩展 SQL 跟踪文件的名称。下面的例子描述了用于决定捕获时间间隔的基于事件的方法。这意味着捕获不会持续一个预定的时间间隔。

```
SQL> @sp_capture
Please enter SID (V$SESSION.SID): 139
SPID  SID  SERIAL#  USERNAME  MACHINE      SERVICE_NAME
-----
19376 139      41757  NDEBES    WORKGROUP\  TEN.oradbpro.com
                        DBSERVER
PROGRAM  MODULE  ACTION  CLIENT_IDENTIFIER
-----
perl.exe  insert_perf5.pl NULL    NULL
```

```

Oracle pid: 16, Unix process pid: 19376, image: oracleTEN1@dbserver1.oradbpro.com
Extended SQL trace file:
/opt/oracle/obase/admin/TEN/udump/ten1_ora_19376.trc
Begin snapshot: 291

```

此时，脚本等待输入。如果你期望在事件发生后尽快结束捕获间隔，那么等待该事件并进入 0。

```

Please enter snapshot interval in seconds (0 to take end snapshot immediately): 0
End snapshot: 301
Begin time: 07.Sep.2007 06:20:07; End time: 07.Sep.2007 06:24:44; Duration (minutes)
: 4.6

```

当捕获一结束，通过运行\$ORACLE\_HOME/rdbms/admin/spreport.sql 脚本会为开始和结束的快照编号生成一份 Statspack 报告。以下是一份包括使用 sp\_capture.sql 脚本捕获的会话级别数据的报告。这些会话具体的部分包括 Session Wait Events、Session Time Model Stats（仅针对 Oracle10g 和之后的版本）和 Session Statistics<sup>①</sup>：

```

Snapshot      Snap Id      Snap Time      Sessions Curs/Sess Comment
~~~~~
Begin Snap:    291 07-Sep-07 06:20:07    21      6.7 SID-139-perl.exe-in
End Snap:      301 07-Sep-07 06:24:44    21      6.7 SID-139-perl.exe-in
Elapsed:                4.62 (mins)
Session Wait Events DB/Inst: TEN/TEN1 Snaps: 291-301
Session Id:        139 Serial#:    41757
-> ordered by wait time desc, waits desc (idle events last)

Event                               Waits   Timeouts   Total Wait   WT
-----
Waits
/txn
-----
log file switch completion          3         1      1 345.605667
3.0
db file sequential read             13         0      0 4.61123077
13.0
control file sequential read        26         0      0 .954307692
26.0
control file parallel write         6         0      0 2.56516667
6.0
db file scattered read               4         0      0    3.045
4.0
log file sync                       2         0      0    5.1105
2.0
latch: library cache                 1         0      0      0
1.0
SQL*Net message from client        283,496         0      72 .254345627
#####
SQL*Net message to client          283,496         0      1 .004825056
#####

```

① 在 Session Statistics 部分，每个事务统计信息都被忽略了。

Session Time Model Stats DB/Inst: TEN/TEN1 Snaps: 291-301
Session Id: 139 Serial#: 41757
-> Total Time in Database calls 182.9s (or 182854062us)
-> Ordered by % of DB time desc, Statistic name

| Statistic | Time (s) | % of DB time |
|-------------------------------------|----------|--------------|
| DB CPU | 179.2 | 98.0 |
| sql execute elapsed time | 176.1 | 96.3 |
| sequence load elapsed time | 32.1 | 17.6 |
| PL/SQL execution elapsed time | 6.7 | 3.7 |
| parse time elapsed | 1.4 | .8 |
| hard parse elapsed time | 0.4 | .2 |
| hard parse (sharing criteria) elaps | 0.0 | .0 |
| repeated bind elapsed time | 0.0 | .0 |
| DB time | 180.6 | |

Session Statistics DB/Inst: TEN/TEN1 Snaps: 291-301
Session Id: 139 Serial#: 41757

| Statistic | Total | per Second |
|-----------------------------------|------------|------------|
| active txn count during cleanout | 948 | 3 |
| consistent gets | 31,335 | 113 |
| cpu used by this session | 18,117 | 65 |
| parse time cpu | 30 | 0 |
| physical read total bytes | 655,360 | 2,366 |
| redo size | 92,965,180 | 335,614 |
| session pga memory max | 4,718,592 | 17,035 |
| sql*net roundtrips to/from client | 283,503 | 1,023 |
| workarea executions - optimal | 56 | 0 |

一个资源配置文件可能来自与会话具体相关的部分，通过使用 Session Time Model Stats 部分的 DB CPU 和 Session Wait Events 部分的等待事件。在 Oracle9i 中，CPU 的消耗使用 CPU used by this session 统计量来表示。响应时间等同于开始和结束的快照覆盖的间隔（报告开头的 Elapsed 值）。测量间隔时间和总等待时间加上 DB CPU 的差值被计算并被报告为 unknown。unknown 的时间占了很大的比例，可能意味着该会话并没有得到 CPU 信息或捕获的代码路径没有完全被插装。也可能是目前已经有一段很长的等待，如 SQL\*Net message from client，它尚未被纳入到 V\$SESSION\_EVENT 中。表 28-1 显示了使用这种方法计算得到的资源配置文件。

表28-1 来自会话级别Statspack报告的资源配置文件

| 响应时间占用者 | 时 间 | 比 例 |
|-----------------------------|---------|--------|
| DB CPU | 179.2 s | 64.64% |
| SQL*Net message from client | 72.0 s | 26.97% |
| unknown | 24.0 s | 8.66% |
| log file switch completion | 1.0 s | 0.03% |

(续)

| 响应时间占用者 | 时 间 | 比 例 |
|---------------------------|---------|---------|
| SQL*Net message to client | 1.0 s | 0.03% |
| Response time | 277.2 s | 100.00% |

你可以把表 28-1 的资源配置文件和同一个会话的另一个资源配置文件（该资源配置文件使用 ESQLTRCPROF 从跨越几乎<sup>①</sup>相同时间间隔的扩展 SQL 跟踪文件计算产生）进行比较。后一个资源配置文件见下一个代码示例。两份报告的 CPU 使用率（179.2 与 218.31）有明显的差别，然而等待事件的数字几乎是一样的。

```
ORACLE version 10.2 trace file. Timings are in microseconds (1/1000000 sec)
Resource Profile
=====
Response time: 239.377s; max(tim)-min(tim): 271.173s
Total wait time: 74.080s
-----
Note: 'SQL*Net message from client' waits for more than 0.005s are considered think
time
Wait events and CPU usage:
Duration      Pct      Count      Average Wait Event/CPU Usage/Think Time
-----
218.316s  91.20%      904144  0.000241s total CPU
70.953s   29.64%      282089  0.000252s SQL*Net message from client
1.349s    0.56%      282092  0.000005s SQL*Net message to client
1.037s    0.43%         3  0.345606s log file switch completion
0.419s    0.18%        16  0.026201s KSV master wait
0.092s    0.04%         8  0.011520s Data file init write
0.062s    0.03%         2  0.030860s log file switch (checkpoint incomplete)
0.060s    0.03%        13  0.004611s db file sequential read
0.025s    0.01%        26  0.000954s control file sequential read
0.023s    0.01%         3  0.007810s think time
0.016s    0.01%         2  0.007905s rdbms ipc reply
0.015s    0.01%         6  0.002565s control file parallel write
0.012s    0.01%         4  0.003045s db file scattered read
0.010s    0.00%         2  0.005110s log file sync
0.005s    0.00%         2  0.002479s db file single write
0.001s    0.00%         2  0.000364s latch: library cache
0.000s    0.00%         2  0.000243s latch: shared pool
-53.019s -22.15%                unknown
-----
239.377s 100.00% Total response time
```

28.3 评估

询问性能优化的目标应该是评估阶段的第一步。如果目标是不现实的，那么预期的目标需要

① 因为需要一些时间（一般少于 3 秒）来拍 Statspack 快照，所以开始和结束快照的时间间隔不会和 SQL 跟踪文件捕获的结果相同。

加以纠正。有一个目标很重要，而该目标是否可以实现也非常重要。当你对测量阶段收集的数据进行评估时，你能发现该目标是否能实现。通常经理或其他客户只会说“让速度尽可能地快”。好吧，至少你已经询问过。

要问的第二个问题是所面临的问题是否是一个真正的数据库性能问题。这可以从来源于扩展 SQL 跟踪文件的思考时间 ESQTRCPROF。如果说 90% 的响应时间是思考时间，并且所有已执行的 SQL 语句完成得迅速合理，那么没有数据库问题。思考时间表示数据库客户端没有要求服务器来处理任何请求。该应用程序不是空闲就是忙于处理那些并非有求于数据库管理系统实例的指令。由于不可能让一个数据库管理员去减少应用程序的思考时间，所以应用程序开发者必须找出占用过多思考时间的部分。

下一个要分清的问题，是响应时间慢是否是由于过多的 CPU 使用率和等待时间过高。该会话使用的 CPU 统计信息或者 Oracle10g 中的 DB CPU 表示 CPU 使用率。如果等待是主要问题，那么解决依赖于等待事件的类型。等待可能是由于较慢的 I/O 系统（例如，db 文件顺序读取）、争夺（例如，入队、锁存器空闲）或者 CPU 资源不足。如果数据库资源管理器被启用，后者的原因可能通过等待事件反映出来。如何减少那些突出的等待事件的等待时间（Oracle10g 有 878 种不同的等待事件）可以作为另一本专门的性能调优书籍的主题。Shee 等[ShDe 2004]在他们的著作中很好地解决了这一问题。

资源配置文件和性能评估工具

评估阶段的主要目标是从之前测量阶段捕获的数据产生一个资源配置文件。资源配置文件的概念通过 Cary Millsap 和 Jeff Holt 的著作 *Optimizing Oracle Performance* 一书[MiHo 2003]流行起来。我的扩展 SQL 跟踪调试工具 ESQTRCPROF 受到了该出版物的强烈影响。然而，把思考时间加入资源配置文件是我的发明。Cary Millsap 和 Jeff Holt 的研究促进了针对扩展 SQL 跟踪文件的商业调试工具的发展，由 Hotsos Enterprises, Ltd 提供。其他用于获取资源配置文件的工具有 TKPROF 和 ESQTRCPROF，在第 27 章有详细描述。

TKPROF与ESQTRCPROF

TKPROF 生成的报告并不包括资源配置文件。尽管如此，TKPROF 报告包括足够的信息，从而可用于生成资源配置文件。同样也可以使用会话级别的 Statspack 数据。响应时间 R 由非递归和递归语句的运行时间加上数据库间调用等待事件来往（SQL\*Net message from/to client）的等待时间。非递归和递归语句需要被添加到全部的总数中，从而来获得 CPU 使用率和等待事件的总数。测量的时间间隔在 TKPROF 10.2 报告的最后一行被报告为 elapsed seconds in trace file。一旦这些数据被计算成 CPU 使用率和所有等待事件，它们需要排好序并组织成资源配置文件。使用 ESQTRCPROF 更简单，它能自动处理所有工作。TKPROF 的另一个缺点是忽略了散列值，它可以在报告中识别 SQL 语句。散列值可被用来将 SQL 语句与实例级（spreport.sql）和 SQL 语句级（sprepsql.sql）的 Statspack 报告或 V\$视图（V\$SQL.HASH\_VALUE）关联起来。

TKPROF 和 ESQTRAPROF 都报告 CPU 的使用情况。通常高的 CPU 使用率同访问高速缓存中的数以千/百万计的块的 SQL 语句相关。如果发生了这种情况，一个差的执行计划，比如包

含不必要的全表扫描、使用了不合适的连接顺序或者使用了错误的索引,那它就是一个害群之马。必须确认导致高 CPU 使用率的 SQL 语句,而且必须检查它们的执行计划。TKPROF 提供了排序选项 `fcchela` 和 `exceela`,两者可以结合使用。如果你怀疑一个 SELECT 语句是跟踪文件中最耗时的语句,使用 `sort=fcchela,exceela`。否则使用 `sort=exceela,fcchela`。TKPROF 不能排序总运行时间,它由解析、执行和提取三个阶段的运行时间组成。此外,在排序的过程中,TKRPOF 并不考虑同游标相关的数据库内调用等待事件。ESQLTRCPOF 总是排序总运行时间,而且还考虑数据库调用等待事件(比如 SQL\*Net message from/to client)之间的总运行时间,但是不考虑思考时间。这能够保证在进行排序时,一个导致多次回路从而产生很大网络延迟的 SQL 语句,比起一个在数据库调用有相同总运行时间但是没有积累那么多的网络延迟的 SQL 语句,前者会标记为更加耗时。

28.4 重现

在实际的性能优化可能开始之前,必须找到重现当前问题的方法。尽可能接近地重现问题代码路径是至关重要的。取决于性能问题的类型,为了重现问题,在原始环境和测试环境中以下因素必须完全相同。

- 硬件的处理能力。
- 操作系统和版本。
- ORACLE DBMS 版本。
- 初始化参数(在文档中说明的和隐藏的),尤其是优化参数。
- 数据库对象统计信息(即优化器统计信息)。
- 数据库块大小。
- 优化的 SQL 语句的绑定变量和绑定数据类型。
- 存储概要。
- SQL profile (只在 Oracle10g 及以后版本出现)。

当创建数据库对象时,请确保你在表空间中使用的是同原始应用程序相同的 DDL,相同的块大小。可以使用包 `DBMS_METADATA` 来为数据库对象抽取 DDL。当重现有绑定变量的 SQL 语句时,使用同原始语句相同的数据类型的绑定变量。12 级 SQL 跟踪文件在 `BINDS` 部分(参见第 24 章),包括绑定变量值以及绑定变量数据类型。SQL\*Plus 和 PL/SQL 变量有可能用来重现绑定数据类型。最后同时比较重要的一点,请确保没有其他使用者在你的测试系统上运行压力测试或者其他资源密集型的程序,因为这可能导致你的测试出现相反的结果。

28.5 改进

改进性能是一项很重大的任务。如果我们不使用更加强大的硬件(有时,即使使用了强大的硬件也可能无法解决问题),以下的程序也有可能改进性能。

- 改变参数(比如, `DB_CACHE_SIZE`、`PARALLEL_EXECUTION_MESSAGE_SIZE`)。

- 添加索引以避免全表扫描。
- 删除不必要的索引。
- 对表和/或索引进行分区。
- 物化视图 (materialized view)。
- 使用绑定变量,而不是在 SQL 语句中使用常量。
- 更正绑定数据类型的不匹配。
- 使用 DBMS\_STATS 以计算更加精确的优化器统计信息。
- 优化动态采样。
- 使用系统统计信息,因为并非没有 DBMS\_STATS 这类的统计信息,基于成本的 SQL 优化器选择更好的执行计划。
- 使用缓存好的序列,而不是用表完成的计数器。
- 对 SQL 语句添加提示(作为提高执行计划质量的最后一步)。
- 存储概要。
- 使用存储概要以支持隐藏的提示(参见 Metalink note 92202.1)。
- SQL profiles (Oracle10g 以及以后版本)。
- SQL 计划管理器 (Oracle11g 以及以后版本才支持)。
- 使用数组插入或者批加载编程接口。
- 减少网络回路次数(比如使用 INSERT RETURNING)。
- PL/SQL 本地编译。
- 改变应用程序编码以减少竞争、解析开销、轮询等。

性能改进应该用额外的指标来文档化。你不能只依赖单一的指标,而应该同时考虑几个指标,然后计算平均值。

28.6 推断

当在测试案例上达到本质的性能改进的时候,推断改进对原始应用程序的影响非常重要。也就是调优能否停止。前面的测试案例中使用了插桩或者启用了 SQL 跟踪,现在应该要关掉了。当试图获得可信的数字来进行预测时,指标入侵是不合时宜的。毕竟,原始应用程序在插桩或者跟踪的时候是不会运行指标入侵的。另一方面,如果原始应用在运行时启用了插桩,那么你的测试用例也要启用。

28.7 安装

通常,在应用程序投入生产之前,应该首先在测试和质量保证系统中进行一些修改。如果最根本的原因是低效的 SQL 编码或者应用程序编码,那么在软件厂商合并必要的改变然后发布软件的一个新版本之前应该要花费很长时间。为了获得支持,需要重组数据库对象的改变需要一个很长的维修期。如果对于在改进阶段使用的额外成本特性(例如,分区表)没有预算,那么要想

别人支持提出的修改那是很困难的。此外，还需要额外的停机时间（downtime）来安装新功能。

28.8 MERITS 方法案例研究

这部分将把 MERITS 方法应用到一个实际的性能问题中。一个数码图像公司从客户那里接受 JPEG 格式的图像文件以进行打印和归档。这个公司的 Web 应用的程序（用 Perl 完成的）在数码相机文件中读取 EXIF 图像的元数据，然后把这个图像的元数据连同图像本身加载到 ORACLE 数据库中。EXIF 数据被网站上的搜索功能使用。图像数据以 BLOB 存储，这样可以在磁盘崩溃的时候提供全恢复（full recoverability）。JPEG 文件的平均大小是 1 MB。联系人说应用程序每分钟可以加载 68 个文件。目标是使每分钟加载文件的数量至少涨三倍。

我选择这个案例是因为它强调了扩展 SQL 跟踪的一些限制。从扩展 SQL 跟踪文件计算出来的响应时间和你接下来很快将看到的实际运行时间之间的显著差别会让你不得不相信对扩展 SQL 跟踪文件的分析通常都是不正确的。如此大的差别是一个异常，而不是一个常态。在这种情况下，这是由于对用 OCI 访问 LOB 的不完整插桩造成的。在这个案例中，响应时间文件观察到的精确度不够高，因此读者可以使用额外的工具和知识来克服这个难题。在这个特殊的案例中，我将演示在 Oracle10g 提供的行动级别中怎么结合使用插桩和统计信息收集以得到响应时间（每次行动的响应时间）的一个精确的表达式（V\$SERV\_MOD\_ACT\_STATS）。如果你仍在运行 Oracle9i，你应该能够从 SQL 跟踪文件中推导出每次行动的运行时间，这只要观察 Oracle9i 中的 module 和 action 项的时间戳即可（参见 22.8 节）。这并没有 Oracle10g 的视图 V\$SERV\_MOD\_ACT\_STATS 提供的
数据好（它包括 DB 时间、DB CPU 和其他统计信息），但是对于找到应用程序在哪个地方花费了最多时间还是足够了。

28

28.8.1 阶段 1——测量

我把文件 awr\_capture.sql 发送给了客户，然后叫 DBA 捕获图像加载程序运行 60 秒的活动。我还叫 DBA 运行脚本 statistics.sql 来创建一个涉及表结构的报告。

28.8.2 阶段 2——评估

我收到 awr\_capture.sql 创建的 SQL 跟踪文件以及 AWR 和 ASH 报告。statistics.sql 报告显示 LOB 以默认块大小存储在表空间中。启用 LOB 数据按行存储。

| LOB Column
Name | Segment Name | Tablespace | Block-
size | Chunk | Pct-
version | Retention | Cache | In
Row |
|--------------------|-------------------|------------|----------------|-------|-----------------|-----------|-------|-----------|
| IMAGE_DATA | IMAGES_IMAGE_DATA | USERS | 8 KB | 8192 | 10 | | NO | YES |

我用 TKPROF 处理这个 SQL 跟踪文件。由于看起来像是 EXEC 调用，而不是 FETCH 调用占响应时间的很大一部分，我使用排序选项 exeela, fchela。

```
$ tkprof ten_ora_3172_img_load.trc ten_ora_3172_img_load.tkp sort=exeela,fchela
```

以下是跟踪文件的 TKPROF 报告的节选：

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS

| call | count | cpu | elapsed | disk | query | current | rows |
|---------|-------|------|---------|------|-------|---------|------|
| Parse | 213 | 0.01 | 0.00 | 0 | 0 | 0 | 0 |
| Execute | 213 | 0.07 | 0.14 | 1 | 71 | 495 | 71 |
| Fetch | 142 | 0.00 | 0.01 | 0 | 142 | 0 | 142 |
| total | 568 | 0.09 | 0.16 | 1 | 213 | 495 | 213 |

Elapsed times include waiting on following events:

| Event waited on | Times
Waited | Max. Wait | Total Waited |
|---|-----------------|-----------|--------------|
| direct path read | 8010 | 0.27 | 13.23 |
| SQL*Net more data from client | 32182 | 0.00 | 0.43 |
| direct path write | 45619 | 0.01 | 0.69 |
| SQL*Net message to client | 8578 | 0.00 | 0.02 |
| SQL*Net message from client | 8578 | 0.03 | 4.91 |
| db file sequential read | 7130 | 0.25 | 26.11 |
| log file sync | 86 | 0.05 | 0.51 |
| log file switch completion | 5 | 0.99 | 3.23 |
| latch: shared pool | 10 | 0.00 | 0.00 |
| latch: library cache | 1 | 0.00 | 0.00 |
| log file switch (checkpoint incomplete) | 8 | 0.99 | 1.76 |

OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

| call | count | cpu | elapsed | disk | query | current | rows |
|---------|-------|------|---------|------|-------|---------|------|
| Parse | 100 | 0.00 | 0.03 | 0 | 0 | 0 | 0 |
| Execute | 166 | 0.14 | 0.18 | 0 | 230 | 46 | 46 |
| Fetch | 199 | 0.01 | 0.04 | 5 | 449 | 0 | 489 |
| total | 465 | 0.15 | 0.27 | 5 | 679 | 46 | 535 |

Elapsed times include waiting on following events:

| Event waited on | Times
Waited | Max. Wait | Total Waited |
|-------------------------|-----------------|-----------|--------------|
| db file sequential read | 5 | 0.01 | 0.03 |

213 user SQL statements in session.

166 internal SQL statements in session.

379 SQL statements in session.

\*\*\*\*\*

Trace file: ten\_ora\_3172\_img\_load.trc

Trace file compatibility: 10.01.00

Sort options: exeela fchela

1 session in tracefile.

213 user SQL statements in trace file.

166 internal SQL statements in trace file.

```
379 SQL statements in session.
*****
Trace file: ten_ora_3172_img_load.trc
Trace file compatibility: 10.01.00
Sort options: exeela fchela
    1 session in tracefile.
    213 user SQL statements in trace file.
    166 internal SQL statements in trace file.
    379 SQL statements in trace file.
    12 unique SQL statements in trace file.
128315 lines in trace file.
    72 elapsed seconds in trace file.
```

请注意总（递归的和非递归的）运行时间（0.43 s）和跟踪文件中的运行秒数的值（72 s）的巨大差别。根据 TKPROF，以下递归语句是运行时间的重要组成部分：

```
update seg$ set type#=:4,blocks=:5,extents=:6,minexts=:7,maxexts=:8,extsize=
:9,extpct=:10,user#=:11,iniexts=:12,lists=decode(:13, 65535, NULL, :13),
groups=decode(:14, 65535, NULL, :14), cachehint=:15, hwmincr=:16, spare1=
DECODE(:17,0,NULL,:17),scanhint=:18
where
ts#=:1 and file#=:2 and block#=:3
```

| call | count | cpu | elapsed | disk | query | current | rows |
|---------|-------|------|---------|------|-------|---------|------|
| Parse | 46 | 0.00 | 0.03 | 0 | 0 | 0 | 0 |
| Execute | 46 | 0.09 | 0.10 | 0 | 230 | 46 | 46 |
| Fetch | 0 | 0.00 | 0.00 | 0 | 0 | 0 | 0 |
| total | 92 | 0.09 | 0.13 | 0 | 230 | 46 | 46 |

我同样用 ESQLTRCPROF 处理了 SQL 跟踪文件。它给出了数以百计的警告，如下所示：

```
Warning: WAIT event 'direct path read' for cursor 4 at line 5492 without prior
PARSING IN CURSOR #4 - ignored for per statement response time accounting
Warning: WAIT event 'direct path write' for cursor 4 at line 5497 without prior
PARSING IN CURSOR #4 - ignored for per statement response time accounting
```

这些警告表明游标 4 的响应时间有点问题。ESQLTRCPROF 的资源配置文件如下所示：

```
ORACLE version 10.2 trace file. Timings are in microseconds (1/1000000 sec)
Resource Profile
=====
Response time: 5.627s; max(tim)-min(tim): 74.957s
Total wait time: 50.974s
-----
Note: 'SQL*Net message from client' waits for more than 0.005s are considered think
time
Wait events and CPU usage:
Duration      Pct      Count      Average Wait Event/CPU Usage/Think Time
-----
26.153s 464.76%      7135 0.003665s db file sequential read
13.234s 235.17%      8010 0.001652s direct path read
```

```
3.232s 57.43%      5 0.646382s log file switch completion
2.604s 46.28%     8507 0.000306s SQL*Net message from client
2.312s 41.08%     71 0.032562s think time
1.768s 31.41%      8 0.220956s log file switch (checkpoint incomplete)
0.690s 12.27%   45619 0.000015s direct path write
0.516s 9.18%      86 0.006004s log file sync
0.434s 7.72%    32182 0.000013s SQL*Net more data from client
0.313s 5.55%    1033 0.000303s total CPU
0.030s 0.53%    8578 0.000003s SQL*Net message to client
0.001s 0.02%     10 0.000109s latch: shared pool
0.000s 0.00%      1 0.000207s latch: library cache
-45.659s -811.39%          unknown
-----
5.627s 100.00% Total response time
```

Total number of roundtrips (SQL\*Net message from/to client): 8578

CPU usage breakdown

```
-----
parse CPU:    0.05s (313 PARSE calls)
exec CPU:     0.25s (379 EXEC calls)
fetch CPU:    0.02s (341 FETCH calls)
```

响应时间 5.627 s 和跟踪文件显示的总运行时间 (max (tim) - min (tim): 74.957 s) 之间的差别同 TKPROF 报告显示的一样明显。同 TKPROF 不同, ESQLTRCPROF 报告指出了还有 45 s 不知道该归于哪一类 (unknown)。总运行时间 50.974 s 证明了应用程序在大部分时间里面都是和 DBMS 交互的, 但是等待时间应该加到解析、执行和提取调用中。

根据 ESQLTRCPROF, 响应时间的最大构成者与同一个没有与之关联的 SQL 语句文本的游标相关。

Statements Sorted by Elapsed Time (including recursive resource utilization)

```
=====
Hash Value: -1 - Total Elapsed Time (excluding think time): 2.976s
```

Cursor 0 - unknown statement (default container for any trace file entries relating to cursor 0)

| DB Call | Count | Elapsed | CPU | Disk | Query | Current | Rows |
|---------|-------|---------|---------|------|-------|---------|------|
| PARSE | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| EXEC | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| FETCH | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| Total | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |

| Wait Event/CPU Usage/Think Time | Duration | Count |
|---------------------------------|----------|-------|
| SQL*Net message from client | 2.431s | 8081 |
| think time | 2.312s | 71 |
| log file sync | 0.516s | 86 |

| | | |
|---------------------------|--------|------|
| SQL*Net message to client | 0.028s | 8152 |
| latch: shared pool | 0.001s | 6 |
| latch: library cache | 0.000s | 1 |
| total CPU | 0.000s | 0 |

TKPROF 没有一个排序选项来合并数据库调用之间的等待时间。因此它不会报告这个 unknown 语句为响应时间的最大构成者。

哪里出错了？由于没有与之关联的 SQL 语句的游标？数据库间调用等待的等待时间（比如 direct path read 和 direct path write）没有加到数据库调用中？难道两个工具都报告了错误的结果？仔细观察扩展跟踪文件会发现 92 956 个等待事件同游标 4 相关。然而，跟踪文件并没有包括游标 4 的单独 PARSING IN CURSOR、PARSE、EXEC 或者 FETCH 项。脚本 awr\_capture.sql 获得的级别 2 的 ERRORSTACK 转储包括游标 4 的一个相当奇怪的值 table\_e\_a\_d21e\_a\_0\_0。很明显，这不是由于缺失 PARSING IN CURSOR 项而引起的问题，这是由于运行中的应用程序的中期会启用跟踪。那仅仅是因为这个游标没有存在一个合适的 SQL 语句。

```
Cursor#4(09050CE4) state=NULL curiob=090C2054
  curflg=1044 fl2=0 par=00000000 ses=6CD86754
  sqltxt(6AAAA544)=table_e_a_d21e_a_0_0
```

正如在第 24 章指出的，只有 PARSE、EXEC 和 FETCH 项才报告 CPU 的使用情况。此外，这三个项报告的运行时间包括由于解析、执行和提取操作产生的等待事件。由于游标 4 没有这些项，等待时间不能相加。响应时间 R 被定义为解析、执行和提取调用的运行时间的总和，再加上数据库调用之间等待时间的总和。R 同跟踪文件报告的时间间隔的巨大不同的原因现在一目了然了。理论上来说，所有数据库调用的运行时间应该等于 CPU 使用加上由于数据库调用引起的等待时间。这之间的差别被报告为 unknown。在这个资源配置文件中，unknown 的影响很大，这是因为游标 4 的等待时间并没有归入到任何一个数据库调用中。因此，游标 4 究竟与什么相关联？以下是跟踪文件中的一些内容：

```
WAIT #4: nam='db file sequential read' ela= 2926 file#=4 block#=90206 blocks=1
obj#=53791 tim=19641952691
WAIT #4: nam='db file sequential read' ela= 1666 file#=4 block#=90221 blocks=1
obj#=53791 tim=19641954572
WAIT #4: nam='direct path read' ela= 275 file number=4 first dba=90206 block cnt=1
obj#=53791 tim=19641964448
WAIT #4: nam='direct path write' ela= 3 file number=4 first dba=90174 block cnt=1
obj#=53791 tim=19641955477
```

通过查询 DBA\_EXTENTS 而进行的 file#=4 和 block#=90206 的转换会产生如下的结果：

```
SQL> SELECT segment_name, segment_type, extent_id
FROM dba_extents
WHERE file_id=4 AND 90206 BETWEEN block_id AND block_id + blocks - 1;
SEGMENT_NAME      SEGMENT_TYPE EXTENT_ID
-----
IMAGES_IMAGE_DATA LOBSEGMENT          40
```

很显然游标 4 同 LOB 加载有关。这一发现是基于 ASH 报告同样列出了一个导致直接路径读和写

操作的游标，但是缺少 SQL 语句文本。这一信息可以在 ASH 报告的 Top SQL Statements 部分找到（参见图 28-1），这是脚本 awr\_capture.sql 创建的。Top DB Objects 部分表明 LOB 是最高一个对象。这时我们可能会得出一个结论：用 OCI（经常被 Perl DBI 内部使用）加载 LOB 数据的插桩效果很差。而扩展 SQL 跟踪文件压根就没有报告加载 LOB 的 CPU 使用情况。

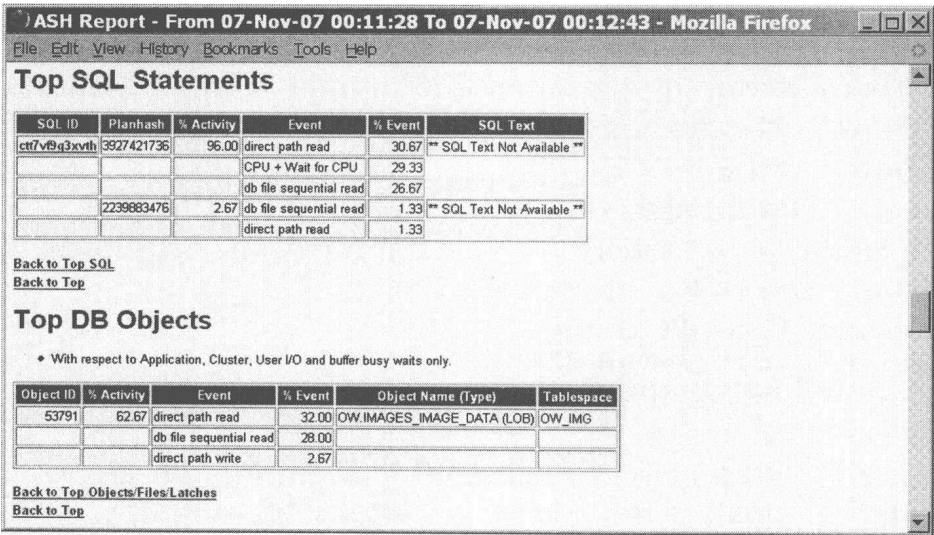


图 28-1 ASH 报告的 Top SQL Statements 部分

根据 ESQLTRCPROF 报告，在表 IMAGES 中插入了 71 行。

Hash Value: 3858514115 - Total Elapsed Time (excluding think time): 0.146s

```
INSERT INTO images (id, date_loaded, exif_make, exif_model, exif_create_date,
exif_iso, exif_f_number, exif_exposure_time, exif_35mm_focal_length, image_data)
VALUES(:id, sysdate, :exif_make, :exif_model,
to_date(:exif_create_date, 'yyyy:mm:dd hh24:mi:ss'),
:exif_iso, :exif_f_number, :exif_exposure_time, :exif_35mm_focal_length,
empty_blob())
```

| DB Call | Count | Elapsed | CPU | Disk | Query | Current | Rows |
|---------|-------|---------|---------|-------|-------|---------|-------|
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| PARSE | 71 | 0.0026s | 0.0156s | 0 | 0 | 0 | 0 |
| EXEC | 71 | 0.1203s | 0.0781s | 1 | 71 | 495 | 71 |
| FETCH | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| Total | 142 | 0.1229s | 0.0938s | 1 | 71 | 495 | 71 |

由于跟踪文件报告的时间间隔是 74.9 秒，因此每分钟插入 56 个 LOB。令我吃惊的是这个 INSERT 语句同样也有 71 个解析调用。这是否意味着 INSERT 语句的解析调用是在循环中完成的，而不是在进行循环之前一次完成。

值得注意的是表 SYS.SEQ\$ 的 71 次更新，占用序列的数据字典基表也出现了：

Hash Value: 2635489469 - Total Elapsed Time (excluding think time): 0.082s

```
update seq$ set increment$=:2,minvalue=:3,maxvalue=:4,cycle#=:5,order$=:6,cache=:7,
highwater=:8,audit$=:9,flags=:10 where obj#=:1
```

| DB Call | Count | Elapsed | CPU | Disk | Query | Current | Rows |
|---------|-------|---------|---------|-------|-------|---------|-------|
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| PARSE | 71 | 0.0026s | 0.0000s | 0 | 0 | 0 | 0 |
| EXEC | 71 | 0.0790s | 0.0781s | 0 | 71 | 142 | 71 |
| FETCH | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| Total | 142 | 0.0815s | 0.0781s | 0 | 71 | 142 | 71 |

这可能意味着一个没有被缓存的序列有可能增加到 71 次。为了验证这个假设，我在跟踪文件中搜索 ESQTRCPROF 报告<sup>①</sup>中的散列值 2 635 489 469，然后取回列 obj# 的绑定变量值。由于脚本 awr\_capture.sql 启用了 12 级 SQL 跟踪，跟踪文件一定会包含绑定变量。从左往右数，第 10 个绑定变量同 obj# 相适应。由于绑定变量是从 0 开始计数，我们就需要找到 Bind#9。

```
Bind#9
oacdtty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
kxsbbbfp=0915bf8c bln=22 avl=04 flg=05
value=53740
```

这就产生了序列的对象标识符，我曾经用它来从 DBA\_OBJECTS 对象中检索信息。事实证明对象 53 740 确实是一个序列，而且没有被缓存。

```
SQL> SELECT object_name, object_type FROM dba_objects WHERE object_id=53740;
OBJECT_NAME  OBJECT_TYPE
-----
IMAGE_ID_SEQ SEQUENCE
SQL> SELECT cache_size
FROM dba_objects o, dba_sequences s
WHERE o.object_id=53740
AND o.owner=s.sequence_owner
AND o.object_name=s.sequence_name;
CACHE_SIZE
-----
0
```

具有相同散列值的其他语句在 Bind#9 中同样以 53 740 为绑定变量值。SYS.SEQ\$ 的更新并不是总响应时间的主要组成部分。然而，它是不必要的开销，用一个 ALTER SEQUENCE 语句就很容易解决。

在 ESQTRCPROF 报告中我还发现了一个问题，有 71 次提交。

```
Statistics:
-----
```

① TKPROF 删除了 SQL 语句文本的散列值。

```

COMMITs (read write): 71 -> transactions/sec 12.617
COMMITs (read only): 0
ROLLBACKs (read write): 0
ROLLBACKs (read only): 0

```

很显然，插入一行和加载一次 LOB 分别被提交了一次，这增加了开销。每一次提交都有可能导致等待等待事件 log file sync。

28.8.3 阶段 3——重现

客户同意将加载 LOB 的 Perl 子程序作为一个单独的 Perl 程序以在测试系统上进行进一步的调查。同样提供 DDL。LOB 和序列都没有启用缓存。

```

CREATE TABLE images(
    id number,
    date_loaded date,
    exif_make varchar2(30),
    exif_model varchar2(30),
    exif_create_date date,
    exif_iso varchar2(30),
    exif_f_number varchar2(30),
    exif_exposure_time varchar2(30),
    exif_35mm_focal_length varchar2(30),
    image_data BLOB,
    CONSTRAINT images_pk PRIMARY KEY(id)
)
LOB (image_data) STORE AS images_image_data;

CREATE SEQUENCE image_id_seq NOCACHE;

```

我做的第一件事是用合适 ORACLE 的 Hotsos ILO<sup>®</sup> 插桩 Perl 程序。用 ILO 插桩是很容易想到的事情。使用程序 HOTSOS\_ILO\_TASK.BEGIN\_TASK 用一个特定 module 和 action 名称来开始一项新任务。BEGIN\_TASK 将前面的 module 和 action 压到一个栈中，正如第 23 章讨论的一样。程序 HOTSOS\_ILO\_TASK.END\_TASK 结束一个任务，还原到前一 module 和 action。Module 和 action 都在 V\$ 视图中得以体现，比如 V\$SESSION 和 V\$SQL。为了在第一个任务开始的时候尽快启用 SQL 跟踪，应用程序自己调用包 HOTSOS\_ILO\_TIMER，如下所示：

```

begin
    hotsos_ilo_timer.set_mark_all_tasks_interesting(mark_all_tasks_interesting=>true,
        ignore_schedule=>true);
end;

```

在整个程序中，我使用了 module 名 img\_load，并定义了两个 action：

- ❑ action “exif\_insert” 包含下一行的新的主键的产生，以及 LOB 定位器（稍后使用 Perl DBI 函数 ora\_lob\_append 来加载 LOB 数据）的检索。
- ❑ action “lob\_load” 包括从磁盘读取 JPEG 文件，以及加载它到 BLOB 列。由于平均图像文

① ILO 是一个免费的软件，可以从 <http://sourceforge.net/projects/hotsos-ilo> 中下载。下载的包包括 HTML 格式的文档。

件大小是 1 MB，LOB 是按块加载。

我假设文件系统在读取 JPEG 文件时不会产生很多的开销。如果这个假设被证明为是错误的，除了用 Hotsos ILO 插桩数据库访问我还应该用 Perl 插桩文件系统的访问。

这个测试程序允许我预测会加载多少 LOB。为了简单起见，一遍又一遍地加载同一个 JPEG 文件，虽然由于文件系统缓存这样做会减少读取图像文件的压力，而原始程序每次都应该读取单独的图像文件。

我设置数据库收集感兴趣的 service、module、action 的统计信息，如下所示：

```
SQL> EXEC dbms_monitor.serv_mod_act_stat_enable('TEN.oradbpro.com', -
> 'img_load', 'exif_insert')
SQL> EXEC dbms_monitor.serv_mod_act_stat_enable('TEN.oradbpro.com', -
> 'img_load', 'lob_load')
```

我用 UNIX 实用工具 time<sup>①</sup>测量一次加载运行的响应时间，这个加载运行包括 10 次 LOB。

```
$ time perl img_load.pl 10 sample.jpg
real    0m9.936s
user    0m0.015s
sys     0m0.015s
```

然后我从 V\$SERV\_MOD\_ACT\_STATS 检索 service、module、action 的相关统计信息。

```
SQL> SELECT action, stat_name, round(value/1000000, 2) AS value
FROM v$serv_mod_act_stats
WHERE service_name='TEN.oradbpro.com'
AND module='img_load'
AND action IN ('exif_insert','lob_load')
AND stat_name in ('DB time', 'DB CPU', 'sql execute elapsed time',
'user I/O wait time')
ORDER BY action, stat_name;
```

| ACTION | STAT_NAME | VALUE |
|-------------|--------------------------|-------|
| exif_insert | DB CPU | .02 |
| exif_insert | DB time | .02 |
| exif_insert | sql execute elapsed time | .01 |
| exif_insert | user I/O wait time | 0 |
| lob_load | DB CPU | 2.7 |
| lob_load | DB time | 8.75 |
| lob_load | sql execute elapsed time | 0 |
| lob_load | user I/O wait time | 5.85 |

这两个 action 的总“DB 时间”（这并不包括网络延迟和思考时间（SQL\*Net message from client）是 8.77 秒。这已经相当接近 UNIX 实用工具 time 报告的时间（9.9 秒）。由于后一次测量包括 Perl 程序的编译和连接到 DBMS 实例的时间，一些差别是允许的。由于插桩，很显然大部分响应时间是由 action “lob\_load” 引起的，它负责处理加载 BLOB 列。

然而，由于 LOB 加载，扩展 SQL 跟踪文件没能成功计算 CPU 使用情况，V\$SERV\_MOD\_ACT\_STATS

① 用 Cygwin 可以使用 Windows 环境下的 time 实用工具。

报告加载 10 次 LOB 花费了 2.7 秒的 CPU 使用。大部分响应时间都集中在了用户 I/O 等待时间上。这同大量 db 文件顺序读相关，还和 SQL 跟踪文件中游标 4 相关的直接路径读/写操作相关。由于 9.93 秒（由 UNIX 实用工具 time 测量得到的响应时间）中至少有 8.77 秒（这两个 action 的总“DB 时间”），即 88% 的时间都花费在 DBMS 实例中，读取 JPEG 文件并没有占很多的响应时间这一假设被证明是正确的。

V\$SERV\_MOD\_ACT\_STATS 中的数字自从实例开始之后就一直累计。为了禁用统计信息收集，需要调用 DBMS\_MONITOR:

```
SQL> EXEC dbms_monitor.serv_mod_act_stat_disable('TEN.oraclepro.com', -
> 'img_load', 'exif_insert')
SQL> EXEC dbms_monitor.serv_mod_act_stat_disable('TEN.oraclepro.com', 'img_load', -
> 'lob_load')
```

用这种方法禁用统计信息收集并不会清理统计信息。如果下一次在相同 module 和 action 上重新统计信息收集，之前获取的测量数据又重新可用了，除非重启实例。

为了比较改进响应时间前后的测量数据，我在事件 10 046 启用了 8 级的 SQL 跟踪，记录了这次的运行结果，因为我不希望由 I/O 引起的额外执行和回路影响我的实验结果。产生的 ESQTRCPROF 报告重现于下：

ORACLE version 10.2 trace file. Timings are in microseconds (1/1000000 sec)

Resource Profile

=====

Response time: 0.779s; max(tim)-min(tim): 8.902s

Total wait time: 6.012s

Note: 'SQL\*Net message from client' waits for more than 0.005s are considered think time

Wait events and CPU usage:

| Duration | Pct | Count | Average Wait Event/CPU Usage/Think Time |
|----------|----------|-------|---|
| 3.834s | 491.92% | 1160 | 0.003305s db file sequential read |
| 1.259s | 161.54% | 1120 | 0.001124s direct path read |
| 0.350s | 44.96% | 1191 | 0.000294s SQL*Net message from client |
| 0.346s | 44.40% | 10 | 0.034603s think time |
| 0.090s | 11.60% | 6227 | 0.000015s direct path write |
| 0.068s | 8.72% | 23 | 0.002954s log file sync |
| 0.060s | 7.66% | 4500 | 0.000013s SQL*Net more data from client |
| 0.004s | 0.52% | 1201 | 0.000003s SQL*Net message to client |
| 0.000s | 0.00% | 119 | 0.000000s total CPU |
| -5.232s | -671.32% | | unknown |

0.779s 100.00% Total response time

Total number of roundtrips (SQL\*Net message from/to client): 1201

CPU usage breakdown

```
-----
parse CPU:    0.00s (46 PARSE calls)
exec CPU:     0.00s (47 EXEC calls)
fetch CPU:    0.00s (26 FETCH calls)
```

Statistics:

```
-----
COMMITs (read write): 10 -> transactions/sec 12.830
```

注意存在 10 个实例有思考时间。这很容易理解，因为进行了十次迭代。这意味着这个分析器能够正确地将访问 JPEG 文件的时间归类为思考时间。

28.8.4 阶段 4——改进

很显然加载 LOB 数据需要改进。从其他涉及 LOB 的项目中我知道，访问那些没有被缓存的 LOB 会非常慢。我同样知道，比一些数据库块大很多的 LOB 得益于更大的块和 chunk 大小。客户数据库和我的测试数据库的数据库块大小都是 8 KB，我配置一个独立的、大小为 16 KB 块大小的缓存池，然后重启测试用例，为 LOB 存储创建一个 16 KB 块大小的表空间。以下是这个任务中涉及的 SQL 语句：

```
SQL> ALTER SYSTEM SET db_16k_cache_size=50m SCOPE=SPFILE;
SQL> CREATE TABLESPACE lob_ts DATAFILE '&data_file_path' SIZE 1G BLOCKSIZE 16384;
```

对于一个生产系统，DB\_16K\_CACHE\_SIZE 的值应该使用比 50 MB 大很多的值。接下来，我将 LOB 段移到新的表空间，增加 LOB 的 chunk 大小到最大值 32 768，然后用表中的另外一列禁止 LOB 数据按行存储。

```
SQL> ALTER TABLE images MOVE LOB (image_data)
STORE AS (TABLESPACE lob_ts DISABLE STORAGE IN ROW
CACHE RETENTION CHUNK 32768);
```

由于移动了表，索引不可用了（如果执行 INSERT 语句会产生 ORA-01502），必须重建主键索引。

```
SQL> ALTER INDEX images_pk REBUILD;
```

可以使用包 DBMS\_REDEFINITION（它支持表和关联索引的在线识别）来减少由于重建索引带来的时间开销。

在应用这些改变之后，在启用插桩以及禁用 SQL 跟踪的情况下，我加载了另外 10 行。

```
$ time perl img_load.pl 10 sample.jpg
real    0m2.550s
user    0m0.045s
sys     0m0.061s
```

响应时间实际上只有 2.5 秒。这是我没有想到的，因此我重复运行了 9 次。这 10 次运行的平均响应时间是 3.02 秒。仅仅前面的变化就将响应时间减半了。

接下来我仔细查看了 Perl 代码。除了循环中进行解析（这在 SQL 跟踪文件和资源配置文件中已经清晰可见了），我看到从文件系统中读取 LOB，然后发送到 DBMS 实例中都是以 8 KB 的

大小进行的。8 KB 看起来有点小，因为我已经将块大小增加到了 16 KB，而 LOB chunk 大小也增加到了 32 KB。以下显示 LOB 列是如何加载的代码的节选：

```
do {
    $bytes_read=sysread(LOBFILE, $data, 8192);
    $total_bytes+=$bytes_read;
    if ($bytes_read > 0) {
        my $src = $dbh->ora_lob_append($lob_loc, $data);
    }
} until $bytes_read <=0;
```

我决定实验极端情况，使用 1 048 576 (1 MB) 的块大小，而不是 8 192 (8 KB)。另外 10 次运行的平均时间是 1.71 秒。减少了 43% 的响应时间。至今为止测试的两处变化已经远远超出了客户制定的目标，因此我本可以停止优化了。但是，我希望获得解析一次多次执行的效果。此外，我还希望指出如何有望通过使用 INSERT RETURNING 来减少客户端和数据库服务器之间的回路次数。

应用程序的原始算法如下所示。

(1) 使用 SQL 语句 SELECT image\_id\_seq.NEXTVAL FROM dual 来增加和检索用来对主键进行编号的序列。

(2) 使用序列值作为列 ID 的主键来插入一行到表 IMAGES 中。INSERT 语句用 empty\_blob() 同样初始化了 BLOB。

(3) 使用列 ID 的索引检索 LOB 定位器，语句为 SELECT image\_data FROM images WHERE id=:id。

这要求对这 3 个独立的语句进行解析和执行。然而，通过使用 INSERT RETURNING 可以将这 3 步合并为 1 步，如下所示：

```
INSERT INTO images (id, date_loaded, exif_make, exif_model, exif_create_date,
exif_iso, exif_f_number, exif_exposure_time, exif_35mm_focal_length, image_data)
VALUES(image_id_seq.NEXTVAL, sysdate, :exif_make, :exif_model,
to_date(:exif_create_date, 'yyyy:mm:dd hh24:mi:ss'), :exif_iso,
:exif_f_number, :exif_exposure_time, :exif_35mm_focal_length, empty_blob())
RETURNING id, rowid, image_data INTO :id, :row_id, :lob_loc
```

不幸的是，这破坏了同 Oracle10g 版本 2 (DBI 版本：1.41, DBD::Oracle 版本：1.15)<sup>①</sup>一起用的 Perl 版本，但是和更近的版本配合得很好。作为一个变通方案，可以单独提取 LOB 定位器。如果使用这个变通方案得当，这 10 次运行的平均响应时间会减少到 1.11 秒。

仍有三个问题亟待解决。

(1) 没有被缓存的序列。

(2) 加载图像的循环中不必要的解析调用。

(3) 在循环内的频繁提交，而不是在完成加载过程之后的一次提交（或者至少是间歇性地提交，例如加载大量图像的时候，10 000 行之后提交一次）。

<sup>①</sup> Oracle11g 也有一些相同版本的 DBI 和 DBD::Oracle，因此也不能使用 Oracle11g 的 Perl 客户端。

通过以下 DDL 语句，我给序列赋值一个 1000 个数字的缓存：

```
SQL> ALTER SEQUENCE image_id_seq CACHE 1000;
```

缓存更多的序列数字并不会增加共享池使用的内存。当一个实例用 SHUTDOWN NORMAL 或者 IMMEDIATE 关闭时，没有用过的序列数字会在数据字典中进行标记。只有在一个实例崩溃，或者用 ABORT 选项进行关闭的时候，序列数字才会丢失。请注意，回滚一个从序列中选择 NEXTVAL 的事务同样会丢弃序列数字。因此，就不得不使用大的序列高速缓存了。

最后，我修改 Perl 程序，让 INSERT 和 SELECT 语句都只解析一次。为了证明这两个语句确实只解析了一次，我启用了事件 10046 的 SQL 跟踪，用 ESQTRCPROF 运行跟踪文件，然后查看解析调用。INSERT 语句的数字如下所示：

| DB Call | Count | Elapsed | CPU | Disk | Query | Current | Rows |
|---------|-------|---------|---------|------|-------|---------|------|
| PARSE | 1 | 0.0002s | 0.0000s | 0 | 0 | 0 | 0 |
| EXEC | 10 | 0.0459s | 0.0156s | 4 | 15 | 75 | 10 |
| FETCH | 0 | 0.0000s | 0.0000s | 0 | 0 | 0 | 0 |
| Total | 11 | 0.0460s | 0.0156s | 4 | 15 | 75 | 10 |

这些数字显示这个语句只解析了一次而执行了 10 次。当然，对绑定变量进行赋值还是在循环中进行。只不过是把解析调用（在 Perl DBI 中是 prepare）移到了循环之外。在语句中使用常量不会有这样的效果，因为在每次的循环迭代中语句文本都会改变，因此必须重新解析语句。减少解析开销是使用绑定变量的一个原因。另一个原因是减少库缓存的争用。以下是整个资源配置文件：

```
ORACLE version 10.2 trace file. Timings are in microseconds (1/1000000 sec)
Resource Profile
=====
Response time: 0.269s; max(tim)-min(tim): 0.582s
Total wait time: 0.320s
-----
Note: 'SQL*Net message from client' waits for more than 0.005s are considered think
time
Wait events and CPU usage:
Duration      Pct      Count      Average Wait Event/CPU Usage/Think Time
-----
0.215s  80.10%      10 0.021544s think time
0.056s  20.93%    4620 0.000012s SQL*Net more data from client
0.035s  13.11%       1 0.035251s log file sync
0.013s   4.94%      33 0.000403s SQL*Net message from client
0.000s   0.06%      43 0.000004s SQL*Net message to client
0.000s   0.00%      41 0.000000s total CPU
-0.051s -19.13%           unknown
-----
0.269s 100.00% Total response time

Total number of roundtrips (SQL*Net message from/to client): 43
```

CPU usage breakdown

```
-----
parse CPU:    0.00s (5 PARSE calls)
exec CPU:     0.00s (24 EXEC calls)
fetch CPU:    0.00s (12 FETCH calls)
```

Statistics:

```
-----
COMMITs (read write): 1 -> transactions/sec 3.718
```

在这个资源配置文件中, 响应时间 R 占了跟踪文件捕获的 0.582 秒的 46%。这一次, 响应时间的 unknown 部分只占了 19%。

我用 V\$SERV\_MOD\_ACT\_STATS 和 DBMS\_MONITOR 再次测量了这两个 action。由于不能在 V\$SERV\_MOD\_ACT\_STATS 中清理统计信息, 这次我使用 action 名 exif\_insert\_imp 和 lob\_load\_imp。当然, 本可能对 V\$SERV\_MOD\_ACT\_STATS 进行两次快照, 然后计算差值, 但是改变 module 名和 action 名更有意义, 因为程序也发生了重大的改变。对 DBMS\_MONITOR 的必要调用如下所示:

```
SQL> EXEC dbms_monitor.serv_mod_act_stat_enable('TEN.oradbpro.com','img_load', -
> 'exif_insert_imp')
SQL> EXEC dbms_monitor.serv_mod_act_stat_enable('TEN.oradbpro.com','img_load', -
> 'lob_load_imp')
```

现在这个系统已经准备好测量另外 10 次迭代了。

```
$ time perl img_load_improved.pl 10 sample.jpg
real    0m0.688s
user    0m0.031s
sys     0m0.000s
```

这次运行运行得很快, 以致我要用毫秒的解决方案获取数字, 以免有些值因为四舍五入而变为 0。

```
SQL> SELECT action, stat_name, round(value/1000000, 3) AS value
FROM v$serv_mod_act_stats
WHERE service_name='TEN.oradbpro.com'
AND module='img_load'
AND action IN ('exif_insert_imp','lob_load_imp')
AND stat_name in ('DB time', 'DB CPU', 'sql execute elapsed time',
'user I/O wait time')
ORDER BY action, stat_name;
```

| ACTION | STAT_NAME | VALUE |
|-----------------|--------------------------|-------|
| exif_insert_imp | DB CPU | .003 |
| exif_insert_imp | DB time | .003 |
| exif_insert_imp | sql execute elapsed time | .003 |
| exif_insert_imp | user I/O wait time | 0 |
| lob_load_imp | DB CPU | .116 |
| lob_load_imp | DB time | .152 |
| lob_load_imp | sql execute elapsed time | .001 |
| lob_load_imp | user I/O wait time | 0 |

这两个 action 的 DB 时间以前是 8.77 秒，现在减少到只有 0.155 秒。这只占实用工具 time 测量出来的响应时间的 22%，因此基于这些数字进行预测并不是一个好主意。请注意，DB 时间并不包括发生在数据调用之间的等待事件，比如 SQL\*Net message from client 和 SQL\*Net message to client。

28.8.5 阶段 5——推断

因为原始测量每分钟 68 个 LOB 是在没有合适插桩以及没有启用 SQL 跟踪的情况下进行的，作为我的推断的基础，我还需要一些测量数字是不随这些因素而变化的。我做了另外 10 次测量，该过程禁用了插桩和 SQL 跟踪。插入 10 个 LOB 的平均运行时间是 0.931 秒。基于这些数字，应用程序每分钟至少应该能够加载 645 个 LOB。根据 8 级跟踪文件揭示的运行时间（0.582 秒）可以推断出每分钟能够加载大约 1 030 个 LOB。而实际数字有可能是介于这两个数字之间。

表 28-2 总结了原始测试案例和优化后的测试案例的差别。这些数字是加载 10 个 LOB 的情况下产生的。运行时间减少到原始值的 1/10 以下。

表28-2 优化前后测量指标的变化

| Metric | 原始测试案例 | 优化后的测试案例 |
|---------------|--------|----------|
| 跟踪文件报告的运行时间 | 8.90秒 | 0.58秒 |
| 总等待时间 | 6.01秒 | 0.32秒 |
| SQL * Net来回次数 | 1 201 | 43 |
| 解析调用次数 | 46 | 5 |
| 执行次数 | 47 | 24 |
| 提取调用次数 | 26 | 12 |
| 思考时间 | 0.34秒 | 0.21秒 |

28.8.6 阶段 6——安装

必须克服的最重大的障碍之一是识别 LOB 段，因为这需要停机时间。除此之外，这些改变很快就得到了支持。在维护期间，在这些表被识别之后，应该要采用新的测量方式。采用的测量方式应该禁用插桩和 SQL 跟踪。吞吐量在每分钟 640~682 个 LOB 变化。这非常接近于推断出来的值——每分钟 645 个 LOB。同原始的吞吐量相比较，加速比超过 10 倍。

28.8.7 小结

这个案例研究强调的是 DBMS 的插桩并非总是可信的，因为有些代码路径不会被完全插桩到。即使在如此恶劣的环境下，插桩的使用使我们能够非常轻易地知道应用程序的哪些部分花费了较多时间。为了获得最佳性能，数据库结构和应用程序编码的双方都应该利用 ORACLE DBMS 提供的强大功能，比如 LOB 和序列的缓存，使用绑定变量减少解析开销，使用 INSERT RETURNING 减少网络来回次数。LOB 的默认设置对获得好的性能没有任何帮助。

28.9 源代码库

表 28-3 列出了本章的源代码文件以及它们的功能。

表28-3 MERITS方法的源代码库

| 文 件 名 | 功 能 |
|----------------------|--|
| awr_capture.sql | 这个脚本捕获扩展SQL跟踪和AWR的性能数据。临时设置_ASH_SAMPLE_ALL=TRUE使得ASH能够为改进的诊断表达式采样空闲等待事件。为跟踪的会话自动生成一个AWR报告和ASH报告。这两个报告都是以HTML的形式产生的，并在当前目录中 |
| ilo_test.sql | 这个SQL脚本能够启用带有Hotsos ILO的扩展SQL跟踪，运行SELECT语句、开始任务、结束任务。可以使用它来了解应用程序插桩写入了哪些跟踪文件项 |
| img_load.pl | 是原始的次优的LOB加载测试案例，包括表和序列使用的DDL语句。要想运行该程序，需将Perl包Image::ExifTool的安装路径加到环境变量PERLSLIB中 |
| img_load_improved.pl | 优化过的LOB加载测试案例 |
| sp_capture.sql | 这个脚本捕获扩展SQL跟踪和Statspack的性能数据。包括一个会话级的Statspack快照。请确保你安装了导致不正确会话级别报告的bug修复程序 |

Part 9

第九部分

Oracle Net

本 部 分 内 容

- 第 29 章 TNS 监听器 IP 地址绑定与 IP=FIRST
- 第 30 章 TNS 监听器 TCP/IP 有效结点检验
- 第 31 章 本地命名参数 ENABLE=BROKEN
- 第 32 章 Oracle Net 配置中默认的主机名

TNS 监听器 IP 地址绑定与 IP=FIRST

默认情况下，在有多个网络适配器的系统中，TNS 监听器（TNS listener）会从任何网络中接收连接。在 Oracle10g 及其之后发布的版本中，可采用 IP=FIRST 约束配置在 listener.ora 中的主机所在网络的连接。

Oracle Database, Oracle Clusterware and Oracle Real Application Clusters Installation Guide 10g Release 2 for AIX 一书包含一个通过在 listener.ora 配置文件中设置 IP=FIRST 来配置 TNS 监听器的例子，但是没有解释其中的意义。在 *Oracle Database Net Services Reference 10g Release 2* 中没有提到这个设置项。在 *Oracle Database Net Services Reference 11g Release 1* 中提到了 IP=FIRST 设置，但是这个文档并不完全正确<sup>①</sup>。这个手册并没有给出所有参数的实现，并且也没有提到涉及 loopback 适配器的监听器行为也会受到 IP=FIRST 的影响。

29.1 IP 地址绑定介绍

互联网中 IP 地址可以按其功能分为三类。

- ❑ 引导 IP 地址（Boot IP address）。
- ❑ 通用 IP（非引导 IP）地址（Common IP address）。
- ❑ 服务 IP 地址（Service IP address）。

网络中的每个系统都有一个唯一的主机名作为标志。赋予这个唯一主机名的 IP 地址叫做引导 IP 地址。这个 IP 地址在引导过程中绑定到网络适配器中。在 UNIX 和 Windows 操作系统中，这个唯一主机名通过 hostname 命令获得，并被映射到引导 IP 地址上。无需其他软件，操作系统本身就可以使用引导 IP 地址。通用地址（非引导）是非引导适配器（引导适配器指的是赋予引导 IP 地址的适配器）使用的地址。当然，一般情况下，非引导适配器也是通过引导过程赋予地址。每个系统都只有一个引导 IP 地址。为了获得一个 UNIX 系统本身的引导 IP 地址，可以 ping 系统本身。

<sup>①</sup> 文档里面有很多错误信息，其中包括语法例子错误和 C 语言预处理宏命令 INADDR\_ANY 的生成不正确。

```
$ ping -c 1 `hostname`
PING dbserver1.oradbpro.com (172.168.0.1) 56(84) bytes of data.
64 bytes from dbserver1.oradbpro.com (172.168.0.1): icmp_seq=0 ttl=64 time=0.029 ms

--- dbserver1.oradbpro.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.029/0.029/0.029/0.000 ms, pipe 2
```

在 Linux 环境下, 参数 `-c` 使命令 `ping` 发送一个包。在 Solaris 环境下, 使用 `ping host_name packet_size count` 来获得系统的启动 IP, 这里 `count` 值设置为 1。

术语服务地址 (service address) 指的是应用软件用来提供特定服务的 IP 地址。通常情况下, 服务地址并不赋予一个单独的网络适配器, 而是作为别名地址添加到已经含有引导地址或者非引导地址的网络适配器中。这种方法叫做 IP 别名制。事实上, 服务地址也是一种非引导地址, 它们之间的区别是服务地址被分配给一个虚拟的适配器, 然而后者被分配一个物理适配器。

在一个单独的操作系统实例中, 每个网络适配器由一个唯一的名字识别。AIX 系统使用 `entN`, Solaris 使用 `hmeN` (`hme` = hundred megabit ethernet) 或者 `geN` (`ge` = gigabit ethernet), Linux 使用 `ethN`, 这里的 `N` 是一个整数, 用来区别同一类型的不同适配器。另外, IP 别名制需要额外的适配器名称。别名 IP 由物理适配器的名称加上一个冒号和一个数字组成。例如, 物理适配器名称为 `eth0`, IP 别名的适配器名称为 `eth0:1`。别名 IP 地址不能随意选取, 必须作为相关物理适配器的网络地址应用在同一个网络中。并且, 别名 IP 地址和掩码有关 (比如下例中掩码: 255.255.255.0)。这些话题已经超出了本书的范围<sup>①</sup>。

集群软件将一个失败结点服务地址重定位到同一类群中的某个存活的结点上。被 Oracle10g Clusterware 赋予的服务 IP 地址叫做 VIP (Virtual IP address, 虚拟 IP 地址)。这是 IP 别名概念的另一个术语。在一个 RAC 集群中, 失败结点的 VIP 被赋予一个存活结点。虚拟 IP 地址采用 IP 别名方法来实现。在 UNIX 用户根目录下使用 `ifconfig` 命令可以轻松实现 IP 别名。下面是一个在 Linux 操作系统中添加一个服务 IP 地址到适配器 `eth1` 中的例子:

```
# ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 00:0C:29:07:84:EC
          inet addr:172.168.0.1  Bcast:172.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe07:84ec/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          ...
# ifconfig eth1:1 inet 172.168.0.11 # IP aliasing on adapter eth1
# ifconfig eth1:1
eth1:1    Link encap:Ethernet  HWaddr 00:0C:29:07:84:EC
          inet addr:172.168.0.11  Bcast:172.168.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:169 Base address:0x2080
# ping -c 1 172.168.0.11
PING 172.168.0.11 (172.168.0.11) 56(84) bytes of data.
64 bytes from 172.168.0.11: icmp_seq=0 ttl=64 time=0.072 ms
```

① 关于子网划分的讨论, 见 <http://en.wikipedia.org/wiki/Subnetwork>。

```

--- 172.168.0.11 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.072/0.072/0.072/0.000 ms, pipe 2

```

29.2 多宿主系统

ORACLE 数据库软件安装的系统有可能拥有多于一个的 NIC (Network Interface Controller, 网络接口控制器)。一个机器有可能属于不同的 LAN 段, 或者它有可能拥有一个单独的网络适配器用来备份和恢复流量。运行 RAC 的聚类系统有一个独立的私有网络叫做内部连接, 用于在每个 RAC DBMS 实例中相互通信。拥有几个网络接口控制器的机器 (即网络适配器) 有时被称为是多宿主 (multihomed) 的, 因为它们存在于多个网络当中。要是人能够像机器一样幸运, 能够同时拥有多个家却不用付额外的税该多好。

术语“多宿主”这个概念有时很模糊。从广义上讲, 它可以用来描述一台机器拥有多个网络适配器, 并连接到一个或者多个网络。从狭义上讲, 它只适用于为获得更高的可靠性而通过多于一个的网络进行网络访问的情况<sup>①</sup>。

有时候, 网络管理员允许在某特定 LAN 段的客户访问位于一个多宿主系统的 DBMS 实例, 而拒绝其他 LAN 段的客户。为了实现这一点, DBA 需要确保 TNS 监听器能够在某个 LAN 段可见, 而在其他段不可见。从技术角度来看, TNS 监听器应该和其中某个适配器的 IP 地址绑定, 而不是其他适配器。

现在是时候讲述那个令人尴尬的缩写词 INADDR\_ANY, 这个词在本章开头的状态一节中提到过。INADDR\_ANY 是一种 C 语言预处理宏, 用于 C 语言套接字编程接口中。在遵循 POSIX 的系统中, 实现了采用套接字 (见 man socket) 的 TCP/IP 和 UDP 网络技术。对于一个服务进程, 例如 TNS 监听器, 套接字编程流程是。

- (1) 创建一个套接字。
- (2) 采用 INADDR\_ANY 将套接字和一个地址绑定。
- (3) 监听请求连接。
- (4) 接受请求连接 (或者拒绝, 第 30 章中有效结点检查中将会讨论)。

这 4 个步骤的 C 语言例程分别为 socket、bind、listen 以及 accept。INADDR\_ANY 告诉操作系统套接字的创造者愿意和任意与它联系的外部系统通信。Solaris 文档中如下描述 INADDR\_ANY[So18 2000]:

通过使用 INADDR\_ANY 这个特定值设置 IP, 或不指定 IPv6 的值 (全为零) 两种方式, 可以实现调用 bind() 时无需指定本地 IP, 这里 bind() 调用可以是主动 TCP 套接字也可以是被动的 TCP 套接字调用。这个特性一般用于本地地址未知或不相关的情况。如果没有指定本地 IP 和 IPv6, 那么在连接时它们将会与用来服务连接的网络接口的地址绑定。

与此相反的是只接受来自某特定网络的主机的连接请求。此时 INADDR\_ANY 的值是一个由全

<sup>①</sup> 狭义定义可参见 Wikipedia, <http://en.wikipedia.org/wiki/Multihoming>。

零组成的 IP 地址 (0.0.0.0)。要保证 TNS 监听器按照预期的运行需要以下三种工具：

- ❑ telnet, 建立一个到 TNS 监听器的连接并保证监听器确实对在一个端口进行监听；
- ❑ netstat, 列出网络连接（通用的）和相关的进程（仅适用于一部分操作系统）；
- ❑ UNIX 系统调用跟踪工具例如 strace (Linux)、truss (Solaris, AIX), 或者 tusc (HP-UX)。

UNIX 系统调用是 UNIX 操作系统内核的接口。POSIX (Portable Operating System Interface, 可移植操作系统接口) 标准是 UNIX 系统调用的专用标准。这个标准由 Open Group® 维护。当然, 也可以应用 tns ping, 但是根据文献 (*Oracle Database Net Services Administrator's Guide 10g Release 2*) 规定, 它需要一个网络服务名称作为参数。幸运的是, 未在相关文档中说明的特性提供了一种节省大量时间的方法, 在命令行中指定地址, 如下例所示:

```
$ tns ping '(ADDRESS=(PROTOCOL=TCP)(Host=172.168.0.1)(Port=1521))'
TNS Ping Utility for Linux: Version 10.2.0.3.0 - Production on 22-JUL-2007 20:13:38
Copyright (c) 1997, 2006, Oracle. All rights reserved.
Attempting to contact (ADDRESS=(PROTOCOL= TCP)(Host=172.168.0.1)(Port=1521))
OK (10 msec)
```

地址规范完全遵循了 tnsnames.ora 中 DESCRIPTION 的相同语法, 但是忽略了与 tns ping 无关的 CONNECT\_DATA 和 FAILOVER\_MODE 两部分。在 UNIX 系统中, 引号是必需的, 因为圆括号对于 shell 有着特殊的意义。从 Oracle10g 开始, tns ping 也支持使用简单的连接格式 host\_name : port /instance\_service\_name, 这种方式不需要网络服务名称解析。

```
C:> tns ping dbserver:1521/ten.oradbpro.com
TNS Ping Utility for 32-bit Windows: Version 10.2.0.1.0 - Production on 14-DEC-2007
18:58:59
Copyright (c) 1997, 2005, Oracle. All rights reserved.
Used parameter files:
C:\oracle\admin\network\admin\sqlnet.ora
Used HOSTNAME adapter to resolve the alias
Attempting to contact (DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=ten.oradbpro.com))
(ADDRESS=(PROTOCOL=TCP)
(HOST=169.254.212.142)(PORT=1521)))
OK (10 msec)
```

当使用 tns ping 这种简单的连接方式时, instance\_service\_name 是可选项。如果指定了该项, tns ping 也没有对该项进行校验。

29.3 IP=FIRST 未启用

让我们来研究一下当 IP=FIRST 不可用时会是什么情况。由于 Oracle9i 不支持 IP=FIRST, 下面的测试都是针对 Oracle10g 做的。但是, Oracle9i TNS 监听器的行为和 Oracle10g 及 Oracle11g 没有 IP=FIRST 时 TNS 监听器的行为相同。测试系统的主机名称为 dbserver1.oradbpro.com。这个主机名称映射的引导 IP 地址是 172.168.0.1。

① 见 <http://www.pasc.org>。

```
$ ping -c 1 `hostname`
PING dbserver1.oradbpro.com (172.168.0.1) 56(84) bytes of data.
64 bytes from dbserver1.oradbpro.com (172.168.0.1): icmp_seq=0 ttl=64 time=0.011 ms
```

29.3.1 主机名

为了验证这一点，在 listener.ora 中关于 TNS 监听器的设置包含了测试系统的主机名称。

```
LISTENER=
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST=dbserver1.oradbpro.com)(PORT=1521))
    )
  )
```

回到我们偏离的关于 UNIX 的套接字网络编程上来，我们来看看当 IP=FIRST 不可用时 TNS 监听器调用 bind 的情况。在 Linux 系统环境下，可以用 strace 工具来跟踪 TNS 监听器的 bind 系统调用。应用 strace 启动和跟踪一个进程的基本语法如下：

```
strace -f -o output_file command arguments
```

选项 -f 使 strace 跨过叉点（创建子进程的点）进行跟踪，-o 用来指定 strace 写入它捕获的系统调用的输出文件名称。那么我们来用 strace 启动一个 TNS 监听器。

```
$ strace -f -o /tmp/strace_no_ip_first.out lsnrctl start
```

没有 IP=FIRST 的约束，跟踪的输出包含一个 bind 调用，这个调用包含一个全零的请求地址 (INADDR\_ANY)。

```
$ grep "bind.*1521" /tmp/strace_no_ip_first.out
25443 bind(8, {sa_family=AF_INET, sin_port=htons(1521), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
```

通过对与主机相关的所有 IP 地址运行 telnet IP\_address listener\_port 来验证 TNS 监听器是否接受任意的网络连接。也可以通过使用 netstat 的输出来确认（必须打开 telnet 和 tnslnsr 的多连接才能获得 netstat 输出）。

```
$ netstat -np|egrep 'Proto|telnet|tnslsnr'
```

| Proto | Recv-Q | Send-Q | Local Address | Foreign Address | State | PID/
Program na |
|-------|--------|--------|----------------------|---------------------|-------------|--------------------|
| me | | | | | | |
| tcp | 0 | 0 | 192.168.10.132:50388 | 192.168.10.132:1521 | ESTABLISHED | 25610/teln |
| et | | | | | | |
| tcp | 0 | 0 | 172.168.0.1:50393 | 172.168.0.1:1521 | ESTABLISHED | 25613/teln |
| et | | | | | | |
| tcp | 0 | 0 | 127.0.0.1:50394 | 127.0.0.1:1521 | ESTABLISHED | 25614/teln |
| et | | | | | | |
| tcp | 0 | 0 | 172.168.0.1:1521 | 172.168.0.1:50319 | ESTABLISHED | 25489/tnsl |
| snr | | | | | | |
| tcp | 0 | 0 | 172.168.0.1:1521 | 172.168.0.1:50320 | ESTABLISHED | 25489/tnsl |
| snr | | | | | | |

```

tcp      0      0 192.168.10.132:1521 192.168.10.132:50388 ESTABLISHED 25489/tnsl
snr
tcp      0      0 172.168.0.1:1521    172.168.0.1:50393    ESTABLISHED 25489/tnsl
snr
tcp      0      0 127.0.0.1:1521      127.0.0.1:50394      ESTABLISHED 25489/tnsl
snr

```

选项-n使 netstat 使用数字而不是名称来表示主机名称和端口。选项-p (Linux 专有) 显示进程的名称。如果使用了-l 选项, 那么在 netstat 的输出 Local Address 栏里将显示 INADDR\_ANY 的值 0.0.0.0。这个选项限制了只输出状态为 LISTEN 的套接字, 也就是还没有和客户端程序相连接的套接字。

```

$ netstat -tnlp | egrep 'Proto|tns'
Proto Recv-Q Send-Q Local Address Foreign Address State      PID/Program name
tcp      0      0 0.0.0.0:1521  0.0.0.0:*      LISTEN    25489/tnslsnr

```

选项-t 限制输出只能到 TCP 套接字。在 Windows 操作系统中, 可以使用选项-abno、-p、tcp 来达到和 UNIX 系统一样的输出。

```

C:> netstat -abno -p tcp
Active Connections
Proto Local Address          Foreign Address        State               PID
TCP   0.0.0.0:1521            0.0.0.0:0              LISTENING           4524
[TNLSNR.exe]
TCP   127.0.0.1:1521          127.0.0.1:2431         ESTABLISHED         4524
[TNLSNR.exe]
TCP   127.0.0.1:2431          127.0.0.1:1521         ESTABLISHED         4836
[telnet.exe]
TCP   192.168.10.1:1521       192.168.10.1:2432      ESTABLISHED         4524
[TNLSNR.exe]
TCP   192.168.10.1:2432       192.168.10.1:1521      ESTABLISHED         4224
[telnet.exe]

```

29.3.2 回环适配器

你可能会惊讶地发现 TNS 监听器也可以和回环适配器 (loopback adapter) 的 127.0.0.1 地址相连接。回环网络适配器可以在没有网络连接的情况下, 提供一个机器范围内的 IP 网络服务, 但是要求 TCP/IP 为需要它的应用程序服务。在主机的配置文件中, 主机名称 localhost 映射到了 IP 地址 127.0.0.1。这个配置文件在 UNIX 系统中对应/etc/hosts, 在 Windows 系统内对应%SYSTEM\_ROOT%\system32\drivers\etc\hosts 这个文件。手提电脑 (laptop computer) 在没有网络连接的情况下, 回环 IP 地址被显式地赋予 TNS 监听器。这个设置保证了在没有网络连接的情况下, 也可以通过 TCP/IP 使用 TNS 监听器。另一种通过使用 TNS 监听器实现本地连接的方式是 IPC 协议。可以用以下 telnet 命令来验证是否有一个进程在监听回环地址的端口 1521。

```

$ telnet localhost 1521
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

```

由于 telnet 没有以错误 Connection refused 结束, TNS 监听器接受了 telnet 的连接请求。strace 的输出包括一个独立的对 IP 地址 127.0.0.1 的 bind 调用。

```
25443 bind(12, {sa_family=AF_INET, sin_port=htons(0),
sin_addr=inet_addr ("127.0.0.1")}, 16) = 0
```

29.3.3 引导 IP 地址

在 Oracle11g 之前的版本中, 当 listener.ora 中用引导 IP 地址替换了系统的主机名称时, TNS 监听器将不会与 INADDR\_ANY 绑定的情况并没有在文档中说明。下面是使用引导 IP 地址后所修改的 TNS 监听器配置:

```
LISTENER=
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL=TCP)(HOST=172.168.0.1)(PORT=1521))
  )
)
```

重启 TNS 监听器之后, 本地地址就与修改过的 IP 地址对应起来了。

```
$ netstat -tnlp|grep tns
Proto Recv-Q Send-Q Local Address      Foreign Address    State    PID/Program name
tcp        0      0 172.168.0.1:1521  0.0.0.0:*           LISTEN   25630/tnslsnr
```

这样设置之后, TNS 监听器就不能与回环地址或者任何其他没有明确配置的 IP 地址相连接。

```
$ telnet localhost 1521
Trying 127.0.0.1...
telnet: connect to address 127.0.0.1: Connection refused
$ tnsping 192.168.10.132:1521
TNS Ping Utility for Linux: Version 10.2.0.3.0 - Production on 08-SEP-2007 15:47:25
Copyright (c) 1997, 2006, Oracle. All rights reserved.
Used parameter files:
/opt/oracle/product/db10.2/network/admin/sqlnet.ora
Used HOSTNAME adapter to resolve the alias
Attempting to contact (DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=192.168.10.132))
(ADDRESS=(PROTOCOL=TCP)(HOST= 192.168.10.132)(PORT=1521)))
TNS-12541: TNS:no listener
```

在这种配置下, 使用 strace 跟踪时, 引导 IP 地址用于 bind 调用过程中:

```
$ strace -f -o /tmp/strace_boot_ip.out lsnrctl start
$ grep "bind.*1521" /tmp/strace_boot_ip.out
25689 bind(8, {sa_family=AF_INET, sin_port=htons(1521),
sin_addr=inet_addr("172.168.0.1")}, 16) = 0
```

29.3.4 服务 IP 地址

为了完成关于 TNS 监听器未在文档中说明的特性的调查, 我们来看看当给 TNS 监听器配置一个与服务 IP 地址相对应的主机名称时会发生什么情况。这里用前面定义的适配器 eth1:1 的别

名 IP 地址 172.168.0.11 作为示例。配置文件/etc/hosts 将这个 IP 地址与主机名 vip-dbserver1 相关联。

```
$ grep 172.168.0.11 /etc/hosts
172.168.0.11    vip-dbserver1.oradbpro.com    vip-dbserver1
```

此时，在 listener.ora 中的 ADDRESS 实体变为：

```
(ADDRESS=(PROTOCOL=TCP)(HOST=vip-dbserver1)(PORT=1521))
```

当赋予了一个和服务 IP 地址（即非引导 IP 地址）相对应的主机名称时，TNS 监听器明确地与此 IP 地址绑定而不使用 INADDR\_ANY。下面的 netstat 的输出证明了这一点：

```
$ netstat -tnlp|egrep 'Proto|tns'
Proto Recv-Q Send-Q Local Address      Foreign Address    State    PID/Program name
tcp      0      0 172.168.0.11:1521  0.0.0.0:*          LISTEN   27050/tnslsnr
```

进一步实验表明，INADDR\_ANY 也不能用于非引导 IP 地址。

29.4 IP=FIRST 开启

为了使 IP=FIRST 可用，在 listener.ora 文件中的 ADDRESS 行必须按照以下进行修改：

```
(ADDRESS=(PROTOCOL=TCP)(HOST=172.168.0.1)(PORT=1521)(IP=FIRST))
```

使用该设置重启 TNS 监听器（使用 lsnrctl 重新加载 listener\_name 是无效的）后，strace 输出文件中的 bind 调用就改变了。之前是 0.0.0.0 (INADDR\_ANY)，现在是系统的主机名称对应的 IP 地址。

```
bind(8, {sa_family=AF_INET, sin_port=htons(1521),
sin_addr=inet_addr("172.168.0.1")}, 16) = 0
```

同时，TNS 监听器在分配地址 172.168.0.1 上也可以运行。

```
$ telnet 172.168.0.1 1521
Trying 172.168.0.1...
Connected to 172.168.0.1.
Escape character is '^]'.
Connection closed by foreign host.
```

几分钟后，TNS 监听器关闭了和 telnet 的连接。输入 Ctrl+C、Ctrl+D 并点击返回来终止连接。正如以上例子中，程序 telnet 随后显示 Connection closed by foreign host 信息。既然 IP=FIRST 在起作用，尝试使用其他除了引导 IP 之外的任意 IP 地址与 TNS 监听器连接都会失败。

```
$ telnet 77.47.1.187 1521
Trying 77.47.1.187...
telnet: connect to address 77.47.1.187: Connection refused
$ telnet localhost 1521
Trying 127.0.0.1...
telnet: connect to address 127.0.0.1: Connection refused
# netstat -tnlp | egrep 'Proto|tns'
Proto Recv-Q Send-Q Local Address      Foreign Address    State    PID/Program name
tcp      0      0 172.168.0.1:1521  0.0.0.0:*          LISTEN   27916/tnslsnr
```

表 29-1 中总结了所有可能出现的情况下 TNS 监听器的 IP 地址绑定行为。注意对于引导 IP 地址、非引导 IP 地址或主机名称、服务 IP 地址或主机名称，TNS 监听器的行为是完全相同的，与 IP=FIRST 这个选项无关。换句话说，IP=FIRST 仅仅当系统的主机名称用于 listener.ora 文件时才起作用。

表29-1 TNS监听器IP地址绑定

| 在listener.ora中HOST设置 | IP=FIRST | 引导IP地址 | 其他的非引导IP地址或服务IP地址（IP别名） | 回环地址 |
|-----------------------|----------|--------|-------------------------|------|
| 系统主机名称
(映射到引导IP地址) | 是 | 是 | 否 | 否 |
| 系统主机名称
(映射到引导IP地址) | 否 | 是 | 是 | 是 |
| 引导IP地址 | 否 | 是 | 否 | 否 |
| 非引导IP地址/主机名 | 否 | 否 | 否 | 否 |
| 服务IP地址/主机名 | 否 | 否 | 否 | 否 |

表的第一列代表了当在配置文件 listener.ora 中使用主机名并设置了 IP=FIRST 时 TNS 监听器的行为。第 3~5 列显示了在第 1, 2 列的设置下，IP 地址的绑定行为。“是”表示 TNS 监听器与该列表头的 IP 地址相绑定。由此可见，只有在进行了第一列设置的情况下，TNS 监听器才与引导 IP 绑定。

29.5 小结

本章探讨了在多宿主系统中 TNS 监听器的 IP 地址绑定，以及在回环适配器上的应用。发现了以下几个未在文档中说明的特点。

- 在赋予系统主机名称的情况下，TNS 监听器使用 INADDR\_ANY 对任何网络可见，包括通过 IP 地址恒为 127.0.0.1 的回环适配器。
- 在赋予引导 IP 地址或非引导 IP 地址或主机名称的情况下，TNS 监听器不使用 INADDR\_ANY，而是明确地和赋予的地址相绑定。
- 当选项 IP=FIRST 可用时，TNS 监听器明确地与配置的主机名称对应的 IP 地址绑定，并且对其他任意 IP 不可见，包括回环地址。只有当系统的主机名称赋给 listener.ora 中的 HOST 参数时，这个选项才相关。

因此，实现 TNS 监听器不对任意网络可见有如下三种方法。

- 使用引导 IP 地址，而不使用系统的主机名称（映射到引导 IP 地址）。
- 使用非引导 IP 地址或服务 IP 地址或主机名称（既不能是系统主机名称，也不能是与引导 IP 地址对应的主机名称）。
- 在 listener.ora 中引用系统主机名称时配置选项 IP=FIRST（需要 Oracle10g 及其以后的版本）。

TNS 监听器 TCP/IP 有效结点检验

监听器有效结点检查可用于阻止 DBMS 实例恶意或错误地与 Oracle Net 连接。它是由 DBA 控制的“可怜人的防火墙”。DBMS 产品实例可以在不添加任何硬件及防火墙软件的情况下与测试实例和开发实例区分开,只需要设置监听器可见的结点列表。有效结点检验已经有文档说明了,但是在 Oracle10g 和 Oracle11g 中参数可以动态设置这一情况并没有在文档中说明。例如配置可以在不终止和重启 TNS 监听器的情况下被启用、改变和删除。这使得有效结点检验这个特性具有较小的扰乱性。

30.1 有效结点检验简介

有效结点检验是一个吸引人的安全特性,使 DBMS 实例远离通过 TCP/IP 的恶意的或错误的 Oracle Net 连接,而不需要操作系统层面的防火墙和 IP 地址过滤。这个特性在 Oracle9i 及其之后的版本中可用,并且不需要额外成本。

这里有一个有趣的例子说明了为什么有效结点性检验是一个有价值的特性。一个拥有几个数据库作业的生产数据库被复制到一台测试机器上。这些数据库作业开始运行,并且其中几个正在使用数据库链接。由于数据库链接定义包含一个完全的 Net 服务名称定义,而不是引用 tnsnames.ora (一个没有在文档中说明的特性) 里面的一个 Net 服务名称,所以测试系统可以访问一个关键的生产系统并引起生产系统的性能衰退。由于这些作业都是只读的,管理员对此也无能为力。想象一下如果这些作业修改了生产数据将会产生什么后果。正确配置的有效结点检验能够避免这个问题。

这个特性由 tcp.validnode\_checking、tcp.invited\_nodes 和 tcp.excluded\_nodes3 个参数控制,如表 30-1 所示。

表30-1 有效结点检验参数

| 名 称 | 目 的 | 值 | 默 认 值 |
|------------------------|------------|---------|-------|
| tcp.validnode_checking | 控制有效结点检验开关 | yes, no | no |

(续)

| 名 称 | 目 的 | 值 | 默 认 值 |
|--------------------|---------------------|---------------------------|-------|
| tcp.invited_nodes | 可能与TNS监听器建立连接的结点列表 | 同一行中逗号分开的主机名称和/或IP地址组成的列表 | 空 |
| tcp.excluded_nodes | 不允许与TNS监听器建立连接的结点列表 | 同一行中逗号分开的主机名称和/或IP地址组成的列表 | 空 |

明显的是，结点有效检验的代码总是执行。只要这个特性不可用，被邀请的和被排除在外的主机列表都为空，此时允许每个客户机的连接。这个推断基于对 TNS 监听器的输出，例如下面是客户到跟踪文件的所有连接的节选，和 tcp.validnode\_checking 设置无关：

```
[12-JUL-2007 20:03:12:268] nttnp: Validnode Table IN use; err 0x0
[12-JUL-2007 20:03:12:268] ntvlser: valid node check on incoming node 10.6.6.64
```

只有当 TNS 监听器的跟踪级别至少为 ADMIN 并且 tcp.validnode\_checking=no 时，跟踪文件中才有迹象表明有效结点检验被关闭：

```
[12-JUL-2007 19:52:28:329] ntvllt: tcp.validnode_checking not turned on
```

让我们来看一下例子。提示符 dbserver\$和 clients\$表明每个命令的运行位置。服务器的 listener.ora 和 sqlnet.ora 两个文件如下所示：

```
dbserver$ head -200 listener.ora sqlnet.ora
==> listener.ora <==
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST =dbserver.oradbpro.com)(PORT = 1521))
      )
    )
  )
trace_level_listener=admin
==> sqlnet.ora <==
NAMES.DIRECTORY_PATH= (TNSNAMES)
```

有效结点检验目前是关闭的，可从上面代码示例中的 sqlnet.ora 配置文件证实。下面是客户机的 tnsnames.ora 文件：

```
client$ cat tnsnames.ora
TEN_TCP.WORLD =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST=dbserver.oradbpro.com)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = TEN)
    )
  )
)
```

我们重启 TNS 监听器并验证一下客户机不能连接到它。为简洁起见，由于 lsnrctl 的输出没

有显示是否配置了有效结点检验，所以省略了 `lsnrctl` 输出。

```
dbserver$ lsnrctl start
```

由于前面的 `sqlnet.ora` 文件没有包含三个有效结点检验参数的任何一个，所以这个特性无效，因而每个客户机都能成功连接。

```
client$ sqlplus -l ndebes/secret@ten_tcp.world
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
SQL> EXIT
Disconnected from Oracle Database 10g Enterprise Edition Release
10.2.0.1.0 - Production
```

30.2 在运行时打开和修改有效结点检验

我已经说过在 `Oracle10g` 和 `Oracle11g` 中，可以动态地打开有效结点检验，也就是说不必关闭和重启 TNS 监听器。注意，以上情况只有当配置文件 `sqlnet.ora` 在 TNS 监听器启动时存在的情况下才成立。让我们通过改变 `sqlnet.ora` 来验证这个说法，下面是在服务器端运行 `lsnrctl reload`<sup>①</sup>：

```
dbserver$ cat sqlnet.ora
NAMES.DIRECTORY_PATH=(TNSNAMES)
tcp.validnode_checking=yes
tcp.excluded_nodes=(client.oradbpro.com)
dbserver$ lsnrctl reload
```

`lsnrctl reload` 的输出没有显示有效结点检验是否开启。现在，从客户机系统 `client.oradbpro.com` 发起的连接尝试失败。

```
client$ sqlplus -l ndebes/secret@ten_tcp.world
SQL*Plus: Release 10.2.0.1.0 - Production on Fri Jul 13 02:06:33 2007
ERROR:
ORA-12537: TNS:connection closed
SP2-0751: Unable to connect to Oracle. Exiting SQL*Plus
```

当然，必须配置 DNS、NIS，或者其他的方法，实现客户机名称到 IP 地址的转换。IP 地址也必须相应写在受邀请的或排除在外的主机列表当中。如果 TNS 监听器的跟踪级别至少为 `USER`，像下面这样表示客户机连接被拒绝的记录就会写入 TNS 监听器的跟踪文件：

```
13-JUL-2007 02:21:02:109] nttvlser: valid node check on incoming node 88.215.114.53
13-JUL-2007 02:21:02:109] nttvlser: Denied Entry: 88.215.114.53
```

设置受邀请的结点列表，将 `client.oradbpro.com` 添加到该列表，并运行另外一个 `reload`，使得客户机可以连接到服务器。

```
dbserver$ cat sqlnet.ora
tcp.validnode_checking=yes
```

① 事实表明，`LSNRCTL` 工具缓存了 TNS 监听器的配置信息。当进行测试时，你应该在命令行上运行 `lsnrctl` 命令，而不能打开这个工具在 `LSNRCTL` 提示符下输入多行命令。后一种方法可能不能检测到 `listener.ora` 文件的变化。

```
tcp.invited_nodes=(client.oradbpro.com)
dbserver$ lsnrctl reload
client$ sqlplus -l ndebes/secret@ten_tcp.world
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
```

如下所示，TNS 监听器的跟踪文件记录了客户机的成功连接：

```
[13-JUL-2007 02:24:44:789] nttvlser: valid node check on incoming node 88.215.114.53
[13-JUL-2007 02:24:44:789] nttvlser: Accepted Entry: 88.215.114.53
```

如果设置了 tcp.invited\_nodes，那么不在这个列表中的结点的访问请求都会被拒绝。

```
dbserver$ cat sqlnet.ora
tcp.validnode_checking=yes
tcp.invited_nodes=(192.168.0.1)
dbserver$ lsnrctl reload
client$ sqlplus -l ndebes/secret@ten_tcp.world
SQL*Plus: Release 10.2.0.1.0 - Production on Fri Jul 13 02:06:33 2007
ERROR:
ORA-12537: TNS:connection closed
SP2-0751: Unable to connect to Oracle. Exiting SQL*Plus
```

当然，被拒绝的主机也包括 TNS 监听器运行的系统，使得随后的通过 TCP/IP 的 LSNRCTL 命令失效。为了使基于 TCP/IP 的 LSNRCTL 命令有效，你需要将本地系统包含在 tcp.invited\_nodes 列表里面。另外一种方法是将 IPC 协议作为 TNS 监听器的第一个 ADDRESS。这个设置告诉 LSNRCTL 工具使用 IPC 与 TNS 监听器通信，从而显然避免了 TCP/IP 的有效结点检验。下面这个例子给出了使用 IPC 协议作为第一个 ADDRESS 实体的 TNS 监听器定义：

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = TEN))
        (ADDRESS = (PROTOCOL = TCP)(HOST =dbserver.oradbpro.com)(PORT = 1521))
      )
    )
  )
```

在默认设置下，Oracle9i 的 TNS 监听器会受到来自远程结点的 STOP 命令的影响。除了应用 TNS 监听器密码来防止其他主机上的人关闭 TNS 监听器之外，也可以应用有效结点检验。该方法的缺点是有可能访问 TNS 监听器的机器都必须包含在受邀请的结点列表里面。这些机器仍然可以远程停止 TNS 监听器，但是应该是可信系统。这个方法对于运行聚类软件的装置来说很有吸引力，因为聚类软件防止 ORACLE TNS 监听器受到结点故障的影响，但不支持 TNS 监听器密码设置（例如，VERITAS Cluster Server 4 之前的版本）。

如果你想采用更轻松的方法，可以只设置 tcp.excluded\_nodes 参数，并将不可能连接到 TNS 监听器的系统添加到参数值列表中，从而与 TNS 监听器服务的实例区分开。所以没提到的结点都可以与 TNS 监听器建立连接。可同时使用主机名和 IP 地址。

同时设置 `tcp.invited_nodes` 和 `tcp.excluded_nodes` 两个参数没什么意义, 因为只要设置了 `tcp.invited_nodes` 参数, 即使没有明确说明为排除在外的结点也会被排除在外。如果一个结点的名称同时包含在 `tcp.excluded_nodes` 和 `tcp.invited_nodes` 当中, `tcp.invited_nodes` 占据优先地位, 这个结点允许被访问。在 Oracle9i 中, 一个结点名称不能解析为 IP 地址时, 跟踪文件就会记录相应的错误信息:

```
[12-JUL-2007 21:25:10:162] nttnp: Validnode Table **NOT** used; err 0x1f7
```

当出现这类错误时, 有效结点检验就会关闭。不幸的是, Oracle9i LSNRCTL 工具并没有向终端写入错误信息。遇到无效的主机名称时, Oracle10g lsnrctl startup 输出失败信息 TNS-12560: TNS: protocol adapter error 和 TNS-00584: Valid node checking configuration error。对于 TNS-00584 应用 oerr 得到:

```
$ oerr tns 584
00584, 00000, "Valid node checking configuration error"
// *Cause:Valid node checking specific Oracle Net configuration is invalid.
// *Action:Ensure the hosts specified in the "invited_nodes" and "excluded_nodes"
// are valid. For further details, turn on tracing and reexecute the operation.
```

如果 TNS 监听器的跟踪文件打开, 那么文件中将包含如下类似的消息:

```
[12-JUL-2007 23:27:16:808] snlinGetAddrInfo: Name resolution failed for
wrong.host.name
[12-JUL-2007 23:27:16:808] nttnp: Validnode Table **NOT** used; err 0x248
```

除了名称解析失败, 使用 lsnrctl reload 加载配置文件成功, 这个错误信息按如下形式记录下来:

```
[13-JUL-2007 00:11:53:427] snlinGetAddrInfo: Name resolution failed for
wrong.host.name
```

对 IP 地址不会进行反向地址转换。因此, 不能转换为主机名称的 IP 地址不会妨碍有效结点检验的运行。可以使用操作系统工具 nslookup 来实现 DNS (Domain Name Service, 域名服务) 主机名称和 IP 地址的转换, 反向转换也可以。要注意的是 nslookup 没有读主机文件 (在 UNIX 系统中是 /etc /hosts; 在 Windows 系统中是 %SYSTEM\_ROOT%\system32\drivers\etc\hosts, 这里 SYSTEM\_ROOT 通常是 C:\WINDOWS)。所以 TNS 监听器可以通过调用 C 编程语言库例程 (gethostbyaddr(), gethostbyname()) 来解析名称或者 IP 地址, 但是 nslookup 不会。

我做了些实验来测试文档中未提到的受邀请的和排除在外的结点列表的可接受的最大长度。我所使用的 Vi 编辑器所允许的最大长度是 2 048 字节<sup>①</sup>。在这个长度下, 这两个参数都能正常运行。假设每个主机名称的平均长度是 30 字节, 那么这个长度可以容纳大概 65 个实例。如果使用 IP 地址, 至少可以容纳 128 个 IP 地址。有效结点列表不能超过一行, 否则会出现 TNS-00583: Valid node checking : unable to parse configuration parameters 错误, 并且 TNS 监听器不能启动。

① 其他编辑器允许一行的长度大于 2 048 个字节。Vim (Vi 升级版) 是 Vi 编辑器的增强版, 支持多于 2 048 字节的行。可以从网址 <http://www.vim.org> 免费获得, 该编辑器可以运行在 UNIX、Windows 和 Mac OS 系统上。

本地命名参数设置 `ENABLE = BROKEN` 未在文档中说明。这个参数可用于 Net 服务名称定义，用来启动检测通信故障的 TCP/IP 保持连接包（keep\_alive packet）的发送。保持连接包并不是通常意义上通过 TCP/IP 连接发送的包。另外，有些 TCP/IP 协议的超时设置是在数小时范围内，从而使得检测通信故障的时长让人无法接受。操作系统的 TCP/IP 参数会影响到所有的 TCP/IP 连接，`ENABLE = BROKEN` 参数设置允许在不改变这些参数的前提下，合理快速地检测通信故障。为了更好地使用 `ENABLE = BROKEN`，需要修改 TCP/IP 保持连接参数的默认值，这些修改并不影响没有开启保持连接包的 TCP/IP 连接。

结点故障与 TCP/IP 协议

在支持虚拟 IP 地址的 Oracle10g Clusterware 问世之前，RAC 集群中一个故障结点会让所有与它连接的客户机等待一个加长时间，很可能会达到两个小时——这是 TCP/IP 默认的连接超时时长。`ENABLE = BROKEN` 应对了 RAC 的高效性环境。它缩短了由于 RAC 集群中一个意外结点故障而使其他与之连接的客户机等待的时间段。

从语法上讲，`ENABLE = BROKEN` 设置和其他如 `FAILOVER` 等的高效性相关参数一样，属于 DESCRIPTION 段。下面是一个使用两个结点的 RAC 集群和 TAF（Transparent Application Failover，透明应用程序故障转移）的 Net 服务名称定义：

```
DBSERVER_TAF.WORLD =
  (DESCRIPTION =
    (ENABLE=BROKEN)
    (FAILOVER=ON)
    (LOAD_BALANCE=OFF)
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = dbserver1.oradbpro.com)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = dbserver2.oradbpro.com)(PORT = 1521))
    )
  )
  (CONNECT_DATA =
    (SERVICE_NAME = TEN)(SERVER = DEDICATED)
    (FAILOVER_MODE=
      (TYPE=SELECT)
      (METHOD=BASIC)
```



```

        (DELAY=5)
        (RETRIES=600)
    )
)
)

```

ENABLE = BROKEN 选项至少从 Oracle8 才可用,它控制着网络连接的保持连接选项的开关。UNIX 网络编程是基于套接字接口的。一个套接字是一个通信的终结点。又一次地, UNIX 系统调用跟踪是观测运行情况的必选工具。下面是 Solaris 系统的一个例子,这里并没有设置 ENABLE = BROKEN:

```

$ truss -o /tmp/truss-no-enable-broken.out -t so_socket, setsockopt \
-v so_socket, setsockopt sqlplus system@DBSERVER_TAF.WORLD
$ cat /tmp/truss-no-enable-broken.out
so_socket(PF_INET, SOCK_STREAM, IPPROTO_IP, "", 1) = 8
setsockopt(8, tcp, TCP_NODELAY, 0xFFFFFFFF7FFF6924, 4, 1) = 0

```

设置 ENABLE=BROKEN 之后, 重复以上操作会得到如下结果:

```

so_socket(PF_INET, SOCK_STREAM, IPPROTO_IP, "", 1) = 8
setsockopt(8, SOL_SOCKET, SO_KEEPALIVE, 0xFFFFFFFF7FFF665C, 4, 1) = 0
setsockopt(8, tcp, TCP_NODELAY, 0xFFFFFFFF7FFF6924, 4, 1) = 0
setsockopt(8, SOL_SOCKET, SO_KEEPALIVE, 0xFFFFFFFF7FFF529C, 4, 1) = 0
setsockopt(8, SOL_SOCKET, SO_KEEPALIVE, 0xFFFFFFFF7FFFD2C, 4, 1) = 0

```

增加的 3 个 setsockopt 系统调用发生在设置 ENABLE = BROKEN 之后。开启发送保持连接包的 C 语言标志是 SO\_KEEPALIVE, 正如上面的输出一样。

在 Solaris 系统中, 有接近 80 个参数控制着 TCP/IP 的网络设置。Solaris Tunable Parameters Reference Manual [SoTu 2007] 这样描述 SO\_KEEPALIVE (nnd 是 Solaris 系统中改变 TCP/IP 参数设置的一个实用工具):

tcp\_keepalive\_interval

这个 nnd 参数设置了一个探测间隔, 当一个 TCP 连接在一个系统范围内空闲时, 这个时间间隔就会首先被发出去。

Solaris 支持 RFC 1122 中描述的 TCP keep-alive 机制。通过在一个 TCP 套接字上设置 SO\_KEEPALIVE 套接字选项可以开启该机制。

为一个套接字开启 SO\_KEEPALIVE 功能后, 当 TCP 连接空闲两个小时后, 第一个 keep-alive 探测就会发出。这里的两个小时是 tcp\_keepalive\_interval 参数的默认值。如果对等实体在 8 分钟之后没有对探测作出反应, TCP 连接就会断开。如想获得更多信息, 请参阅 tcp\_keepalive\_abort\_interval。

你可以通过设置 TCP\_KEEPALIVE\_THRESHOLD 套接字选项来修改默认的等待时长, 使每个应用都可以在每个套接字上设置自己的等待时长。设置值可以是百万秒之内的无符号整数值。参见 tcp (7P)。

Solaris 手册随后说明这个参数的保证是不可靠的, 所以应该保持其默认值不变。据我的了解, ORACLE DBMS 软件没有实现 tcp\_keepalive\_threshold。因此, 为了替代修改 keep-alive 设置,

Solaris 文档推荐修改重新传输超时 (tcp\_rexmit\_interval\_max 和 tcp\_ip\_abort\_interval)。

前段时间，我在 Oracle8i 环境下测试证明了在 tcp\_keepalive\_interval 和 tcp\_keepalive\_abort\_interval 按照需要设置好之后，ENABLE = BROKEN 功能实用。使用 truss 工具跟踪可以证明 Oracle10g 也实现了该功能。记住，在任何 RAC 集群投入使用之前，要对所有这些 TCP/IP 设置进行适当的测试，使得系统没有任何机会给远程系统发消息，从而表示这些套接字应该关闭了。这些测试包括拔掉网络电缆（保持拔掉的状态）、关掉服务器、或者其他任何能使系统垮掉（在 Solaris 系统中使用 Stop + A）的方法。

转换话题之前，我来解释一下为什么 ENABLE = BROKEN 会被认为是一个过时的特征。应用 Oracle10g 集群软件和虚拟 IP 地址技术，与一个故障主机连接的 IP 地址可以在存活的节点上恢复连接。客户机的 re-transmits 会被重定向到一个存活结点，由于不知道该失效结点开放的套接字，所以会失败。作为虚拟地址的失效备份，Oracle Clusterware 重刷 ARP(Address Resolution Protocol，地址解析协议)高速缓存，该存储器实现 IP 地址与局域网适配器 MAC (Medium Access Control，媒体访问控制) 地址的转换。这些并没有在文档中说明，但是对数据库客户机实现成功再连接是必需的。因为在连接过程中需要知道 IP 地址和 MAC 地址的最新对应关系。在 Linux 系统内，可以在 \$ORA\_CRS\_HOME/bin/racgvip 脚本中，执行/sbin/arping -q -U -c 3 -I adapter ip\_address 这个命令实现 ARP 高速缓冲存储器的刷新。无论在 UNIX 系统还是在 Windows 系统中，都可使用 arp -a 命令来显示最新的 MAC 与 IP 地址的对应关系。

关于 TCP/IP 和失效备援主题的其他信息，参见 Metalink note 249213.1。根据这条注释，Sun 公司建议设置 tcp\_keepalive\_interval、tcp\_ip\_abort\_cinterval (阻止到故障结点的连接尝试必须等待 3 分钟) 和 tcp\_ip\_abort\_interval (默认情况下，一个连接在 8 分钟之内没有收到回复就会断开连接)。不幸的是，Metalink 注释中没有提到当套接字 SO\_KEEPAIVE 没有设置时，tcp\_keepalive\_interval 会被忽略。当设置 SO\_KEEPAIVE 时，需要设置 ENABLE = BROKEN。

我建议修改 tcp\_ip\_abort\_cinterval 来防止在虚拟 IP 地址与存活结点建立连接之前，连接尝试被锁定长达 3 分钟。我建议缩减参数 tcp\_ip\_abort\_interval 和 tcp\_rexmit\_interval\_max 的值分别为 45 秒（默认值为 8 分钟）和 30 秒钟（默认值为 3 分钟）。这些参数必须在数据库的客户端机器上修改——记住，当客户机采用这些缩减的超时值时，服务器的性能会下降。表 31-1 说明了无论在什么情况下都要保持所有的超时时段在一分钟之内。

表31-1 推荐的TCP / IP 参数值

| 参 数 | 默 认 值 | 建 议 值 | 单 位 |
|-------------------------|-----------------|-----------------|-----|
| tcp_rexmit_interval_max | 60 000 (60 秒) | 10 000 (10秒) | ms |
| tcp_ip_abort_interval | 480 000 (8分钟) | 45 000 (45 秒) | ms |
| tcp_ip_abort_cinterval | 180 000 (3分钟) | 30 000 (30秒) | ms |

Oracle Net 配置中默认的主机名

配置文件 `listener.ora` 和 `tnsnames.ora` 中的主机名称可以保留不设置这并没有在文档中说明。因此，当主机名变化时可以不修改 `listener.ora` 配置文件。只要 TNS 监听器使用默认的主机名（也就是本地系统的主机名，这个主机名是唯一识别的），当定制用于生成 TNS 监听器配置的脚本时就不必考虑主机名的变化情况。

32.1 默认主机名

在一个网络服务名的描述中，可以略过主机名。略过主机名（或者 IP 地址）的语法简单描述为 `(HOST =)`。如果主机名或 IP 地址是一个空字符串，那它的默认值为本地系统的主机名。本地主机名通过命令 `hostname` (UNIX 系统和 Windows 系统) 获得，也可以通过 C 库例程 `gethostname` 得到。由于 `gethostname` 不是 UNIX 系统调用，所以不能由系统调用跟踪工具，例如 `truss` 获得。下面给出了在命令行上把一个网络服务名称描述传递给 `tnsping` 的例子：

```
$ tnsping "(ADDRESS=(PROTOCOL=TCP)(Host=)(Port=1521))"
TNS Ping Utility for Linux: Version 10.2.0.3.0 - Production on 22-JUL-2007 22:13:07
Attempting to contact (ADDRESS=(PROTOCOL= TCP)(Host=)(Port=1521))
OK (0 msec)
```

在配置文件 `listener.ora` 中也应用了相同的语法格式，如下例所示：

```
LISTENER_DBSERVER1 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS=(PROTOCOL=TCP)(HOST=)(PORT=1521)(IP=FIRST))
    )
  )
```

UNIX 命令 `hostname` 返回系统的主机名。

```
$ ping -c 1 `hostname`
PING dbserver1.oradbpro.com (172.168.0.1) 56(84) bytes of data.
64 bytes from dbserver1.oradbpro.com (172.168.0.1): icmp_seq=0 ttl=64 time=0.072 ms
```

```

--- dbserver1.oradbpro.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.072/0.072/0.072/0.000 ms, pipe 2

```

在配置文件 `listener.ora` 中主机名空缺的情况下，TNS 监听器就会使用命令 `hostname` 返回的主机名。

```

$ lsnrctl start listener_dbserver1
LSNRCTL for Linux: Version 10.2.0.3.0 - Production on 08-SEP-2007 16:32:09
Copyright (c) 1991, 2006, Oracle. All rights reserved.
Starting /opt/oracle/product/db10.2/bin/tnslsnr: please wait...
TNSLSNR for Linux: Version 10.2.0.3.0 - Production
System parameter file is /opt/oracle/product/db10.2/network/admin/listener.ora
Log messages written to /opt/oracle/product/db10.2/network/log/listener_dbserver1.log
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=172.168.0.1)(PORT=1521)))

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=)(PORT=1521)(IP=FIRST)))
STATUS of the LISTENER
-----
Alias                     listener_dbserver1
Version                   TNSLSNR for Linux: Version 10.2.0.3.0 - Production
Start Date                08-SEP-2007 16:32:09
Uptime                    0 days 0 hr. 0 min. 3 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File   /opt/oracle/product/db10.2/network/admin/listener.ora
Listener Log File         /opt/oracle/product/db10.2/network/log/listener_dbserver1.log
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=172.168.0.1)(PORT=1521)))
The listener supports no services
The command completed successfully

```

当然，也可以在配置文件 `tnsnames.ora` 中省略主机名，但是这只对数据库服务器本身有用，因为远程机器上的客户端大部分时候需要与数据库服务器连接。在一个客户端系统里，`(HOST=)` 应该和 `(HOST = client_host_name)` 保持一致，从而使客户端不能与服务器的 TNS 监听器建立连接。

32.2 关闭默认监听器

默认主机名的另一个用途是关闭默认监听器 `LISTENER`。我们假设有多个监听器运行在系统里，每个监听器都只监听一个 DBMS 实例，并且每个监听器都由一个包含它所服务的 DBMS 数据库实例名称的唯一标识。所以没有用到默认名称 `LISTENER`。由于默认监听器不需要在 `listener.ora` 文件中进行配置，如果一个 DBA 忘记为 `lsnrctl` 命令提供监听器名称，这时默认监听器 `LISTENER` 就会启动。如果正确的监听器需要 `$TNS_ADMIN/sqlnet.ora` 里的 `TNS_ADMIN` 特殊

设置来启动有效结点检验，这时会给安全带来问题。监听器的其他安全相关参数，如 `ADMIN_RESTRICTIONS_listener_name`<sup>①</sup>，可能对非默认监听器有效，但对默认监听器配置无效。因此在 `listener.ora` 中用普通的一节关闭默认监听器是很有意义的。

在 UNIX 平台上，1~1 023 的端口号只在根权限下运行。如果 Oracle 监听器使用了这个范围之内的端口号，它会报错 `TNS-12546: TNS: permission denied`。Windows 系统没有为端口号设置上述限制。但是无效的端口号 0 可以用来防止默认监听器启动。下面重现了实现该功能的监听器配置，该配置适用于 UNIX 和 Windows 系统并且与主机名无关：

```
# Disable default listener
LISTENER =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST=)(PORT = 0))
  )
```

这个设置阻止了默认监听器的启动：

```
$ lsnrctl start
...
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=)(PORT=0)))
TNS-01103: Protocol specific component of the address is incorrectly specified
TNS-12533: TNS:illegal ADDRESS parameters
TNS-12560: TNS:protocol adapter error
TNS-00503: Illegal ADDRESS parameters
```

如果需要，错误信息可以作为设置 `TNS_ADMIN` 的提示信息，并为 `lsnrctl` 命令提供正确的监听器名称。

① 通过 `ADMIN_RESTRICTIONS_listener=on` 设置，监听器拒绝 SET 命令，它可能来自远程系统的入侵者。该设置仅允许通过本地系统的 `lsnrctl reload` 来修改设置。

Part 10

第十部分

实时应用集群

本 部 分 内 容

- 第 33 章 会话断开连接、负载均衡与 TAF
- 第 34 章 不重装就移除 RAC 选项

会话断开连接、负载 均衡与 TAF

没有任何手册说明 ALTER SYSTEM DISCONNECT SESSION 和 TAF（透明应用故障转移）之间的联系，包括最近发布的 Oracle11g 第一版。同样，也没有任何文档说明数据库实例的事务关闭和命令 SHUTDOWN TRANSACTIONAL 之间的联系。这里的数据库实例的事务关闭指的是关闭操作，如果有用户事务仍在处理，那么这个操作会推迟到所有用户完成工作提交或工作撤销才会执行。第三个没有归档的联系存在于会话断开连接和 TAF 之间，这里的会话断开连接由 PL/SQL 包 DBMS\_SERVICE 实现，适用于 Oracle10g 和 Oracle11g。

SQL 语句 ALTER SYSTEM DISCONNECT SESSION、SQL\*Plus 命令 SHUTDOWN TRANSACTIONAL 和 PL/SQL 包 DBMS\_SERVICE 可以恰当地断开和重连接开启透明应用故障转移功能的数据库会话与 DBMS 实例的连接。这个特性在性能维护方面很有用。该特性也可以用在当 RAC 集群的一个或者多个结点陆续不可用时，各 DBMS 实例之间重新平衡负载的情况。

33.1 透明应用故障转移介绍

透明应用故障转移是构建于 Oracle 调用接口（OCI）的自动数据库会话重建特性。这个特性主要是为了 RAC 环境下遇到集群结点故障而重建数据库会话，但是这个功能是完全独立于 RAC 的，可以应用到像 Data Guard 环境这样的单实例环境下。TAF 与 JDBC Thin 驱动不兼容，因为该驱动不是构建在 OCI 之上的。以下是配置了透明应用故障转移的 Net 服务名称实例：

```
taftest.oradbpro.com =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = dbserver1)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = dbserver2)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = TEN.oradbpro.com)
      (FAILOVER_MODE =
        (TYPE = select)
        (METHOD = basic)
```



```

    (RETRIES = 36)
    (DELAY = 5)
  )
)
)

```

前面定义的 Net 服务名定义指导 Oracle Net 做如下工作。

- 开启会话恢复功能（或者重新连接到一个存活的实例）。只要定义里面存在 `FAILOVER_MODE` 这个词，就不需要在 `DESCRIPTION` 小节里面添加 `(FAILOVER_ON)` 来显式地要求重连接。
- 试图自动并透明地重新运行数据库连接中断时正在进行的 `SELECT` 语句。
- 在尝试重新连接前等待 5 秒钟（`DELAY`）。
- 最多尝试 36 次重新建立连接，也就是必须在 3 分钟之内重建一个连接。当重连接时长（`DELAY` 乘以 `RETRIES`）耗尽，如果连接中断的原因是结点或实例故障，Oracle Net 就会向客户端发送错误信息 `ORA-03113: end-of-file on communication channel`。如果会话断开并且重连接时长耗尽，Oracle Net 就会向客户发送错误信息 `ORA-00028: your session has been killed`。如果在出现这些错误其中的一个之后，会话尝试运行 `SQL` 语句，它会引发错误 `ORA-03114: not connected to ORACLE`。在这种情况下，Oracle Net 会放弃使用 TAF，而开启一个新的数据库会话。

在 RAC 环境下，人们常常在 `DESCRIPTION` 小节里面添加指引（`LOAD_BALANCE = ON`）。其实在未设置的情况下，会话也分布在可用的 RAC 实例当中。如果想更深入了解与 TAF 和负载均衡相关的本地命名参数（`tnsnames.ora`）的相关信息，请参阅 *Oracle Database Net Services Reference* 手册。

33.2 改变系统断开会话设置

当发生结点故障和实例故障时，TAF 会启动。实例故障情形可以用 `SHUTDOWN ABORT` 来模拟。文档中没有提到的是当数据库会话被显式中断时，TAF 也会启动。相关语法如下：

```
ALTER SYSTEM DISCONNECT SESSION 'sid, serial#' [POST_TRANSACTION] [IMMEDIATE];
```

这里的参数 `sid` 和 `serial #` 和 `V$SESSION` 里面相同名字的列相对应。关键字 `POST_TRANSACTION` 在客户机的下一个 `COMMIT` 或者 `ROLLBACK` 后要求断开连接。关键字 `IMMEDIATE` 要求无论是否存在开启的事务都要马上断开数据库会话。在这条语句中至少给出这两个关键字之一。在没有开启事务的情况下，`POST_TRANSACTION` 在断开连接的时间点上和 `IMMEDIATE` 拥有相同的效果。然而，在会话重新建立方面，两者的实现存在很大不同。当指定了 `IMMEDIATE` 时，单独指定或者和关键字 `POST_TRANSACTION` 一起被指定时，TAF 不会起作用。但是，当关键字 `POST_TRANSACTION` 独自指定时，TAF 就会起作用。顺便提一下，当使用 `ALTER SYSTEM KILL SESSION` 时，或者使用 `UNIX` 命令 `kill-TERM`<sup>①</sup> 终止客户服务器进程时，TAF 也不会干预。

① 命令 `kill-TERM` 和命令 `kill-9` 是相同的。`TERM` 是 9 号信号的缩略名。所有信号的列表包含在 C 语言的头文件 `/usr/include/sys/signal.h` 中。

33.2.1 SELECT 故障转移

正如前面所述的，设置 TAF 只需要一个恰当配置的 Net 服务名称（例如在 `tnsnames.ora` 配置文件里面）和一个构建在 OCI 之上的数据库客户端。为了避免使用枯燥的 SQL\*Plus 来描述 TAF，我选择使用构建在 OCI 之上的 Perl DBI（见第 22 章）。下文描述的屏幕输出来自 Perl DBI 程序 `dbb.pl`<sup>①</sup>，这个程序可以执行任意的 SQL 语句和 PL/SQL 模块。我在被 SQL\*Plus 的不可读输出烦恼了数年，及在使用 `COLUMN name FORMAT` 命令梳理混淆的查询结果上花费了大量时间之后，写了该程序。

程序 `dbb.pl` 从标准输入读取信息，直到在一行的开头遇到一个斜线（/）为止。此时，这个程序为读入的语句做执行准备之后执行这些语句。如果该程序通过检验 DBI 句柄属性 `NUM_OF_FIELDS` 大于零探测到 SELECT 语句，它会读取这些行，并以自动调整的列宽显示出来。列宽会被自动调整到适当的宽度，这个宽度恰好适应 8 列中较宽的列头或较大的列值，从而使查询结果更可读。这个特性使得该程序和没有该特性的 SQL\*Plus 相比节约了大量的时间。眼下，这是程序 `dbb.pl` 的最佳表现。

在下面的测试中，我稍微改进了由 `dbb.pl` 调用的一个例程，使它以每读一行就停顿的方式迭代地读取所有行。这种方式保证了在程序读取行的循环过程中，可以相应中断请求。为了避免以块读取（也被称为数组读取方式，指的是在每次读取调用过程中，返回多于一行的数据）带来的缓存问题，必须保证表里面的行数大于读取的数组的长度。如果忽略这个问题，将会导致所有的行都被检索到客户端的缓存中。客户仅仅需要从客户端缓存中读取信息，而不需要和 DBMS 实例进行交互，从而使 SELECT 语句根本不需要重新启动。在使用 SQL\*Plus 进行 TAF SELECT 故障转移测试时，我推荐使用下面的设置来避免缓冲问题：

```
SQL> SET ARRAYSIZE 1
SQL> SET PAGESIZE 1
SQL> SET PAUSE "Hit enter to continue ..."
```

SQL\*Plus 将会做如下操作：

- 解析执行这些语句；
- 一次读取一行（在 SQL 的跟踪文件中，可以看到 `FETCH ... r=1`，参见第 24 章）；
- 推迟第一行数据的显示，直到用户点击回车；
- 显示每一行之后等待用户点击回车显示下一行。

这使测试人员可以在读取循环的过程中中断读取操作。没有这些设置，你需要从一个大表中进行选择给中断 SELECT 语句充足的时间。

回到 `dbb.pl` 的描述上，我要指出一些关于会话重建的副作用，这些副作用经常被忽视，也没有在文档中明显地提及。在 *Oracle Database Net Services Administrators Guide 10g Release 2 13-15* 页提到：

服务器端的程序变量，例如 PL/SQL 包语句，在发生故障的时候会丢失；TAF 也不能恢复这些变量的设置。可通过故障转移回调来初始化这些变量。

① Perl 程序 `dbb.pl`（数据库浏览器）包含在第 22 章的源代码库中。

正如下面所描述的，有很多是 TAF 无法恢复的：

- SET ROLE 语句的功能（或者 DBMS\_SESSION.SET\_ROLE），使重新建立的会话的优先级小于原来的会话；
- 安全应用角色功能；
- ALTER SESSION 语句的功能，例如开启 SQL 跟踪或修正 NLS 设置；
- 客户标识（V\$SESSION.CLIENT\_IDENTIFIER）；
- 程序实行的模块和行为（见第 24 章）。

事实上，TAF 除了可以恢复会话本身和 SELECT 之前的游标位置之外，不能恢复其他任何事情。并且记住 SELECT 不一定能够正常执行<sup>①</sup>。ORACLE 医生规定使用回调函数作为这种情况的补救措施。回调函数是客户通过调用 OCI 函数（如果用 Java 编程，这里应该是 JDBC 方法）注册的一个子例程。然后，OCI 认为在某些事件发生之后，执行客户的回调函数这一任务。TAF 回调可以为下列事件注册：

- 会话故障转移的开端；
- 未成功的故障转移尝试，客户可能指定 OCI 继续进行连接尝试；
- 会话重新建立的成功完成；
- 不成功的故障转移，没有尝试的可能。

相关细节已在以下两本书的文档中进行了说明，*Oracle Call Interface Programmer's Guide 10g Release 2* 和 *Oracle Database JDBC Developer's Guide and Reference 10g Release 2*。对于那些没有为 OCI 回调函数（包括 Perl DBI 模块）提供接口的编程语言，可以通过核对某些错误信息，如 ORA-25408: can not safely replay call 来检测会话故障转移。

下面是一个关于会话重建成功和重新启动的 SELECT 语句的示例。为了指明这些语句是用 dbb.pl 运行的，而不是 SQL \*Plus；提示符使用 DBB>。Perl 程序 dbb.pl 的启动方式和 SQL \*Plus 相同，都是通过传递一个连接字符串来实现的。

```
$ dbb.pl app_user/secret@taftest.oradbpro.com
```

接下来，设置了模块、行为和客户端标识符。

```
DBB> begin
      dbms_application_info.set_module('taf_mod', 'taf_act');
      dbms_session.set_identifier('taf_ident');
end;
/
1 Row(s) Processed.
```

由于 SELECT\_CATALOG\_ROLE 不是 APP\_USER 的默认角色，所以必须在能够访问 V\$SESSION 之前，通过 SET ROLE 语句来启动 SELECT。另外，改变了 NLS 日期格式。

```
DBB> SET ROLE select_catalog_role
/
```

① SELECT 故障转移可能会报错 “ORA-25401: can not continue fetches”。

```

0 Row(s) Processed.
DBB> ALTER SESSION SET nls_date_format='dd. Mon yy hh24:mi'
/
0 Row(s) Processed.
DBB> SELECT sid, serial#, audsid, logon_time, client_identifier, module, action
FROM v$session
WHERE username='APP_USER'
/
Row 1 fetched. Hit enter to continue fetching ...
SID SERIAL# AUDSID LOGON_TIME      CLIENT_IDENTIFIER MODULE ACTION
-----
116 23054 110007 05. Aug 07 14:40 taf_ident      taf_mod taf_act
1 Row(s) processed.

```

注意分配给客户的是会话 116，会话序列号是 23 054，审计会话标识符<sup>①</sup>是 110 007。审计标识符是在会话建立的时候通过从序列 SYS. AUDSES\$选择 NEXTVAL 形成的。当对会话进行审计的时候，审计标识符在数据库的生命周期内唯一标识一个会话，这个标志符保存在 DBA\_AUDIT\_TRAIL.SESSIONID 中。会话的日期格式包括月名称。客户标识符、模块、行为要传达给 DBMS。在 V\$SESSION 中，和查询故障转移相关的列保证了对于这个会话 TAF 功能是开启的。

```

DBB> SELECT failover_type, failover_method, failed_over
FROM v$session
WHERE username='APP_USER'
/
Row 1 fetched. Hit enter to continue fetching ...
FAILOVER_TYPE FAILOVER_METHOD FAILED_OVER
-----
SELECT      BASIC      NO
1 Row(s) processed.

```

接下来，APP\_USER 进入一个读取循环，从表 EMPLOYEES 中提取行数据。注意程序 dbb.pl 在每次读取调用之后的停顿，并不显示任何数据。

```

DBB> SELECT employee_id, first_name, last_name, email FROM hr.employees
/
Row 1 fetched. Hit enter to continue fetching ...
Row 2 fetched. Hit enter to continue fetching ...

```

在和程序 dbb.pl 运行窗口不同的一个窗口中，以 DBA 身份断开这个会话。

```

SQL> ALTER SYSTEM DISCONNECT SESSION '116,23054' POST_TRANSACTION;
System altered.

```

回到 dbb.pl 程序运行的窗口中，持续点击回车直到所有的行都显示出来。

```

Row 3 fetched. Hit enter to continue fetching ...
...
Row 107 fetched. Hit enter to continue fetching ...
EMPLOYEE_ID FIRST_NAME LAST_NAME EMAIL
-----

```

① 审计会话标识符 V\$SESSION.AUDSID 也可以通过如下语句检索：SELECT userenv(sessionid) FROM dual。

```

198 Donald      OConnell    DOCONNEL
...
197 Kevin      Feeney      KFEENEY

```

107 Row(s) processed.

SELECT 语句在没有任何中断迹象的情况下完成。现在是时候看看列 V\$SESSION.FAILED\_OVER 的值。

```

DBB> SELECT failover_type, failover_method, failed_over
FROM v$session
WHERE username='APP_USER'
/
error code: 942, error message: ORA-00942: table or view does not exist (error possi
bly near <*> indicator at char 56 in 'SELECT failover_type, failover_method, failed_
over FROM <*>v$session WHERE username='APP_USER'
')
```

上面所运行的从表 V\$SESSION 里选择的 SELECT 失败了。这也充分表明了会话的重建并没有恢复这个会话的所有特性。我们再次启用 SELECT\_CATALOG\_ROLE。

```

DBB> SET ROLE select_catalog_role
/
0 Row(s) Processed.
DBB> SELECT failover_type, failover_method, failed_over FROM v$session WHERE
username='APP_USER'
/

Row 1 fetched. Hit enter to continue fetching ...
FAILOVER_TYPE FAILOVER_METHOD FAILED_OVER
-----
SELECT        BASIC          YES
1 Row(s) processed.

```

V\$SESSION.FAILED\_OVER 的值前面是 NO，现在是 YES。这说明 TAF 成功了。那么原来数据库会话的其他特性呢？它们都丢失了。数据格式、客户标识符、模块和行为现在都是默认值。

```

DBB> SELECT sid, serial#, audsid, logon_time, client_identifier,
module, action
FROM v$session
WHERE username='APP_USER'
/

Row 1 fetched. Hit enter to continue fetching ...
SID SERIAL# AUDSID LOGON_TIME          CLIENT_IDENTIFIER MODULE  ACTION
-----
133  15197 110008 05.08.2007 14:49:23          perl.exe
1 Row(s) processed.

```

新会话的审计标识符是 110 008。Perl DBI 自动在 DBMS 中注册模块名为 perl.exe。

33.2.2 在事务末的故障转移

既然我们说到这里，就验证一下 DISCONNECT SESSION POST\_TRANSACTION 允许正在

进行的事务处理完成，然后通过 TAF 启动会话重建功能。这种情况的一个测试例子如下：

- (1) 启动一个删除一行的事务；
- (2) 运行另外一个 DELETE 语句，这个语句由于行被另外一个会话锁定而处于 TX 队列；
- (3) 在等待解锁的过程中，为这个事务标记断开；
- (4) 成功完成事务并重新建立连接。

这个情景的第一步是开启一个 DELETE 事务。

```
DBB> DELETE FROM hr.employees WHERE employee_id=190
/
1 Row(s) Processed.
```

以 DBA 身份，使用 SQL \* Plus（或者 dbb.pl，如果你喜欢它自动调整列大小的特性）检验 APP\_USER 存在一个开启的事务并在 HR.EMPLOYEES<sup>①</sup>中用 EMPLOYEE\_ID = 180 锁定行。

```
SQL> SELECT s.sid, s.serial#, s.event, t.start_time, t.status
FROM v$transaction t, v$session s
WHERE s.taddr=t.addr
AND s.sid=139;
      SID      SERIAL#  START_TIME          STATUS
-----
      139         74 07/30/07 15:34:25      ACTIVE
SQL> SELECT employee_id FROM hr.employees WHERE employee_id=180 FOR UPDATE NOWAIT;
EMPLOYEE_ID
-----
      180
```

作为 APP\_USER，使用 EMPLOYEE\_ID = 180 删除表 EMPLOYEES 中的行。这个会话必须等待 DBA 释放锁。

```
DBB> DELETE FROM hr.employees WHERE employee_id=180
/
```

以 DBA 身份，为 APP\_USER 的会话标识断开连接。

```
SQL> ALTER SYSTEM DISCONNECT SESSION '139,74' POST_TRANSACTION;
System altered.
```

以 DBA 身份，验证事务仍然处于活跃状态，此时 APP\_USER 仍然等待行锁。然后 COMMIT，这样就释放了在 EMPLOYEES 上的锁。

```
SQL> SELECT s.sid, s.serial#, s.event, t.start_time, t.status
FROM v$transaction t, v$session s
WHERE s.taddr=t.addr
AND s.sid=139;
      SID SERIAL# EVENT                                START_TIME          STATUS
-----
      139      74 enq: TX - row lock contention 07/30/07 15:34:25 ACTIVE
SQL> COMMIT;
Commit complete.
```

① 一个包含样例模式 HR 的导出转储文件包括在源代码库中。

在另外一个窗口，dbb.pl 显示了所处理的行数，这个行数我们用 COMMIT 语句来对应。

```
1 Row(s) Processed.
DBB> COMMIT
/
0 Row(s) Processed.
```

在这个 COMMIT 之后，DBMS 断开了 APP\_USER 的会话。由 TAF 重建的连接开启了一个新会话。

```
DBB> SET ROLE SELECT_CATALOG_ROLE
/
0 Row(s) Processed.
DBB> SELECT sid, serial#, logon_time, client_identifier, module, action, failed_over
FROM v$session
WHERE username='APP_USER'
/
Row 1 fetched. Hit enter to continue fetching ...
SID SERIAL# LOGON_TIME CLIENT_IDENTIFIER MODULE ACTION FAILED_OVER
-----
139 76 30-JUL-07 perl.exe YES
1 Row(s) processed.
```

前面的输出表明会话故障转移已启用 (FAILED\_OVER = YES)。问题是事务是否成功完成。

```
DBB> DELETE FROM hr.employees WHERE employee_id IN (180, 190)
/
0 Row(s) Processed.
```

由于没有出现任何错误信息，可见事务成功完成。重新运行同样的 DELETE 语句没有找到任何匹配的行，可见这些行被成功删除了。

在实例层次上，实现完全相同功能的是 SHUTDOWN TRANSACTIONAL。因此，在一个维护窗口开始运行时，所有的与某个实例相连接的数据库会话会转移到提供客户端请求服务的实例上。SHUTDOWN TRANSACTIONAL 的可选关键字 LOCAL 适用于分布式数据库环境，而不是 RAC 环境。当 LOCAL 关键字指定后，实例只需等待本地事务完成，而忽略分布式事务。

33.3 会话中断和 DBMS\_SERVICE

Oracle10g 和 Oracle11g 包含了包 DBMS\_SERVICE，提供管理实例服务和 TAF 的功能。初次使用时，这个包允许在服务器端配置 TAF，而不允许在客户端配置 tnsnames.ora 文件。DBMS\_SERVICE 是目前为止最精密的服务方法。在一个 RAC 集群中，当使用 DBCA 创建集群服务的时候在后台调用 DBMS\_SERVICE。但是，DBMS\_SERVICE 也可以在 RAC 和单实例环境下直接调用。在 RAC 环境下，由于集群数据库服务建立了 Oracle Clusterware 和 DBMS\_SERVICE 之间的整合，所以集群数据库服务应该使用 DBCA 进行配置。使用 DBMS\_SERVICE 建立起来的服务不会在实例启动的时候自动启动。在 DBCA 建立了适当的集群资源的情况下，Oracle Clusterware 执行这个任务。

33.3.1 使用 DBMS\_SERVICE 创建服务

创建一个服务至少需要提供一个服务名称和一个网络名称。服务名称是一个标识符，存储在数据词典当中。网络名称是实例服务名称（见本书引言中的“实例服务名称与网络服务名称”），这个名称注册监听器并且会在客户端的文件 `tnsnames.ora` 里，使用 `SERVICE_NAME` 引用来描述网络服务名称。

`DBMS_SERVICE.CREATE_SERVICE` 将服务插入到字典表 `SERVICE$` 中，该字典表位于视图 `DBA_SERVICES` 之下。程序 `DBMS_SERVICE.START_SERVICE` 将网络名称添加到初始化参数 `SERVICE_NAMES` 中，使该服务能够在监听器中注册。在为监听器注册服务时，初始化参数 `DB_DOMAIN` 的值被添加到网络名后面，如果 `DB_DOMAIN` 的值已经存在则不再添加。使用参数 `NETWORK_NAME` 指定的实例服务名称不区分大小写。服务名称和网络名称必须是唯一的。否则会提示错误信息 `ORA-44303: service name exists` 或 `ORA-44314: network name already exists`。下面的实例代码使用前面例子中使用的 TAF 设置创建和启动了一个服务：

```
SQL> BEGIN
    dbms_service.create_service(
        service_name=>'TAF_INST_SVC',
        network_name=>'taf_inst_svc_net_name',
        failover_method=>dbms_service.failover_method_basic,
        failover_type=>dbms_service.failover_type_select,
        failover_retries=>36,
        failover_delay=>12
    );
    dbms_service.start_service('TAF_INST_SVC', DBMS_SERVICE.ALL_INSTANCES);
END;
/
PL/SQL procedure successfully completed.
```

现在这个新服务在 `DBA_SERVICES` 中存在了。

```
SQL> SELECT name, network_name, failover_method method, failover_type type,
failover_retries retries, failover_delay delay
FROM sys.dba_services;
```

| NAME | NETWORK_NAME | METHOD | TYPE | RETRIES | DELAY |
|------------------|-----------------------|--------|--------|---------|-------|
| ----- | | | | | |
| SYS\$BACKGROUND | | | | | |
| SYS\$USERS | | | | | |
| TEN.oradbpro.com | TEN.oradbpro.com | | | | |
| TAF_INST_SVC | taf_inst_svc_net_name | BASIC | SELECT | 36 | 12 |

由于服务已经启动，新服务同样在 `GV$ACTIVE_SERVICES` 里面作为活跃服务注册。下面视图列出了对所有实例都活跃的服务：

```
SQL> SELECT inst_id, name, network_name, blocked
FROM gv$active_services
WHERE name NOT LIKE 'SYS$%';
```

| INST_ID | NAME | NETWORK_NAME | BLOCKED |
|---------|------|--------------|---------|
| ----- | | | |


```

1 TAF_INST_SVC      taf_inst_svc_net_name NO
1 TEN.oradbpro.com TEN.oradbpro.com      NO
2 TAF_INST_SVC      taf_inst_svc_net_name NO
2 TEN.oradbpro.com TEN.oradbpro.com      NO

```

由于网络名称缺少一个域后缀，所以将数据库域名 (DB\_DOMAIN) oradbpro.com 追加到网络名称后面。网络名称也被添加到参数 SERVICE\_NAMES 的实例服务名称列表中。

```

SQL> SELECT name, value FROM v$parameter
WHERE name in ('db_domain', 'service_names');
NAME          VALUE
-----
db_domain      oradbpro.com
service_names  taf_inst_svc_net_name

```

参数 SERVICE\_NAMES 的每个实例服务名称集合在一个或者多个监听器中注册。注意在一个 RAC 集群中，你需要设置参数 REMOTE\_LISTENER 的值为一个引用了集群中所有远程结点的网络服务的名称，从而使实例能够在远程结点上注册实例服务名称。为了避免出现本地监听器没有运行在默认端口 1521 的情况，你同时必须设置参数 LOCAL\_LISTENER。由于使用常数 DBMS\_SERVICE.ALL\_TNSTATANCES 作为程序 DBMS\_SERVICE.START\_SERVICE 的第二个参数，新实例服务的名称被两个实例注册到了监听器中，下面所示的命令 lsnrctl 的输出证明了这一点。

```

$ lsnrctl services listener_dbserver1
...
Services Summary...
...
Service "taf_inst_svc_net_name.oradbpro.com" has 2 instance(s).
  Instance "TEN1", status READY, has 1 handler(s) for this service...
    Handler(s):
      "DEDICATED" established:0 refused:0 state:ready
      LOCAL SERVER
  Instance "TEN2", status READY, has 1 handler(s) for this service...
    Handler(s):
      "DEDICATED" established:0 refused:0 state:ready
      REMOTE SERVER

```

用这种方式进行配置，就没有必要在客户端的配置文件 tnsnames.ora 里面开启 TAF 了。TAF 已经为实例服务名称 taf\_inst\_svc\_net\_name.oradbpro.com 开启，就像下面显示的网络服务名称 taf\_net\_svc.oradbpro.com 没有包含 FAILOVER\_MODE 一节。服务端的 TAF 设置覆盖了客户端的 TAF 设置。注意传递给 DBMS\_SERVICE (后缀使用数据库域名) 的网络名称用于设置 tnsnames.ora 中 SERVICE\_NAME 的值。

```

taf_net_svc.oradbpro.com =
(DESCRIPTION =
  (ADDRESS = (PROTOCOL= TCP)(Host=dbserver1)(Port= 1521))
  (CONNECT_DATA = (SERVICE_NAME = taf_inst_svc_net_name.oradbpro.com))
)

```

33.3.2 DBMS\_SERVICE 和 TAF 下的会话中断

现在是时候测试 DBMS\_SERVICE 下的会话断开的情况了，为将来的性能维护和负载均衡做准备。以 APP\_USER 身份进行连接，使用前文提到的网络服务名运行一个 SELECT 语句。

```
$ sqlplus app_user/secret@taf_net_svc.oradbpro.com
```

```
SQL> SET PAUSE "Hit enter to continue ..."
```

```
SQL> SET PAUSE ON
```

```
SQL> SELECT * FROM audit_actions;
```

```
Hit enter to continue ...
```

```
ACTION NAME
```

```
-----
```

```
0 UNKNOWN
```

```
1 CREATE TABLE
```

```
...
```

作为 DBA，验证 TAF 设置生效。

```
SQL> SELECT inst_id, sid, serial#, audsid, logon_time, service_name,
```

```
failover_type, failover_method, failed_over
```

```
FROM gv$session
```

```
WHERE username='APP_USER';
```

```
INST_ID SID SERIAL# AUDSID LOGON_TIME SERVICE_NAME FAILOVER_TYPE
```

```
-----
```

```
2 143 4139 120036 05.08.2007 19:27:58 TAF_INST_SVC SELECT
```

```
FAILOVER_METHOD FAILED_OVER
```

```
-----
```

```
BASIC
```

```
NO
```

现在停止为客户提供服务的实例 2（在前面的 SELECT 输出结果中为 INST\_ID=2）上的服务。

实例 2 的名称为 TEN2。可以通过集群中任意的实例关闭某项服务。

```
SQL> EXEC dbms_service.stop_service('TAF_INST_SVC', 'TEN2')
```

```
PL/SQL procedure successfully completed.
```

该操作在 GV\$ACTIVE\_SERVICES 中，将服务从实例 2 中删除了：

```
SQL> SELECT name FROM gv$active_services WHERE inst_id=2;
```

```
NAME
```

```
-----
```

```
TEN.oradbpro.com
```

```
SYS$BACKGROUND
```

```
SYS$USERS
```

实例 TEN2 也从监听器的服务总结表中删除了。

```
$ lsnrctl services listener_dbserver1
```

```
...
```

```
Service "taf_inst_svc_net_name.oradbpro.com" has 1 instance(s).
```

```
Instance "TEN1", status READY, has 1 handler(s) for this service...
```

```
Handler(s):
```

```
"DEDICATED" established:0 refused:0 state:ready
```

```
LOCAL SERVER
```

```
The command completed successfully
```

DBMS\_SERVICE.DISCONNECT\_SESSION 影响本地实例使用某项服务的所有会话。这样，为了断开所有的会话，需要连接到为客户提供服务的实例。这个程序有一个文档中未提及的参数 DISCONNECT\_OPTION。这个参数的默认值是数值型常量 DBMS\_SERVICE.POST\_TRANSACTION。也可以设置为 DBMS\_SERVICE.IMMEDIATE 的值。这些常量和 SQL 语句 ALTER SYSTEM DISCONNECT SESSION 里支持的关键字 POST\_TRANSACTION 和 IMMEDIATE 有相同的意义。我们来断开所有通过服务 TAF\_INST\_SVC 建立起来的会话。

```
SQL> EXEC dbms_service.disconnect_session('TAF_INST_SVC')
```

要知道即使传递的服务名称不存在，DBMS\_SERVICE.DISCONNECT\_SESSION 也会成功完成。除了开启事务的会话外，其他的会话断开立即生效。

```
SQL> SELECT inst_id, sid, serial#, audsid, logon_time, service_name,
failover_type, failover_method, failed_over
FROM gv$session
WHERE username='APP_USER';
INST_ID SID SERIAL# AUDSID LOGON_TIME          SERVICE_NAME
-----
      2 143    4139 120036 05.08.2007 19:27:58 TAF_INST_SVC
FAILOVER_TYPE FAILOVER_METHOD FAILED_OVER
-----
SELECT          BASIC          NO
```

一旦 APP\_USER 试图使用 SELECT 语句来检索剩余的行，系统将会探测到断开状态并打开一个到实例的连接，该实例能提供所请求的服务。在 GV\$SESSION 中再次运行 SELECT 将得到如下信息：

```
INST_ID SID SERIAL# AUDSID LOGON_TIME          SERVICE_NAME
-----
      1 151    1219 130018 05.08.2007 19:31:35 TAF_INST_SVC
FAILOVER_TYPE FAILOVER_METHOD FAILED_OVER
-----
SELECT          BASIC          YES
```

APP\_USER 的会话被重连接到实例 1，从 INST\_ID = 1、不同的 AUDSID 和比较迟的 LOGON\_TIME 中可以看出。由于所有实例的序列缓存，AUDSID 的值在重连接之后，不是必须增长的。TAF 也为断开时有正在进行事务的会话建立重连接。对于这些会话，当它们提交任务或撤销任务之后才断开连接。和 ALTER SYSTEM DISCONNECT SESSION POST\_TRANSACTION 相同，DBMS\_SERVICE.DISCONNECT\_SESSION 也在所有正在进行的事务完成之后断开会话。事实上，DBMS\_SERVICE.DISCONNECT\_SESSION 不过是在 ALTER SYSTEM DISCONNECT SESSION POST\_TRANSACTION 之上的一层。这个未归档的事实可以从源文件 \$ORACLE\_HOME/rdbms/admin/dbmssrv.sql 中看出来。

DBMS\_SERVICE 的新特性与参数 SERVICE\_NAMES 相关的优良特性之间的整合完全不符合逻辑。语句 ALTER SYSTEM SET SERVICE\_NAMES 在 DBA\_SERVICES 中添加了一个服务，然而从 SERVICE\_NAMES 中删除这个服务时却没有从 DBA\_SERVICES 中删除。这个操作仅仅关闭了服务并将其从 V\$ACTIVE\_SERVICES 中删除。

为了删除服务，必须在所有实例中关闭该服务。由数据库名称和数据库域名后缀组成的默认

服务不能删除。由于没有报错，看起来好像可以用 DBMS\_SERVICE.STOP\_SERVICE 关闭默认服务，但是交叉测试表明该服务并没有关闭。这里的交叉测试包括核对 lsnrctl services listener\_name 得到的服务总结和 V\$ACTIVE\_SERVICES。这里引入一个新的错误信息如 default service cannot be stopped 会更合适。例如，我的 DBMS 实例设置了 db\_name = TEN 和 db\_domain = oradbpro.com，则在 DBA\_SERVICES 和 V\$ACTIVE\_SERVICES 里面自动生成一个实例服务名 TEN.oradbpro.com，这个服务既不能关闭也不能删除。

33.4 小结

由日常维护或手动负载均衡引起的数据库服务破坏可以通过适当配置 TAF 和会话中断得到大大改善。会话中断可能发生在三个层面：实例层、服务层和会话层。表 33-1 总结了各层次的命令。

表33-1 会话断开语句

| 层 次 | SQL 语句or PL/SQL 程序调用 | 客户端程序 |
|-----|---|-------------------------|
| 实例 | SHUTDOWN TRANSACTIONAL | SQL*Plus、Oracle11g JDBC |
| 服务 | DBMS_SERVICE.DISCONNECT_SESSION (SERVICE_NAME => 'name',
DISCONNECT_OPTION => option); <sup>①</sup> | Any |
| 会话 | ALTER SYSTEM DISCONNECT SESSION 'sid, serial#' POST
_TRANSACTION; | Any |

虽然在 Oracle9i 中已经在实例层和会话层实现了恰当的断开功能，但是服务层的断开功能到 Oracle10g 才第一次引入。在客户配置了并使用 TAF 的情况下，这些特性可以使数据库客户端隐藏实例启动。在 Oracle10g 及其之后的版本中，应该在服务器端使用 DBMS\_SERVICE<sup>®</sup>对 TAF 进行设置。

33.5 源代码库

表 33-2 列出了本章源代码文件和它们的功能。

表33-2 会话中断、负载均衡和TAF源代码库

| 文 件 名 | 功 能 |
|------------------|--------------------------------|
| hr.dmp | Oracle10g导出转储文件，包括样例模式HR的数据库对象 |
| tnsnames.ora.smp | 范例网络服务名称定义，用来在客户端配置TAF |

① 参数 DISCONNECT\_OPTION 在版本 10.2.0.1 中不可用。这个参数的默认值是 DBMS\_SERVICE.POST\_TRANSACTION。断开选项 DBMS\_SERVICE.IMMEDIATE 会立即断开会话连接。

只有同一节点上存在 Oracle 集群软件（从 Oracle10g 强制启动）或第三方集群软件（vendor cluster software）等 Oracle DBMS 实例时，Oracle 实时应用集群（RAC）才能运行。本章说明集群软件失效时引起的问题，这些问题反过来又会阻止 RAC 实例打开数据库。

Oracle 集群软件使用一个集群注册表（Cluster Registry）来保存配置信息。在网络失效时，它还可以将一些特殊信息写入表决磁盘（voting disk）来协调集群行为。在紧急情况下，比如所有表决磁盘失效或保存集群注册表备份的所有设备都失效时，Oracle Clusterware 就不能启动。在这种严重失效的情形下，需要将一个新的 SAN（Storage Area Network，存储区域网）连接到系统，在 SAN 磁盘阵列中创建逻辑单元，以及配置分区以使得逻辑单元对数据库服务器可见，上述操作往往需要耗费数小时甚至数天。尽管相对其早期版本已有巨大进步，但是 Oracle Clusterware 仍可能导致一些非必要的节点重启，从而造成计划外停机时间。这是导致 RAC 失效的另一个原因。本章解释怎样使用一个未在文档中说明的 make 命令来从 oracle 可执行文件中移除 RAC 选项。ASM（自动存储管理器）不能在非 Oracle 集群环境中运行，因此，为了使用单实例的 ASM，需要一个能将 RAC 的 Oracle Clusterware 环境转换为一个本地的 Oracle Clusterware 环境的程序。有了这样的程序，无需使用 OUI（Oracle Universal Installer，Oracle 通用安装器）重新安装任何软件，也无需使用 OPatch 修改 ORACLE\_HOME，一个运行 RAC 的系统就能够迅速地转变为一个运行单个 ORACLE 实例的系统，从而大大减少上述严重事故所导致的停机时间。

34.1 连接 ORACLE 软件

使用 OUI 在 UNIX 系统上安装 ORACLE 服务器软件时，许多程序，如实现数据库内核的 \$ORACLE\_HOME/bin/oracle 等，都将与系统中静态及共享库连接。OUI 调用 make 并将 makefile（make 程序的描述文件）\$ORACLE\_HOME/rdbms/lib/ ins\_rdbms.mk 作为参数传递给 make。其他的 makefile，如 \$ORACLE\_HOME/network/lib/ins\_net\_server.mk 被用于连接 Oracle 网络组件，而另一些则被用于连接 SQL\*Plus。

在安装补丁集合（使用 OUI）或临时补丁（使用 OPatch）时也有类似的步骤。大多数补丁通过用纠正了缺陷的新版本替换目标模块来修改 \$ORACLE\_HOME/lib 中的静态库。可执行程序 oracle 此时必须重新连接以便能从静态库中获得修改过的目标模块。对于那些知道如何使用 ar

和 make 命令的人来说,很容易就可以手动安装临时补丁。这在 OPatch 失效(不论何种原因)时是很有用的。比如,通过从静态库中提取目标模块并将其与装载了补丁的目标模块比较,新版的 OPatch(从版本 1.0.0.0.54 以及 10.2.0.x 开始)能够核实一个新的目标模块(扩展名为.o,如 dbsdsv.o)是否已被正确嵌入到静态库(扩展名为.a,如 libserver10.a)中。然而,因为 Solaris 10 在目标文件中使用换行符来填充对象文件(可以使用 `od -c filename` 来验证这一点),在 64 位的 Solaris 上的 Oracle 环境中之前合理的测试却失败了。OPatch 给出错误信息 `Archive failed: failed to update` 并终止了临时补丁的安装。设置环境变量 `OPATCH_DEBUG=TRUE`(相关文档见 *Oracle Universal Installer and OPatch User's Guide 10g Release 2*)后再进行安装,我们发现 `ar` 命令并没有失效,而是验证操作失败了。这一问题已经受到重视(详见 Metalink note 353150.1)。

替换目标模块的方法也被用于添加或移除 ORACLE 服务器可选项。这些可选项必须通过 Oracle 数据库管理系统的企业版来购置。目前,存在 10 个可选项,其中包括:

- ❑ 标签安全 (Label Security);
- ❑ 分区 (Partitioning);
- ❑ RAC (Real Application Clusters, 实时应用集群);
- ❑ 空间配置 (Spatial);
- ❑ 数据挖掘 (Data Mining)。

Oracle10g 的 OUI 有一个缺陷:即使相关 OUI 界面没有选择该项,“数据挖掘”可选项也会被安装。下面简要叙述的增加和移除可选项的方法,可以用于解决这个 bug。

在 SQL\*Plus 启动时,可以看到安装了哪些可选项。

```
$ sqlplus ndebes/secret
SQL*Plus: Release 10.2.0.3.0 - Production on Thu Jul 26 02:21:43 2007
Copyright (c) 1982, 2006, Oracle. All Rights Reserved.
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production
With the Partitioning, Real Application Clusters, Oracle Label Security and Data
Mining options
```

SQL\*Plus 从视图 V\$OPTION 中获得这些信息。

```
SQL> SELECT parameter, value FROM v$option
WHERE parameter IN ('Partitioning','Real Application Clusters',
'Oracle Label Security','Data Mining')
```

| PARAMETER | VALUE |
|---------------------------|-------|
| Partitioning | TRUE |
| Real Application Clusters | TRUE |
| Oracle Label Security | TRUE |
| Data Mining | TRUE |

当一个可选项通过连接被增加或移除时,相应地,V\$OPTION.VALUE 也变为 TRUE 或 FALSE。

34.2 案例研究

在本节,我们将模拟一个 RAC 环境下所有表决硬盘失效的情形来为移除 RAC 可选项创建一

个应用场景。你是否知道：一个使用连接到 RAC 的 oracle 可执行程序的 Oracle 实例，在本地节点的集群组服务（cluster group service，Oracle Clusterware 的一个组件）失效时，若没有 CLUSTER\_DATABASE 参数，是无法启动的。这对于 ASM 和 RDBMS 都同样适用（即 instance\_type={ASM|RDBMS}）。

34.2.1 模拟表决磁盘失效

表决磁盘是 Oracle Clusterware 用于在连接失效时确保数据库完整性的设备。假设你丢失了所有为 Oracle Clusterware 配置的表决磁盘，这将引起 Oracle Clusterware 退出。下面是一个模拟所有表决磁盘丢失的快速测试。在这个例子中，因为使用外部冗余（即存储子系统的冗余）而不是 Oracle Clusterware 的三倍镜像冗余，每个表决磁盘都只有它自己（即没有备份）。测试在 Red Hat Advanced Server 4 上进行。

Linux 没有合适的裸设备（raw device），因此 raw 命令必须将裸设备绑定到块设备（block device）<sup>①</sup>。一个表决磁盘的失效可以通过将一个表决磁盘设备绑定到一个未使用的块设备上模拟。在下面的测试系统中，/dev/md1 就是这样的一个块设备。打开它时并不抛出错误，但是返回 0 字节。模拟表决磁盘失效的具体步骤如下：

```
1 # wc -c /dev/md1
2 0 /dev/md1
3 # crsctl query css votedisk # ask Clusterware for configured voting disks
4 0.      0      /opt/oracle/votedisk
5 located 1 votedisk(s).
6 # ls -l /opt/oracle/votedisk # character special device with major nr. 162
7 crw-r--r--  1 oracle oinstall 162, 1 Jul 20 22:40 /opt/oracle/votedisk
8 # raw -q /opt/oracle/votedisk # which block device is it bound to?
9 /dev/raw/raw1: bound to major 8, minor 8
10 # ls -l /dev/sda8 # major device number 8 and minor 8 is sda8 (SCSI disk)
11 brw-rw----  1 root disk 8, 8 Jul 24 03:15 /dev/sda8
12 # raw /opt/oracle/votedisk /dev/md1 # rebind to the wrong device
13 /dev/raw/raw1: bound to major 9, minor 1
14 # crsctl start crs
15 Attempting to start CRS stack
16 The CRS stack will be started shortly
17 # crsctl check crs
18 Failure 1 contacting CSS daemon
19 Cannot communicate with CRS
20 Cannot communicate with EVM
```

正如 crsctl check crs 命令的输出（见第 18 行）所表明的，Oracle Clusterware 在没有表决磁盘的情况下是无法启动的。让我们按行来仔细分析各个命令。

- 第 1 行：命令 wc（词语计数）从 /dev/md1 中读取操作成功，但没有读到任何东西。这个设备文件将被用于模拟一个失效的表决磁盘。

<sup>①</sup> 因为对裸设备的直接操作已被废弃，Oracle10g R2 在 Linux 上的 ASM 实例也支持块设备。Red Hat 鼓励软件开发商在他们的应用中通过 O\_DIRECT 标志打开块设备而不是直接请求裸设备。

- ❑ 第 3 行：通过 `crsctl query css votedisk` 命令可以列出所配置的表决磁盘。
- ❑ 第 4 行：唯一被配置的表决磁盘是 `/opt/oracle/votedisk`。这不是设备本身，而是一个字符专用设备文件（character special device file，又称裸设备文件），该文件指向代表表决磁盘的块设备。
- ❑ 第 7 行：`/opt/oracle/votedisk` 的 major 编号和 minor 编号分别是 162 和 8。在 Linux 中，所有裸设备的 major 编号是 162。Major 编号区分系统中的各个设备类。Minor 编号区分一个设备类中的各个实例。在 Linux 中，裸设备实例的 minor 编号从 1 到 255。
- ❑ 第 8 行：`/opt/oracle/votedisk` 当前的绑定情况可以通过 `raw -q raw_device_file` 命令查询。
- ❑ 第 9 行：`raw` 命令并不关心裸设备的名字是否有意义（比如“rawn”有更多的含义）。裸设备 `/opt/oracle/votedisk` 被绑定到 major 编号为 8（SCSI 磁盘）和 minor 编号为 8 的块设备上。
- ❑ 第 11 行：major 编号和 minor 编号均为 8 的块设备是 `/dev/sda8`，即它是系统中第一块 SCSI 磁盘的第 8 个分区。
- ❑ 第 12 行：重置裸设备 `/opt/oracle/votedisk` 的绑定到 `/dev/md1` 上。这一操作成功了，因为 Oracle Clusterware 被关闭，同时导致字符专用设备文件不可读。
- ❑ 第 14 行：试图启动 Oracle Clusterware。
- ❑ 第 17 行：查看 Oracle Clusterware 进程的状态。
- ❑ 第 18 行：因为表决磁盘失效，Oracle Clusterware 无法启动。

奇怪的是，没有错误信息写入任何日志文件（测试在 10.2.0.3.0 上进行）。通常，在 Linux 上，CRS（Cluster Ready Services，集群就绪服务）用 `/bin/logger` 将日志信息写入到 `/var/log/messages` 中<sup>①</sup>。当不能通过 `$ORA_CRS_HOME/log/nodename/alertnodename.log`（这里的 `nodename` 是系统的主机名）或者在 `$ORA_CRS_HOME/log` 中的其他日志文件诊断问题时，你可以参考 CRS 所写的文件。但是，在上面的例子中，因为缺乏表决磁盘，CRS 无法运行。

```
Jul 26 20:54:44 dbserver1 logger: Cluster Ready Services waiting on dependencies.
Diagnostics in /tmp/crsctl.30644.
```

失败的原因见 `/tmp/crsctl.30644`，如下所示：

```
# more /tmp/crsctl.30644
Failure reading from offset 2560 in device votedisk
Failure 1 checking the CSS voting disk 'votedisk'.
Not able to read adequate number of voting disks
```

在 `/etc/init.d/init.cssd` 中有一个循环，该循环调用 `crsctl check boot` 直至返回退出码 0。在所有的表决磁盘都不可获得时，由于不会返回退出码 0，CRS 将陷入死循环。<sup>②</sup>

① `/bin/logger` 命令是一个 Shell 命令，是系统日志模块的接口。

② 当所有的 OCR 备份都无法获得时也会发生同样的事情。在我的测试中出现这种情形时抛出的错误是 `OCR initialization failed accessing OCR device: PROC-26: Error while accessing the physical storage Operating System error [Success] [0]`。


```
# crsctl check boot
Failure reading from offset 2560 in device votedisk
Failure 1 checking the CSS voting disk 'votedisk'.
Not able to read adequate number of voting disks
# echo $?
6
```

另一种来模拟表决磁盘或者其他类型的文件（如 Oracle 集群注册表、ASM 磁盘或者数据文件）失效的方法是用二进制的 0 来重写文件中起始部分的一些块。这可以用下列 dd (device dump, 设备转储) 命令实现：

```
dd if=/dev/zero bs=8192 count=2 of=voting_disk_file
```

显然，这是一种带有破坏性的测试方法。因此，你需要在进行这个尝试之前用 dd 命令备份你的表决磁盘。不过，这种方法有一个优点：尽管 raw 命令不能重新映射一个正在使用的块，这种测试仍可以在 Oracle Clusterware 运行时进行<sup>①</sup>。这种测试方法中产生的错误信息都被正确地记录在文件夹 \$ORA\_CRS\_HOME/log/nodename 下的日志文件中。

回到我前面的论断：一个连接到 RAC 的 oracle 可执行程序，在 Oracle Clusterware 失效时是无法启动一个 DBMS 实例的。如果要使用 ASM，ASM 实例必须在 DBMS 实例之前启动。下面是一些证据（由 Oracle 安装文件的所有者运行 SQL\*Plus）：

```
$ env ORACLE_SID=+ASM1 sqlplus / as sysdba
SQL*Plus: Release 10.2.0.3.0 - Production on Thu Jul 26 02:35:22 2007
Copyright (c) 1982, 2006, Oracle. All Rights Reserved.
Connected to an idle instance.
SQL> STARTUP
ORA-29701: unable to connect to Cluster Manager
```

为了完备起见，让我们再来确认一下由 DBMS 实例给出的一条相似的错误信息：

```
$ env ORACLE_SID=TEN1 sqlplus / as sysdba
SQL> STARTUP NOMOUNT
ORA-29702: error occurred in Cluster Group Service operation
```

这表明当集群组服务（Cluster Group Services）无法获得时，即便是 STARTUP、NOMOUNT 命令都无法工作。

34.2.2 使用 make 工具移除 RAC 可选项

当没有使用 ASM 时，我们可以很容易地将 oracle 可执行程序从 RAC 转变为单一实例，即可以在集群运行时获得单一实例。要实现这个操作，需要由 Oracle 安装环境的所有者（一般是 oracle）来调用 make 命令，将未在相关文档中说明的目标配置设置为 rac\_off。

```
$ cd $ORACLE_HOME/rdbms/lib
$ make -f ins_rdbms.mk rac_off ioracle
...
/usr/bin/ar d /opt/oracle/product/db10.2/rdbms/lib/libknlopt.a kcsn.o
```

<sup>①</sup> 抛出的错误是 Error setting raw device (Device or resource busy)。

```

/usr/bin/ar cr /opt/oracle/product/db10.2/rdbms/lib/libknlpt.a /opt/oracle/product/
db10.2/rdbms/lib/ksnks.o
- Linking Oracle
gcc -o /opt/oracle/product/db10.2/rdbms/lib/oracle ... -lknlpt ...
...
mv /opt/oracle/product/db10.2/rdbms/lib/oracle /opt/oracle/product/db10.2/bin/oracle
chmod 6751 /opt/oracle/product/db10.2/bin/oracle

```

这里最有趣是 ar 命令。ar 命令先将目标文件 kcsn.o 从 libknlpt.a (kernel options library, 内核选项库) 中移除, 然后将目标文件 ksnks.o 加入到 libknlpt.a 中。这个过程中将移除 RAC 选项。再次启动将产生如下信息:

```

$ env ORACLE_SID=TEN1 sqlplus / as sysdba
SQL> STARTUP
ORA-00439: feature not enabled: Real Application Clusters

```

错误 ORA-00349 出现的原因是: RAC 环境中的 DBMS 实例只能在参数 CLUSTER\_DATABASE=TRUE 时运行。而且, 当连接到 RAC 时, 这个参数的值只能取 TRUE。因此, 我们需要将 CLUSTER\_DATABASE 设置为 FALSE。如果你正在使用 SPFILE, 那么你将无法在不开启一个实例的情况下修改这个参数。然而, 此时开启一个实例却是不可能的。因此, 你需要移除 SPFILE 中的二进制内容, 从而得到一个纯文本文件, 同时需要移除参数 CLUSTER\_DATABASE (这样可使得参数的值恢复为默认配置 FALSE)。再次尝试启动的结果如下:

```

$ cd $ORACLE_HOME/dbs
$ strings spfileTEN1.ora | grep -v cluster > pfileTEN1.ora
$ env ORACLE_SID=TEN1 sqlplus / as sysdba
SQL> STARTUP NOMOUNT PFILE=pfileTEN1.ora
ORACLE instance started.
Total System Global Area  314572800 bytes
Fixed Size                  1261564 bytes
Variable Size              201326596 bytes
Database Buffers           104857600 bytes
Redo Buffers                7127040 bytes
SELECT value FROM v$option WHERE parameter='Real Application Clusters';
VALUE
-----
FALSE

```

RDBMS 实例终于成功启动。但是成功的前提是你没有使用 ASM 存储。可是如果使用了 ASM 又会怎样呢? 让我们在如下情形下启动一个 ASM 实例: CRS 没有开启, RAC 可选项被移除, 参数 CLUSTER\_DATABASE 的值为 FALSE。

```

$ strings spfile+ASM1.ora|grep -v cluster > pfile+ASM1.ora
$ env ORACLE_SID=+ASM1 sqlplus / as sysdba
SQL> STARTUP NOMOUNT PFILE=pfile+ASM1.ora
ORA-29701: unable to connect to Cluster Manager

```

这时我们无法成功。因为 ASM 实例与 RDBMS 实例之间的通信依赖于 Oracle Clusterware 的 OCSSD (Oracle Cluster Synchronization Service Daemon, Oracle 集群同步服务守护进程)。

34.2.3 转换 CRS 环境为本地环境

ASM 不需要完整的 CRS 栈。只需精简过的 CRS，且不需要运行除了 OCSSD 之外的守护进程，就可以运行 ASM。OCSSD 实现了集群同步服务，它通过第三方集群软件监控集群节点的运行状况；当然，如果第三方软件不存在时，它就会自己监控。它向 ORACLE 实例提供关于集群成员的消息；向 CRSD 和 EVMD 提供关于集群运行状况的信息。后面的两个守护进程（即 CRSD 和 EVMD）对于 ASM 实例来说不是必须的。

本地 CRS 环境不需要任何表决磁盘。因为只有一个节点可以进行表决。系统文件可作为 OCR（Oracle Cluster Registry，Oracle 集群注册表）使用，因此不需要任何裸设备。转换过程包括：备份 OCR 以及其他配置文件，运行 \$ORA\_CRS\_HOME/install 和 \$ORA\_CRS\_HOME/bin 中的 shell 脚本文件。

/etc/inittab 中有 3 个条目标记了用于 RAC 的 CRS 环境，如下所示：

```
$ grep '^h' /etc/inittab
h1:35:respawn:/etc/init.d/init.evmd run >/dev/null 2>&1 </dev/null
h2:35:respawn:/etc/init.d/init.cssd fatal >/dev/null 2>&1 </dev/null
h3:35:respawn:/etc/init.d/init.crsd run >/dev/null 2>&1 </dev/null
```

/etc/ocr.loc<sup>①</sup>中的参数 local\_only 的值是 FALSE。

```
$ cat /etc/oracle/ocr.loc
ocrconfig_loc=/opt/oracle/ocr
local_only=FALSE
```

一个本地 CRS 环境只在/etc/inittab 中有一个条目，参数 local\_only 被设定为 TRUE。另一个不同处是在目录结构/etc/oracle/下存在一个 scls\_scr。我们首先要备份 CRS 当前的配置。这个过程需要：用 dd 命令来备份集群注册表，备份配置文件和 shell 脚本，保存/etc/inittab 中有关 CRS 的条目。这些操作需要在 root 用户下进行：

```
# dd if=/opt/oracle/ocr bs=1048576 of=ocr.bin
258+1 records in
258+1 records out
# ocrdump ocr.txt
tar cvfP crs_local_only_false.tar /etc/oracle /etc/init.d/init.crs \
/etc/init.d/init.crsd /etc/init.d/init.cssd /etc/init.d/init.evmd \
/etc/rc0.d/K96init.crs /etc/rc1.d/K96init.crs /etc/rc2.d/K96init.crs \
/etc/rc3.d/S96init.crs /etc/rc4.d/K96init.crs /etc/rc5.d/S96init.crs \
/etc/rc6.d/K96init.crs /etc/inittab
# grep '^h[1-3]' /etc/inittab > inittab.crs
```

现在，当前的配置都已经保存下来，我们可以放心地修改了。脚本 rootdelete.sh 从/etc/inittab 中移除 CRS 的条目并通知 init 相关的变动，移除/etc/init.d 以及/etc/rc[0-6].d，删除/etc/oracle/scls\_scr 但保留/etc/oracle/ocr.loc。如果 CRS 陷入前面提到过的死循环，就关闭由 init.cssd 用参数 startcheck 调用的脚本进程。

① 在某些平台上，ocr.loc 文件位于 /var/opt/oracle 下。

```
# ps -e -o pid,ppid,comm,args |fgrep init.|grep -v grep
4584    1 init.evmd      /bin/sh /etc/init.d/init.evmd run
4877    1 init.cssd      /bin/sh /etc/init.d/init.cssd fatal
4878    1 init.crsd      /bin/sh /etc/init.d/init.crsd run
8032  4584 init.cssd      /bin/sh /etc/init.d/init.cssd startcheck
8194  4878 init.cssd      /bin/sh /etc/init.d/init.cssd startcheck
8247  4877 init.cssd      /bin/sh /etc/init.d/init.cssd startcheck
# kill 8032 8194 8247
# $ORA_CRS_HOME/install/rootdelete.sh
Shutting down Oracle Cluster Ready Services (CRS):
Stopping resources. This could take several minutes.
Error while stopping resources. Possible cause: CRSD is down.
Shutdown has begun. The daemons should exit soon.
Checking to see if Oracle CRS stack is down...
Oracle CRS stack is not running.
Oracle CRS stack is down now.
Removing script for Oracle Cluster Ready services
Updating ocr file for downgrade
Cleaning up SCR settings in '/etc/oracle/scls_scr'
```

现在，适用于 RAC 的 CRS 配置已经被移除，我们可以使用在文档中说明的脚本 `localconfig` 来安装本地 CRS 配置，以便可以使用 ASM。

```
# $ORA_CRS_HOME/bin/localconfig add
Successfully accumulated necessary OCR keys.
Creating OCR keys for user 'root', privgrp 'root'..
Operation successful.
Configuration for local CSS has been initialized
Adding to inittab
Startup will be queued to init within 30 seconds.
Checking the status of new Oracle init process...
Expecting the CRS daemons to be up within 600 seconds.
CSS is active on these nodes.
    dbserver1
CSS is active on all nodes.
Oracle CSS service is installed and running under init(1M)
```

精简的 CRS 栈现在开始运行了。你可以用 `crsctl check css` 命令来检查它的状态。

```
$ crsctl check css
CSS appears healthy
$ crsctl check crs
CSS appears healthy
Cannot communicate with CRS
Cannot communicate with EVM
$ ps -ef|grep ocssd |grep -v grep
oracle  14459    1 0 01:42 ? 00:00:00 /opt/oracle/product/crs10.2/bin/ ocssd.bin
```

`crsctl check crs` 命令没能成功连接到 CRSD 和 EVMD，因为 CRSD 和 EVMD 已经被取消了。`ocr.loc` 中参数 `local_only` 的值现在是 TRUE，且在 `/etc/inittab` 中只有一条有关 OCSSD 的条目。

```
# cat /etc/oracle/ocr.loc
ocrconfig_loc=/opt/oracle/product/crs10.2/cdata/localhost/local.ocr
local_only=TRUE
# grep 'h[1-3]' /etc/inittab
h1:35:respawn:/etc/init.d/init.cssd run >/dev/null 2>&1 </dev/null
```

ASM 实例也能够启动了。使用 ASM 存储的所有 RDBMS 实例也能够加载控制文件并打开数据库了。

```
$ env ORACLE_SID=+ASM1 sqlplus / as sysdba
SQL> STARTUP PFILE=$ORACLE_HOME/dbs/pfile+ASM1.ora
ASM instance started
Total System Global Area      83886080 bytes
Fixed Size                     1260216 bytes
Variable Size                  57460040 bytes
ASM Cache                      25165824 bytes
ASM diskgroups mounted
SQL> EXIT
$ env ORACLE_SID=TEN1 sqlplus / as sysdba
SQL> ALTER DATABASE MOUNT;
Database altered.
SQL> ALTER DATABASE OPEN;
Database altered.
SQL> SELECT name FROM v$datafile WHERE file#=1;
NAME
-----
+DG/ten/datafile/system.259.628550039
```

启动或关闭 OCSSD 是由 /etc/init.d/init.cssd {start|stop} 命令来实现的。

```
# /etc/init.d/init.cssd stop
Stopping CSSD.
Shutting down CSS daemon.
Shutdown request successfully issued.
Shutdown has begun. The daemons should exit soon.
```

34.2.4 重启适用于 RAC 的 CRS 环境

一旦表决磁盘恢复, RAC 可选项就可以重新连接, 而适用于 RAC 的 CRS 配置也可以重启了。在运行 make 命令前, 确保关闭所有的 RDBMS 和 ASM 实例。

```
$ cd $ORACLE_HOME/rdbms/lib
$ make -f ins_rdbms.mk rac_on ioracle
```

对于 root 用户, 可以运行 localconfig delete 命令来删除当前的 CRS 配置。

```
# $ORA_CRS_HOME/bin/localconfig delete
Stopping CSSD.
Shutting down CSS daemon.
Shutdown request successfully issued.
Shutdown has begun. The daemons should exit soon.
```

如果没有用脚本 \$ORA\_CRS\_HOME/install/rootdeinstall.sh 用二进制 0 重写 OCR 文件, 那么

就没有必要用备份文件恢复 OCR。因为本地配置无法进入保存 OCR 的裸设备，所以 OCR 的备份文件只是一个预防措施。后面的步骤包括：恢复 CRS 脚本、向/etc/inittab 中添加完整的条目，并将 inittab 的变化情况通知给 init 进程。

```
# tar xvfP crs_local_only_false.tar # -P extracts with absolute path
/etc/oracle/
...
/etc/rc6.d/K96init.crs
# cat inittab.crs >> /etc/inittab # append CRS entries to inittab
# telinit q # notify init of changes to inittab
```

过了一会儿，CRS 进程开始运行。

```
# ps -ef|grep d\\bin
root      319 31977 1 02:21 ? 00:00:01 /opt/oracle/product/crs10.2/bin/crsd.bin reboo
t
oracle    738 32624 0 02:22 ? 00:00:00 /opt/oracle/product/crs10.2/bin/ocssd.bin
oracle    32577 31975 0 02:21 ? 00:00:01 /opt/oracle/product/crs10.2/bin/evmd.bin
# crsctl check crs
CSS appears healthy
CRS appears healthy
EVM appears healthy
```

如果集群资源配置正确，CRS 就会自动开启本地 ASM 和 RDBMS 实例。多个 RAC 实例也开始启动，并打开相同的数据库。

```
$ env ORACLE_SID=TEN1 sqlplus / as sysdba
SQL> SELECT inst_number, trim(INST_NAME) inst_name FROM v$active_instances;
INST_NUMBER INST_NAME
-----
1 dbserver1.oradbpro.com:TEN1
2 dbserver2.oradbpro.com:TEN2
```

34.3 小结

本章讨论了在所有表决磁盘和 OCR 设备失效时恢复 RAC 集群中单个节点的应急操作。该操作也适用于其他因服务器失效而导致 Oracle Clusterware 无法运行的场景。该操作中修改了 Oracle Clusterware 的配置以及 ASM 和 RDBMS 实例。该操作不是破坏性的，一旦引起集群失效的问题被解决，所有的改变都可以恢复。

Part 11

第十一部分

实用工具

本部分内容

- 第 35 章 OERR
- 第 36 章 数据恢复管理器管道接口
- 第 37 章 ORADEBUG SQL\*Plus 命令

UNIX和Linux版本的Oracle软件都带有OERR实用工具。利用该工具可以迅速获取ORACLE数据管理系统中的错误信息文本内容，这些错误信息根据错误码来编排。错误码由功能标识名(facility name)和一个正数组成。而且，OERR读取错误码对应的cause和action部分，指示错误原因和解决方案，有助于随时解决错误码所指示的问题。

在Oracle10g的文档中，提到了OERR脚本和TNS错误（见*Oracle Database Net Services Administrator's Guide*）。但文档中没有提及OERR支持除了TNS以外的其他类型的错误。*SQL\*Plus User's Guide and Reference Release 9.2*指出“UNIX OERR脚本现在识别以SP2-为前缀的错误码；并显示其错误原因和解决方法”，但没有说明怎么使用OERR工具。文档中还忽略了一点，那就是：由OERR读取的信息文件中，记录了DBMS实例所涉及的大多数事件（例如，扩展SQL跟踪的10046事件），并对这每一个事件都有简短的描述。

该OERR实用工具不适用于Windows。由第三方开发的适用于Windows的替代方案，可在互联网上免费获得。

35.1 OERR脚本介绍

有时候，在错误发生时，运行在ORACLE数据库管理系统服务器上的程序只抛出了错误码，而没有给出相关信息。这时，OERR脚本就能派上用场了，因为它能提供与错误码关联的详细错误信息。但它所能做的不仅仅是这些。如果相关信息存在的话，它还可以打印出可能的原因和针对该错误的解决方案。唯一遗憾的就是，它“知道”错误是什么，也“知道”怎么解决它，但是它却没有自己去解决那个问题！这里有一个OERR运行的例子：

```
$ oerr ora 2292
02292, 00000,"integrity constraint (%s.%s) violated - child record found"
// *Cause: attempted to delete a parent key value that had a foreign
//      key dependency.
// *Action: delete dependencies first then parent or disable constraint.
```

针对UNIX平台分发的ORACLE DBMS软件都带有OERR实用工具。该工具通过Bourne Shell脚本\$ORACLE\_HOME/bin/oerr来运行。因为该脚本位于\$ORACLE\_HOME/bin目录下，因此无需在环境变量PATH中添加一个目录来使用OERR工具。当以无参数的形式运行时，OERR

将打印如下信息：

```
$ oerr
Usage: oerr facility error
```

Facility is identified by the prefix string in the error message.
For example, if you get ORA-7300, "ora" is the facility and "7300"
is the error. So you should type "oerr ora 7300".

If you get LCD-111, type "oerr lcd 111", and so on.

OERR 读取 UNIX 版本的 ORACLE 数据管理系统所附带的纯文本的错误信息文件。Windows 版本中只有二进制的错误信息文件。纯文本错误信息文件的扩展名是.msg，而二进制的错误信息文件的后缀名是.msb。错误信息文件的文件路径是\$ORACLE\_HOME/component/mesg/facilityus.msg。借助于\$ORACLE\_HOME/lib/facility.lis 文件，OERR 将错误码的功能标识名映射到相应的组件名称，并使用 UNIX 工具 awk 来从错误信息文件中读取对应的错误信息。

很多数据库管理员对 OERR 已经很熟悉。但是，较少人知道，该工具的功能不仅限于支持 ORA。比如，如果你运行 Oracle Clusterware 来支持 RAC 时收到错误信息 CLSS-02206: local node evicted by vendor node monitor，那么 OERR 工具会给出如下建议：

```
$ oerr clss 2206
2206, 1, "local node evicted by vendor node monitor"
// *Cause: The Operating System vendor's node monitor evicted the local node.
// *Action: Examine the vendor node monitor's logs for relevant information.
```

一些常见的功能标识名和组件名称在表 35-1 中列出。表中没有列出全部条目。Oracle10g 一共有 174 个功能标识。

表35-1 功能标识

| 功能标识 | 描 述 |
|------|--------------------------------------|
| CLSR | 关于高可用性Oracle实时应用集群（RACHA）的信息文件 |
| CLSS | 关于Oracle集群同步服务的信息文件 |
| CLST | 关于集群启动服务共享模块的信息文件 |
| DBV | 数据库验证工具（DBVERIFY） |
| DGM | Oracle Data Guard中间件命令行工具DGMGRL |
| EXP | 导出工具 |
| IMP | 导入工具 |
| LDAP | Oracle因特网轻量级目录访问服务器（OiD LDAP Server） |
| LPX | XML解析器 |
| LRM | 关于CORE参数管理的错误信息文件 |
| LSX | XML Schema处理器 |
| NID | 新数据库Id（NID工具） |
| OCI | Oracle编程接口 |
| ORA | Oracle数据库管理系统服务器 |

(续)

| 功能标识 | 描 述 |
|------|----------------------------|
| PCP | Pro*C/C++ C/SQL/PLS/DDl解析器 |
| PLS | PL/SQL |
| PLW | PL/SQL警告 |
| PROT | Oracle集群注册表(OCR)工具 |
| RMAN | 数据恢复管理 |
| SP2 | SQL*Plus |
| TNS | Oracle网络服务（透明网络底层） |
| UDE | 数据批量导出 |
| UDI | 数据批量导入 |
| UL | SQL*Loader |

35.2 检索未在文档中说明的事件

除了已经在文档中说明的错误信息外，ORACLE 错误信息文件还包含了那些未在文档中说明的事件代码。这些事件代码的范围是 10000~10999，系统将采取与处理错误码一样的方式处理它们。跟真正的错误码一样，它们都有各自的关联信息。在 oerr 的输出信息中，在 cause 和 action 部分会描述更多与这些事件相关的信息。

下面的 shell 脚本（见文件 oerr.sh）列出了关于某个特定的 ORACLE 数据库管理系统版本的所有事件的信息。

```
event=10000
counter=0
while [ $event -lt 11000 ]
do
    text=`oerr ora $event`
    if [ "$text" != "" ]; then
        counter=`expr $counter + 1`
        echo "$text"
    fi
    event=`expr $event + 1`
done
echo "$counter events found."
```

在 Oracle9i 上运行该脚本可以得到 623 个事件，在 Oracle10g 上可以得到 713 个事件，Oracle11g 上有 761 个事件。下面是运行脚本 oerr.sh 所得到的事件的摘要，包含了一些为人熟知的事件，例如 10046 与 10053 事件。

```
10046, 00000, "enable SQL statement timing"
// *Cause:
// *Action:
10047, 00000, "trace switching of sessions"
// *Cause:
// *Action:
```

```

10048, 00000, "Undo segment shrink"
// *Cause:
// *Action:
10049, 00000, "protect library cache memory heaps"
// *Cause:
// *Action: Use the OS memory protection (if available) to protect library
//          cache memory heaps that are pinned.
10050, 00000, "sniper trace"
// *Cause:
// *Action:
10051, 00000, "trace OPI calls"
// *Cause:
// *Action:
10052, 00000, "don't clean up obj$"
// *Cause:
// *Action:
10053, 00000, "CBO Enable optimizer trace"
// *Cause:
// *Action:
10056, 00000, "dump analyze stats (kdg)"
// *Cause:
// *Action:
10057, 00000, "suppress file names in error messages"
// *Cause:
// *Action:

```

如果是在非 UNIX 平台上，只要有 ORACLE 客户端，你就可以在任何平台上运行下列匿名 PL/SQL 程序块（见文件 oerr.sql）。

```

SET SERVEROUTPUT ON SIZE 1000000
DECLARE
    err_msg VARCHAR2(4000);
    counter number:=0;
BEGIN
    FOR err_num IN 10000..10999
    LOOP
        err_msg := SQLERRM (-err_num);
        IF err_msg NOT LIKE '%Message '||err_num||' not found%' THEN
            DBMS_OUTPUT.PUT_LINE (err_msg);
            counter:=counter+1;
        END IF;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE (counter || ' events found. ');
END;
/

```

可惜的是，使用 PL/SQL 无法获得由 oerr 报告的关于 cause 和 action 部分的其他信息。尽管如此，人们还是可以从 *Oracle Database Error Messages Guide* 中找到这些信息。下面是运行脚本 oerr.sql 的例子：

```
$ sqlplus system/manager@ten.oradbpro.com @oerr.sql
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production
With the Partitioning, Real Application Clusters, Oracle Label Security and Data
Mining options

ORA-10000: control file debug event, name 'control_file'
ORA-10001: control file crash event1
ORA-10002: control file crash event2
ORA-10003: control file crash event3
ORA-10004: block recovery testing - internal error
ORA-10005: trace latch operations for debugging
...
ORA-10998: event to enable short stack dumps in system state dumps
ORA-10999: do not get database enqueue name
713 events found.
```

源代码库中有 Oracle9i、Oracle10g 以及 Oracle11g 的 10000~10999 所有事件的详细列表。

35.3 源代码库

表 35-2 中列出了本章的源代码文件以及它们的功能。

表35-2 OERR工具源代码库

| 文 件 名 | 功 能 |
|----------------|-----------------------------|
| events_10g.txt | Oracle10g中从10000~10999的事件列表 |
| events_11g.txt | Oracle11g中从10000~10999的事件列表 |
| events_9i.txt | Oracle9i中从10000~10999的事件列表 |
| oerr.sh | 检索和统计一个数据库管理系统版本的所有事件（事件编号） |
| oerr.sql | 检索和统计所有事件的PL/SQL脚本 |

ORACLE 数据库管理系统支持管道 (pipe) 作为数据库会话之间的交互手段。它分为私有管道和公共管道。为了增强安全性, 私有管道只为 SYS 用户所拥有, 而不能为其他用户所访问。数据恢复管理 (RMAN) 管道接口应用了私有管道。它非常适合于将 RMAN 集成到可移植的第三方数据备份和恢复工具中。同时, 它也适合于执行鲁棒性强的脚本, 这些脚本可以处理目录实例失效 (catalog instance failures)、数据块损坏以及实时应用集群上的跨节点并行数据备份与恢复等。

RMAN 管道接口已在文档中说明。文档中还规定, 所有从 RMAN 中用管道 (DBMS\_PIPE) 方式发送和接收的内容必须是 VARCHAR2 数据类型。但是, 关于如何将 VARCHAR2 类型的数据打包成管道消息, 比如一个管道消息中含一个还是多个 VARCHAR2 类型的数据, 以及如何将错误栈中的消息包装成管道消息等, 都没有相关的文档规定。而且, 关于 RMAN 在发生错误时是终止还是继续运行也没有规定。

36.1 数据恢复管理介绍

Oracle8 及后续版本都有备份和恢复工具 RMAN, 任何使用过该工具的人都知道, 它可以在终端窗口与用户交互。RMAN 可以从脚本中读取命令 (如 `rman CMDFILE script_name.rvc`)。RMAN 通过检查传递给调用它的程序或脚本的退出代码来确定是否运行成功。退出代码为 0 则表示成功, 非 0 则表示失败。RMAN 可以将操作记录写入日志文件 (参数是 LOG, 以前是 MSGLOG)。

当引入更多高级特性时, 选项之间如果发生冲突就会导致错误。比如, 当使用了恢复目录而恢复目录不存在时, 如果 RMAN 在 NOCATALOG 模式下, 数据能够备份, 只要等目录实例存在时, 使其与控制文件再同步 (resynchronization) 即可。等到下一次 RMAN 执行备份、恢复或列举命令而连接到目录时, 控制文件与目录之间的再同步会自动进行。但如果 RMAN 在命令行选项 CATALOG 模式下, 或者脚本中有 CONNECT CATALOG 声明时, RMAN 就会抛出错误并退出。这时, 备份就不会执行。手动实现再同步可以使用命令 RESYNC CATALOG。这个命令还可以用来同步控制文件与独立系统上的多个 RMAN 目录, 以便能更好地管理目录, 而不需要 Data Guard 或集群来保护目录。

因为目录缺失导致的备份无法进行的风险是可以很容易避免的。需要解决的关键问题是, 调

用 RMAN 运行一个脚本中一系列命令的程序, 如何得知 RMAN 中发生的错误的原因和问题关键所在。当因数据块损坏而无法正常工作备份时, 也需要解决这个问题。实际上, 如果为 RMAN 设定了可以容忍的损坏的数据块的最大数量, 那么当实际损坏的数据块数目小于这个最大数量时, 备份依然可以完成。SET MAXCORRUPT FOR DATAFILE 命令可以用来设置这个最大数量。当备份由于数据块损坏而失败时, 调用 RMAN 的程序可以通过管道接口很容易地检查 RMAN 返回的错误栈。如果程序得到错误信息提示 ORA-19566: exceeded limit of 0 corrupt blocks for file *filename*, 这时, 程序可以检测到哪些数据文件受到了损坏数据块的影响, 并在选择合适的 MAXCORRUPT 设置后重启备份进程。

当传递一个脚本给 RMAN 时, 一种很好的做法是同时使用两个不同的脚本(一个用于备份数据库, 另一个用于备份归档重做日志)来确保备份数据库时所发生的错误不会导致 RMAN 退出, 且此时可以不必执行 BACKUP ARCHIVELOG 命令。因为这一切, 包括启动一个 RMAN 的实例都可以通过 PL/SQL 来实现(使用 DBMS\_SCHEDULER 与外部作业, 见第 19 章), 所以一个基于管道接口的第三方工具会具有高度的可移植性。

36.2 DBMS\_PIPE 介绍

DBMS\_PIPE 是可以在数据库客户端之间异步传递消息的包。与高级队列不同, 没有提供消息的持久化机制。在实例关闭时, 消息就会丢失。DBMS\_PIPE 包使得数据库客户端可以调用一个基本上不连接到 ORACLE 数据库的应用程序上的功能, 只要这个应用程序能够通过管道消息处理请求即可。因为引入了外部程序代理, 访问数据库管理系统实例之外的服务就成为可能了。

一个管道消息包含一个或多个条目, 这些条目都是 ORACLE DBMS 基本数据类型, 如 CHAR、DATE、NUMBER、RAW 或者 VARCHAR2。消息的内容不会受事务处理的影响。组装消息要使用 DBMS\_PIPE.PACK\_MESSAGE 增加条目, 而发送消息要调用 DBMS\_PIPE.SEND\_MESSAGE。接收消息时, 就会调用 DBMS\_PIPE.RECEIVE\_MESSAGE。要读取消息中的条目, 就要调用 DBMS\_PIPE.UNPACK\_MESSAGE。

有两种类型的管道: 私有管道和公共管道。共有管道可以被任何数据库客户端使用, 而私有管道只有具备 SYSDBA 特权的事务才可以使用。RMAN 使用私有管道以确保只有具备特权的数据库客户端才可以向它发送指令。

使用 PIPE 选项可以让 RMAN 从私有管道中读取命令。

```
$ rman target / pipe '"PIPE"' log rman.log
```

当以这种方式启动时, RMAN 会创建两个私有管道: 一个用于接收消息, 另一个用于发送回复。后缀\_IN 用于输入管道的命名, 而后缀\_OUT 用于输出管道的命名。前缀 ORA\$RMAN\_用于两者的命名。下面是查询 V\$DB\_PIPES 视图所返回的结果, 从中我们可以看到前述例子所对应的私有管道的名字:

```
SQL> SELECT * FROM v$db_pipes;
OWNERID NAME                                TYPE      PIPE_SIZE
-----
```

| | | |
|----------------------|---------|-----|
| o ORA\$RMAN_PIPE_OUT | PRIVATE | 354 |
| o ORA\$RMAN_PIPE_IN | PRIVATE | 353 |

36.3 RMAN\_PIPE\_IF 包

在源代码库中,我提供了一个 PL/SQL 包来解释 DBMS\_PIPE 的使用以及 RMAN 所发送的消息。包的名字是 RMAN\_PIPE\_IF (IF 是 interface 的缩写)。在写这个包时,我总结了以下几点经验。

- RMAN 通常为每个管道消息发送一行输出。
- RMAN 的错误栈会在一个消息中包含多行 (VARCHAR2 类型的条目)。
- 发送给 RMAN 的命令可能每个消息只有一个条目且包含换行符。
- 在管道模式下, RMAN 不会像在执行脚本时那样,将所接收到的命令写入到日志文件中。因此,备份工具在调用处于管道模式的 RMAN 时,就要先将命令写入日志文件后,然后再将命令通过 DBMS\_PIPE 发送给 RMAN。
- 当 RMAN 准备处理下一条命令时,它会发送一条管道消息,该消息仅含有 RMAN-00572: waiting for dbms\_pipe input 这一条目。
- RMAN 不会因为错误终止,相反,它会继续运行,并通过发送 RMAN-00572 消息来表明它准备接收下一条命令。

除了处理前面所提到的内容外, RMAN\_PIPE\_IF 还提供检查 RMAN 发送的管道消息的方法,并提供如下附加功能。

- 通过检测 RMAN-00572: waiting for dbms\_pipe input, 它可以指示什么时候 RMAN 准备好接收下一条命令。
- 它可以将 RMAN 错误栈信息中多个 VARCHAR2 条目连接成包含一个单一的 VARCHAR2 字符串的单一消息,并将其返回给调用者。
- 命令失效时,它返回 RMAN 错误码 (RMAN-*nnnnn*) 以及从错误栈顶部取得的错误消息给调用者。
- 当命令失效且存在 ORACLE DBMS 错误时,它返回 ORACLE DBMS 错误码 (ORA-*nnnnn*) 以及从错误栈顶部取得的错误消息给调用者。
- 当有程序调用它来接收一条消息时,如果 RMAN 输出管道中没有消息,在超过最长等待时间 1 秒后,它会返回一条空消息以及值为 0 的行的计数。

36.4 RMAN\_PIPE\_IF 包详述

RMAN\_PIPE\_IF 中的源代码的情况将在后面源代码章节说明。我在代码中添加了大量的注释来解释包的功能以及参数。因为模式 SYS 已被保留给数据字典,因此,该包是使用 AUTHID CURRENT\_USER 在模式 SITE\_SYS 下创建的。这样,拥有 SYSDBA 权限的客户端可以在保留 SYSDBA 权限的情况下执行该包。否则,只要具备它的所有者 SITE\_SYS 拥有的权限,就可以执行该包,然而此时很可能无法访问 RMAN 所使用的私有管道。

```

create or replace package site_sys.rman_pipe_if authid current_user as
--send: 返回码与 DBMS_PIPE.SEND_MESSAGE 相同: 0 表示成功
function send(pipe_arg varchar2, --与 RMAN 命令行选项 PIPE 的参数相同
msg varchar2) return number; --通过 DBMS_PIPE 发送给 RMAN 的消息可能包含多行
--receive: 如果管道为空, 返回 0, 否则返回接收的条目 (1 行或者多行)
function receive(pipe_arg varchar2, msg out varchar2, --通过 DBMS_PIPE 从 RMAN 接收到的消息
-- wait_for_input: 0: RMAN 还没准备好执行下一条命令
--1: RMAN 发送 RMAN-00572: waiting for dbms_pipe input 之后准备好执行下一条命令,
wait_for_input out number,
rman_error out number, --来自 RMAN 错误栈的 RMAN-nnnnn 错误码, 如果无错则为 0
rman_msg out varchar2, --错误码为 RMAN-nnnnn 的错误详细信息, 如果无错则为 NULL
ora_error out number, --来自 RMAN 错误栈的 ORA-nnnnn 错误码, 如果无错则为 0
ora_msg out varchar2 --错误码为 ORA-nnnnn 的错误详细信息, 如果无错则为 NULL
) return number;
end;
/

```

因为包的所有者是 SITE\_SYS, 需要给用户 SITE\_SYS 赋予在 DBMS\_PIPE 运行的执行许可 (EXECUTE)。源代码库中的 rman\_pipe\_if.sql 文件有该包的详细内容, 因为太长, 不在此详细展示。

36.5 使用 RMAN\_PIPE\_IF 包

在本节中, 我将展示在 SQL\*Plus 中使用 RMAN\_PIPE\_IF 包的用途。与我在在 SQL\*Plus 中所调用的命令一样, 使用管道接口的备份与恢复工具能从编程语言, 如 PL/SQL、C (支持 Pro\*C) 或 Perl (支持 DBI 以及 DBD::Oracle) 中调用相同的命令。在以下部分, 我们将着力解决在本章开始提出的问题:

- 连接目录失败, 导致 RMAN 退出所执行的脚本;
- 数据文件损坏, 使得 RMAN 取消备份指令, 并退出所执行的脚本。

首先, RMAN 需要以管道接口选项来启动。注意, 单词 PIPE 上的双引号确保 RMAN 不会将其误认为一个关键词。而外层的单引号确保 shell 不会移除 PIPE 两侧的双引号。

```
$ rman TARGET / PIPE "'PIPE'" LOG rman.log
```

初始化完成后, RMAN 会提示它已经准备好接收命令。我们需要接收 RMAN 启动后发送的消息。调用 RMAN\_PIPE\_IF 包需要一系列 SQL\*Plus 绑定变量。大多数变量都对应到 RMAN\_PIPE\_IF 包中函数的参数, 因此它们的含义应该是自明的。变量 msg 用来表示接收到的消息, 而 cmd 用来表示发送给 RMAN 的命令或脚本的文本内容。变量 rv 表示调用 RMAN\_PIPE\_IF 时的返回值。

```

SQL> VARIABLE rv NUMBER
SQL> VARIABLE msg VARCHAR2(4000)
SQL> VARIABLE cmd VARCHAR2(4000)

```



```
SQL> VARIABLE rman_error NUMBER
SQL> VARIABLE ora_error NUMBER
SQL> VARIABLE rman_msg VARCHAR2(4000)
SQL> VARIABLE ora_msg VARCHAR2(4000)
SQL> VARIABLE pipe_arg VARCHAR2(1000)
SQL> VARIABLE wait_for_input NUMBER
```

接着，在 RMAN 启动后，我们需要设定用做 PIPE 选项的参数的标识符。该标识符会作为私有管道名字的一部分。与前面的例子一样，我们使用单词 PIPE。

```
SQL> SET NULL <NULL>
SQL> BEGIN
SQL>   :pipe_arg:='PIPE';
SQL> END;
SQL> /
```

现在我们可以通过调用 ORAS\$RMAN\_PIPE\_OUT 以尝试从私有管道中获取信息：

```
SQL> EXEC :rv:=site_sys.rman_pipe_if.receive(:pipe_arg, :msg, :wait_for_input, -
> :rman_error, :rman_msg, :ora_error, :ora_msg)
PL/SQL procedure successfully completed.
SQL> COL msg FORMAT a90
SQL> COL rv FORMAT 99
SQL> SET LINESIZE 130
SQL> COL wait_for_input HEADING "Wait|for|input" FORMAT 99999
SQL> COL rman_error HEADING "RMAN|Error" FORMAT 99999
SQL> COL ora_error HEADING "ORA|Error" FORMAT 99999
SQL> SELECT :rv rv, :msg msg, :wait_for_input wait_for_input FROM dual;
RV MSG                                Wait for input
-----
1 connected to target database: TEN (DBID=2848896501)                0
```

RMAN 发送一个只有单个条目（或单行）的消息。这可以根据其返回值 RV=1 得知。RMAN 中返回的消息表明它连接到了目标数据库。此时，RMAN 尚未准备好接收命令，因为 WAIT\_FOR\_INPUT=0。让我们接着查看下一条消息。

```
SQL> EXEC :rv:=site_sys.rman_pipe_if.receive(:pipe_arg, :msg, :wait_for_input, -
> :rman_error, :rman_msg, :ora_error, :ora_msg)
PL/SQL procedure successfully completed.
SQL> SELECT :rv rv, :msg msg, :wait_for_input wait_for_input FROM dual;
RV MSG                                Wait for input
-----
1 RMAN-00572: waiting for dbms_pipe input                            1
```

现在 RMAN 准备接收命令了 (WAIT\_FOR\_INPUT=1)，因此我们让 RMAN 去连接目录数据库。

```
SQL> BEGIN
  :cmd:='CONNECT CATALOG rman/rman@ten_tcp.world;';
  -- send a message
  :rv:=site_sys.rman_pipe_if.send(:pipe_arg, :cmd);
END;
/
PL/SQL procedure successfully completed.
```

```
SQL> PRINT rv
RV
---
0
```

让我们找出 CONNECT CATALOG 命令是否已经成功完成（为了简便起见，对 site\_sys.rman\_pipe\_if.receive 的调用此处不详述）。

```
SQL> SELECT :rv rv, :msg msg FROM dual;
RV MSG
```

```
-----
4 RMAN-00571: =====
RMAN-00569: ===== ERROR MESSAGE STACK FOLLOWS =====
RMAN-00571: =====
RMAN-04004: error from recovery catalog database: ORA-12170: TNS:Connect timeout
occurred
```

发生了一个阻止 RMAN 连接到目录数据库的错误。尽管如此，我们很快会看到，RMAN 仍然继续运行，并准备接收更多的命令。但是，首先，让我们先看看关于这个由 RMAN\_PIPE\_IF 返回的 RMAN 错误码的附带信息。

```
SQL> SELECT :rman_error rman_error, :rman_msg rman_msg, :ora_error ora_error,
:wait_for_input wait_for_input
FROM dual;
```

| RMAN
Error RMAN_MSG | ORA
Error | Wait
for
input |
|---|--------------|----------------------|
| 4004 RMAN-04004: error from recovery catalog database: TNS:Connect timeout occurred | 0 | 0 |

RMAN\_PIPE\_IF 包从错误栈中提取了错误码 RMAN-04004。在错误栈中没有独立的 ORA-*nnnnnn* 错误（而是包含在 RMAN-04004 错误消息中）。此时 RMAN 尚未准备好接收下一条命令（WAIT\_FOR\_INPUT=0）。我们继续接收下一条消息。

```
SQL> EXEC :rv:=site_sys.rman_pipe_if.receive(:pipe_arg, :msg, :wait_for_input, -
> :rman_error, :rman_msg, :ora_error, :ora_msg)
PL/SQL procedure successfully completed.
SQL> SELECT :rv rv, :msg msg FROM dual;
RV MSG
```

```
-----
1 RMAN-00572: waiting for dbms_pipe input
```

现在，RMAN 又开始准备接收下一条命令了。于是，尽管 RMAN 没有连接到目录，我们还是发送一条 BACKUP 命令。

```
SQL> BEGIN
:cmd:='run {
    backup format "DB-%d-%u.bkp" tablespace appdata;
}';
-- send a message
:rv:=site_sys.rman_pipe_if.send(:pipe_arg, :cmd);
```

```

END;
/
PL/SQL procedure successfully completed.
SQL> PRINT rv
RV
---
0

```

让我们来看看 RMAN 会做出什么反应。

```

SQL> EXEC :rv:=site_sys.rman_pipe_if.receive(:pipe_arg, :msg, :wait_for_input, -
> :rman_error, :rman_msg, :ora_error, :ora_
msg)
PL/SQL procedure successfully completed.
SQL> SELECT :rv rv, :msg msg FROM dual;
RV MSG
-----
1 Starting backup at 18.07.07 16:54

```

RMAN 开始备份表空间 APPDATA。接着, RMAN 开始发送一些消息来说明备份的进展情况。这里, 我只列举其中两条消息来加以说明。我省略了用来接收这些消息的代码。

```

SQL> SELECT :rv rv, :msg msg FROM dual;
RV MSG
-----
1 using target database control file instead of recovery catalog
...
SQL> SELECT :rv rv, :msg msg FROM dual;
RV MSG
-----
1 input datafile fno=00007 name=F:\ORADATA\TEN\APPDATA01.DBF

```

在接收更多消息之后, 会发现备份由于数据块损坏而失败。当然, 这是因为我故意损坏了表空间 APPDATA<sup>①</sup>的一个代码段。下面是 RMAN 在遇到数据损坏时发送的消息:

```

SQL> SELECT :rv rv, :msg msg FROM dual;
RV MSG
-----
6 RMAN-00571: =====
RMAN-00569: ===== ERROR MESSAGE STACK FOLLOWS =====
RMAN-00571: =====
RMAN-03009: failure of backup command on ORA_DISK_1 channel at 07/18/2007 17:01:
59
ORA-19566: exceeded limit of 0 corrupt blocks for file F:\ORADATA\TEN\APPDATA01.
DBF
RMAN-10031: RPC Error: ORA-19583

```

让我继续检查其他错误消息。

① 一个数据块损坏可以在表空间或数据文件离线时通过 UNIX 命令 dd 来实现 (Cygwin 也有这个命令)。下面这个命令会用许多二进制的“0”来覆盖第 10 个 8 KB 的数据块:
dd bs=8192 if=/dev/zero of=data\_file\_name count=1 seek=9 conv=notrunc。

```
SQL> SELECT :rman_error rman_error, :rman_msg rman_msg, :ora_error ora_error,
:wait_for_input wait_for_input
FROM dual;
```

| RMAN
Error | RMAN_MSG | ORA
Error | Wait
for
input |
|---------------|---|--------------|----------------------|
| 3009 | RMAN-03009: failure of backup command on ORA_DISK_1 channel
at 07/18/2007 17:13:55 | 19566 | 0 |

错误栈顶端的 RMAN 错误是 RMAN-03009。另外，还有一个 ORA-19566 错误。ORA-*nnnnn* 错误的文本内容保存在绑定变量 ORA\_MSG 中。

```
SQL> SELECT :ora_msg "ORA-nnnnn error" FROM dual;
ORA-nnnnn error
```

```
-----
ORA-19566: exceeded limit of 0 corrupt blocks for file F:\ORADATA\TEN\APPDATA01.DBF
```

绑定变量中保存的错误信息包含了受损坏数据块影响的文件名。我们可以很容易从错误信息的结尾部分提取该文件名。知道了损坏文件的名字，我们就可以很容易地设定合适的 MAXCORRUPT 值来重启备份过程，最终让 DBMS 实例完成备份过程。

```
SQL> BEGIN
:cmd:='run {
    set maxcorrupt for datafile "F:\ORADATA\TEN\APPDATA01.DBF" to 10;
    backup format "DB-%d-%u.bkp" tablespace appdata;
  }';
:rv:=site_sys.rman_pipe_if.send(:pipe_arg, :cmd);
END;
/
PL/SQL procedure successfully completed.
SQL> PRINT rv
RV
---
0
```

因为数据文件中只有一个损坏块，因此 RMAN 报告备份成功并返回所创建的备份块的名字。

```
SQL> SELECT :rv rv, :msg msg FROM dual;
RV MSG
```

```
-----
1 piece handle=DB-TEN-JHIN5G18.BKP tag=TAG20070718T172808 comment=NONE
```

成功备份后，需要在 V\$DATABASE\_BLOCK\_CORRUPTION 中添加一个条目来描述数据块的损坏信息。

```
SQL> SELECT * FROM v$database_block_corruption;
FILE#      BLOCK#      BLOCKS CORRUPTION_CHANGE# CORRUPTION_TYPE
-----
7          20          1          0 FRACTURED
```

注意：如果备份由于数据损坏而失败，那么上述这些消息就不会添加到 V\$DATABASE\_BLOCK\_CORRUPTION 中。为了修复这些损坏的数据，需要通过运行 RMAN BLOCKRECOVER 命令（如 BLOCKRECOVER DATAFILE 7 BLOCK 20）来进行块介质恢复（block media recovery）。

关闭 RMAN，可以发送一条包含命令 EXIT 的消息如下：

```
SQL> BEGIN
      :cmd:='exit;';
      :rv:=site_sys.rman_pipe_if.send(:pipe_arg, :cmd);
      END;
/
```

RMAN 将返回另一条消息并退出。

```
SQL> EXEC :rv:=site_sys.rman_pipe_if.receive(:pipe_arg, :msg, :wait_for_input, -
> :rman_error, :rman_msg, :ora_error, :ora_msg)
RV MSG
-----
1 Recovery Manager complete.
```

在 RMAN 关闭后，任何试图从管道中读取消息的努力都会导致抛出 ORA-06556: the pipe is empty, cannot fulfill the unpack\_message request 的异常。为了方便用户，RMAN\_PIPE\_IF 包会捕捉该异常，并返回一条空消息和值为 0 的行计数。当然，上述情形不是仅在 RMAN 关闭后才出现。当 RMAN 或者 DBMS 实例没有能够完成已经启动的命令时，它们可能会在很长时间内未返回任何消息。RMAN\_PIPE\_IF 包在等待一条消息超过最长延时 1 秒后仍未接收到任何消息，也会返回一条空消息和值为 0 的行计数。

```
SQL> EXEC :rv:=site_sys.rman_pipe_if.receive(:pipe_arg, :msg, :wait_for_input, -
> :rman_error, :rman_msg, :ora_error, :ora_msg)
PL/SQL procedure successfully completed.
SQL> SELECT :rv rv, :msg msg FROM dual;
RV MSG
-----
0 <NULL>
```

36.6 验证备份块

另一个有意思的功能是可以很容易地通过命令 VALIDATE 来检查备份块的一致性。VALIDATE 逐块读取和验证备份块，但并不试图做任何的修复。VALIDATE CHECK LOGICAL 相对于 VALIDATE 而言会对备份块进行更加深入的检测。

当 RMAN 以 MSGNO 选项启动时，所有由它包装到管道消息（除终止消息 Recovery Manager complete 之外）中的条目（或行）都会以 RMAN-*nnnnn* 开头。涉及备份块的消息以 RMAN-08530 开头，示例如下：

```
RV MSG
-----
1 RMAN-08530: piece handle=A16J2S5GD.BKP tag=TAG20071205T194829 comment=NONE
```

这一特性有助于我们提取备份块的名字，从而在包含特定备份块的备份块集（backup set）上运行 VALIDATE 命令。备份块集可以从视图 V\$BACKUP\_PIECE 中检索到。

```
SQL> SELECT recid, piece#, tag, status
FROM V$backup_piece
WHERE handle='A16J2S5GD.BKP';
RECID PIECE# TAG STATUS
-----
31 1 TAG20071205T194829 A
```

RECID 列显示了可以用于 VALIDATE 命令的备份块集的键值。

```
$ rman target / msgno
Recovery Manager: Release 10.2.0.1.0 - Production on Wed Dec 5 19:39:27 2007
RMAN-06005: connected to target database: TEN (DBID=2870266532)
RMAN> validate backupset 31;
RMAN-06009: using target database control file instead of recovery catalog
RMAN-08030: allocated channel: ORA_DISK_1
RMAN-08500: channel ORA_DISK_1: sid=32 devtype=DISK
RMAN-08097: channel ORA_DISK_1: starting validation of archive log backupset
RMAN-08003: channel ORA_DISK_1: reading from backup piece A16J2S5GD.BKP
RMAN-08023: channel ORA_DISK_1: restored backup piece 1
RMAN-08511: piece handle=A16J2S5GD.BKP tag=TAG20071205T194829
RMAN-08182: channel ORA_DISK_1: validation complete, elapsed time: 00:00:07.
```

如果备份块集包含了一个损坏的备份块，那么 VALIDATE 命令就会失败，但 RMAN 并不将此情形写入错误栈。

```
RMAN> VALIDATE BACKUPSET 31;
RMAN-12016: using channel ORA_DISK_1
RMAN-08097: channel ORA_DISK_1: starting validation of archive log backupset
RMAN-08003: channel ORA_DISK_1: reading from backup piece A16J2S5GD.BKP
RMAN-01005: ORA-19870: error reading backup piece A16J2S5GD.BKP
ORA-19587: error occurred reading 0 bytes at block number 1
```

损坏的备份块会在实例的警告日志 (alert log) 中记录。

```
Wed Dec 05 19:59:41 2007
Corrupt block 1 found during reading backup piece, file=A16J2S5GD.BKP, corr_type=1
```

36.7 跨节点并行备份与恢复

通过给一个单一的数据库实例分配多个通道 (channel)，RMAN 的备份与恢复操作可以实现并行化。多个 RAC 实例之间的并行化可以通过显示分配通道和提供一个连接字符串（这个字符串可以被解析为多个单独的实例）来实现。但是，这种硬编码的跨节点备份的鲁棒性不好。如果其中一个 RAC 实例不存在，备份就会失败。

通常，每个 RAC 节点都有一到两个连接到 SAN 存储阵列的光纤通道主机总线适配器 (Fibre Channel host bus adapter)，以及一个或多个公共局域网适配器。暂且不考虑用于备份与恢复的数据路径（光纤通道上的局域网或者非局域网），当系统中没有其他瓶颈时，我希望备份操作可以适应于不同数量的局域网或光纤通道适配器。

每个缺失的块都需要一个相互协作的流程，这个流程可以接收 BACKUP DATABASE、RESTORE DATABASE、RESTORE TABLESPACE 等命令的各种形式，并将任务分解成小的部分，并分配给 RMAN

的多个实例来执行。只要 CONNECT TARGET 命令提供的连接字符串是正确的，那么 RMAN 实例在哪里运行（是在集群内部还是外部）就不重要了。在执行备份操作的时候，每个 RAC 节点分配一个磁带驱动器。但是，恢复操作时，因为难以确保所需要的文件正好位于同一个磁带，情况就变得更复杂。因此，引入磁盘池取代磁带驱动或者作为磁带驱动的补充，就可以缓解多个磁带驱动之间的并发问题。

RMAN 管道接口为操作这样的一个跨节点并行备份与恢复协作流程提供了很好的基础。例如，当这样一个协作流程接收到 BACKUP DATABASE 命令时，它就会从 V\$DATAFILE 检索出数据文件并将数据文件以循环的方式分配给多个 RMAN 实例。当其中一个实例完成备份时，它就会通过 DBMS\_PIPE 发送一个 RMAN-00572: waiting for dbms\_pipe input 消息给协作流程。这时，协作流程就可以通过发送另一个 BACKUP DATAFILE 命令给 RMAN 进程来备份下一个数据文件。这个分治算法可能实现初步的负载均衡：由于负载较低的节点拥有较高的 I/O 带宽，因而能够处理更多的数据文件（在给定数据文件都具备标准大小的情况下）。

前一节讲到的 RMAN\_PIPE\_IF 包提供了从不同 RAC 节点上的 RMAN 实例发送和接收消息的基本方法。但是，因为数据库管道是局部化于特定实例的，因此，不可能将消息写入 RAC 实例 1 上的管道 P 后，再从 RAC 实例 2 的管道 P 上读取该消息。因此，在一个多 RAC 实例的运行环境中，要么通过单个数据库会话的数据库链接来调用 RMAN\_PIPE\_IF 以及 DBMS\_PIPE，要么是协作流程为每个实例打开一个数据库会话。当用 PL/SQL 来操作协作流程时，采用前一方法；而用 Perl DBI 或其他支持多个并发的数据库会话的编程语言时就采用后一种方法。我在第 19 章的源代码库中提供了一个利用 PL/SQL 进行一系列备份操作的原型方法。只需要手工运行多个 RMAN 实例来并行备份同一个数据库的不同部分，就可以对跨节点并行备份可以获得的加速比进行初步评估。

36.8 源码库

表 36-1 列出了本章的源代码文件及其功能。

表36-1 数据恢复管理（Recover Manager）源代码库

| 文 件 名 | 功 能 |
|------------------|---|
| rman_pipe_if.sql | 包含PL/SQL包RMAN_PIPE_IF，可用于通过DBMS_PIPE从Recover Manager发送命令和接收回复 |

SQL\*Plus 命令 ORADEBUG 是诊断性能和挂起问题的有效工具。另外，它能够用来核实正确的 IP 地址，来帮助实时应用集群实例间的通信。在 Oracle11g 之前的版本，ORADEBUG TRACEFILE\_NAME 是用来确定一个进程写入的追踪文件的名称的唯一途径。<sup>①</sup>

Oracle9i Real Application Clusters Administration Release 2 手册中包含了 ORADEBUG DUMP 命令的实例。*Oracle9i Database Administrator's Guide Release 2 (9.2) for Windows* 一书中有一节命名为“使用 ORADEBUG 工具”。但是，那里给出的信息很不完整，因为它对在运行更多命令前需要连接到一个进程等都没有介绍。在 Oracle10g 和 Oracle11g 中，唯一提到 ORADEBUG 是 *Error Messages Guide*。

37.1 ORADEBUG 介绍

因为 ORADEBUG 主要是给 Oracle 支持人员 (Oracle Support Personnel) 使用的，因此自从 SQL\*Plus7.3 引入它以来，就很少有文档记载它。尽管如此，即使对于数据库管理的新手，许多 ORADEBUG 命令都是很有用的。

正如它的名字所暗示的那样，ORADEBUG 主要用于调试和跟踪。ORADEBUG 是一个 SQL\*Plus 命令，在 SQL\*Plus prompt 就可以进入，除了需要连接到一个具有 SYSDBA 权限的 DBMS 实例外，无需其他特殊的配置。它可以被用于：

- 使得 SQL 能够追踪你自己的服务进程以及外部服务进程；
- 确定一个进程正往哪个追踪文件写入；
- 转储内部的 ORACLE 结构来诊断数据库挂起以及内存损坏情况；
- 转储数据文件头部信息或撤销分割头部；
- 确定 DBMS 实例使用哪些共享内存块和信号量；
- 找出 RAC 实例使用哪些互联地址和协议；
- 修改 SGA 中的数据结构。

最后一条很有可能只与 Oracle 支持人员有关。

<sup>①</sup> 在 Oracle11g 中，V\$ 的固定视图 V\$PROCESS 中已经添加了 TRACEFILE 列。

37.2 ORADEBUG 使用步骤

使用 ORADEBUG 的基本步骤如下：

- (1) 启动 SQL\*Plus 并以 SYSDBA 来连接；
- (2) 连接到一个 ORACLE 实例的进程中；
- (3) 执行一条或多条 ORADEBUG 命令；
- (4) 检查屏幕输出或者追踪文件；
- (5) 从一个进程中退出（在退出 SQL\*Plus 或加入另一个进程后会自动退出当前进程）。

37.3 ORADEBUG 命令介绍

SQL\*Plus 帮助工具没有涉及 ORADEBUG，但是 ORADEBUG 命令本身的 help 选项会打印出所有相关的命令。

```
SQL> ORADEBUG HELP
HELP          [command]          Describe one or all commands
SETMYPID                      Debug current process
SETOSPID          <ospid>         Set OS pid of process to debug
SETORAPID         <orapid> ['force'] Set Oracle pid of process to debug
SHORT_STACK                      Dump abridged OS stack
DUMP              <dump_name> <lvl> [addr] Invoke named dump
DUMPSGA          [bytes]          Dump fixed SGA
DUMPLIST                      Print a list of available dumps
EVENT            <text>           Set trace event in process
SESSION_EVENT    <text>           Set trace event in session
DUMPVAR          <p|s|uga> <name> [level] Print/dump a fixed PGA/SGA/UGA variable
DUMPTYPE         <address> <type> <count> Print/dump an address with type info
SETVAR           <p|s|uga> <name> <value> Modify a fixed PGA/SGA/UGA variable
PEEK             <addr> <len> [level] Print/Dump memory
POKE             <addr> <len> <value> Modify memory
WAKEUP           <orapid>         Wake up Oracle process
SUSPEND                      Suspend execution
RESUME                      Resume execution
FLUSH                      Flush pending writes to trace file
CLOSE_TRACE                      Close trace file
TRACEFILE_NAME              Get name of trace file
LKDEBUG                      Invoke global enqueue service debugger
NSDBX                      Invoke CGS name-service debugger
-G          <Inst-List | def | all> Parallel oradebug command prefix
-R          <Inst-List | def | all> Parallel oradebug prefix (return output
SETINST      <instance# .. | all> Set instance list in double quotes
SGATOFILE    <SGA dump dir>       Dump SGA to file; dirname in double quotes
DMPCOWSGA    <SGA dump dir>       Dump & map SGA as COW; dirname in double
quotes
MAPCOWSGA    <SGA dump dir>       Map SGA as COW; dirname in double
quotes
HANGANALYZE  [level] [syslevel]  Analyze system hang
```

| | |
|--------------|--|
| FFBEGIN | Flash Freeze the Instance |
| FFDEREGISTER | FF deregister instance from cluster |
| FFTERMINST | Call exit and terminate instance |
| FFRESUMEINST | Resume the flash frozen instance |
| FFSTATUS | Flash freeze status of instance |
| SKDSTTPES | <ifname> <ofname> Helps translate PCs to names |
| WATCH | <address> <len> <self exist all target> Watch a region of memory |
| DELETE | <local global target> watchpoint <id> Delete a watchpoint |
| SHOW | <local global target> watchpoints Show watchpoints |
| CORE | Dump core without crashing process |
| IPC | Dump ipc information |
| UNLIMIT | Unlimit the size of the trace file |
| PROCSTAT | Dump process statistics |
| CALL | <func> [arg1] ... [argn] Invoke function with arguments |

ORADEBUG HELP 命令为每一个命令列出了一行帮助文本。但是，没有比 ORADEBUG HELP 能提供更多详细信息的其他工具了。

37.3.1 连接到一个进程

在 ORADEBUG 命令执行之前，必须加入一个目标进程。这有下列三种选择。

- ❑ 连接你自己的服务进程——为你的 SQL\*Plus 会话提供服务的进程。
- ❑ 根据 ORACLE 进程标识连接到一个外部服务进程。
- ❑ 根据操作系统进程标识连接到一个外部服务进程。

注意：这里并没有要求 ORADEBUG 连接到一个在同一集群的其他节点上的 RAC 实例的进程。相关的命令、参数以及怎么获得这些参数的值在表 37-1 中列出。

表37-1 将ORADEBUG连接到一个进程

| 命 令 | 目 的 |
|-------------------------------|----------------------------------|
| ORADEBUG SETMYPID | 连接到你自己的服务进程 |
| ORADEBUG SETORAPID <i>pid</i> | 连接到一个外部服务进程，且pid= V\$PROCESS.PID |
| ORADEBUG SETOSPID <i>spid</i> | 连接到一个外部服务进程，且spid=V\$PROCESS.PID |

1. SETMYPID

如果你想在自己的会话中运行一些 SQL 或 PL/SQL 语句来记录一些性能和计划的执行情况，你需要使用 ORADEBUG SETMYPID 命令。

```
SQL> ORADEBUG SETMYPID
Statement processed.
SQL> ORADEBUG UNLIMIT
Statement processed.
SQL> ORADEBUG EVENT 10046 TRACE NAME CONTEXT FOREVER, LEVEL 8
Statement processed.
SQL> SELECT ... /* run whatever statements you want to trace */
SQL> ORADEBUG TRACEFILE_NAME
```

```
/opt/oracle/obase/admin/TEN/udump/ten1_ora_24953.trc
SQL> ORADEBUG EVENT 10046 TRACE NAME CONTEXT OFF
Statement processed.
```

前面的例子充分地展示了追踪你自己的会话的过程。ORADEBUG UNLIMIT 命令与 ALTER SESSION SET MAX\_DUMP\_FILE\_SIZE=UNLIMITED 有相同的效果，只不过前者影响的是附加进程而已。ORADEBUG EVENT 能够设置附加进程中的事件（另见本书第 3 部分）。ORADEBUG TRACEFILE\_NAME 能够检索附加进程的追踪文件的名称。注意：追踪文件名称是随会话级参数 TRACEFILE\_IDENTIFIER 的变动而变动的。使用 ORADEBUG EVENT 可以追踪进程级别的事件，但不能追踪数据库会话级别的事件。当然，如果是使用专用服务器，两者效果是一样的。尽管如此，当在一个共享的服务进程中启用追踪时，被这个共享服务进程所支持的数据库会话也是可以追踪的。在使用共享服务器时，可以使用 DBMS\_SYSTEM 包来将事件设定在会话级别（见第 20 章）。

2. SETOSPID

SETOSPID 的一个使用场景是：使用诸如 top（任何 UNIX 平台）、prstat（Solaris）、glance（HP-UX）或 nmon（AIX）之类的工具的 ORACLE 进程占用大量 CPU。这时，你应该将 SETOSPID 连接到一个进程，以 8 或 12 级的 10046 事件启用 SQL 追踪。下面是一个例子：

```
$ top
top - 22:04:20 up 54 min, 5 users, load average: 0.77, 0.76, 0.83
Tasks: 149 total, 2 running, 147 sleeping, 0 stopped, 0 zombie
Cpu(s): 10.8% us, 41.0% sy, 0.0% ni, 34.9% id, 13.0% wa, 0.3% hi, 0.0% si
Mem: 1034584k total, 796000k used, 238584k free, 39272k buffers
Swap: 524280k total, 0k used, 524280k free, 502048k cached

  PID USER      PR  NI  VIRT  RES  SHR  S %CPU  %MEM    TIME+  COMMAND
 13756 oracle    16   0  394m  68m  64m  R   50   6.8   0:05.95 oracle
 24887 oracle    15   0  391m  30m  27m  S   14   3.0   0:21.68 oracle
SQL> ORADEBUG SETOSPID 13756
Oracle pid: 19, Unix process pid: 13756, image: oracle@dbserver1.oradbpro.com (TNS
V1- V3)
```

3. SETORAPID

SETORAPID 在你知道展示性能问题和挂起的进程的会话标识（V\$SESSION.SID）的情况下能派上用场。通过连接 V\$SESSION 和 V\$PROCESS 这两个视图，你可以得到一个进程标识符（V\$PROCESS.PID），从而能够连接到那个进程。SETOSPID 中需要用到的操作系统进程标识符 SPID 也可以从 V\$PROCESS 中检索得到。下面的例子表明 V\$SESSION.SID=151 的那个进程十分重要：

```
SQL> SELECT pid, spid
FROM v$process p, v$session s
WHERE s.sid=151 and s.paddr=p.addr;
  PID SPID
-----
    19 15365
SQL> ORADEBUG SETORAPID 19
Unix process pid: 15365, image: oracle@dbserver1.oradbpro.com (TNS V1-V3)
```

Oracle 企业管理器 (Oracle Enterprise Manager) 中的 Top Sessions 可以用来确定哪些数据库会话消耗了较多的资源。一旦你知道了 V\$SESSION.SID, 你既可以通过 SETORAPID 来使用 V\$PROCESS.PID, 也可以通过 SETOSPID 来使用 V\$PROCESS.SPID。下面一个是在 Windows 平台上运行的例子:

```
SQL> ORADEBUG SETOSPID 4172
Oracle pid: 16, Windows thread id: 4172, image: ORACLE.EXE (SHAD)
SQL> ORADEBUG SETORAPID 16
Windows thread id: 4172, image: ORACLE.EXE (SHAD)
```

当跟踪数据收集完成时, 可以用 ORADEBUG EVENT 10046 TRACE NAME CONTEXT OFF 关闭追踪。

37.3.2 ORADEBUG IPC

ORADEBUG IPC 能够用于找出一个 ORACLE 实例可能使用了哪些共享的 UNIX 内存块。如果平台使用信号量来进行同步<sup>①</sup>, 那么关于信号量的信息也会被跟踪到。RAC 实例还将关于互联 IP 地址与协议等的信息写入跟踪文件。这对于在 Oracle9i 中核实互联地址十分重要, 因为 Oracle10g 以前的版本不将这些信息写入警告日志中。

在 UNIX 操作系统中, 每个共享内存块和信号量集都有一个唯一的标识。这些标识可以通过命令 ipcs 列出。使用 ipcs-mb 可以列出共享内存块及其拥有者和大小。

```
$ ipcs -mb
IPC status from /dev/mem as of Mon Nov 19 15:30:49 MEZ 2007
T      ID      KEY      MODE      OWNER      GROUP      SEGSZ
Shared Memory:
m      6 0x00006000 --rw-rw-rw-   root      system    1928456
m      7 0xc3f37a04 --rw-r----- oracle     dba      5050753024
m      8 0x0d000b59 --rw-rw-rw-   root      system      1440
```

接下来的部分将说明如何转储 IPC 信息。跟踪文件来自 RAC 实例。因此, 它包含的信息不止是共享内存块标识, 还有集群互联的相关信息。这些互联信息在 Oracle9i 和 Oracle10g 中都以字符串 “SSKGXPT” 标记。

```
$ sqlplus "/ as sysdba"
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.8.0 - 64bit Production
With the Real Application Clusters option
SQL> ORADEBUG SETMYPID
Statement processed.
SQL> ORADEBUG IPC
Information written to trace file.
SQL> ORADEBUG TRACEFILE_NAME
/var/opt/oracle/udump/fhs91_oracle_1065152.trc
```

下面的跟踪文件摘要包含了一些相关部分。值得注意的是, 这里的标识为 7 的内存块与前面用 ipcs 命令输出的内存块是一致的。

① 在 AIX 平台上, postwait 内核扩展替代了信号量。

```
Shared Memory:
ID          KEY
7           0xc3f37a04
```

下面的跟踪文件片段指出 IP 地址 172.16.0.1 被用作 RAC 互联地址。通信协议是 UDP (User Datagram Protocol, 用户数据报协议)。

```
SSKGXPT 0x1028d484 flags SSKGXPT_READPENDING active network 0
info for network 0
        socket no 8      IP 172.16.0.1      UDP 52608
```

ORADEBUG 也可能被 ASM 实例调用。下面这个例子描述了 ASM 实例对信号量的使用：

```
$ env ORACLE_SID=+ASM1 sqlplus / as sysdba
SQL> ORADEBUG SETMYPID
Statement processed.
SQL> ORADEBUG IPC
Information written to trace file.
SQL> ORADEBUG TRACEFILE_NAME
/opt/oracle/obase/admin/+ASM/udump/+asm1_ora_19685.trc
SQL> !grep -A 1 "Semaphore List" /opt/oracle/obase/admin/+ASM/udump/+asm1_ora_19685.
trc
Semaphore List=
884736
$ ipcs -s
----- Semaphore Arrays -----
key      semid      owner      perms      nsems
0x2c13c9dc 884736      oracle     640        44
0xcbcd5e70 1146881      oracle     640        154
```

ASM 实例使用了标识为 884736 的较小信号量集，而 RDBMS 实例则使用了一个较大的标识为 154 的信号量集<sup>①</sup>。

当系统中多个实例中的一个实例因没有清理共享内存块和（或）信号量而失败时，ORADEBUG IPC 是个理想的工具，可以收集足够的信息，用命令 `ipcrm` 来进行清理。上面例子中以粗体重复书写过两次的 UNIX IPC 标识（884736）可以被用来通过 `ipcrm` 命令清理共享内存块和信号量。IPC 标识在一个 Oracle 实例关闭和重新启动后不会被再次利用。当一台电脑的资源有限时，失败的实例占用共享内存块和信号量会导致资源紧张，从而使得新的 Oracle 实例无法启动。在这种情况下，用 `ipcrm` 命令清除两种 IPC 结构正是解决问题的办法。

37.3.3 ORADEBUG SHORT\_STACK

一个程序的调用栈显示了程序之间调用其他函数的路径。如果一个程序挂起，那么程序调用栈就可以显示它在调用路径中的哪一步骤挂起。可以使用两种方式通过 ORADEBUG 获得程序的调用栈：

- ERRORSTACK 诊断转储
- The SHORT\_STACK 命令

<sup>①</sup> 信号量个数的分配取决于初始化参数 PROCESSES，其值在 ASM 实例和 RDBMS 实例中分别是 40 和 150。

ERRORSTACK 转储将结果存储在一个很大的跟踪文件里面。SHORT\_STACK 更适合于迅速找出一个服务进程位于调用路径中的哪一个步骤。该命令的输出被发送到终端窗口而不是跟踪文件。该命令在 Oracle10g 及其以后的版本中都存在。下面是一个例子：

```
SQL> ORADEBUG SETOSPID 14807
Oracle pid: 34, Unix process pid: 14807, image: oracleDZAV024@l012r065
SQL> ORADEBUG SHORT_STACK
ksdxfstk()+32<-ksdxcxb()+1547<-sspuser()+111<-__funlockfile()+64<-kdsgrp()+283<- kdsf
br()+228<-qertbFetchByRowID()+889<-qergiFetch()+315<-qergiFetch()+315<- qernsFetch()
+603<-qerseFetch()+158<-subex1()+176<-subsr3()+195<-evaopn2()+4188<- expepr..7()+470
<-expeal()+82<-qerflFetchOutside()+158<-rwsfcd()+88<-insfch()+496<- insdrv()+592<-in
scovexe()+404<-insExecStmtExecIniEngine()+85<-insexe()+343<- opiexe()+9038<-opipls()
+2107<-opiodr()+984<-rpidrus()+198<-skgmstack()+158<- rpidru()+116<-rpiswu2()+420<-r
pidrv()+1519<-psddr0()+438<-psdnal()+339<- pev EXIM()+216<-pfrinstr_EXIM()+53<-pfr
un_no_tool()+65<-pfrun()+906<- pls ql_run()+841<-peicnt()+298<-kkxexe()+504<-opiexe()
+9038<-opipls()+2107<- opiodr()+984<-rpidrus()+198<-skgmstack()+158<-rpidru()+116<-
rpiswu2()+420<- rpidrv()+1519<-psddr0()+438<-psdnal()+339<-pevm_EXIM()+216<-pfrinstr
_EXIM()+53<- pfrun_no_tool()+65<-pfrun()+906<-pls ql_run()+841<-peicnt()+298<-kkxex
e()+504<- opiexe()+9038<-kpoal8()+2280<-opiodr()+984<-ttcpip()+1235<-opitsk()+1298<-
opiino()+1028<-opiodr()+984<-opidrv()+547<-sou2o()+114<-opimai_real()+163<- main()+
116<-__libc_start_main()+234<-_start()+42
```

当前的调用步骤在输出的顶部。当然，转储调用栈本身会影响调用栈。一般认为 ksdxcxb 是操作栈转储的调试回调函数。假定当命令 SHORT\_STACK 被它连接到的进程接收时，调用步骤 kdsgrp 会被执行。调用路径中的名字将在 Metalink 支持平台上被当做搜索关键词。

37.3.4 诊断转储

Oracle9i 第 2 版支持 85 种不同的诊断转储，Oracle10g 支持 146 种，而 Oracle11g 支持 165 种。因为转储种类众多，因此，很难全部讲述。要列出各种转储的名字，可以运行 ORADEBUG DUMPLIST。

1. CONTROLF

这个选项用于以不同的细节级别转储控制文件。这里有一个以级别 1 进行转储的例子：

```
Received ORADEBUG command 'DUMP CONTROLF 1' from process Windows thread id: 3580,
image:
DUMP OF CONTROL FILES, Seq # 545 = 0x221
V10 STYLE FILE HEADER:
  Compatibility Vsn = 169869568=0xa200100
  Db ID=1156831202=0x44f3d7e2, Db Name='ORCL'
  Activation ID=0=0x0
  Control Seq=545=0x221, File size=430=0x1ae
  File Number=0, Blksiz=16384, File Type=1 CONTROL
```

2. EVENTS

EVENTS 转储不是真正的诊断转储。它只是将激活的事件写入跟踪文件。如果你不能确定一个会话、进程或实例中哪个事件是活动的，下面有一种很好的方式去查询。示例如下：

```
SQL> ALTER SESSION SET EVENTS '10046 trace name context forever, level 8';
Session altered.
SQL> ALTER SESSION SET EVENTS '4031 trace name heapdump level 3';
Session altered.
SQL> ORADEBUG SETMYPID
Statement processed.
SQL> ORADEBUG DUMP EVENTS 1
Statement processed.
SQL> ORADEBUG TRACEFILE_NAME
/opt/oracle/obase/admin/TEN/udump/ten1_ora_20206.trc
```

由 ORADEBUG DUMP EVENTS 生成的采样跟踪文件的内容如下所示：

```
Dump event group for level SESSION
TC Addr Evt#(b10) Action TR Addr Arm Life
B72DB4F4 4031 1 b72db560 0 0
    TR Name          TR level TR address    TR arm    TR life    TR type
    HEAPDUMP                3 0          1          2 -1221853072
B72DB3F8 10046 1 b72db464 0 0
    TR Name          TR level TR address    TR arm    TR life    TR type
    CONTEXT                8 0          -1          2 -1221853072
```

表 37-2 中列出了能被 EVENTS 转储支持的级别以及设置相应级别的方法。注意, ORADEBUG EVENT 中事件集的范围是那些之前已被 ORADEBUG SETORAPID 或类似命令连接到的进程。

表37-2 ORADEBUG DUMP EVENTS的级别和范围

| 级 别 | 事件范围 | 使用的命令 |
|-----|------|----------------------------------|
| 1 | 会话 | ALTER SESSION、DBMS_SYSTEM.SET_EV |
| 2 | 进程 | ORADEBUG EVENT |
| 4 | 实例 | ALTER SYSTEM |

3. ERRORSTACK

如前所述, 一个程序的调用栈可以通过 ERRORSTACK 的诊断转储获得。在转储级别较高时, ERRORSTACK 转储可以获得比调用栈更多的信息。表 37-3 给出了级别 1、2、3 的转储所得的跟踪文件的组成结构的一个概览。该表适用于 Oracle10g。跟踪文件的结构在不同 Oracle 版本之间会有区别。

表37-3 ERRORSTACK跟踪文件的内容

| 跟踪文件内容 | 级别1 | 级别2 | 级别3 |
|-----------------|-----|-----|-----|
| 当前SQL语句 | 是 | 是 | 是 |
| 调用栈轨迹 | 是 | 是 | 是 |
| 进程状态 (包含会话等待历史) | 否 | 是 | 是 |
| 会话或实例级别的活动事件 | 否 | 否 | 是 |
| 游标转储 | 否 | 否 | 是 |

下面是一个 ERRORSTACK 转储的例子：

```

SQL> ORADEBUG SETOSPID 6524
Oracle pid: 16, Unix process pid: 6524, image: oracleTEN1@dbserver1.oradbpro.com
SQL> ORADEBUG DUMP ERRORSTACK 1
Statement processed.
SQL> ORADEBUG TRACEFILE_NAME
/opt/oracle/obase/admin/TEN/udump/ten1_ora_6524.trc
SQL> !less /opt/oracle/obase/admin/TEN/udump/ten1_ora_6524.trc
Received ORADEBUG command 'DUMP ERRORSTACK 1' from process Unix process pid: 6518,
image:
*** 2007-09-09 12:36:02.525
ksedmp: internal or fatal error
Current SQL statement for this session:
SELECT sys_context('userenv', 'sessionid'),sys_context('userenv', 'client_identifier
'),sys_context('userenv', 'client_info'),sys_context('userenv', 'host'), /* correspo
nds to v$session.machine */sys_context('userenv', 'os_user'), /* corresponds to v$se
ssion.osuser */ sys_context('userenv', 'terminal')FROM dual
----- Call Stack Trace -----
calling      call      entry      argument values in hex
location     type      point      (? means dubious value)
-----
ksedst()+27   call      ksedst1()   1 ? 1 ?
ksedmp()+557  call      ksedst()    1 ? 2A120F04 ? 76010B ?
              2A11FC8E ? 0 ? BFFF8638 ?
ksdxfdmp()+1382  call      0C94496E    1 ? 83CD95B ? C816D60 ?
              BFFFB7DC ? 83E55FD ?
              2EAF9940 ?
ksdxcb()+1321  call      00000000    BFFFBAB8 ? 11 ? 3 ?
              BFFFBAB8 ? BFFFBAB8 ?
sspuser()+102  call      00000000    1 ? 2000 ? 0 ? 0 ? 0 ? 0 ?
0071A7A0      signal    00000000    C ? BFFFBF70 ? BFFFBFF0 ?
ntttd()+155   call      snttread()  D ? C86BEC6 ? 810 ? 0 ?
...

```

当栈被转储后，服务进程将会等待从客户端的调用点 ntttd 发来的网络包。V\$SESSION 的等待事件 SQL\*Net message from client 证实了这一点。

```

SQL> SELECT p.spid, s.event, s.state
FROM v$session s, v$process p
WHERE s.username='NDEBES'
AND s.paddr=p.addr;
SPID      EVENT                                STATE
-----
6524      SQL*Net message from client WAITING

```

调用点 ntttd 等待 UNIX 系统的 read 调用来返回。使用系统调用跟踪工具 strace 可以发现，系统的 read 调用是通过文件描述符 13 来进行的。该文件描述符表示了被跟踪的专用服务进程与客户端之间的一个套接字连接。

```

$ strace -p 6524
Process 6524 attached - interrupt to quit
read(13, <unfinished ...>

```



```
Process 6524 detached
$ ls -l /proc/6524/fd/13
lrwx----- 1 oracle oinstall 64 Sep  9 13:08 /proc/6524/fd/13 -> socket:[4514905]
```

4. HANGANALYZE

HANGANALYZE 命令执行悬挂分析转储 (hang analysis dump)。这类转储用来分析数据库的悬挂事件。请参考十分详细的 Metalink note 61552.1 以获取关于这个话题的更多信息。接下来是一个级别为 1 的悬挂分析转储。其例子的操作背景如下。

(1) 会话 141 (V\$SESSION.SID=141) 在表上执行 LOCK TABLE IN EXCLUSIVE MODE。

(2) 会话 147 试图插入同一个表格。

(3) 会话 147 执行的 INSERT 语句必须等待会话 141 释放表锁后方可执行。

相应的悬挂分析转储在后面将列出。链表列出了等待的会话，其中包含了多个会话，这表明位于链表头部（左侧）的会话阻塞了在同一链表中的其他会话。

```
=====
HANG ANALYSIS:
=====
Open chains found:
Chain 1 : <cnode/sid/sess_srno/ospid/wait_event> :
          <0/141/47/3580/No Wait> -- <0/147/518/6064/enq: TM - contention>
Other chains found:
Chain 2 : <cnode/sid/sess_srno/ospid/wait_event> :
          <0/145/117/5244/jobq slave wait>
Chain 3 : <cnode/sid/sess_srno/ospid/wait_event> :
          <0/150/38/440/Streams AQ: qmn slave idle wait>
Chain 4 : <cnode/sid/sess_srno/ospid/wait_event> :
          <0/152/1/5440/Streams AQ: waiting for time man>
Chain 5 : <cnode/sid/sess_srno/ospid/wait_event> :
          <0/154/1/4584/Streams AQ: qmn coordinator idle>
Extra information that will be dumped at higher levels:
[level 4] : 1 node dumps -- [REMOTE_WT] [LEAF] [LEAF_NW]
[level 5] : 4 node dumps -- [SINGLE_NODE] [SINGLE_NODE_NW] [IGN_DMP]
[level 6] : 1 node dumps -- [NLEAF]
[level 10] : 11 node dumps -- [IGN]
```

5. MODIFIED\_PARAMETERS

通过 MODIFIED\_PARAMETERS 选项进行的转储可以将通过 ALTER SESSION 或 ALTER SYSTEM 命令修改过的初始化参数保存到跟踪文件中。只需要级别 1 的转储就可以做到这一点。更高的级别并不能提供更详细的信息。在下面的例子中，初始化参数 PGA\_AGGREGATE\_TARGET 和 SKIP\_UNUSABLE\_INDEXES 被修改了。不过，这些参数以及其新值将会出现在通过 MODIFIED\_PARAMETERS 选项进行转储后所得的文件中。

```
SQL> ALTER SYSTEM SET pga_aggregate_target=512m;
System altered.
SQL> ALTER SESSION SET skip_unusable_indexes=true;
System altered.
SQL> ORADEBUG DUMP MODIFIED_PARAMETERS 1
```

Statement processed.

跟踪文件中包含了每个修改过的参数的修改情况。

```
Received ORADEBUG command 'DUMP MODIFIED_PARAMETERS 1' from process Windows thread
id: 3580, image:
DYNAMICALLY MODIFIED PARAMETERS:
  pga_aggregate_target      = 536870912
  skip_unusable_indexes     = TRUE
```

6. PROCSTAT 以及进程级别的操作系统统计

通过 ORADEBUG PROCSTAT 命令可以将进程级别的操作系统的统计结果转储到跟踪文件中。视图 V\$SESSTAT 中统计 CPU used by this session (本会话的 CPU 使用情况)，这里对 CPU 使用情况的统计粒度要粗一些。

```
SQL> ORADEBUG PROCSTAT
SQL> ORADEBUG TRACEFILE_NAME
/opt/oracle/obase/admin/TEN/udump/ten1_ora_8739.trc
SQL> !cat /opt/oracle/obase/admin/TEN/udump/ten1_ora_8739.trc
----- Dump of Process Statistics -----
User time used = 500
System time used = 1750
...
Page reclaims = 66357
Page faults = 1
...
Voluntary context switches = 2775
Involuntary context switches = 65892
```

37.4 小结

SQL\*Plus 中的 ORADEBUG 对于解决问题和诊断性能是必不可少的工具。使用 ORADEBUG 的基本步骤包括：连接到一个进程，设置一个事件或者进行诊断转储，检索跟踪文件名称，与连接到的进程断开。ORADEBUG 可以提供很多通过其他手段无法获得的信息，比如一个进程或会话中的事件集。本章讲述了一些最常见和最有用的 ORADEBUG 命令。实际上，ORADEBUG 中还存在更多有用的功能，比如在 RAC 环境下用来调用全局队列服务调试器 (global enqueue service debugger) 的 ORADEBUG LKDEBUG 命令。

Part 12

第十二部分

附 录

本 部 分 内 容

- 附录 A 启用和禁用 DBMS 可选项
- 附录 B 参考书目
- 附录 C 术语表

启用和禁用 DBMS 可选项

ORACLE DBMS 可选项，在使用 OUI（通用安装器）进行安装时会被安装，但可以通过运行某些特定 make 命令取消。当然，这只适用于 UNIX 平台。这样做的好处是在取消某些可选项时不必重启 OUI，这样可以避免执行那些在初始化安装时所需的分区表或索引等耗时较多的工作。因此，一个好的做法就是使用 OUI 安装所有的可选项，而在运行某些特定命令时取消那些不需要的选项。make 命令也可以用于取消那些在安装过程中没有选择却被 OUI 错误地加入到系统中的可选项。make 命令运行时必须以 \$ORACLE\_HOME/rdbms/lib 作为当前目录，且总是以如下形式出现：

```
make -f ins_rdbms.mk target1 ... [ targetN ] ioracle
```

这里的占位符 target1 以及 targetN 是 make 命令的目标参数（见表 A-1）。

表A-1 用于开启和取消DBMS可选项的make命令的目标参数

| DBMS选项 | 开启命令参数 | 取消命令参数 |
|---------------------------------------|---------|----------|
| 数据挖掘 (Data Mining) | dm_on | dm_off |
| 数据挖掘引擎评价 (Data Mining Scoring Engine) | dmse_on | dmse_off |
| 数据库审计 (Database Vault) | dv_on | dv_off |
| 标签安全 (Label Security) | lbac_on | lbac_off |
| OLAP (联机分析处理) | olap_on | olap_off |
| 分区 (Partitioning) | part_on | part_off |
| RAC (实时应用集群) | rac_on | rac_off |
| 空间配置 (Spatial) | sdo_on | sdo_off |

下面是一个移除分区选项和 RAC 选项的例子：

```
$ make -f ins_rdbms.mk part_off rac_off ioracle
```

在执行 make 命令前，必须关闭任何可能修改 ORACLE\_HOME 的 DBMS 实例。

- [Bunc 2000] Bunce, Tim; Carty Alistair, *Programming the Perl DBI*, O'Reilly, 2000, <http://www.oreilly.com>
- [CPAN] Comprehensive Perl Archive Network, <http://www.cpan.org>
- [EnGo 2004] Ensor, Dave; Gorman, Tim; Hailey, Kyle; Kolk Anjo; Lewis, Jonathan; McDonald, Connor; Millsap, Cary; Morle, James; Mogens, Nørgaard; Ruthven, David, *Oracle Insights: Tales of the Oak Table*, Apress, 2004, <http://www.apress.com>
- [Engs 2005] B. Engsig, *Designing applications for performance and scalability*, 2005, [http://www.oracle.com/technology/deploy/performance/pdf/designing\\_applications\\_for\\_performance\\_and\\_scalability.pdf](http://www.oracle.com/technology/deploy/performance/pdf/designing_applications_for_performance_and_scalability.pdf)
- [Kyte 2005] Kyte, Thomas, *Expert Oracle Database Architecture: 9i and 10g Programming Techniques and Solutions*, Apress, 2005, <http://www.apress.com>
- [Lewi 2005] Lewis, Jonathan, *Cost-Based Oracle Fundamentals*, Apress, 2005, <http://www.apress.com>
- [MiHo 2003] Millsap, Cary; Holt, Jeff, *Optimizing Oracle Performance*, O'Reilly, 2003, <http://www.oreilly.com>
- [OJ10 2005] *Oracle Database 10g Release 2 (10.2.0.3) JDBC Drivers JavaDoc*, Oracle Corp., [http://download.oracle.com/otn/utilities\\_drivers/jdbc/10201/javadoc.zip](http://download.oracle.com/otn/utilities_drivers/jdbc/10201/javadoc.zip)
- [OJ92 2002] *Oracle9i 9.2.0.8 JDBC Drivers JavaDoc*, Oracle Corp., [http://download.oracle.com/otn/utilities\\_drivers/jdbc/9205/javadoc.tar](http://download.oracle.com/otn/utilities_drivers/jdbc/9205/javadoc.tar)
- [OL92 2002] *Oracle9i Release 2 Documentation Library*, Oracle Corp., 2002, <http://www.oracle.com/pls/db92/db92.homepage>
- [OL10 2005] *Oracle10g Release 2 Documentation Library*, Oracle Corp., 2005, <http://otn.oracle.com>
- [OL11 2007] *Oracle11g Release 1 Documentation Library*, Oracle Corp., 2007, <http://otn.oracle.com>
- [ShDe 2004] Shee, Richmond; Deshpande, Kirtikumar; Gopalakrishnan, K, *Oracle Wait Interface: A Practical Guide to Performance Diagnostics and Tuning*, McGraw-Hill/Osborne, 2004, <http://www.>

oraclepress.com

[Sol8 2000] *Solaris 8 Reference Manual Collection*, Sun Microsystems, Inc., 2000, <http://docs.sun.com>

[SoTu 2007] *Solaris Tunable Parameters Reference Manual*, Sun Microsystems, Inc., 2007, <http://docs.sun.com>

[VaDe 2001] Vaidyanatha, Gaja Krishna; Deshpande, Kirtikumar; Kostelac, John A., *Oracle Performance Tuning 101*, McGraw-Hill Osborne Media, 2001, <http://www.oraclepress.com>

[WaCh 2000] Wall, Larry; Christiansen Tom; Orwant, John, *Programming Perl*, O'Reilly, 2000, <http://www.oreilly.com>

应用程序插桩 (application instrumentation)

插桩是一个软件评估自身性能的能力。一个插桩了的 ORACLE 应用程序将模块、动作和客户端标识符传递给 DBMS, 允许数据库管理员收集相关的度量信息。

并发 (concurrency)

并发是指计算机系统在同一时间内执行多个进程的能力。并发进程可以通过进程间通信进行交互。

连接字符串 (connect string)

一个连接字符串包含用户名、密码以及网络服务名 (可选) 或其他用于连接到一个 Oracle 网络服务 TNS 监听器的配置 (可选), 如 ndebes/secret@ten.oradbpro.com。

连接池 (connection pool)

连接池是一系列数据库连接的集合, 通常由应用服务器来维护。当连接池被初始化并打开后, 与 ORACLE 实例的连接也就打开了; 在连接池的整个生命周期中, 连接一直保持打开状态, 直至因为空闲超时而取消。连接池的客户端, 比如 Java Bean, 请求一个连接来运行一些 SQL 语句, 就会将一个连接加入到连接池。连接池使得系统不必为每个请求都建立一个新的数据库会话, 从而可以降低开销。

一致读 (consistent read)

一致读是基于 SCN (System Change Number, 系统改变号) 的数据快照, 这个快照记录的是 SQL 语句初始化时数据的状况。如果在游标打开后数据发生了变化, 系统仍将通过撤销表空间中的撤销段来重新获得 SQL 语句初始化时数据的值。

基于成本的SQL优化器 (CBO)

基于成本的优化器是 ORACLE DBMS 的一个组成部分。它仔细分析 SQL 语句并寻求最优的执行方案。它会计算各种可能方案的成本, 并选择成本最低方案来执行。

当前读 (current read)

从当前数据块中读取数据。这与一致读正好相反, 因为一致读并不从最新的数据中读取数据。

游标 (cursor)

严格说, 游标是用于迭代遍历查询结果集中各行的一个控制结构。从 ORACLE 服务器的角

度来说，游标可以用来执行任何 SQL 或 PL/SQL 语句。

数据定义语言 (Data Definition Language, DDL)

用于创建诸如表和索引等数据库对象的 SQL 语句。

数据库 (database)

数据库是存储在计算机系统中结构化的数据或记录的集合。数据库管理系统可以操纵它来对查询作出反应。一个 ORACLE 数据库包括控制文件、重做日志文件和数据文件。

数据库管理系统 (Database Management System, DBMS)

数据库管理系统是用于控制数据库中数据的存储、管理和检索的程序集合。

数据操纵语言 (Data Manipulation Language, DML)

数据操纵语言是检索和修改数据库中数据的 SQL 语句。数据操纵语言包括 SELECT、INSERT、MERGE、UPDATE 和 DELETE 等 SQL 语句。

数据库调用 (database call)

数据库调用是 ORACLE 服务器进程对服务一个客户端请求所执行的解析、执行或提取操作。

动态性能视图 (V\$固定视图) (dynamic performance view, V\$ fixed view)

该视图尽可能地展示 ORACLE 数据库管理系统的内部过程，并为度量数据库性能提供了途径。这些视图的名字都以 V\$ 开头。V\$ 固定视图并不与数据库中的片段 (segment) 关联。它们的多数都基于 SGA (系统全局区) 中的内存结构，因而具有动态性。还有一些是基于控制文件记录分区 (control file record section)。

端到端的度量 (end-to-end metrics)

端到端的度量是指 ORACLE 数据库管理系统所提供的基于应用程序插桩的度量。

队列 (enqueue)

队列是一种锁机制，可以按先进先出 (FIFO) 的方式对请求和服务进行排队。队列允许多种访问模式，如共享或互斥方式。

执行 (execute)

执行一个 SQL 语句或 PL/SQL 程序块。比如执行 DDL 或 DML 语句所表示的操作。执行 SELECT 语句时，游标指向结果集的第一行。

提取 (fetch)

指通过 SELECT 语句进行的数据检索。

空闲等待 (idle wait)

指 ORACLE 服务器进程响应请求之间的间隔时间。当用户在思考而没有提交请求时，服务器的空闲等待时间就会不断累加。

实例 (instance, 即 DBMS 实例)

通过运行 \$ORACLE\_HOME/bin/oracle 程序 (UNIX 平台) 或 %ORACLE\_HOME%\bin\oracle.exe (Windows 平台) 来实现 ORACLE 数据库服务器的一系列进程和存储结构。

实例服务名 (instance service name)

实例服务名是一个实例通过一个或多个 TNS 监听器注册的服务名。它们可以通过初始参数

SERVIC\_NAMES 来设定。在 Oracle10g 中, DBMS\_SERVICE 包可以用于添加实例服务。在网络服务名定义中, SERVICE\_NAME 表示实例服务名。

插桩 (instrumentation)

一些计算机语言中, 用于辅助程序度量自身, 并记录每个程序模块花费了多少时间的程序代码。

开锁 (latch)

这是 ORACLE 服务器的一种同步机制: 为保证数据结构一致性, ORACLE 服务器在某段程序代码的临界区上执行互斥操作, 保证一次只有一个进程可以进入该临界区。进程会重复地试图获取开锁, 这样会消耗 CPU 时间 (“自旋”; 未文档化的参数 SPIN\_COUNT 控制次数)。与队列不同, 开锁并不对请求进行排队, 且只允许互斥访问。开锁通过自动测试和设置机器指令来实现。

延迟 (latency)

延迟是指初始化某项操作与初始化操作的结果可见之间的一段时间。该词的意思与上下文高度相关。该词经常被用于有关性能调优的环境中。比如, 数据库服务器与客户端之间不必要的信息往返可能会因为网络延迟而造成性能损失。

锁 (lock)

见队列。

逻辑读 (logical read)

从缓存在系统全局区中的 ORACLE 数据块中读取数据称为逻辑读。

网络服务名 (net service name)

网络服务名是在配置文件 tnsnames.ora 或诸如 LDAP (Lightweight Directory Access Protocol, 轻量级目录访问协议) 这样的目录服务中定义的。网络服务名由 tnsping 命令在连接字符串中指定。

分页 (paging)

通过操作系统将虚拟内存的一部分数据写入硬盘的机制称为分页机制。当物理内存 (RAM) 不足或耗尽时就会发生分页。分页使得操作系统可以执行所需虚拟内存大于系统 RAM 容量的程序。

解析 (parse)

对 SQL 语句或 PL/SQL 程序块执行的句法和语义分析, 并包括对适用优先权的确认。每个语句或块都关联到一个游标。

性能 (performance)

性能是一个计算机或软件系统所执行的有效工作量相对于其消耗的时空资源的比值。好性能意味着更短的响应时间和适当的资源消耗; 差性能则意味着响应时间太长以及消耗过多的资源。

物理读 (physical read)

ORACLE 实例中的一些进程请求操作系统从硬盘上的一个或多个数据块中读取数据, 这样的过程就是物理读。

物理写 (physical write)

ORACLE 实例中的一些进程请求操作系统向硬盘上的一个或多个数据块中写入数据, 这样

的过程就是物理写。例如，当数据库写进程（DBWn）将缓冲区缓存中修改的数据块写入磁盘时就会发生物理写。其他例子，如专用或共享服务器进程向排序段或 LOB 段写入的过程。

轮询（polling）

不间断地主动请求某个资源。轮询经常会导致大量 CPU 时间的浪费。通常，只要可能的话，就应该让进程休眠来等待资源，而不是主动地轮询。

可移植性操作系统接口（POSIX）

POSIX（Portable Operating System Interface，可移植性操作系统接口）是一系列相关标准的总称，这些标准定义了应用编程接口（API），使得符合这些标准的程序可以在任何种类的 UNIX 操作系统上运行。

随机存取存储器（RAM）

随机存取存储器是一个计算机系统的易失性主存（volatile main memory）。

实时应用集群（RAC）

运行在集群硬件上的多实例版的 ORACLE 数据库管理系统，具有高可扩展性和高可靠性。RAC 要求集群软件的支持，这类软件有 Oracle Clusterware、HP-UX ServiceGuard、IBM HACMP、Veritas Cluster、Sun Cluster 及 HP TruCluster 等。

递归 SQL 语句（recursive SQL）

ORACLE 数据库管理系统由用户 SYS 在数据字典上执行的 SQL 语句，执行这些语句是为了服务于数据库客户端发来的请求。另见用户级 SQL。

资源配置文件（resource profile）

通常以表格形式展示的响应时间的分摊情况。资源配置文件的每行都包含一个响应时间的贡献者，如 CPU 时间消耗或等待事件。理想情况下，该表会包含以下列：贡献者的名字、占响应时间的百分比、所消耗的响应时间（单位为秒）、贡献者被调用的次数以及平均每次调用所消耗的时间。这个表格按照贡献的响应时间占总时间的百分比降序排列。

响应时间（response time）

一个电子系统为了完成一个用户请求所需要消耗的时间。响应时间包括 CPU 消耗时间和等待时间。

基于规则的 SQL 优化器（rule-based SQL optimizer, RBO）

相对于更高级的基于成本的 SQL 优化器而言，基于规则的 SQL 优化器是更为简单的先驱者。基于规则的 SQL 优化器根据一些固定的规则来产生执行计划。它不关心表格的大小和列的选择情况。自 Oracle10g 发布后，这种优化器已经被废弃了。

结构化查询语言（Structured Query Language, SQL）

SQL 是关系数据库管理系统中用于检索和管理数据的计算机语言。它可以创建、修改和移除数据库模式对象（DDL）、数据操作（DML），以及对数据库对象的存取控制进行管理。

扩展 SQL 跟踪文件（extended SQL trace file）

扩展 SQL 跟踪文件是记录一个或多个数据库客户端行为的日志。记录的行为包括：解析、执行和提取。跟踪级别 1、4、8 和 12 可以控制跟踪日志文件所记录的行为的详细程度。

可扩展性 (scalability)

可扩展性是指一个系统、网络或应用程序，在工作量的规模发生很大改变时，仍能很好地工作，或只需通过逐渐扩大自身的规模就可以正常工作。

休眠 (sleeping)

当所需的资源不具备时，进程就可能会进入休眠状态。休眠中的进程不会占用 CPU 时间。因此，休眠是一种比轮询更好的方式。

交换 (swapping)

将虚拟内存中的数据从 RAM 转移到磁盘。

系统全局区 (System Global Area, SGA)

内存结构中的数据库缓冲区缓存、共享池、java 池、大池、流池以及日志缓冲区。在 UNIX 系统中，SGA 常驻共享内存区。

系统改变号 (System Change Number, SCN)

对 ORACLE 数据库中数据的每一次修改都会来用一个单调递增的数字，即系统改变号。备份、恢复和一致读等操作都是基于系统改变号来实现的。

系统调用 (system call)

运行在操作系统和用户级程序之间的例程。系统调用在内核模式 (kernel mode) 下运行，而应用级代码在用户模式 (user mode) 下运行。操作设备，如硬盘或网络适配器，都是由系统调用实现的。UNIX 系统调用都遵循 POSIX 标准。

思考时间 (think time)

思考时间是来自负载测试及标杆管理 (load testing and benchmarking) 领域的术语。人们在操作电子信息服务时，理解从服务端返回的信息需要一定的时间。服务器在处理完上一条请求后，到从用户接收到下一条请求前的这一段间隔时间称为思考时间。在这段时间里，用户通常在思考下一步该做什么。

用户级 SQL (user level SQL)

数据库用户直接执行的 SQL 语句 (非递归 SQL 语句)。

等待事件 (wait event)

等待事件可以用来确定引起数据库会话响应时间增加的原因。通过等待接口或 SQL 追踪文件，我们可以确定一个数据库会话等待的事件以及累计等待时间。

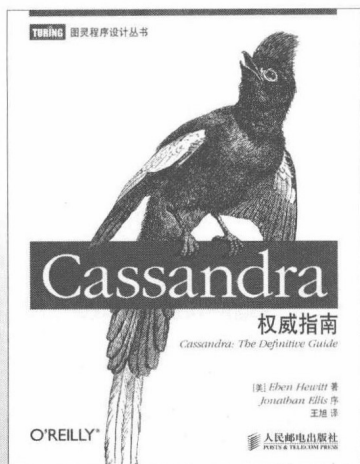
等待接口 (wait interface)

等待接口是指一系列 V\$ 固定视图，这些视图用来展示数据库客户端正在等待的事件。这些 V\$ 视图包括了现在处于等待中的数据库会话，也包括了过去等待过的会话的总体情况。等待接口常用于诊断性能问题。

X\$ 固定表 (X\$ fixed tables)

X\$ 固定表是一种未被文档化的 ORACLE 数据库管理系统的内部结构。V\$ 固定视图基于 X\$ 固定表。X\$ 固定表通常简称为 X\$ 表，不过它们不是通常意义上的表，因为它们没有映射到表空间中的片段。

图灵最新重点图书



Cassandra 权威指南
书号: 978-7-115-25854-0
译者: 王旭
作者: Eben Hewitt
定价: 59.00 元

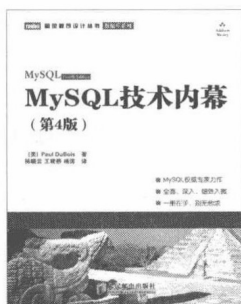
- ▶ 深入理解 Cassandra 面向列的数据结构的原理。
- ▶ 学习如何对 Cassandra 进行数据的写入、更新和读取。
- ▶ 研究如何在 Cassandra 集群中, 根据应用的需求来增加或删除节点。
- ▶ 通过示例演示了如何从关系型数据库向 Cassandra 迁移一个可以工作的应用。
- ▶ 介绍了如何使用 Java、Python 和 C# 来写客户端。
- ▶ 介绍了如何使用 JMX 接口来监控集群的工作状况、内存情况等。
- ▶ 为优化性能进行内存设置、数据存储以及缓存的调优。

本书是一本广受好评的Cassandra 图书。与传统的关系型数据库不同, Cassandra 是一种开源的分布式存储系统。书中介绍了它无中心架构、高可用、无缝扩展等引人注目的特点, 讲述了如何安装、配置Cassandra 及如何在其上运行实例, 还介绍了对它的监控、维护和性能调优手段, 同时还涉及了Cassandra 相关的集成工具Hadoop 及其类似的其他NoSQL 数据库。

本书适合数据库开发人员与网站开发者阅读。



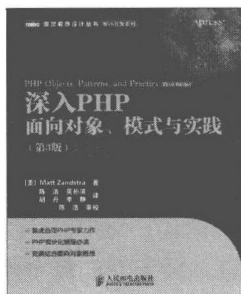
项目经理应该知道的 97 件事
书号: 978-7-115-25100-8
定价: 39.00 元



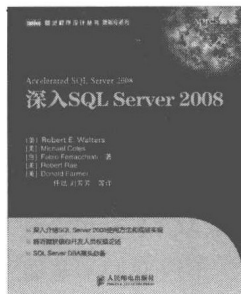
MySQL 技术内幕 (第 4 版)
书号: 978-7-115-25595-2
定价: 139.00 元



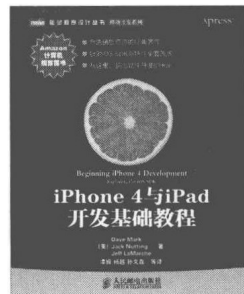
Flash 游戏编程基础教程
书号: 978-7-115-25434-4
定价: 75.00 元



深入 PHP: 面向对象、模式与实践 (第 3 版)
书号: 978-7-115-25624-9
定价: 69.00 元



深入 SQL Server 2008
书号: 978-7-115-25619-5
定价: 99.00 元



iPhone 4 与 iPad 开发基础教程
书号: 978-7-115-25552-5
定价: 79.00 元